N° d'ordre: 3624

THÈSE

Présentée devant

devant l'Université de Rennes 1

pour obtenir

le grade de : DOCTEUR DE L'UNIVERSITÉ DE RENNES 1
Mention INFORMATIQUE

par

Alexandra DESMOULIN

Équipe d'accueil : DIONYSOS - IRISA
École Doctorale : Matisse
Composante universitaire : IFSIC

Titre de la thèse :

Test d'interopérabilité de protocoles : de la formalisation des critères d'interopérabilité à la génération des tests

soutenue le 6 décembre 2007 devant la commission d'examen

M. :	Raymond	MARIE	Président
MM. :	Richard	CASTANET	Rapporteurs
	Alexandre	PETRENKO	
MM. :	Thierry	JÉRON	Examinateurs
	César	VIHO	

Remerciements

Je remercie Monsieur Raymond MARIE, Professeur à l'Université de Rennes I, qui m'a fait l'honneur de présider ce jury.

Je remercie Monsieur Richard CASTANET, Professeur à l'ENSEIRB (Bordeaux) et Alexandre PETRENKO, Chercheur Principal au CRIM de Montréal (Canada) d'avoir bien voulu accepter la charge de rapporteur. Merci pour toutes ces remarques qui m'ont permis d'améliorer ce travail.

Je tiens également à remercier Monsieur Thierry JÉRON (IRISA/INRIA, équipe VERTECS) pour l'évaluation minitieuse de ce travail. Merci pour les questions et remarques qui m'ont permis d'améliorer ce travail. Je remercie également Monsieur Raymond MARIE (IRISA/Université de Rennes 1, équipe DIONYSOS) qui a accepté d'examiner ce travail.

Je remercie César VIHO qui a encadré ce travail et qui m'a fait confiance dès mon premier stage à l'IRISA puis pendant mon DEA et cette thèse.

Je remercie tous les membres des équipes dans lesquelles j'ai réalisée cette thèse (ARMOR puis DIONYSOS) qui m'ont accueillie et ont mis à ma disposition toutes les ressources nécessaires pour effectuer mes travaux dans de bonnes conditions. Merci à toutes les personnes de l'équipe, les permanents, les doctorants (Alexandre, Ali, Ana Paula, Ariel, Fatma, Francine, Gilles, Joanna, Kamal, Kandaraj, Majd, Martin, Pablo, Thierry, Yézékael, etc) et ingénieurs (Annie, Anthony, Antoine, Bruno, Frédéric, Gildas) avec qui j'ai partagé de bons moments.

Un merci particulier aux personnes avec qui j'ai partagé mon bureau, Kamal Singh puis Fatma Bouabdallah-Othman, et aux membres passés et présents du groupe Test de l'équipe ARMOR puis DIONYSOS : Anthony Baire, Francine NGani, Annie Floch, et Ariel Sabiguero.

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Dans le domaine des réseaux, des normes sont définies, décrivant les services qui doivent être rendus par les protocoles effectivement implémentés. Les implémentations sont ensuite réalisées par différents constructeurs. Elles doivent donc être testées de façon à vérifier leur capacité à fonctionner "correctement" dans un contexte opérationnel. De plus, ces protocoles sont utilisés pour mettre en relation des systèmes développés par différents constructeurs pouvant opter pour des choix d'implémentation, de méthode et/ou d'outils différents lors du processus de développement. C'est pourquoi le test des implémentations de protocoles est une étape importante de leur développement.

L'objectif de l'activité de test est donc de vérifier qu'un produit réalisé répond bien aux objectifs définis avant cette réalisation. Dans le domaine des réseaux, les produits testés sont des implémentations de protocoles développées par différents fournisseurs. Le cahier des charges du produit est alors sa spécification, généralement une norme. Cette norme ne décrit généralement pas comment implémenter ce cahier des charges. Ce que décrit cette norme, c'est le ou les services que doivent rendre le protocole et les règles sur lesquelles les systèmes doivent s'appuyer pour communiquer correctement avec d'autres systèmes.

Il existe plusieurs méthodes de test pour vérifier qu'une implémentation sera capable de fonctionner "correctement" dans un contexte opérationnel. Parmi ces tests, les tests dits fonctionnels s'intéressent aux fonctionnalités d'un ou plusieurs systèmes sans prendre en compte leur structure interne. Les systèmes testés sont alors des "boîtes-noires" donc le comportement n'est connu que par les interactions à travers leurs interfaces avec l'environnement ou d'autres systèmes. Parmi ces différents types de tests fonctionnels, on peut distinguer :

le test de conformité qui vérifie qu'une implémentation d'un système est conforme à sa spécification

le test d'interopérabilité qui vérifie que plusieurs systèmes sont capables de communiquer tout en fournissant les services décrits dans leur spécification

le test de robustesse qui vérifie le comportement de l'implémentation dans des conditions anormales de fonctionnement (pannes, erreurs de manipulation, etc)

le test de performances qui vérifie les paramètres de performance d'une implémentation (délais, débit, etc)

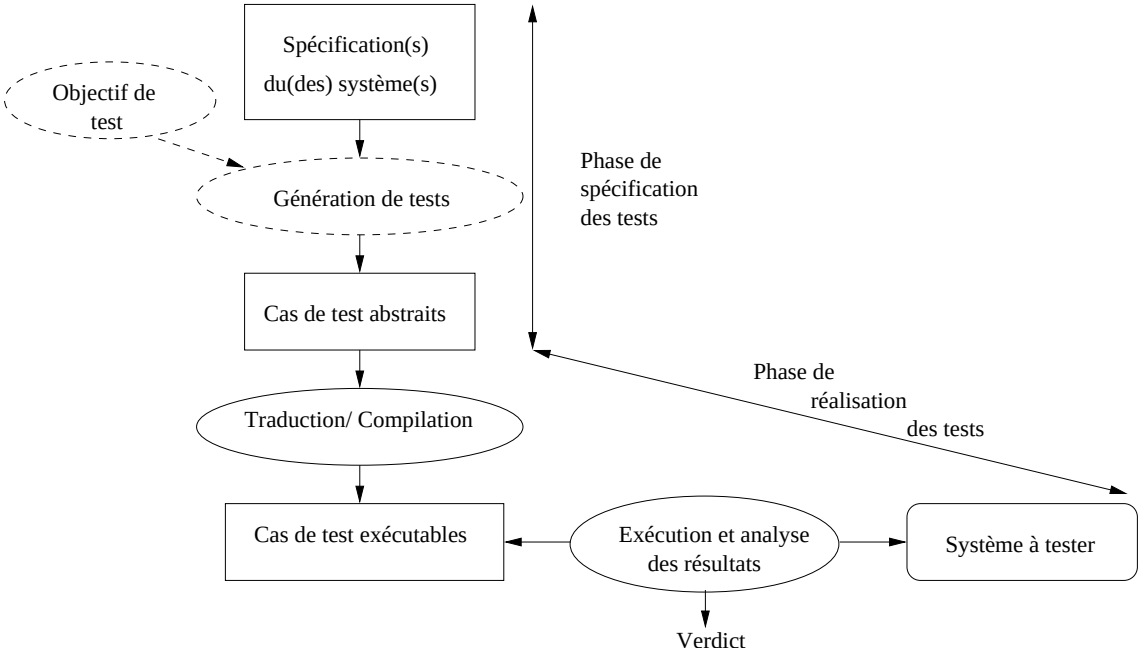


FIG. 1.1 – Étapes de l'activité de test

Quelque soit le type de test effectué, l'activité de test fait intervenir un *système sous test* (SUT pour System Under Test), un *système de test* (TS pour Test System) et l'environnement via lequel le SUT et le TS communiquent. Selon le contexte de test, le SUT est composé de une ou plusieurs *implémentations à tester* (IUT pour Implementation Under Test). L'objectif de l'activité de test est alors de générer à partir des spécifications des systèmes testés les cas de test (TC pour Test Case) exécutables sur le SUT. Généralement, cette génération de test est basée sur un objectif de test (TP pour Test Purpose). Un objectif de test décrit une propriété à vérifier lors des tests. Un cas de test est un test élémentaire destiné à tester cet objectif de test. Un ensemble de cas de test compose une suite de test. L'exécution d'un cas de test renvoie un verdict qui peut prendre les valeurs $Pass$ si l'objectif de test a été atteint et aucun erreur n'a été

observée, $Fail$ si une erreur a été observée lors des tests, ou $Inconclusive$ si le comportement observé est correct par rapport à la ou les spécifications mais ne correspond pas à l'objectif visé par le test.
Plusieurs étapes (cf. figure 1.1) sont nécessaires avant d'aboutir à l'exécution des tests sur un SUT. Deux phases principales peuvent être distinguées. La première est la phase de spécification ou de génération des tests abstraits (ATS, Abtract Test Suite) pendant laquelle les tests sont spécifiés sous une forme abstraite. La seconde phase est la phase de réalisation des tests. Cette phase regroupe l'étape de compilation des test abstraits en tests exécutable (ETS, Executable Test Suite), l'étape d'exécution des tests exécutables obtenus ainsi que l'analyse des résultats des tests. Les travaux présentés dans cette thèse concernent la première phase, c'est-à-dire la spécification d'un test abstrait à partir d'une ou plusieurs spécifications et d'un objectif de test.

Dans cette thèse, nous nous sommes intéressés au test d'interopérabilité qui met en relation plusieurs implémentations. Il n'existe pas de cadre formel pour le test d'interopérabilité, contrairement au contexte du test de conformité pour lequel il existe au moins une méthodologie normalisée [ISO94], des définitions, et des méthodes et outils de génération de test. Nous avons proposé différentes définitions formelles de la notion d'interopérabilité en fonction des contextes existant actuellement en pratique pour la réalisation de tests d'interopérabilité. Sur la base de ces définitions, nous avons ensuite développé des algorithmes pour la génération de tests.

1.2 Motivation

Dans cette thèse, nous nous sommes intéressés à l'application de Techniques de Description Formelles (FDT pour Formal Description techniques) au test d'interopérabilité comme cela a été fait dans le contexte du test de conformité. Ces deux types de tests portent sur des implémentations et leurs spécifications mais leurs objectifs sont différents. L'objectif du test de conformité est de vérifier qu'une implémentation se comporte comme prévu dans sa spécification. Les objectifs du test d'interopérabilité sont à la fois de vérifier que plusieurs implémentations (basées sur des protocoles conçus pour fonctionner ensemble) sont capables d'interagir et que, lors de leur interaction, elles rendent les services prévus dans la ou les spécifications.

A l'heure actuelle, on peut considérer que le test de conformité est relativement précisément défini. En effet, il existe de nombreux travaux dans ce domaine [VTKB92, Pha94, BP94, LY96, Tre96, Hee98, Tre99, Mor00, FS03, FTW04], mais également une norme (norme ISO/IEC IS9646 [ISO94]) définissant un cadre méthodologique pour ce test. De plus, différents travaux dans le domaine de la conformité ont permis de définir formellement la notion de conformité grâce à des relations d'implantations

[Pha94] ou des relations de conformité [VTKB92, Tre99] décrivant sous quelles conditions une implémentation peut être considérée conforme à sa spécification. A partir de ces définitions, des méthodes de génération automatique de tests de conformité ont été développées, ainsi que des outils implémentant ces méthodes (tels que TGV [JJ05], TorX [TB03], SAMSTAG, TVEDA, TestComposer, etc). Tout ceci permet une définition précise de la notion de conformité et a fait progresser le domaine du test de conformité.

Contrairement au contexte du test de conformité, il n'existe pas de cadre méthodologique normalisé pour le test d'interopérabilité malgré quelques tentatives pour donner à la notion d'interopérabilité une définition formelle [VBT01, CK94] ou de définir des algorithmes de génération de tests d'interopérabilité [BCKZ02, TKS03, EFTSY04, HLSG04]. Différentes raisons peuvent expliquer cette situation : en particulier, la conformité des implémentations est considérée comme une étape indispensable avant de vérifier si ces implémentations sont capables d'interopérer.

Actuellement, la plupart des tests d'interopérabilité sont écrits "à la main" à partir des normes décrivant le comportement attendu des implémentations. C'est le cas par exemple des tests d'interopérabilité exécutés lors de Plugtests. Un Plugtest est un évènement de tests d'interopérabilité où des constructeurs font valider l'interopérabilité de leurs produits avec les autres produits du marché. Parmi ces évènements d'interopérabilité, il existe pour les Plugtests organisés par l'ETSI [ETS] pour différents protocoles (par exemple IPv6, Mobile WiMAX, GRID Middleware and Infrastructures, Mobile applications, etc), les Plugtests [IRI] auxquels participe l'équipe DIONYSOS de l'IRISA sur les protocoles de la pile IPv6, etc. L'approche est pragmatique : les tests sont écrits manuellement à partir des spécifications et d'un objectif de test. Mais cette opération est également complexe puisque plusieurs spécifications, généralement des normes décrivant informellement le comportement du protocole, sont manipulées pour pouvoir décrire un test. La notion d'interopérabilité utilisée pour décrire les tests est une notion ad hoc par rapport à l'objectif des tests en cours. Elle peut être différente d'un test à l'autre, et est donc source d'erreur lors de l'écriture des tests. Tout cela a pour conséquence une définition souvent floue de la notion d'interopérabilité, et une non-interopérabilité effective des implémentations malgré les tests.

L'objectif de cette étude est donc de remédier à ces problèmes par une approche formelle du test d'interopérabilité, c'est-à-dire en donnant des définitions formelles de la notion d'interopérabilité, permettant de mieux définir les propriétés à vérifier lors de tests d'interopérabilité, et en définissant des méthodes de génération de tests associées. Il faut donc tout d'abord définir les différents contextes, ou architectures de test d'interopérabilité de test d'interopérabilité que l'on peut rencontrer en pratique (observées par exemple lors des Plugtests). Ensuite il faut définir formellement la no-

tion d'interopérabilité dans ces différents contextes ; ces définitions doivent également aider à avoir une définition claire de la notion d'interopérabilité. Sur la base de ces définitions, il est possible de développer des méthodes de génération automatique de tests d'interopérabilité qui peuvent permettre à la fois une meilleure confiance dans les suites de tests développées, mais également un gain de temps grâce à l'automatisation de la génération de tests.

1.3 Contribution

Les travaux décrits dans cette thèse font suite à des travaux sur l'interopérabilité déjà réalisés dans l'équipe [VBT01] précisant les différentes architectures de test pouvant être utilisées lors du test d'interopérabilité de deux implémentations, et donnant quelques pistes sur la définition formelle de la notion d'interopérabilité correspondant à ces contextes. Les travaux présentés dans cette thèse portent sur deux contextes différents : le test d'interopérabilité de deux implémentations (aussi appelé contexte one-to-one) et le test d'interopérabilité dit multi-implémentations ou multiparties qui met en relation N implémentations ($N > 2$).

1.3.1 Contexte d'interopérabilité one-to-one

Dans un premier temps, nous nous sommes intéressés au contexte de test d'interopérabilité one-to-one qui se concentre sur l'interaction de deux implémentations. Ce contexte constitue la majeure partie de l'activité actuelle de test d'interopérabilité : en effet, la plus grande partie des tests d'interopérabilité actuels consiste à vérifier soit l'interopérabilité de deux implémentations, soit l'interopérabilité d'une implémentation avec un système déjà opérationnel composé de plusieurs implémentations (pouvant être vu comme la deuxième implémentation lors des tests). C'est donc aussi sur ce contexte que porte la majorité du travail présenté dans cette thèse. Ce travail peut être décomposé en deux parties :

- la définition formelle de la notion d'interopérabilité (chapitre 3)
- l'utilisation de ces définitions pour développer des méthodes de génération automatique de tests d'interopérabilité (chapitre 4)

La définition formelle de la notion d'interopérabilité nécessite au préalable d'identifier les différents contextes dans lesquels l'interopérabilité de deux implémentations peut être testée. Ensuite, nous avons défini les objectifs du test d'interopérabilité à prendre en compte et les modèles formels associés à ces objectifs. A partir de cette étude, nous avons pu définir des *critères d'interopérabilité* pour le contexte du test d'interopérabilité one-to-one. Un critère d'interopérabilité définit formellement sous quelles conditions deux implémentations peuvent être considérées interopérables. Ces

critères décrivent des conditions sur les implémentations en interaction, mais prennent également en compte les actions prévues dans les spécifications de ces systèmes. En effet, l'interopérabilité vérifie également, en plus de la capacité des implémentations à communiquer entre elles, que le service prévu dans les spécifications est effectivement rendu pendant l'interaction des implémentations.

Différents critères d'interopérabilité ont été définis en fonction des différents contextes (représentés par les architectures de test utilisables) possibles pour le test d'interopérabilité. Il convient ensuite de comparer le pouvoir de ces différents critères en fonction de leur capacité à détecter la non-interopérabilité des implémentations. Le choix du critère d'interopérabilité utilisé ensuite pour décrire des tests d'interopérabilité sera fonction de paramètres tels que l'observabilité des différentes interfaces des systèmes ou l'exigence d'interopérabilité requise lors des tests (en fonction des résultats des autres tests déjà réalisés par exemple). Nous donnons également dans le chapitre 3 des éléments pour le choix du critère d'interopérabilité en fonction de l'architecture de test.

Lors de cette comparaison entre critères d'interopérabilité, nous nous sommes aperçu que certains critères étaient équivalents en terme de détection de la non-interopérabilité. En particulier, nous avons démontré une équivalence entre un critère correspondant à un contexte de test d'interopérabilité distribué *sans* synchronisation et un critère correspondant à un contexte de test d'interopérabilité distribué *avec* synchronisation. Or, le problème le plus souvent rencontré lors de la génération de tests d'interopérabilité avec les méthodes classiques (dont la dérivation manuelle) est l'explosion combinatoire générée par la construction (complète ou partielle) d'une vue globale du SUT composé des implémentations interconnectées. Ceci suggère la possibilité d'utiliser cette équivalence entre critères d'interopérabilité pour tenter d'éviter ce problème d'explosion combinatoire.

A partir des critères d'interopérabilité définis et de l'équivalence décrite ci-dessus, nous avons défini une méthode pour générer automatiquement des tests d'interopérabilité dans le contexte one-to-one. Cette méthode évite le problème d'explosion du nombre d'états qui est généralement rencontré lors des approches classiques (vue globale du SUT) de génération de tests d'interopérabilité. Cette méthode répond aux deux exigences du test d'interopérabilité : vérification à la fois de l'interaction et de la fourniture du ou des services pendant l'interaction. De plus, elle permet, sous certaines conditions de contrôlabilité des cas de test, de réutiliser certains outils développés dans le cadre du test de conformité. En cas de besoin, elle peut également être utilisée pour dériver des méthodes de génération de tests d'interopérabilité dans des contextes où les testeurs n'ont pas accès à toutes les interfaces des implémentations.

Cette méthode a été implémentée avec les API de représentation d'IOLTS de l'outil CADP [GLM01] et appliquée sur des exemples dont des protocoles réels. Les cas de test d'interopérabilité ainsi générés par cette méthode ont été comparés aux cas de test générés par une approche classique. Cette expérimentation a confirmé que la méthode développée est applicable en pratique pour générer des cas de test d'interopérabilité ayant la même capacité de détection de la non-interopérabilité que les approches classiques, et qu'elle évite le problème d'explosion combinatoire rencontré généralement avec ces approches.

1.3.2 Contexte d'interopérabilité multi-implémentations

Le contexte one-to-one n'est pas le seul contexte d'interopérabilité abordé dans cette thèse. Nous nous sommes également intéressés au contexte multi-implémentations mettant en relation N implémentations. Ce contexte est rencontré lorsqu'il est impossible de s'appuyer sur un contexte one-to-one pour tester l'interopérabilité des implémentations, comme par exemple lorsqu'aucune des entités n'a atteint un niveau satisfaisant dans son développement. C'est donc un contexte qui met en relation N ($N > 2$) implémentations, mais surtout qui ne peut pas être ramené à du test d'interopérabilité one-to-one en représentant $N - 1$ des N implémentations comme un système global interagissant avec l'implémentation de numéro N. Actuellement, le test d'interopérabilité dans ce contexte multi-implémentations est le plus souvent réalisé en connectant les N implémentations entre elles, et en observant puis analysant les traces générées lors de cette interconnexion, c'est-à-dire sans définir préalablement des cas de test. Ce mode de test est appelé test passif (ou passive testing). Son utilisation pour le test d'interopérabilité multi-implémentations est due à la complexité des objets manipulés : N implémentations, donc N spécifications et donc des cas de test pouvant contenir des évènements exécutables sur N systèmes différents lors des tests.

Dans le chapitre 5 de cette thèse, nous avons étudié quelles définitions du contexte one-to-one pouvaient être généralisées et quels concepts devaient être définis ou redéfinis pour ce contexte plus général de test d'interopérabilité. De même que dans le contexte one-to-one, une approche formelle du test d'interopérabilité multi-implémentations doit s'intéresser successivement aux architectures de test possibles dans ce contexte, à la définition de la notion d'interopérabilité, puis à la génération de test. Le chapitre 5 de cette thèse porte principalement sur les étapes jusqu'à la définition formelle de la notion d'interopérabilité de N implémentations.

Tout d'abord, nous nous sommes intéressés aux différents contextes dans lesquels l'interopérabilité de N implémentations peut être testée. Ces contextes peuvent dans certains cas être rapprochés de contextes de l'interopérabilité de 2 implémentations (contexte global par exemple). Nous avons ainsi donné une classification des diffé-

rentes architectures de test d'interopérabilité qu'il est possible d'utiliser pour tester l'interopérabilité de N implémentations.

Une fois, la définition des différents contextes effectuées, nous avons eu à gérer un concept qu'il n'est pas nécessaire de prendre en compte dans le contexte one-to-one, mais dont l'influence est très importante dans le contexte multi-implémentation : la topologie d'interconnexion des implémentations à tester. En effet, dans le contexte one-to-one, les deux implémentations sont forcement connectées l'une à l'autre. Dans le contexte multi-implémentations, chaque implémentation est au moins connectée à une autre implémentation du réseau, mais différentes configurations sont possibles en fonction des protocoles testés. Nous avons donc dû modéliser la topologie servant à connecter les implémentations pour pouvoir la prendre en compte dans les tests d'interopérabilité, mais aussi étudier l'influence de la configuration utilisée dans les différents concepts rencontrés lors d'une approche formelle du test d'interopérabilité de N implémentations.

Sur le plan de la formalisation, certains concepts ont du être repris et modélisés pour prendre en compte la topologie et le nombre d'implémentations interconnectées lors du test d'interopérabilité. Ainsi, nous avons défini de nouveaux critères d'interopérabilité adaptés au contexte multi-implémentations qui peuvent être vu comme une généralisation des critères d'interopérabilité one-to-one. Nous donnons également quelques pistes pour le génération de tests d'interopérabilité multi-implémentations.

1.4 Plan

Dans le chapitre 2, nous présentons un état de l'art du domaine. La première section (section 2.1) de ce chapitre est consacrée aux méthodes de test d'implémentations de protocoles. Puis la section 2.2 s'intéresse aux différents objets considérés lors d'une activité de test. La section 2.3 porte ensuite sur les modèles existants pour modéliser les objets intervenant dans l'activité de test et en particulier le modèle (et définitions associées) utilisé dans la suite du document. Ensuite, nous nous concentrons sur le processus de formalisation du test de conformité, l'apport de cette formalisation et les outils dérivés (section 2.4). Puis nous faisons dans la section 2.5 un état de l'art des définitions, architectures de test et méthodes existants dans le domaine du test d'interopérabilité.

Le chapitre 3 porte sur la formalisation de la notion d'interopérabilité dans le cadre de l'interaction de deux implémentations. Dans un premier temps (section 3.2), nous récapitulons les architectures de test d'interopérabilité existant dans ce contexte et leur influence sur une définition formelle de l'interopérabilité. Puis certaines définitions as-

sociées au modèle utilisé doivent être complétées ou redéfinies pour prendre en compte les particularités du test d'interopérabilité : c'est l'objectif de la section 3.3. La section 3.4 porte sur le modèle des spécifications et les propriétés que ces dernières doivent vérifier pour permettre l'interopérabilité des implémentations correspondantes. Les sections 3.5 et 3.6 sont consacrées à donner des définitions formelles de la notion d'interopérabilité. Différentes définitions sont données pour prendre en compte les différents contextes d'observation et de contrôle des implémentations. Les sections 3.7 et 3.8 de ce chapitre sont ensuite consacrées à appliquer et à étudier les propriétés de ces définitions formelles de la notion d'interopérabilité. Enfin, la section 3.9 donne des éléments pour aider au choix d'un critère d'interopérabilité pour le test.

Le chapitre 4 porte sur les approches pour la génération automatique de tests d'interopérabilité développées à partir des définitions formelles du chapitre 3. Dans la section 4.2, nous rappelons tout d'abord les définitions des objets intervenant dans l'activité de test, puis nous adaptons les modèles formels de ces objets au contexte du test d'interopérabilité. Les sections 4.3, 4.4 et 4.5 décrivent ensuite différentes méthodes et approches de génération automatique de tests d'interopérabilité basées sur les définitions formelles du chapitre 3. La section 4.6 résume ensuite la méthode proposée pour la génération de cas de test d'interopérabilité. Puis, la section 4.7 s'intéresse au lien entre la méthode décrite et une approche distribuée du test d'interopérabilité. Ensuite, la section 4.8 est consacrée à l'application de ces méthodes sur des exemples et à l'interprétation des résultats obtenus. Puis la section 4.9 décrit une amélioration possible de la méthode proposée et la section 4.10 s'intéresse aux autres utilisations possibles de la méthode proposée. Enfin, la section 4.11 conclut ce chapitre sur la génération de test d'interopérabilité one-to-one basée sur des définitions formelles de la notion d'interopérabilité.

Le chapitre 5 porte sur la définition formelle de la notion d'interopérabilité dans le contexte plus général de l'interaction de N implémentations. Tout d'abord, dans la section 5.2, nous nous intéressons aux architectures de test d'interopérabilité possibles dans ce contexte multi-implémentations. Puis la section 5.3 porte sur les problèmes associés au choix de topologie ; en particulier, il est nécessaire de modéliser la topologie et d'étudier le niveau d'influence du mode de connexion des N implémentations sur le test d'interopérabilité. Ensuite, dans la section 5.4, nous adaptons quelques définitions au contexte du test d'interopérabilité multi-implémentations. Puis, dans la section 5.5, nous nous intéressons aux propriétés des spécifications considérées pour le test d'interopérabilité. La section 5.6 porte ensuite sur les définitions formelles de la notion d'interopérabilité de N implémentations. Enfin, ces définitions sont appliquées sur des exemples dans la section 5.7, et nous donnons des pistes pour la génération de test dans ce contexte dans la section 5.8.

Le chapitre 6 propose une synthèse des travaux présentés dans ce documents, ainsi que des perspectives pour des travaux futurs.

Chapitre 2

Contexte et état de l'art : méthodes et modèles pour le test de protocoles

2.1 Introduction

Le but de l'activité de test est de vérifier qu'un produit rend bien le(s) service(s) prévu(s). Dans le domaine des protocoles réseaux, le ou les services sont décrits dans la *spécification* du produit. Cette spécification est généralement une norme ou un standard défini par des organismes de normalisation tels que l'ITU (International Telecommunication Union), l'ISO (International Standards Organisation), l'IEEE (Institute of Electrical and Electronics Engineers), l'IETF (Internet Engineering Task Force), etc. Ces normes sont le cahier des charges du produit considéré. Ce produit est appelé *implémentation* du protocole.
Différents types de *test* sont définis dans la littérature et peuvent être utilisés pour vérifier que les implémentations de protocoles fonctionneront correctement dans un contexte opérationnel. Ces types de test varient en fonction des objectifs qu'ils visent. Parmi les tests utilisés pour vérifier le comportement des implémentations de protocoles des réseaux informatiques, on peut distinguer les tests suivants.

Test de conformité : l'objectif du test de conformité (cf. figure 2.1) est de vérifier qu'une implémentation est conforme à sa spécification. L'implémentation (ou IUT pour Implementation Under Test) n'est connue que par ses interactions avec l'environnement via les évènements exécutés à ses interfaces : c'est le contexte de test dit "boîte noire". De nombreux travaux ont été réalisés dans le domaine de la formalisation du test de conformité (par exemple [Tre99, VTKB92, Pha94]) qui est précisément défini et normalisé par l'ISO/IEC dans la norme ISO9646 [ISO94].

Test d'interopérabilité : le test d'interopérabilité met en relation plusieurs implémentations. La figure 2.1 décrit le contexte de test d'interopérabilité comportant

deux implémentations appelé one-to-one. Chacune de ces implémentations est une boîte noire. L'objectif du test d'interopérabilité est alors de s'assurer à la fois que les implémentations interagissent (c'est-à-dire communiquent) correctement et qu'elles rendent les services prévus dans leur spécification pendant leur interaction. Contrairement au test de conformité, il n'existe actuellement pas de cadre formel ou de norme définissant précisément le test d'interopérabilité.

Autres types de tests : D'autres types de tests sont également applicables aux implémentations de protocoles tels que le test de robustesse ou le test de performances. L'objectif du test de robustesse est de vérifier le comportement de l'implémentation dans des conditions anormales de fonctionnement : tentatives d'utilisation abusives, pannes, erreurs de manipulation, etc. Le test de performance sert à vérifier des paramètres de performances de l'IUT tels que les délais de transmission, le débit maximal, le nombre de connexion supportées en parallèle.
Ces types de tests (conformité, interopérabilité, robustesse et performances) s'intéressent au comportement d'implémentations observées à leurs interfaces. Il existe également d'autres types de tests tels que, par exemple, les tests structurels qui s'intéressent au code des implémentations.

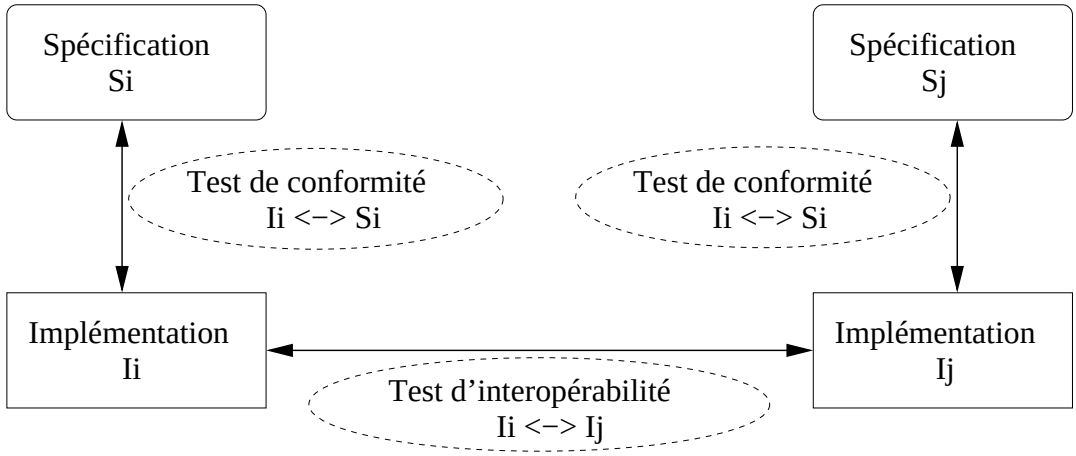


FIG. 2.1 – Tests de conformité et d'interopérabilité

Dans cette thèse, nous nous sommes intéressés plus particulièrement au test d'interopérabilité (parfois également appelé en français test d'interfonctionnement). Contrairement au test de conformité pour lequel un cadre méthodologique formel existe, le domaine du test d'interopérabilité souffre d'un manque de formalisme malgré quelques tentatives de définitions formelles [CK94, VBT01] ou de méthodes permettant de générer automatiquement des cas de test d'interopérabilité [BCKZ02, SKKR03, HLSG04, EFTSY04]. Nous étudions ici la possibilité d'appliquer des méthodes formelles au contexte du test d'interopérabilité. La première étape est de déterminer les différents

contextes dans lesquels des tests d'interopérabilité peuvent être exécutés. Une classification a été faite pour l'interopérabilité de deux implémentations dans [VBT01] (voir section 2.5.3). Nous reprenons cette classification et la précisons et complétons dans la suite. A partir de cette classification, nous pouvons définir formellement la notion d'interopérabilité, puis développer, à partir des définitions obtenues, des algorithmes permettant de décrire une méthode de génération de tests d'interopérabilité adaptée au contexte rencontré.

L'organisation de ce chapitre est la suivante.

Tout d'abord, la section 2.2 s'intéresse aux étapes et objets impliqués dans une activité de test tels que cas de test, objectifs de test ou verdicts. Nous donnons les définitions de ces notions qui pourront être précisées dans les chapitres suivants pour tenir compte des particularités du test d'interopérabilité.

Ensuite, nous nous intéressons aux modèles permettant de représenter les objets considérés par le test d'interopérabilité. Ces modèles sont présentés dans la section 2.3. Ces modèles ont été définis dans le cadre de la formalisation du test de conformité. En effet, comme indiqué notamment dans [GRSS90, APRS93, WP94, KSM96, VBT01], le test de conformité et le test d'interopérabilité, bien qu'ayant des objectifs différents, comportent quelques similitudes. En particulier, la conformité et l'interopérabilité portent sur les mêmes objets : spécifications, implémentations, testeurs. Les modèles formels utilisés dans le domaine du test de conformité pour représenter un certain nombre d'objets peuvent donc être réutilisés dans le cadre de l'interopérabilité.

Dans la section 2.4, nous nous intéressons au cadre formel existant pour le test de conformité. En effet, ce type de test est précisément défini avec des architectures de test, définitions formelles, algorithmes et outils de génération automatiques de test. Il est donc intéressant d'étudier au préalable les définitions formelles et méthodes de génération de test de conformité avant de décrire de telles définitions et algorithmes pour le test d'interopérabilité.

La dernière section de ce chapitre (section 2.5) s'intéresse à l'état de l'art du test d'interopérabilité. Nous nous intéressons aux définitions de la notion d'interopérabilité et du test d'interopérabilité existant dans la littérature, ainsi qu'aux différents contextes ou architectures de test dans lequel des tests d'interopérabilité peuvent être opérés. Puis, nous décrivons les définitions formelles existants pour la notion d'interopérabilité et les méthodes et approches de génération automatique de tests d'interopérabilité qui ont été développées.

2.2 Activité de test

2.2.1 Tests de conformité et d'interopérabilité

Le test d'interopérabilité et le test de conformité sont deux types de tests utilisés dans le but de vérifier que la ou les implémentations fonctionneront dans un contexte opérationnel. Les objectifs de ces deux types de test sont différents :

- le test de conformité s'intéresse à une implémentation prise isolement et compare cette implémentation avec sa spécification.
- le test d'interopérabilité se focalise sur l'interaction de plusieurs implémentations. Ces implémentations doivent agir conformément à leur spécification (donc rester conforme pendant l'interaction), mais surtout communiquer correctement.

Malgré cette différence d'objectifs, les tests d'interopérabilité et de conformité comportent également des points communs.

- Dans les deux types de test, les systèmes testés (une IUT en conformité ou plusieurs implémentations interconnectées en interopérabilité) ne sont connus que par les évènements exécutés sur leurs interfaces.
- Ces tests portent sur les mêmes objets : spécifications, implémentations, testeurs.
- Les deux types de test se basent sur la spécification (resp. sur les spécifications en interopérabilité) du système (resp. des systèmes en interopérabilité) à tester pour la description des cas de test à exécuter.

Les différents objets intervenant dans l'activité de test ont été précisément définis et modélisés dans le cadre de la formalisation du test de conformité. Certaines de ces définitions peuvent être réutilisées dans le domaine du test d'interopérabilité, d'autres devront être soit redéfinis, soit remodélisés. Dans la section 2.2.2, nous donnons la définition de quelques concepts existants à la fois dans le test de conformité et d'interopérabilité. Les définitions de cette section sont celles du test de conformité, plus précisément de la norme ISO/IS 9646 [ISO94]. La plupart de ces concepts ont la même définition générale en conformité et en interopérabilité : c'est leur mise en pratique et le modèle associé qui est différent selon le contexte. C'est pourquoi nous donnons ici seulement les définitions littérales de ces notions et objets. Les définitions formelles seront adaptées au contexte du test d'interopérabilité et définies dans les chapitres suivants.

2.2.2 Activité de test : notions et objets

Dans cette section, nous nous intéressons aux différentes étapes de l'activité de test et aux notions et objets correspondants. La plupart des concepts et définitions pour l'activité de test de conformité décrit dans cette section provient de la norme ISO/IS

9646 [ISO94].
Les différentes étapes lors d'une approche formelle de l'activité de test sont :

1. la spécification du système (composé de une ou plusieurs implémentations) et la description des propriétés à tester (objectifs de test) sur le système. Cette phase de spécification comprend la description des objets en présence avant la génération de test, c'est-à-dire la spécification du ou des systèmes à tester, la configuration dans laquelle le ou les systèmes vont être testés (architecture de test), la description du ou des objectifs de la suite de tests, etc.
2. la génération des cas de test abstraits décrivant les actions que le ou les testeurs doivent exécuter ou observer. A partir des objets décrits à l'étape précédente, des cas de tests abstraits (formant une suite de tests) sont générés. Ils décrivent la suite des actions nécessaires pour atteindre l'objectif d'un test spécifique avec les verdicts associés.
3. la compilation des tests abstraits en test exécutables et l'exécution de ces tests sur le système sous test (ou SUT pour System Under Test). Cette étape permet d'obtenir les cas de tests exécutables sur le SUT. Le résultat de l'exécution d'un test sur le SUT se traduit alors par l'attribution d'un verdict. Le SUT est composé d'une implémentation dans le cadre du test de conformité, et de plusieurs implémentations interconnectées dans le cadre du test d'interopérabilité.

Remarque: Dans la littérature, il existe différentes définitions pour un SUT (System Under Test) et une IUT (Implementation Under Test). Dans la suite, nous utiliserons ces notions dans les cas suivants (adaptés au contexte du test d'interopérabilité) :

- IUT : une IUT désigne une implémentation à tester.
- SUT : un SUT désigne un système sous test composé de plusieurs implémentations interconnectées. Il y a donc plusieurs IUTs dans un SUT en test d'interopérabilité.

Nous nous intéressons dans ce document plus particulièrement aux phases 1 et 2 jusqu'à la description des cas de test abstraits et des verdicts associés. Nous précisons ici les définitions des principaux objets considérés : objectifs de test, cas de test et verdicts.

Objectif de test Un objectif de test (ou TP pour Test Purpose) décrit de manière informelle quelle est la propriété particulière ou le comportement spécifique à tester. Il sert à sélectionner un cas de test afin de tester un aspect particulier du système sous test. Il décrit généralement un enchaînement d'actions (non nécessairement consécutives) à observer durant l'exécution des tests.

Cas de test Un cas de test est un test élémentaire destiné à tester une fonctionnalité particulière (l'objectif de test). Dans le cas du test de conformité, un cas de test est

composé de stimuli de test (messages envoyés par le testeur au SUT) et d'évènements du SUT (les réponses à ces stimuli). Les évènements possibles dans un cas de test d'interopérabilité sont décrits dans la section 4.2.2 avec le modèle formel associé.
Un cas de test comprend généralement un *préambule*, un *corps de test* et un *postambule*. Le corps de test est l'ensemble des évènements qui sont essentiels pour atteindre l'objectif de test. Le préambule (ou préface) est la séquence des évènements de test nécessaires pour définir le passage de l'état de test stable (début du cas de test élémentaire) à l'état initial d'où démarrera le corps du test. Le postambule (ou postface) est la séquence des évènements de test nécessaires pour définir le passage du corps du test à un état stable correspondant à la fin du cas de test élémentaire. Le cas de test contient également les verdicts associés à ses possibles exécutions.

Verdicts Le résultat de l'exécution d'un test sur le SUT (système à tester) se traduit par l'attribution d'un verdict. Le verdict peut prendre les valeurs "succès" (PASS), "échec" (FAIL) ou "non concluant" (INC ou INCONCLUSIVE). Nous donnons ici la sémantique de ces différents verdicts. Cette sémantique est la même quelque soit le type de test considéré. Cependant, comme tenu des différences d'objectifs entre les différents tests d'implémentations de protocoles, cette sémantique illustre des propriétés (vérifiée ou non) différentes sur les implémentations testées.

PASS Le verdict PASS ou "succès" signifie que rien d'incorrect (par rapport à l'objectif de test) n'a été observé lors de l'exécution des tests. De plus, l'objectif de test a été atteint.

FAIL Le verdict FAIL ou "échec" signifie que le résultat de test observé ne correspond pas à ce qui est prévu. Le résultat observé contient au moins un élément invalide par rapport à la ou aux spécifications correspondantes.

INCONCLUSIVE Le verdict INCONCLUSIVE ou "non concluant" signifie que ni le verdict PASS, ni le verdict FAIL ne peuvent être rendus. En effet, comme il est impossible de contrôler totalement le SUT, le comportement observé peut à la fois être prévu dans la ou aux spécifications correspondantes et ne pas correspondre à l'objectif de test visé par le cas de test.

2.3 Modèles et principales notations

Les approches formelles pour des tests tels que les tests de conformité ou d'interopérabilité s'appuient sur des modèles des différents objets considérés. En effet, ces types de tests vérifient sur les systèmes à tester (les implémentations) des propriétés des spécifications correspondantes. Il faut donc définir un modèle pour représenter les spécifications. Bien que leur comportement ne soit pas supposé connu, il est également nécessaire de disposer d'un modèle de description des comportements possibles des implémentations.

2.3.1 Modélisation des spécifications et implémentations

Différents modèles peuvent être utilisés pour représenter formellement une spécification de protocole : il existe des modèles de type algèbre de processus, automates, etc. Dans cette thèse, nous nous intéressons plus précisément aux modèles de type automate qui permettent de représenter les spécifications et les implémentations par une successions d'évènements d'entrées et sorties, évènements qui sont observés aux interfaces des implémentations dans un contexte de test "boîte noire". Deux groupes de modèles sous forme d'automates sont utilisés dans le domaine du test [BP94, Jér01]. Ces modèles sont les modèles de type *FSMs* (Finite State Machine) [LY96] et *LTSs* (Labelled Transition Systems) [Pha94, VTKB92].

– Un FSM est un automate de type machine de Mealy dont chaque transition est étiquetée par une entrée et une sortie. Différentes variantes des FSMs sont utilisées dans la littérature [Pet00] : FSM déterministe, FSM non-déterministe, Input/Output FSM, EFSM (Extended Finite State Machine), CFSM (Communicating Finite State Machine), CEFSM (Communicating Extended Finite State Machine), etc. Un FSM est déterministe s'il n'existe, pour un état donné et une entrée, qu'une seule sortie et un seul état suivant possibles. Un EFSM est un FSM qui intègre la manipulation de variables et de prédicats sur ces variables. Les CFSM (modèle utilisé par exemple dans [TKS03, PYvBD96]) et CEFSM (utilisé par exemple dans [BPGQ02]) sont des FSMs utilisés pour décrire l'échange entre systèmes via un canal de communication.
  Ce type de modèles associant à une transition un couple entrée/sortie est plus particulièrement utilisé pour modéliser des systèmes déterministes associant à chaque entrée une unique sortie de l'implémentation. La modélisation de d'autres types de systèmes avec un modèle de type FSM nécessite la manipulation de FSMs non-déterministes incluant une sortie appelée "null" modélisant l'absence de sortie, ce qui rend l'utilisation d'un modèle de type FSM complexe pour de tels systèmes.
– Un LTS est un *système de transitions étiquetées*, c'est-à-dire un automate qui associe à chaque transition une étiquette représentant un évènement. Différentes variantes des LTSs ont également été définies selon les besoins du domaine du test [BT00] : IOLTS (Input-Output Labelled Transition Systems) [Pha94, VTKB92], IOTS (Input-Output Transition Systems) [Tre92], MIOTS (Multiple Input-Output Transition Systems) [Hee98], etc. Cet évènement peut être de type entrée (input), sortie (output) ou évènement interne (non exécuté sur les interfaces du systèmes). Un IOLTS est une version de LTS distinguant explicitement les évènements d'entrée et de sortie. Les autres modèles de type LTS sont définis en fonction de propriétés attendues des systèmes modélisés. Par exemple, un IOTS est un LTS distinguant explicitement les entrées des sorties et pour lequel toutes les entrées sont toujours exécutables dans tous les états du système.

Cet ensemble de modèles est plus particulièrement utilisé pour modéliser des systèmes asynchrones qui peuvent exécuter indifféremment une entrée ou une sortie dans un état donné.

Le point commun entre les modèles de type LTS et les modèles de type FSM est qu'ils modélisent les systèmes (spécifications et/ou implémentations) par un ensemble d'états et d'actions associées aux transitions entre états. La principale différence se situe donc dans le mode d'étiquetage des transitions. En effet, une transition d'un FSM est étiquetée par un évènement constitué d'une entrée *et* d'une sortie, c'est-à-dire que le système ne peut pas accepter de nouvelle entrée tant qu'il n'a pas produit de sortie en réaction à l'entrée précédente (à moins de considérer une sortie "null" représentant l'absence de sortie, ce qui alourdit la représentation du système). Par contre, un IOLTS (LTS partitionnant l'ensemble des actions en ensembles d'entrées et de sorties) peut avoir des transitions étiquetées soit par une entrée soit par une sortie. Ceci permet de modéliser un comportement asynchrone de systèmes pouvant exécuter plusieurs entrées ou plusieurs sorties successivement.

Dans le domaine des réseaux, les protocoles sont généralement des systèmes réactifs asynchrones. C'est-à-dire que plusieurs sorties peuvent être générées en réaction à une entrée, ou que plusieurs entrées peuvent être nécessaires avant l'envoi d'une sortie. De plus, dans le contexte du test d'interopérabilité, il est également nécessaire de pouvoir modéliser les communications asynchrones entre systèmes et de décrire précisément sur quelles interfaces sont exécutés les différents évènements. Comme la communication entre entités de réseaux est généralement asynchrone, il est également nécessaire de distinguer explicitement les évènements d'entrées (inputs), les évènements de sorties (outputs) et les évènements internes.

Pour toutes ces raisons (et en particulier le caractère asynchrone des systèmes considérés), nous avons choisi d'utiliser dans cette thèse le modèle des IOLTS (Input-Output Labelled Transition Systems) pour modéliser les différents objets considérés (spécifications, IUT, cas de test, etc). En effet, il permet la distinction explicite entre les évènements d'entrées (inputs), les évènements de sorties (outputs) et les évènements internes tout en décrivant précisément l'interface d'exécution des évènements.

Remarque: Ce modèle a, entre autres, été utilisé pour la définition des "relations de conformité" [Tre92, Tre99, Pha94] ou dans quelques unes des études formelles sur l'interopérabilité [VBT01, CK94].

2.3.2 Modèle des systèmes de transitions à entrées et sorties (IOLTS)

Le modèle des systèmes de transitions à entrées et sorties (IOLTS pour Input-Output Labelled Transition System) [VTKB92] est une adaptation du modèle des systèmes de transitions étiquetées (LTS pour Labelled Transition System). Un IOLTS

différencie trois types d'évènements : les entrées (ou réception de message sur une interface du système), les sorties (comme l'envoi d'un message via une interface du système) et les évènements internes (actions non exécutées sur une interface du système comme la mise à jour d'une variable ou d'une horloge).

Définition 2.1 (Système de transitions à entrées et sorties (IOLTS))
Un système de transitions à entrées et sorties (IOLTS pour Input-Output Labelled Transition System) est un quadruplet M = $(Q^M, \Sigma^M, \Delta^M, q_0^M)$ *tel que :*

- Q^M *est l'ensemble (non vide) des états du système*
- $q_0^M \in Q^M$ *est l'état initial*
- Σ^M *représente l'ensemble des évènements exécutables (entrées et/ou sorties) aux points d'interactions (ou interfaces) du système.*
  $\Sigma^M \subseteq P^M \times \{?, !\} \times A^M$ *où* P^M *est l'ensemble fini des interfaces (ports) via lesquelles le système communique avec son environnement ou avec d'autres systèmes, " ?" et " !" représentent respectivement une entrée (réception) et une sortie (envoi) de message,* A^M *est l'alphabet de messages d'entrée-sortie échangés par le système via ses interfaces. Ainsi,* $p?a$ *(respectivement* $p!a$*) représente la réception (respectivement l'envoi) du message* a *via l'interface* p*.*
- $\Delta^M \subseteq Q^M \times (\Sigma^M \cup \{\tau\}) \times Q^M$ *est la relation de transition du système.* $\tau \notin A^M$ *représente un évènement interne (non observable).* $(q, \alpha, q') \in \Delta^M$ *peut également être noté* $q \xrightarrow{\alpha}_M q'$ *(ou* $q \xrightarrow{\alpha} q'$ *s'il n'y a pas d'ambiguïté sur l'IOLTS concerné).*

Σ^M peut être décomposé de façon à distinguer les évènements d'entrée et les évènements de sortie : $\Sigma^M = \Sigma_I^M \cup \Sigma_O^M$ où Σ_I^M est l'ensemble des évènements d'entrée et Σ_O^M celui des évènements de sortie.

Considérons un IOLTS M, un évènement observable sur une interface du système $\alpha \in \Sigma^{\text{M}}$ tel que $\alpha = p.\{?, !\}.m$, un évènement exécutable par le système $\mu_i \in \Sigma^{\text{M}} \cup \{\tau\}$, une succession (ou trace) d'évènements $\sigma \in (\Sigma^{\text{M}})^*$, et $q, q', q_i \in Q^{\text{M}}$ des états de M. Nous définissons les notations[1] :

- $q \xrightarrow{\mu_1 \ldots \mu_{\text{n}}}_{\text{M}} q'$ signifie $\exists\ q_0 = q, q_1 ..., q_n = q'$ tels que $q_{i-1} \xrightarrow{\mu_i}_{\text{M}} q_i$
- $q \overset{\epsilon}{\Rightarrow}_{\text{M}} q'$ signifie soit que $q = q'$, soit que $q \xrightarrow{\tau \ldots \tau}_{\text{M}} q'$
- $q \overset{\alpha}{\Rightarrow}_{\text{M}} q'$ pour $\alpha \neq \epsilon$ signifie $\exists\ q_1, q_2 : q \overset{\epsilon}{\Rightarrow}_{\text{M}} q_1 \xrightarrow{\alpha}_{\text{M}} q_2 \overset{\epsilon}{\Rightarrow}_{\text{M}} q'$.
- $q \overset{\sigma}{\Rightarrow}_{\text{M}} q'$ signifie $\exists\ q_0 = q, q_1 \ldots, q_n = q'$ tels que $\forall i \in [1, n], q_{i-1} \overset{\mu_i}{\Rightarrow}_{\text{M}} q_i$ et $\sigma = \mu_1 \cdots \mu_n$, c'est-à-dire que $q \overset{\mu_1 \cdots \mu_n}{\Longrightarrow}_{\text{M}} q'$
- $Out(q) =_\Delta \{\alpha \in \Sigma_{\text{o}}^{\text{M}} \mid \exists\ q' \text{ and } q \xrightarrow{\alpha}_{\text{M}} q'\}$ est l'ensemble des sorties possibles dans l'état q. De même, $In(q) =_\Delta \{\alpha \in \Sigma_{\text{i}}^{\text{M}} \mid \exists\ q' \text{ and } q \xrightarrow{\alpha}_{\text{M}} q'\}$ est l'ensemble des entrées que peut traiter un système dans l'état q, et $\Gamma(q) =_\Delta \{\alpha \in \Sigma^{\text{M}} \mid \exists\ q' \text{ and } q \xrightarrow{\alpha}_{\text{M}} q'\}$ est l'ensemble des entrées et sorties exécutables dans l'état q.

[1] Dans ces définitions et dans la suite, $=_\Delta$ signifie "par définition".

- q ***after*** $\sigma =_\Delta \{q' \in Q^{\text{M}} \mid q \stackrel{\sigma}{\Rightarrow}_{\text{M}} q'\}$ est l'ensemble des états qui peuvent être atteints à partir de l'état q par la suite d'actions σ. Par extension, l'ensemble des états atteignables à partir de l'état initial de l'IOLTS M est $(q_0^{\text{M}}$ ***after*** $\sigma)$ et est représenté par $(M$ ***after*** $\sigma)$. De même, $Out(M,\sigma) =_\Delta Out(M$ ***after*** $\sigma)$ est l'ensemble des sorties exécutables par le système M après la trace σ, $\Gamma(M,\sigma)$ l'ensemble des évènements (entrées et sorties) exécutables par M sur ses interfaces, $\Gamma(M,\sigma)$ l'ensemble des entrées et sorties exécutables par M après la trace σ et $In(M,\sigma)$ l'ensemble des entrées que peut traiter M après la trace σ.
- $Traces(q) =_\Delta \{\sigma \in (\Sigma^{\text{M}})^* \mid q$ ***after*** $\sigma \neq \emptyset\}$ est l'ensemble des traces exécutables à partir de q. $Traces(M) =_\Delta Traces(q_0^{\text{M}})$.

Blocage Nous définissons ici la notion de blocage. Cette notion est liée en pratique à l'absence d'observation pendant le test. Il existe trois types de blocages : le livelock, le deadlock et le blocage de sortie (voir figure 2.2).

Livelock Un état est dans un blocage de type livelock si le système peut exécuter une boucle d'évènements internes. Pour un état q d'un IOLTS M, un livelock est défini par $q \in livelock(M) =_\Delta q \xrightarrow{\tau \ldots \tau}_{\text{M}} q$.

Blocage de sortie Un état est en blocage de sortie (ou outputlock) s'il ne possède pas de transition tirable interne ou d'émission. Il ne peut donc évoluer que par une entrée, c'est-à-dire un message reçu sur une de ses interfaces. Pour un état q d'un IOLTS M, un blocage de sortie est défini par $q \in outputlock(M) =_\Delta \Gamma(q) \subseteq \Sigma_I^{\text{M}}$.

Deadlock Un état est bloqué par un blocage de type deadlock s'il ne possède aucune transition tirable. C'est un état de blocage permanent. Ce type de blocage est un cas particulier du blocage de sortie. Pour un état q d'un IOLTS M : $q \in deadlock(M) =_\Delta \Gamma(q) = \emptyset$.

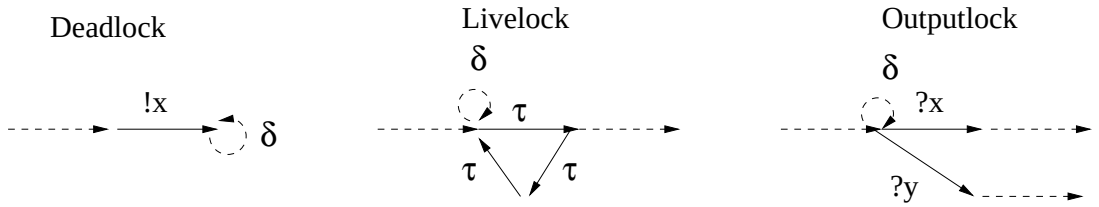


FIG. 2.2 – Types de blocages possibles

Un état de blocage est alors défini par $q \in quiescent(M) =_\Delta q \in deadlock(M) \vee q \in outputlock(M) \vee q \in livelock(M)$. Tous ces blocages se traduisent par un silence (quiescence) lors du test. Dans le contexte du test, il faut être en mesure d'observer ces blocages éventuels. Pour cela, on modélise le blocage dans un état $q \in quiescent(M)$ par $q \xrightarrow{\delta}_{\text{M}} q$, où δ est considéré comme un évènement de sortie observable.

Dans [Tre99] et dans [Jér01], l'IOLTS obtenu après modélisation de l'information sur les blocages est appelé IOLTS suspensif (suspensive IOLTS). Dans [Tre99], cet IOLTS ne considèrent que les blocages de type deadlock ou outputlock alors que l'IOLTS suspensif de [Jér01] considèrent les trois types de blocages décrits ci-dessus. Dans cette étude, nous utilisons également l'appellation IOLTS suspensif pour décrire l'IOLTS avec modélisation de l'information sur les blocages. Pour un IOLTS M donné, l'IOLTS suspensif obtenu après calcul des blocages est noté $\Delta(M)$.

Remarque: En pratique, on utilise un timer pour observer les silences lors de l'exécution des tests.

2.3.3 Interaction

Différentes opérations sur les IOLTS existent en fonction des besoins de modélisation. Parmi ces opérations, l'interaction entre deux IOLTS peut servir à décrire l'interaction entre un testeur et l'implémentation testée (dans le cas du test de conformité par exemple) ou le comportement global de deux entités en interaction dans le cadre du test d'interopérabilité. Dans cette section, nous donnons la définition de la composition synchrone [Tre96, JJ05, CK94]. Par la suite, dans le cadre de la formalisation du test d'interopérabilité, nous serons amenés à définir d'autres modèles comme celui de l'interaction en fonction du mode d'interaction (synchrone ou asynchrone) ou de l'architecture de test d'interopérabilité considérée.

Définition 2.2 (Composition synchrone $\|_S$) *La composition synchrone de deux IOLTS M_1 et M_2 est notée $M_1\|_S M_2 =_\Delta (Q^{M_1} \times Q^{M_2}, \Sigma^{M_1\|_S M_2}, \Delta^{M_1\|_S M_2}, (q_0^{M_1}, q_0^{M_2}))$ où $\Sigma^{M_1\|_S M_2} \subseteq \Sigma^{M_1} \cup \Sigma^{M_2}$, et la relation de transition $\Delta^{M_1\|_S M_2}$ est définie par les relations suivantes, $\forall (q_1, q_2) \in Q^{M_1} \times Q^{M_2}$:*

$$\frac{(q_1, \tau, q_1') \in \Delta^{M_1}}{((q_1, q_2), \tau, (q_1', q_2)) \in \Delta^{M_1\|_S M_2}}, \frac{(q_2, \tau, q_2') \in \Delta^{M_2}}{((q_1, q_2), \tau, (q_1, q_2')) \in \Delta^{M_1\|_S M_2}} \tag{2.1}$$

$$\frac{(q_1, \alpha, q_1') \in \Delta^{M_1}, (q_2, \bar{\alpha}, q_2') \in \Delta^{M_2}, \alpha \in \Sigma^{M_1}, \bar{\alpha} \in \Sigma^{M_2}, (\alpha, \bar{\alpha}) = (!a, ?a)}{((q_1, q_2), \alpha, (q_1', q_2')) \in \Delta^{M_1\|_S M_2}} \tag{2.2}$$

$$\frac{(q_1, \alpha, q_1') \in \Delta^{M_1}, \alpha \in \Sigma^{M_1}, \bar{\alpha} \notin \Sigma^{M_2}, (\alpha, \bar{\alpha}) = (!a, ?a)}{((q_1, q_2), \alpha, (q_1', q_2)) \in \Delta^{M_1\|_S M_2}} \tag{2.3}$$

$$\frac{(q_2, \alpha, q_2') \in \Delta^{M_2}, \alpha \in \Sigma^{M_2}, \bar{\alpha} \notin \Sigma^{M_1}, (\alpha, \bar{\alpha}) = (!a, ?a)}{((q_1, q_2), \alpha, (q_1, q_2')) \in \Delta^{M_1\|_S M_2}} \tag{2.4}$$

Cette opération de composition synchrone considère plusieurs types d'évènements. Tout d'abord, les évènements internes étant exécutés en interne d'un des deux systèmes en interaction, ils peuvent être propagés dans l'interaction : c'est la règle (2.1). Pour

les autres évènements (évènements α), tout dépend de l'existence ou non d'un évènement miroir $\bar{\alpha}$. Cet évènement miroir permet en effet la synchronisation par la mise en correspondance entre l'entrée et la sortie correspondante sur les deux systèmes en interaction : c'est la règle (2.2). Les deux dernières règles servent à propager les évènements pour lesquels il n'existe pas d'évènement miroir, c'est-à-dire les évènements qui ne sont pas utilisés pour la synchronisation entre les deux systèmes.

La définition de la composition présentée ici est celle qui avait été définie pour synchroniser les testeurs et l'implémentation dans le domaine du test de conformité. Cette définition sera ensuite adaptée (section 3.3.2) pour modéliser l'interaction entre deux implémentations de protocoles dans le cadre du test d'interopérabilité, puis étendue pour modéliser l'interaction de N implémentations pour un contexte testant l'interopérabilité N systèmes (section 5.4).

2.3.4 Projection

Lors de l'activité de test (de conformité ou d'interopérabilité), il peut être nécessaire d'observer certains évènements particuliers parmi toutes les traces possibles d'un système. Ces traces sont ainsi réduites aux messages que l'on souhaite plus particulièrement observer ou traiter (par exemple des messages exécutables sur une interface particulière). Elles peuvent être obtenues par une *projection* des traces moyennant la définition de critères pour sélectionner les évènements souhaités. Dans la définition qui suit, ces critères sont définis via un ensemble X d'évènements.

Définition 2.3 (Projection d'une trace sur un ensemble) *Soit M un IOLTS, $\sigma \in (\Sigma^M)^*$ une trace, $\alpha \in \Sigma^M$, et un ensemble X. La projection de σ sur X est notée σ/X et est définie par : $\epsilon/X = \epsilon, (\alpha.\sigma)/X = \sigma/X \; if \; \alpha \notin X$, et $(\alpha.\sigma)/X = \alpha.(\sigma/X) \; if \; \alpha \in X$.*

Dans le cadre de la définition formelle du test d'interopérabilité, il peut également être nécessaire de définir d'autres types de projection (par exemple la projection d'un IOLTS sur un ensemble). Ces définitions seront décrites dans la suite (par exemple section 3.3.3) en fonction des besoins de modélisation.

2.4 Test de conformité

2.4.1 Architecture de test de conformité

Architecture générale pour le test de conformité Le test de conformité met en relation un ou des testeurs avec une implémentation appelée IUT pour "Implementation Under Test". Le comportement de cette implémentation est observée via ses interfaces ou IAPs (Implementation Access Point). Le testeur interagit avec l'IUT à travers ses

interfaces (cf. figure 2.3) appelées selon les cas points de contrôle et d'observation (PCO, Point of Control and Observation) ou points d'observation (PO, Point of Observation). Ce testeur (TS, Test System) peut être composé de plusieurs composants de test (TC, Test Components) dédiés à certaines interfaces particulières et/ou certaines fonctions du TS. Il existe différents types de composants de test qui se distinguent selon la fonctionnalité de l'interface observée et/ou contrôlée.

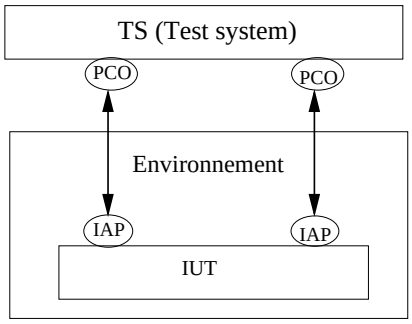


FIG. 2.3 – Architecture générale du test de conformité

Remarque: Les définitions ci-dessus sont également valables dans le contexte du test d'interopérabilité qui met également en relation des testeurs et des implémentations.

Différents contextes pour le test de conformité Les systèmes réels présentant une grande diversité de configurations, différentes méthodes de test ont été définies pour répondre aux différentes possibilités de contrôle et d'observation de l'implémentation à tester. Ces méthodes de test peuvent être décrites via l'architecture de test correspondante [ISO94, WSG98]. Ces architectures de test prennent en compte la fonctionnalité de l'interface observée/contrôlée : par exemple, pour un protocole d'une architecture protocolaire en couches, certains testeurs sont dédiés aux interfaces destinées à être connectées aux couches supérieures, d'autres aux interfaces vers les couches inférieures. Le mode de connexion des testeurs entre eux et à ces différents types d'interfaces décrit la méthode de test de conformité. Les différentes méthodes de test de conformité sont détaillée dans [WSG98]. Dans le domaine du test d'interopérabilité, différentes architectures de test peuvent également être distinguées (cf. section 2.5.3), et ces différentes architectures de test peuvent avoir une influence sur la notion d'interopérabilité considérée (cf. chapitre 3) et la méthode de génération de test (cf. chapitre 4).

2.4.2 Définitions formelles de la notion de conformité

Le test de conformité, qui vérifie qu'une implémentation prise isolément se comporte comme prévu dans sa spécification, a été précisément défini dans la norme ISO/IS 9646 [ISO94]. Cette norme traite entre autres des concepts relatifs au test de conformité, de la spécification des suites de tests abstraites, ou de la réalisation des tests.

Sur la base de cette norme, des études ont été menées pour donner des définitions formelles permettant d'automatiser la génération de tests de conformité. Différentes études, dont [Tre92, Pha94, VTKB92], ont également permis de modéliser et formaliser la notion de conformité d'une implémentation vis-à-vis de sa spécification. Différentes notions de conformité, appelées généralement *relations de conformité* ou *relations d'implantation*, ont été définies sur la base de modèles formels tels que celui des IOLTS. Une relation de conformité décrit formellement les conditions sous lesquelles une implémentation peut être considérée conforme à sa spécification. Nous présentons ici deux de ces définitions les plus utilisées en pratique : les relations de conformité **ioconf** [VTKB92] et **ioco** [Tre99].

La relation de conformité **ioconf** [VTKB92] est une relation entre l'IOLTS I modélisant une implémentation et l'IOLTS S modélisant la spécification sur laquelle est basée cette implémentation. Cette relation (cf. définition 2.4) dit que I est ioconf-conforme à S si, après toute trace de la spécification S observée sur l'implémentation I, les sorties de l'implémentation sont prévues dans la spécification.

Définition 2.4 (Relation de conformité ioconf)
I **ioconf** $S =_{\Delta} \forall \sigma \in Traces(S), Out(I, \sigma) \subseteq Out(S, \sigma)$

La relation de conformité **ioco** [Tre99] gère, en plus des conditions déjà décrites dans **ioconf**, les blocages possibles pour l'implémentation pendant le test. Cette relation (cf. définition 2.5) dit que I est ioco-conforme à S si, après toute trace suspensive (incluant de possibles blocages) de la spécification S observée sur l'implémentation I, les sorties et blocages de l'implémentation sont prévues dans la spécification.

Définition 2.5 (Relation de conformité ioco)
I **ioco** $S =_{\Delta} \forall \sigma \in Traces(\Delta(S)), Out(\Delta(I), \sigma) \subseteq Out(\Delta(S), \sigma)$

La relation de conformité **ioco** est la définition de la notion de conformité la plus utilisée actuellement en pratique. Il est possible de définir d'autres relations de conformité en fonction des besoins, par exemple selon l'architecture de test utilisée ou l'environnement dans lequel ont lieu les tests. De même, dans le chapitre suivant, nous serons amenés à donner plusieurs définitions formelles de l'interopérabilité en fonction du contexte considéré.

2.4.3 Outils et approches de génération de test

A partir des différentes définitions formelles de la notion de conformité, différentes méthodes et approches ont été décrites pour la génération automatique de tests de conformité. Selon les cas, ces méthodes (et outils correspondants) peuvent s'appuyer sur des modèles de types FSM (c'est la cas des méthodes décrites dans [LY96]) ou LTS ([Tre96, JCTG96, Mor00] par exemple). Elles peuvent aussi être conçues directement sur la base de langages utilisés dans le domaine de FDTs (Formal Description Techniques) tels que SDL (Specification and Description Language, [IT02]), Lotos (Language Of Temporal Ordering Specification), Estelle (Extended State Transition Language, [ISO89]), etc. Dans cette section, nous présentons quelques uns des outils existants [BFS05] pour la génération de tests de conformité et les techniques de génération de tests correspondantes.

TVEDA L'outil TVEDA [GR98] est un outil réalisé par le CNET (Centre National d'Etudes des Télécommunications) qui est construit sur les travaux de [Pha94]. Le principe est le suivant. Un automate étendu est produit à partir d'une spécification dans les langages Estelle [ISO89] ou SDL [IT02]. Des objectifs de tests, un par branche de transition de cet automate peuvent alors être générés automatiquement. Puis le test correspondant est généré par construction partielle du système de transition de la spécification. La génération de test peut se faire par les méthodes de squelettes de test transition par transition (SQ) et de tour de transition étendues (TT) basées sur le modèle des FSMs et décrites dans [Pha94, LY96, Jér01].

TTCgeN TTCgeN est un générateur de test produit par Verilog. TTCgeN était inclus dans l'environnement ObjectGeode de Verilog [Tel]. La spécification est décrite en SDL et l'objectif de test peut être un observateur écrit dans le langage GOAL (Geode Observation Automata Language), un diagramme MSC (Message Sequence Charts) ou un scénario construit de manière interactive au cours de la simulation du modèle SDL. La génération de test est réalisée via le calcul par un parcours en largeur du produit synchrone de la spécification en SDL et de l'objectif de test traduit si nécessaire en un observateur GOAL.

SAMSTAG SAMSTAG (SDL and MSC baSed Test cAse Generation) [GHN93] est une méthode et un outil permettant de générer des cas de test dans le langage TTCN à partir d'une spécification SDL et d'un objectif de test décrit avec des diagrammes MSC. La méthode consiste en plusieurs étapes parmi lesquelles : simulation de la spécification dans le but de trouver un préambule, une exécution de l'objectif de test et un postambule, réduction des actions internes non-observables, et prise en compte du non-déterminisme observable par l'ajout des messages ne satisfaisant pas l'objectif de test.

TorX L'outil TorX [TB03] a été réalisé dans le cadre du projet "Côte de Résyte" (pour "COnformance TEsting of REactive SYSTEms") [TB02, VTKB92, Tre99]. TorX est un outil combinant génération de test basée sur la relation de conformité **ioco** et exécution de test. Le test est construit pendant son exécution sur l'implémentation. Lorsque l'implémentation émet un message (ou dans le cas d'un blocage), celui-ci est comparé aux messages autorisés dans la spécification. Si le message n'est pas conforme, un verdict "fail" est émis. Sinon, le test continue jusqu'à génération d'un verdict "pass". Il n'y a pas dans TorX de notion d'objectif de test. Le parcours est aléatoire et tient compte des réactions de l'IUT.

TGV TGV (Test Generation with Verification technology) [BFS05, Mor00, JJ05, JCTG96, FJJV97] est un prototype de génération automatique de tests de conformité qui est intégré dans la boîte à outils CADP [GLM01] dont il utilise des bibliothèques. TGV a été développé en commun par l'équipe Pampa de l'IRISA à Rennes et le laboratoire Verimag Grenoble.

TGV permet de générer des cas de test à l'aide d'une technique d'exploration à la volée de l'espace d'états. Cette exploration est guidée par un objectif de test. La génération de test de conformité implémentée dans TGV est basée sur la relation de conformité **ioco** (définition 2.5 de la section 2.4.2). Les entrées de TGV sont une spécification (sous forme d'IOLTS dans les langages aldébaran [Fer89] ou bcg de CADP [GLM01] ou dans les langages SDL, Lotos) et un objectif de test (sous forme d'IOLTS dans les langages aldébaran ou bcg). Les cas de test générés sont des IOLTS équipés d'état-puits "pass", "fail" et "inconclusive" représentant les verdicts.

La génération de test implémentée dans TGV est basée sur les techniques issues de la vérification "à la volée". Dans TGV, cela consiste à construire le cas de test pendant la construction du modèle de la spécification visible depuis l'environnement. La construction est guidée par l'objectif de test et s'arrête lorsque le cas de test est complet : il n'est donc pas forcement nécessaire de calculer le modèle complet du comportement de la spécification visible depuis l'environnement.

2.5 Test d'interopérabilité

2.5.1 Problématique du test d'interopérabilité

Le test d'interopérabilité consiste à s'assurer que plusieurs implémentations différentes (de protocoles conçus pour fonctionner ensemble : cf. propriété d'iop-compatibilité des spécifications, section 3.4 du chapitre 3) interagissent correctement tout en rendant le service escompté. La différence principale entre tests de conformité et d'interopérabilité réside donc dans le fait que le test de conformité teste une implémentation par rapport à sa norme alors que le test d'interopérabilité teste des implémentations

entre elles.

La norme ISO/IEC IS9646 [ISO94] définissant un cadre pour le test de conformité reconnaît que le test de conformité augmente la probabilité d'interopérabilité, mais ne la garantit pas. En effet, le test de conformité est non exhaustif. Comme tout test, le test de conformité est utilisé pour détecter la non-conformité, et non pas pour prouver la conformité d'une implémentation. De plus, des options d'implémentation incompatibles ou des normes ambiguës peuvent aussi être à l'origine des problèmes d'interopérabilité non détectés lors de tests de conformité. Il faut donc, en complément des tests de conformité réalisés indépendamment sur chaque implémentation, tester l'interopérabilité d'implémentations appelées à être interconnectées.

Actuellement, la plupart des cas de test d'interopérabilité sont décrits "manuellement". Pour décrire manuellement les tests, des objectifs de test sont généralement utilisés. Ils définissent les propriétés que les tests doivent permettre de vérifier. Puis, à l'aide des spécifications des protocoles à tester, les testeurs décrivent les chemins possibles correspondant à ces objectifs de test. Des suites de test sont ainsi décrites grâce à un ensemble d'objectifs de test, et utilisées pour tester des implémentations par exemple lors de sessions d'interopérabilité appelées Plugtests ou Interoperability Events. Ces sessions d'interopérabilité sont des évènements de tests d'interopérabilité où des constructeurs font valider l'interopérabilité de leurs produits avec les autres produits du marché. Les laboratoires de test participent à ces sessions d'interopérabilité en proposant les suites de test permettant de valider les produits testés. Les Plugtests organisés par l'ETSI [ETS] pour le test de différents protocoles (par exemple IPv6, Mobile WiMAX, GRID Middleware and Infrastructures, Mobile applications, etc), les Plugtests auxquels participent l'équipe DIONYSOS de l'IRISA [IRI] sur les protocoles de la pile IPv6 ou le connectathon [Con] organisé par Sun microsystems (protocoles NFS, RPC, XDR, etc) sont des exemples de ces sessions d'interopérabilité.
L'activité de dérivation des suites de test d'interopérabilité (à exécuter lors de ces sessions d'interopérabilité) étant réalisée "manuellement" à l'aide de spécifications complexes et non nécessairement formelles, il y a un risque d'erreurs important lors de l'écriture des cas de test qui nécessitent donc d'être également vérifiés. De plus, la description manuelle des cas de test doit être réalisée pour chaque objectif de test : une suite de test demande donc beaucoup de temps pour être développée.

Dans cette thèse, nous nous sommes intéressés à l'application de techniques de description formelle (FDT) au domaine du test d'interopérabilité. En effet, l'utilisation de FDT nécessite un modèle des spécifications, mais une fois ce modèle validé, les suites de test peuvent être générées automatiquement à partir des objectifs de test.
Bien qu'il n'existe pas de cadre formel pour le test d'interopérabilité, certaines études ont été menées pour tenter de définir la notion d'interopérabilité ou décrire des algo-

rithmes permettant de générer automatiquement des suites de tests d'interopérabilité. Dans cette section, nous nous intéressons tout d'abord (section 2.5.2) aux définitions informelles de la notion d'interopérabilité et du test d'interopérabilité existant dans la littérature. Ensuite, nous décrivons (section 2.5.3) les différents contextes (notamment architecturaux) dans lesquels peuvent être exécutés des tests d'interopérabilité. Puis, nous décrivons les tentatives de définitions formelles existantes pour la notion d'interopérabilité (section 2.5.4) ainsi que les méthodes et approches de génération automatique de tests d'interopérabilité qui ont été développées (section 2.5.5).

2.5.2 Définitions existantes de l'interopérabilité

Les études les plus anciennes sur la notion d'interopérabilité et le test d'interopérabilité ont consisté à des tentatives pour essayer de définir plus clairement la notion d'interopérabilité. D'autres ont donné des indications sur ce que doit apporter le test d'interopérabilité. Quelques études ont également été faites concernant les liens entre interopérabilité et conformité.

Dans [Rep94], rapport de l'ETSI (European Telecommunications Standards Institute) sur les méthodes de test et de spécification, la propriété d'interopérabilité est découpée en plusieurs propriétés (toutes appelées interopérabilité) classifiées selon le niveau où est observée cette propriété, chaque propriété englobant l'interopérabilité du niveau au-dessous. Cette classification considère ainsi plusieurs niveaux d'interopérabilité, en partant du niveau protocolaire jusqu'à la perception de l'interopérabilité par l'utilisateur du système. Parmi les différents types d'interopérabilité, l'*interopérabilité de protocoles* est définie comme la capacité pour des implémentations d'échanger des PDUs (Protocol Data Units). Les concepts d'*interopérabilité des services* (capacité à rendre le service prévu lors de l'interaction) et *interopérabilité des applications* (au niveau des données) sont ensuite ajoutés pour permettre l'interopérabilité des systèmes considérés. Le test d'interopérabilité doit donc avoir pour objectif de vérifier ces trois propriétés.
Dans [BBB+96], différentes définitions de la notion d'interopérabilité sont également données en fonction du niveau considéré dans l'architecture : il y a une différence entre le point de vue du fournisseur du produit et du réseau, et celui de l'utilisateur. La définition générale de l'interopérabilité est : "la capacité de deux ou plusieurs applications à communiquer en utilisant un mécanisme spécifique, dans un environnement connu, de façon à atteindre les objectifs de l'utilisateur". Cette classification en fonction des couches protocolaires se retrouve également dans [GRSS90] où les définitions informelles de l'interopérabilité sont orientées soit couches protocolaires, soit systèmes.

D'autres définitions de la notion d'interopérabilité peuvent encore être trouvées dans la littérature : par exemple [CK94, VB94, KK96, DK03, MRW04]. Cependant, le

point commun entre toutes ces définitions de l'interopérabilité est de donner plusieurs objectifs au test d'interopérabilité. Ces objectifs sont :

- au niveau de l'interaction entre systèmes, la vérification de la capacité des systèmes à communiquer entre eux
- au niveau de l'interaction des systèmes avec les couches ou systèmes supérieurs, la capacité à rendre les services demandés par les utilisateurs (couches supérieures en général)

Parmi les autres études s'intéressant au rôle du test d'interopérabilité, certaines s'intéressent plus particulièrement aux liens et différences entre test d'interopérabilité et test de conformité. C'est le cas, par exemple, de [APRS93, KSM96, Kan98].
[APRS93] propose une combinaison des tests de conformité et d'interopérabilité basée sur la classification des cas de test de chaque type : les cas de test d'interopérabilité sont vus comme la composition de certains cas de test de conformité. Les cas de test de conformité sont ainsi classifiés en fonction de la possibilité de sélectionner des cas de test d'interopérabilité à partir de ces cas de test de conformité.
Dans [KSM96], le test d'interopérabilité est défini comme l'évaluation d'un produit pour déterminer s'il se comporte comme prévu lors de l'interaction avec un autre produit alors que le test de conformité est l'évaluation d'un produit pour déterminer s'il se comporte comme prévu lors de l'interaction avec un système de référence (le testeur). Les deux types de test sont comparés sur leur capacité à détecter les problèmes potentiellement rencontrés en situation réelle de fonctionnement. Les principales conclusions sont les suivantes. Tout d'abord, même s'il ne permet pas d'assurer l'interopérabilité, le test de conformité augmente le degré de confiance sur les capacités d'interopérabilité des systèmes. Ensuite, le test de conformité est décrit comme une méthode plus simple que le test d'interopérabilité pour localiser les éventuels problèmes. Cependant, le test d'interopérabilité permet de vérifier des propriétés sur les implémentations non vérifiables avec le test de conformité. La conclusion de cette étude est donc que ces deux types de tests sont complémentaires.
Dans [Kan98], différents types de tests de conformité et de tests d'interopérabilité sont classifiés et comparés. Le test de conformité est décomposé en *interface conformance testing* (chaque interface d'une implémentation est testée séparément) et *entity conformance testing* (chaque implémentation est testée globalement). Plusieurs architectures de test sont définies pour le test d'interopérabilité et comparées avec les contextes de test de conformité définis. Cette comparaison est utilisée pour proposer des méthodes de réutilisation et d'adaptation des suites de test de conformité en fonction de l'architecture de test d'interopérabilité.
Parmi les conclusions de ces études, nous pouvons donc retenir que :

- compte-tenu des points communs entre ces deux types de test, il est généralement possible de réutiliser/réadapter certains tests de conformité en interopérabilité
- compte-tenu des différences d'objectifs de ces deux types de test et de la non-

exhaustivité du test, un seul mode de test n'est pas suffisant pour permettre de garantir l'interopérabilité des implémentations, et des méthodes de test précises sont nécessaires dans les deux types de tests

2.5.3 Architectures de test d'interopérabilité one-to-one

Dans cette section, nous nous intéressons aux architectures de test d'interopérabilité. Ces architectures de test d'interopérabilité sont définies dans le cadre de l'interaction de deux implémentations, également appelée contexte one-to-one. Elles ont été classifiées par l'équipe antérieurement à cette thèse dans [VBT01].
Dans ce section, nous définissons les différentes architectures de test d'interoperabilité qu'il est possible de rencontrer en contexte one-to-one. Ensuite, dans le chapitre 3 (section 3.2), nous reprécisons certaines notions liées à cette classification de façon à identifier ce que cette classification implique dans la définition de la notion d'interopérabilité et dans la génération de test. En particulier, nous nous intéresserons dans ce chapitre aux possibilités d'observation et de contrôle des évènements que permettent une architecture de test d'interopérabilité.
A ce jour, nous n'avons pas trouvé d'études sur les architectures de test d'interopérabilité de N implémentations ($N > 2$). Les résultats de nos études pour ces contextes de test sont décrits dans le chapitre 5 (section 5.2).

La classification des différentes architectures de test d'interopérabilité définie dans [VBT01] et récapitulée dans cette section prend en compte des contextes définis dans [WSG98, UK99], mais également des définitions (en particulier pour les connexions entre interfaces des testeurs et des implémentations) de [ISO94] et des observations sur les configurations utilisées en pratique pour le test d'interopérabilité de deux implémentations.

Dans l'architecture générale de test d'interopérabilité (voir figure 2.4) définie pour l'interaction de deux implémentations, le système sous test (SUT pour System Under Test) est composé de deux implémentations sous test (IUT for Implementation Under Test, IUT_1 et IUT_2 sur la figure 2.4). Dans cette architecture, deux types d'interfaces peuvent être distinguées :

- les interfaces inférieures (LI_i pour Lower Interfaces) d'une implémentation sont les interfaces utilisées pour l'interaction avec l'autre implémentation
- les interfaces supérieures (UI_i pour Upper Interfaces) sont utilisées pour la communication avec l'environnement ou les couches supérieures, c'est-à-dire pour fournir les services prévus

La figure 2.4 représente l'architecture générale du test d'interopérabilité avec tous les éléments pouvant être présents dans une architecture de test d'interopérabilité. Le

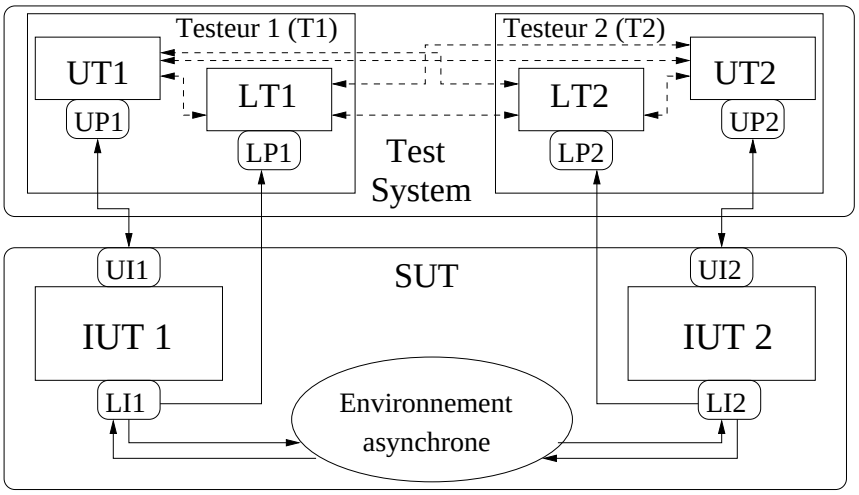


FIG. 2.4 – Architecture générale de test d'interopérabilité

système de test (TS : Test System) est l'élément chargé d'observer et/ou de contrôler le SUT. Il peut être composé de tout ou partie des éléments suivants.

- Le testeur supérieur UT_i (Upper Tester) est chargé du **contrôle et de l'observation** des évènements sur l'interface UI_i, via le PCO (Point of Control and Observation) supérieur UP_i.
- Le testeur inférieur LT_i (Lower Tester) est chargé de l'**observation** des évènements sur l'interface LI_i, via le PO (Point of Observation) inférieur LP_i.
- Le testeur T_i est la partie de TS chargée du contrôle et/ou de l'observation de l'implémentation IUT_i. Ce testeur est donc composé des testeurs UT_i et/ou LT_i.

Remarque: Notons ici la différence entre les interfaces LI_i et UI_i et donc également entre les testeurs UT_i et LT_i. Les interfaces inférieures ne sont pas contrôlables (un testeur ne peut pas envoyer de stimuli vers ces interfaces) mais seulement observables lors du test d'interopérabilité. En effet, l'envoi de stimuli vers ces interfaces biaiserait le test d'interopérabilité, de tels messages n'étant pas des messages de l'interaction. Le testeur LT_i est donc chargé de l'observation des évènements sur les interfaces LI_i. Par contre, comme les interfaces supérieures ne sont pas utilisées pour l'interaction, le testeur UT_i peut à la fois *observer et contrôler* les évènements exécutés sur l'interface à laquelle il est connecté.

Selon les situations, il est possible que toutes les interfaces des implémentations ne soient pas accessibles par le ou les testeurs. On peut ainsi distinguer différentes architectures de test d'interopérabilité.
En fonction du type d'interfaces auxquelles les testeurs ont accès, on peut distinguer les architectures suivantes.

Définition 2.6 (Architecture de test inférieure) *L'architecture de test est dite inférieure si les testeurs ont accès uniquement aux interfaces inférieures.*

Un exemple d'architecture de test inférieure est représentée dans la figure 2.5(c). Cette situation est possible dans le cas où les interfaces supérieures sont encapsulées dans le protocole de couche supérieure. Par exemple, dans le cas du test du protocole IP dans une pile protocolaire TCP/IP opérationnelle, les testeurs n'ont pas accès aux interfaces supérieures du protocole. Ces interfaces sont en effet encapsulées dans le protocole TCP implémenté au-dessus d'IP. Par contre, les interfaces inférieures d'IP peuvent être observées, par exemple grâce à l'utilisation d'outils tels qu'Ethereal[2] (analyseur réseau capable de capturer les paquets de données circulant sur un réseau).

Définition 2.7 (Architecture de test supérieure) *L'architecture de test est dite supérieure si les testeurs ont accès uniquement aux interfaces supérieures.*

Un exemple d'architecture de test supérieure est représentée dans la figure 2.5(b). Cette situation est généralement rencontrée lorsque le protocole de couche inférieure empêche la visualisation des messages qui lui sont transmis. Cette situation peut être rencontrée lors du test de protocoles applicatifs au niveau service, par exemple, dans le cas du test du protocole http auquel l'accès se fait via les primitives d'accès au service. Dans ce cas et si toute la pile des protocoles au-dessous d'http est implémentée et opérationnelle, seules les primitives du service sont accessibles (interfaces supérieure), mais pas la transmission des messages via les couches inférieures.

Définition 2.8 (Architecture de test totale) *L'architecture de test est dite totale dans le cas où les testeurs peuvent observer et/ou contrôler les deux types d'interfaces.*

Un exemple d'architecture de test totale est représentée dans la figure 2.5(a). Par rapport aux architectures de test supérieures et inférieures qui correspondent à des cas particuliers où les testeurs n'ont pas accès à toutes les interfaces, l'architecture de test totale correspond au cas général, à la situation la plus généralement rencontrée en pratique.
Différentes architectures de test d'interopérabilité peuvent également être distinguées en fonction du mode d'accès aux implémentations. La figure 2.6 représente ces différents modes d'accès aux implémentations en considérant une architecture de test totale.

Définition 2.9 (Architecture de test unilatérale) *L'architecture de test d'interopérabilité est dite unilatérale si les testeurs n'ont accès qu'à une seule implémentation durant son interaction avec la deuxième implémentation.*

[2] `http://www.ethereal.com/`

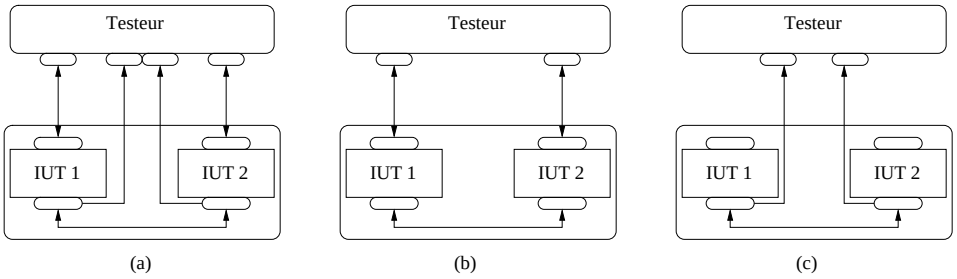


FIG. 2.5 – Exemples *(mode de test global)* d'architectures de test totale (a), supérieure (b) et inférieure (c)

L'architecture de test unilatérale (totale) est représentée dans la figure 2.6(b). Cette situation est possible par exemple dans des cas où l'une des deux implémentations est encapsulée dans un autre système : les testeurs n'ont alors pas accès aux interfaces de cette implémentation bien qu'elle interagisse avec l'autre implémentation. Il est également possible de ne pas avoir accès à l'une des deux implémentations pour des raisons de confidentialité : un constructeur peut autoriser le test d'interopérabilité avec son implémentation sans autoriser l'accès aux interfaces de son implémentation.

Définition 2.10 (Architecture de test bilatérale) *L'architecture de test est dite bilatérale si les testeurs ont accès aux deux implémentations, mais séparément.*

L'architecture de test bilatérale (totale) est représentée dans la figure 2.6(c). Dans cette architecture, il n'y a pas de connexion entre les testeurs T_1 et T_2, donc pas de synchronisation entre ces testeurs.

Définition 2.11 (Architecture de test globale) *L'architecture de test est dite globale si les testeurs ont accès aux deux implémentations avec une vue globale lors des tests.*

L'architecture de test globale (totale) est représentée dans la figure 2.6(a). C'est également ce point de vue qui avait été considéré dans la figure 2.5 lors de la définition des possibilités d'accès aux différents types d'interfaces. Cette architecture est actuellement la plus utilisée en pratique. Ces deux dernières architectures (globale et bilatérale) correspondent à une même observabilité des interfaces des implémentations : la différence se situe au niveau des testeurs qui peuvent avoir soit une vue globale du SUT (architecture globale) qui impose une synchronisation entre les testeurs, soit une vue séparée de chacune des IUTs (architecture bilatérale) ce qui évite une synchronisation entre testeurs souvent complexe.

Remarque: Les architectures de test unilatérale, bilatérale et globale peuvent être combinées avec les architectures de test totale, inférieure et supérieure. On obtient

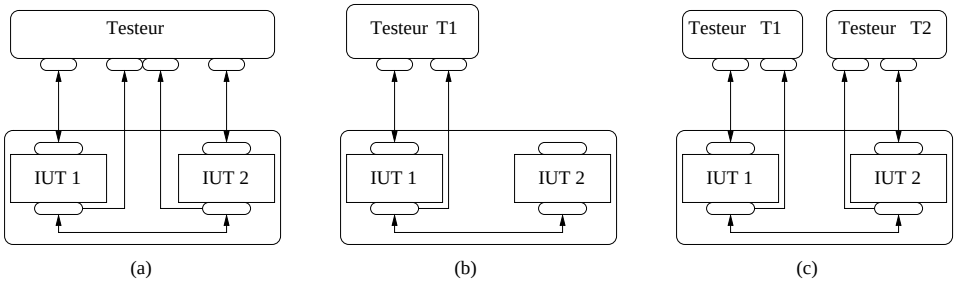


FIG. 2.6 – Architectures de test globale totale (a), unilatérale totale (b) et bilatérale totale (c)

ainsi neuf architectures de test d'interopérabilité : globale totale, unilatérale totale, bilatérale totale, globale supérieure, unilatérale supérieure, bilatérale supérieure, globale inférieure, unilatérale inférieure, bilatérale inférieure. La figure 2.5 représente ainsi les architectures de test globale totale (figure 2.5(a), comme la figure 2.6(a)), globale supérieure et globale inférieure.

2.5.4 Approches formelles du test d'interopérabilité

Quelques études ont été menées pour donner des définitions formelles à la notion d'interopérabilité. C'est le cas de [CK94] ou [VBT01] dont les définitions utilisent le modèle des IOLTS, ou de [WP94] basé sur un modèle algébrique pour les spécifications. Ces tentatives de définition pour la notion d'interopérabilité sont décrites dans cette section.

Dans [WP94], un modèle algébrique combiné à la logique temporelle est utilisé. L'interopérabilité de deux systèmes est définie à l'aide de plusieurs propriétés. Les deux implémentations doivent vérifier des propriétés de sûreté (qui spécifie le comportement permis lors de séquences de communications), de vivacité (qui exige l'exécution d'actions particulières par les implémentations) et être des paires communicants. Cette dernière propriété signifie que si une des implémentations réalise un service en tant qu'émetteur, l'autre implémentation doit être capable de réaliser ce service en tant que récepteur. Cette définition considère donc à la fois des propriétés sur le service rendu par les implémentations et sur leur capacité à communiquer.

Dans [CK94], l'interopérabilité est définie comme la capacité de deux (ou plus) systèmes à fournir un service donné tout en étant conforme à un protocole donné. La définitions formelle de la notion d'interopérabilité est donnée à travers une relation $R(I, S)$. La relation $R(I, S)$ est définie par $R(I, S) = \forall \sigma \in Traces(S)$, $(\sigma \in Traces(I) \Rightarrow Out(I, \sigma) \subseteq Out(S, \sigma))$. L'interopérabilité est ensuite définie via trois conditions : $R(I_1 \| I_2, S_1 \| S_2) \wedge R(I_1, S_1) \wedge R(I_2, S_2)$. Cette définition for-

melle est un premier pas dans la formalisation de la notion d'interopérabilité, mais elle ne prend pas en compte les différents contextes possibles pour le test d'interopérabilité. De plus, la dérivation de tests d'interopérabilité basée sur cette définition est limitée par l'explosion du nombre d'états de l'interaction des spécifications $S_1 \| S_2$.

Dans [VBT01], plusieurs définitions formelles de la notion d'interopérabilité sont données en fonction des contextes possibles pour le test d'interopérabilité. Ces définitions, appelées "relations d'interopérabilité" sont basées sur la relation de conformité **ioconf**. Cependant, ces relations ne prennent en compte ni les blocages possibles lors des tests, ni la vérification de la communication entre les deux implémentations. De plus, seules quelques pistes sont données pour la génération de test basée sur ces relations d'interopérabilité.

2.5.5 Méthodes et approches de génération automatique de tests d'interopérabilité

Plusieurs études portent sur la définition de méthodes ou algorithmes pour générer automatiquement des suites de tests d'interopérabilité. La plupart de ces méthodes sont basées soit sur un modèle de type FSM, soit sur un modèle de type LTS ou IOLTS. Par contre, il n'y a pas forcement de définition formelle de l'interopérabilité à la base de ces méthodes : ces méthodes sont en général basées soit sur des modèles de fautes, soit sur la recherche de certains types d'erreurs typiques de l'interopérabilité, ou soit sur des parcours aléatoires de l'interaction des spécifications.

La méthode de génération de test décrite dans [EFTSY04] fait partie des méthodes basées sur la recherche d'un type d'erreurs particulier. En effet, l'objectif de cette méthode est la recherche des livelocks exécutables par le système global, composé par l'interaction des implémentations. Les interfaces utilisées pour l'interaction ne sont pas observées (architecture globale supérieure). La méthode vérifie également que le système composé des deux implémentations est conforme à la spécification décrivant le comportement global extérieur de ce système (appelé reachability graph).

Une autre étude basée sur un modèle de fautes est l'étude de [SKKR03]. Cette étude se focalise sur les fautes lors de la communication entre implémentations. La génération des cas de test est basée sur [TKS03, SKKC04]. Un algorithme sert à calculer la couverture de la suite de tests obtenue, puis la complète. Des hypothèses fortes sont faites dans cette étude. Par exemple, les implémentations sont considérées conformes à leur spécification et si une erreur est observée lors des tests d'interopérabilité, une seule des implémentations peut être la source de cette erreur. La génération de tests d'interopérabilité de [SKKR03, TKS03, SKKC04] est basée sur un ensemble d'algorithmes permettant successivement de générer une suite de tests, puis de calculer la

couverture des tests, et de compléter cette suite de test. Cette génération de tests d'interopérabilité est basée sur une analyse du graphe d'accessibilité de l'interaction des spécifications privilégiant les échanges entre les deux implémentations et les liens causaux entre évènements exécutés par les deux implémentations en interaction.

Il existe également des études qui portent sur le test d'interopérabilité de certains protocoles particuliers ou de propriétés particulières des protocoles testées : par exemple, [KOS+00] se focalise sur le cas particulier des protocoles de la pile TCP/IP, tandis que [WWY04] s'intéresse aux contraintes de temps dans les systèmes temps-réels.

L'étude décrite dans [KOS+00] a ainsi pour objectif de générer des tests d'interopérabilité pour un ensemble de protocoles particuliers, les protocoles de la pile TCP/IP. Le problème étudié par les auteurs est le test de protocoles de la pile TCP/IP dans un contexte opérationnel, avec les problèmes de détection des erreurs que pose un tel contexte (problèmes d'accès aux implémentations de chaque protocole de la pile, recherche du protocole/système comportant l'erreur trouvée, etc). La méthode décrite est conçue en fonction des particularités des protocoles de la pile TCP/IP et de leur mode d'interaction.

L'étude présentée dans [WWY04] se focalise sur la gestion des contraintes de temps dans les systèmes temps-réels lors du tests d'interopérabilité de ces systèmes. Le modèle utilisé est le modèle des CTIOAs (Communicating Timed Input Output Automaton), une version étendue des FSMs qui décrit les contraintes de temps par des expressions linéaire utilisant des valeurs d'horloges locales. Les cas de test d'interopérabilité sont d'abord générés à partir d'une analyse du graphe d'accessibilité de l'interaction des spécifications. Puis une analyse de l'exécutabilité des séquences de test générées est faite. Les contraintes linéaires (contraintes de temps) des horloges locales sont converties en contraintes pour des horloges globales. Les valeurs initiales des horloges sont également sélectionnées de façon à obtenir des cas de test exécutables. Ainsi, les cas de test d'interopérabilité sont générés à partir de l'interaction des spécifications, et l'étude de [WWY04] se focalise sur l'ajout des contraintes de temps sur ces cas de test.

La méthode décrite dans [HLSG04] est basée sur une architecture de test d'interopérabilité avec accès à un seul des composants en interaction (architecture de type unilatérale). L'objectif de cette méthode est une couverture complète des interactions entre systèmes sans redondance. Bien qu'un seul des deux systèmes soit accessibles, une analyse de l'interaction entre spécifications (reachability analysis) est utilisée comme base pour la génération de test, pour connaître les possibilités d'interactions entre systèmes et donc couvrir ces interactions.

Les méthodes de génération de tests d'interopérabilité décrites dans [BCKZ02]

réutilisent les principes de méthodes développées pour un contexte de test de type test embarqué. Ces méthodes de génération de tests d'interopérabilité sont en effet dérivées de la méthode "hit-or-jump" de [CLRZ99]. L'architecture de test est une architecture globale et l'objectif est la couverture des transitions où les deux entités échangent des messages. Le principe de l'algorithme "hit-or-jump" est utilisé pour éviter de construire le modèle complet de l'interaction des spécifications.

Il est également possible de réutiliser les techniques de génération "à la volée" (ou en anglais "on-the-fly" [JCTG96]), définies pour la génération automatique de tests de conformité, pour définir une approche pour la génération automatique de tests d'interopérabilité. C'est le cas dans [KC00] qui prend en entrée un objectif de test et deux spécifications. Le cas de test correspondant est généré à la volée avec un algorithme de recherche en profondeur, considérant une interaction entre les systèmes réduite à une seule interface de communication. Cette méthode s'appuie sur la définition de [CK94] qui compare une vue globale de l'interaction des implémentations à un modèle de l'interaction des spécifications.

2.5.6 Conclusion

Nous avons vu dans cette section qu'il existe quelques tentatives pour définir précisément la notion d'interopérabilité aussi bien informellement (en langage naturel) que formellement. Cependant, ces tentatives n'ont pas donné lieu à une définition communément admise pour la notion d'interopérabilité telle qu'il en existe en conformité avec la notion de relation de conformité. Ainsi, les différentes méthodes de génération de tests d'interopérabilité présentées dans la section 2.5.5 sont basées sur différentes définitions de la notion d'interopérabilité. Cependant, on peut déduire de toutes ces tentatives de définition de la notion d'interopérabilité et des méthodes de génération de tests ce que la définition de la notion d'interopérabilité et les méthodes associées doivent prendre en compte. Il faut ainsi vérifier que les implémentations communiquent bien (couverture des interactions), et qu'elles rendent les services prévus lors de cette interaction (comportement correct lors de l'interaction). De plus, la plupart de ces méthodes de génération de tests d'interopérabilité est basée sur une vue globale du système composé des deux implémentations en interaction.

Dans la suite, nous nous sommes donc intéressés à la définition formelle de la notion d'interopérabilité dans le contexte de l'interaction de deux implémentations. Nous avons ensuite étudié comment dériver, à partir de ces définitions formelles, des algorithmes permettant de générer automatiquement des tests d'interopérabilité. Ces tests doivent à la fois permettre de vérifier les différentes propriétés des définitions formelles et prendre en compte les contextes de test d'interopérabilité existants. Pour le cas plus général de l'interaction de N implémentations, une première étude nous a

permis de proposer une classification des architectures de test et des définitions formelles de la notion d'interopérabilité dans ce contexte. Toutes ces contributions sont développées dans les chapitres suivants. La définition formelle de la notion d'interopérabilité de deux implémentations est dans le chapitre 3, la génération de test basée sur ces définitions dans le chapitre 4, et notre étude du contexte de l'interaction de N implémentations dans le chapitre 5.

Chapitre 3

Définition formelle de la notion d'interopérabilité : cas de deux implémentations

3.1 Introduction

Le test d'interopérabilité met en relation des implémentations pour vérifier qu'elles communiquent correctement tout en fournissant les services décrits dans leurs spécifications. Le contexte d'interopérabilité one-to-one (ou "un-contre-un") se concentre plus particulièrement sur l'interopérabilité entre deux implémentations. C'est le contexte le plus couramment rencontré en pratique soit par la mise en relation d'exactement deux implémentations, soit par le test d'interopérabilité d'une implémentation avec un système déjà fonctionnel composé de N implémentations. C'est donc également le contexte sur lequel porte la plupart des études visant à la formalisation du test d'interopérabilité [CK94, KC00, VBT01, SKKR03, HLSG04]. L'objectif de ce chapitre est de donner des définitions formelles de la notion d'interopérabilité dans ce contexte mettant en relation deux implémentations.

Contrairement au test de conformité [ISO94], il n'existe actuellement pas de norme définissant un cadre méthodologique pour le test d'interopérabilité. Plusieurs étapes sont nécessaires pour pouvoir définir formellement la notion d'interopérabilité et pouvoir choisir sur quelle définition doivent se baser les tests d'interopérabilité.

1. Il faut définir les différents contextes dans lesquels des tests d'interopérabilité one-to-one peuvent être exécutés. Notons qu'il existe plusieurs contextes pour le test de conformité (cf. section 2.4.1) bien que ce test ne concerne qu'une seule implémentation. Or, le test d'interopérabilité est un type de test qui met en relation plusieurs implémentations dans un contexte opérationnel. Ainsi, il existe

plusieurs méthodes et architectures de test associées pour le test d'interopérabilité. Une classification des architectures d'interopérabilité de deux implémentations a été faite dans [VBT01] et est récapitulée dans la section 2.5.3.

2. Plusieurs définitions formelles de la notion d'interopérabilité peuvent être données en fonction du contexte de test ou des besoins de détection de la non-interopérabilité. Ces définitions, que nous avons appelées ***critères d'interopérabilité***, décrivent formellement les conditions que doivent vérifier deux implémentations pour être considérées interopérables. En fonction des besoins de détection de la non-interopérabilité et du contexte de test (mode d'accès aux interfaces des implémentations par exemple), ces critères d'interopérabilité peuvent prendre en compte des conditions définissant l'interopérabilité de façon plus ou moins précise.

3. En fonction du contexte de test d'interopérabilité et des besoins de détection de la non-interopérabilité, un critère d'interopérabilité peut être choisi comme base pour les tests d'interopérabilité. Ceci nécessite de connaître le pouvoir de détection de la non-interopérabilité de chacun des critères définis.

Dans ce chapitre, nous nous focalisons sur la définition formelle de la notion d'interopérabilité dans le contexte one-to-one. L'organisation de ce chapitre est la suivante. La section suivante 3.2 porte sur les architectures de test d'interopérabilité possibles pour le contexte one-to-one. La classification des contextes possibles ayant été décrite dans la section 2.5.3, nous nous intéressons plus précisément à la prise en compte des différents éléments d'une architecture de test et des différents contextes pour des définitions formelles de l'interopérabilité. Ensuite, la section 3.3 porte sur la modélisation de propriétés et opérations nécessaires pour pouvoir formaliser la notion d'interopérabilité. Tester l'interopérabilité de deux implémentations nécessite que les spécifications sur lesquelles ces implémentations sont basées permettent l'interopérabilité. La section 3.4 décrit donc les conditions que doivent respecter les spécifications considérées pour permettre l'interopérabilité des systèmes les implémentant. Les sections 3.5 et 3.6 portent ensuite sur la formalisation de la notion d'interopérabilité. Tout d'abord, nous décrivons plus en détails ce qu'impliquent les différentes propriétés à vérifier. Puis nous donnons différentes définitions de la notion d'interopérabilité appelées critères d'interopérabilité. Dans la section 3.7, nous illustrons ces critères d'interopérabilité par une application à des exemples de spécifications et d'implémentations. Enfin, nous comparons ces critères d'interopérabilité entre eux (section 3.8) et nous donnons des éléments pour le choix d'un critère d'interopérabilité en fonction de l'architecture du test d'interopérabilité dans la section 3.9.

3.2 Architectures de test d'interopérabilité

L'interopérabilité de deux implémentations peut être testée dans plusieurs configurations différentes. Ces configurations correspondent aux différentes possibilités de contrôle et d'observation du système à tester. Dans le contexte du test d'interopérabilité one-to-one, le système à tester (SUT) est composé de deux implémentations interconnectées. Une classification de ces différentes architectures de test d'interopérabilité a été réalisée dans [VBT01] et présentées dans la section 2.5.3 du chapitre 2. Dans cette section, nous reprécisons les éléments d'une architecture de test d'interopérabilité qu'il est nécessaire de prendre en compte dans une définition formelle de la notion d'interopérabilité. Nous nous intéressons également au rapport entre les différents modes d'observation et de contrôle des interfaces des implémentations et la notion d'interopérabilité pouvant être testée dans un contexte donné.

Tout d'abord, considérons l'architecture de test générale de la figure 2.4 section 2.5.3. Dans cette architecture, deux types d'interfaces ont été différenciées en fonction de leur rôle. Dans cette section, nous nous intéressons plus particulièrement aux possibilité de contrôle et d'observation de ces différentes interfaces.

- les *interfaces supérieures* UI_i sont utilisées pour la communication avec l'environnement. Ces interfaces sont à la fois **observables et contrôlables** par les testeurs qui y sont connectés. En effet, ces testeurs peuvent observer les évènements exécutés sur ces interfaces, mais il peuvent également envoyer des stimuli (correspondant aux messages que l'environnement peut normalement transmettre aux IUTs) vers ces interfaces. Cette possibilité pour le testeur d'envoyer des stimuli aux interfaces correspond ainsi à la propriété de contrôlabilité des interfaces supérieures.
- les *interfaces inférieures* LI_i d'une implémentation sont les interfaces utilisées pour l'interaction avec l'autre implémentation. Les évènements exécutés par les implémentations sur ces interfaces correspondent à la transmission de messages (envoi et réception) servant à la communication entre les implémentations. Les testeurs ne peuvent donc, pendant le test d'interopérabilité, qu'observer ces interfaces. En effet, l'envoi de message par le testeur sur ces interfaces biaiserait le test d'interopérabilité : ce message ne correspond pas à des interactions entre les deux IUTs. Les méthodes de génération de test devront ainsi prendre en compte la **non-contrôlabilité** de ces interfaces.

Mode d'observation des évènements : Les implémentations peuvent exécuter deux types d'évènements, entrées et sorties, sur leurs différentes interfaces. Selon le type d'évènements et le type d'interface, les possibilités d'observation/contrôle de ces évènements diffèrent.

- Cas des évènements de sortie. Les sorties sur les interfaces supérieures corres-

pondent à des réceptions du testeur qui est directement connecté à de telles interfaces. Sur une interface inférieure, une sortie peut être observée par le testeur, mais il n'est pas le récepteur du message correspondant.

– Cas des évènements d'entrée. Le testeur sait quels messages ont été placés en entrée des interfaces des implémentations. Il sait donc quels messages doivent avoir été reçus par l'IUT et sur quelle interface. Par contre, ce que ne sait pas le testeur dans un contexte où l'IUT est une boîte noire, c'est la prise en compte effective du message reçu par l'implémentation. Ainsi, un testeur ne peut pas savoir si une IUT a été capable de traiter les messages qui lui ont été transmis. Par abus de langage, c'est cette opération de traitement du message par l'IUT que nous appelerons réception ou évènement d'entrée. Et ainsi, nous serons amené à considérer qu'*une entrée n'est pas observable par le testeur*. Cependant, bien qu'une entrée ne soit pas observable, des éléments sur son exécution sont connus (en particulier l'évènement d'envoi du message vers l'implémentation).

Nous avons vu dans la section 2.5.3 qu'il est possible de distinguer les architectures de test d'interopérabilité en fonction du type d'interfaces auquel les testeurs ont accès. On obtient ainsi les architectures de test d'interopérabilité inférieure, supérieure et totale. Les architectures de test d'interopérabilité inférieure (resp. supérieure) ne permettant pas de connaître les évènements exécutés sur les interfaces supérieures (resp. inférieures), la décision sur l'interopérabilité doit être prise en fonction d'observations partielles : cette restriction sur les observations lors du test d'interopérabilité doit apparaître dans les définitions formelles s'appuyant sur un tel contexte.

Les architectures de test d'interopérabilité ont aussi été classifiées en fonction du mode d'accès aux implémentations, c'est-à-dire du point de vue du ou des testeurs. Les architectures de test d'interopérabilité unilatérale, bilatérale et globale ont ainsi été définies.
L'architecture de test d'interopérabilité unilatérale ne permet, comme les architectures de test d'interopérabilité inférieure et supérieure, que des verdicts basés sur des observations partielles des évènements exécutés : une seule implémentation est observée/contrôlée lors de l'interaction des deux implémentations. Dans une telle architecture, les testeurs n'ont pas accès aux interfaces de l'autre IUT. Ces interfaces peuvent être inaccessibles pour plusieurs raisons, en général soit pour des problèmes de confidentialité, soit parce que l'implémentation est encapsulée dans une pile protocolaire complète. Dans les deux cas, les interfaces de l'implémentation sont utilisées mais par un autre système que les testeurs du test d'interopérabilité. Il n'est donc pas possible de faire des hypothèses restraignant les comportements possibles de l'IUT non accessible lors du test d'interopérabilité.

Les architectures de test d'interopérabilité bilatérale et globale ont accès aux mêmes interfaces. La différence entre ces deux architectures réside dans le mode d'observation

de ces interfaces. En effet, l'architecture de test d'interopérabilité bilatérale illustre une situation de test distribué. Un testeur différent, T_1 et T_2 respectivement, est connecté à chaque implémentation, I_1 et I_2 respectivement, mais sans synchronisation entre les testeurs. Cela veut dire que les causalités entre évènements exécutés sur chacune des implémentations ne sont pas prises en compte dans les cas de test correspondants. Elle ne sont donc pas non plus prises en compte dans la définition formelle de la notion d'interopérabilité correspondant à cette architecture. L'architecture de test d'interopérabilité globale correspond également à une situation de test distribué, mais avec coordination (via une procédure de synchronisation appelée TCP pour Test Coordination Procedure) entre les testeurs. Les cas de test et les définitions formelles correspondantes doivent donc refléter ce point de vue global du système composé des deux implémentations en interaction.

Ces différentes architectures peuvent être combinées (par exemple, architecture inférieure globale ou architecture bilatérale totale, etc). Nous obtenons ainsi neuf contextes principaux pour le test d'interopérabilité. Les critères d'interopérabilité définis dans la suite (section 3.6) doivent prendre en compte les différentes possibilités d'observation des interfaces des implémentations, et les modes d'observation correspondants pour décrire les conditions sous lesquelles les implémentations sont considérées interopérables.

3.3 Quelques notions formelles pour le test d'interopérabilité

Les définitions présentées dans la suite de ce chapitre utilisent le modèle des IOLTS présenté dans la section 2.3.2 et les notations associées. Dans le cadre de la formalisation de la notion d'interopérabilité, nous introduisons de nouvelles notations. Certaines définitions d'opérations sur les IOLTS doivent également être complétées pour permettre de prendre en compte les particularités du contexte du test d'interopérabilité.

3.3.1 Quelques définitions et notations

Les principales notations associées aux IOLTS ont été définies dans la section 2.3.2. Nous ajoutons à celles-ci les notations de cette section qui prennent en compte l'architecture de test d'interopérabilité générale et les types d'interfaces correspondants.

Les interfaces inférieures et supérieures n'ayant pas le même rôle dans l'architecture de test, il est important de distinguer précisément les évènements exécutables sur chaque type d'interfaces. L'ensemble Σ^M des évènements exécutables aux interfaces

de l'IOLTS M peut ainsi être décomposé de façon à distinguer les évènements exécutables sur les interfaces inférieures et supérieures : $\Sigma^M = \Sigma^M_U \cup \Sigma^M_L$ où Σ^M_U (resp. Σ^M_L) est l'ensemble des évènements exécutables sur les interfaces supérieures (resp. inférieures).

Dans le contexte du test d'interopérabilité, nous avons besoin de connaître la correspondance entre l'évènement représentant une sortie sur une interface inférieure et l'évènement représentant la réception de ce message par l'entité homologue. Cette correspondance est décrite dans la définition 3.1 du miroir d'un évènement de l'interaction.

Définition 3.1 (Miroir d'un évènement de l'interaction : iop-miroir)
Soit L_{ij} un lien reliant l'interface (de type inférieure) l_i d'un IOLTS M_i à l'interface l_j d'un IOLTS M_j. L'évènement miroir (ou iop-miroir) d'un évènement μ tel que $\mu \in \Sigma^{M_i}_L$ est défini par : $\bar{\mu} = l_j!a$ si $\mu = l_i?a$ et $\bar{\mu} = l_j?a$ si $\mu = l_i!a$.

La notion de blocage ou quiescence a été définie pour modéliser le silence d'une implémentation. Cette notion est représentée par l'évènement δ qui est inclus dans l'ensemble des évènements de sortie et observé en pratique grâce à l'usage de timers. Dans le contexte du test d'interopérabilité, il est nécessaire d'identifier le ou les systèmes bloqués ou pour lesquels les blocages sont autorisés. En effet, cela peut permettre par exemple de savoir de quelle implémentation provient l'erreur qui a conduit à un silence lors d'un test d'interopérabilité. Nous utilisons la notation $\delta(i)$ pour modéliser un blocage dans l'IOLTS M_i. Le blocage du système composé de deux IOLTS M_1 et M_2 est alors noté $\delta = \delta(1) \wedge \delta(2)$ indiquant que M_1 et M_2 sont tous deux dans un état de blocage.

3.3.2 Interaction asynchrone

Dans la section 2.3.3, nous avons donné la définition de l'interaction synchrone de deux systèmes, utilisée par exemple pour décrire la synchronisation entre un testeur et l'implémentation testée dans le cas du test de conformité. Dans le contexte du test d'interopérabilité, il est nécessaire de modéliser l'interaction entre deux systèmes interconnectés tels que les deux implémentations ou les deux spécifications correspondantes.

Dans le cas général, l'interaction entre deux implémentations de protocoles est une interaction asynchrone. En effet, les deux implémentations interagissent généralement à travers les couches inférieures que ce soit dans un réseau réel ou lors du test d'interopérabilité. Le caractère asynchrone de l'interaction peut être modélisé, comme dans [VTKB92], par une file FIFO fiable dans chaque sens de communication de chaque

lien reliant les deux IOLTS. Dans cette section, nous définissons le modèle de l'interaction asynchrone qui est considéré dans la suite pour décrire l'interaction entre deux implémentations ou entre les deux spécifications correspondantes.

Pour définir le modèle de l'interaction asynchrone (c'est-à-dire le graphe d'accessibilité de deux systèmes M_1 et M_2 interagissant à travers un environnement asynchrone), nous nous sommes intéressés aux opérations de modélisation d'un environnement asynchrone [VTKB92, JJTV99] et d'interaction synchrone (cf. définition 2.2, section 2.3.3). La définition de l'interaction asynchrone doit être basée sur différentes règles provenant de ces opérations.

Tout d'abord, intéressons-nous à la représentation d'un état de l'interaction. Pour identifier un état de l'interaction asynchrone de deux systèmes M_1 et M_2, plusieurs informations doivent être connues :

- État courant de M_1
- État courant de M_2
- État de le ou des files d'attente de M_2 vers M_1 (c'est-à-dire en entrée de M_1)
- État de le ou des files d'attente de M_1 vers M_2 (c'est-à-dire en entrée de M_2)

Toutes ces informations sont de plus nécessaires pour identifier quels évènements sont exécutables dans un état donné de l'interaction. En particulier, il est nécessaire de connaître, en plus de l'état courant du/des systèmes concernés, l'état des files d'attente pour savoir si une réception sur une interface inférieure peut avoir lieu.

Définition 3.2 (Interaction asynchrone d'interopérabilité one-to-one)
L'interaction asynchrone de deux IOLTS $M_1 = (Q^{M_1}, \Sigma^{M_1}, \Delta^{M_1}, q_0^{M_1})$ *et* $M_2 = (Q^{M_2}, \Sigma^{M_2}, \Delta^{M_2}, q_0^{M_2})$ *(reliés via leurs interfaces inférieures) est notée* $M_1 \|_{\mathcal{A}} M_2 = (Q^{M_1\|_{\mathcal{A}}M_2}, \Sigma^{M_1\|_{\mathcal{A}}M_2}, \Delta^{M_1\|_{\mathcal{A}}M_2}, (q_0^{M_1\|_{\mathcal{A}}M_2}))$ *avec :*

- *un état* $(q_1, q_2, < f1 >, < f2 >) \in Q^{M_1\|_{\mathcal{A}}M_2}$ *est tel que* $q_1 \in Q^{M_1}$, $q_2 \in Q^{M_2}$, $< f1 >$ *représente l'état des files d'attente en entrée de* M_1 *(c'est-à-dire les files de* M_2 *vers* M_1*) et* $< f2 >$ *l'état des files d'attente en entrée de* M_2 *(c'est-à-dire les files de* M_1 *vers* M_2*).*
- $q_0^{M_1\|_{\mathcal{A}}M_2}$ *est l'état initial de l'interaction et est tel que (*ϵ *décrivant une file FIFO vide) :* $q_0^{M_1\|_{\mathcal{A}}M_2} = (q_0^{M_1}, q_0^{M_2}, < \epsilon \cdots \epsilon >, < \epsilon \cdots \epsilon >)$*.*
- $\Sigma^{M_1\|_{\mathcal{A}}M_2} \subseteq \Sigma^{M_1} \cup \Sigma^{M_2}$.
- *la relation de transition* $\Delta^{M_1\|_{\mathcal{A}}M_2}$ *est définie par les règles suivantes :*

R1 (Evènements internes et exécutables sur une interface supérieure)

$$\frac{(q_1, a, q_1') \in \Delta^{M_1}, a \in \Sigma_U^{M_1} \cup \{\tau\}}{((q_1, q_2, < f1 >, < f2 >), a, (q_1', q_2, < f1 >, < f2 >)) \in \Delta^{M_1\|_{\mathcal{A}}M_2}} \quad (3.1)$$

$$\frac{(q_2, a, q_2') \in \Delta^{M_2}, a \in \Sigma_U^{M_2} \cup \{\tau\}}{((q_1, q_2, < f1 >, < f2 >), a, (q_1, q_2', < f1 >, < f2 >)) \in \Delta^{M_1 \| _{\mathcal{A}} M_2}} \quad (3.2)$$

R2 (Sorties sur une interface inférieure)

$$\frac{(q_1, li!m, q_1') \in \Delta^{M_1}, li!m \in \Sigma_L^{M_1} \cap \Sigma_O^{M_1}}{((q_1, q_2, < f1 >, < \ldots f2_i \cdots >), li!m, (q_1', q_2, < f1 >, < \ldots m.f2_i \cdots >)) \in \Delta^{M_1 \| _{\mathcal{A}} M_2}} \quad (3.3)$$

$$\frac{(q_2, li!m, q_2') \in \Delta^{M_2}, li!m \in \Sigma_U^{M_2} \cap \Sigma_O^{M_2}}{((q_1, q_2, < \cdots f1_i \cdots >, < f2 >), li!m, (q_1, q_2', < \cdots m.f1_i \cdots >, < f2 >)) \in \Delta^{M_1 \| _{\mathcal{A}} M_2}} \quad (3.4)$$

R3 (Entrées sur une interface inférieure)

$$\frac{(q_1, li?m, q_1') \in \Delta^{M_1}, li?m \in \Sigma_L^{M_1} \cap \Sigma_I^{M_1} \wedge (q_1, q_2, < \cdots f1_i.m \cdots >, < f2 >) \in Q^{M_1 \| _{\mathcal{A}} M_2}}{((q_1, q_2, < \cdots f1_i.m \cdots >, < f2 >), li?m, (q_1', q_2, < \cdots f1_i \cdots >, < f2 >)) \in \Delta^{M_1 \| _{\mathcal{A}} M_2}} \quad (3.5)$$

$$\frac{(q_1, li?m, q_1') \in \Delta^{M_1}, li?m \in \Sigma_L^{M_1} \cap \Sigma_I^{M_1} \wedge (q_1, q_2, < \cdots f1_i.m \cdots >, < f2 >) \in Q^{M_1 \| _{\mathcal{A}} M_2}}{((q_1, q_2, < \cdots f1_i.m \cdots >, < f2 >), li?m, (q_1', q_2, < \cdots f1_i \cdots >, < f2 >)) \in \Delta^{M_1 \| _{\mathcal{A}} M_2}} \quad (3.6)$$

R4 (Blocages)

$$\frac{(q_1, \delta, q_1) \in \Delta(M_1)}{((q_1, q_2, < f1 >, < f2 >), \delta(1), (q_1, q_2, < f1 >, < f2 >)) \in \Delta^{M_1 \| M_2}} \quad (3.7)$$

$$\frac{(q_2, \delta, q_2) \in \Delta(M_2)}{((q_1, q_2, < f1 >, < f2 >), \delta(2), (q_1, q_2, < f1 >, < f2 >)) \in \Delta^{M_1 \| M_2}} \quad (3.8)$$

$$\frac{\begin{array}{c}(\ ((q_1, q_2, < f1 >, < f2 >), \delta(1), (q_1, q_2, < f1 >, < f2 >)) \in \Delta^{M_1 \| M_2}) \\ \wedge \ (\ ((q_1, q_2, < f1 >, < f2 >), \ \delta(2), (q_1, q_2, < f1 >, < f2 >) \in \Delta^{M_1 \| M_2})\end{array}}{((q_1, q_2, < f1 >, < f2 >), \ \delta, \ (q_1, q_2, < f1 >, < f2 >)) \in \Delta^{M_1 \| M_2}} \quad (3.9)$$

Détaillons maintenant les principes des différentes règles servant à définir la relation de transition de l'intercation asynchrone dans la définition 3.2 :

R1 (Evènements internes et exécutables sur une interface supérieure) Ces évènements ne dépendent pas de l'état des files d'attente entre les deux systèmes, mais seulement de l'état courant du système concerné par l'évènement. L'évènement a est donc exécutable lors de l'interaction s'il est exécutable pour le système M_i concerné dans l'état courant q_i de M_i. De plus, l'exécution de ces évènements n'influe pas sur l'état des files d'attente.

R2 (Sorties sur une interface inférieure) une sortie $li!m$ est exécutable par le système M_j lors de l'interaction asynchrone si ce système est dans un état où cette sortie est prévue. Le message m est alors ajouté dans la file d'attente de type FIFO de M_j vers M_k ($(j,k) = (1,2)$) numéro i (numéro correspond aux numéros des interfaces de M_j et M_k via lesquelles doit transiter ce message).

R3 (Entrées sur une interface inférieure) Pour qu'une entrée sur une interface inférieure doit exécutable, il faut que le récepteur soit dans un état où cette réception est prévue et que le message correspondant ait été envoyé par l'autre système. C'est-à-dire que le message m à recevoir soit présent dans la file FIFO en entrée de l'interface inférieure. La réception enlève alors le message de la file.

R4 (Blocages) Dans cette définition de l'interaction asynchrone, deux types de blocages sont représentés : les blocages de type $\delta(1)$ et $\delta(2)$ et les blocages de types δ. $\delta(1)$ (resp. $\delta(2)$) représente une situation où seul M_1 (resp. M_2) peut être silencieux (quiescent).

Ainsi, dans la suite, $M_1 \|_{\mathcal{A}} M_2$ représentera l'interaction asynchrone avec modélisation des blocages décrite dans la définition 3.2.

3.3.3 Projection

Dans la section 2.3.4, nous avons donné la définition de la projection d'une trace sur un ensemble d'évènements. Dans le cadre du test d'interopérabilité, il peut être nécessaire de modéliser le comportement d'un système sur une interface ou un ensemble d'interfaces. Ce comportement peut être obtenu par la projection de l'IOLTS représentant le système sur l'ensemble des évènements exécutables sur les interfaces considérées. Dans cette section, nous définissons donc la projection d'un IOLTS sur un ensemble d'évènements.

Le modèle de la projection d'un IOLTS $M = (Q, \Sigma, \Delta, q_0)$ sur un ensemble d'évènements X avec préservation de l'information sur les blocages autorisés est obtenu par les étapes suivantes.

1. Calcul de $\Delta(M)$

2. Projection de $\Delta(M)$ sur l'ensemble d'évènements X. Les évènements n'appartenant pas à X sont remplacés par des évènements internes τ. Notons que les blocages sont considérés comme appartenant à l'ensemble X même si cette appartenance n'est pas décrite explicitement. Cette étape donne pour résultat un IOLTS M_X tel que $M_X = (Q, \Sigma_X, \Delta(X), q_0)$ avec $\Sigma_X = X$ et $\Delta(X)$ défini par :
   – $\forall (q_1, a, q_1') \in \Delta$, $a \in X$, $(q_1, a, q_1') \in \Delta(X)$, $a \in \Sigma_X$
   – $\forall (q_1, a, q_1') \in \Delta$, $a \notin X$, $(q_1, \tau, q_1') \in \Delta(X)$, $a \notin \Sigma_X$
3. Calcul de $\Delta(M_X)$. En effet, il est possible que l'étape de projection ait créé des livelocks dans l'IOLTS M_X.
4. Déterminisation de $\Delta(M_X)$.
   Le résultat de cette déterminisation est l'IOLTS $M/X = (M/X, \Sigma_{M/X}, \Delta_{M/X}, q_0^X)$ tel que :
   – $Q_{M/X} = 2^Q$ est l'ensemble des parties de Q.
   – $\Sigma_X = X$
   – $q_0^X = q_0$ after ϵ
   – $\Delta_{M/X}$ est tel que : $(p, a, p') \in \Delta_{M/X}$ si $p = p'$ after a, avec p, $p' \in 2^Q$ et $a \in \Sigma_{M/X}$.

3.4 Propriétés sur les spécifications

Modèle des spécifications Soit S_1 et S_2 les modèles des spécifications sur lesquelles sont basées les implémentations I_1 et I_2 dont on veut tester l'interopérabilité. Les normes des protocoles décrivent différents choix possibles pour l'implémentation. En effet, une implémentation de protocole ne réalise pas forcément toutes les fonctions décrites dans la norme. La norme définit des *profils* décrivant des fonctionnalités obligatoires et différentes fonctionnalités optionnelles parmi toutes celles possibles. Les spécifications S_1 et S_2 manipulées dans la suite correspondent aux profils de la spécification correspondante implémentés respectivement dans I_1 et I_2. C'est donc ces profils S_1 et S_2 qui doivent être définis de façon à permettre l'interopérabilité des systèmes I_1 et I_2 les implémentant.

Compatibilité des spécifications Les (profils des) spécifications S_1 et S_2 sur lesquelles sont basées les implémentations testées I_1 et I_2 doivent être définies de façon à permettre l'interaction. Nous définissons cette propriété comme la propriété d'***iop-compatibilité*** des spécifications (pour propriété de compatibilité des spécifications pour l'interopérabilité). Deux spécifications sont considérées iop-compatibles si et seulement si, pour tout message envoyé par une des spécifications sur une des ses interfaces inférieures (interfaces utilisées pour l'interaction) après une trace de l'interac-

tion, la réception correspondante est prévue dans l'autre spécification. Formellement, cette propriété d'iop-compatibilité est dans la définition 3.3.

Définition 3.3 (Propriété d'iop-compatibilité)
Soit $S_1 = (Q^{S_1}, \Sigma^{S_1}, \Delta^{S_1}, q_0^{S_1})$ *et* $S_2 = (Q^{S_2}, \Sigma^{S_2}, \Delta^{S_2}, q_0^{S_2})$ *deux spécifications.* S_1 *et* S_2 *sont iop-compatibles ssi*
$\forall \sigma.a.\sigma' \in Traces(S_1 \|_{\mathcal{A}} S_2)$, $a \in Out_{\Sigma_L}(S_1 \|_{\mathcal{A}} S_2, \sigma)$, $\sigma' = \beta_1 ... \beta_l$, $\Rightarrow \exists \beta_i$ *tel que* $\beta_i = \bar{a}$

Plusieurs méthodes peuvent être utilisées pour vérifier cette propriété sur les spécifications. La méthode la plus utilisée en pratique s'appuie sur la simulation parallèle du modèle des spécifications (ou plutôt du modèle des profils des spécifications correspondant aux implémentations). L'interconnexion de ces spécifications est modélisée et l'exécution du système obtenue simulée. Cette simulation des spécifications via leur parcours parallèle permet de vérifier la propriété d'iop-compatibilité ou de détecter les traces risquant de mettre en échec cette propriété. Dans le cas où une trace met en échec la compatibilité des profils des spécifications, il peut être possible d'en informer les organismes de normalisation ayant décrit la norme pour que le profil correspondant puisse être corrigé/complété. Dans certains autres cas, certaines entrées peuvent être volontairement non-spécifiées. Ainsi, dans certaines spécifications, on fait l'hypothèse que, dans tout état, on peut recevoir n'importe quel message de l'alphabet des entrées. Du coup, ces entrées ne sont pas décrites dans tous les états pour éviter de surcharger la spécification. Dans ce dernier cas, il faut compléter le modèle des spécifications de façon à ce qu'elle vérifient la propriété d'iop-compatibilité.

Complétion des spécifications Dans certaines situations, il est possible que les spécifications considérées aient été désignées pour interagir mais qu'elles ne vérifient pas la propriété d'iop-compatibilité. C'est le cas par exemple quand il y a sous-spécification de certains évènements d'entrée. Dans ce type de situations, il est nécessaire de compléter le modèle des spécifications de façon à ce que la propriété d'iop-compatibilité soit vérifiée par les spécifications considérées pour le test d'interopérabilité.
Plusieurs méthodes pour compléter des spécifications existent, comme par exemple les complétions dites "angelic completion" et "demonic completion" de [vdBRT04]. Cependant, toutes ces méthodes de complétion ont été décrites pour un contexte de test de conformité. Le mode de complétion utilisé pour que des spécifications vérifient la propriété d'iop-compatibilité doit refléter ce que les protocoles sous-entendent lorsque la réception d'un message particulier n'est pas spécifiée.
Dans le cadre des protocoles, comme dit ci-dessus, l'hypothèse est généralement faite que, dans tout état, on peut recevoir n'importe quel message de l'alphabet des entrées. La réception d'un message non-spécifié (c'est-à-dire en dehors de l'alphabet d'entrée) est alors considérée comme une erreur. C'est donc ce comportement que nous avons

choisi de modéliser dans notre mode de complétion. Un état-puit, appelé "trap", est ainsi ajouté. Les entrées ajoutées par la complétion conduisent dans cet état où tous les évènements de l'alphabet de la spécification sont alors possibles. Cette complétion est décrite dans la définition 3.4.
Cependant, dans d'autres contextes ou si les spécifications proposent une autre interprétation des réceptions non explicitement décrites, d'autres méthodes de complétion peuvent être utilisées : c'est le cas en particulier lorsque la spécification prévoit que les entrées non-spécifiées dans un état donné sont rejetées, mais sans que le système change d'état courant.

Définition 3.4 (Iop-complétion d'une spécification)
Soit $S_1 = (Q^{S_1}, \Sigma^{S_1}, \Delta^{S_1}, q_0^{S_1})$ *et* $S_2 = (Q^{S_2}, \Sigma^{S_2}, \Delta^{S_2}, q_0^{S_2})$ *deux spécifications.* S_1 *est iop-complétée par rapport à* S_2 *par* $Iop\_Comp(S_1) = (Q^{Iop\_Comp(S_1)}, \Sigma^{Iop\_Comp(S_1)}, \Delta^{Iop\_Comp(S_1)}, q_0^{S_1})$ *avec :*

- $Q^{Iop\_Comp(S_1)} = Q^{S_1} \cup \{trap\}$
- $\Sigma^{Iop\_Comp(S_1)} = \Sigma^{S_1} \cup \{m | \bar{m} \in \Sigma_L^{S_2} \cap \Sigma_O^{S_2} \wedge m \notin \Sigma_I^{S_1}\}$
- *la relation de transition* $\Delta^{Iop\_Comp(S_1)}$ *est définie par :*
  - $Traces(S_1) \subseteq Traces(Iop\_Comp(S_1))$ *et*
  - $\forall \sigma \in Traces(S_1 \| _{\mathcal{A}} S_2)$,
    si $\exists\ a \in Out_{\Sigma_L}(S_2, \sigma/\Sigma^{S_2})$ *tel que* $\forall\ \sigma.a.\sigma' \in Traces(S_1 \| _{\mathcal{A}} S_2)$ *avec* $\sigma' = \beta_1 ... \beta_l$, $\nexists \beta_i$ *tel que* $\beta_i = \bar{a}$,
    alors $\forall q \in (S_1\ after\ \sigma/\Sigma_1)$, $(q, \bar{a}, trap) \in \Delta^{Iop\_Comp(S_1)}$

Remarque: Dans la suite de ce document, nous considérons que les spécifications sont iop-compatibles.

3.5 Notion d'interopérabilité et définitions formelles

3.5.1 Notion d'interopérabilité

Le test d'interopérabilité s'intéresse à des implémentations pendant leur interaction. Dans le contexte one-to-one, le test d'interopérabilité met en relation deux implémentations et est utilisé pour vérifier que ces implémentations sont capables de communiquer correctement tout en fournissant le service prévu. Il y a donc deux propriétés à vérifier lors du test d'interopérabilité :

1. **Propriété *Pr_Int* (vérification de l'interaction) :** les deux implémentations doivent communiquer correctement. C'est-à-dire que les messages envoyés par une implémentation via ses interfaces inférieures doivent être reçus par l'implémentation en interaction : c'est la propriété ***Pr_Int_1***. De plus, ces messages doivent être des messages correspondant à ceux qui sont prévus dans la spécification de l'émetteur, ce qui correspond à la propriété ***Pr_Int_2***.

2. **Propriété *Pr_Serv* (vérification du service) :** les messages envoyés par les implémentations (durant l'interaction) sur les interfaces supérieures doivent correspondre à ceux décrits dans les spécifications respectives.

Ainsi, les sorties exécutées par les implémentations doivent être correctes sur les interfaces supérieures (vérification du service, propriété Pr_Serv) et les interfaces inférieures (vérification de l'interaction, propriété Pr_Int_2). Pour la vérification de l'interaction, il faut également vérifier que la réception des messages est effectivement réalisée par l'implémentation homologue (propriété Pr_Int_1). Le test d'interopérabilité de deux implémentations doit donc vérifier deux types de propriétés qui composeront les critères d'interopérabilité définis dans la section 3.6 :

1. Vérifier que les sorties exécutées sur les deux types d'interfaces correspondent à ce qui est prévu dans les spécifications (section 3.5.2).
2. Vérifier que les sorties sur les interfaces inférieures par chacune des implémentations sont effectivement reçues par l'implémentation homologue (section 3.5.3).

Remarque: Les conditions définies ici et dans la suite de cette section portent sur des implémentations $I_1 = (Q^{I_1}, \Sigma^{I_1}, \Delta^{I_1}, q_0^{I_1})$ et $I_2 = (Q^{I_2}, \Sigma^{I_2}, \Delta^{I_2}, q_0^{I_2})$ implémentant respectivement les spécifications $S_1 = (Q^{S_1}, \Sigma^{S_1}, \Delta^{S_1}, q_0^{S_1})$ et $S_2 = (Q^{S_2}, \Sigma^{S_2}, \Delta^{S_2}, q_0^{S_2})$.

3.5.2 Vérification des sorties et des silences

Les objectifs du test d'interopérabilité sont de vérifier à la fois l'interaction entre implémentations et la réalisation du service pendant l'interaction. Pour ces deux objectifs, il faut vérifier que les sorties envoyées par les implémentations sur toutes leurs interfaces sont des sorties prévues dans les spécifications. De plus, pour que cette vérification soit complète, il faut aussi vérifier que les blocages observés sont des blocages prévus dans les spécifications [DV05].

Cette vérification des sorties se fait en comparant les sorties et blocages observés avec les sorties et blocages prévus dans les spécifications. En fonction de l'architecture de test d'interopérabilité, plusieurs approches pour effectuer cette comparaison sont possibles. Ces différentes approches se retrouvent dans les différents critères d'interopérabilité définis dans la section 3.6. Nous donnons ici les définitions formelles des propriétés Pr_Serv et Pr_Int_2 prenant en compte ces différentes architectures de test. $\forall \sigma_1 \in Traces(\Delta(S_1))$, $\sigma_1 = \sigma/\Sigma^{S_1}$, $\exists \sigma \in Traces(S_1 \|_{\mathcal{A}} S_2) \Rightarrow Out((I_1 \|_{\mathcal{A}} I_2)/\Sigma^{S_1}$, $\sigma_1) \subseteq Out(\Delta(S_1),\sigma_1)$.

Conditions prenant en compte des architectures d'interopérabilité totale Il y a trois architectures de test possibles pour le contexte d'interopérabilité totale : architec-

ture globale totale, architecture unilatérale totale, et architecture bilatérale totale (cf. sections 2.5.3 du chapitre 2 et 3.2 du chapitre 3).
L'approche de test d'interopérabilité globale totale compare les sorties observées globalement sur toutes les interfaces des deux implémentations en interaction avec les sorties prévues dans l'interaction des spécifications dans le même contexte. Formellement, le condition permettant la vérification des sorties et blocages dans un contexte global total est donc :
$\forall \sigma \in Traces(S_1 \|_{\mathcal{A}} S_2), Out(I_1 \|_{\mathcal{A}} I_2, \sigma) \subseteq Out(S_1 \|_{\mathcal{A}} S_2, \sigma)$.

L'approche de test d'interopérabilité unilatérale totale considère les sorties observées sur les interfaces d'une des implémentations lors de son interaction avec la deuxième implémentation. Elle compare ces sorties avec les sorties prévues dans la spécification de l'implémentation considérée. Formellement, la condition permettant la vérification des sorties et blocages dans un contexte unilatéral total (se focalisant sur l'observation de I_1 pendant son interaction avec I_2) est définie par :

L'approche de test d'interopérabilité bilatérale totale compare les sorties observées séparément sur toutes les interfaces des deux implémentations en interaction avec les sorties prévues dans les spécifications correspondantes. La condition sur les sorties et blocages correspondant à ce contexte est en fait composée de deux conditions correspondant à des contextes d'interopérabilité unilatérale totale. La première partie se focalise ainsi sur les sorties et blocages de I_1 pendant son interaction avec I_2 tandis que la deuxième partie porte sur les sorties et blocages de I_2 pendant son interaction avec I_1.
Remarque : les conditions décrites dans cette section peuvent également être décrite en utilisant la relation de conformité **ioco**. Cependant, ces définitions ne sont pas tout à fait équivalentes à la définition correspondante utilisant **ioco** à cause des différentes hypothèses considérées en conformité et en interopérabilité. La condition basée sur une approche globale totale peut aussi s'écrire $I_1 \|_{\mathcal{A}} I_2 \, ioco \, S_1 \|_{\mathcal{A}} S_2$. De même, la condition considérant un contexte unilatéral total peut s'écrire $(I_1 \|_{\mathcal{A}} I_2)/\Sigma^{S_1} \, ioco \, S_1$.

Conditions prenant en compte des contextes d'interopérabilité inférieure ou supérieure Les trois contextes d'interopérabilité précédents peuvent aussi se décliner pour des contextes où les testeurs n'ont accès qu'à un seul type d'interfaces. Soit X le type d'interfaces considéré ; on pourra avoir $X = L$ (interfaces inférieures) ou $X = U$ (interfaces supérieures). Nous définissons ici les conditions d'interopérabilité globale et unilatérale sur les sorties et blocages pour un contexte dans lequel seules les interfaces décrites dans l'ensemble X sont accessibles.

Dans ce contexte, une condition d'interopérabilité globale considère, avec un point de vue global, les sorties observées sur l'ensemble des interfaces de X et compare ces sorties avec ce qui est autorisé dans la même situation par le modèle des spécifi-

cations projeté sur les interfaces observables. Notons que la trace considérée devant être exécutée par les implémentations (et observée par le testeur), elle doit aussi être limitée aux interfaces observables dans le contexte considéré. Formellement, on obtient la condition suivante : $\forall \sigma \in Traces((S_1 \|_{\mathcal{A}} S_2)/\Sigma_X)$, $Out((I_1 \|_{\mathcal{A}} I_2)/\Sigma_X, \sigma) \subseteq Out((S_1 \|_{\mathcal{A}} S_2)/\Sigma_X, \sigma)$.

De même, la condition d'interopérabilité unilatérale pour un contexte d'interopérabilité limité aux interfaces de X est obtenue à l'aide de projection sur l'ensemble des évènements observables sur les interfaces X : $\forall \sigma_1 \in Traces(\Delta(S_1)/\Sigma_X)$, $\forall \sigma \in Traces((S_1 \|_{\mathcal{A}} S_2)/\Sigma_X)$, $\sigma/\Sigma^{S_1} = \sigma_1 \Rightarrow Out((I_1 \|_{\mathcal{A}} I_2)/\Sigma_X^{S_1}, \sigma_1) \subseteq Out(\Delta(S_1)/\Sigma_X, \sigma_1)$.

3.5.3 Vérification des entrées et dépendances causales entre évènements

Un des objectifs du test d'interopérabilité est de vérifier que les deux implémentations en interaction communiquent correctement. Pour cela, il faut vérifier que les messages envoyés par une implémentation sur son interface inférieure soient corrects par rapport à la spécification correspondante (cf. propriété Pr_Int_2 de la section 3.5.1). C'est ce qui est effectué par la vérification des sorties et des silences décrite dans la section 3.5.2. Il faut également vérifier que ces messages sont effectivement reçus (c'est-à-dire traités) par l'implémentation réceptrice (cf. propriété Pr_Int_1 de la section 3.5.1).

Remarque : ce type de problème peut être rapproché des problèmes de diagnostic [CL98, BLPZ98] qui consistent à vérifier qu'une propriété sur des évènements internes est satisfaite en se basant sur des observations partielles. En effet, on veut ici vérifier la prise en compte d'une entrée à partir de l'observation de d'autres évènements.

La vérification de la réception effective d'un message correspond à la gestion des entrées par le test d'interopérabilité. Cependant, lors de l'activité de test, les entrées ne peuvent pas être observées par les testeurs. En effet, le testeur peut savoir si un message a été transmis à une implémentation via son interface inférieure, mais pas si cette implémentation a effectivement traité le message reçu. Elle peut par exemple être bloquée parce qu'elle n'a pas pu traiter ce message ou l'avoir effectivement reçu et donc être capable de continuer à s'exécuter.

Comme les entrées ne peuvent pas être observées, il faut se baser sur d'autres évènements (observables) pour pouvoir vérifier qu'une entrée particulière a bien été exécutée et ainsi vérifier que les implémentations sont capables de communiquer. Ainsi, vérifier qu'une entrée μ a bien été reçue implique d'avoir déterminé l'ensemble des sorties qui ne peuvent être exécutées (et donc observées) que si la réception a bien été

exécutée. Cet ensemble de sortie est calculé à partie des ***dépendances causales*** entre évènements (ici entre une entrée et un ensemble de sorties). Cet ensemble de sorties dépendances causales d'une entrée est défini dans la définition 3.5.

Définition 3.5 (Ensemble de sorties dépendances causales d'une entrée)
L'ensemble des sorties de la spécification $S = (Q^S, \Sigma^S, \Delta^S, q_0^S)$ *qui sont des dépendances causales d'une entrée* μ *après une trace* σ *est noté* $CDep(S, \sigma, \mu)$ *et est défini par :*
$CDep(S, \sigma, \mu) = \{\alpha_i \in \Sigma_O^S \setminus \{\delta\} \mid \forall (q, q'), q_0^S \stackrel{\sigma}{\Rightarrow} q \stackrel{\mu}{\rightarrow} q', \exists q_i, q' \stackrel{\sigma_i.\alpha_i}{\Longrightarrow} q_i, \sigma_i \in (\Sigma_I^S)^*\}$, *où* $\sigma_i \in (\Sigma_I^S)^*$ *est la trace (qui peut ne pas contenir d'élément) associée à la sortie* $\alpha_i \in CDep(S, \sigma, \mu)$ *(c'est-à-dire le chemin entre* μ *et* α_i*).*

Cet ensemble est donc défini comme un ensemble de sorties ($\{\alpha_i \in \Sigma_O^S \setminus \{\delta\}\}$, ensemble sans évènement de blocage). La trace σ mène de l'état initial q_0 à un état q tel que μ peut être exécuté dans cet état. La spécification S est dans l'état q' après l'exécution de μ. L'ensemble des sorties dépendances causales est ensuite l'ensemble des sorties exécutables après des traces notées σ_i (trace ne contenant pas de sortie) amenant S dans un état q_i. Si $\sigma_i = \epsilon$, alors $q_i = q'$. Compte-tenu des dépendances entre évènements d'un IOLTS, ces sorties sont des évènements qui doivent être exécutés si $\sigma.\mu$ a été exécuté par le système correspondant. Il est cependant possible que ces sorties nécessitent l'exécution combinée de $\sigma.\mu$ et d'autres évènements (décrits par la trace σ_i).

A partir de la définition de cet ensemble de dépendances causales d'une entrée, il est possible de définir une condition à vérifier lors des tests pour vérifier l'exécution effective d'une entrée. La condition que nous décrivons dans cette section concerne la vérification de l'exécution d'une entrée $\bar{\mu}$ par l'implémentation I_2, la sortie correspondante étant la sortie μ envoyée par l'implémentation I_1. Cette condition porte sur toutes les sorties μ exécutables par I_1 après une trace de l'interaction notée σ. Cette trace σ peut être décomposée par projection en une trace σ_1 composée d'évènements de I_1 et une trace σ_2 d'évènements de I_2. La condition de gestion des entrées (cf. propriété Pr_Int_1 de la section 3.5.1) dit alors que la réception effective de $\bar{\mu}$ par I_2 implique l'observation d'une sortie qui dépend causalement de la réception de $\bar{\mu}$. Cette condition prévoit aussi la possibilité d'exécuter quelques évènements entre la sortie μ et l'entrée correspondante $\bar{\mu}$, et entre cette réception $\bar{\mu}$ et une sortie qui en est une dépendance causale. Dans la condition, les évènements possibles entre μ et $\bar{\mu}$ sont décrit par les traces σ' telles que $\sigma' \in ((\Sigma^{S_1} \cup \Sigma^{S_2}) \setminus \bar{\mu})^* \cup \{\epsilon\}$. De même, les évènements possibles entre $\bar{\mu}$ et une sortie sont décrits dans les traces σ_i telles que $\sigma_i \in (\Sigma_I^{S_2})^* \cup \{\epsilon\}$. Notons que l'ensemble $CDep$ des dépendances causales est un ensemble ordonné : σ_i est ainsi la trace des évènements qui doivent être exécutés entre $\bar{\mu}$ et $CDep(i)$.
Formellement, la condition (correspondant à la propriété Pr_Int_1) sur la réception

effective des messages envoyés par une implémentation à l'autre implémentation est définie par (I_1 est l'émetteur, I_2 le récepteur) :
$Pr\_Int\_1(I_2, I_1) =_\Delta \forall \sigma \in Traces(S_1 \|_\mathcal{A} S_2)$, $\sigma_1 = \sigma/\Sigma^{S_1} \in Traces(\Delta(S_1))$, $\sigma_2 = \sigma/\Sigma^{S_2} \in Traces(\Delta(S_2))$, $\forall \mu \in Out_{\Sigma_L^{I_1}}(\Delta(I_1), \sigma_1), \forall \sigma' \in ((\Sigma^{S_1} \cup \Sigma^{S_2} \cup \{\delta(1), \delta(2), \delta\}) \setminus \{\bar{\mu}\})^*$, $\sigma.\mu.\sigma'.\bar{\mu} \in Traces(S_1 \|_\mathcal{A} S_2)$,
$\bar{\mu} \in In(I_2, \sigma_2.(\sigma'/\Sigma^{I_2})) \Rightarrow$
$Out(I_2, \sigma_2.(\sigma'/\Sigma^{I_2}).\bar{\mu}.\sigma_i) \subseteq CDep(S_2, \sigma_2.(\sigma'/\Sigma^{I_2}), \bar{\mu})$ avec $\sigma_i \in (\Sigma_I^{S_2})^*$.

Remarque: $Out_{\Sigma_L^M}(M, \sigma)$ représente l'ensemble $Out(M, \sigma)$ réduit aux évènements de Σ_L^M.

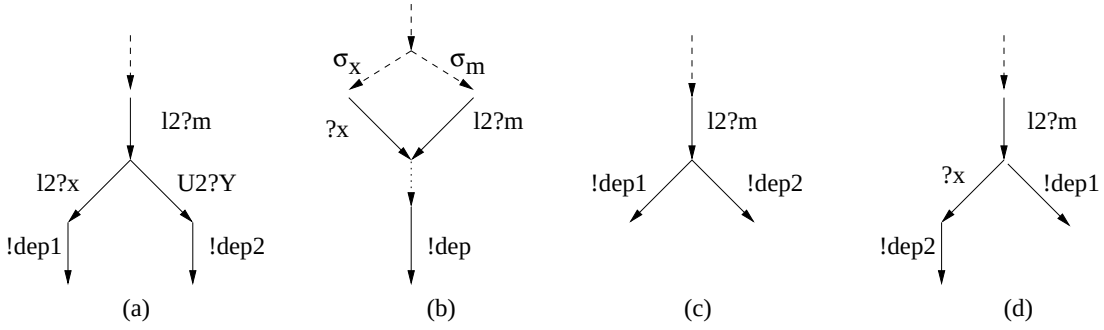


FIG. 3.1 – Exemples de dépendances causales de l'entrée $l2?m$

La figure 3.1 illustre quelques possibilités classiquement observées de dépendances causales d'une entrée $\bar{\mu} = l2?m$. L'absence de précision pour l'interface de transmission des messages signifie que l'interprétation n'est pas dépendante du type (supérieure ou inférieure) d'interface considéré.
Le cas le plus simple est l'exemple (c) : dans cet exemple, les sorties $!dep1$ et $!dep2$ sont toutes deux des sorties dépendances causales de $\bar{\mu}$ et leur observation permet de conclure sur l'exécution de $\bar{\mu} = l2?m$.
Dans l'exemple (a), $!dep1$ et $!dep2$ sont également des sorties dépendances causales de $\bar{\mu}$. Cependant, leur exécution nécessite une autre entrée ($l2?x$ ou $U2?Y$). Il faut donc que le message correspondant (x ou Y) soit transmis à l'implémentation (soit par le testeur soit par l'autre implémentation). L'exécution de la sortie dépendance causale permet ici de conclure à l'exécution combinée de $\bar{\mu}$ et $l2?x$ (sortie $!dep1$) ou de $\bar{\mu}$ et $U2?Y$ (sortie $!dep2$). Dans les deux cas, cette dépendance causale permet de conclure que $\bar{\mu} = l2?m$ a bien été exécuté.
Dans l'exemple (b) de la figure 3.1, $!dep$ est une dépendance causale de $l2?m$, mais peut également être une dépendance causale de $?x$. La conclusion dans une telle situation dépend de la connaissance des évènements exécutés avant $l2?m$ ou $?x$. En effet, si par exemple, x n'a pas été envoyé à l'implémentation, $!dep$ permet de conclure sur

l'exécution de $l2?m$. Par contre, si les messages m et x ont tous deux été transmis à l'implémentation, l'observation de $!dep$ peut ne pas suffire pour conclure sur l'exécution de $l2?m$. Le verdict ici serait $Inconclusive$.
Dans l'exemple (d) de la figure 3.1, $!dep1$ et $!dep2$ sont deux sorties dépendances causales de $\bar{\mu} = l2?m$. $!dep1$ est une sortie dépendance causale de $\bar{\mu}$ qui peut être exécutée directement après $\bar{\mu}$ alors que l'exécution de $!dep2$ est dépendante à la fois de $\bar{\mu}$ et de $?x$ (dans ce cas, la trace σ' de la condition définie ci-dessus est telle que $\sigma' = ?x$). Le choix entre ces deux traces possibles après $\bar{\mu}$ est basé sur des calculs internes (donc des évènements non-observables). Dans le cas où l'implémentation ne choisit pas d'exécuter l'évènement $!dep1$, un blocage est généralement observé. Dans ce cas, pour pouvoir observer une sortie dépendance causale de $\bar{\mu}$, il faut que le message x soit envoyé à l'implémentation. Dans le cas où l'entrée $?x$ doit être exécutée sur une interface supérieure, c'est le testeur qui doit fournir ce message. Dans le cas où l'entrée $?x$ doit être exécutée sur une interface inférieure, il faut placer l'autre implémentation en position d'envoyer le message correspond. Cet exemple montre ainsi l'importance de mémoriser la trace σ' des évènements à exécuter avant chaque sortie dépendance causale de façon à placer l'implémentation dans un état où elle peut exécuter une telle sortie.

Remarque 1 : dans toute cette section, nous affirmons que "une entrée n'est pas un évènement observable". En fait, les testeurs peuvent savoir quel message est envoyé vers quelle interface (en observant l'activité sur cette interface avec ethereal/wireshark par exemple) et donc connaître les entrées exécutées par l'implémentation. Cependant, lorsqu'un message est mis en entrée d'une interface, l'implémentation n'est pas forcement capable de traiter le message correspondant et de réaliser les différentes actions correspondant à cette réception. Par abus de langage, c'est cette opération de traitement que nous appelons entrée ou réception dans l'affirmation "qu'une entrée n'est pas un évènement observable". Ainsi, ce que l'on veut vérifier ici est la capacité de l'implémentation à traiter les messages reçus.

Remarque 2 : Les évènements utilisés pour vérifier le traitement effectif d'une entrée par une implémentation sont des évènements décrits dans la spécification correspondante comme des évènements dépendants de l'entrée considérée. Ainsi, une hypothèse sur le comportement de l'implémentation est implicitement faite. En effet, pour qu'une telle méthode de vérification de traitement d'une entrée soit applicable, il faut que l'implémentation correspondante se comporte comme prévu dans sa spécification. En particulier, il n'est pas possible ici de différencier une erreur due au non-traitement d'une entrée et une erreur due à une mauvaise implémentation des évènements consécutifs de l'entrée considérée.

La condition ci-dessus a été définie pour un contexte de test d'interopérabilité avec accès aux deux types d'interfaces, et accès aux deux implémentations.

Dans un contexte unilatéral, la condition se focalisera sur la trace σ_2, mais vérifiera tout de même que la sortie μ est autorisée par la spécification de I_1.
Dans les contextes avec accès seulement aux interfaces inférieures, les sorties dépendances causales ne peuvent être que des sorties sur les interfaces inférieures. Comme les testeurs n'ont pas accès aux interfaces supérieures, il est possible qu'il y ait un blocage dans les traces σ ou σ_i dû à l'attente d'un stimulus sur les interfaces supérieures. Ainsi, comme ces interfaces ne sont pas accessibles (ni pour observation, ni pour contrôle), l'exécution de l'entrée $\bar{\mu}$ peut ne pas être vérifiable en pratique.
Dans les contextes avec accès seulement aux interfaces supérieures, cette condition ne peut être mise en pratique. En effet, il n'est pas possible de savoir quels messages ont effectivement été envoyés sur les interfaces inférieures, donc inutile de vérifier si les entrées correspondantes ont bien été exécutées.

3.5.4 Modèle des implémentations

Dans le cadre du test de conformité, différentes hypothèses étaient appliquées sur les implémentations. En particulier, les implémentations étaient considérées "input-complete", c'est-à-dire qu'elles sont toujours capables de recevoir les messages envoyés par les testeurs sur leurs interfaces. Cette hypothèse est logique dans un contexte de test de conformité où seuls le testeur et l'IUT interagissent, et que le testeur a pour objectif de s'assurer que l'IUT réagit correctement par rapport aux stimuli (prévus dans la spécification). La situation est totalement différente pour le test d'interopérabilité où le testeur interagit avec des IUTs, chacune interagissant avec l'autre IUT. Ainsi, le testeur ne contrôle pas ce que chacune des IUTs peut envoyer à l'autre sur les interfaces inférieures. En l'occurrence, ces messages peuvent ne pas correspondre à ce qui est prévu dans la spécification. De façon analogue à ce qui est fait pour le test de conformité, il convient donc de définir des hypothèses réalistes pour le test d'interopérabilité, qui traduisent cette spécificité.

On peut remarquer que la connexion entre testeurs et interfaces des implémentations est comparable dans les contextes de conformité et d'interopérabilité pour les interfaces supérieures. Nous avons donc choisi de faire l'hypothèse que les implémentations étaient capables de traiter à tout moment les messages qui leur sont fournis via les interfaces supérieures.

Par contre, le rôle des interfaces inférieures est complètement différent pour l'interopérabilité et la conformité. En effet, les interfaces inférieures sont seulement observables en interopérabilité alors qu'elles sont contrôlables en conformité : les testeurs transmettent les messages lors du test de conformité tandis que lors du test d'interopérabilité, les messages reçus par ces interfaces sont envoyés par l'implémentation en interaction. L'un des objectif de l'interopérabilité est alors (via les dépendances causales entre évènements) de vérifier si ces messages ont effectivement été reçus. Il ne

faut donc pas faire d'hypothèse sur la capacité à recevoir les entrées pour les interfaces inférieures.

Ainsi, l'hypothèse (faite en conformité) que les implémentations sont capables de recevoir les messages envoyés sur leurs interfaces n'est valable dans le contexte du test d'interopérabilité que pour les interfaces supérieures, connectées de façon directe aux testeurs.

Remarque : Différentes hypothèses peuvent également être faites au niveau de l'environnement : environnement lent, environnement raisonnable, etc. Dans le cadre du test d'interopérabilité, nous considérons un environnement "raisonnable". C'est-à-dire qu'il n'est pas nécessaire d'attendre que les files d'attente des systèmes soient vides pour envoyer un stimulus (cas de l'environnement lent), mais qu'un stimulus peut être envoyé à l'IUT (ou aux IUTs) dans le cas où les files sont non vides mais que ce stimulus est attendu par l'implémentation avant le traitement de l'entrée en attente. L'hypothèse d'un environnement raisonnable est ainsi moins contraignante que celle d'un environnement lent car elle permet d'adapter les stimulus envoyés par le(s) testeur(s) aux situations prévues dans les spécifications.

3.6 Définitions formelles : critères d'interopérabilité

Dans cette section, nous présentons différentes définitions formelles de la notion d'interopérabilité appelées ***critères d'interopérabilité*** (ou en anglais *interoperability criteria* ou *iop criteria*), ou *critères d'iop*. Un critère d'interopérabilité one-to-one décrit formellement les différentes conditions que doivent remplir deux implémentations pour être considérées interopérables.
Ces différents critères d'interopérabilité sont basés sur les différents contextes (ou architectures) d'interopérabilité décrits dans la section 3.2.
Dans cette section, nous classons les critères d'interopérabilité définis en fonction du type d'interfaces auxquelles les testeurs accèdent pendant les tests.

3.6.1 Critères d'interopérabilité totale

Dans cette section, nous présentons les critères d'interopérabilité considérant que les testeurs ont accès aux deux types d'interfaces (inférieures et supérieures). Ces critères sont appelés critères d'interopérabilité totale (ou *total interoperability criteria*).
Le critère d'interopérabilité unilatérale totale iop_U (unilateral total iop criterion) dit que deux implémentations sont interopérables si et seulement si :

– après une trace (suspensive) de de la spécification S_1 observée aux interfaces de I_1 lors de son interaction avec I_2, toutes les sorties (incluant les blocages)

observées aux interfaces de l'implémentation I_1 lors de son interaction avec I_2 doivent être prévues dans la spécification S_1,
– et, les messages reçus par I_1 sur son interface inférieure doivent être traités par cette implémentation sous réserve que ces messages soient autorisés par la spécification (S_2) de l'émetteur (I_2).

Définition 3.6 (Critère d'interopérabilité unilatérale iop_U)
$I_1\ iop_U\ I_2 =_\Delta$

1. $\forall \sigma_1 \in Traces(\Delta(S_1)), \sigma_1 = \sigma/\Sigma^{S_1}, \exists \sigma \in Traces(S_1 \|_\mathcal{A} S_2) \Rightarrow Out((I_1\|_\mathcal{A} I_2)/\Sigma^{S_1}, \sigma_1) \subseteq Out(\Delta(S_1),\sigma_1)$
2. *et,* $\forall \sigma_1 = \sigma/\Sigma^{S_1} \in Traces(\Delta(S_1))$ *tel que* $\sigma \in Traces(S_1\|_\mathcal{A} S_2), \forall \mu \in Out_{\Sigma_L^{I_2}}(I_2, \sigma/\Sigma^{I_2}), \forall \sigma' \in [(\Sigma^{S_1} \cup \Sigma^{S_2} \cup \{\delta(1), \delta(2), \delta\}) \setminus \{\bar{\mu}\}]^*, \sigma.\mu.\sigma'.\bar{\mu} \in Traces(S_1\|_\mathcal{A} S_2), \bar{\mu} \in In(I_1, \sigma_1.(\sigma'/\Sigma^{I_1})) \Rightarrow Out(I_1, \sigma_1.(\sigma'/\Sigma^{I_1}).\bar{\mu}.\sigma_i) \subseteq CDep(S_1, \sigma_1.(\sigma'/\Sigma^{I_1}), \bar{\mu}), \sigma_i \in (\Sigma_I^{S_1})^*$

Le critère d'interopérabilité unilatérale n'est pas symétrique. En effet, il est possible d'avoir $I_1\ iop_U\ I_2$ alors que $\neg\ I_2\ iop_U\ I_1$ (ou inversement).

Le critère d'interopérabilité bilatérale totale iop_B (bilateral total iop criterion) dit que deux implémentations sont interopérables si et seulement si les deux critères d'interopérabilité unilatérale $I_1\ iop_U\ I_2$ et $I_2\ iop_U\ I_1$ sont vérifiés.

Définition 3.7 (Critère d'interopérabilité bilatérale iop_B)
$I_1\ iop_B\ I_2 =_\Delta I_1\ iop_U\ I_2 \wedge I_2\ iop_U\ I_1$

Le critère d'interopérabilité globale totale iop_G (global total iop criterion) dit que deux implémentations sont interopérables si et seulement si :
– après une trace (suspensive) de l'interaction des spécifications et observée lors de l'interaction des implémentations, toutes les sorties (incluant les blocages) observées aux interfaces des implémentations doivent être prévues dans l'interaction des spécifications,
– et, les messages envoyés par chacune des implémentations (via ses interfaces inférieures) à l'autre implémentation doivent être effectivement reçues par cette dernière. Ces messages devant respecter la première partie du critère d'interopérabilité, ils doivent être autorisés par les spécifications lors de leur interaction.

Définition 3.8 (Critère d'interopérabilité globale iop_G)
$I_1\ iop_G\ I_2 =_\Delta$

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

2. *et,* $\forall\{i,j\} = \{1,2\}, i \neq j$,
$\forall\sigma \in Traces(S_i \|_{\mathcal{A}} S_j)$, $\sigma_i = \sigma/\Sigma^{S_i} \in Traces(S_i)$, $\sigma_j = \sigma/\Sigma^{S_j} \in Traces(S_j)$,
$\forall\mu \in Out_{\Sigma_L^{I_i}}(I_i, \sigma/\Sigma^{S_i})$, $\forall\sigma' \in [(\Sigma^{S_i} \cup \Sigma^{S_j} \cup \{\delta(i), \delta(j), \delta\}) \setminus \{\bar{\mu}\}]^*$, $\sigma.\mu.\sigma'.\bar{\mu} \in Traces(S_i \|_{\mathcal{A}} S_j)$, $\bar{\mu} \in In(I_j, \sigma_j.(\sigma'/\Sigma^{I_j})) \Rightarrow Out(I_j, \sigma_j.(\sigma'/\Sigma^{I_j}).\bar{\mu}.\sigma_k) \subseteq CDep($
$S_j, \sigma_j.(\sigma'/\Sigma^{I_j}), \bar{\mu})$, $\sigma_k \in (\Sigma_I^{S_j})^*$

Remarque: Ce critère correspond au contexte d'interopérabilité pour l'interaction de deux implémentations le plus général et à l'architecture de test d'interopérabilité la plus utilisée en pratique.

Ces deux derniers critères d'interopérabilité (iop_B et iop_G), contrairement à iop_U, sont des critères symétriques : $I_1 iop_B I_2 \Leftrightarrow I_2 iop_B I_1$ et $I_1 iop_G I_2 \Leftrightarrow I_2 iop_G I_1$. De plus, ces deux critères considèrent des contextes avec accès à toutes les interfaces des deux implémentations. La différence principale entre critères d'interopérabilité globale et bilatérale réside dans le mode d'observation et de contrôle de ces interfaces. En effet, iop_G considère le système composé des deux implémentations en interaction de façon globale tandis que iop_B considère chaque implémentation séparément lors de leur interaction. Ainsi, dans des tests d'interopérabilité basés sur iop_G, les testeurs connectés aux deux implémentations doivent se synchroniser au fur et à mesure des observations et envois de stimuli, alors que cette synchronisation entre testeurs T_1 et T_2 (de l'architecture de la figure 2.4 section 3.2) n'est pas nécessaire dans le cadre de tests d'interopérabilité basés sur iop_B.

3.6.2 Critères d'interopérabilité basés interfaces inférieures

Dans cette section, nous nous intéressons aux critères d'interopérabilité basés sur les interfaces inférieures (ou *lower interoperability criteria*), c'est-à-dire aux critères qui se focalisent sur la vérification de la communication entre les deux implémentations. Ces critères d'interopérabilité sont notés iop^L et appelés critères d'interopérabilité inférieure par analogie avec le type d'interfaces accédées lors des tests.

Le critère d'interopérabilité unilatérale basé interfaces inférieures iop_U^L dit que deux implémentations sont interopérables si et seulement si :

- après une trace projetée sur les interfaces inférieures de S_1 et observée aux interfaces inférieures de I_1, toutes les sorties (incluant les blocages) observées aux interfaces inférieures de I_1 doivent être prévues dans la spécification S_1. La trace considérée est une trace suspensive, c'est-à-dire incluant potentiellement des blocages. De plus, l'observation de cette trace aux interfaces inférieures de I_1 doit être réalisée lors de l'interaction de I_1 et I_2.
- et, les messages reçus par I_1 sur son interface inférieure doivent être traités par cette implémentation sous réserve que ces messages soient autorisés par la spécification (S_2) de l'émetteur (I_2).

Définition 3.9 (Critère d'iop unilatérale basé interfaces inférieures iop_U^L)
$I_1 iop_U^L I_2 =_\Delta$

1. $\forall \sigma_1 \in Traces(\Delta(S_1)/\Sigma_L)$, $\sigma_1 = \sigma/\Sigma^{S_1}$, $\exists \sigma \in Traces((S_1 \|_\mathcal{A} S_2)/\Sigma_L) \Rightarrow Out((I_1 \|_\mathcal{A} I_2)/\Sigma_L^{S_1}, \sigma_1) \subseteq Out(\Delta(S_1)/\Sigma_L, \sigma_1)$
2. *et,* $\forall \sigma_1 = \sigma/\Sigma_L^{S_1} \in Traces(\Delta(S_1)/\Sigma_L^{S_1})$ *tel que* $\sigma \in Traces((S_1 \|_\mathcal{A} S_2)/(\Sigma_L^{S_1} \cup \Sigma_L^{S_2}))$, $\forall \mu \in Out(I_2/\Sigma_L^{S_2}, \sigma/\Sigma^{I_2})$, $\forall \sigma' \in [(\Sigma_L^{S_1} \cup \Sigma_L^{S_2} \cup \{\delta(1), \delta(2), \delta\}) \setminus \{\bar{\mu}\}]^*$, $\sigma.\mu.\sigma'.\bar{\mu} \in Traces((S_1 \|_\mathcal{A} S_2)/(\Sigma_L^{S_1} \cup \Sigma_L^{S_2}))$, $\bar{\mu} \in In(I_1/\Sigma_L^{S_1}, \sigma_1.\ (\sigma'/\Sigma^{I_1})) \Rightarrow Out(I_1/\Sigma_L^{S_1}, \sigma_1.(\sigma'/\Sigma^{I_1}).\bar{\mu}.\sigma_i) \subseteq CDep(S_1/\Sigma_L^{S_1}, \sigma_1.(\sigma'/\Sigma^{I_1}), \bar{\mu})$, $\sigma_i \in (\Sigma_I^{S_1} \cap \Sigma_L^{S_1})^*$

Remarque: L'ensemble des dépendances causales $CDep(S_1/\Sigma_L^{S_1}, \sigma_1.(\sigma'/\Sigma^{I_1}), \bar{\mu})$ ne contient que des sorties (sans blocages) sur les interfaces inférieures. Cependant, il est possible que l'évènement observé soit un blocage sans que les testeurs puissent "débloquer" les implémentations et donc permettre l'observation d'une sortie de l'ensemble des dépendances causales. C'est le cas particulièrement lorsque l'observation d'une sortie dépendance causale nécessite, en plus de l'entrée considérée, un stimulus sur une interface supérieure. Dans une architecture générale, ce stimulus peut être appliqué par les testeurs, mais pas dans une architecture d'interopérabilité inférieure. Ainsi, les sorties dépendances causales ne sont pas observées à cause de l'inaccessibilité des interfaces supérieures de cette architecture. Cependant, les évènements observés par les testeurs restent conformes aux spécifications. Cette vérification dans une telle architecture de test peut ainsi générer des verdicts inconclusifs.

De même que précédemment, le critère d'interopérabilité bilatérale basé interfaces inférieures iop_B^L dit que deux implémentations sont interopérables si et seulement si les deux critères d'interopérabilité unilatérale inférieure $I_1 iop_U^L I_2$ et $I_2 iop_U^L I_1$ sont vérifiés.

Définition 3.10 (Critère d'iop bilatérale basé interfaces inférieures iop_B^L)
$I_1 iop_B^L I_2 =_\Delta I_1 iop_U^L I_2 \wedge I_2 iop_U^L I_1$

Le critère d'interopérabilité globale basé interfaces inférieures iop_G^L dit que deux implémentations sont interopérables si et seulement si :

- après une trace (suspensive) de l'interaction des spécifications projetée sur les interfaces inférieures et observée sur les interfaces inférieures des implémentations lors de leur interaction, toutes les sorties (incluant les blocages) observées aux interfaces inférieures des implémentations doivent être prévues dans l'interaction des spécifications,
- et, les messages envoyés par chacune des implémentations (via ses interfaces inférieures) à l'autre implémentation doivent être effectivement reçus par cette dernière, ce qui ne peut être vérifié que via une sortie dépendance causale du message reçu exécutée sur les interfaces inférieures de I_1.

Définition 3.11 (Critère d'iop globale basé interfaces inférieures iop_G^L)
$I_1 iop_G^L I_2 =_\Delta$

1. $\forall \sigma \in Traces((S_1 \|_\mathcal{A} S_2)/\Sigma_L), Out((I_1 \|_\mathcal{A} I_2)/\Sigma_L, \sigma) \subseteq Out((S_1 \|_\mathcal{A} S_2)/\sigma_L, \sigma)$
2. *et,* $\forall \{i,j\} = \{1,2\}, i \neq j,$
   $\forall \sigma \in Traces((S_i \|_\mathcal{A} S_j)/(\Sigma_L^{S_1} \cup \Sigma_L^{S_2})), \sigma_i = \sigma/\Sigma^{S_i} \in Traces(S_i/\Sigma^{S_i}), \sigma_j = \sigma/\Sigma^{S_j} \in Traces(S_j/\Sigma^{S_j}), \forall \mu \in Out(I_i/\Sigma_L^{S_i}, \sigma/\Sigma^{S_i}), \forall \sigma' \in [(\Sigma_L^{S_i} \cup \Sigma_L^{S_j} \cup \{\delta(i), \delta(j), \delta\}) \setminus \{\bar{\mu}\}]^*, \sigma.\mu.\sigma'.\bar{\mu} \in Traces((S_i \|_\mathcal{A} S_j)/(\Sigma_L^{S_1} \cup \Sigma_L^{S_2})), \bar{\mu} \in In(I_j, \sigma_j.(\sigma'/\Sigma^{I_j})) \Rightarrow Out(I_j/\Sigma_L^{S_j}, \sigma_j.(\sigma'/\Sigma^{I_j}).\ \bar{\mu}.\sigma_k) \subseteq CDep(S_j/\Sigma_L^{S_j}, \sigma_j.(\sigma'/\Sigma^{I_j}), \bar{\mu}), \sigma_k \in (\Sigma_I^{S_j} \cap \Sigma_L^{S_j})^*$

Remarque: La même remarque que pour le critère d'interopérabilité unilatérale inférieure s'applique au critère d'interopérabilité globale inférieure : les entrées sur les interfaces inférieures peuvent ne pas être vérifiables. La cause en est de même le non-accès aux interfaces supérieures. C'est le cas par exemple lorsque la première sortie dépendance causale est une sortie exécutable sur une interface supérieure. Dans ce cas, cette sortie n'est pas observable. De plus, les évènements qui permettrait d'exécuter une sortie dépendance causale sur une interface inférieure peuvent nécessiter l'exécution d'évènements supplémentaires. En particulier, ces évènements peuvent être des stimuli sur une interface supérieure dont l'exécution n'est pas connue, mais également qui peuvent sortir du cadre de l'objectif de test (cf. exemple dans la section 3.7).

3.6.3 Critères d'interopérabilité basés interfaces supérieures

Dans cette section, nous nous intéressons aux critères d'interopérabilité basés interfaces supérieures (ou *upper interoperability criteria*), c'est-à-dire aux critères qui se focalisent sur la vérification de la fourniture du service lors de l'interaction des spécifications. Ces critères sont également appelés critères d'interopérabilité supérieure (par analogie avec le type d'interface accédées lors des tests) et sont noté iop^U.
Comme ces critères considèrent un contexte sans accès aux interfaces inférieures, les propriétés Pr_Int_1 et Pr_Int_2 ne peuvent être vérifiées dans cette architecture. Ainsi, les entrées sur les interfaces inférieures ne sont pas vérifiées. En effet, l'observation des messages envoyés entre implémentations n'étant pas possible dans ce contexte, cela reviendrait à vérifier la réception d'un message sans avoir d'information sur l'exécution de l'envoi de ce message. Les critères d'interopérabilité définis dans cette section se limitent donc à vérifier que les sorties exécutées sur les interfaces supérieures des implémentations sont correctes par rapport aux spécifications dans le contexte connu par les testeurs.

Définition 3.12 (Critère d'iop unilatérale basés interfaces supérieures iop_U^U)
$I_1 iop_U^U I_2 =_\Delta \forall \sigma_1 \in Traces(\Delta(S_1)/\Sigma_U), \sigma_1 = \sigma/\Sigma^{S_1}\ \exists \sigma \in Traces((S_1 \|_\mathcal{A} S_2)/\Sigma_U),$
$\Rightarrow Out((I_1 \|_\mathcal{A} I_2)/\Sigma_U^{S_1}, \sigma_1) \subseteq Out(\Delta(S_1)/\Sigma_U, \sigma_1)$

Définition 3.13 (Critère d'iop bilatérale basés interfaces supérieures iop_B^U)
$I_1 iop_B^U I_2 =_\Delta I_1 iop_U^U I_2 \wedge I_2 iop_U^U I_1$

Définition 3.14 (Critère d'iop globale basés interfaces supérieures iop_G^U)
$I_1 iop_G^U I_2 =_\Delta \forall \sigma \in Traces((S_1 \|_\mathcal{A} S_2)/\Sigma_U), Out((I_1 \|_\mathcal{A} I_2)/\Sigma_U, \sigma) \subseteq Out((S_1 \|_\mathcal{A} S_2)/\Sigma_U, \sigma)$

3.6.4 Quelques propriétés des différents critères d'interopérabilité

Les critères d'interopérabilité ne sont pas transitifs : quelque soit le critère considéré iop_X, $I_1\ iop_X\ I_2\ \wedge I_2\ iop_X\ I_3\ \not\Rightarrow\ I_1\ iop_X\ I_3$.

Compte-tenu des dépendances entre évènements sur les interfaces supérieures et inférieures, la vérification des critères d'interopérabilité inférieure et supérieure d'un contexte particulier ne signifie pas la vérification du critère d'interopérabilité totale correspondant. En effet :

- $I_1\ iop_U^U I_2\ \wedge\ I_1\ iop_U^L\ I_2\ \not\Rightarrow\ I_1\ iop_U\ I_2$ mais $I_1\ iop_U\ I_2\ \Rightarrow\ I_1\ iop_U^U\ I_2$ et $I_1\ iop_U\ I_2\ \Rightarrow\ I_1\ iop_U^L\ I_2$
- $I_1\ iop_B^U\ I_2\ \wedge\ I_1\ iop_B^L\ I_2\ \not\Rightarrow\ I_1\ iop_B\ I_2$ mais $I_1\ iop_B\ I_2\ \Rightarrow\ I_1\ iop_B^U\ I_2$ et $I_1\ iop_B\ I_2\ \Rightarrow\ I_1\ iop_B^L\ I_2$
- $I_1\ iop_G^U\ I_2\ \wedge\ I_1\ iop_G^L\ I_2\ \not\Rightarrow\ I_1\ iop_G\ I_2$ mais $I_1\ iop_G\ I_2\ \Rightarrow\ I_1\ iop_G^U\ I_2$ et $I_1\ iop_G\ I_2\ \Rightarrow\ I_1\ iop_G^L\ I_2$

Dans les contextes d'interopérabilité inférieure et supérieure, il peut être possible de définir d'autres critères d'interopérabilité pour tenter, via les dépendances causales entre évènements, de pallier à l'absence d'accès à une partie des interfaces. Cependant, ce type de critères ne peut s'appliquer que dans des cas où les spécifications prévoient des dépendances entre les évènements exécutés sur les deux types d'interfaces. En particulier, de tels critères peuvent être définis dans le cas où les spécifications alternent des évènements sur les deux interfaces : à chaque entrée sur une interface supérieure (resp. inférieure) doit succéder une sortie sur une interface inférieure (resp. supérieure). Ce type de spécifications peut correspondre à des protocoles qui transmettent un message provenant de la couche supérieure après une manipulation de ce message (évènements internes) tels que du chiffrement, de l'ajout de codes de détection d'erreurs ou de d'autres informations, etc. Dans le cas général, il est possible que plusieurs échanges aient lieu sur les interfaces inaccessibles avant que les testeurs observent un évènement, ce qui rend complexe voire impossible l'interprétation des dépendances entre évènements.

3.7 Un exemple d'application des critères d'interopérabilité

Dans cette section, nous illustrons les critères d'interopérabilité définis par une application sur des spécifications et implémentations.

Considérons les spécifications de la figure 3.2. Ces spécifications représentent un protocole de demande de connexion de type client/serveur. S_1 est le client et, S_2 et S_2' sont deux versions du serveur.

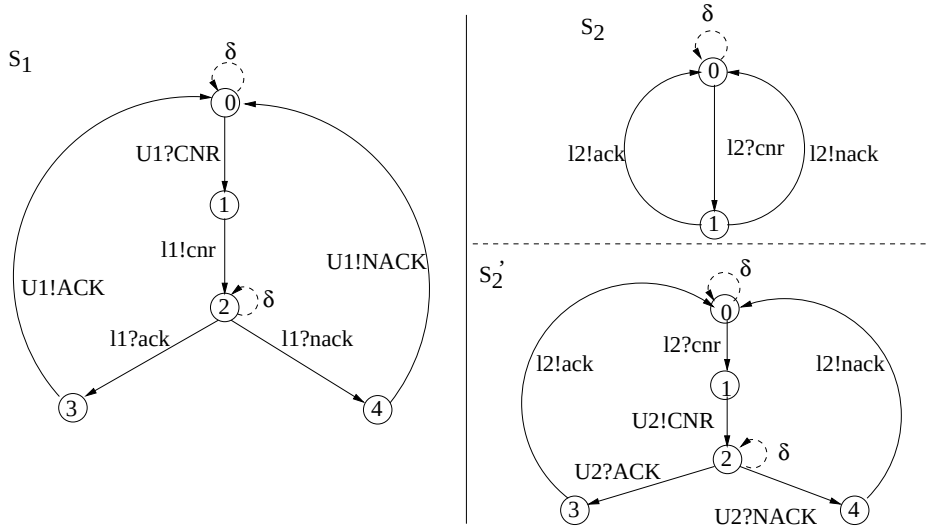


FIG. 3.2 – Spécifications S_1 et S_2

Description du protocole : S_1 reçoit une demande de connexion de sa couche supérieure (évènement $U1?CNR$). S_1 transmet cette demande à S_2 (envoi de la demande par S_1 : $l1!cnr$, réception par S_2 : $l2?cnr$). S_2 peut ensuite décider de répondre positivement (envoi de $l2!ack$) ou négativement (envoi de $l2!nack$). Notons qu'ici le choix pour S_2 d'accepter ou refuser la connexion se fait sur la base de calculs internes non-observables aux interfaces des systèmes implémentant S_2 (donc non-représentés sur le modèle de S_2). Cette réponse à la demande de connexion est reçue par S_1 (évènements $l1?ack$ ou $l1?nack$) qui la transmet à sa couche supérieure (évènements $U1?ACK$ ou $U1?NACK$).
La principale différence entre S_2 et S_2' se situe au niveau du choix pour la réponse à la demande de connexion. S_2 choisit entre ack et $nack$ en fonction d'éléments internes, tandis que S_2' transmet la demande à sa couche supérieure (évènement $U2!CNR$) qui répond positivement ($U2?ACK$) ou négativement ($U2?NACK$). C'est cette réponse de la couche supérieure qui est envoyée à S_1 par le serveur (message $l2!ack$ ou $l2!nack$).

La propriété d'iop-compatibilité est vérifiée pour cet exemple : on a $S_1\ iop\_comp\ S_2$ et $S_1\ iop\_comp\ S_2$.

Considérons maintenant les implémentations I_1, I_2 de la figure 3.3 et une implémentation I_3 telle que $I_3 = S_1$. Ce sont toutes les trois des implémentations de S_1. De même, prenons $I_6 = S_2$. Les implémentations I_4, I_5 (de la figure 3.3) et I_6 sont des implémentations de S_2. Nous n'avons pas défini d'implémentations de S_2'. Nous étudierons uniquement l'interopérabilité de I_1 et I_2 avec S_2'.
Avant d'appliquer les critères d'interopérabilité à ces implémentations, intéressons-nous à leur conformité :

conformité des implémentations au sens d'ioconf

- Pour les implémentations de la spécification S_1 : I_1 **ioconf** S_1, $\neg I_2$ **ioconf** S_1 (à cause de la sortie $U1!NACK$ après l'entrée $l1?ack$ qui n'est pas conforme à la spécification) et bien sûr, $I_3 = S_1$ **ioconf** S_1.
- Pour les implémentations de la spécification S_2 : I_4 **ioconf** S_2, I_5 **ioconf** S_2 et bien sûr, $I_6 = S_2$ **ioconf** S_2. Il n'y a en effet dans ces implémentations aucune sortie non décrite dans la spécification.

conformité des implémentations au sens d'ioco Les implémentations non ioconf-conformes ne sont pas non plus ioco-conformes. Par contre, certaines implémentations ioconf-conformes ne sont pas ioco-conformes.

- Pour les implémentations de la spécification S_1 : $\neg I_2$ **ioco** S_1 et, $I_3 = S_1$ **ioco** S_1, mais $\neg I_1$ **ioco** S_1 (à cause d'un blocage non-autorisé en cas d'envoi par le testeur à I_1 du message $l1?nack$ et de l'hypothèse du test de conformité qui considère les implémentations input-complètes).
- Pour les implémentations de la spécification S_2 : I_4 **ioco** S_2, I_5 **ioco** S_2 et $I_6 = S_2$ **ioco** S_2.

Appliquons maintenant les critères d'interopérabilité aux implémentations de la figure 3.3 basées sur les spécifications de la figure 3.2.

critères d'interopérabilité totale

- interaction entre I_1 et I_4 : $I_1\ iop_G\ I_4$ et $I_1\ iop_B\ I_4$ (car $I_1\ iop_U\ I_4$ et $I_4\ iop_U\ I_1$). En effet, les deux implémentations implémentent toutes les deux l'acceptation de connexion (et aucune n'implémente le refus de connexion).
- interaction entre I_2 et I_5 : $I_2\ iop_G\ I_5$ et $I_2\ iop_B\ I_5$ (car $I_2\ iop_U\ I_5$ et $I_5\ iop_U\ I_2$). L'erreur dans l'implémentation de I_2 se situe au niveau de la branche d'acceptation de connexion, mais I_5 n'implémente que le refus de connexion, donc I_2 n'exécutera jamais sa branche comportant une erreur en interagissant avec I_5.
- interaction entre I_1 et I_5 : $\neg\ I_1\ iop_G\ I_5$ et $\neg\ I_1\ iop_B\ I_5$ (car $\neg\ I_1\ iop_U\ I_5$ et $I_5\ iop_U\ I_1$). En effet, lors de l'interaction entre I_1 et I_5, I_1 n'est pas capable de traiter le message $nack$ qui lui est envoyé par I_5. Les testeurs observeront

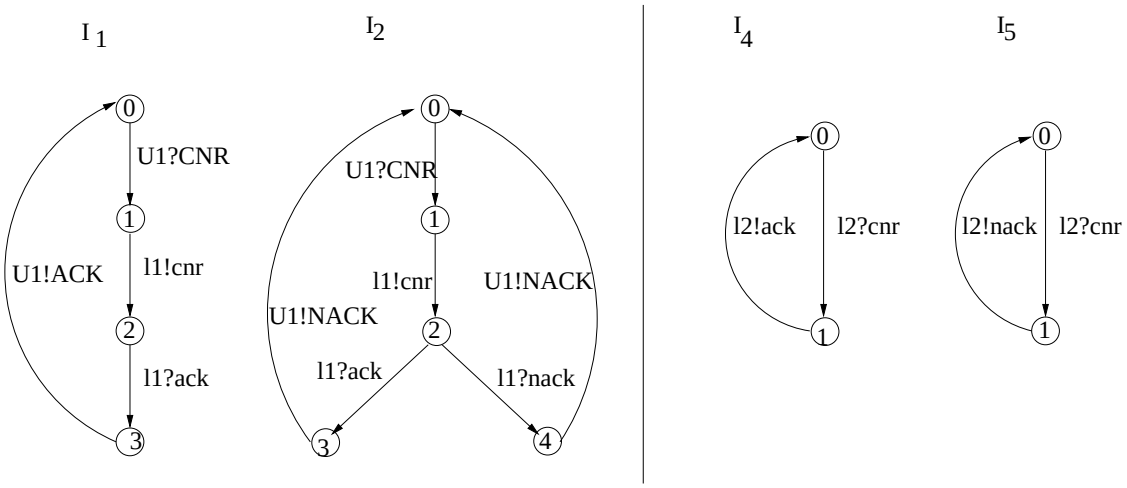


FIG. 3.3 – Implémentations I_1 et I_2 de S_1, et I_4 et I_5 de S_2

donc un blocage qui n'est pas dans l'ensemble des dépendances causales du message $nack$ dans la spécification S_1.
N.B. : les résultats pour l'interaction entre I_1 et $I_6 = S_2$ sont les mêmes que pour cette interaction.

– interaction entre I_2 et I_4 : $\neg\ I_2\ iop_G\ I_4$ et $\neg\ I_2\ iop_B\ I_4$ (car $\neg\ I_2\ iop_U\ I_4$ et $I_4\ iop_U\ I_2$). Cette non-interopérabilité est due à l'envoi par I_2 d'un message $NACK$ après réception d'un ack sur son interface inférieure.
Les résultats pour l'interaction entre I_2 et $I_6 = S_2$ sont les mêmes à cause du même envoi non-autorisé de $NACK$ par I_2.

critères d'interopérabilité inférieure Les implémentations de S_1 de la figure 3.3 sont toutes interopérables avec les implémentations de S_2 si on se base sur des critères d'interopérabilité inférieure. Comme $iop_X \Rightarrow iop_X^L$, nous nous intéressons ici aux interactions pour lesquelles les critères d'interopérabilité totale détectaient une situation de non-interopérabilité.

– Pour l'interaction entre I_2 et I_4, le message non prévu dans les spécifications est l'évènement $U1!NACK$ (après $l1?ack$), un message sur une interface supérieure à laquelle le contexte d'interopérabilité inférieure n'accède pas. Cette situation de non-interopérabilité n'est donc pas détectée par les critères d'interopérabilité inférieure.
– Pour l'interaction entre I_1 et I_5, la dépendance causale permettant de savoir si I_1 a reçu le message $nack$ envoyé par I_5 est le message ACK. Or cette sortie n'est pas observable car elle s'exécute sur une interface (supérieure) inaccessible par les testeurs dans cette architecture de test inférieure. La première sortie exécutable sur une interface accessible dans cette architecture est la sortie cnr. Mais cette sortie n'est exécutée qu'après l'évènement $U1?CNR$

qui, non seulement n'est ni contrôlable ni observable, mais surtout correspond à un deuxième demande de connexion. Ce qui signifie que pour observer une sortie dépendance causale de $nack$, il faudrait "redémarrer" le protocole de demande de connexion. L'évènement observé est donc un blocage, correct d'après le modèle projeté de la spécification S_1. Le comportement est donc considéré comme correct sur la base des critères d'interopérabilité inférieure.

critères d'interopérabilité supérieure Là aussi, si on se base sur des critères d'interopérabilité supérieure, les implémentations de S_1 (respectivement I_1, I_2 de la figure 3.3, et $I_3 = S_2$) sont interopérables avec les implémentations de S_2 (respectivement I_4, I_5 de la figure 3.3, et $I_6 = S_2$). Cependant, quelques cas de figure méritent des explications. En effet :

- Cas de l'interaction de I_2 et I_4 : les évènements connus dans un contexte supérieur sont $U1?CNR$ et $U1!NACK$ ce qui est correct d'après la spécification S_1. Il n'y a en effet dans un tel contexte aucun moyen de savoir si le message transmis par I_4 à I_2 est le message ack ou $nack$.
- Pour l'interaction entre I_1 et I_5, aucun message non prévu n'est envoyé (puisque cette interaction ne générera pas de sortie sur les interfaces supérieures), et le blocage des deux implémentations est autorisé dans la version projetée sur les interfaces supérieures de leurs spécifications.

On peut remarquer que dans nos exemples, les critères d'interopérabilité inférieure et supérieure ne détectent aucune situation de non-interopérabilité : c'est dû à la simplicité des spécifications prises en exemple. En effet, les évènements de S_1 alternent message sur l'interface supérieure et l'interface inférieure, tandis que S_2 n'exécute que des messages sur son interface inférieure.
Considérons maintenant l'interaction de I_1 et I_2 avec S_2' :

Interaction I_1 avec S_2' les critères d'interopérabilité vérifiés et non-vérifiés sont les mêmes que lors l'interaction entre I_1 et S_2.

Interaction I_2 avec S_2' les résultats sont les mêmes que pour l'interaction entre I_2 et S_2 pour les critères d'interopérabilité totale et inférieure, mais pas pour les critères d'interopérabilité supérieure.

- Critère d'interopérabilité globale supérieure : $\neg\, I_2\, iop_G^U\, S_2'$. En effet, $U2?ACK$ doit être suivi de $U1!ACK$ et non pas de $U1!NACK$.
- Critères d'interopérabilité unilatérale supérieure : localement à chaque implémentation, il n'y a pas de sortie (ou blocage) non décrit dans les spécifications. On a donc $I_2\, iop_U^U\, S_2'$ et $S_2'\, iop_U^U\, I_2$, d'où $I_2\, iop_B^U\, S_2'$. Par contre, l'utilisation des dépendances causales entre les couples d'évènements ($U2?ACK$, $l2!ack$), ($\mu = l2!ack$, $\bar{\mu} = l1?ack$) puis ($l1?ack$, $U1!ACK$) permettrait de détecter cette erreur. En effet, dans cette exemple de spécifications (S_1 et S_2'), contrairement à S_2, il y a alternance (et donc dépendance) entre évènements

sur chacune des interfaces.

Comparons les résultats des critères d'interopérabilité supérieure obtenus pour l'interopérabilité des implémentations de S_1 (implémentations I_1 et I_2 de la figure 3.3) avec S'_2, et les résultats de ces critères pour l'interopérabilité des implémentations de S_1 avec S_2. La principale différence entre ces deux exemples concerne la détection de la sortie $NACK$ non prévue sur l'interface $U1$ (après une entrée ack sur $l1$). Cette erreur n'est pas détectée avec les critères d'interopérabilité unilatéraux quand I_2 interagit avec S_2. Par contre, elle est détectée avec le critère d'interopérabilité globale supérieure quand I_2 interagit avec S'_2 (mais pas avec les critères d'interopérabilité supérieure bilatérale ou unilatérale). En effet, S'_2, contrairement à S_2, peut exécuter des évènements sur les interfaces supérieures : la réponse à la demande de connexion vient de la couche supérieure. Le calcul de $(S_1 \| _{\mathcal{A}} S'_2)/\Sigma_U$ ($\neq (S_1/\Sigma_U \| _{\mathcal{A}} S'_2/\Sigma_U)$ d'où la non-détection de l'erreur en contexte bilatéral supérieur) permet de connaître la dépendance entre les évènements $U2?ACK$ et $U1!ACK$. Ainsi, la présence d'évènements sur une interface supérieure dans la spécification S'_2 permet au critère d'interopérabilité globale supérieure de détecter des erreurs dans I_2 observées lors de l'interaction de I_2 et S'_2.

3.8 Comparaison entre critères d'interopérabilité

Le choix d'un critère d'interopérabilité sur lequel doivent se baser les tests se fait en fonction du contexte d'interopérabilité considéré, mais également en fonction des besoins en terme de détection de la non-interopérabilité. Ainsi, dans cette section, nous comparons les différents critères définis entre eux de façon à connaître leur pouvoir en terme de détection de la non-interopérabilité.

Soit $\mathcal{SBIR}$ l'ensemble des critères d'interopérabilité définis dans la section 3.6. La comparaison entre critères d'interopérabilité peut être formalisée par une relation d'équivalence de test et par une relation d'ordre entre critères, la relation $\sqsubseteq_{iop} \subseteq \mathcal{SBIR} \times \mathcal{SBIR}$.

Définition 3.15 $\forall\ iop_x, iop_y \in \mathcal{SBIR}$,

$$iop_x \sqsubseteq_{iop} iop_y =_{\Delta} \ \forall I_1, I_2 \in \mathcal{IOLTS},\ I_1\ iop_x\ I_2 \Rightarrow I_1\ iop_y\ I_2$$

Ainsi, $iop_x \sqsubseteq_{iop} iop_y$ signifie que des tests d'interopérabilité basés sur le critère d'interopérabilité iop_x détectent plus de situations de non-interopérabilité que des tests d'interopérabilité basés sur iop_y. Par extension, l'équivalence de test entre deux critères d'interopérabilité est représentée par la relation $\cong_{iop}$. Nous notons $iop_x \not\sim_{iop} iop_y$ pour indiquer que les critères iop_x et iop_y ne sont pas comparables. La figure 3.4 rassemble les comparaisons entre les différents critères d'interopérabilité (dans cette table, $\not\sim_{iop}$ est représenté par "-").

	iop_U^L	iop_B^L	iop_G^L	iop_U^U	iop_B^U	iop_G^U	iop_U	iop_B
iop_G	$\sqsubseteq_{iop}$	$\sqsubseteq_{iop}$	$\sqsubseteq_{iop}$	$\sqsubseteq_{iop}$	$\sqsubseteq_{iop}$	$\sqsubseteq_{iop}$	$\sqsubseteq_{iop}$	$\cong_{iop}$
iop_B	$\sqsubseteq_{iop}$	$\sqsubseteq_{iop}$	$\sqsubseteq_{iop}$	$\sqsubseteq_{iop}$	$\sqsubseteq_{iop}$	$\sqsubseteq_{iop}$	$\sqsubseteq_{iop}$	
iop_U	$\sqsubseteq_{iop}$	-	-	$\sqsubseteq_{iop}$	-	-		
iop_G^U	-	-	-	$\sqsubseteq_{iop}$	$\sqsubseteq_{iop}$			
iop_B^U	-	-	-	$\sqsubseteq_{iop}$				
iop_U^U	-	-	-					
iop_G^L	$\sqsubseteq_{iop}$	$\sqsubseteq_{iop}$						
iop_B^L	$\sqsubseteq_{iop}$							
iop_U^L								

FIG. 3.4 – Comparaison entre critères d'interopérabilité

La plupart des résultats décrits dans le tableau 3.4 sont obtenus par les observations suivantes :

- les critères d'interopérabilité inférieures et supérieures basent leur conclusion sur des observations obtenues sur des ensembles d'interfaces différentes. Leur détection de la non-interopérabilité n'est donc pas comparable.
- les critères d'interopérabilité totale ont un pouvoir de détection de la non-interopérabilité plus important que les critères d'interopérabilité inférieure ou supérieure car les conclusions sont basées sur des observations effectuées sur les deux types d'interfaces alors que les critères d'interopérabilité inférieure ou supérieure ne connaissent qu'une partie des évènements exécutés.
- de même, les critères d'interopérabilité globale et bilatérale ont un pouvoir de détection de la non-interopérabilité plus important que les critères d'interopérabilité unilatérale car les conclusions sont basées sur des observations effectuées sur les deux implémentations là où un critère d'interopérabilité unilatérale ne connaît que les évènements exécutés par une implémentation lors de l'interaction.

Cependant, pour l'équivalence entre critère d'interopérabilité bilatérale totale et critère d'interopérabilité globale totale (formalisée dans le théorème 3.1), une preuve formelle est nécessaire.

Théorème 3.1 *soit I_1 et I_2 deux implémentations,*

$$I_1 \, iop_G \, I_2 \Leftrightarrow I_1 \, iop_B \, I_2$$

La preuve de ce théorème nécessite tout d'abord la définition et la preuve des trois lemmes suivants (lemmes 3.1, 3.2 et 3.3).

Le lemme 3.1 dit que, lors d'une interaction asynchrone, les sorties qui peuvent être exécutées par le système global composé de deux entités M_1 et M_2 sont les mêmes que les sorties exécutées par chacun des deux systèmes M_1 et M_2. Sa preuve est basée sur la définition de l'interaction asynchrone, plus particulièrement sur le caractère asynchrone de la communication entre deux systèmes.

Lemme 3.1
Soit M_1 et M_2 deux IOLTS.
$\forall \sigma \in Traces(M_1 \|_{\mathcal{A}} M_2)$, $Out(M_1 \|_{\mathcal{A}} M_2, \sigma) = Out(\Delta(M_1), \sigma/\Sigma^{M_1}) \cup Out(\Delta(M_2), \sigma/\Sigma^{M_2})$

Preuve

1. Soit $(q_1, q_2, < f_1 >, < f_2 >) \in [(M_1 \|_{\mathcal{A}} M_2)\ after\ \sigma]$ et $a \in Out(M_1 \|_{\mathcal{A}} M_2, \sigma)$. D'après la définition de l'interaction asynchrone (cf. section 3.3.2), soit $a \in \Sigma^{M_1} \cup \{\delta(1)\}$ (c'est-à-dire. $a \in Out(\Delta(M_1), \sigma/\Sigma^{M_1})$), soit $a \in \Sigma^{M_2} \cup \{\delta(2)\}$ (c'est-à-dire. $a \in Out(\Delta(M_2), \sigma/\Sigma^{M_2})$).
   Ainsi, $Out(M_1 \|_{\delta} M_2, \sigma) \subseteq Out(\Delta(M_1), \sigma/\Sigma^{M_1}) \cup Out(\Delta(M_2), \sigma/\Sigma^{M_2})$.
2. Pour le sens $Out(\Delta(M_1), \sigma/\Sigma^{M_1}) \cup Out(\Delta(M_2), \sigma/\Sigma^{M_2}) \subseteq Out(M_1 \|_{\mathcal{A}} M_2, \sigma)$, intéressons-nous aux règles (3.1), (3.2), (3.3) et (3.4) de la définition de l'interaction asynchrone (définition 3.2 de la section 3.3.2). σ est une trace de l'interaction : il existe donc un état $(q_1, q_2, < f_1 >, < f_2 >)$ tel que $(q_1, q_2, < f_1 >, < f_2 >) \in [(M_1 \|_{\mathcal{A}} M_2)\ after\ \sigma]$. Soit $\alpha \in Out(\Delta(M_1), \sigma/\Sigma^{M_1}) \cup Out(\Delta(M_2), \sigma/\Sigma^{M_2})$. Si $\alpha \in Out(\Delta(M_1), \sigma/\Sigma^{M_1})$, alors $a \in Out(M_1 \|_{\mathcal{A}} M_2, \sigma)$ d'après les règles (3.1) et (3.3) de l'interaction asynchrone. Si $\alpha \in Out(\Delta(M_2), \sigma/\Sigma^{M_2})$, alors $a \in Out(M_1 \|_{\mathcal{A}} M_2, \sigma)$ d'après les règles (3.2) et (3.4) de l'interaction asynchrone. D'où $\alpha \in Out(M_1 \|_{\mathcal{A}} M_2, \sigma)$.

Conclusion : $Out(M_1 \|_{\mathcal{A}} M_2, \sigma) = Out(\Delta(M_1), \sigma/\Sigma^{M_1}) \cup Out(\Delta(M_2), \sigma/\Sigma^{M_2})$ □

Remarque: Ce lemme n'est valable que pour une interaction *asynchrone*. En effet, la preuve de ce lemme s'appuie sur la définition de l'interaction asynchrone qui, contrairement à l'interaction synchrone, dit qu'une sortie peut être exécutée même si le récepteur n'est "pas encore prêt".

De plus, ce lemme n'est plus valable non plus si on s'intéresse aux ensembles de sorties sur lesquels sont basés les critères d'interopérabilité inférieure et supérieure.
En effet, $(M_1 \|_{\mathcal{A}} M_2)/\Sigma_X \neq ((\Delta(M_1)/\Sigma_X^{M_1}) \|_{\mathcal{A}} (\Delta(M_2)/\Sigma_X^{M_2}))$,
d'où $Out((M_1 \|_{\mathcal{A}} M_2)/\Sigma_X, \sigma) \neq Out(\Delta(M_1)/\Sigma_X^{M_1}, \sigma/\Sigma^{M_1}) \cup Out(\Delta(M_2)/\Sigma_X^{M_1}, \sigma/\Sigma^{M_2})$. Cette différence vient du fait que projeter après avoir calculé l'interaction permet de garder des dépendances entre évènements sur différentes interfaces que l'on ne retrouve pas si on calcule l'interaction des IOLTS projetés.

Le lemme 3.2 dit que le modèle de l'interaction asynchrone de deux IOLTS M_1 et M_2 est égal au modèle de l'interaction asynchrone de la vue de l'exécution de ces IOLTS lors de leur interaction, c'est-à-dire à l'interaction de $(M_1\|_{\mathcal{A}}M_2)/\Sigma^{M_1}$ et $(M_2\|_{\mathcal{A}}M_1)/\Sigma^{M_2}$. La preuve de ce lemme est réalisée en étudiant les traces exécutables lors de ces interactions.

Lemme 3.2
Soit M_1 et M_2 deux IOLTS,

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

Preuve

1. Soit $\sigma_1 \in Traces((M_1\|_{\mathcal{A}}M_2)/\Sigma^{M_1})$, $\sigma_2 \in Traces((M_2\|_{\mathcal{A}}M_1)/\Sigma^{M_2})$, et $\sigma \in Traces(\ ((M_1\|_{\mathcal{A}}M_2)/\Sigma^{M_1})\|_{\mathcal{A}}((M_2\|_{\mathcal{A}}M_1)/\Sigma^{M_2}))$ tel que $\sigma/\Sigma^{M_1} = \sigma_1$ et $\sigma/\Sigma^{M_2} = \sigma_2$.
   On a alors : $\sigma_1 \in Traces(\Delta(M_1))$ et $\sigma_2 \in Traces(\Delta(M_2))$. Ainsi, comme $\sigma/\Sigma^{M_1} = \sigma_1$ et $\sigma/\Sigma^{M_2} = \sigma_2$, $\sigma \in Traces(M_1\|_{\mathcal{A}}M_2)$.
2. Soit $\sigma \in Traces(M_1\|_{\mathcal{A}}M_2)$ tel que σ peut être calculé comme l'interaction de deux traces $\sigma_1 \in Traces(\Delta(M_1))$ et $\sigma_2 \in Traces(\Delta(M_2))$. C'est-à-dire que σ est une trace de l'interaction de M_1 et M_2 décomposable en deux traces σ_1 et σ_2, exécutables respectivement sur chacun des deux IOLTS lors de l'interaction, noté $\sigma = \sigma_1\|_{\mathcal{A}}\sigma_2$.
   On a alors $\sigma_1 = \sigma/\Sigma^{M_1}$ et $\sigma_2 = \sigma/\Sigma^{M_2}$.
   Ainsi $\sigma_1 \in Traces((M_1\|_{\mathcal{A}}M_2/\Sigma^{M_1}))$, $\sigma_2 \in Traces((M_2\|_{\mathcal{A}}M_1/\Sigma^{M_2}))$
   et $\sigma = \sigma_1\|_{\mathcal{A}}\sigma_2 \in Traces(((M_1\|_{\mathcal{A}}M_2)/\Sigma^{M_1})\|_{\mathcal{A}}((M_2\|_{\mathcal{A}}M_1)/\Sigma^{M_2}))$.

Conclusion : $Traces((M_1\|_{\mathcal{A}}M_2)/\Sigma^{M_1})\|_{\mathcal{A}}((M_2\|_{\mathcal{A}}M_1)/\Sigma^{M_2})) = Traces(M_1\|_{\mathcal{A}}M_2)$ d'où $(M_1\|_{\mathcal{A}}M_2)/\Sigma^{M_1})\|_{\mathcal{A}}((M_2\|_{\mathcal{A}}M_1)/\Sigma^{M_2}) = M_1\|_{\mathcal{A}}M_2$
□

Le lemme 3.3 dit que les sorties observées sur M_1 lors de son interaction avec M_2 après une trace donnée sont des sorties qui pourraient être observées sur M_1 prise isolement après la même trace. La preuve de ce lemme est évidente si on remarque que l'IOLTS $(M_1\|_{\mathcal{A}}M_2)/\Sigma^{M_1}$ est un IOLTS composé d'évènements de $\Sigma^{(M_1\|_{\mathcal{A}}M_2)}/\Sigma^{M_1}$, évènements qui sont donc des évènements de Σ^{M_1}.

Lemme 3.3 *Soit M_1 et M_2 deux IOLTS. Soit $\sigma \in Traces(M_1\|_{\mathcal{A}}M_2)$ tel que $\sigma_1 = \sigma/\Sigma^{M_1} \in Traces(\Delta(M_1))$.*

$$Out((M_1\|_{\mathcal{A}}M_2)/\Sigma^{M_1}, \sigma_1) \subseteq Out(\Delta(M_1), \sigma_1)$$

Grâce à ces lemmes, il est possible de prouver le théorème 3.1. Dans la preuve ci-dessous, nous nous concentrons sur l'équivalence entre les premières parties des critères d'interopérabilité globale et bilatérale (premières parties de iop_B et iop_G). En

effet, les parties décrivant la vérification des entrées sont identiques dans les deux critères : le point de vue pour la vérification des entrées est un point de vue bilatéral dans les deux critères. Dans la preuve ci-dessous, nous utilisons donc les notations iop_B et iop_G pour les parties vérification des sorties et blocages de ces deux critères.

Preuve Pour prouver le théorème 3.1, prouvons que $I_1\, iop_B\, I_2 \Rightarrow I_1\, iop_G\, I_2$, puis que $I_1\, iop_G\, I_2 \Rightarrow I_1\, iop_B\, I_2$.

1. Prouvons d'abord que $I_1\, iop_B\, I_2 \Rightarrow I_1\, iop_G\, I_2$.
   Soit $\sigma \in Traces(S_1 \|_{\mathcal{A}} S_2)$, $\sigma_1 \in Traces(\Delta(S_1))$ tel que $\sigma_1 = \sigma/\Sigma^{S_1}$, $\sigma_2 \in Traces(\ \Delta(S_2))$ tel que $\sigma_2 = \sigma/\Sigma^{S_2}$.
   D'après le critère d'interopérabilité iop_B, $Out((I_1 \|_{\mathcal{A}} I_2)/\Sigma^{S_1}, \sigma_1) \subseteq Out(\Delta(S_1),$ $\sigma_1)$ et $Out((I_2 \|_{\mathcal{A}} I_1)/\Sigma^{S_2}, \sigma_2) \subseteq Out(\Delta(S_2), \sigma_2)$.
   On a donc : $Out((I_1 \|_{\mathcal{A}} I_2)/\Sigma^{S_1}, \sigma_1) \cup Out((I_2 \|_{\mathcal{A}} I_1)/\Sigma^{S_2}, \sigma_2) \subseteq Out(\Delta(S_1), \sigma_1) \cup$ $Out(\Delta(S_2), \sigma_2)$
   D'après le lemme 3.1, $Out[((I_1 \|_{\mathcal{A}} I_2)/\Sigma^{S_1} \|_{\mathcal{A}} (I_2 \|_{\mathcal{A}} I_1)/\Sigma^{S_2}), \sigma] \subseteq Out(S_1 \|_{\mathcal{A}} S_2, \sigma)$.
   D'après le lemme 3.2, $Out(I_1 \|_{\mathcal{A}} I_2, \sigma) \subseteq Out(S_1 \|_{\mathcal{A}} S_2, \sigma)$. Ce qui signifie que $I_1\, iop_G\, I_2$.
2. Prouvons maintenant que $I_1\, iop_G\, I_2 \Rightarrow I_1\, iop_B\, I_2$.
   Soit I_1, I_2, S_1, S_2 tel que $I_1\, iop_G\, I_2$.
   Soit $\sigma_1 \in Traces(\Delta(S_1))$ tel que $\sigma_1 = \sigma/\Sigma^{S_1}$ avec $\sigma \in Traces(S_1 \|_{\mathcal{A}} S_2)$.
   D'après la définition de $I_1\, iop_G\, I_2$, on a : $Out(I_1 \|_{\mathcal{A}} I_2, \sigma) \subseteq Out(S_1 \|_{\mathcal{A}} S_2, \sigma)$.
   Après projection sur Σ^{S_1} : $Out((I_1 \|_{\mathcal{A}} I_2)/\Sigma^{S_1}, \sigma_1) \subseteq Out(\ (S_1 \|_{\mathcal{A}} S_2)\ /\Sigma^{S_1}, \sigma_1)$.
   D'après le lemme 3.3, $Out((I_1 \|_{\mathcal{A}} I_2)/\Sigma^{S_1}, \sigma_1) \subseteq Out(\Delta(S_1), \sigma_1)$, c'est-à-dire $I_1\, iop_U\, I_2$. Et comme le critère d'interopérabilité iop_G est symétrique, on obtient de même $I_1\, iop_G\, I_2 \Rightarrow Out((I_2 \|_{\mathcal{A}} I_1)/\Sigma^{S_2}, \sigma_2) \subseteq Out(\Delta(S_2), \sigma_2)$, c'est-à-dire $I_2 iop_U I_1$.
   Ce qui signifie qu'on a $I_1\, iop_G\, I_2 \Rightarrow I_1\, iop_B\, I_2$.

Conclusion : $I_1\, iop_G\, I_2 \Rightarrow I_1\, iop_B\, I_2$ et $I_1\, iop_B\, I_2 \Rightarrow I_1\, iop_G\, I_2$ donc $I_1\, iop_G\, I_2 \Leftrightarrow I_1\, iop_B\, I_2$. □

Nous avons ainsi prouvé l'équivalence, en terme de détection de la non-interopérabilité, du critère d'interopérabilité globale totale et du critère d'interopérabilité bilatérale totale. Or, on peut remarquer que le critère d'interopérabilité globale totale est basée sur un modèle de l'interaction asynchrone des spécifications. Le calcul de cette interaction est généralement la cause du problème d'explosion combinatoire rencontré lors des approches classiques de génération de test d'interopérabilité. De plus, lorsque le test d'interopérabilité est réalisé de façon distribuée, l'approche globale nécessite l'écriture souvent complexe et source d'erreurs de procédures de coordination entre les testeurs de façon à distribuer un cas de test global. Le critère d'interopérabilité bilatérale totale, par contre, correspond à une approche distribuée sans synchronisa-

tion entre les testeurs et ne s'appuie pas sur un modèle de l'interaction asynchrone des spécifications. Ainsi, il peut être intéressant de baser la génération de cas de test sur ce critère d'interopérabilité bilatérale totale de façon à éviter le problème d'explosion combinatoire et l'écriture de procédures de coordination. C'est cette piste qui est étudiée dans la génération de test décrite dans le chapitre 4.

Notons également qu'avant toute activité de test, les testeurs doivent se mettre d'accord sur les objectifs des tests à réaliser, et donc, dans le cas du test d'interopérabilité, sur le critère d'interopérabilité servant de base aux tests. Ce choix d'un critère d'interopérabilité nécessite la prise en compte de plusieurs paramètres sur les systèmes testés et les besoins de test. L'objectif de la section suivante (section 3.9) est ainsi de donner des éléments pour ce choix.

3.9 Éléments pour le choix d'un critère d'interopérabilité pour le test

Nous avons défini neuf critères d'interopérabilité basés sur les différents contextes de test d'interopérabilité que l'on peut rencontrer en pratique (cf. sections 3.6 et 3.8). Nous avons également comparé ces critères en terme de détection de la non-interopérabilité et prouvé l'équivalence du critère d'interopérabilité bilatérale et du critère d'interopérabilité globale. Dans cette section, nous nous intéressons au choix d'un critère d'interopérabilité utilisé comme base pour la génération de tests d'interopérabilité.

Les paramètres à prendre en compte pour le choix du critère d'interopérabilité pour la génération de tests sont les suivants :

Architecture de test/accès aux interfaces des implémentations L'architecture de test d'interopérabilité, qui détermine l'accès aux interfaces des implémentations en interaction, est le premier critère à prendre pour compte pour choisir le critère de l'interopérabilité considéré lors de la génération de tests. En effet, le critère d'interopérabilité choisi ne doit pas considérer d'évènements à exécuter sur des interfaces inaccessibles lors des tests. L'architecture de test d'interopérabilité réduit ainsi le nombre de critères d'interopérabilité utilisables.

Par exemple, une architecture de test d'interopérabilité supérieure (resp. inférieure) ne peut exécuter que des tests basés sur un critère d'interopérabilité supérieure (resp. inférieure). Par contre, pour une architecture de test d'interopérabilité totale, des tests d'interopérabilité dérivés à partir de critères d'interopérabilité inférieure, supérieure ou totale peuvent être utilisés.

Dans un contexte de test d'interopérabilité avec accès à toutes les interfaces du systèmes, le choix d'un critère d'interopérabilité globale ou bilatérale peut se faire en fonction de la capacité de synchronisation des testeurs. Par exemple, si la synchronisation entre testeurs risque de ralentir les tests (à cause du temps

de transmission entre testeurs ou de la complexité de la procédure de synchronisation), il peut être plus intéressant d'utiliser des tests dérivés à partir d'un critère d'interopérabilité bilatérale. L'équivalence entre critère d'interopérabilité globale et bilatérale peut en particulier être utilisée pour dériver des tests d'interopérabilité ne nécessitant pas de procédure de coordination entre testeurs.

Niveau d'observation/contrôle souhaité ou exigé En fonction du type de protocole ou des tests déjà réalisés sur les implémentations, les besoins en terme de détection de la non-interopérabilité peuvent varier. La table 3.4 qui compare les différents critères en terme de détection de la non-interopérabilité peut alors servir de référence pour ce choix. Notons cependant que les critères d'interopérabilité basés sur une observation partielle des interfaces des implémentations détectent moins de situations de non-interopérabilité que les critères d'interopérabilité totale. Leur usage devrait donc être limité aux contextes les nécessitant : cf. section 2.5.3 du chapitre 2 pour des exemples de contextes/protocoles imposant l'utilisation d'architectures et donc de critères basés interfaces inférieures ou basés interfaces supérieures.

Complexité des tests et de la génération de test La taille des objets manipulés lors de la génération de test (spécifications des systèmes à tester, cas de test, etc) peut également avoir une influence dans le choix de la définition de l'interopérabilité considérée. Par exemple, il peut être impossible de dériver des tests d'interopérabilité basés sur un critère d'interopérabilité globale à cause de l'explosion combinatoire du nombre d'états que peut engendrer cette dérivation. Dans ce cas, un critère permettant d'éviter cette explosion combinatoire (critère d'interopérabilité bilatérale) serait préféré pour la génération de test.

Dans un contexte général où toutes les interfaces des deux implémentations sont accessibles par le testeur, il peut être intéressant de s'appuyer sur un critère d'interopérabilité bilatérale. En effet, ce choix évite la construction de l'interaction des spécifications qui peut causer une explosion du nombre d'états, mais également la construction de procédures de coordination complexes et sources d'erreurs. De plus, ce critère d'interopérabilité a été prouvé équivalent en terme de détection de la non-interopérabilité au critère d'interopérabilité globale. Il est donc intéressant de vérifier cette équivalence en pratique. Ce sont ces pistes que nous avons explorées dans notre approche de génération de tests d'interopérabilité basée sur des définitions formelles. Cette approche est décrite dans le chapitre 4.

Chapitre 4

Génération de tests d'interopérabilité : cas de deux implémentations

4.1 Introduction

Dans ce chapitre, nous nous intéressons à la génération de tests d'interopérabilité qui permet de définir automatiquement les cas de test nécessaires au test de l'interopérabilité de deux implémentations. Cette génération automatique s'appuie sur un modèle des spécifications : les implémentations dont on veut tester l'interopérabilité dérivent de ces spécifications. De plus, comme cela est généralement le cas en pratique, il est possible de baser cette génération de tests sur des *objectifs de test d'interopérabilité*. Ces objectifs de test servent à décrire les propriétés sur lesquelles doivent se focaliser un cas de test ou une suite de tests d'interopérabilité. Les tests générés sont ensuite appliqués sur le SUT (pour System Under Test) composé de deux IUTs (pour Implementation Under Test) interconnectées. Cette exécution renvoie un verdict. Ce verdict est rendu à la fois par rapport à l'objectif de test défini et par rapport à la définition de l'interopérabilité considérée pour la génération de test.

Dans le chapitre précédent, nous avons défini différents critères d'interopérabilité. Un critère d'interopérabilité définit formellement les propriétés que doivent satisfaire deux implémentations pour être considérées interopérables. Dans ce chapitre, nous nous intéressons aux méthodes de génération de test d'interopérabilité qui peuvent être développées sur la base de ces définitions formelles. L'objectif est donc de définir des algorithmes permettant de générer automatiquement des cas de test d'interopérabilité à partir de deux spécifications (à partir desquelles les IUTs ont été développées) et d'un objectif de test (cf. figure 4.1).

En fonction de l'architecture de test d'interopérabilité et du critère d'interopérabilité sélectionnés, différentes approches peuvent être appliquées pour la génération de

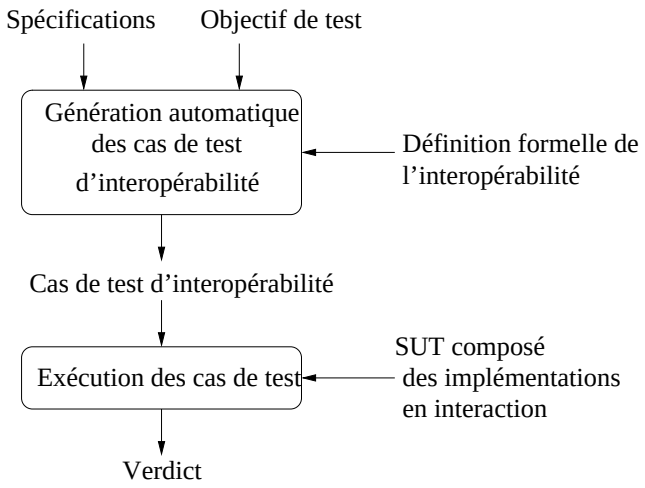


FIG. 4.1 – Génération automatique de cas de test d'interopérabilité

test. Dans ce chapitre, nous nous intéressons plus particulièrement à la comparaison de deux approches dites globale et bilatérale. Ces approches sont basées sur des critères d'interopérabilité considérant un accès à toutes les interfaces des implémentations mais avec des points de vue différents : critère d'interopérabilité bilatérale totale et critère d'interopérabilité globale totale. En effet, ces deux critères d'interopérabilité sont équivalents en terme de détection de la non-interopérabilité (voir théorème 3.1 section 3.8).

Le contexte le plus généralement considéré en pratique correspond à l'architecture de test d'interopérabilité globale totale. Le problème majeur de la génération de tests d'interopérabilité dans un tel contexte est l'explosion combinatoire du nombre d'états : la manipulation des entités considérées lors de la génération de test peut conduire à stocker un nombre d'états (générés pour le cas de test ou créés temporairement) trop important par rapport à la capacité des systèmes sur lesquels sont exécutés les algorithmes de génération. Ce problème d'explosion du nombre d'états est dû à l'étape de recherche des chemins correspondant à l'objectif de test dans l'interaction des spécifications (même construite à-la-volée ou seulement partiellement). Comme les critères d'interopérabilité bilatérale totale et globale totale sont équivalents en terme de détection de la non-interopérabilité, nous avons comparé les approches globale (totale) et bilatérale (totale) de génération de test d'interopérabilité. En effet, contrairement à l'approche globale, l'approche bilatérale ne se base pas sur un modèle de l'interaction des spécifications. Or, la construction de cette interaction est généralement la cause du problème d'explosion du nombre d'états.

Nous avons ainsi développé une méthode de génération de test d'interopérabilité qui, basée sur le critère d'interopérabilité bilatérale totale, permet d'éviter l'explosion

combinatoire du nombre d'états. Nous avons également complété cette méthode par un algorithme de calcul des dépendances causales entre entrées et sorties. Cet algorithme permet de vérifier la prise en compte effective des messages émis par l'IUT homologue (cf. propriété Pr_Int_1 définie dans la section 3.5.1 du chapitre 3). Ces algorithmes ont été implémentés grâce aux bibliothèques de la boîte à outils Cadp [GLM01]. Ils ont ainsi pu être appliqués sur un exemple de protocole de demande de connexion. Cette application a permis de vérifier à la fois que l'approche bilatérale permettait d'éviter l'explosion du nombre d'états rencontrée avec une approche globale, et l'équivalence de ces deux approches en terme de verdicts.

La décomposition de ce chapitre est la suivante. La section 4.2 récapitule les définitions des objets (objectifs de test, cas de test, verdicts) impliqués dans l'activité de test d'interopérabilité (adaptées des définitions plus générales de la section 2.2.2), et les modèles formels associés aux objets. La section 4.3 est consacrée à l'approche globale de génération de tests d'interopérabilité. Nous décrivons cette approche, basée sur le critère d'interopérabilité globale totale, et la comparons avec les approches habituellement utilisées pour le test d'interopérabilité. La section 4.4 porte ensuite sur l'approche bilatérale de génération de tests d'interopérabilité. Nous décrivons les différents étapes de cette approche et les algorithmes correspondants. Nous définissons également les conditions sous lesquelles un outil de génération de test de conformité peut être utilisé dans cette approche pour générer une partie du cas de test. Finalement, nous nous intéressons à la complexité de cette approche. La section 4.5 porte ensuite sur l'algorithme calculant un ensemble de sorties dépendances causales d'une entrée de façon à compléter, lors d'une des étapes de l'approche bilatérale de génération de test, les cas de test d'interopérabilité. La section 4.6 définit ainsi une méthode complète de génération de cas de test d'interopérabilité à partir de l'approche bilatérale complétée par la vérification des entrées grâce à l'algorithme de dépendances causales. Dans la section 4.7, nous étudions le lien entre les cas de test générés par cette méthode et l'approche distribuée du test d'interopérabilité. Nous appliquons ensuite cette méthode et l'approche globale à des exemples de spécifications dans la section 4.8 et comparons les résultats des deux approches. La section 4.9 décrit une amélioration de la méthode proposée et la section 4.10 s'intéresse à l'adaptation possible de cette méthode pour des architectures de test d'interopérabilité inférieure et supérieure. Finalement, la conclusion sur ce chapitre est dans la section 4.11.

4.2 Activité de test d'interopérabilité : définitions

Dans la section 2.2.2 du chapitre 2, nous avons donné la définition d'objets impliqués dans l'activité de test tels que les objectifs de test, les cas de test ou les verdicts. Dans cette section, nous adaptons (si nécessaire) les définitions de ces notions au

cadre du test d'interopérabilité. Nous définissons également les modèles formels des objectifs de test et cas de test adaptés au contexte du test d'interopérabilité de deux implémentations.

4.2.1 Objectif de test d'interopérabilité

En pratique, les tests d'interopérabilité sont dérivés à partir d'objectifs de test. Un objectif de test est utilisé pour décrire les propriétés que doit vérifier une suite de tests particulière. Lorsque les tests d'interopérabilité sont dérivés manuellement, un objectif de test d'interopérabilité est une description informelle du comportement à tester. Généralement, c'est une séquence incomplète d'actions qui doivent être exécutées par les implémentations lors de leur interaction.

Différents modèles, tels que la logique temporelle, les algèbres de processus, les MSCs (Messages Sequences Charts [GHN93]) ou des modèles dérivés des IOLTS [JJ05], peuvent être utilisés pour décrire les objectifs de test. Un modèle formel, dérivé des IOLTS, a été défini pour les objectifs de test utilisés en test de conformité dans le cadre de la réalisation de l'outil de génération de tests de conformité TGV [JJ05, FJJV97]. Le modèle formel de représentation d'un objectif de test d'interopérabilité (ou iop-TP pour interoperability test purpose) utilisé dans cette thèse est une adaptation pour l'interopérabilité de ce modèle. En fonction des besoins de nos algorithmes, nous pourrons également considérer des versions simplifiées de ce modèle.

Définition 4.1 (iop-TP)
Un iop-TP (objectif de test d'interopérabilité) iop-TP$= (Q^{iop-TP}, \Sigma^{iop-TP}, \Delta^{iop-TP}, q_0^{iop-TP})$ *est représenté par un IOLTS déterministe et complet équipé d'états-puits servant à sélectionner le comportement ciblé appelés* $Accept^{iop-TP}$ *et* $Refuse^{iop-TP}$.
L'alphabet des évènements d'un iop-TP est l'union des alphabets des évènements des spécifications considérées par le test d'interopérabilité, c'est-à-dire que $\Sigma^{iop-TP} = \Sigma^{S_1} \cup \Sigma^{S_2}$ *avec* $\Sigma_O^{iop-TP} = \Sigma_O^{S_1} \cup \Sigma_O^{S_2}$ *et* $\Sigma_I^{iop-TP} = \Sigma_I^{S_1} \cup \Sigma_I^{S_2}$.
iop-TP est un IOLTS complet signifie que, dans chaque état de iop-TP, tous les évènements autorisés dans les spécifications sont possibles : $\forall q \in Q^{iop-TP}$, $\forall a \in \Sigma^{iop-TP}$, $q \xrightarrow{a}_{\text{iop-TP}}$, *et, pour chaque état-puit* $q \in Q^{iop-TP}$, *il existe une boucle pour chaque évènement :* $\forall\, a \in \Sigma^{iop-TP}$, $q \xrightarrow{a}_{\text{iop-TP}} q$.

La principale différence entre un objectif de test d'interopérabilité et un objectif de test de conformité est dans l'ensemble des évènements possibles pour l'objectif de test. Tout d'abord, un objectif de test d'interopérabilité peut contenir des évènements des deux spécifications sur lesquelles sont basées les implémentations dont on veut tester l'interopérabilité. De plus, l'un des objectifs de l'interopérabilité étant de vérifier la réception de messages en provenance de l'implémentation en interaction, il est possible

qu'un objectif de test d'interopérabilité se termine par une entrée. Ceci sera alors vérifié en observant les sorties qui sont des dépendances causales de cette entrée.

4.2.2 Cas de test d'interopérabilité

4.2.2.1 Définition

Dans le cadre du test de conformité, un cas de test pouvait contenir deux types d'évènements : l'envoi de stimuli par le testeur et la réception de messages en provenance de l'implémentation.
Dans un cas de test d'interopérabilité (iop-TC pour interoperability test case), les types d'évènements possibles diffèrent en fonction de l'interface sur laquelle l'exécution a lieu. Trois types d'évènements peuvent ainsi être distingués :

- envoi par le testeur de stimuli vers les interfaces supérieures des implémentations
- réception de messages en provenance des interfaces supérieures des implémentations correspondant à la réalisation du service par les implémentations
- observation des évènements sur les interfaces inférieures des implémentations : c'est l'ensemble des évènements notés $?(\mu)$. Dans un évènement $?(\mu)$ d'un cas de test d'interopérabilité, l'évènement à observer μ peut être aussi bien une sortie (ou blocage) qu'une entrée. En effet, bien que les entrées ne soit pas observables par les testeurs, elles doivent être décrites dans les cas de test d'interopérabilité. Leur "observation" est faite en deux étapes. La première est l'observation de la transmission du message vers l'interface inférieure considérée. La deuxième est la conclusion sur l'exécution effective à l'aide des sorties qui sont des dépendances causales de l'évènement d'entrée.

Définition 4.2 (T-IOLTS)
Un T-IOLTS iop-TC modélisant un cas de test d'interopérabilité est un IOLTS étendu tel que iop-TC $= (Q^{iop-TC}, \Sigma^{iop-TC}, \Delta^{iop-TC}, q_0^{iop-TC})$ *avec :*

- Q^{iop-TC} *est l'ensemble (fini) des états du cas de test.* $\{PASS, FAIL, INC\} \subseteq Q^{iop-TC}$ *sont des états-puits représentant les verdicts du cas de test d'interopérabilité.* $q_0^{iop-TC} \in Q^{iop-TC}$ *est l'état initial du cas de test d'interopérabilité.*
- $\Sigma^{iop-TC} \subseteq \{\mu | \bar{\mu} \in \Sigma_U^{S_1} \cup \Sigma_U^{S_2}\} \cup \{?(\mu) | \mu \in \Sigma_L^{S_1} \cup \Sigma_L^{S_2}\}$ *est l'ensemble des évènements exécutables par le cas de test d'interopérabilité. L'ensemble des évènements* μ *tels que* $\bar{\mu} \in \Sigma_U^{S_1} \cup \Sigma_U^{S_2}\}$ *est l'ensemble des évènements exécutables par les testeurs connectés aux interfaces supérieures des implémentations. L'ensemble des évènements de type* $?(\mu)$ *tels que* $\mu \in \Sigma_L^{S_1} \cup \Sigma_L^{S_2}$ *est l'ensemble des observations possibles sur les interfaces inférieures des implémentations.*
- Δ^{iop-TC} *est la relation de transition du cas de test d'interopérabilité.*

Un cas de test iop-TC est tel que

1. *de tout état du cas de test d'interopérabilité, un verdict doit être accessible :* $\forall q \in Q^{iop-TC}$, $\exists \sigma$ *tel pour* $q' \in \{PASS, FAIL, INC\}$, $q \xrightarrow{\sigma}_{iop-TC} q'$.

2. un cas de test d'interopérabilité doit être contrôlable (cf. définition 4.3).

4.2.2.2 Propriétés

Dans cette section, nous nous intéressons aux propriétés sur les cas de test d'interopérabilité.

Un cas de test iop-TC est non biaisé par rapport aux spécifications S_1 et S_2 et un critère d'interopérabilité iop_X si et seulement si : $\forall I_1, I_2$ deux IOLTS, $\exists \sigma_1 \in Trace(I_1), \sigma_2 \in Trace(I_2)$ telles que $\sigma \in Trace(I_1 \|_{\mathcal{A}} I_2)$ et $\sigma_1 = \sigma/\Sigma^{I_1}$ et $\sigma_2 = \sigma/\Sigma^{I_2}$, si le SUT exécute σ lors de l'application de iop-TC et que le verdict $Fail$ est renvoyé, alors on a $\neg I_1\ iop_X\ I_2$.

Notons également que les transitions menant à l'état $Fail$ ne sont pas toujours explicitées dans la représentation du cas de test d'interopérabilité. Ces transitions peuvent être ajoutées en complétant l'iop-TC par des transitions de label $otherwise$ et menant à l'état $Fail$. Dans le contexte du test d'interopérabilité, le label $otherwise$ représente toutes les sorties et blocages des implémentations non décrits dans l'état courant, c'est-à-dire des réceptions du testeur et des observations de sorties sur les interfaces inférieures des implémentations.

Intéressons-nous maintenant à la propriété de contrôlabilité des cas de test d'interopérabilité. Dans le contexte du test de conformité, un cas de test est contrôlable si le test ne donne jamais à choisir entre deux sorties ou entre une entrée et une sortie, c'est-à-dire formellement, si on considère un cas de test de conformité conf-TC : $\forall q \in Q^{conf-TC}, \forall a \in \Sigma_O^{conf-TC}, q \xrightarrow{a}_{conf-TC} \Rightarrow \forall b \neq a, q \overset{b}{\not\longrightarrow}_{conf-TC}$. La différence dans le contexte d'interopérabilité provient du fait que les cas de test d'interopérabilité permettent trois type d'évènements : envois du testeur, réception de messages des interfaces supérieures des implémentations, et observation des évènements de l'interaction.

Définition 4.3 (Contrôlabilité d'un cas de test d'interopérabilité)
Soit iop-TC$= (Q^{iop-TC}, \Sigma^{iop-TC}, \Delta^{iop-TC}, q_0^{iop-TC})$ *un cas de test d'interopérabilité tel que* $\Sigma^{iop-TC} = \Sigma_O^{iop-TC} \cup \Sigma_I^{iop-TC} \cup \Sigma_{Obs}^{iop-TC}$ *où* Σ_O^{iop-TC} *représente l'ensemble des sorties du testeur,* Σ_I^{iop-TC} *l'ensemble des réceptions du testeur et* Σ_{Obs}^{iop-TC} *l'ensemble des observations.*
iop-TC est contrôlable si et seulement si le test ne donne jamais à choisir entre une sortie du testeur et un autre évènement : $\forall q \in Q^{iop-TC}, \forall a \in \Sigma_O^{iop-TC}, q \xrightarrow{a}_{iop-TC} \Rightarrow \forall b \neq a, q \overset{b}{\not\longrightarrow}_{iop-TC}$.

4.2.3 Verdicts d'interopérabilité

Les verdicts d'interopérabilité (ou iop-verdicts) sont les mêmes que les verdicts de test de conformité. Dans le modèle du cas de test d'interopérabilité, ces verdicts sont représentés par des états-puits. La sémantique de ces verdicts est la suivante :

PASS Le verdict PASS signifie que le comportement du SUT (composé des deux IUTs en interaction), observé pendant le test, satisfait l'objectif de test d'interopérabilité et est valide par rapport aux spécifications considérées et au critère d'interopérabilité considéré.

FAIL Le verdict FAIL signifie que le résultat de test observé ne correspond pas à ce qui est prévu dans la ou dans les spécifications considérées. Au moins une erreur a été observée lors de l'interaction des implémentations.

INCONCLUSIVE Le verdict INCONCLUSIVE ou "non concluant" signifie que ni le verdict PASS, ni le verdict FAIL ne peuvent être rendus. En effet, comme il est impossible de contrôler totalement le SUT, le comportement observé peut à la fois être prévu dans les spécifications et ne pas correspondre à l'objectif de test visé par le cas de test. Ce verdict est rendu par exemple quand une des spécifications prévoit le choix entre deux évènements et que l'évènement exécuté par l'implémentation correspondante est correct mais ne correspond pas à l'objectif de test.

4.3 Génération de tests d'interopérabilité basée sur le critère d'interopérabilité globale

Dans cette section, nous nous intéressons à la génération de tests d'interopérabilité basée sur une vue globale du système composé des deux implémentations en interaction. Cette approche de la génération de test, appelée approche globale, est basée sur une définition de l'interopérabilité qui correspond au critère d'interopérabilité globale totale.

4.3.1 Approches classiques pour la génération de tests d'interopérabilité

Le critère d'interopérabilité globale totale (et plus particulièrement sa partie vérification des sorties et blocages : propriétés Pr_Int_2 et Pr_Serv de la section 3.5.1) correspond à la définition sur laquelle s'appuie généralement la dérivation (manuelle ou automatique) de tests d'interopérabilité. Par exemple, la dérivation de test manuelle (cf. section 2.5.1) recherche les exécutions possibles permettant d'observer dans les spécifications un objectif de test d'interopérabilité. De même, les méthodes décrites

dans [KC00, BCKZ02, HLSG04] (cf. section 2.5.5) sont basées sur une vue globale du système composé des deux implémentations en interaction et sur le graphe modélisant l'interaction des spécifications. Dans la plupart des cas, ce graphe, appelé graphe d'accessibilité (reachability graph), n'est cependant construit que de façon incomplète en fonction de l'objectif de test.

Dans le critère d'interopérabilité globale, la vérification des sorties et blocages est basée sur la comparaison entre les sorties et blocages observés sur les interfaces du SUT composé des deux IUTs et les sorties et blocages prévus dans l'interaction des spécifications dans la même situation. "Dans la même situation" signifie que l'exécution du SUT précédant la sortie (ou le blocage) observée doit être une exécution (ou trace) possible de l'interaction asynchrone des spécifications. La différence entre les différentes approches globales existantes se situe dans la méthode de calcul de ces traces de l'interaction.

4.3.2 Méthode basée sur les critères d'interopérabilité globale

L'approche de génération de tests d'interopérabilité globale, telle qu'elle peut être dérivée du critère d'interopérabilité globale totale, est représentée dans la figure 4.2. Cette approche s'appuie sur un modèle de l'interaction des spécifications et sur un objectif de test d'interopérabilité. La génération automatique de tests d'interopérabilité doit alors chercher les chemins permettant d'exécuter l'objectif de test dans l'interaction des spécifications. L'exécution du cas de test ainsi généré renvoie un verdict par rapport à la réalisation de l'objectif de test d'interopérabilité lors de l'interaction des implémentations.

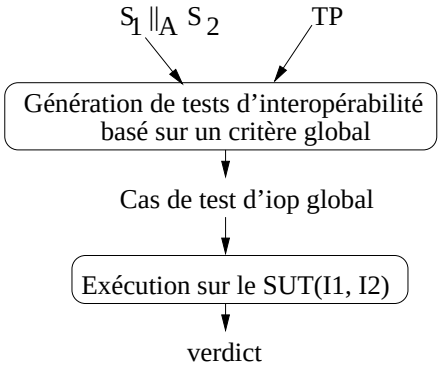


FIG. 4.2 – Approche globale de génération de test d'interopérabilité

Plusieurs méthodes peuvent être mises en oeuvre pour implémenter la phase de génération de tests :

- Une première possibilité consiste à calculer le modèle de l'interaction asynchrone des spécifications, puis à chercher dans cette interaction les chemins permettant d'exécuter l'objectif de test d'interopérabilité défini. C'est la méthode décrite dans la figure 4.2. Une fois l'interaction asynchrone des spécifications modélisée, si cette interaction est modélisable sans explosion combinatoire du nombre d'états, la génération du cas de test d'interopérabilité peut utiliser un outil de génération de tests de conformité. En effet, un tel outil recherche dans une spécification les traces correspondant à un objectif de test. Cependant, il faut que cet outil soit capable de générer des cas de tests contrôlables pour le contexte du test d'interopérabilité. Or, le contrôle des interfaces inférieures est différent dans les contextes de conformité et d'interopérabilité. Nous verrons plus précisément sur un exemple lors de l'application de la méthode globale section 4.8 ce qu'implique cette différence de contrôle des interfaces inférieures dans les deux contextes.
- L'autre méthode de génération est celle utilisée dans [KC00] ou [BCKZ02]. Elle correspond également à ce qui est réalisé dans la méthode globale "manuelle". Il s'agit de calculer une interaction incomplète guidée par l'objectif de test des spécifications contenant les différents chemins permettant d'exécuter cet objectif de test. Ce type de méthodes a été développé de façon à tenter d'éviter l'explosion combinatoire du nombre d'états généralement rencontrée lors du calcul de l'interaction des spécifications. Cependant, les scénarios correspondant à un objectif de test d'interopérabilité pouvant être multiples, le problème d'explosion combinatoire du nombre d'états peut également être rencontré dans ce type de méthode.

Dans la suite, la méthode appelée "méthode globale" sera la méthode comprenant les étapes suivantes (cf. figure 4.2) :

1. calcul du modèle de l'interaction asynchrone des spécifications $S_1 \|_{\mathcal{A}} S_2$ (cf. section 4.3.3 pour les problèmes d'explosion combinatoire liés à cette étape).
2. génération du (ou des) cas de test d'interopérabilité correspondant à l'objectif de test d'interopérabilité iop-TP défini. Cette étape correspond à un parcours du graphe représentant l'interaction asynchrone des spécifications $S_1 \|_{\mathcal{A}} S_2$. Lors de ce parcours, les différentes séquences correspondant à l'objectif de test d'interopérabilité iop-TP sont recherchées.
   Cette étape peut réutiliser un outil de génération de test de conformité. En effet, un outil de génération de test de conformité comme TGV [JJ05] (cf. section 2.4.3) est un outil qui recherche dans une spécification les chemins correspondant à un objectif de test de conformité. Un tel outil peut donc être utilisé pour rechercher dans l'interaction des spécifications les chemins correspondant à un objectif d'interopérabilité.
   Cependant, compte-tenu des différences entre contextes de conformité et d'in-

teropérabilité, des modifications doivent être apportées aux cas de test renvoyés par l'outil de génération de test. En particulier, dans le contexte du test d'interopérabilité, les testeurs ne contrôlent pas l'interaction. Il faut donc que les cas de test d'interopérabilité prennent en compte qu'il n'est pas possible de connaître à l'avance le message qu'une implémentation va envoyer à l'implémentation homologue. Les modifications (et complétion) des cas de test à réaliser ici sont les mêmes que celles à effectuer dans la méthode bilatérale que nous proposons dans la suite de ce chapitre : cf. dans la section 4.4.2.2 pour les détails de ces modifications.

3. calcul des dépendances causales entre entrées et sorties (voir section 4.5 pour la méthode de calcul) permettant de savoir si les entrées décrites dans le cas de test d'interopérabilité ont effectivement été exécutées (vérification de la propriété Pr_Int_1 de la définition de la notion d'interopérabilité de la section 3.5.1).
   En effet, les cas de test obtenus après les étapes 1 et 2 permettent de vérifier les propriétés Pr_Int_2 et Pr_Serv (cf. section 3.5.1) de la définition de l'interopérabilité, mais pas la propriété Pr_Int_1 qui dit que les messages envoyés par une implémentation via ses interfaces inférieures doivent être reçus par l'implémentation en interaction. Cette étape permet ainsi de calculer les sorties qui sont des dépendances causales de l'entrée à vérifier. L'observation d'une de ces sorties permet de conclure à l'exécution effective de l'entrée considérée (modélisée par un évènement $?(\mu)$ dans le modèle d'un cas de test d'interopérabilité défini dans la section 4.2.2).
4. exécution du cas de test d'interopérabilité sur le SUT composé des deux implémentations interconnectées. Cette exécution renvoie un verdict d'interopérabilité V tel que défini dans la section 4.2.3.

4.3.3 Problème d'explosion combinatoire du nombre d'états

Le principal inconvénient de l'approche globale est le risque d'explosion du nombre d'états lors du calcul de l'interaction **asynchrone** des spécifications. En effet, l'interaction asynchrone entre deux systèmes permet que un message envoyé par l'un de ces systèmes ne soit pas reçu immédiatement par le système homologue. Ceci est modélisé par des files FIFO en entrée de chaque interface inférieure servant à l'interconnexion (voir modèle de l'interaction section 3.3.2). Ainsi, lors de la modélisation de l'interaction, les messages sont stockés dans ces files entre leur envoi par un système et leur réception par l'autre. Un état de l'interaction asynchrone est donc modélisé par les états respectifs des deux implémentations en interaction et les états de files FIFO modélisant l'asynchronisme entre ces implémentations.
Considérant des files FIFO *bornées* de taille f, des spécifications de n états, et m messages différents dans l'alphabet d'entrée sur les interfaces inférieures des spécifications, le nombre d'états de cette interaction asynchrone est alors de l'ordre de

$O((n.m^f)^2)$. Ce nombre d'états peut même être infini si les files FIFO ne sont pas bornées. Ainsi, le calcul de l'interaction des spécifications peut ne pas être possible à cause de l'explosion du nombre d'états.
Un autre problème, corrélé au problème d'explosion du nombre d'états, est la taille des cas de test d'interopérabilité générés quand il a été possible de calculer l'interaction des spécifications. En effet, dans un cas de test global, il existe une synchronisation entre les évènements exécutés sur les deux implémentations. La conséquence en est que, lorsqu'il existe plusieurs ordonnancements possibles entre évènements exécutés par les deux implémentations, tous ces ordonnancements doivent être décrits dans le cas de test d'interopérabilité correspondant. Ceci peut augmenter considérablement la taille du cas de test d'interopérabilité en terme de nombre d'états et de transitions du cas de test. Cette taille peut être telle qu'elle rend difficile l'exécution et surtout l'interprétation de l'exécution des cas de test d'interopérabilité ainsi obtenus.

Conclusion : le problème principal d'une approche globale de test d'interopérabilité de deux implémentations provient de l'explosion du nombre d'états possibles lors du calcul de l'interaction des spécifications ou plus généralement de la taille des objets manipulés. Il est souvent impossible de calculer cette interaction.

4.4 Approche bilatérale de la génération de tests d'interopérabilité

Dans cette section, nous décrivons l'approche de génération de test que nous avons développée [DV07b] et qui est basée sur le critère d'interopérabilité bilatérale totale. Cette approche s'appuie sur l'équivalence entre ce critère et le critère d'interopérabilité globale totale qui a été prouvée (cf. théorème 3.1) dans la section 3.8.

4.4.1 Principe général de l'approche bilatérale

L'approche bilatérale de génération de cas de test d'interopérabilité est basée sur l'équivalence, en terme de détection de la non-interopérabilité, entre le critère d'interopérabilité bilatérale totale et le critère d'interopérabilité globale totale. L'idée est que, grâce à cette équivalence, les cas de test d'interopérabilité dérivés par une approche bilatérale doivent permettre de détecter les mêmes situations de non-interopérabilité que les cas de test d'interopérabilité qui seraient dérivés avec une approche globale à partir du même objectif de test.

Cette approche considère deux spécifications S_1 et S_2 (éventuellement $S_1 = S_2$) qui sont les spécifications des implémentations I_1 et I_2 dont on veut tester l'interopérabilité. La génération de test est basée sur un objectif de test d'interopérabilité noté iop-TP. L'idée est de générer deux cas de test d'interopérabilité unilatérale, chacun

basé sur l'une des deux spécifications. L'exécution de chacun de ces cas de test renvoie un verdict local, et la combinaison de ces deux verdicts correspond au verdict d'interopérabilité rendu par rapport à iop-TP et au critère d'interopérabilité bilatérale.
Pour cela, on peut dériver deux objectifs de test unilatéraux iop-TP$_i$ ($i \in \{1, 2\}$) qui ne contiennent que des évènements d'une des spécifications. La dérivation de ces iop-TPs unilatéraux doit être telle qu'il n'y ait pas de perte d'information par rapport à l'iop-TP global (cf. section 4.4.2.1 pour la méthode de dérivation). Il est alors possible de baser la génération de chacun des deux cas de test unilatéraux iop-TC$_i$ sur l'objectif de test unilatéral iop-TP$_i$ et la spécification S_i correspondants. L'objectif ici est d'éviter la construction de l'interaction des spécifications qui est généralement la source de l'explosion combinatoire du nombre d'états lors de l'approche globale (cf. section 4.3.3).
A partir d'un objectif de test unilatéral iop-TP$_i$ et de la spécification correspondante S_i, les cas de test d'interopérabilité peuvent ainsi être générés grâce à un outil de génération de tests d'interopérabilité $\mathcal{F}$ tel que $\mathcal{F}$: (S_i, iop-TP$_i$) $\rightarrow$ iop-TC$_i$, $i \in \{1, 2\}$. Cet outil $\mathcal{F}$ cherche les chemins exécutant iop-TP$_i$ dans la spécification S_i. Un outil de parcours de graphe peut être utilisé dans cette étape pour chercher les séquences correspondant à chacun des iop-TP$_i$ dans la spécification S_i respective. Comme un outil de génération de test de conformité comme TGV [JJ05] (cf. section 2.4.3) est un outil qui recherche dans une spécification les chemins correspondant à un objectif de test de conformité, un tel outil peut être réutilisé pour cette recherche. Cependant, à cause des différences entre conformité et interopérabilité, les cas de test conf-TC$_i$ générés par un outil de génération de test de conformité ne sont pas utilisables directement. Ils doivent être modifiés et complétés de façon à être exécutables dans un contexte de test d'interopérabilité (cas de test iop-TC$_i$) : cf. section 4.4.2.2 pour la génération des cas de test iop-TC$_i$ à partir des iop-TP$_i$ et S_i correspondants.
Les différentes étapes de cette approche bilatérale de génération de tests d'interopérabilité sont détaillées dans la section suivante.

4.4.2 Description de la méthode de génération de tests

Les différentes étapes de l'approche bilatérale de génération de tests d'interopérabilité sont décrites dans la figure 4.3. Les données en entrée sont les deux spécifications S_1 et S_2 et l'objectif de test d'interopérabilité iop-TP. Nous donnons ici les étapes générales de cette approche : ces étapes sont détaillées dans la suite de cette section et dans la section suivante.

1. La première étape de l'approche bilatérale consiste à dériver deux objectifs de test unilatéraux iop-TP$_1$ et iop-TP$_2$ à partir de l'objectif de test global en entrée de la méthode (cf. section 4.4.2.1). L'objectif de cette étape est de générer un objectif de test unilatéral pour chacun des systèmes en interaction. Chaque iop-TP$_i$ ne contient que des évènements de la spécification S_i correspondante. Ainsi, il est possible de recherche dans la spécification S_i les évènements correspondant

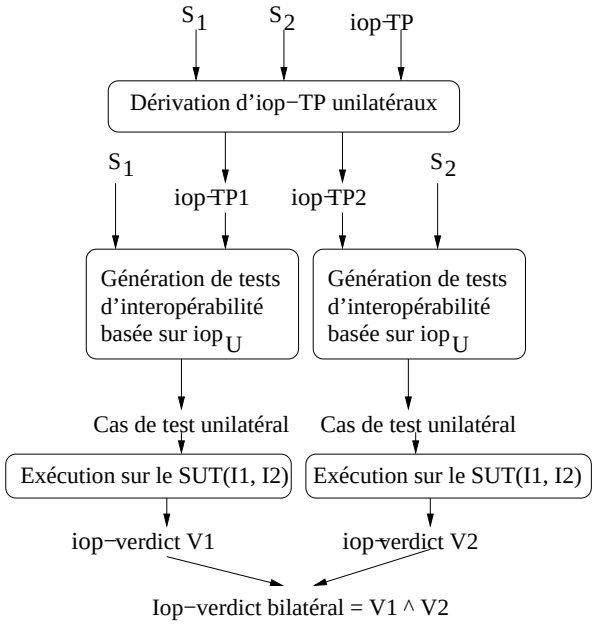


FIG. 4.3 – Approche bilatérale de génération de test d'interopérabilité

à cet objectif de test.

2. La deuxième étape est la génération des cas de test d'interopérabilité. Des cas de test d'interopérabilité unilatéraux sont générés à partir des objectifs de test unilatéraux et des spécifications. Le cas de test unilatéral iop-TC_1 (resp. iop-TC_2) est ainsi généré à partir de iop-TP_1 et S_1 (resp. iop-TP_2 et S_2). Cette étape, détaillée dans la suite de cette section, peut être décomposée ainsi :
   (a) Le chemin permettant d'exécuter l'objectif de test unilatéral est généré à l'aide d'un outil de génération de test de conformité (tel que par exemple TGV [JJ05]). Les conditions d'utilisation de l'outil TGV pour générer les chemins correspondant à l'objectif de test d'interopérabilité unilatéral sont précisées dans la section 4.4.2.2.
   (b) Les cas de test obtenus, qui sont des cas de test de conformité, sont modifiés de façon à prendre en compte les différences entre tests de conformité et d'interopérabilité. En particulier, les modifications sont dues au fait qu'en contexte de test d'interopérabilité, les évènements de l'interaction ne sont pas contrôlables. Ces modifications sont détaillées dans la section 4.4.2.2.
   (c) Les cas de test unilatéraux obtenus sont complétés de façon à prendre en compte la vérification des entrées. Cette opération est réalisée à l'aide d'un algorithme basé sur les dépendances causales entre entrées et sorties. L'algorithme et son application à cette méthode sont décrits dans la section 4.5.

3. Chacun des cas de test unilatéraux est exécuté sur l'IUT concernée lors de son interaction avec l'autre implémentation.
4. Chaque cas de test renvoie un verdict lors de l'exécution. Ces verdicts sont des verdicts d'interopérabilité locaux (ou unilatéraux). Leur combinaison permet d'obtenir le verdict global relatif à l'objectif de test iop-TP initial. Ce verdict est un verdict d'interopérabilité "bilatéral" et est obtenu par les règles de précédences évidentes suivantes : $PASS \wedge PASS = PASS$, $PASS \wedge INC = INC$, $PASS \wedge FAIL = FAIL$, $INC \wedge FAIL = FAIL$, $INC \wedge INC = INC$ et $FAIL \wedge FAIL = FAIL$.

Dans les sections 4.4.2.1 et 4.4.2.2, nous détaillons les deux étapes principales de la méthode qui sont les étapes 1 et 2, c'est-à-dire la dérivation d'objectifs de test unilatéraux et la génération des cas de test d'interopérabilité correspondants. L'étape 2c correspondant à la génération des sorties qui sont des dépendances causales d'une entrée dont on veut vérifier l'exécution est détaillée dans la section 4.5.

4.4.2.1 Dérivation d'objectifs de test unilatéraux

L'algorithme permettant de dériver deux objectifs de test unilatéraux à partir de l'objectif de test d'interopérabilité iop-TP est donné figure 4.4. Les iop-TPs considérés dans cet algorithme sont une version simplifiée du modèle de la section 4.2.1. Cette version simplifiée décrit un objectif de test avec une seule action possible après chaque état, c'est-à-dire que l'iop-TP est une séquence d'évènements telle que $\forall\ \sigma$, $|\Gamma$(iop-TP, $\sigma)| \leq 1$. Dans ce cas, un seul état-puit $Accept^{iop-TP}$ est utilisé pour sélectionner le comportement que l'on souhaite tester.
Les fonctions suivantes sont utilisées dans cet algorithme : *remove_last*, *last_event*, *nb_Event* et *error*. La fonction *remove_last* prend en paramètre une trace d'évènements, enlève le dernier évènement de cette trace et renvoie le résultat de cette opération. Elle est définie par : remove_last($\sigma.a$) $= \sigma$. La fonction *last_event* prend en paramètre une trace d'évènements et renvoie le dernier évènement de cette trace. Elle est définie par : last_event(σ)= ϵ if σ= ϵ et last_event(σ)= a if σ= $\sigma_1.a$. La fonction nb_Event(Ens) renvoie le nombre d'évènements présents dans l'ensemble Ens. La fonction *error* renvoie la raison de l'erreur avant de sortir de l'algorithme.
L'algorithme est également basé sur une fonction *add_precursor*(μ, S, iop-TP) décrite dans la figure 4.4. Cette fonction prend en paramètre un évènement μ, une spécification S, un objectif de test unilatéral en construction iop-TP et l'autre objectif de test en construction iop-TP'. Elle recherche un prédécesseur de μ dans la spécification S tel que ce prédécesseur est un évènement à exécuter sur une interface inférieure et elle ajoute à l'objectif de test unilatéral en construction iop-TP l'iop-miroir (calculé selon la définition 3.1 de la section 3.3.1) de ce prédécesseur. Deux situations sont possibles. Soit des évènements ont été ajouté à iop-TP' ce qui permet d'utiliser les informations de iop-TP' pour borner la recherche de prédécesseur. Soit iop-TP'$= \epsilon$

Procédure MAIN
Données en entrée : S_1, S_2 : spécifications
iop-TP : iop-TP global ;
Données renvoyées : $\{\text{iop-TP}_i\}_{i=1,2}$: iop-TPs unilatéraux ;
Invariant : $S_k = S_{3-i}$ (* S_k est l'autre spécification *) ; iop-TP= $\mu_1 ... \mu_n$
Initialisation : iop-TP$_i := \epsilon \; \forall i \in \{1, 2\}$;
DEBUT
for ($j = 0$;$j \leq n$;j++) **do**
if ($\mu_j \in \Sigma_L^{S_i}$) **then** /* cas α */
iop-TP$_i$:=iop-TP$_i$.μ_j ; iop-TP$_k$:=iop-TP$_k$.$\bar{\mu_j}$ **end(if)**
if ($\mu_j \in \Sigma_L^{S_k}$) **then** /* cas β */
iop-TP$_i$:=iop-TP$_i$.$\bar{\mu_j}$; iop-TP$_k$:=iop-TP$_k$.μ_j **end(if)**
if ($\mu_j \in \Sigma_U^{S_i}$) **then** /* cas γ */
iop-TP$_k$:=add_precursor(μ_j, S_i, iop-TP$_k$, iop-TP$_i$) ;
iop-TP$_i$:=iop-TP$_i$.μ_j **end(if)**
if ($\mu_j \in \Sigma_U^{S_k}$) **then** /* cas η */
iop-TP$_i$:=add_precursor(μ_j, S_k, iop-TP$_i$, iop-TP$_k$) ;
iop-TP$_k$:=iop-TP$_k$.μ_j **end(if)**
if ($\mu_j \notin \Sigma^{S_k} \cup \Sigma^{S_i}$) **then** /* cas λ */
error(iop-TP non valide : $\mu_j \notin \Sigma^{S_1} \cup \Sigma^{S_2}$) **end(if)**
end(for)
FIN

fonction add_precursor(μ, S, iop-TP, iop-TP') : **return** iop-TP
DEBUT
if (iop-TP'$\neq \epsilon$) **then**
a_j =last_event(iop-TP')
$M = \{q \in Q^S$; $\exists$ q'|(q, $\bar{a}_j, q') \wedge \sigma = \bar{a}_j.\omega.\mu \in Traces(q)\}$
if ($\forall q \in M, \sigma \notin Traces(q)$) **then** error(no path to μ) **end(if)**
while (e=last_event(ω) $\notin \Sigma_L^S \cup \{\epsilon\}$) **do** ω=remove_last(ω) **end(while)**
if (e $\in \Sigma_L^S$) **then** iop-TP=iop-TP.$\bar{e}$ **end(if)**
else found=false ; /* cas iop-TP'=ϵ */
q = research_1st_state(μ, S)
while ($\neg$ found) **do**
Γ^{-1} =set_of_events_before(q, S)
for (i=0 ; i<nb_Event(Γ^{-1}) $\wedge \neg$ found ; i++) **do**
$\alpha = \Gamma^{-1}[i]$
Let $q_1 \in Q^S$ such that (q_1, α, q) $\in \Delta^S$
if $\alpha \in \Sigma_L^S$ **then** iop-TP=iop-TP.$\bar{\alpha_i}$; found=true **end(if)**
end(for)
q=q_1
end(while)
end(if)
FIN

FIG. 4.4 – Algorithme de dérivation des iop-TP$_{S_i}$ à partir de iop-TP

et dans ce cas, il n'existe pas d'information permettant de borner la recherche de prédécesseur. Les fonctions research_1st_state et set_of_events_before sont utilisées. La fonction research_1st_state(μ, S) utilise un algorithme de recherche en profondeur pour trouver le premier état q de S tel que μ est exécutable dans l'état q. La fonction set_of_events_before(q, S) recherche les évènements de S tels que l'exécution d'un de ces évènements amène la spécification S dans l'état q.

Le principe de l'algorithme est le suivant. Soit μ un évènement de iop-TP. Selon μ, plusieurs possibilités :

- Cas $\mu_j \in \Sigma_L^{S_i}$ et $\mu_j \in \Sigma_L^{S_k}$ (cas α et β) : μ est un évènement qui doit être exécuté sur une interface inférieure (servant à l'interaction). Dans ce cas, d'après la définition d'un évènement miroir de l'interaction (ou iop-miroir, définition 3.1 de la section 3.3.1), cela signifie que les deux évènements μ et $\bar{\mu}$ (c'est-à-dire l'envoi et la réception d'un message) doivent être exécutés par les deux systèmes de spécifications S_1 et S_2. Les évènements μ et $\bar{\mu}$ sont donc ajoutés aux iop-TPs unilatéraux adéquates, c'est-à-dire $\mu \in \Sigma^{S_i}$ est ajouté à iop-TP$_i$ et $\bar{\mu}$ (qui est donc un évènement de Σ^{S_j}) est ajouté à iop-TP$_j$.
- Cas $\mu_j \in \Sigma_U^{S_i}$ et $\mu_j \in \Sigma_U^{S_k}$ (cas γ et η) : μ est un évènement qui doit être exécuté sur une interface supérieure donc non utilisée pour l'interaction entre les deux implémentations. Dans ce cas, la situation est plus complexe. En effet, pour ce type d'évènement, il n'existe pas d'évènement miroir sur l'autre spécification. L'évènement $\mu \in \Sigma^{S_i}$ peut être ajouté directement à iop-TP$_i$, mais pour iop-TP$_j$, la recherche d'un évènement situé dans la même trace que μ est nécessaire. L'évènement à ajouter à iop-TP$_j$ est alors le miroir d'un prédécesseur de μ dans S_i.
- Cas $\mu_j \notin \Sigma^{S_k} \cup \Sigma^{S_i}$ (cas λ) : μ n'est ni un évènement de S_1, ni un évènement de S_2. Dans ce cas, iop-TP n'est pas un objectif de test valide puisqu'il contient des évènements qui ne sont pas prévus dans les spécifications des systèmes testés.

4.4.2.2 Dérivation des cas de test unilatéraux

L'étape de dérivation des cas de test unilatéraux prend en entrée les objectifs de test unilatéraux iop-TP$_1$ et iop-TP$_2$ obtenus à l'étape précédente et les deux spécifications S_1 et S_2. L'objectif est de générer deux cas de test d'interopérabilité unilatéraux servant à vérifier l'exécution de iop-TP$_i$ par l'implémentation I_i basée sur la spécification S_i, lors de l'interaction de I_i avec i_j.
Cette étape recherche dans une spécification les traces correspondant à un objectif de test. Elle peut donc réutiliser un outil de génération de test de conformité tel que TGV [JJ05]. Cet outil prend en entrée une spécification S_1 (resp. S_2) et l'iop-TP unilatéral correspondant iop-TP$_1$ (resp. iop-TP$_2$). Il génère le cas de test de conformité correspondant conf-TC$_1$ (resp. conf-TC$_2$). Ces cas de test conf-TC$_1$ et conf-TC$_2$ doivent être

modifiés et complétés de façon à obtenir les cas de test d'interopérabilité iop-TC_1 et iop-TC_2.

Les modifications à appliquer sur conf-TC_1 et conf-TC_2 sont dues plus particulièrement aux différences de contrôle des interfaces inférieures entre les contextes de conformité et d'interopérabilité.

L'une des modifications à réaliser est l'adaptation des types d'évènements du cas de test au contexte de l'interopérabilité (cf. définition des évènements possibles pour un iop-TC dans la section 4.2.2). En effet, dans un cas de test d'interopérabilité, les évènements que peut exécuter un testeur sont l'envoi de stimuli vers les interfaces supérieures et la réception de messages en provenance de ces interfaces, et l'observation de la communication entre implémentations via les interfaces inférieures. Il faut donc faire apparaître l'observation de ces interactions, c'est-à-dire l'observation des émissions et des réceptions sur les interfaces inférieures.

Cette différence entre interfaces inférieures et supérieures en interopérabilité a également une influence sur la contrôlabilité des cas de test (cf. section 4.2.2 pour la définition d'un cas de test contrôlable en conformité et en interopérabilité). Comme les interfaces inférieures sont contrôlables en conformité, un cas de test de conformité ne prévoit dans chaque état qu'une seule réception sur les interfaces inférieures de l'implémentation : le message correspondant est envoyé par le testeur donc contrôlé lors des tests de conformité. Dans le cas du contexte du test d'interopérabilité, les réceptions sur les interfaces inférieures correspondent à des envois de l'autre implémentation : elle ne sont donc pas contrôlées par le testeur. Il faut donc que le cas de test prévoit toutes les entrées sur les interfaces inférieures pouvant correspondre à des envois de l'autre implémentation. Pour cette raison, la sortie de l'outil de génération de test de conformité que nous transformons pour l'interopérabilité n'est pas le cas de test de conformité mais le graphe de test complet (ou CTG pour Complete Test Graph). Un graphe de test complet est un IOLTS qui contient toutes les séquences correspondant à un objectif de test donné. Cet IOLTS contient donc toutes les entrées sur les interfaces inférieures. Les IOLTS conf-TC_1 et conf-TC_2 auxquels sont appliquées les modifications sont donc les CTGs renvoyés par l'outil de génération de test de conformité utilisé.

Les modifications à opérer sur les CTGs conf-TC_1 et conf-TC_2 de façon à obtenir les cas de test d'interopérabilité iop-TC_1 et iop-TC_2 sont les suivantes :

– Adaptation des types d'évènements au contexte du test d'interopérabilité. Il n'y a pas de modification pour les évènements exécutés sur les interfaces supérieures. En effet, dans le contexte du test d'interopérabilité comme dans celui du test de conformité, le testeur qui est connecté à ces interfaces peut les contrôler. Par contre, en interopérabilité, les testeurs connectées aux interfaces inférieures des implémentations observent les interactions, mais ne contrôlent pas

ces évènements. Les modifications sur les types d'évènements consistent donc à transformer les échanges entre un testeur et une interface inférieure décrits pour la conformité en des observations par les testeurs du test d'interopérabilité. Par exemple, si un évènement $Tl?m$ (réception du message m par le testeur Tl connecté à l'interface inférieure l) est décrit dans conf-TC$_i$, alors ce message est remplacé par $?(l!m)$ qui signifie que le testeur doit observer l'envoi du message m sur l'interface inférieure l de l'implémentation. De même, $Tl!m$ est remplacé par $?(l?m)$.

– Choix d'un cas de test d'interopérabilité dans les CTGs obtenus après transformation des types d'évènements. En effet, le CTG peut contenir contenir plusieurs cas de test d'interopérabilité. Il faut donc choisir un cas de test d'interopérabilité contrôlable (cf. section 4.2.2) dans chacun des CTGs générés.

Les cas de test d'interopérabilité obtenus après les modifications ci-dessus sont basés uniquement sur la partie "vérification des sorties et blocages" du critère d'interopérabilité bilatérale. Il faut ensuite compléter ces cas de test unilatéraux de façon à prendre en compte les entrées (c'est-à-dire la deuxième partie du critère d'interopérabilité). Cette prise en compte des entrées est faite à l'aide des dépendances causales entre entrées et sorties. L'algorithmique liée à cette prise en compte est décrite dans la section 4.5.
Les différentes étapes de la génération des cas de test d'interopérabilité à partir d'un outil de génération de tests de conformité sont résumées dans la figure 4.5.

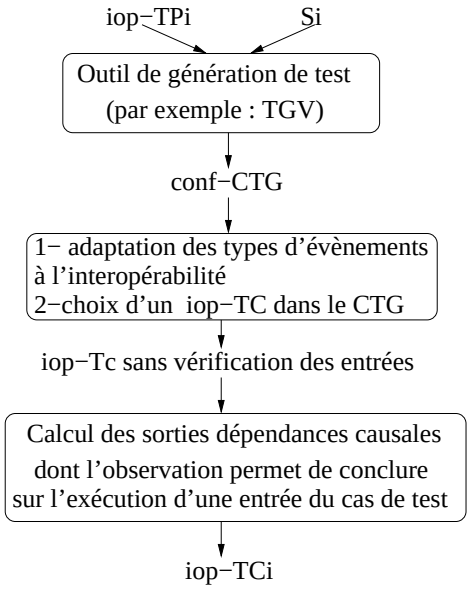


FIG. 4.5 – Génération d'un cas de test d'interopérabilité unilatérale

4.4.3 Complexité et comparaison avec la méthode globale

Deux étapes interviennent principalement dans le calcul de la complexité de l'approche bilatérale : ce sont les étapes de dérivation des objectifs de test unilatéraux et de génération des cas de test d'interopérabilité correspondants.
La première de ces étapes (dérivation des objectifs de test unilatéraux : cf. algorithme de la figure 4.4) est linéaire par rapport à la taille maximum des spécifications. En effet, la première partie de l'algorithme est linéaire : c'est un simple parcours de l'objectif de test qui est un IOLTS de petite taille (en nombre d'états et transitions) comparé à un IOLTS représentant une spécification. La deuxième partie de cet algorithme (recherche d'un prédécesseur, exécutée uniquement si l'évènement traité est un évènement s'exécutant sur une interface supérieure) est également linéaire : c'est un parcours incomplet d'une spécification utilisant une structure de pile.
La seconde étape de l'approche bilatérale est la génération des cas de test unilatéraux. Cette génération de test est basée sur l'utilisation d'un outil de génération de tests de conformité suivie de l'adaptation des types d'évènements au contexte de l'interopérabilité. Dans notre méthode, nous avons utilisé l'outil de génération de tests de conformité TGV (cf. section 2.4.3) pour générer les CTGs (graphes de test complet) à modifier pour obtenir les cas de test d'interopérabilité. Comme TGV [JCTG96] est linéaire en complexité, l'étape de génération de cas de test est également linéaire.
Les modifications sur les IOLTS renvoyés par TGV sont réalisées via un parcours de ces IOLTS. Tous les évènements des IOLTS en sortie de TGV devant être parcourus, un parcours de type parcours en profondeur ou en largeur peut être utilisé indifféremment dans cette étape. Quelque soit le type de parcours choisi, la complexité de cette opération est également linéaire.
Ainsi, les différentes opérations impliquées dans la génération des cas de test d'interopérabilité avec complétion par la prise en compte des entrées ont au pire une complexité linéaire par rapport à la taille des spécifications (resp. des iop-TPs). Cette méthode est donc moins coûteuse que le calcul de $S_1 \|_{\mathcal{A}} S_2$ à effectuer lors d'une approche globale qui est en $o(n^2)$. De plus, si un cas de test global peut être généré via une approche globale, la méthode bilatérale permet de générer un cas de test d'interopérabilité bilatérale équivalent. Cette méthode permet également de générer des cas de test d'interopérabilité quand une approche globale fait face à un problème d'explosion combinatoire du nombre d'états.

N.B. : Les cas de test obtenus à ce stade (méthode globale ou méthode bilatérale) sont ensuite complétés pour la vérification des entrées grâce à un algorithme basé sur les dépendances causales entre entrées et sorties. Cet algorithme est le même dans les deux méthodes. Ce qui est différent, ce sont les données en entrée de l'algorithme. En particulier, cet algorithme prend en entrée les cas de test renvoyés par la première partie de chaque méthode, c'est-à-dire "l'iop-TP sans vérification des entrées", et la ou les spéci-

fications correspondantes. La vérification des entrées pour la méthode globale est donc basée sur les données de grande taille (en nombre d'états et transitions) que sont un cas de test global et les deux spécifications, là où la même opération pour l'algorithme bilatéral prend en entrée un cas de test unilatéral et la spécification correspondante.

4.5 Vérification des entrées

Dans cette section, nous nous intéressons à l'algorithmique permettant de compléter, en prenant en compte la vérification des entrées, les cas de test d'interopérabilité obtenus par la méthode bilatérale. Nous décrivons également les possibilités d'utilisation d'un tel algorithme avec une approche globale de la génération de test d'interopérabilité.

4.5.1 Motivation

L'un des objectifs du test d'interopérabilité est de vérifier que les implémentations interconnectées arrivent à communiquer. Il faut donc, en plus de la vérification des sorties, vérifier que les messages envoyés par une implémentation sont effectivement reçus par l'autre implémentation. Comme vu dans la section 3.5.3, les entrées ne sont pas observables par les testeurs. Un testeur connecté à une interface inférieure connaît les messages transmis à l'implémentation via cette interface, mais il ne sait pas si l'implémentation a effectivement traité ces messages. La vérification des entrées se fait donc en observant les sorties qui ne peuvent avoir lieu que si l'implémentation a effectivement pris en compte l'entrée considérée (correspondant à la réception d'un message envoyé par l'autre entité).

En interopérabilité, il est possible de définir des objectifs de test dont le dernier évènement (transition vers l'état-puit $Accept^{iop-TP}$) est une entrée d'une des implémentations. Ce type d'objectif de test répond au besoin de vérification de la capacité de communication entre les implémentations. De plus, la dérivation d'un objectif de test global en deux objectifs de test unilatéraux (via l'ajout dans un des deux iop-TPs unilatéraux d'un iop-miroir) peut également aboutir à des objectifs de test d'interopérabilité dont le dernier évènement est une entrée.

Dans ce type de situation (objectif de test dont le dernier évènement est une entrée), la génération de test (réalisée ici à l'aide d'un outil de génération de tests tel que TGV) renvoie un cas de test dont le dernier évènement (transition vers $PASS$) est l'entrée considérée, et ceci sans postambule. Il faut alors, pour répondre à l'objectif de test, générer les sorties et traces associées qui permettront de vérifier si cette entrée a été exécutée. Le cas de test est ainsi complété de façon à répondre à tous les objectifs du

test d'interopérabilité.

4.5.2 Algorithme de calcul des dépendances causales

Dans cette section, nous considérons un cas de test d'interopérabilité unilatérale TC_i dont la dernière transition (transition vers $PASS$) a un évènement de type "observation d'une entrée sur une interface inférieure". Ce cas de test unilatéral est un cas de test se focalisant sur l'implémentation I_i de spécification S_i. Soit μ l'entrée de S_i à vérifier (c'est-à-dire l'évènement étiquetant la transition vers $PASS$ de TC_i). L'objectif de l'algorithme de dépendances causales est alors de calculer les sorties de S_i (et les traces de S_i entre μ et ces sorties) qui seront observées sur I_i si l'entrée μ a effectivement été exécutée.

L'algorithme permettant de calculer les dépendances causales de l'entrée μ dans la spécification S_i est représenté dans la figure 4.6. Cet algorithme prend en entrée trois types de données :

- la **spécification** S_i qui est la spécification de l'implémentation réceptrice, et donc la spécification sur laquelle on veut calculer les dépendances entre évènements. S_j est la spécification de l'autre système du SUT.
- une **trace** σ qui est une trace de S_i. Cette trace est une trace de S_i qui peut être exécutée lors de l'interaction entre les systèmes basés sur S_i et S_j. De plus, cette trace correspond à une trace d'un cas de test unilatéral. En général, on utilise ici la trace σ telle que $\sigma.\mu$ est la trace de TC_i reliant de façon la plus directe possible $q_0^{TC_i}$ à $Pass^{TC_i}$.
- l'**entrée** $\mu = l_i?m$ dont on veut vérifier l'exécution grâce à l'observation d'une sortie qui en dépend causalement. Cette entrée correspond à un message qui a été envoyé par l'implémentation I_j (de la spécification S_j) : $\bar{\mu} = l_j!m$ tel que l_j est l'interface de I_j reliée à l'interface l_i de I_i.

Les données que doit renvoyer l'algorithme de la figure 4.6 sont :

- m le nombre de sorties dépendances causales de l'entrée μ calculées par l'algorithme.
- Un ensemble de couples $(CDep(S_i, \sigma, \mu)[i], T[i])$
  1. le premier élément d'un de ces couples est une sortie appartenant à l'ensemble $CDep(S_i, \sigma, \mu)$ des sorties qui sont des dépendances causales de l'entrée μ d'après la spécification S_i.
  2. le deuxième élément est une trace $T[i]$ (correspondant à la trace σ' de la définition de la condition basée sur les dépendances causales entre une entrée et des sorties, cf. section 3.5.3). $T[i]$ donne la trace σ' à exécuter entre l'entrée μ et la sortie dépendance causale $CDep(S_i, \sigma, \mu)[i]$.
     Il est en effet important de connaître cette trace σ' pour le cas où la sortie $CDep(S_i, \sigma, \mu)[i]$ dépend de l'exécution de μ, mais également d'une

Procédure MAIN
Données en entrée : S_i : Spécification,
σ : trace d'évènements de S_i,
μ : entrée de S_i
Données renvoyées : $CDep(S_i, \sigma, \mu)$: ensemble (ordonné) de dépendances causales,
T : ensemble (ordonné) des traces σ'
exécutables entre μ et une sortie,
m : nombre d'évènements de l'ensemble $CDep$
Initialisation : $\Gamma := \Gamma(S_i, \sigma.\mu)$;
$m :=$nb_event(Γ)
for $(j := 0; j \leq m; j++)$ **do**
create(find[j]) ; find[j]=false ;
create($T[j]$) ; $T[j] := \epsilon$
end(for)
Invariant : $m \geq nb\_event(\Gamma)$
DEBUT
while $\exists x(x < n)$, find[x] :=false **do**
for $(j := 0; j < m; j++)$ **do**
if (find[j]=false) **do** $Evt := \Gamma(S_i, \sigma.\mu.T[j])$
if ($Evt[0] \in \Sigma_O^{S_i}$) **do**
find[j] :=true ;
create($CDep(S_i, \sigma, \mu)$[j]) ; $CDep(S_i, \sigma, \mu)$[j] := Evt[0]
else $T[j]$:= $T[j]$.Evt(0) **end(if)**
if (nb_event(Evt)> 1) **do**
for($k := 1$; k $\leq$ nb_event(Evt) ; $k++$) **do**
m++ ;
create($T[m]$) ; $T[m] := T[j]$
create(find[m]) ; find[m] :=false
if ($Evt[k] \in \Sigma_O^{S_i}$) **do**
find[m] :=true ;
create($CDep(S_i, \sigma, \mu)$[m]) ; $CDep(S_i, \sigma, \mu)$[j] := Evt[k]
else $T[m]$:= $T[m]$.Evt(k)
end(if)
end(for)
end(if)
end(if)
end(for)
end(while)
FIN

FIG. 4.6 – Algorithme de recherche des sorties dépendances causales de l'entrée μ dans la spécification S_i

autre entrée en plus de μ. Si cette entrée doit être exécutée sur une interface supérieure, il faut donc appliquer le stimulus correspondant pour que $CDep(S_i, \sigma, \mu)[i]$ s'exécute et ainsi pouvoir conclure sur l'exécution de μ. Si la deuxième entrée doit être exécutée sur une interface inférieure, il faut pouvoir déterminer les évènements conduisant à l'envoi correspondant : l'observation de $CDep(S_i, \sigma, \mu)[i]$ permettra alors de conclure à l'exécution de μ, mais également à l'exécution de cette autre entrée.

L'algorithme de la figure 4.6 utilise les fonctions suivantes : $nb\_event(Ens)$ et $create(\ Data)$. La fonction $nb\_event(Ens)$ renvoie le nombre de données contenues dans l'ensemble de données Ens. La fonction $create(Data)$ crée une nouvelle donnée, appelée $Data$, en fonction des besoins de l'algorithme.

L'algorithme de la figure 4.6 est basé sur le principe des algorithmes de parcours de graphe de type recherche en largeur d'abord (breadth-first search algorithms). L'algorithme démarre dans l'état q de S_i tel que $q = S_i\, after\, \sigma.\mu$. Les évènements exécutables par S_i après la trace $\sigma.\mu$ (c'est-à-dire les évènements de l'ensemble $\Gamma(S_i, \sigma.\mu)$) sont examinés. Les sorties sont ajoutées dans l'ensemble des dépendances causales tandis que sur les autres branches, on continue la recherche jusqu'au prochain évènement de sortie. Les traces à exécuter entre μ et chaque sortie dépendance causale de μ sont mémorisées dans l'ensemble T de traces, tel que $T[i]$ est la trace à exécuter entre μ et la sortie $CDep(S_i, \sigma, \mu)$[i] de l'ensemble des sorties dépendances causales de l'entrée μ.

4.5.3 Approches de génération de test d'interopérabilité et utilisation des dépendances causales entre évènements

L'algorithme de la figure 4.6 et présenté dans la section 4.5.2 est ainsi l'algorithme qui sert à compléter la méthode bilatérale (étape "dépendances causales pour vérification des entrées" de la figure 4.5). Cet algorithme sert à "compléter" des cas de test unilatéraux iop-TC_1 et iop-TC_2.
Le cas de test iop-TC_i en entrée de l'algorithme s'arrête sur un verdict $Pass$ associé à une transition étiquetée par une entrée. Ce verdict est transformé en verdict provisoire $(Pass)$ qui indique qu'il n'y a pas eu d'erreur lors du test, mais qui ne conclut pas de façon définitive sur la réalisation de l'objectif de test durant les tests. Le dernier évènement à vérifier est alors l'exécution de l'entrée.

L'algorithme de recherche des dépendances causales renvoie un ensemble de sorties avec les chemins associés à ces sorties. Des verdicts peuvent ensuite être associés à ces évènements. Des exemples d'associations verdicts/évènements observés sont éga-

lement donnés dans la section 3.5.3 lors de la définition de l'ensemble $CDep(S, \sigma, \mu)$ et de la définition de la condition de vérification des entrées.

- Verdict $Pass$: ce verdict est associé aux sorties dépendances causales de l'entrée μ considérée. Il indique que l'observation de la sortie correspondante permet de conclure que l'entrée μ a effectivement été exécutée.
- Verdict $Fail$: ce verdict est associé à toutes les sorties qui pourraient être observées après l'entrée μ, mais également aux blocages non prévus dans les spécifications après l'exécution de l'entrée. Il indique soit que l'entrée μ n'a pas été exécutée, soit que des erreurs d'implémentation existent dans les évènements après μ. Dans les deux cas (qu'il n'est pas possible de distinguer puisque les implémentations sont des boites noires), l'observation d'une telle entrée correspond à une erreur, d'où le verdict $Fail$.
- Verdict $Inconclusive$: ce verdict est associé aux situations où il n'est pas possible de conclure sur l'exécution effective de l'entrée. En particulier, ce verdict est rencontré dans deux types de situations.
  - Le premier type de situation correspond à l'observation d'une sortie α de $CDep(S, \sigma, \mu)$, mais qui peut être également due à un autre évènement exécuté, en particulier dans le cas où un évènement non-décrit dans la trace σ' (des évènements à exécuter entre μ et la sortie dépendance causale) a été observé. En effet, dans une telle situation, il n'est pas possible de savoir si l'exécution de la sortie α a eu lieu à cause de l'exécution de μ ou à cause de cet autre évènement. Il n'est donc pas possible de savoir si μ a effectivement été exécutée. Dans le cas où le testeur n'a pas accès à une partie des interfaces de l'implémentation lors des tests, il est également possible que l'observation d'une sortie α de $CDep(S, \sigma, \mu)$ ne permette pas de conclure à l'exécution de μ. C'est le cas si α peut également être une dépendance causale d'une autre entrée, entrée susceptible d'être exécutée sur une interface à laquelle les testeurs n'ont pas accès. Comme il n'est pas possible de savoir si le message correspondant à cette autre entrée a été envoyé, il n'est pas possible de savoir de quelle entrée α est une dépendance causale et donc de conclure sur l'exécution de μ.
  - Le deuxième type de situation correspond au cas où un blocage (autorisé dans la spécification correspondante) est observé sans possibilité de générer une sortie de l'ensemble des dépendances causales. La trace σ' des évènements à exécuter entre μ et une des sorties dépendances causales étant connue, il est généralement possible d'envoyer des stimuli qui permettront l'exécution des évènements de σ' et donc au final d'une sortie dépendance causale de μ. Cependant, il est possible que σ' contiennent une autre entrée sur une interface inférieure : or l'envoi du message correspondant n'est pas contrôlé. Dans ce cas, l'observation d'un blocage n'est pas une erreur, mais il n'est cependant pas possible de conclure sur l'exécution effective de μ.
    Cette situation (observation d'un blocage) est également rencontrée dans le

cas d'architectures de test inférieures. Par exemple, il est possible qu'une sortie dépendance ait été exécutée mais sur une interface supérieure, ou que l'exécution d'une sortie de $CDep(S, \sigma, \mu)$ soit conditionnée à l'envoi d'un stimulus sur une interface supérieure. Dans ces deux cas, l'évènement observé sera également un blocage sans possibilité de conclure sur l'exécution effective de μ.

Une telle méthode de calcul de dépendances causales entre évènements peut être adaptée à d'autres contextes. Par exemple, cet algorithme peut être adapté à une approche globale, à condition d'avoir pu dériver le cas de test d'interopérabilité global. L'algorithme prend alors en entrée le cas de test (se terminant par une entrée dont on veut vérifier l'exécution) et les deux spécifications. Il cherche les sorties dépendances causales sur la spécification du récepteur, tout en construisant si nécessaire le comportement global des spécifications entre l'entrée à vérifier et les sorties dépendances causales.

4.6 Une méthode complète de génération de tests d'interopérabilité

Une méthode de génération de tests d'interopérabilité basée sur une approche bilatérale a été définie section 4.4. Cette approche est complétée par l'algorithmique pour la vérification des entrées décrite section 4.5. Dans cette section, nous décrivons la méthode complète [DV07a] de génération de cas de test d'interopérabilité à partir d'un objectif de test. Cette méthode est basée sur les approches et algorithmes décrits dans les sections 4.4 et 4.5. Cette méthode prend en entrée deux spécifications S_1 et S_2 et un objectif de test d'interopérabilité iop-TP. Les différentes étapes de la méthode "complète" de génération de tests d'interopérabilité basée sur une approche bilatérale sont les suivantes.

1. Dérivation à partir de l'objectif de test d'interopérabilité iop-TP de deux objectifs de test unilatéraux iop-TP_1 et iop-TP_2.
2. Utilisation d'un outil de génération de test de conformité, dans notre cas l'outil TGV (Test Generation with Verification technology, [JJ05]) pour générer les CTGs (Complete Test Graph) contenant les différents cas de test de conformité dérivables à partir de S_1 et iop-TP_1, et à partir de S_2 et iop-TP_2.
3. Adaptation des CTGs obtenus conf-TC_1 (à partir de S_1 et iop-TP_1) et conf-TC_2 (à partir de S_2 et iop-TP_2) de façon à obtenir des cas de test d'interopérabilité. Pour cela, deux opérations doivent être réalisées. Ces opérations sont :
   – l'extraction d'un cas de test contrôlable en interopérabilité à partir de chacun des CTGs conf-TC_1 et conf-TC_2. Un tel cas de test ne doit pas permettre au

testeur de choisir arbitrairement entre deux stimuli vers une interface supérieure, mais doit prévoir toutes les entrées possibles sur une interface inférieure (interface non contrôlable en interopérabilité).
– l'adaptation des types d'évènements au contexte du test d'interopérabilité : les testeurs communiquent directement avec les interfaces supérieures des implémentations tandis qu'ils se contentent d'observer la communication entre implémentations via leurs interfaces inférieures.

Les cas de test obtenus après ces deux opérations sont des cas de test d'interopérabilité unilatérale. Cependant, ils peuvent être incomplets par rapport à l'objectif de test iop-TP ou par rapport à iop-TP_1 ou iop-TP_2. En effet, ces cas de test d'interopérabilité ne contiennent pas d'information permettant de vérifier si une entrée a effectivement été exécutée.

4. Application -si nécessaire- de l'algorithme basé sur les dépendances causales entre entrées et sorties de façon à permettre la vérification des entrées. Cet algorithme s'applique quand la transition vers l'état $PASS$ du cas de test est une entrée (non-observable par les testeurs). Ceci peut être dû soit à l'objectif de test iop-TP, soit à la dérivation de iop-TP en iop-TP_1 et iop-TP_2. En effet, si l'évènement associé à la transition vers l'état $Accept^{iop-TP}$ de l'un des objectifs de test iop-TP_1 et iop-TP_2 est une entrée sur une interface inférieure, le cas de test généré est tel que cette entrée est l'évènement associée au verdict $PASS$. Comme cette entrée n'est pas un évènement observable, il faut ajouter, grâce aux dépendances causales entre cette entrée et des sorties, les évènements qui, en complétant le cas de test, permettent de vérifier l'exécution effective de l'entrée.
   Après cette opération, iop-TC_1 et iop-TC_2 (dérivés à partir de conf-TC_1 et conf-TC_2) sont les cas de test d'interopérabilité unilatéraux basés sur S_1 et iop-TP_1, et sur S_2 et iop-TP_2.

5. Les cas de test d'interopérabilité unilatéraux iop-TC_1 et iop-TC_2 sont exécutés sur le SUT (System Under Test) composé des deux implémentations interconnectées. Cette exécution des deux cas de test donne lieu à deux verdicts locaux (ou unilatéraux). La combinaison de ces deux verdicts permet d'obtenir le verdict sur l'interopérabilité des deux implémentations relatif à l'objectif de test iop-TP.

Cette méthode et la méthode basée sur une approche globale "classique" (cf. section 4.3.2) ont été appliquées sur un exemple de spécifications de façon à confirmer à la fois leur équivalence en terme de verdicts et que la méthode bilatérale permet d'éviter le problème d'explosion du nombre d'états.

4.7 Quelques éléments sur l'exécution des cas de test d'interopérabilité et test distribué

Les cas de test d'interopérabilité bilatéraux obtenus avec la méthode décrite dans la section précédente suggèrent une approche distribuée du test d'interopérabilité. En effet, un cas de test d'interopérabilité bilatérale est composé de deux cas de test d'interopérabilité unilatérale. Chaque cas de test unilatéral doit être exécuté sur une des deux implémentations lors de l'interaction de celle-ci avec la deuxième implémentation. De plus, l'exécution de ces cas de test d'interopérabilité unilatérale ne nécessite pas de synchronisation entre les testeurs connectés à chacune des implémentations.

Un approche distribuée sans synchronisation du test d'interopérabilité est une approche du test d'interopérabilité considérant qu'un testeur différent est relié à chaque implémentation. Chacun de ces testeurs se focalise sur les évènements exécutés par l'IUT auquel il est connecté. Pour appliquer une approche distribuée (sans synchronisation) aux cas de test générés par la méthode bilatérale, il faut donc que les deux cas de test d'interopérabilité unilatérale composant un cas de test d'interopérabilité bilatérale puissent être exécutés en parallèle sur les deux implémentations lors de leur interaction.

Le problème est donc de savoir sous quelles conditions les deux cas de test unilatéraux générés peuvent être exécutés sous forme d'un cas de test d'interopérabilité distribuée. Les étapes de la génération de cas de test d'interopérabilité pouvant générer un résultat empêchant cette exécution distribuée sont plus particulièrement les étapes de dérivation des objectifs de test d'interopérabilité unilatérale (cf. cas des objectifs de test iop-TP_1 et iop-TP_2 dans l'exemple présenté dans la section 4.8.2.2) et la sélection du cas de test d'interopérabilité unilatérale dans le CTG généré. En effet, c'est lors de ces étapes qu'il y a un choix entre plusieurs traces de la spécification : lors d'un calcul de prédécesseur pour la dérivation d'un objectif de test unilatéral ou lors du choix entre plusieurs stimuli (par exemple) lors de la sélection d'un cas de test dans un CTG.
Prenons l'exemple de l'étape de sélection d'un cas de test d'interopérabilité dans le CTG généré. Lors de cette étape, dans une approche bilatérale, chacun des cas de test d'interopérabilité unilatérale est sélectionné dans le CTG correspondant. Ce choix peut ainsi renvoyer deux cas de test iop-TC_1 et iop-TC_2 tels que ces deux cas de test correspondent à des scénarios d'interaction des implémentations différents. Dans un tel cas, il est possible qu'aucun des verdicts locaux (ou un seul sur les deux) ne soit renvoyé lors des tests. Il faut donc que le choix du cas de test unilatéral iop-TC_i dans le CTG correspondant prenne en compte le choix qui a été fait pour iop-TC_j dans l'autre CTG. Ainsi, les cas de test d'interopérabilité obtenus sont des cas de test unilatéraux exécutables en parallèle (et donc avec une approche distribuée).

Lors de la complétion d'un cas de test unilatéral via le calcul des sorties dépendances causales d'une entrée dont on veut vérifier l'exécution effective, il peut être nécessaire de compléter également l'autre cas de test unilatéral pour permettre une exécution distribuée. En effet, il est possible que cette complétion ajoute d'autres évènements en plus de la sortie dépendance causale de l'entrée à vérifier (évènements à exécuter entre l'entrée à vérifier et chacune des sorties dépendances causales calculées). Si ces évènements sont d'autres entrées en provenance de l'autre implémentation, il faut également prévoir dans l'autre cas de test les envois de messages correspondants et les stimuli pouvant provoquer ces envois. De même, de façon à conserver la possibilité d'exécuter les cas de test générés de façon distribuée, toute modification de l'un des cas de test doit être réalisée en prenant en compte l'autre cas de test qui sera également complété si besoin.

Conclusion : Les cas de test d'interopérabilité obtenus avec une approche bilatérale suggèrent une approche distribuée du test d'interopérabilité. Cependant, les choix entre plusieurs traces des spécifications (correspondant toutes à l'objectif de test d'interopérabilité) peuvent renvoyer des cas de test d'interopérabilité unilatérale qui ne peuvent pas être exécutés en parallèle. Bien que l'approche soit bilatérale, ces choix doivent donc être faits en tenant compte des *deux* systèmes en interaction, de façon à pouvoir exécuter les cas de test générés en parallèle. Une approche distribuée sans synchronisation est alors utilisée pour l'exécution des tests.

4.8 Application

4.8.1 Implémentation de la méthode de génération de tests d'interopérabilité

La méthode bilatérale de génération de cas de test d'interopérabilité a été implémentée en utilisant les bibliothèques de la boîte à outils CADP [GLM01]. CADP (Construction and Analysis of Distributed Processes) est une boîte à outils pour la description de protocoles de communication et de systèmes distribués. CADP contient des outils pour stocker et manipuler des systèmes de transitions dans différents formats. CADP contient également divers outils pour la vérification, la simulation et le test. Parmi ces outils, l'outil de génération de tests de conformité TGV [JJ05] -qui est utilisé dans une des étapes de la méthode bilatérale- est connecté aux différents formats manipulables avec les bibliothèques de CADP.
Parmi les formats permettant de stocker et manipuler les systèmes de transitions, nous avons choisi d'utiliser un format proche sémantiquement du modèle des IOLTS considéré dans cette étude. C'est le format BCG[1] (Binary Coded Graphs) qui est une re-

[1]voir manuel en ligne : `http://www.inrialpes.fr/vasy/cadp/man/bcg.html`

présentation compacte d'automates visualisables ensuite grâce au format "BCG postscript". BCG est à la fois un format de fichier pour stocker des systèmes de transitions et un ensemble d'APIs (dont des fonctions pour la lecture, l'écriture ou le parcours des automates), de bibliothèque logicielles et d'outils (dont des outils de conversion vers d'autres formats ou d'extraction d'informations telles que nombre d'états, transitions, etc).
Le format BCG, parmi d'autres formats, peut être utilisé pour représenter les spécifications en entrée de TGV. Les objectifs de test doivent être représentés dans le format BCG ou dans le format AUT (Aldébaran, format permettant de représenter les mêmes systèmes de transition que le format BCG mais textuellement). Pour cette raison, nous avons choisi de représenter les spécifications des systèmes et les objectifs de test d'interopérabilité aux formats AUT ou BCG (obtenu par conversion de l'automate au format AUT). Les algorithmes et méthodes décrits dans les sections précédentes ont ainsi été implémentés en utilisant les fonctions de l'API BCG permettant de lire, d'écrire et de parcourir un automate au format BCG.

Nous avons implémenté à l'aide des outils de CADP à la fois la méthode bilatérale et une version de l'approche globale.
Pour la méthode bilatérale, nous avons tout d'abord implémenté la dérivation de l'objectif de test en deux objectifs de test unilatéraux. Puis nous avons appliqué les iop-TPs unilatéraux obtenus en entrée de TGV avec les spécifications correspondantes. Nous utilisons l'option "-ctg" de TGV[2] pour générer le graphe de test complet (ou CTG) au lieu d'un cas de test de conformité. Les résultats ont ensuite été modifiés comme décrit dans la section 4.4.1 de façon à générer les cas de test d'interopérabilité (avec application de l'algorithme utilisant les dépendances entre évènements quand nécessaire).
Pour la méthode globale, nous avons implémenté une version qui commence par le calcul de l'interaction des spécifications (cf. figure 4.2 de la section 4.3.2). Cette méthode utilise ensuite TGV pour générer les CTGs portant sur l'objectif de test d'interopérabilité globale et l'interaction des spécifications. Puis les sorties de TGV sont transformées de façon à obtenir le cas de test d'interopérabilité global relatif à l'objectif de test défini.

4.8.2 Application sur une version client/serveur d'un protocole de connexion

4.8.2.1 Spécifications et objectifs de test d'interopérabilité

Spécifications Les spécifications de ce protocole de demande de connexion en version client/serveur sont représentées dans la figure 4.7. Ce sont les spécifications qui avaient été présentées dans la section 3.7. S_1 est la spécification du client (qui demande

[2] cf. `http://www.inrialpes.fr/vasy/cadp/man/tgv.html`

la connexion) et S_2 celle du serveur (qui accepte ou refuse la connexion). Les interfaces de S_1 sont les interfaces $U1$ (interface supérieure) et $l1$ (interface inférieure). S_2 n'a qu'une interface qui est l'interface inférieure $l2$. Les interfaces inférieures $l1$ de S_1 et $l2$ de S_2 sont reliées entre elles et servent pour l'interaction des implémentations de ces spécifications. Cette dernière information est nécessaire (en entrée de nos algorithmes) pour permettre le calcul d'un évènement $\bar{\mu}$ à partir d'un évènement μ.

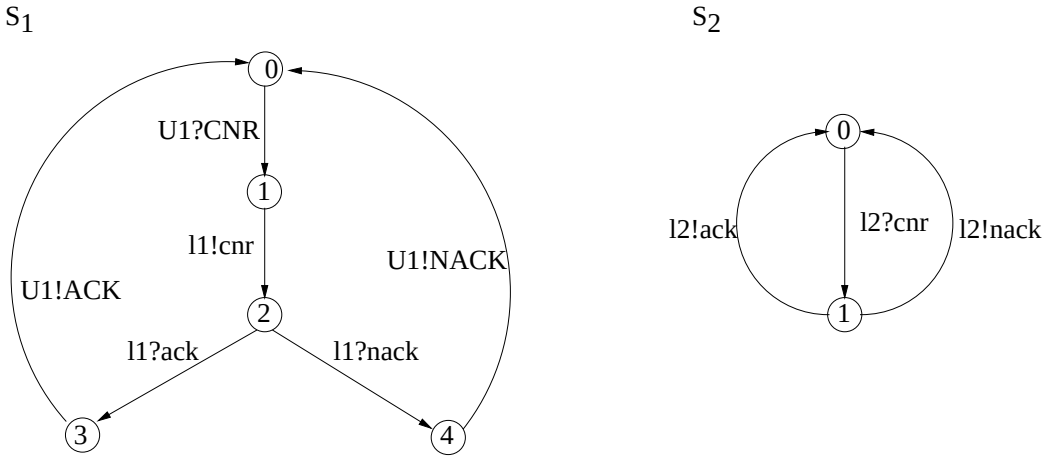


FIG. 4.7 – Spécifications S_1 et S_2 en mode client/serveur

Description succincte du protocole : S_1 reçoit une demande d'établissement de connexion de sa couche supérieure (évènement $U1?CNR$) et transmet cette demande à S_2 ($l1!cnr$ et $l2?cnr$). S_2 peut ensuite décider de répondre positivement (envoi de $l2!ack$) ou négativement (envoi de $l2!nack$). Cette réponse à la demande de connexion est reçue par S_1 (évènements $l1?ack$ ou $l1?nack$) qui la transmet à sa couche supérieure (évènements $U1?ACK$ ou $U1?NACK$).

Objectifs de test Nous considérons ici les objectifs de test d'interopérabilité de la figure 4.8. Ces trois objectifs de test d'interopérabilité sont notés iop-TP$_1$, iop-TP$_2$ et iop-TP$_3$.

Description des objectifs de test :

- Avec iop-TP$_1$ (figure 4.8(a)), on souhaite tester les situations où une demande de connexion CNR envoyée à I_1 (implémentation dérivée de S_1) sur son interface supérieure $U1$ aboutirait, après communication entre les deux systèmes, à une acceptation de cette connexion ACK transmise par I_1 à sa couche supérieure via l'interface $U1$.
- iop-TP$_2$ (figure 4.8(b)) a pour objectif de tester qu'une demande de connexion CNR envoyée à I_1 sur son interface supérieure $U1$ aboutit à une acceptation

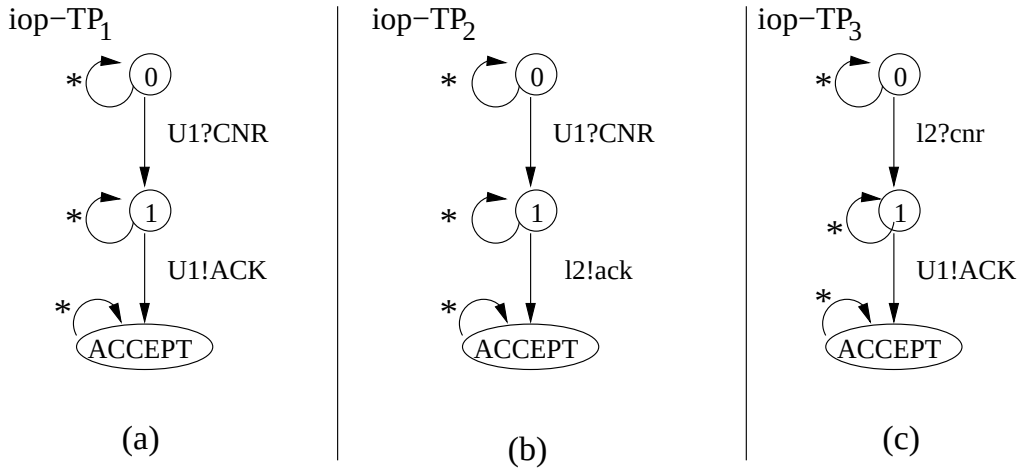


FIG. 4.8 – Objectifs de test d'interopérabilité pour un protocole de demande de connexion

de cette connexion ack envoyée par I_2 (implémentation dérivée de S_2) à I_1 via l'interface $l2$ de I_2.

– iop-TP$_3$ (figure 4.8(c)) servira à tester qu'une demande de connexion cnr reçue par I_2 sur son interface inférieure aboutit à une acceptation de cette connexion ACK transmise par I_1 à sa couche supérieure via l'interface $U1$.

4.8.2.2 Application de l'approche bilatérale

Nous appliquons ici l'approche bilatérale sur les trois objectifs de test d'interopérabilité de de la figure 4.8 et les spécifications S_1 et S_2 du protocole de demande de connexion en version client/serveur.

Dérivation des objectifs de test unilatéraux La première étape de cette approche est la dérivation de deux objectifs de test unilatéraux pour chaque objectif de test global. Pour cela, l'algorithme de la figure 4.4 est appliqué. Les objectifs de test unilatéraux sont donnés dans la figure 4.9.
iop-TP$_1^1$ et iop-TP$_1^2$ de la figure 4.9(a) sont les objectifs de test unilatéraux dérivés à partir de iop-TP$_1$ et des spécifications S_1 et S_2. De même, iop-TP$_2^1$ et iop-TP$_2^2$ de la figure 4.9(b) (resp. iop-TP$_3^1$ et iop-TP$_3^2$ de la figure 4.9(c)) sont les objectifs de test unilatéraux dérivés à partir de iop-TP$_2$ (resp. iop-TP$_3$).

Intéressons-nous aux résultats de cette dérivation des objectifs de test d'interopérabilité globaux en deux objectifs de test unilatéraux.

Dérivation des iop-TPs unilatéraux à partir de iop-TP$_1$ L'objectif de test iop-TP$_1$ est composé de deux évènements à exécuter sur des interfaces supérieures de I_1. La construction de iop-TP$_1^1$ est donc simple : iop-TP$_1^1$ = iop-TP$_1$ (cf figure

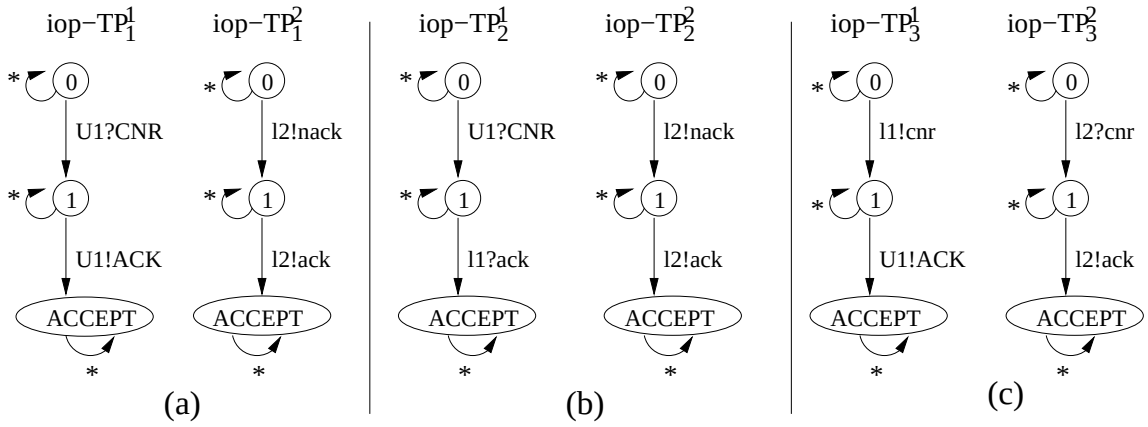


FIG. 4.9 – Objectifs de test unilatéraux dérivés à partir de iop-TP$_1$, iop-TP$_2$ et iop-TP$_3$

4.9(a)).
Par contre, pour la construction de iop-TP$_1^2$, deux appels à la procédure de recherche de prédécesseur sont nécessaires. Cette procédure (cf. algorithme de la figure 4.4 dans la section 4.4.2.1) recherche le premier prédécesseur exécutable sur une interface inférieure trouvé dans la spécification S_1. Pour l'évènement $U1?CNR$, le premier prédécesseur trouvé par l'algorithme (en amont de l'état initial de la spécification S_1) est $U1!NACK$ qui est également un évènement à exécuter sur une interface supérieure (évènement qui n'a pas de miroir au sens de la définition 3.1 de la section 3.3.1 du miroir d'un évènement de l'interaction). Le prédécesseur suivant est alors $l1?nack$: cet évènement a un iop-miroir qui est $l2!nack$. Le message ajouté dans iop-TP$_1^2$ est donc $l2!nack$. Le deuxième évènement de iop-TP$_1$ à traiter est $U1!ACK$ qui a pour prédécesseur $l1?ack$ d'iop-miroir $l2!ack$. On obtient donc un objectif de test unilatéral iop-TP$_1^2$ composé des évènements $l2!nack$ et $l2!ack$ (cf figure 4.9(a)).
Remarque : dans cet exemple, il y a plusieurs prédécesseurs possibles pour l'évènement $U1?CNR$ qui sont $U1!NACK$ et $U1!ACK$. Selon les cas, on peut choisir de traiter un seul des prédécesseurs (c'est ce qui est fait ici) ou de générer tous les iop-TP$_1^2$ possibles en fonctions des différents prédécesseurs rencontrés par l'algorithme.

Dérivation des iop-TPs unilatéraux à partir de iop-TP$_2$ L'objectif de test iop-TP$_2$ commence par l'évènement $U1?CNR$ tout comme iop-TP$_1$: les premiers évènements de iop-TP$_2^1$ et iop-TP$_2^2$ (transitions entre états 0 et 1) sont donc les mêmes que pour iop-TP$_1^1$ et iop-TP$_1^2$. L'autre évènement de iop-TP$_2$ est l'évènement $l2!ack$, exécutable sur une interface inférieure. Il n'est donc pas nécessaire de faire appel à la recherche de prédécesseur pour cet évènement : $l1?ack$ est ajouté à TP_2^1, et $l2!ack$ à iop-TP$_2^2$, ce qui donne les objectifs de test de la figure 4.9(b).

Dérivation des iop-TPs unilatéraux à partir de iop-TP$_3$ L'objectif de test iop-TP$_3$ commence par l'évènement $l2?cnr$. Les premiers évènements de iop-TP$_3^1$ et iop-TP$_3^2$ sont donc respectivement $l1!cnr$ et $l2?cnr$. Le deuxième évènement de iop-TP$_3$ est $U1!ACK$ qui, comme dans iop-TP$_1$, a pour prédécesseur $l1?ack$ de miroir $l2!ack$. Les évènements ajoutés à iop-TP$_3^1$ et iop-TP$_3^2$ sont donc respectivement $U1!ACK$ et $l2!ack$. Le résultat est dans la figure 4.9(c).

Remarque sur iop-TP$_1^2$ et iop-TP$_2^2$: Les objectifs de test d'interopérabilité unilatérale iop-TP$_1^2$ et iop-TP$_2^2$ ont été dérivés avec, lors de l'exécution de l'algorithme de la figure 4.4, une recherche d'un prédécesseur de l'évènement $U1?CNR$. A la suite de cette opération, l'évènement ajouté dans l'objectif de test en construction est l'iop-miroir d'un prédécesseur trouvé en amont de l'état initial de la spécification S_1. Les objectifs de test d'interopérabilité ainsi générés (iop-TP$_1^2$ et iop-TP$_2^2$) contiennent ainsi deux évènements qui ne sont pas dans la même trace d'exécution de la spécification S_2. Nous décrivons dans la suite (génération des cas de test) les problèmes que cela pose par rapport aux cas de test générés. Puis dans la section 4.9, nous nous intéressons aux situations pouvant conduire à une telle recherche de prédécesseur et aux modifications possibles sur l'algorithme de dérivation des objectifs de test unilatéraux permettant de résoudre ces problèmes.

Génération des cas de test Les objectifs de test unilatéraux de la figure 4.9 sont utilisés en entrée de TGV pour générer les CTGs qui seront ensuite transformés en cas de test d'interopérabilité. Les deux entrées de TGV sont tour à tour S_1 et iop-TP$_1^1$, S_2 et iop-TP$_1^2$, S_1 et iop-TP$_2^1$, S_2 et iop-TP$_2^2$, S_1 et TP$_3^1$, et S_2 et TP$_3^2$.
Dans un premier temps, nous allons nous intéresser aux CTGs et aux transformations appliquées sur ceux-ci pour l'exemple de l'objectif de test iop-TP$_2$. En effet, cet exemple d'iop-TP comporte des actions sur les deux entités, une sur l'interface supérieure et l'autre sur l'interface inférieure. De plus, comme iop-TP$_1^2$ se termine par une entrée, cet exemple permet de décrire l'exécution de toutes les étapes de la méthode, vérification des entrées comprise. Pour les deux autres objectifs de test (iop-TP$_1$ et iop-TP$_3$), nous donnerons et commenterons les cas de test d'interopérabilité obtenus après application des modifications.

Génération des cas de test unilatéraux pour iop-TP$_2$ Considérons les CTGs représentés dans la figure 4.10. Ces IOLTS représentent les CTGs dérivés par TGV avec en entrée, pour conf-TC$_2^1$, iop-TP$_2^1$ et la spécification S_1, et pour conf-TC$_2^2$, iop-TP$_2^2$ et la spécification S_2. La figure 4.10 donne la représentation de ces CTGs du point de vue des testeurs de conformité considérés par TGV : l'interface $TU1$ du testeur est

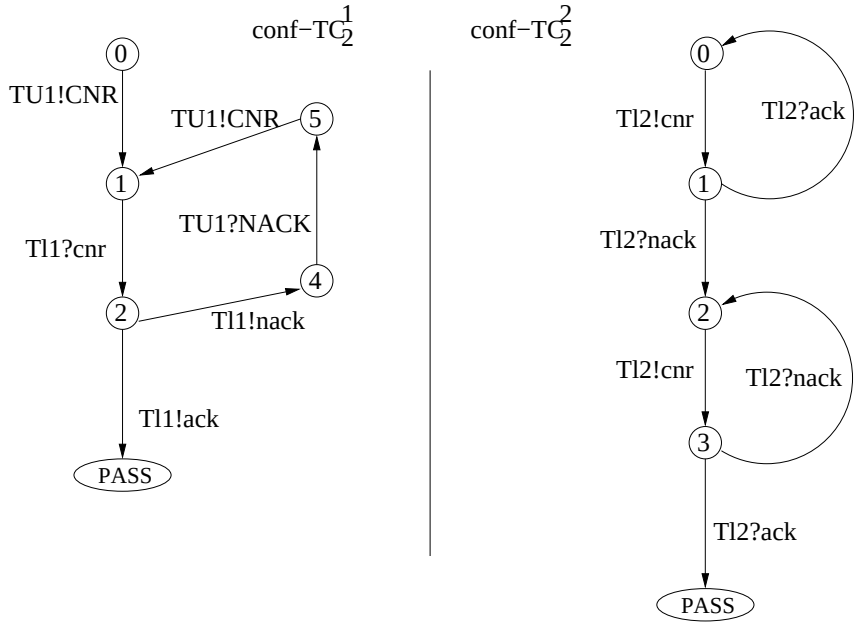


FIG. 4.10 – CTGs en sortie de TGV pour les objectifs de test iop-TP$_2^1$ et iop-TP$_2^2$

connectée à l'interface supérieure $U1$ de I_1 tandis que l'interface $Tl1$ (resp. $Tl2$) est connectée à l'interface inférieure $l1$ de I_1 (resp. $l2$ de I_2).
Sur l'exemple du CTG conf-TC$_2^1$, la principale différence entre le cas de test de conformité que renverrait TGV et le CTG représenté ici se situe dans l'état 2. En effet, en conformité, les évènements ack et $nack$ sont envoyés par des testeurs. Ces évènements sont donc contrôlables. Dans cette situation où l'objectif de test est l'évènement ack, un cas de test de conformité contrôlable ne contient donc pas la branche du CTG traitant de l'évènement $nack$ (sur conf-TC$_2^1$, la branche passant par les états 4 et 5). Dans le contexte du test d'interopérabilité, les évènements ack et $nack$ sont envoyés par l'autre implémentation, et ne sont donc pas contrôlables. Il faut donc garder les deux branches (ack et $nack$).

A partir du CTG obtenu (ou plutôt du cas de test contrôlable pour l'interopérabilité choisi dans le CTG), la première phase de modifications consiste à adapter les types d'évènements au contexte du test d'interopérabilité. En effet (cf. section 4.2.2), les types d'évènements possibles dans un contexte de test d'interopérabilité ne sont pas les mêmes qu'en conformité : un testeur d'interopérabilité peut envoyer des messages vers une interface supérieure, recevoir des messages en provenance d'une telle interface, et observer les évènements de l'interaction sur les interfaces inférieures. Nous notons ici $T1$ le testeur connecté à l'interface supérieure $U1$ de I_1. Après modifications des évènements, les cas de test d'interopérabilité obtenus sont représentés dans la figure 4.11.

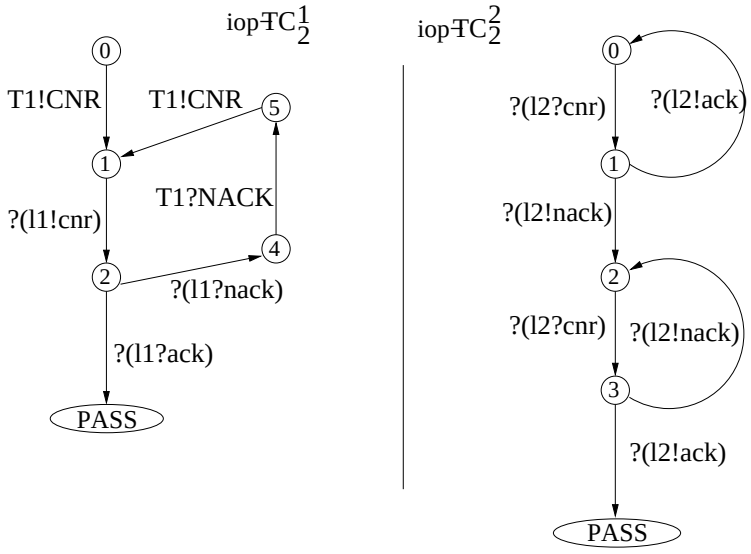


FIG. 4.11 – Cas de test d'interopérabilité unilatéraux iop-TC$_2^1$ et iop-TC$_2^2$

Remarque: Par soucis de simplification, nous n'avons pas représenté dans les cas de test les transitions vers les états $Fail$. Notons simplement que tout message envoyé par une implémentation sur une de ses interfaces et non décrit dans le cas de test conduira à un verdict $Fail$. Les blocages ne sont pas non plus représentés sur les cas de test des figures 4.11 et 4.12. Nous avons choisi sur ces figures de nous concentrer sur les autres types d'évènements de façon à alléger nos figures et à permettre plus facilement la comparaison des cas de test en terme de verdict, ce qui est notre objectif dans cette section. Cependant, comme le critère d'interopérabilité bilatérale et l'outil TGV gèrent les blocages, les cas de test d'interopérabilité effectivement générés contiennent également cette information.

Dépendances causales et cas de test générés pour iop-TP$_2$ Les cas de test d'interopérabilité représentés sur la figure 4.11 ne sont pas complets par rapport à l'objectif de test. En effet, le verdict $Pass$ de iop-TC$_2^1$ est associé à l'évènement $?(l1?ack)$ qui est l'observation d'une entrée sur l'interface inférieure $l1$ de I_1. Cependant, cet évènement n'est pas observable ; il faut donc appliquer ici l'algorithme de calcul des sorties dépendances causales. Le verdict $Pass$ de la transition de label $?(l1?ack)$ devient alors un verdict temporaire (en attendant le verdict final donné à la suite de la vérification de l'exécution de l'entrée). Dans cet exemple, l'entrée $l1?ack$ n'a qu'une sortie dépendance causale, la sortie $U1!ACK$. Cet évènement, ou plutôt vu du testeur l'évènement $UT1?ACK$, est donc ajouté en fin de cas de test et le verdict $PASS$ définitif lui est

associé.

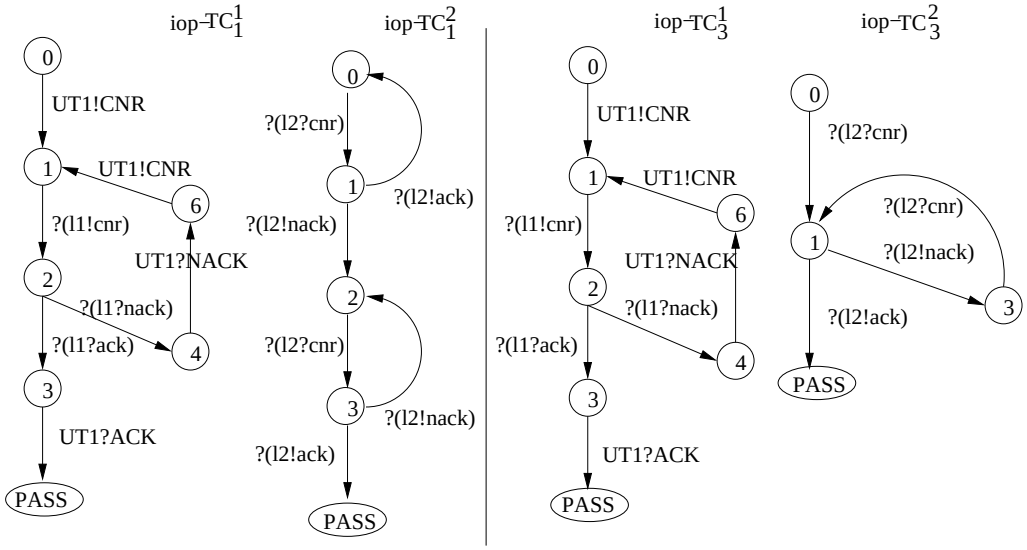


FIG. 4.12 – Cas de test d'interopérabilité générés par méthode bilatérale

Cas de test unilatéraux pour iop-TP$_1$ et iop-TP$_3$ La figure 4.12 donne les cas de test d'interopérabilité générés par l'approche bilatérale pour les objectifs de test iop-TP$_1$ et iop-TP$_3$. Les mêmes étapes que pour obtenir iop-TC$_2^1$ et iop-TC$_2^2$ à partir de iop-TP$_2$ ont été nécessaires. Cependant, les cas de test générés pour iop-TP$_1$ et iop-TP$_3$ ne se terminent pas sur une entrée sur une interface inférieure : il n'a donc pas été nécessaire de compléter les cas de test à l'aide de dépendances entre évènements pour ces deux exemples.
La différence entre les objectifs de test d'interopérabilité iop-TP$_1$ et iop-TP$_3$ se situait dans le premier évènement de ces objectifs de test : $U1?CNR$ pour iop-TP$_1$ et $l2?cnr$ pour iop-TP$_3$. Par contre, le deuxième (et dernier) évènement décrit dans ces objectifs de test est le même : $U1!ACK$.

Intéressons-nous d'abord aux cas de test unilatéraux iop-TC$_1^1$ et iop-TC$_3^1$ des implémentations de S_1. Ces deux cas de test d'interopérabilité unilatérale sont identiques. En effet, les objectifs de test unilatéraux générés pour S_1 à partir de iop-TP$_1$ et iop-TP$_3$ (cf. figure 4.9) correspondent respectivement aux traces $U1?CNR.U1!ACK$ et $l1!cnr.U1!ACK$. Le dépendance entre les évènements $U1?CNR$ et $l1!cnr$ sur S_1 conduit ainsi à deux cas de test identiques à partir de ces deux objectifs de test unilatéraux.

Intéressons-nous maintenant aux cas de test unilatéraux iop-TC$_1^2$ et iop-TC$_3^2$ des implémentations de S_2. La dérivation de iop-TP$_1^2$ a nécessité la recherche d'un prédécesseur dans les traces en amont de l'état initial de la spécification S_1. Cette recherche de prédécesseur a ajouté l'évènement $l2!nack$ à iop-TP$_1^2$. Elle a aussi pour conséquence la création, dans iop-TC$_1^2$, d'une trace correspondant à un parcours de S_2 avec retour à l'état initial, via $l2!nack$.
La dérivation de iop-TP$_3^2$ n'a pas entraîné cette recherche d'un prédécesseur pour le premier évènement. Il n'y a donc pas dans iop-TC$_3^2$ ce parcours complet de S_2. C'est la raison pour laquelle l'exécution de iop-TC$_1^2$ jusqu'à un état $Pass$ nécessite au moins quatre transitions alors que le minimum de transitions à exécuter pour amener iop-TC$_3^2$ dans un état $Pass$ est seulement de deux.

Cette ajout dans iop-TP$_1^2$ d'un parcours d'une trace de S_2 partant et retournant dans l'état initial de S_2 a des conséquences sur l'exécution en parallèle des deux cas de test iop-TP$_1^1$ et iop-TP$_1^2$. En effet, cette boucle a été introduite dans iop-TP$_1^2$, mais il n'existe pas de boucle correspondante sur iop-TP$_1^1$. Ces deux cas de test ne sont donc pas exécutables en parallèle, contrairement à iop-TP$_3^1$ et iop-TP$_3^2$ (pour lesquels aucune boucle du même type n'a été introduite) qui peuvent être exécutés en parallèle sur I_1 et sur I_2 lors de leur interaction.

Cas de test d'interopérabilité bilatérale générés et test distribué Nous avons vu dans la section 4.7 que les cas de test générés par l'approche bilatérale peuvent être des cas de test exécutables via une approche distribuée du test d'interopérabilité.
Dans notre exemple, les cas de test iop-TC$_3^1$ et iop-TC$_3^2$ peuvent être exécutés en même temps sur les implémentations de S_1 et S_2 pendant leur interaction. Cette exécution correspond ainsi à l'exécution du cas de test d'interopérabilité bilatérale sur le SUT composé des deux implémentations en interaction.
Par contre, pour les cas de test générés à partir de iop-TP$_1$ et iop-TP$_2$, cette approche distribuée de l'exécution des tests pose problème. Ceci est dû à l'étape de dérivation des objectifs de test unilatéraux, et plus particulièrement à l'étape de recherche de prédécesseur. En effet, cette recherche de prédécesseur a renvoyé un prédécesseur situé en amont de l'état initial de la spécification S_1. La conséquence est que les cas de test iop-TC$_1^2$ et iop-TC$_2^2$ contiennent un "préambule" réalisant un parcours de S_1 avec retour à l'état initial avant les évènements correspondant effectivement à l'objectif de test global iop-TP$_1$ ou iop-TP$_2$. L'équivalent de ce préambule n'existant pas dans les cas de test iop-TC$_1^1$ et iop-TC$_2^1$, l'exécution distribuée de ces cas de test bilatéraux n'est pas possibles. Cependant, les corrections apportées à l'algorithme de dérivation des objectifs de test unilatéraux (cf. section 4.9) permettent également de générer des cas de test exécutables de façon distribuée.

Nous avons décrit la génération, via une approche bilatérale, des cas de test d'in-

teropérabilité pour les objectifs de test iop-TP_1, iop-TP_2 et iop-TP_3. Dans la section suivante, nous décrivons la génération de cas de test basés sur une approche globale et les mêmes objectifs de test, et nous comparons les résultats de ces deux approches en terme de verdict.

4.8.2.3 Approche classique

L'approche globale de génération de test est ici appliquée sur les spécifications de la figure 4.7 et les objectifs de test de la figure 4.8.

Génération des cas de test La première étape de cette approche telle que représentée dans la section 4.3.2 est la construction d'un modèle de l'interaction des spécifications. Pour les spécifications S_1 et S_2 de la figure 4.7, cette interaction est calculable sans explosion combinatoire du nombre d'états et est représentée dans la figure 4.13. Cette interaction ne dépend pas ici de la taille des files d'attentes modélisant le caractère asynchrone de la communication. En effet, dans ces spécifications, chaque message envoyé sur une interface inférieure est reçu par l'autre système avant toute nouvelle exécution d'un évènement par ce récepteur.

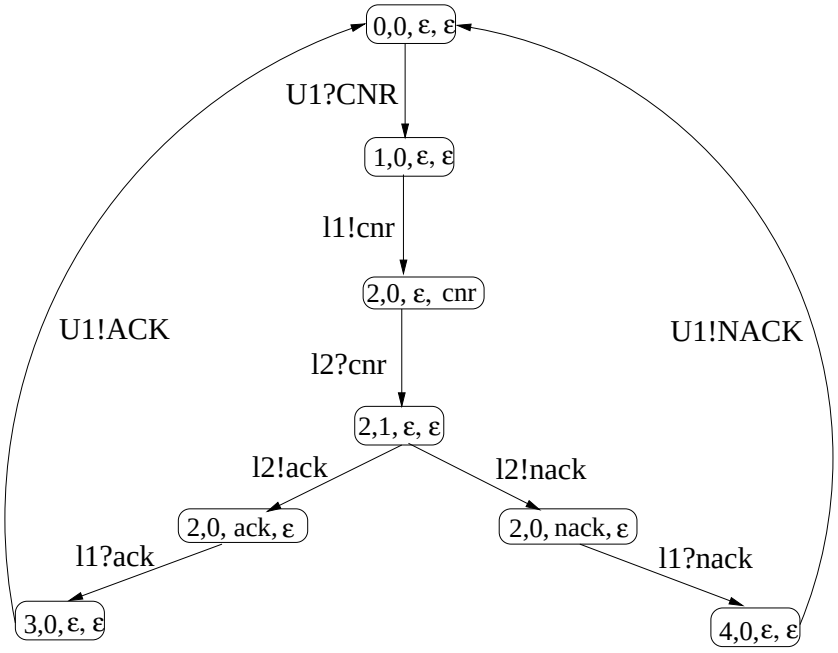


FIG. 4.13 – Interaction asynchrone $S_1 \|_{\mathcal{A}} S_2$ des spécifications S_1 et S_2

Cette interaction est ensuite utilisée comme entrée de TGV avec un des objectifs de test d'interopérabilité de la figure 4.8. De même que pour l'approche bilatérale, et pour les mêmes raisons (non-contrôle des interfaces inférieures en interopérabilité), la

sortie de TGV considérée est non pas un cas de test mais le CTG contenant tous les cas de test permettant d'exécuter l'objectif de test en entrée.
De même que dans le contexte bilatéral, des modifications doivent être apportées sur les sorties de TGV : modification des types d'évènements, dépendances causales pour vérification des entrées (particulièrement dans le cas où l'objectif de test global se termine par une entrée), choix d'un cas de test contrôlable en interopérabilité dans le CTG, etc. Les cas de test d'interopérabilité globaux sont représentés dans la figure 4.14. Aucun de ces cas de test d'interopérabilité globaux ne se termine par un évènement de type entrée car les objectifs de test dont ils dérivent ne se terminent pas non plus par une entrée. Il n'est donc pas nécessaire de compléter les cas de test via les dépendances causales entre entrées et sorties.

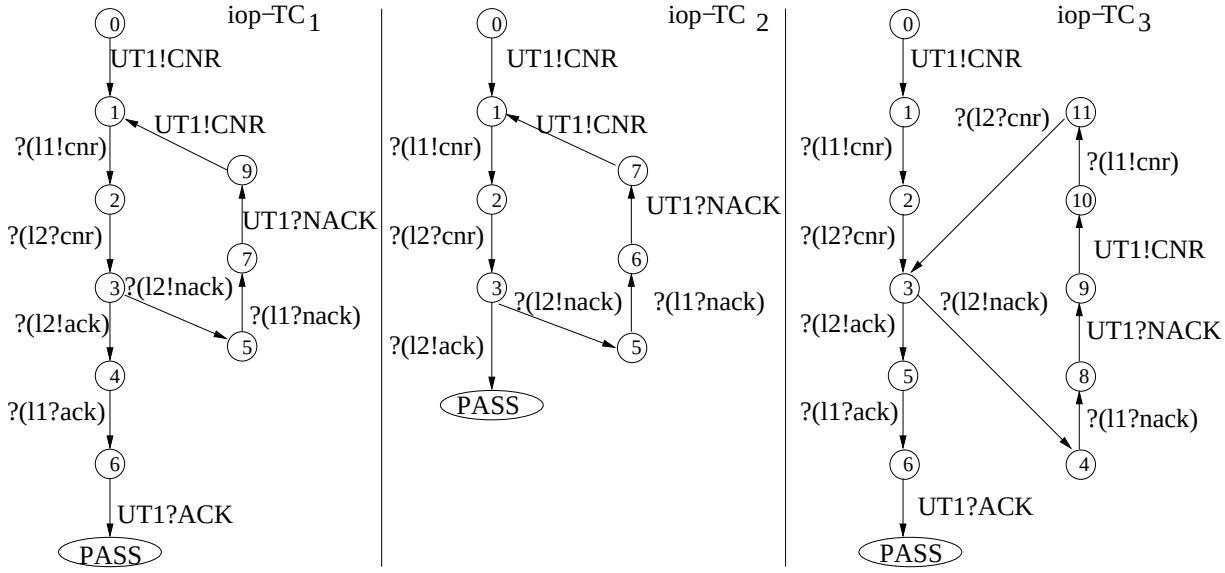


FIG. 4.14 – Cas de test d'interopérabilité globaux basés sur $S_1 \|_{\mathcal{A}} S_2$ et les objectifs de test iop-TP$_1$, iop-TP$_2$ et iop-TP$_3$

Comparaison avec les cas de test générés par l'approche bilatérale Comparons maintenant les cas de test globaux de la figure 4.14 avec les cas de test bilatéraux (composés de deux cas de test unilatéraux) des figures 4.11 et 4.12. Comme il y a équivalence en terme de détection de la non-interopérabilité du critère d'interopérabilité bilatérale totale et du critère d'interopérabilité globale totale, les verdicts obtenus après les mêmes exécutions d'évènements doivent être les mêmes. L'objectif de la comparaison des cas de test est de vérifier cette équivalence en terme de verdicts.
Pour les cas de test d'interopérabilité dérivés de l'objectif de test iop-TP$_3$, cette équivalence est aisément vérifiable. En effet, on a iop-TC$_3^1 \|_{\mathcal{A}}$ iop-TC$_3^2$ = iop-TC$_3$.
Pour les cas de test dérivés de iop-TP$_1$ et iop-TP$_2$, nous n'avons pas cette égalité

cette égalité à cause du parcours de S_2 introduit dans iop-TC$_1^2$ et iop-TC$_2^2$ par la recherche d'un prédécesseur de $U1?CNR$. Mais les verdicts peuvent être comparés en étudiant les traces des cas de test. On peut ainsi remarquer que $Traces$(iop-TC$_i$)$/S_j \subseteq Traces$(iop-TC$_i^j$). Plus particulièrement $Traces$(iop-TC$_1$)$/S_1 = Traces$(iop-TC$_1^1$) et $Traces$(iop-TC$_2$)$/S_1 = Traces$(iop-TC$_2^1$). Pour les traces de iop-TC$_1/S_2$ et iop-TC$_2/S_2$, les différences proviennent du parcours de S_2 introduit dans iop-TC$_1^2$ et iop-TC$_2^2$ par la recherche d'un prédécesseur de $U1?CNR$. On peut cependant remarquer que les situations de non-interopérabilité qui seraient détectées avec iop-TC$_1$ et iop-TC$_2$ le sont avec les cas de test bilatéraux composés respectivement de iop-TC$_1^1$ et iop-TC$_1^2$, et de iop-TC$_2^1$ et iop-TC$_2^2$. Quand la méthode bilatérale modifiée (de façon à ne pas rechercher de prédécesseur en amont de l'état initial d'une spécification) est appliquée, les cas de test générés différents de ceux générés ici sont iop-TC$_1^2$ et iop-TC$_2^2$. Nous réétudierons donc l'équivalence en terme de verdict pour ces exemples dans la section 4.9, section dans laquelle les modifications à apporter à l'algorithme de dérivation des objectifs de test unilatéraux sont décrites.

Ainsi, cette application confirme l'équivalence entre approche bilatérale et approche globale en terme de détection de la non-interopérabilité.

4.8.3 Application sur une version complète du protocole de connexion

L'application sur l'exemple précédent a été réalisée de façon à confirmer que notre méthode dite bilatérale permet de détecter au moins autant de situation de non-interopérabilité que l'on détecterait à partir des mêmes objectifs de test et de l'approche classique (ou globale). Dans cette section, nous appliquons ces méthodes sur un autre exemple de façon à montrer comment l'approche bilatérale permet d'éviter le problème d'explosion du nombre d'états rencontré avec l'approche globale.

Spécifications et objectifs de test Nous considérons dans cette section la spécification de la figure 4.15 qui représente le même protocole de demande de connexion, mais donne aux deux systèmes la possibilité d'être client et serveur. Les objectifs de test sont les mêmes que précédemment : cf. figure 4.8.

Dans la suite, nous utilisons S_1 (resp. S_2) pour désigner la spécification servant de base à l'implémentation I_1 (resp. I_2) de l'architecture de test d'interopérabilité. S_1 (resp. S_2) est une version de la spécification S de la figure 4.15 telle que l'interface l de S est notée $l1$ (resp. $l2$) dans S_1 (resp. S_2) ou dans l'implémentation correspondante I_1 (resp. I_2). Les interfaces supérieures des entités sont numérotées de la même façon. De plus, un lien relie les interfaces inférieures $l1$ et $l2$.

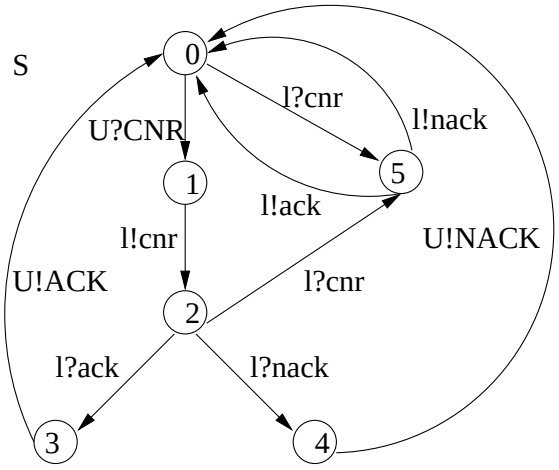


FIG. 4.15 – Spécification S du mode complet client et serveur

Application de l'approche bilatérale La dérivation des cas de test d'interopérabilité à l'aide de la méthode bilatérale se fait via les mêmes étapes que dans l'exemple précédent. Nous donnons tout d'abord les objectifs de test unilatéraux générés (cf. figure 4.16). La différence entre ces objectifs de test unilatéraux et ceux du protocole en mode client/serveur provient de la recherche de prédécesseur pour l'évènement $U1?CNR$: cette opération avait permis d'ajouter à iop-TP$_1^2$ et iop-TP$_2^2$ l'évènement $l2!nack$ (cf. figure 4.9), alors qu'ici l'évènement ajouté est $l2?ack$ (cf. figure 4.16) : l'évènement $l1!ack$ est le premier prédécesseur de $U1?CNR$ exécutable sur une interface inférieure trouvé par l'algorithme de recherche de prédécesseur. Pour les cas de test unilatéraux, nous ne représentons figure 4.17 qu'un seul exemple, celui basé sur l'objectif de iop-TP$_3$. En effet, c'est l'exemple pour lequel la dérivation des objectifs des test unilatéraux n'est pas basée sur une recherche de prédécesseur en amont de l'état initial de la spécification $S_1 = S$.

La figure 4.17 donne les cas de test unilatéraux pour iop-TP$_3$, ou plutôt les CTGs renvoyés et réadaptés pour le test d'interopérabilité. La différence entre les CTGs et les cas de test se situe dans l'état 0 qui laisse le choix entre l'envoi d'un stimulus par le testeur et l'exécution d'une entrée sur une interface inférieure. L'un de ces évènements est contrôlé par le testeur, l'autre non. Il est donc possible d'extraire un cas de test d'interopérabilité des CTGs qui soit plus simple que ceux de la figure 4.17, mais à condition de rester cohérent dans la simplification. Dans cet exemple, la partie de iop-TC$_3^2$ comprenant les états 1, 3, 6 et 7 (et les transitions partant ou arrivant dans ces états) peut être supprimée si on décide de ne pas appliquer le stimulus $UT2!CNR$ et d'attendre que I_2 se débloque grâce à un message de I_1 (message cnr). Dans ce cas, l'état 2 de iop-TC$_3^1$ et les transitions correspondantes peuvent également être supprimés : cette partie du cas de test ne sera pas exécutée si I_2 ne prend pas l'initiative de la demande de connexion. Toute autre simplification sur ces cas de test (ou simplification

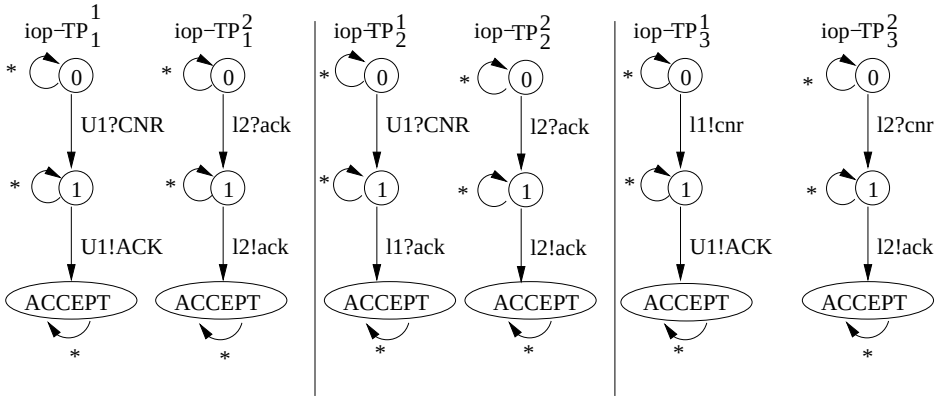


FIG. 4.16 – Objectifs de test unilatéraux dérivés à partir de iop-TP$_1$, iop-TP$_2$ et iop-TP$_3$

de iop-TC$_3^1$ sans la simplification de iop-TC$_3^2$) risque d'entraîner des incohérences lors de l'exécution en parallèle de ces deux cas de test. Notons que ces CTGs peuvent aussi correspondre directement à un cas de test d'interopérabilité : c'est alors le testeur qui décide pendant l'exécution du test s'il applique le stimulus dans l'état initial ou s'il se place en attente d'un évènement. Cependant, le "démarrage" du test nécessite dans cet exemple qu'un stimulus soit appliqué sur au moins une des deux implémentations.

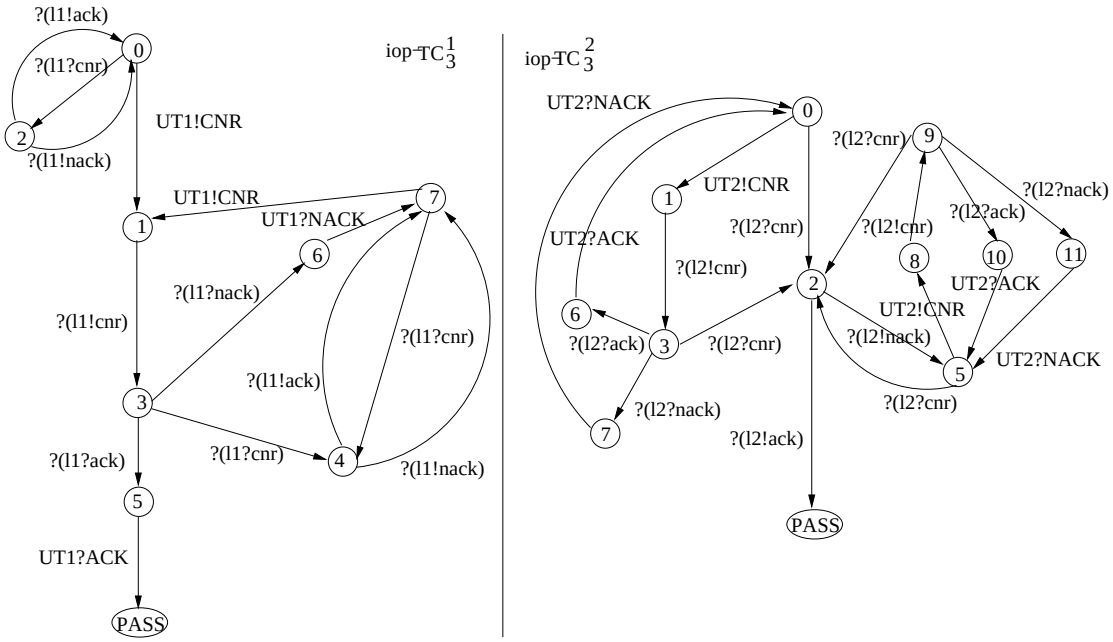


FIG. 4.17 – Cas de test unilatéraux générés pour iop-TP$_3$

L'application de l'approche bilatérale a ainsi permis de générer deux cas de test unilatéraux pour chacun des trois objectifs de test définis. Intéressons-nous maintenant

à l'application de l'approche globale sur ces mêmes exemples.

Approche classique Dans cet exemple, le nombre d'états de l'interaction des spécifications et des cas de test d'interopérabilité globaux dépend de la borne des files d'attente FIFO en entrée des interfaces inférieures des systèmes. Par exemple, en bornant les files d'attente des interfaces $l1$ et $l2$ des systèmes S_1 et S_2 avec une seule place, le modèle pour l'interaction des spécifications que nous avons calculé est composé de 454 états et 1026 transitions.
Nous récapitulons dans le tableau suivant, pour les trois exemples d'objectifs de test iop-TP$_1$, iop-TP$_2$ et iop-TP$_3$, les nombres d'états et transitions (sous la forme "nombre d'états/nombre de transitions") :

- des cas de test unilatéraux iop-TC$_i^1$ et iop-TC$_i^2$ (i est le numéro de l'objectif de test global) dérivés par la méthode bilatérale avec les objectifs de test de la figure 4.16 et les spécifications $S_1 = S$ et $S_2 = S$
- de l'interaction des cas de test iop-TC$_i^1$ et iop-TC$_i^2$, correspondant à la vue globale de l'exécution des deux cas de test d'interopérabilité unilatéraux sur le SUT composé des deux IUTs I_1 et I_2
- des cas de test TC_i globaux basés sur l'interaction des spécifications

	iop-TP$_1$ (i=1)	iop-TP$_2$ (i=2)	iop-TP$_3$ (i=3)
iop-TC$_i^1$ (iop-TC unilatéral)	9/17	8/16	9/17
iop-TC$_i^2$ (iop-TC unilatéral)	13/24	13/24	12/22
iop-TC$_i^1 \|\|_{\mathcal{A}}$ iop-TC$_i^2$	851/2363	752/2122	405/1146
iop-TC$_i$ (iop-TC global)	525/1108	347/741	548/1151

Pour cet exemple d'application et sur un système avec une mémoire de 1 Gb et un processeur de 2GHz, nous n'avons pas pu calculer l'interaction $S\|_{\mathcal{A}}S$ des spécifications à partir d'une taille des files d'attente bornée à 4 places. Cette taille de file peut correspondre à une situation où au maximum quatre demandes de connexion peuvent être en attente de traitement. Nous donnons donc ici les chiffres (nombres d'états et transitions) des objets manipulés lorsque la taille des files d'attente est bornée à 3 places. Dans ce cas, l'interaction des spécifications est composée de 47546 états et 114158 transitions.

	iop-TP$_1$	iop-TP$_2$	iop-TP$_3$
iop-TC$_i^1$	9/17	8/16	9/17
iop-TC$_i^2$	13/24	13/24	12/22
iop-TC$_i^1 \|\|_{\mathcal{A}}$ iop-TC$_i^2$	19546/57746	19468/57614	19405/57386
iop-TC$_i$	54435/120400	18014/40793	54456/120443

On peut remarquer à partir de cet exemple :

1. nous faisons face à un problème d'explosion du nombre d'états en appliquant

l'approche globale sur une spécification a priori simple (composée de 6 états et 10 transitions). Le format de l'interaction des cas de test unilatéraux confirme d'ailleurs que beaucoup de scénarios différents peuvent être observés si une vue globale du SUT est préférée.

2. l'approche bilatérale permet de générer des cas de test même lorsque l'approche globale n'a pas pu être appliquée. De plus, ces cas de test ne dépendent pas de la taille des files d'attente des systèmes en interaction.

4.8.4 Conclusion

L'application des approches bilatérales et globales sur différentes spécifications d'un protocole de demande de connexion et trois objectifs de test d'interopérabilité définis pour ces spécifications permet de conclure que :

1. en terme de détection de la non-interopérabilité, l'application des deux méthodes confirme l'équivalence des critères d'interopérabilité globale et bilatérale. C'est-à-dire que, bien que les testeurs observent/contrôlent les deux implémentations séparément pendant leur interaction, l'approche bilatérale du test d'interopérabilité permet de détecter toutes les situations de non-interopérabilité qui seraient détectées en se basant sur une vue globale du SUT composé des deux implémentations.

2. l'algorithmique basée sur les dépendances causales permet de compléter les cas de test générés, en particulier dans le cas où les objectifs de test se terminent par une entrée sur une interface inférieure de l'implémentation. Ceci permet de vérifier que les entrées (qui sont des évènements non-observables) sont effectivement exécutées. L'un des objectifs du test d'interopérabilité est en effet de vérifier que les implémentations communiquent correctement, ce qui implique que les messages envoyées par une implémentation doivent être effectivement reçues par l'autre implémentation.

3. l'approche bilatérale permet de générer des cas de test même lorsque l'approche globale fait face à un problème d'explosion combinatoire du nombre d'états. Ainsi, l'approche bilatérale de génération de cas de test d'interopérabilité permet d'éviter cette explosion combinatoire du nombre d'états rencontrée lors de la construction de l'interaction des spécifications dans les méthodes classiques et qui peut empêcher la génération des cas de test.

4.9 Améliorations de l'approche bilatérale

4.9.1 Modifications de l'algorithme de dérivation des objectifs de test d'interopérabilité unilatérale

Nous avons vu lors de l'application de l'approche globale, et en particulier de l'exécution de l'algorithme de dérivation des objectifs de test d'interopérabilité unilatérale (cf. figure 4.4), qu'il arrivait que la recherche d'un prédécesseur d'un évènement de l'iop-TP global soit faite en amont de l'état initial de la spécification correspondante. Comme observé dans les objectifs de test unilatéraux iop-TP$_1^2$ et iop-TP$_2^2$ (figure 4.9), et dans les cas de test unilatéraux correspondants iop-TC$_1^2$ et iop-TC$_2^2$ (figures 4.11 et 4.12), cette situation génère des objectifs de test unilatéraux non-satisfaisants (et donc des cas de test unilatéraux également non-satisfaisants). En effet, ces cas de test intègrent un parcours de la spécification S_2 qui n'est pas dans l'objectif de test initial.

Étudions les circonstances qui peuvent conduire à de tels objectifs et cas de test unilatéraux. L'opération en cause est l'exécution de la fonction de recherche de prédécesseur, plus particulièrement lorsque cette opération est réalisée pour un évènement qui est le premier de l'iop-TP global. Dans le cas où l'évènement traité n'est pas le premier de l'iop-TP, la recherche de prédécesseur est en effet bornée par la connaissance des évènements déjà ajoutés dans les iop-TPs unilatéraux. Par contre, lorsque l'évènement traité est le premier de l'iop-TP global, il n'y a pas de borne pour la recherche de prédécesseur. Ainsi, si aucun prédécesseur exécutable sur une interface inférieure n'a été trouvé avant l'état initial de la spécification, la recherche de prédécesseur continue en amont de cet état initial. Un iop-miroir d'un prédécesseur trouvé en amont de l'état initial de la spécification est ainsi ajouté à l'iop-TP unilatéral en construction. Cet iop-TP unilatéral est ensuite utilisé pour générer un cas de test unilatéral. Cette génération de test recherche les traces de la spécification correspondant à l'objectif de test. Comme le premier évènement de l'objectif de test est un évènement exécuté dans des traces permettant de revenir dans l'état initial, un parcours complet de la spécification est généré au début du cas de test.

La solution pour éviter de générer ce parcours complet de la spécification dans un cas de test est de borner la recherche de prédécesseur. Dans la fonction $add\_precursor$, la recherche n'est actuellement pas bornée lorsque l'autre objectif de test en construction (notée iop-TP' dans la fonction) ne contient pas d'évènements. Cela signifie que l'évènement μ dont on recherche le prédécesseur est le premier évènement de l'objectif de test global. Cette recherche de prédécesseur étudie chacun des évènements précédent l'évènement μ. Si un évènement exécutable sur une interface inférieure est trouvé, l'iop-miroir de cet évènement est ajouté à l'objectif de test unilatéral en construction. Sinon, on continue la recherche arrière. Il faut donc ici borner cette recherche arrière.

La condition pour borner la recherche de prédécesseur est la suivante : si l'état q de S, dont on doit étudier les évènements étiquetant les transitions d'entrée pour trouver un prédécesseur exécutable sur une interface inférieure, est l'état initial de S, alors la recherche de prédécesseur s'arrête. Ainsi, la recherche de prédécesseur s'arrête avant d'avoir trouvé un évènement à ajouter au cas de test unilatéral. Le cas de test unilatéral aura alors un évènement de moins que l'objectif de test global. C'est cette version modifiée de l'algorithme qui est appliquée sur l'exemple d'objectif de test iop-TP$_1$ dans la section 4.9.2.
Remarque: La recherche de prédécesseur en amont de l'état initial effectuée dans la version précédente de l'algorithme avait pour résultat d'insérer une trace σ en début d'un des cas de test unilatéraux. Cette trace σ correspondait à une boucle dans la spécication correspondante : elle contenait une exécution de cette spécification partant de son état initial et retournant dans ce même état. Les évènements du cas de test correspondant à la suite de σ correspondaient ensuite aux traces du cas de test global généré pour le même objectif de test global. Ainsi, la borne de la recherche de prédécesseur permet ainsi de supprimer cette boucle et donc d'obtenir des cas de test équivalents à ceux qui seraient obtenus avec l'approche globale.

4.9.2 Application du nouvel algorithme sur le protocole de connexion en mode client/serveur

Nous appliquons dans cette section la version modifiée de la méthode bilatérale sur l'exemple du protocole de demande de connexion en mode client/serveur (spécifications S_1 et S_2 de la figure 4.7). Nous utilisons les objectifs de test iop-TP$_1$ et iop-TP$_2$ comme exemples. En effet, ces objectifs de test correspondent aux exemples pour lesquels une recherche de prédécesseur en amont de l'état initial de S_1 était réalisée. Les objectifs de test unilatéraux obtenus sont ceux de la figure 4.18.

Les objectifs de test iop-TP$_1^1$ et iop-TP$_2^1$ sont les mêmes avec les deux versions de l'algorithme. En effet, il n'y avait pas eu de recherche de prédécesseur en amont de l'état initial d'une spécification pour ces objectifs de test.
Les objectifs de test iop-TP$_1^2$ et iop-TP$_2^2$ générés avec la version modifiée de l'algorithme ne contiennent qu'un seul évènement. En effet, la recherche de prédécesseur étant bornée dans cette version de l'algorithme, aucun message n'a été ajouté dans iop-TP$_1^2$ et iop-TP$_2^2$ (objectifs de test unilatéraux de S_2) pour l'évènement $U1?CNR$ de iop-TP$_1$ et iop-TP$_2$.

Les cas de test unilatéraux générés à partir de ces nouveaux objectifs de test sont représentés dans la figure 4.19 (avant calcul des sorties dépendances causales de $?(l1?ack)$ pour iop-TC$_2^1$). Les cas de test iop-TC$_1^2$ et iop-TC$_2^2$ sont différents de ceux qui avaient été obtenus avec la première version de l'algorithme. Surtout, ces cas de test sont plus cohérents par rapport aux objectifs de test initiaux iop-TP$_1$ et iop-TP$_2$.
De plus, on peut remarquer que ces cas de test bilatéraux (composés chacun de deux

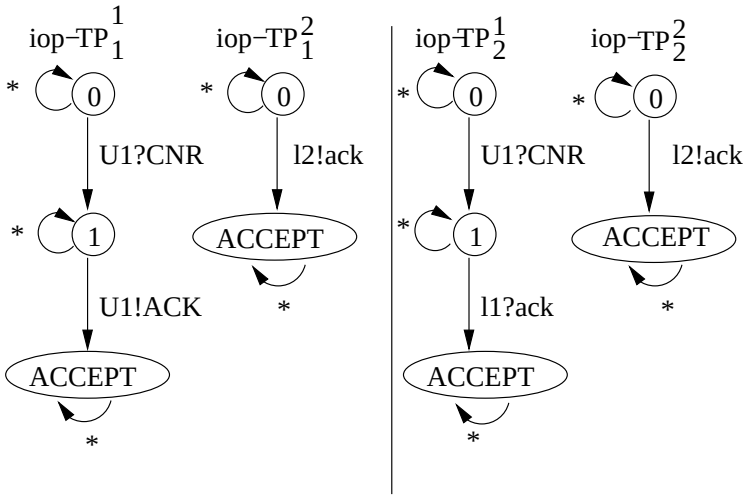


FIG. 4.18 – Objectifs de test unilatéraux (algorithme modifié) pour iop-TP_1 et iop-TP_2

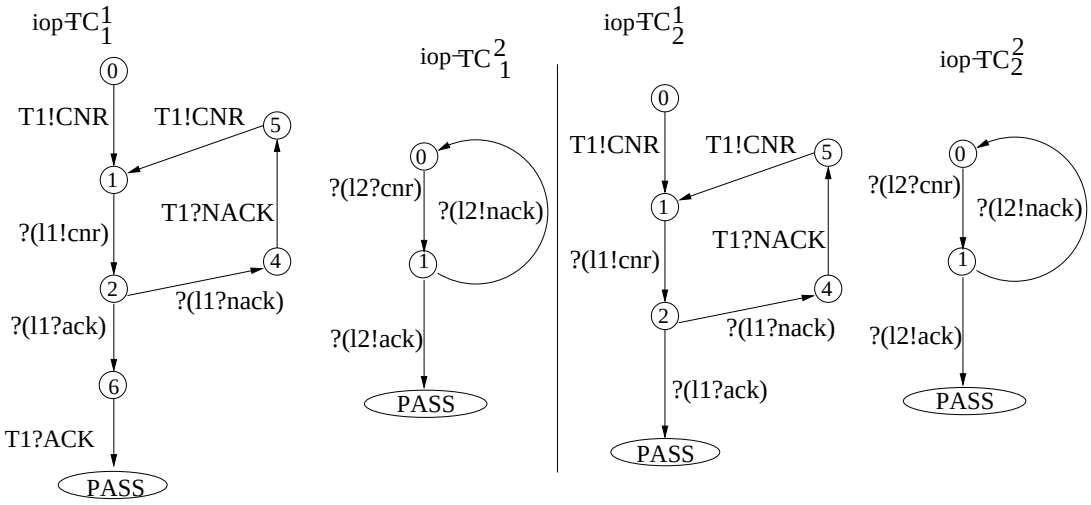


FIG. 4.19 – Cas de test unilatéraux (algorithme modifié) pour iop-TP_1 et iop-TP_2

cas de test unilatéraux) sont composés d'exactement les mêmes traces que les cas de test globaux représentés dans la figure 4.14.

Ainsi, cette nouvelle version de l'algorithme de dérivation des objectifs de test d'interopérabilité unilatérale borne la recherche de prédécesseur. Cette version de l'algorithme permet de générer des objectifs de test unilatéraux et des cas de test d'interopérabilité unilatérale plus cohérent par rapport à l'objectif de test global. Les cas de test d'interopérabilité bilatérale générés avec cette version modifiée de la méthode sont équivalents en terme de détection de la non-interopérabilité avec les cas de test d'interopérabilité globale. De plus, les cas de test obtenus (contrairement à ceux obtenus avec la première version de l'algorithme) sont exécutables avec une approche distribuée (cf. section 4.7).

4.10 Autres utilisations de la méthode proposée

Nous donnons dans cette section quelques idées pour adapter tout ou partie de notre méthode à des contextes particuliers où, par exemple, toutes les interfaces ne sont pas accessibles. En effet, la méthode développée ici est prévue pour une architecture de test d'interopérabilité permettant l'accès à toutes les interfaces des implémentations. Dans le cas où certaines interfaces ne sont pas accessibles, il faut donc adapter les algorithmes présentés ici de façon à avoir la meilleure détection de la non-interopérabilité possible.

Nous avons vu dans le chapitre 3 qu'il n'y a pas équivalence entre critères d'interopérabilité bilatérale et globale dans les architectures supérieures et inférieures. Ceci est dû au fait que les architectures de test d'interopérabilité globale totale et bilatérale totale ont accès à *toutes* les interfaces des implémentations, même si le mode d'observation des deux implémentations est différent dans ces deux architectures. Ainsi, le critère bilatéral n'est pas basé sur l'interaction des spécifications, mais il utilise tout de même certaines synchronisations implicites entre évènements, qui sont d'une part l'ordonnancement entre les évènements exécutés sur différentes interfaces d'un même système, et d'autre part la correspondance entre une sortie sur une interface inférieure d'une implémentation et l'entrée correspondante sur l'interface inférieure de l'autre implémentation. Une partie des évènements exécutés n'étant pas observables dans les architectures supérieures et inférieures, cette "synchronisation implicite" ne peut être utilisée dans les critères d'interopérabilité bilatérale correspondants. Ainsi, il n'y a pas d'équivalence entre critères d'interopérabilité bilatérale et globale dans ces contextes.

Pour avoir la meilleure détection de la non-interopérabilité possible dans des contextes d'architectures inférieures ou supérieures sans avoir à calculer l'interaction des spé-

cifications, il est possible de combiner différents algorithmes. Nous donnons ici un exemple d'adaptation de la méthode de génération de test présentée dans ce chapitre à une architecture de test d'interopérabilité supérieure. Cette adaptation de notre méthode détecte particulièrement bien les situations de non-interopérabilité dans le cas où les spécifications alternent des évènements sur les deux interfaces : à chaque entrée sur une interface supérieure (resp. inférieure) doit succéder une sortie sur une interface inférieure (resp. supérieure), comme dans l'exemple des spécifications S_1 et S'_2 de la figure 3.7 de la section 3.2 du chapitre 3. Ce type de spécifications peut correspondre à des protocoles qui transmettent un message provenant de la couche supérieure après une manipulation de ce message (évènements internes) tels que du chiffrement, de l'ajout de codes de détection d'erreurs ou de d'autres informations, etc.
Une telle propriété sur les spécifications permet en effet de tester l'interopérabilité dans un architecture de test supérieure en se basant sur les dépendances entre évènements à exécuter sur des interfaces de type différent. Pour cela, on génère des cas de test par l'approche bilatérale, puis ces cas de test sont projetés sur les interfaces supérieures de façon à garder seulement les évènements observables dans l'architecture de test considérée. A partir des cas de test avant projection, on peut calculer les dépendances entre évènements sur les interfaces supérieures des implémentations. Par exemple, un message $U1?X$ doit entraîner $l1!x'$ qui correspond à la réception $l2?x'$ sur I_2 qui doit elle-même entraîner une sortie $U2!X'$: ainsi il y a dépendance causale entre $U1?X$ et $U2!X'$. Ces dépendances causales entre évènements sur $U1$ et $U2$ permettent d'ajouter une synchronisation à des cas de test générés par une approche bilatérale (donc a priori sans synchronisation). Cette ajout de synchronisation leur donne une détection de la non-interopérabilité équivalente aux cas de test globaux qui seraient générés dans la même situation. Mais la génération de cas de test globaux se base sur $(S_1 \| _{\mathcal{A}} S_2)/(\Sigma_U^{S_1} \cup \Sigma_U^{S_2})$, c'est-à-dire que la projection a lieu après le calcul de l'interaction des spécifications source de problème d'explosion combinatoire du nombre d'états.
Notons qu'ici, même s'ils ont été générés au départ via une approche bilatérale, les cas de test doivent être exécutés sur une architecture de test globale, puisqu'une synchronisation a été ajoutée pour leur assurer un meilleur pouvoir de détection de la non-interopérabilité. De plus, une telle méthode peut être utilisée de façon générale avec une architecture de test d'interopérabilité supérieure mais n'aura pas une aussi bonne détection de la non-interopérabilité dans le cas où les spécifications n'ont pas la propriété décrite ci-dessus.

D'autres méthodes de génération de tests d'interopérabilité peuvent être dérivées sur les mêmes principes que la méthode décrite dans cette section. Nous ne décrivons pas ici toutes ces méthodes. En effet, ces méthodes doivent être développées au cas par cas en fonction de l'architecture de test utilisée, de la nature/complexité des spécifications, du critère de détection choisi, etc.

4.11 Conclusion

Dans ce chapitre, nous nous sommes intéressés à la génération de test d'interopérabilité basée sur une définition formelle de la notion d'interopérabilité de deux implémentations. Nous avons défini les modèles formels associés aux objets manipulés lors du test de l'interopérabilité de deux implémentations. Nous nous sommes surtout intéressés aux approches possibles pour la génération de tests dans le cas où toutes les interfaces des implémentations sont accessibles par les testeurs (cas le plus courant en pratique). Nous avons développé pour ce contexte une méthode complète de génération de tests d'interopérabilité basée sur une approche bilatérale. Cette méthode permet de générer (à partir d'objectifs de test) des cas de test répondant aux deux objectifs du test d'interopérabilité : la vérification de l'interaction correcte entre les deux implémentations, et la vérification de la réalisation des services attendus lors de l'interaction entre implémentations.

La méthode de génération de tests d'interopérabilité que nous avons développée et décrite dans ce chapitre a plusieurs avantages par rapport aux approches classiques. Tout d'abord, elle ne nécessite pas de modèle global (ni complet, ni même partiel) de l'interaction des spécifications, ce qui permet d'éviter le problème d'explosion combinatoire du nombre d'états rencontré avec des approches classiques existantes. De plus, elle ne nécessite pas la réalisation de procédure de synchronisation entre testeurs, l'architecture de test bilatérale étant une architecture distribuée sans synchronisation. Enfin, les cas de test bilatéraux générés par notre méthode ont la même capacité de détection de la non-interopérabilité que les cas de test qui seraient générés avec une approche globale. Par ailleurs, il s'agit à notre connaissance de la première méthode qui prend en compte les entrées. Via le calcul des dépendances causales entre entrées et sorties, nous sommes en mesure de vérifier la prise en compte par l'implémentation réceptrice des messages émis par l'autre implémentation.

Dans ce chapitre, nous avons également appliqué cette méthode et l'approche classique de façon à illustrer les contributions de l'approche bilatérale de génération de tests d'interopérabilité. Nous avons aussi donné des pistes pour adapter cette méthode à des architectures de test ne permettant pas l'accès à toutes les interfaces des implémentations.

La génération de test décrite dans ce chapitre et les définitions formelles du chapitre précédent sont définies pour un contexte de test d'interopérabilité one-to-one, c'est-à-dire un contexte mettant en relation deux implémentations. Dans le chapitre suivant, nous nous intéressons au contexte plus général du test d'interopérabilité de N implémentations ($N > 2$).

Chapitre 5

Test d'interopérabilité multi-implémentations

5.1 Introduction

Les deux chapitres précédents portaient sur la formalisation et la génération automatique de tests d'interopérabilité dans le cadre de l'interaction de deux implémentations. Dans ce chapitre, nous nous intéressons au contexte plus général de l'interaction de N implémentations. C'est le contexte du test d'interopérabilité de multiples implémentations ou M-iop test pour "multiple implementation interoperability testing". La situation est plus complexe que le contexte one-to-one à cause du nombre d'entités à manipuler lors des tests. Cependant, les objectifs du test d'interopérabilité dans ce contexte restent les mêmes : les N implémentations doivent communiquer correctement tout en rendant les services prévus dans les spécifications correspondantes.

Ce contexte est rencontré moins couramment que le contexte one-to-one (deux implémentations ou une implémentation face à un système déjà opérationnel composé de N entités). En effet, il n'est utilisé que lorsqu'il est impossible de s'appuyer sur un contexte one-to-one pour tester l'interopérabilité des implémentations, comme par exemple lorsqu'aucune des entités n'a atteint un niveau satisfaisant dans son développement. Actuellement, le test d'interopérabilité dans ce contexte multi-implémentations est le plus souvent réalisé en connectant les N implémentations entre elles, et en observant puis analysant les traces générées lors de cette interconnexion : c'est du test "passif" qui est donc utilisé, ce qui permet d'éviter la génération de cas de test d'interopérabilité, complexe à cause de la taille et du nombre des objets à manipuler (N implémentations, donc N spécifications). Cette méthode n'étant pas basée sur des cas de test, on ne sait pas à l'avance quelles seront les fonctionnalités effectivement testées. Le risque est donc que les fonctionnalités conduisant à des situations de non-interopérabilité ne soient pas testées.

Les objectifs du test d'interopérabilité sont les mêmes dans les contextes one-to-one et multi-implémentations. La principale différence entre les deux contextes vient du nombre de systèmes à manipuler qui introduit de nouveaux problèmes à plusieurs étapes de la formalisation du test d'interopérabilité. Particulièrement, la manipulation de N systèmes introduit de nouvelles données à prendre en compte dans les étapes de configuration préalables au test d'interopérabilité. La configuration du SUT (System Under Test) composé des N ($N > 2$) implémentations a ensuite une influence dans toutes les étapes de l'approche formelle du test d'interopérabilité. Le test d'interopérabilité multi-implémentations n'est donc pas un passage à l'échelle du contexte one-to-one, mais plutôt un contexte différent : de nouvelles propriétés sur les systèmes (dont par exemple la topologie d'interconnexion des N implémentations) sont à prendre en compte dans ce contexte.

Dans ce chapitre, nous étudions dans quels contextes les définitions et méthodes décrites pour le contexte one-to-one peuvent être adaptées au contexte du test d'interopérabilité de multiples implémentations et quelles autres doivent être réécrites pour ce contexte. Il existe des définitions du contexte one-to-one qui peuvent en effet être généralisées, alors que d'autres ne sont pas adaptables car ne permettant pas de prendre en compte la complexité des connexions entre les N systèmes.
Un des points à prendre à compte avec N implémentations est la topologie d'interconnexion des implémentations testées. En effet, dans le contexte one-to-one, il n'y a qu'une topologie possible pour connecter les deux implémentations : la topologie relie simplement les deux implémentations de façon directe. Par contre, pour tester l'interopérabilité de N implémentations, plusieurs topologies (bus, anneau, maillage, étoile, hybride, etc) sont possibles. Le test d'interopérabilité vérifiant la capacité des implémentations à communiquer entre elles, il est important de connaître quelle implémentation est dans la situation de pouvoir communiquer avec quelle autre, à l'intérieur du réseau d'interconnexion des N implémentations. Cette configuration influence également les possibilités de configurations des testeurs, les conditions de compatibilité des spécifications ou la définition de l'interopérabilité servant de base aux tests.
Dans ce chapitre, nous définissons les contextes et les possibilités de configuration des N implémentations. C'est-à-dire que nous nous intéressons d'abord aux architectures de test et aux topologies possibles pour le test d'interopérabilité de N implémentations. Dans la deuxième partie de ce chapitre, nous nous intéressons à la définition formelle de la notion d'interopérabilité de N systèmes, et à toutes les définitions qui doivent être adaptées à ce contexte préalablement à la définition de critères d'interopérabilité.

L'organisation de ce chapitre est la suivante. La section 5.2 étudie les architectures de test possibles pour le contexte du test d'interopérabilité de N implémentations. De même que dans le contexte one-to-one, il existe différentes possibilités pour les

connexions entre testeurs et implémentations aboutissant à plusieurs classes d'architectures de test d'interopérabilité. Dans cette section, nous comparons également les contextes et architectures de test définis avec les architectures définies pour le contexte one-to-one (voir section 3.2).
La section 5.3 s'intéresse à la topologie connectant les N implémentations dans le test d'interopérabilité et à sa modélisation. La topologie est un des éléments de la configuration pour le test d'interopérabilité qui a le plus d'influence sur les étapes suivantes permettant de générer les cas de test. Dans la section 5.3, nous nous intéressons donc au problème du choix d'une topologie pour le test d'interopérabilité de N implémentations et à sa modélisation, permettant ainsi de prendre en compte le mode de connexion des implémentations à toutes les étapes du test d'interopérabilité.
Après avoir défini le contexte du test d'interopérabilité des N implémentations aussi bien au niveau de la topologie que de l'architecture de test, nous nous intéressons aux modèles des données à manipuler. En effet, certaines définitions formelles (dont celle de l'opération d'interaction) doivent être redéfinies pour prendre en compte les N systèmes. Les sections 5.4 et 5.5 portent ainsi sur la définition de modèles formels adaptés au contexte multi-implémentations, puis sur l'adaptation de la propriété d'iop-compatibilité des spécifications au contexte considéré (et à la topologie associée).
Dans la section 5.6, nous nous intéressons à la définition de la notion formelle de l'interopérabilité dans les différents contextes présentés section 5.2. Comme dans le chapitre 3, nous appelons ces définitions des critères d'interopérabilité. Nous appliquons ensuite ces critères d'interopérabilité du contexte M-iop sur un exemple dans la section 5.7.
Finalement, nous donnons des pistes pour la génération de test basé sur les critères d'interopérabilité dans la section 5.8.

5.2 Architectures de test

Dans le contexte de test d'interopérabilité multi-implémentations, comme dans le cas des contextes du test de conformité ou du test d'interopérabilité one-to-one [WSG98, VBT01] (cf. sections 2.4.1 et 2.5.3), il existe plusieurs modes d'accès aux implémentations testées. Selon ces modes d'accès (observation et/ou contrôle), plusieurs architectures de test d'interopérabilité multi-implémentations peuvent être définies. L'objectif de cette section est donc de décrire ces différentes architectures et les contextes dans lesquelles elles sont généralement utilisées.

Dans le contexte de test d'interopérabilité multi-implémentations, de même que pour le test d'interopérabilité en contexte one-to-one, deux types d'interfaces peuvent être différenciées pour les implémentations.
Les interfaces supérieures (*UI* pour Upper Interfaces) sont les interfaces utilisées pour

la communication avec l'environnement (ou les couches supérieures, ou l'utilisateur du service, etc), c'est-à-dire les interfaces par lesquelles transitent les messages correspondant à des demandes de service ou à la réalisation de ce service. Comme dans le contexte de test d'interopérabilité one-to-one, ces interfaces sont à la fois observables et contrôlables par les testeurs, c'est-à-dire que ces testeurs peuvent envoyer des stimuli vers ces interfaces et recevoir des messages provenant d'elles.
Les interfaces inférieures (*LI* pour Lower Interfaces) sont les interfaces utilisées pour la communication entre implémentations, c'est-à-dire les interfaces par lesquelles transitent les messages destinés à l'interaction entre entités. Ces interfaces peuvent être observées par les testeurs qui leur sont connectées, c'est-à-dire que le testeur peut connaître les messages envoyés par une implémentation à une autre implémentation. La différence avec le contexte du test d'interopérabilité one-to-one vient de la nécessité d'identifier précisément le destinataire des messages transmis par une interface inférieure.

Selon la façon dont le ou les testeurs sont connectés aux interfaces des implémentations, différentes architectures de test peuvent être définies. De même que dans le contexte one-to-one, des architectures peuvent être définies en fonction du type d'interfaces observées/contrôlées. L'architecture de test d'interopérabilité peut ainsi être dite inférieure, supérieure ou totale si seules les interfaces inférieures, supérieures ou les deux types d'interfaces sont connectées à des testeurs.
Dans cette section, nous nous intéressons principalement aux architectures de test d'interopérabilité totale qu'il est possible de définir en fonction du mode d'observation des implémentations. En contexte one-to-one, ces architectures de test sont dérivables en architectures de test supérieures ou inférieures en fonction des besoins (cf. section 2.5.3). De même, dans le contexte de test d'interopérabilité multi-implémentations, les architectures de test d'interopérabilité basés interfaces supérieures ou inférieures peuvent être déduites des architectures de test totales.

5.2.1 Architecture de test d'interopérabilité one-against-N

Le test d'interopérabilité dit *one-against-N* (ou en français "un-contre-N") (cf. figure 5.1) est un contexte utilisé pour tester l'interopérabilité de $N+1$ implémentations en ayant déjà une connaissance sur l'interopérabilité de N de ces implémentations. Dans ce contexte, nous considérons un réseau composé de N implémentations qui interopèrent correctement. Une nouvelle implémentation, l'implémentation de numéro $N+1$, est alors connectée à ce réseau. L'objectif du test d'interopérabilité est alors de vérifier que le nouveau réseau composé des $N+1$ implémentations fonctionne correctement. Le test d'interopérabilité one-against-N est donc utilisé pour répondre à la question "le système de $N+1$ implémentations, obtenu par la connexion d'une implémentation supplémentaire aux N implémentations interagissant entre elles, va-t-il

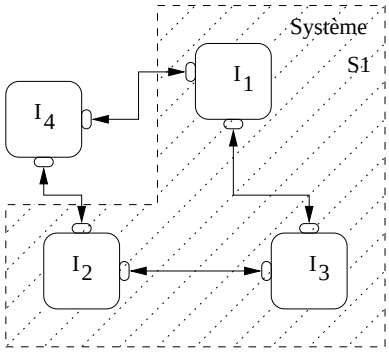


FIG. 5.1 – Architectures de test d'interopérabilité one-against-N

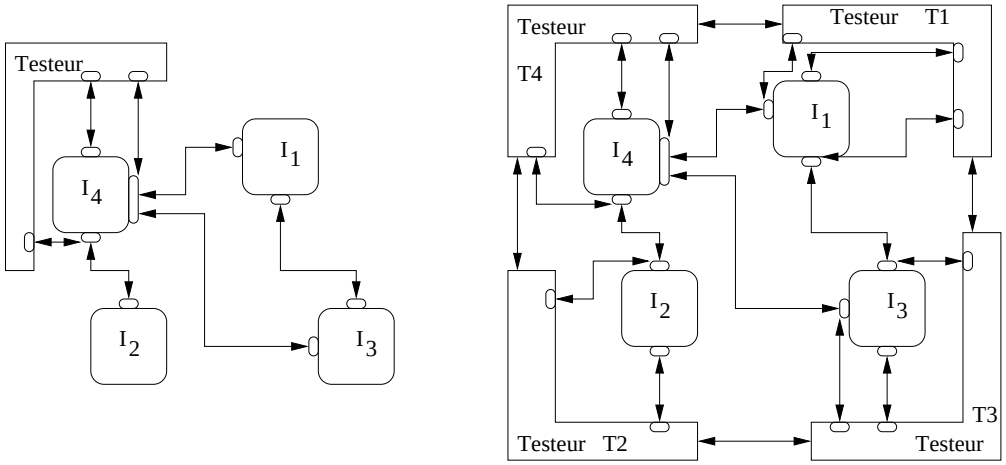
continuer à fonctionner ?".

Le contexte one-against-N est un contexte utilisé pour tester l'interopérabilité de plus de deux implémentations, mais ses caractéristiques en font un cas particulier du contexte de test d'interopérabilité one-to-one. En effet, le sous-réseau composé des N implémentations déjà interconnectées peut être vu globalement comme un seul système. Le test d'interopérabilité teste alors l'interopérabilité entre ce système composé des N implémentations et l'implémentation $N + 1$, c'est-à-dire en fait l'interopérabilité entre deux systèmes.
Pour cette raison, nous ne traiterons pas le contexte one-against-N dans la suite de ce chapitre. En effet, les définitions et méthodes de génération de test définies pour le contexte de test d'interopérabilité one-to-one peuvent être appliquées au contexte de test d'interopérabilité one-against-N.

5.2.2 Architectures de test d'interopérabilité multipartie

Le contexte le plus général pour le test d'interopérabilité de N implémentations est le contexte dit *multiparti* (ou multiparty interoperability testing context). Dans ce contexte, il n'y a pas d'hypothèse sur l'interopérabilité existant dans un sous-réseau formé par $N - 1$ implémentations. Toutes les implémentations sont connectées entre elles suivant une topologie préalablement définie. L'objectif est ainsi de tester si ces N implémentations sont capables d'interagir dans cette configuration et de rendre les services prévus dans les spécifications correspondantes.

Dans la suite de ce chapitre, le contexte multiparti sera le seul contexte d'interopérabilité multi-implémentation considéré. Plusieurs architectures de test d'interopérabilité multiparties peuvent être définies en fonction du mode d'accès des interfaces des N implémentations par les composants du système de test (TS, Test System). Dans cette section, nous présentons les architectures de test multipartie (ou M-iop) totales, c'est-à-dire considérant que l'accès aux deux types d'interfaces est possible.



Architecture de test unilatérale multipartie (a)

Architecture de test globale multipartie (b)

FIG. 5.2 – Architectures de test d'interopérabilité multipartie

Architecture de test M-iop globale L'architecture de test d'interopérabilité multi-implémentations globale (ou global M-iop testing architecture) est représentée dans la figure 5.2(b). Cette architecture de test est appelée globale (comme dans le cas du test d'interopérabilité de deux implémentations) car la décision de l'interopérabilité est prise à partir d'une vue globale du système composé par les N implémentations.
Dans cette architecture de test, le contrôle et l'observation est réalisé globalement par l'ensemble des N testeurs T_i. Une procédure de coordination entre ces testeurs (représentée sur la figure par la connexion entre les testeurs) sert pour leur coordination et synchronisation. Cette architecture de test correspond à un contexte de test parallèle (ou distribué) *avec* synchronisation entre testeurs.

Compte tenu de la complexité du SUT (composé de N IUTs) à tester globalement, cette architecture de test est souvent utilisée dans un contexte de test "passif". Le test passif est une méthode de test qui ne repose que sur l'observation des traces d'évènements émanant du SUT et leur comparaison avec la spécification. Dans le cas du test d'interopérabilité "passif", il peut être nécessaire de générer des stimuli au début du test de façon à initialiser la communication entre IUTs et à choisir (via des primitives de demande de service par exemple) la fonctionnalité testée. Ce mode de test pour une architecture de test d'interopérabilité multi-implémentations globale a l'avantage de ne pas utiliser de cas de test global difficile voire impossible à dériver à cause de la complexité et du nombre des systèmes considérés.

Architecture de test M-iop unilatérale L'architecture de test d'interopérabilité multi-implémentations unilatérale (ou unilateral M-iop testing architecture) est représentée dans la figure 5.2(a) : dans cette exemple, l'architecture de test se concentre sur les évènements exécutés par l'implémentation I_4 lors de son interaction avec les trois autres implémentations du SUT.
Cette architecture de test est appelée unilatérale car elle se concentre sur une seule des N implémentations pendant son interaction avec les $N-1$ autres implémentations. Selon la topologie interconnectant les N implémentations, l'architecture de test M-iop unilatérale se concentre donc sur ce qui se passe sur les interfaces supérieures d'une implémentation i mais également sur les k interfaces inférieures de cette implémentation qui lui servent à communiquer directement avec $k \leq N-1$ autres implémentations.
L'implémentation sur laquelle l'architecture de test se concentre est une implémentation sous test (ou IUT). Les autres implémentations peuvent être des implémentations de référence (RI pour Reference Implementation) dans le cas où leur comportement est connu précisément (ou qu'elles ont atteint un niveau de développement satisfaisant) mais peuvent également être des IUTs.

Cette architecture de test peut être utilisée dans le cas où les interfaces de certaines implémentations ne sont pas accessibles pour les testeurs (cf. section 2.5.3 pour des exemples d'architectures de test unilatérale en contexte one-to-one), mais également dans un but de simplifier le test d'interopérabilité en restreignant le nombre d'objets manipulés.

Architecture de test M-iop multi-unilatérale L'architecture de test d'interopérabilité multi-unilatérale (ou multi-unilateral M-iop testing architecture) peut être comparée au contexte de l'interopérabilité one-to-one bilatérale. En effet, dans cette architecture de test multi-implémentations multi-unilatérale, chaque testeur T_i réalise séparément l'observation et/ou le contrôle de l'implémentation I_i, via une architecture de test multi-implémentations unilatérale (de même que, dans l'architecture bilatérale, chaque testeur se concentrait sur une des deux implémentations). Cette architecture de test n'est pas représentée sur la figure 5.2 : c'est en fait l'architecture de la figure 5.2(b) (architecture de test multi-implémentations globale) mais sans les liens entre testeurs puisque dans le contexte multi-unilatéral, il n'y a aucune coordination entre les actions des différents testeurs. Cette architecture de test correspond donc à un contexte de test parallèle (ou distribué) *sans* synchronisation.

Remarque: Comme il n'y a pas de coordination entre les testeurs, la dérivation d'un cas de test d'interopérabilité dans ce contexte est moins complexe que dans le contexte de test M-iop globale.

5.3 Topologie

Différentes topologies sont possibles pour interconnecter les N implémentations dont on veut tester l'interopérabilité. La topologie correspond à la description des différents liens existants pour connecter les implémentations entre elles. Puisque l'un des objectifs du test d'interopérabilité est de vérifier que la communication entre les implémentations est correcte, le test d'interopérabilité est dépendant de la façon dont les implémentations testées sont interconnectées.
Dans cette section, nous étudions l'influence de la topologie sous deux angles : sa modélisation pour la définition formelle de la notion d'interopérabilité, et l'influence du choix d'une topologie sur les objets manipulés lors d'une approche formelle du test d'interopérabilité.

5.3.1 Choix d'une topologie pour le test d'interopérabilité

Différentes topologies sont possibles pour interconnecter les N implémentations dont on veut tester l'interopérabilité. L'un des objectifs du test d'interopérabilité est de vérifier que la communication entre les implémentations est correcte. Le test d'interopérabilité est donc dépendant de la façon dont les implémentations testées sont interconnectées. Ainsi, le choix de la topologie utilisée pour interconnecter les implémentations testées est le premier point à déterminer avant de décrire des tests d'interopérabilité. Deux points sont à déterminer dans la topologie :

1. le nombre d'implémentations à interconnecter pour le test d'interopérabilité
2. les connexions (liens) entre ces implémentations

Remarque: En pratique, une partie des implémentations connectées lors du test d'interopérabilité peuvent être des *implémentations de référence* (ou RI pour Reference Implementation). Ces implémentations sont des implémentations qui ont atteint un niveau de développement satisfaisant (ayant subi plusieurs campagnes de tests de conformité sans Fail par exemple).

Tout d'abord, intéressons-nous aux topologies utilisées en pratique pour le test d'interopérabilité de N implémentations. Actuellement dans la pratique, les cas de test d'interopérabilité multi-implémentations (également appelés scénarios) sont décrits manuellement. Les testeurs choisissent les fonctionnalités du protocole qu'ils veulent vérifier lors des tests, et déterminent la topologie à mettre en place en fonction des besoins de la fonctionnalité considérée. Par exemple, dans le cas du test d'interopérabilité d'un routeur avec d'autres systèmes (routeurs et/ou hôtes), le choix du test d'une fonctionnalité peut impliquer la présence d'un nombre plus ou moins important d'hôtes ou de routeurs dans le réseau formé par toutes ces implémentations. De plus, en

fonction des besoins du test, ces routeurs et hôtes peuvent être connectés entre eux suivant différentes topologies. Ainsi, différentes topologies sont généralement utilisées en fonction des fonctionnalités testées. Ces différentes topologies impliquent l'interconnexion d'un nombre variables d'implémentations, dont certaines sont des implémentations de référence (RI, Reference Implementation) et d'autres sont les implémentations sous test (ou IUT).

Dans une approche formelle du test d'interopérabilité de N implémentations, les spécifications sont représentées par des modèles de type automate. Nous utiliserons dans cette étude le modèle des IOLTS (cf. section 2.3). Ce modèle d'une spécification au format IOLTS contient, parmi d'autres informations, la liste des interfaces du système associée (dans l'ensemble des messages exécutables par un système) à l'alphabet des messages pouvant être transmis via ces interfaces. Or selon la topologie utilisée pour le test d'interopérabilité, le nombre d'entités directement connectées à un système particulier varie et donc également le nombre d'interfaces de ce système. Ainsi, quand on applique une approche formelle au contexte du test d'interopérabilité multi-implémentations, il faut minimiser les changements de topologie contrairement à l'approche actuellement où les cas de test sont dérivés manuellement. En effet, chaque changement de topologie impose une réécriture du modèle des spécifications considérées. Or cette étape de modélisation et la validation du modèle associée est une des étapes les plus coûteuses de l'approche formelle.

Il faut donc, lors d'une approche formelle du test d'interopérabilité multi-implémentations, trouver une *topologie minimale* permettant de tester toutes les fonctionnalités considérées par l'ensemble des cas de test à générer. Ce choix s'appuie sur la modélisation des différentes topologies qui seraient utilisées en pratique, mais également sur les équivalences pour le test d'interopérabilité entre topologies. Par exemple, considérons deux fonctionnalités à tester f_1 et f_2. Le test d'interopérabilité se focalisant sur f_1 nécessite au minimum l'interconnexion de $N-1$ systèmes tandis que le test d'interopérabilité se focalisant sur f_2 doit s'appuyer sur au moins N implémentations interconnectées. Dans le cas où le passage de la topologie pour f_1 et à la topologie pour f_2 consiste à l'ajout de l'implémentation N, il est possible de tester les deux fonctionnalités avec la topologie interconnectant N implémentations. Cette topologie est ainsi la topologie minimale permettant de tester à la fois les fonctionnalités f_1 et f_2.

Pour déterminer ces équivalences entre topologies et pouvoir définir la topologie minimale pour le test d'interopérabilité, nous définissons un modèle matriciel représentant les topologies. Puis nous nous intéressons aux représentations des topologies de base et aux opérations de passage d'un modèle d'une topologie au modèle d'une autre topologie, en particulier dans le cas du passage de $N-1$ à N implémentations.

5.3.2 Modèle de description de la topologie

Pour pouvoir prendre en compte la topologie lors d'une approche formelle de test d'interopérabilité multi-implémentations, il est important de définir un modèle permettant de prendre en compte l'existence ou l'absence de connexion entre les implémentations. Comme dans [Tör99], nous utilisons une matrice pour représenter la topologie, c'est à dire les liens existant entre implémentations.
(L_{ij}) est la matrice représentant la topologie connectant les N implémentations. $L_{ij} = 1$ (resp. $L_{ij} = 0$) quand un lien existe (resp aucun n'existe) entre I_i et I_j (ou le modèle de leurs spécifications S_i et S_j). Par défaut, L_{ii} prend la valeur 0, ce qui indique que nous ne prenons pas en compte les boucles locales (loopback).
La matrice est donc une matrice carrée dont l'ordre correspond au nombre d'entités interconnectées. Dans le cas général où les liens sont bidirectionnels, les matrices de représentation de la topologie sont des matrices symétriques. De plus, comme $L_{ii} = 0$, la diagonale principale est composée de 0 quelque soit la topologie.

5.3.3 Représentation des topologies de base

Pour connecter N entités, différentes topologies sont possibles. Parmi les topologies de base utilisées dans le domaine des réseaux informatiques, il y a les topologies en bus, de maillage, en arbre, en anneau, ou également des topologies hybrides (qui combinent plusieurs de ces topologies de base). Dans cette section, nous nous intéressons à la représentation de ces topologies et aux liens entre les différentes représentations de topologies. L'objectif est de déterminer quelle est la représentation "type" pour chacune des topologies de base et également de déterminer comment déterminer le modèle d'une topologie connectant $N + 1$ implémentations à partir de la topologie congruente servant à connecter N implémentations. Les implémentations considérées lors du test d'interopérabilité seront numérotées, dans le cas d'une topologie de base, de façon à ce que la matrice manipulée lors des tests soit la matrice "type" de la topologie. Cette connaissance des matrices de représentation des topologies de base est également utile lors de la détermination de la topologie minimale pour le test d'interopérabilité, et des équivalences entre topologies pour le test d'une fonctionnalité.

5.3.3.1 Matrices de représentation des topologies de base

Nous donnons ici la représentation des topologies suivantes : topologie en bus, topologie en étoile, topologie en anneau, et topologie de maillage. La numérotation considérée est précisée pour chacune des topologies. Il est possible qu'une autre numérotation ait été utilisée pour identifier les entités connectées. Dans ce cas, la matrice représentant la topologie peut être représentée en permutant des lignes et colonnes de la matrice de représentation habituelle de la topologie. Chaque permutation entre

deux systèmes i et j est représentée par la permutation des lignes et colonnes i et j correspondantes de la matrice représentant la topologie.

Topologie en bus La topologie en bus (N implémentations) peut être représentée par une matrice qui a ses diagonales inférieures (scalaires $L_{12}, \cdots, L_{N-1N}$) et supérieures (scalaires $L_{21}, \cdots, L_{NN-1}$) égales à 1 tandis que les autres scalaires de la matrice sont à 0 : $(L_{ij}) = \begin{cases} L_{ij} = 1 & si\ (i = j+1\ \vee\ j = i+1) \\ L_{ij} = 0 & sinon \end{cases}$.
Cette matrice correspond à un bus connectant l'IUT$_1$ à l'IUT$_2$, l'IUT$_2$ à l'IUT$_3$, ..., et l'IUT$_{N-1}$ à l'IUT$_N$.

Topologie en étoile Soit k le numéro de du système auquel sont connectés les $N-1$ autres implémentations. Dans ce cas, la matrice représentant la topologie en étoile est : $(L_{ij}) = \begin{cases} L_{ij} = 1 & si\ (i = k\ \vee\ j = k)\ \wedge\ i \neq j \\ L_{ij} = 0 & sinon \end{cases}$

Topologie en anneau La matrice représentant la topologie en anneau, anneau connectant l'IUT$_1$ à l'IUT$_2$, l'IUT$_2$ à l'IUT$_3$, ..., l'IUT$_{N-1}$ à l'IUT$_N$, et l'IUT$_N$ à l'IUT$_1$, est : $(L_{ij}) = \begin{cases} L_{ij} = 1 & si\ (i = j+1\ \vee\ j = i+1) \\ & \quad \vee\ (i = 1\ \wedge\ j = N) \\ & \quad \vee\ (i = N\ \wedge\ j = 1) \\ L_{ij} = 0 & sinon \end{cases}$.

Maillage Pour la topologie de maillage, qui relie chaque implémentation aux $N-1$ autres, la matrice est définie par : $(L_{ij}) = \begin{cases} L_{ij} = 0 & si\ i = j \\ L_{ij} = 1 & sinon \end{cases}$

Pour les autres topologies telles que les arbres ou les topologies hybrides, il n'existe pas de modèle type de base. En effet, il existe trop de formats différents d'arbres de N entités ou de topologies hybrides (combinant plusieurs topologies) connectant N systèmes.

5.3.3.2 Représentation de la matrice d'une topologie à partir d'une matrice de rang inférieur

Soit M_N la matrice représentant la topologie connectant N implémentations. Cette matrice peut être calculée à partir de la matrice M_{N-1} décrivant la topologie interconnectant un sous-réseau de $N-1$ implémentations et un vecteur V_N décrivant la connexion de l'implémentation de numéro N. Un matrice C représentant les nouvelles connexions à effectuer est construite à partir de V_N. De plus, si le passage de la topologie décrite dans M_{N-1} à la topologie décrite dans M_N impose de déconnecter au préalable certaines des entités (cas de la topologie en anneau par exemple), ces déconnexions seront décrites dans une matrice notée D. Sinon, D est une matrice nulle du même ordre que M_N. On a alors $M_N = A.M_{N-1}.{}^tA - D + C$ avec :

$$A = \left(\begin{array}{c} I_{N-1} \\ \hline 0 \quad \cdots \quad 0 \end{array}\right) \quad \text{et } C = \left(\begin{array}{c|c} 0 & V_N \\ \hline {}^tV_N & \end{array}\right)$$

La condition d'utilisation de cette égalité est que l'ajout de l'entité N au réseau existant de $N-1$ systèmes n'ait pas déconnecté de liens entre systèmes. Dans le cas des topologies en bus, étoile, anneau et maillage, il est possible de donner la valeur par défaut du vecteur V_N. De plus, la topologie en anneau est la seule de ces quatre topologies pour laquelle la matrice D décrivant les déconnexions n'est pas la matrice nulle.

- Bus : $V_N(i) = \begin{cases} V_N(i) = 1 & si\ i = N-1 \\ V_N(i) = 0 & sinon \end{cases}$
- Étoile (centrée sur k) : $V_N(i) = \begin{cases} V_N(i) = 1 & si\ i = k \\ V_N(i) = 0 & sinon \end{cases}$
- Anneau : le passage d'anneau de $N-1$ entités à un anneau de N entités nécessite de déconnecter au préalable l'IUT$_{N-1}$ et l'IUT$_1$, pour pouvoir connecter l'IUT$_{N-1}$ à l'IUT$_N$, et l'IUT$_N$ à l'IUT$_1$. Dans ce cas, la matrice D (d'ordre N) est telle que : $(D_{ij}) = \begin{cases} L_{ij} = 1 & si\ (i = 1\ \wedge\ j = N-1) \\ & \vee\ (i = N-1\ \wedge\ j = 1) \\ L_{ij} = 0 & sinon \end{cases}$.
  Le vecteur V_N est alors défini par : $V_N(i) = \begin{cases} V_N(i) = 1 & si\ i = 1 \vee i = N-1 \\ V_N(i) = 0 & sinon \end{cases}$
- Maillage : $V_N(i) = \begin{cases} V_N(i) = 0 & si\ i = N \\ V_N(i) = 1 & sinon \end{cases}$

Cette égalité peut également être utilisée pour définir les matrices des topologies en arbre ou des topologies hybrides. Il faut alors déterminer au préalable une topologie de base pour les entités 1 à $N-1$ (ou récursivement pour $N-2$ systèmes, etc). La matrice M_{N-1} est calculée par permutation de lignes et colonnes dans la matrice représentant la topologie entre la numérotation des matrices de base et la numérotation des systèmes dans la topologie courante. Le vecteur V_N sert alors à décrire les connexions entre le système numéro N et les $N-1$ autres entités.

5.3.3.3 Détermination de la topologie à utiliser pour le test d'interopérabilité

Ces modèles de représentation des topologies de base et l'opération permettant de déterminer le modèle d'une topologie connectant N systèmes à partir de la topologie connectant $N-1$ et les modèles des topologies de base peuvent aider pour déterminer les topologies à utiliser dans les deux situations suivantes.

1. Le premier contexte est la *détermination de la topologie minimale* pour le test d'interopérabilité. L'objectif est ici de déterminer, en étudiant les topologies équivalentes pour le test d'interopérabilité, quelle topologie peut permettre de

tester le plus grand nombre de fonctionnalités (cf. également section 5.3.1). En effet, dans une approche formelle du test d'interopérabilité multi-implémentations, chaque changement de topologie implique une redéfinition du modèle des spécifications (ou au moins du modèle des spécifications pour lesquelles les connexions ont été modifiées). Il faut donc minimiser les topologies différentes à utiliser pour le test d'interopérabilité, l'objectif étant de définir *une* topologie pouvant permettre de réaliser tous les tests.
Pour cela, les topologies qui seraient utilisées pour tester les différentes fonctionnalités sont tout d'abord représentées en se basant sur la numérotation des topologies de base. On obtient ainsi différents modèles de topologies possibles devant être utilisées pour différentes fonctionnalités à tester. Ensuite, il faut étudier le modèle de ces différentes topologies de façon à rechercher une topologie représentée par une matrice M_N telle que toutes les autres topologies définies peuvent être représentées par des matrices congruentes M_{N-i} de M_N. La topologie représentée par M_N (qui peut être parmi les topologies prédéfinies ou être une nouvelle topologie) est alors la topologie minimale permettant de tester les fonctionnalités considérées.

2. Le deuxième contexte pouvant utiliser les modèles de topologie de base et liens entre matrices M_{N-1} et M_N est l'*organisation d'un campagne de tests d'interopérabilité* en s'appuyant sur les dépendances entre topologies pour minimiser les reconfigurations (connexions/ déconnexions) à faire lors des tests. Lorsque différents cas de test d'interopérabilité avec les topologies correspondantes ont été définis, il faut organiser l'exécution de ces cas de test de façon à réaliser la campagne de test en évitant un trop grand nombre de connexions et déconnexions entre systèmes (à cause du risque d'erreurs, des besoins de reconfiguration, etc).
   La modélisation des différentes topologies correspondant aux cas de test d'interopérabilité définis peut permettre de minimiser à la fois le nombre de changements de topologies lors de la campagne de test, et le nombre de connexions et déconnexions nécessaire à chaque changement. En effet, le nombre de changements de topologies peut être minimisé en identifiant, grâce au modèle des différentes topologies, celles qui sont utilisées plusieurs fois lors de la campagne. Il est ainsi possible d'organiser les tests de façon à exécuter tous les cas de test correspondant à une topologie donnée avant de changer de topologie pour d'autres séries de test. De plus, comme on l'a vu dans la section précédente, les connexions et déconnexions peuvent être modélisées par des matrices appelées C et D : moins ces matrices ont d'éléments à 1, moins il y a de connexions ou déconnexions à réaliser pour passer d'une topologie à l'autre. Ainsi, il est possible de se baser sur de telles opérations de façon à organiser un ordre pour les changements de topologies qui minimise les connexions et déconnexions à effectuer à chaque changement.

5.4 Définitions préalables

5.4.1 Liens entre systèmes, interfaces et évènement miroir

Comme dans le contexte de test d'interopérabilité one-to-one, les spécifications et les implémentations sont représentés par des IOLTS. Cependant, dans un contexte avec N entités en interaction, il est nécessaire de préciser le modèle pour les interfaces et les liens entre entités. Soit M_i et M_j deux entités parmi les N en interaction : $i, j \in \{1, \ldots, N\}$, $i \neq j$. L'interface de M_i utilisée pour l'interaction avec M_j est alors notée l_{ij}. L_{ij} (resp. L_{ji}) représente le lien entre les interfaces l_{ij} de M_i (resp. l_{ji} de M_j) et l_{ji} de M_j (resp. l_{ij} de M_i). La notation U_i est toujours utilisée pour représenter une interface supérieure de M_i.
Avec une telle notation des liens et interfaces, l'évènement miroir $\bar{\mu}$ d'un évènement $\mu = l_{ij}!m$ (resp. $\mu = l_{ij}?m$) est défini par $\bar{\mu} = l_{ji}?m$ (resp. $\bar{\mu} = l_{ji}!m$).

5.4.2 Interactions de N systèmes

L'interaction asynchrone de N systèmes M_i ($i = \{1, \ldots, N\}$) est noté $\|_{\mathcal{A}}(M_i)_{i=\{1,\ldots,N\}}$ ou $\|_{\mathcal{A}}(M_i)$ dans le cas où il n'y a pas d'ambiguïté sur les valeurs possibles de i. Sa définition est basée sur les mêmes principes que la définition de l'interaction asynchrone de deux systèmes de la section 3.3.2 du chapitre 3. Dans cette section, nous donnons tout d'abord le définition de l'interaction synchrone de N systèmes, puis les règles permettant d'obtenier l'interaction asynchrone de N systèmes.

Définition 5.1 (Interaction synchrone de N systèmes)
Soit N IOLTS M_i, $i \in \{i, \ldots, N\}$, tels que chaque M_i est défini par $M_i = (Q^{M_i}, \Sigma^{M_i}, \Delta^{M_i}, q_0^{M_i})$. L'interaction synchrone des N IOLTS est définie par $\|_S(M_i)_{i=\{1\ldots,N\}} = (Q^{\|_S(M_i)}, \Sigma^{\|_S(M_i)}, \Delta^{\|_S(M_i)}, q_0^{\|_S(M_i)})$.
$Q^{\|_S(M_i)}$ est l'ensemble des états de l'interaction, et $q_0^{\|_S(M_i)}$ est l'état initial tel que $q_0^{\|_S(M_i)} \in Q^{\|_S(M_i)}$. L'identification de chaque état se fait à partir de l'état courant de chacun des N systèmes et de l'état des files d'attente des interfaces inférieures de ces systèmes.
$\Sigma^{\|_S(M_i)}$ est l'ensemble des évènements exécutables lors de l'interaction tel que $\Sigma^{\|_S(M_i)} \subseteq \bigcup_{i=\{1\ldots,N\}}(\Sigma^{M_i})$.
$\Delta^{\|_S(M_i)} = Q^{\|_S(M_i)} \times (\Sigma^{\|_S(M_i)} \cup \tau) \times Q^{\|_S(M_i)}$ est la relation de transition de l'interaction synchrone et est définie par les règles suivantes :

$$\frac{(q_i, a, q_i') \in \Delta^{M_i}, a \in \Sigma_U^{M_i} \cup \{\tau\}}{((q_1, ..., q_i, ..., q_N), a, (q_1, ..., q_i', ..., q_N)) \in \Delta^{\|_S(M_i)}} \quad (5.1)$$

$$\frac{(q_i, a, q_i') \in \Delta^{M_i}, (q_j, \bar{a}, q_j') \in \Delta^{M_j}, a \in \Sigma_L^{M_i}, \bar{a} \in \Sigma_L^{M_j}}{((q_1, ..., q_i, ..., q_j, ..., q_N), a, (q_1, ..., q_i', ..., q_j', ..., q_N)) \in \Delta^{\|_S(M_i)}} \quad (5.2)$$

$$\frac{(q_i, \delta, q_i) \in \Delta(M_i)}{((q_1, ..., q_i, ..., q_N), \delta(i), (q_1, ..., q_i, ..., q_N)) \in \Delta^{\|_S(M_i)}} \tag{5.3}$$

$$\frac{\forall i \in \{1 - N\},\ ((q_1, ..., q_i, ..., q_N), \delta(i), (q_1, ..., q_i, ..., q_N)) \in \Delta^{\|(M_i)}}{((q_1, ..., q_i, ..., q_N), \delta, (q_1, ..., q_i, ..., q_N)) \in \Delta^{\|_S(M_i)}} \tag{5.4}$$

Pour obtenir l'interaction asynchrone de N, nous modélisons le caractère asynchrone de la transmission entre deux des N systèmes par des files FIFO (comme dans la section 3.3.2 du chapitre 3). Ainsi, chaque état de cette interaction est défini par l'état courant de chaque système et l'état des files d'attente entre les systèmes. Les règles pour calculer l'interaction asynchrone sont donc les suivantes :

1. pour les évènements sur les interfaces supérieures (non concernée par l'interaction), la règle (5.1) de l'interaction synchrone s'appliquent.
2. pour les évènements sur les interfaces inférieures, les règles peuvent être décomposées en règles pour les sorties et règles pour les entrées.
   (a) Sortie sur une interface inférieure : le message envoyé est placé dans la file d'attente correspondante (cf. règle R2 de la définition 3.2 de la section 3.3.2).
   (b) Entrée sur une interface inférieure : le message doit être présent en première position dans la file d'attente FIFO pour pouvoir être reçu par le système. Les règles correspondantes sont de la même forme que la règle R3 de la définition 3.2 de la section 3.3.2.
3. pour les évènements de type blocage, les règles (5.3) et (5.4) de l'interaction synchrone s'appliquent.

Remarque : Cette définition est un modèle qui, dans la plupart des cas, ne pourra être calculée à cause de l'explosion du nombre d'états. Il est cependant nécessaire d'avoir un modèle de l'interaction asynchrone pour pouvoir ensuite s'intéresser à des traces particulières observées lors de l'interaction de N systèmes. De plus, cette interaction est utilisée pour modéliser différentes propriétés et ainsi pouvoir comparer des propriétés attendues mais non vérifiables avec d'autres propriétés vérifiables en pratique.

A partir du modèle de l'interaction de N systèmes et des modèles et opérations déjà définis pour le test d'interopérabilité, il est possible de définir formellement la notion d'interopérabilité de N implémentations. Tout d'abord, nous nous intéressons aux conditions qui doivent être vérifiées par les spécifications de ces systèmes, avant toute activité de test d'interopérabilité.

5.5 Conditions sur les spécifications

Le test d'interopérabilité de N implémentations vérifie entre autres la communication correcte entre ces systèmes. Il faut donc que les spécifications sur lesquelles sont basées ces implémentations permettent cette communication. Cette capacité de communication est définie, comme dans le contexte one-to-one (cf. section 3.4 du chapitre 3), par la propriété d'iop-compatibilité des spécifications (ou propriété de compatibilité des spécifications pour l'interopérabilité).

Propriété d'iop-compatibilité des spécifications Dans le contexte multi-implémentations, cette propriété est définie en deux étapes : iop-compatibilité de 2 spécifications pendant l'interaction avec les $N-2$ autres, puis iop-compatibilité des N spécifications. Tout d'abord, la propriété d'iop-compatibilité de deux spécifications (parmi les N considérées) s'intéresse aux échanges sur un lien entre deux systèmes lors de l'interaction des N spécifications. Cette propriété (cf. définition 5.2) dit que S_k et S_l ($k, l \in \{1, \ldots, N\}$, $k \neq l$) sont considérées iop-compatibles si et seulement si, il existe une connexion entre S_k et S_l lors de l'interaction des N systèmes ($L_{kl} = 1$), et, pour tout message envoyé par une des spécifications sur une de ses interfaces inférieures lors de l'interaction, la réception correspondante est prévue dans la spécification du récepteur. Dans le cas où $N = 2$, cette propriété est équivalente à la propriété d'iop-compatibilité du contexte one-to-one.

Définition 5.2 (Iop-compatibilité de 2 spécifications)
$IopComp(S_k, S_l) =_\Delta L_{kl} \wedge$
$\forall \sigma \in Traces(\|_\mathcal{A}(S_i)_{L_{kl}})$, $\forall \sigma.a.\sigma' \in Traces(\|_\mathcal{A}(S_i)_{L_{kl}})$, $a \in Out_L(\|_\mathcal{A}(S_i)_{L_{kl}}, \sigma)$, $\sigma' = \beta_1 ... \beta_m$, $\Rightarrow \exists \beta_i$ *tel que* $\beta_i = \bar{a}$.

Notation : dans la définition 5.2, $\|_\mathcal{A}(M_i)_{L_{kl}}$ est l'interaction de M_k et M_l pendant leur interaction avec les $N-2$ autres implémentations. $Out_L(M, \sigma)$ est l'ensemble des sorties exécutables *sur une interface inférieure* après la trace σ.

A partir de la définition de la propriété d'iop-compatibilité de deux spécifications, la propriété d'iop-compatibilité de N spécifications peut être définie. Cette propriété (cf. définition 5.3) est vérifiée si et seulement si $IopComp(S_k, S_l)$ est vérifiée quelque soit $\forall k, l \in \{1, \ldots, N\}$, $k \neq l$ tels qu'il existe un lien entre S_k et S_l.

Définition 5.3 (Iop-compatibilité de N spécifications)
$IopComp((S_i)_{i \in \{1,\ldots,N\}}) =_\Delta \forall k, l \in \{1, \ldots, N\}$, $k \neq l$, $L_{kl} \Rightarrow IopComp(S_k, S_l)$

Remarque 1 : la condition sur l'existence d'un lien entre deux systèmes (condition L_{kl}) est présente dans les deux définitions de propriétés. En effet, cette condition est

nécessaire dans la définition 5.3 car $IopComp$ s'appuie sur la vérification de la capacité à communiquer de toutes les entités interconnectées. Cette condition n'est a priori pas nécessaire dans la définition 5.2 si la propriété d'iop-compatibilité de deux spécifications n'est utilisée que dans la définition de la propriété d'iop-compatibilité de N spécifications. Mais l'absence de cette condition dans la définition 5.2 signifierait qu'il est possible de vérifier l'iop-compatibilité de deux spécifications alors qu'il n'existe pas de connexion entre elles. Pour cette raison, nous avons préféré conserver la redondance de la condition L_{kl} d'existence d'un lien entre les spécifications.

Remarque 2 : La condition d'iop-compatibilité de N spécification de la définition 5.3 est une condition nécessaire, mais pas suffisante. En effet, elle s'appuie sur la vérification de la propriété d'iop-compatibilité de pour chaque paire de spécifications interconnectées dans la topologie utilisée. Ce qui n'est pas pris en compte, c'est la vue globale du système, et donc l'influence que peuvent avoir des évènements exécutés par d'autres spécifications sur les deux spécifications S_k et S_l considérées. Cependant, une condition nécessaire et suffisante devrait s'appuyer sur un modèle de l'interaction des N spécifications permettant de connaître les ordonnancements possibles entre les évènements exécutés par les différents spécifications. Compte tenu du nombre d'ordonnancements généralement possibles pour les évènements de N systèmes, cette interaction de N spécifications est impossible à modéliser. Nous avons donc choisi de définir une condition nécessaire mais non suffisante pour la propriété d'iop-compatibilité de N spécifications.

Vérification de l'iop-compatibilité des spécifications Comme dans le contexte one-to-one (cf. section 3.4), il est possible de vérifier la propriété d'iop-compatibilité des N spécifications par simulation du comportement parallèle des modèles des N spécifications. Ce mode de vérification permet de déterminer quelles sont les traces pouvant mettre en échec la propriété d'iop-compatibilité. L'autre avantage de ce mode de vérification est que l'interaction des N spécifications est simulée. L'ordonnancement des évènements de ces N systèmes peut donc être pris en compte par ce mode de vérification, bien qu'il soit impossible de donner un modèle de l'interaction des N spécifications.

Input-complétude des spécifications Comme dans le contexte one-to-one (cf. section 3.4), il est possible de compléter les spécifications si nécessaire. Le principe est le même que dans le contexte one-to-one (cf. définition 3.4), pour une spécification S_i, les entrées pouvant être reçues en provenance des autres spécifications (S_k telles que $L_{ik} = 1$) sont ajoutées et les transitions correspondantes conduisent dans un état-puit appelé $trap$ tel que dans cet état, tout évènement de la spécification est exécutable.

5.6 Définitions formelles de l'interopérabilité de multi-implémentations

Dans cette section, nous nous intéressons aux définitions formelles de la notion d'interopérabilité de N ($N > 2$) implémentations.

5.6.1 Principes généraux

Le test d'interopérabilité multi-implémentations est utilisé pour vérifier que les N ($N > 2$) implémentations interconnectées pour le test sont capables de communiquer correctement tout en fournissant le service prévu. Les propriétés à vérifier en contexte multi-implémentations sont donc les mêmes qu'en contexte one-to-one (cf. section 3.5.1) :

1. **Propriété *Pr_Int* (vérification de l'interaction) :** les N ($N > 2$) implémentations doivent communiquer correctement. C'est-à-dire que les messages envoyés par une implémentation à une autre du SUT (composé des N implémentations) doivent être effectivement reçus par l'implémentation réceptrice : c'est la propriété ***Pr_Int_1***. De plus, ces messages doivent être des messages correspondant à ceux qui sont prévus dans la spécification de l'émetteur, ce qui correspond à la propriété ***Pr_Int_2***.
2. **Propriété *Pr_Serv* (vérification du service) :** les messages envoyés par les implémentations (durant l'interaction) sur leurs interfaces supérieures doivent correspondre au service décrit dans les spécifications respectives.

Ainsi, il faut à la fois vérifier que les sorties sur les différentes interfaces des implémentations sont correctes par rapport aux spécifications correspondantes, et que les messages servant à l'interaction sont effectivement reçus.
La vérification des sorties se fait en comparant les sorties observées aux interfaces des interfaces des implémentations avec les sorties prévues dans les spécifications correspondantes. La vérification des entrées se fait, comme en contexte one-to-one (cf. section 3.5.3) en calculant l'ensemble des sorties dépendances causales de l'entrée à vérifier.
Dans la section suivante, nous donnons les critères d'interopérabilité basés sur ces propriétés et considérant les architectures de test d'interopérabilité multi-implémentations définies dans la section 5.2.

5.6.2 Critères d'interopérabilité multi-implémentations

Dans cette section, nous présentons différentes définitions formelles de la notion d'interopérabilité de N implémentations. Ces définitions considèrent N implémentations $I_i = (Q^{I_i}, \Sigma^{I_i}, \Delta^{I_i}, q_0^{I_i})$ ($i \in \{1, \cdots, N\}$) implémentant chacune une spéci-

fication $S_i = (Q^{S_i}, \Sigma^{S_i}, \Delta^{S_i}, q_0^{S_i})$. Comme dans le contexte one-to-one, nous appelons ces définitions des critères d'interopérabilité. Les critères d'interopérabilité multi-implémentations sont également appelés critères M-iop (pour "multi-implementation interoperability").

Le **critère M-iop global** $iopM_G$ (Global M-iop criterion) dit que N implémentations sont considérées interopérables ssi :

1. après une trace (suspensive) de l'interaction asynchrone des implémentations (prévue dans l'interaction des spécifications), toutes les sorties (comprenant également les silences) observées pendant l'interaction asynchrone des implémentations sont prévues (décrites) dans les spécifications.
2. *Et* les messages envoyés par une des implémentations via une interface inférieure à une autre implémentation qui lui est directement connectée doivent être reçus (traités) par l'implémentation réceptrice.

Définition 5.4 (Critère M-iop global $iopM_G$)
$iopM_G(I_1, ..., I_N) = \forall \sigma \in Traces(\|_{\mathcal{A}}(S_i)), Out(\|_{\mathcal{A}}(I_i), \sigma) \subseteq Out(\|_{\mathcal{A}}(S_i), \sigma)$ *et*
$\forall \{i, j\}$ *tels que* $L_{ij} = 1$,
$\forall \sigma \in Traces(\|_{\mathcal{A}}(S_n)_{L_{ij}})$, $\sigma_i = \sigma/\Sigma^{S_i} \in Traces(\Delta(S_i))$, $\sigma_j = \sigma/\Sigma^{S_j} \in Traces(\Delta(S_j))$, $\forall \mu \in Out(I_i, \sigma_i)$, $\forall \sigma' \in [(\Sigma^{S_j} \cup \Sigma^{S_i}) \setminus \{\bar{\mu}\}]^* \cup \{\epsilon\}$, $\sigma.\mu.\sigma'.\bar{\mu} \in Traces(\|_{\mathcal{A}}(S_n)_{L_{ij}})$, $\bar{\mu} \in In(I_j, \sigma_j.(\sigma'/\Sigma^{I_j})) \Rightarrow Out(I_j, \sigma_j.(\sigma'/\Sigma^{I_j}).\bar{\mu}.\sigma_k) \in Cdep(S_j, \sigma_j.(\sigma'/\Sigma^{I_j}), \bar{\mu})$, $\sigma_k \in (\Sigma_I^{S_j})^* \cup \{\epsilon\}$

Le **critère M-iop unilatéral** $iopM_U$ (Unilateral M-iop criterion) dit qu'une implémentation I_i est considérée interopérable avec $N-1$ autres implémentations ssi :

1. après une trace de $\Delta(S_i)$ observée lors de l'interaction des N implémentations, toutes les sorties et silences observés sur les interfaces de I_i sont décrites/ prévues dans $\Delta(S_i)$.
2. *Et* I_i doit être capable de recevoir les messages envoyés par les implémentations qui lui sont directement (sous réserve que ces messages soient prévus dans la spécification de l'émetteur).

Définition 5.5 (Critère M-iop unilatéral $iopM_U$)
$I_i\ iopM_U\ ((I_j)_{j\in\{1,\cdots,N\}\setminus i}) = \forall \sigma_i \in Traces(\Delta(S_i)), \forall \sigma \in Traces(\|_{\mathcal{A}}(S_j)), \sigma/\Sigma^{S_i} = \sigma_i \Rightarrow Out((\|_{\mathcal{A}}(I_i))/\Sigma^{S_i}, \sigma_i) \subseteq Out(\Delta(S_i), \sigma_i)$ *et*
$\forall j$ *tel que* $L_{ji} = 1$,
$\forall \sigma \in Traces(\|_{\mathcal{A}}(S_n)_{L_{ij}})$, $\sigma_i = \sigma/\Sigma^{S_i} \in Traces(\Delta(S_i))$, $\sigma_j = \sigma/\Sigma^{S_j} \in Traces(\Delta(S_j))$, $\forall \mu \in Out(I_i, \sigma_i)$, $\forall \sigma' \in [(\Sigma^{S_j} \cup \Sigma^{S_i}) \setminus \{\bar{\mu}\}]^* \cup \{\epsilon\}$, $\sigma.\mu.\sigma'.\bar{\mu} \in Traces(\|_{\mathcal{A}}(S_n)_{L_{ij}})$, $\bar{\mu} \in In(I_j, \sigma_j.(\sigma'/\Sigma^{I_j})) \Rightarrow Out(I_j, \sigma_j.(\sigma'/\Sigma^{I_j}).\bar{\mu}.\sigma_k) \in Cdep(S_j, \sigma_j.(\sigma'/\Sigma^{I_j}), \bar{\mu})$, $\sigma_k \in (\Sigma_I^{S_j})^* \cup \{\epsilon\}$

Le **critère M-iop multi-unilatéral** $iopM_{M-U}$ (Multi-unilateral M-iop criterion) est vérifié si les N critères M-iop unilatéraux sont vérifiés.

Définition 5.6 (Critère M-iop multi-unilatéral $iopM_{M-U}$)
$iopM_{M-U}(I_1, ..., I_N) = \forall i \in \{1, \cdots, N\}\ I_i\ iopM_U\ ((I_j)_{j\in\{1,\cdots,N\}\setminus\{i\}})$

Critères d'interopérabilité multi-implémentations orientés interfaces inférieures ou supérieures De même que dans le contexte one-to-one, il est possible de définir des critères d'interopérabilité M-iop supérieure ou inférieure. Ces critères d'interopérabilité peuvent être déduits des critères d'interopérabilité multi-implémentations totales grâce à des projections sur les interfaces accessibles. Le principe pour déduire les critères M-iop orientés interfaces inférieures ou supérieures des critères M-iop totale est basé sur le même type de projection que celle qui permet de déduire les critères d'interopérabilité orientés interfaces inférieures ou supérieures du contexte one-to-one (cf. sections 3.6.2 et 3.6.3).

5.6.3 Comparaison entre critères d'interopérabilité

Dans la section 3.8 du chapitre 3, nous avons prouvé l'équivalence, en terme de détection de la non-interopérabilité, des critères d'interopérabilité bilatérale totale et globale totale. Dans cette section, nous démontrons l'équivalence entre deux critères appelés critère M-iop multi-unilatéral $iopM_{M-U}$ et critère M-iop global $iopM_G$. Cette preuve est une généralisation de la preuve du chapitre 3.

Théorème 5.1 $iopM_G(I_1, ..., I_N) \Leftrightarrow iopM_{M-U}(I_1, ..., I_N)$

La preuve du théorème 5.1 nécessite la preuve de trois lemmes préliminaires.

Lemme 5.1 *Considérons N IOLTS $M_1, M_2, \cdots, M_N$, et une trace σ telle que $\sigma \in Traces(\|_{\mathcal{A}}(M_i))$,*

$$Out(\|_{\mathcal{A}}(M_i), \sigma) = \cup_{i=\{1,\cdots,N\}} Out(\Delta(M_i), \sigma/\Sigma^{M_i})$$

Preuve: Cette équivalence reflète le caractère asynchrone de l'interaction entre les IOLTS. Sa preuve est basée sur la définition de l'interaction asynchrone de N systèmes.

1. Si une sortie est exécutée par le système composé de N entités, cela signifie qu'une de ces entités doit avoir exécuter cette sortie : $Out(\|_{\mathcal{A}}(M_i), \sigma) \subseteq \cup_{i=\{1,\cdots,N\}} Out(\ \Delta(M_i), \sigma/\Sigma^{M_i})$
2. D'après les règles de la définition de l'interaction asynchrone entre N entités composant le système global et sur le même principe que la preuve du lemme 3.1 de la section 3.8, on obtient également que :
   $\cup_{i=\{1,\cdots,N\}} Out(\Delta(M_i), \sigma/\Sigma^{M_i}) \subseteq Out(\|_{\mathcal{A}}(M_i), \sigma)$.

Lemme 5.2 *Soit* $M_1, M_2, \cdots, M_N \in \mathcal{IOLTS}$. *On a :*

$$(\|_{\mathcal{A}}(M_i)/\Sigma^{M_1})\|_{\mathcal{A}} \cdots \|_{\mathcal{A}}(\|_{\mathcal{A}}(M_i)/\Sigma^{M_N}) = (\|_{\mathcal{A}}(M_i))$$

c'est-à-dire $\|_{\mathcal{A}}(\|_{\mathcal{A}}(M_i)/\Sigma^{M_k}) = (\|_{\mathcal{A}}(M_i))$

Preuve:

1. Soit $\sigma_i \in Traces((\|_{\mathcal{A}}(M_i))/\Sigma^{M_k}))\ \forall i \in \{1, \cdots, N\}$.
   Soit $\sigma = \|_{\mathcal{A}}(\sigma_i) \in Traces(\|_{\mathcal{A}}(\|_{\mathcal{A}}(M_i))/\Sigma^{M_k}))$.
   Alors, $\sigma_i \in Traces(\Delta(M_i))$.
   Ainsi, $\sigma = \|_{\mathcal{A}}(\sigma_i) \in Traces(\|_{\mathcal{A}}(M_i))$.
2. Soit $\sigma \in Traces(\|_{\mathcal{A}}(M_i))$ tel que $\sigma = \|_{\mathcal{A}}(\sigma_i)$ avec $\sigma_i \in Traces(\Delta(M_i))\ \forall i \in \{1, \cdots, N\}$. Nous avons $\sigma_i = \sigma/\Sigma^{M_i}$.
   Ainsi $\sigma_i \in Traces(((\|_{\mathcal{A}}(M_i))/\Sigma^{M_i}))$
   et $\sigma = \|_{\mathcal{A}}(\sigma_i) \in Traces((\|_{\mathcal{A}}(\|_{\mathcal{A}}(M_i)/\Sigma^{M_k}))))$.

Lemme 5.3 *Soit* $M_1, \cdots, M_N \in \mathcal{IOLTS}$, $\forall i \in \{1, \cdots, N\}, \sigma_i \in Traces(\Delta(M_i))$ *et* $\sigma_i = \sigma/\Sigma^{M_i}$.

$$Out((\|_{\mathcal{A}}(M_k))/\Sigma^{M_i}, \sigma/\Sigma_{M_i}) \subseteq Out(\Delta(M_i), \sigma_i)$$

Preuve: $(\|_{\mathcal{A}}(M_k))/\Sigma^{M_i}$ est un IOLTS composé d"évènements de $\Sigma^{(\|_{\mathcal{A}}(M_k))/\Sigma^{M_i}} \subseteq \Sigma^{M_i}$.

A partir de ces trois lemmes, il est possible de prouver l'équivalence entre les conditions vérifiant les sorties des critères M-iop global et M-iop multi-unilatéral. De plus, les conditions vérifiant l'exécution des sorties sont les mêmes dans le deux critères. La preuve dans la suite servant à vérifier l'équivalence entre conditions sur les sorties est donc suffisante pour prouver le théorème 3.1.

Preuve: 1) Prouvons d'abord que la condition sur les sorties de $iopM_{M-U}(I_1, ..., I_N)$ implique la condition sur les sorties de $iopM_G(I_1, ..., I_N)$.
Soit $\sigma \in Traces(\|_{\mathcal{A}}(S_k))$, $\sigma_j \in Traces(\Delta(S_j))$ tel que $\sigma_j = \sigma/\Sigma^{S_j}\ \forall j \in \{1, \cdots, N\}$.
Selon $iopM_{M-U}(I_1, ..., I_N)$, on a :

$$\forall j \in \{1, \cdots, N\}, Out((\|_{\mathcal{A}}(I_k))/\Sigma^{S_j}, \sigma_j) \subseteq Out(\Delta(S_j), \sigma_j) \quad .$$

Ainsi, $Out((\|_{\mathcal{A}}(I_k))/\Sigma^{S_1}, \sigma_1) \cup \cdots \cup Out((\|_{\mathcal{A}}(I_k))/\Sigma^{S_N}, \sigma_N) \subseteq Out(\Delta(S_1), \sigma_1) \cup \cdots \cup Out(\Delta(S_N), \sigma_N)$.
D'après le lemme 5.1, $Out(\|_{\mathcal{A}}(\|_{\mathcal{A}}(I_k)/\Sigma^{S_j}), \sigma) \subseteq Out(\|_{\mathcal{A}}(S_k), \sigma)$.
D'après le lemme 5.2, $Out(\|_{\mathcal{A}}(I_k), \sigma) \subseteq Out(\|_{\mathcal{A}}(S_k), \sigma)$.
2) Prouvons maintenant que la condition sur les sorties de $iopM_G(I_1, ..., I_N)$ implique la condition sur les sorties de $iopM_{M-U}(I_1, ..., I_N)$.
Soit $I_1, \cdots, I_N$, $S_1, \cdots, S_N$ tels que $iopM_G(I_1, ..., I_N)$. Soit $\sigma_i \in Traces(\Delta(S_i))$ tel que $\sigma_i = \sigma/\Sigma^{S_i}$ avec $\sigma \in Traces(\|_{\mathcal{A}}(S_k))\ \forall i \in \{1, \cdots, N\}$.
D'après $iopM_G(I_1, ..., I_N)$, on a : $Out(\|_{\mathcal{A}}(I_k), \sigma) \subseteq Out(\|_{\mathcal{A}}(S_k), \sigma)$.

Après une projection sur Σ^{S_i} : $\forall i \in \{1, \cdots, N\}$, $Out(\|_{\mathcal{A}}(I_k)/\Sigma^{S_i}, \sigma/\Sigma^{S_i}) \subseteq Out(\|_{\mathcal{A}}(S_k)/\Sigma^{S_i}, \sigma/\Sigma^{S_i})$.
D'après le lemme 5.3 : $Out(\|_{\mathcal{A}}(I_k)/\Sigma^{S_i}, \sigma_i) \subseteq Out(\Delta(S_i), \sigma_i)\ \forall i \in \{1, \cdots, N\}$.

Ainsi, nous avons prouvé que le critère M-iop global et le critère M-iop multi-unilatéral sont équivalent en terme de détection de la non-interopérabilité. De plus, le critère M-iop unilatéral a un pouvoir de détection de la non-interopérabilité inférieure à celui de ces deux critères, mais la vérification du critère M-iop global ou du critère M-iop multi-unilatéral implique la vérification de N critères M-iop unilatéraux correspondants.

5.7 Application des définitions formelles sur un exemple

Dans cette section, nous illustrons les critères d'interopérabilité M-iop à l'aide d'une application de ces critères sur un exemple comportant trois spécifications. Cet exemple est représenté dans la figure 5.3. S_2 peut être vu comme un boîtier de chiffrement qui relaie les messages de S_1 vers S_3 et vice-versa. Il y a également un lien direct entre S_1 et S_3. La topologie connectant ces trois systèmes est également représentée dans la figure et est telle que : $\forall i \in \{1, 2, 3\}$, $\forall j \in \{1, 2, 3\}$, $i \neq j$, $L_{ij} = 1$.

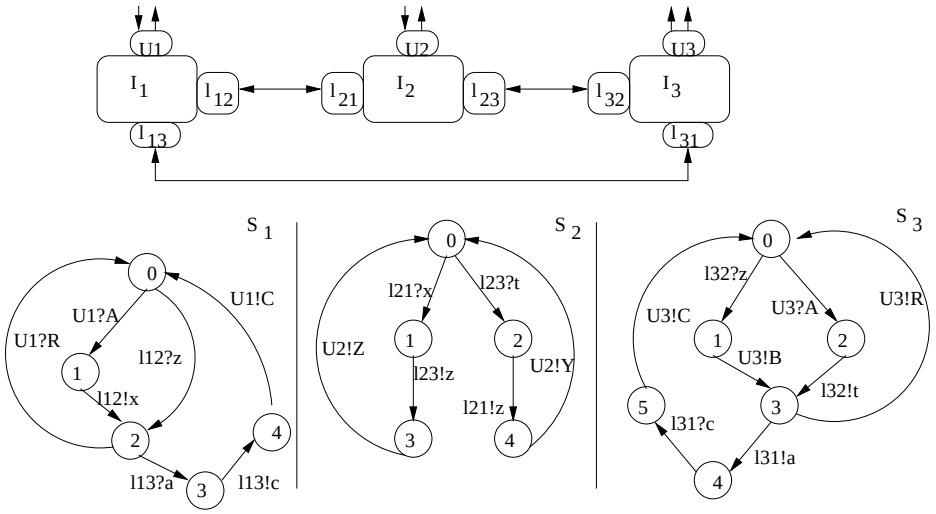


FIG. 5.3 – Exemple de spécifications (N=3) : communication via un relais

Description du protocole : S_1 reçoit une demande de service de la part de la couche supérieure : évènement $U1?A$. La demande est transmise à S_2 (évènements $l12!x$ et $l21?x$) qui traduit et envoie à S_3 (évènement $l23!z$), puis informe sa couche supérieure de cette action (évènement $U2!Z$). S_3 reçoit la demande de service (évènement

$l32?z$) et transmet à sa couche supérieure (évènement $U3!B$). Deux choix sont alors possibles : soit S_3 choisit de ne pas répondre à la demande de service et transmet l'information à sa couche supérieure (évènement $U3!R$ avec retour à l'état initial), soit S_3 répond directement à S_1 (évènements $l31!a$ et $l13?a$). S_1 accuse alors réception de la réponse de S_3 (évènements $l13!c$ et $l31?c$), et les deux systèmes transmettent cette information a leur couche supérieure (évènements $U1!C$ et $U3!C$). Une communication du même type est également possible si la demande de service est faite par la couche supérieure de S_3 (évènement $U3?A$). Dans ce cas, S_1 reçoit consécutivement le message relayé par S_2 (évènement $l12?z$) et le message direct de S_3 (évènement $l13?a$). S_1 est également doté d'une possibilité de "reset" (évènement $U1?R$).

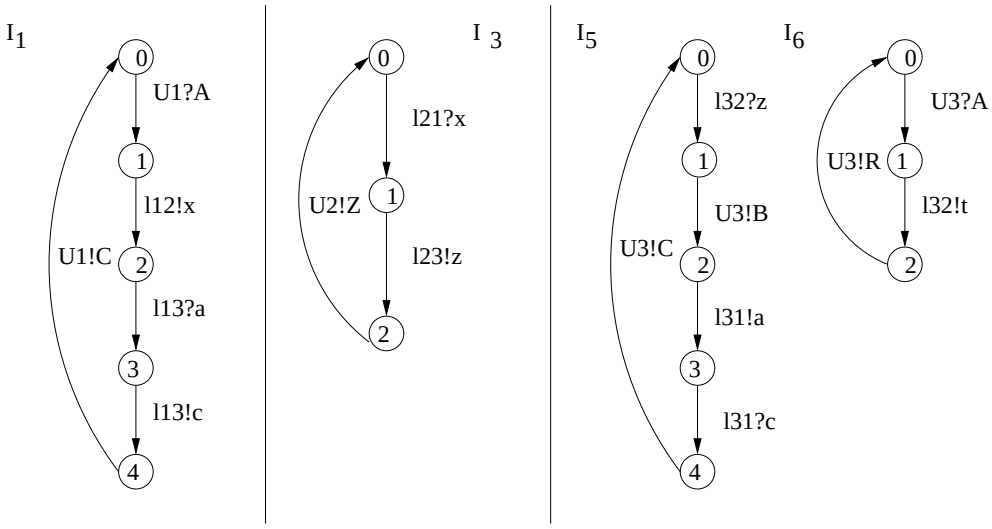


FIG. 5.4 – Implémentations de spécifications de la figure 5.3

Considérons les implémentations de la figure 5.4. I_1 et $I_2 = S_1$ sont des implémentations de S_1, I_3 et $I_4 = S_2$ des implémentations de S_2, et I_5, I_6, et $I_7 = S_3$ des implémentations de S_3. Étudions les interactions suivantes.

– Interaction entre I_1, I_3 et I_5. Avec les critères d'interopérabilité de la section 5.6.2, on a les résultats suivants : $iopM_G(I_1, I_3, I_5)$ et $iopM_{M-U}(I_1, I_3, I_5)$. En effet, ces trois implémentations n'implémentent pas complètement leurs spécifications respectives. Mais chacune d'entre elles est capable de recevoir les messages qui lui sont envoyés par les deux autre. Et aucune sortie non prévue par les spécifications n'est exécutée lors de l'interaction de ces trois implémentations.

– Interaction entre I_1, $I_4 = S_2$ et I_6. Avec les critères d'interopérabilité de la section 5.6.2, on a les résultats suivants : $\neg iopM_G(I_1, I_4, I_6)$ et $\neg iopM_{M-U}(I_1, I_4, I_6)$. Au niveau des critères M-iop unilatéral, on a $\neg I_6\ iopM_U\ (I_1, I_4)$ mais I_4

$iopM_U\ (I_1, I_6)$. En effet, lors de l'interaction de ces trois implémentations, I_6 n'exécute pas de sortie non prévue dans leur spécification, et est capable de recevoir les messages envoyés par les deux autres entités. Par contre, I_6 n'est pas capable de recevoir le message z envoyé par I_4 (évènement $l23!z$) : la sortie $U3!B$ dépendance causale de $l32?z$ n'est pas observé ce qui permet de détecter que z n'a pas été effectivement reçu.

Dans le cas du critère d'interopérabilité unilatérale du point de vue de I_1 interagissant avec I_4 et I_6, aucun message non prévu dans la spécification S_1 n'est observée lors de l'interaction. Le problème concerne un scénario d'interaction avec une demande de service de I_6. Ainsi, I_6 exécute les évènements $U3?A$ puis $l32!t$. Le message t est reçu par $I_4 = S_2$ (évènement $l23?t$) qui envoie z à I_1 (évènement $l21!z$). I_1 ne prévoit pas la réception de ce message z (évènement $l12?z$ de S_1 non implémenté par I_1). Mais la vérification de l'exécution de cette entrée peut poser problème. En effet, les traces dans S_1 permettant d'observer des sorties dépendances causales de $l12?z$ sont $l13?a.l13!c$ et $U1?R.U1?A.l12!x$. Dans le premier cas, la sortie $l13!c$ est donc une dépendance causale de $l12?z$ et $l13?a$. L'envoi de a par I_6 n'étant pas contrôlable, la vérification de l'exécution de $l12?z$ via cette trace renverra un verdict $Inconclusive$. Par contre, $U1?R$ et $U1?A$ sont deux évènements contrôlables correspondant à des stimuli du testeur. L'exécution de ces deux stimuli sans la sortie $l12!x$ permet donc de détecter la non-interopérabilité. Cependant, le deuxième stimuli correspondant à une deuxième demande de service ($U1?A$) que les testeurs peuvent donc choisir de ne pas envoyer en fonction de l'objectif de leurs tests. Ainsi, cette interaction correspondant donc à une situation où $\neg I_1\ iopM_U\ (S_2, I_6)$ est difficile à vérifier en pratique et risque d'aboutir à des verdicts $Inconclusive$.

– Interaction entre I_2, I_3 et I_7. Rappelons que $I_2 = S_1$ et $I_7 = S_3$. Avec les critères d'interopérabilité de la section 5.6.2, on a les résultats suivants : $\neg iopM_G(I_1, I_3, I_7)$ et $\neg iopM_{M-U}(I_1, I_3, I_7)$. Au niveau des critères M-iop unilatéral, on a $\neg I_3\ iopM_U\ (I_2, I_7)$, mais $I_2\ iopM_U\ (I_3, I_7)$ et $I_7\ iopM_U\ (I_2, I_3)$. En effet, la non-interopérabilité de ces trois implémentations est détectée lorsque I_7 envoie t à I_3 qui n'est pas capable de traiter ce message et donc d'envoyer la sortie dépendance causale z.

Dans les trois exemples d'interaction, la complexité de la vérification provient des dépendances entre les messages de chacune de trois implémentations : l'exécution des évènements d'une implémentation I_i dépendent ici des exécutions des évènements des deux autres implémentations connectées I_j et I_k. Par exemple, dans l'exemple d'interaction entre I_1, I_3 et I_5, chacune des trois implémentations communiquent avec les deux autres lors de l'interaction. Dans le cas de l'interaction entre I_1, I_4 et I_6, la sor-

tie $U3!B$ dont l'absence permet de détecter la non-interopérabilité doit être exécutée après échange de messages entre I_1 et I_4, puis entre I_4 et I_6. C'est cette complexité des échanges (ici avec pourtant seulement trois systèmes) qui rend également complexe la description des cas de test permettant de vérifier l'interopérabilité dans une tel contexte.

Conclusion : nous avons appliqué les critères d'interopérabilité multi-implémentations sur un exemple comportant trois entités interconnectées. Les critères d'interopérabilité définis permettent de détecter les situations de non-interopérabilité introduites dans les implémentations proposées. En particulier, quand les systèmes interconnectés contiennent des choix d'implémentations différents, la non-interopérabilité est détectée grâce à la vérification de l'exécution effective des entrées. De plus, cette application sur un exemple confirme l'équivalence en terme de détection de la non-interopérabilité entre les critères M-iop multi-unilatéral et M-iop global. Ceci suggère la possibilité d'utiliser les mêmes approches que celles du contexte one-to-one.

5.8 Problématique de la génération de test

Nous avons défini dans ce chapitre des critères d'interopérabilité multi-implémentations. Nous avons également prouvé l'équivalence entre les critères M-iop global et multi-unilatéral. Dans le chapitre 4, nous avons défini une méthode de génération de test d'interopérabilité en contexte one-to-one basée sur l'équivalence entre le critère d'interopérabilité globale totale et le critère d'interopérabilité bilatérale totale. L'équivalence entre critère M-iop global et critère M-iop multi-unilatérale suggère donc qu'un approche similaire à l'approche bilatérale définie pour le contexte one-to-one puisse être possible en contexte de test d'interopérabilité multi-unilatéral.

Cependant, la complexité d'une telle approche pour la génération de tests d'interopérabilité est liée au nombre de systèmes à considérer dans le contexte multi-implémentations. En particulier, cette complexité intervient dans l'étape de dérivation des objectifs de test d'interopérabilité unilatérale. En contexte multi-implémentations, il faut pouvoir dériver cet objectif de test d'interopérabilité global en N objectifs de test unilatéraux (autant que d'implémentations interconnectées). Ceci implique de calculer, pour un évènement de l'objectif de test global μ tel que $\mu \in \Sigma^{S_i}$, un évènement lié causalement sur chacune des $N-1$ autres spécifications S_j ($j \in \{1, \cdots, N\} \setminus \{i\}$). Or, la propriété d'iop-compatibilité des spécifications (cf. section 5.5) est une propriété vérifiée pour des paires de spécifications lors de l'interaction des N systèmes. Cela signifie que les dépendances entre évènements de plusieurs spécifications sont des dépendances considérant des paires de spécifications (et basées, comme en contexte one-to-one, sur les dépendances entre un envoi et la réception du message correspondant).

Cependant, la dérivation d'un objectif de test d'interopérabilité global en N objectifs de test unilatéraux nécessite le calcul de dépendances multiples. En effet, pour chacun des évènements $\mu \in \Sigma^{S_i}$ décrits dans l'objectif de test global, un évènement dépendant causalement de μ sur chacune des $N-1$ spécifications S_j ($j \in \{1, \cdots, N\} \setminus \{i\}$) doit être calculé qu'il existe un lien connectant S_j à S_i ou qu'il n'en existe pas. Ainsi, il faut pouvoir calculer des dépendances multiples entre évènements, et ceci en utilisant la transitivité des dépendances entre évènements des N spécifications. Ces calculs de dépendances entre évènements correspondent donc à une opération très complexe. C'est donc cette opération qu'il faut pouvoir minimiser pour appliquer au contexte du test d'interopérabilité multi-implémentations une approche similaire à celle définie pour le contexte du test d'interopérabilité one-to-one.

Par contre, une fois ces dépendances multiples entre évènements calculées, il est aisé de générer des cas de test d'interopérabilité à partir des objectifs de test unilatéraux ainsi obtenus. Cette opération consiste en effet à générer N cas de test unilatéraux à partir des N objectifs de test unilatéraux dérivés et des N spécifications correspondantes. C'est donc une opération qui correspond à un parcours simple de graphe : les chemins permettant d'exécuter un objectif de test unilatéral sont cherchés dans la spécification correspondante. Comme il n'est pas nécessaire de construire l'interaction des N spécifications (ni totale, ni partielle), ni l'interaction de certaines de ces spécifications parmi les N spécifications, cette opération évite le problème d'explosion combinatoire du nombre d'états.
De plus, les cas de test d'interopérabilité ainsi obtenus sont des cas de test "complètement" distribués. En effet, ils correspondent à un contexte où chaque testeur parallèle s'occupe de l'implémentation à laquelle il est dédié. De tels cas de test évitent également la synchronisation entre testeurs qui est une opération complexe et source d'erreurs. Une telle approche facilite ainsi la phase d'exécution des tests d'interopérabilité multi-implémentations.

Une méthode complète pour calculer les objectifs de test unilatéraux serait donc une contribution importante pour le test d'interopérabilité multi-implémentations. De plus, l'étude réalisée sur les topologies (cf. section 5.3) permettrait d'organiser avantageusement la compagne de tests.

Chapitre 6

Conclusion et perspectives

Dans le domaine des réseaux, le test d'interopérabilité est utilisé pour vérifier que plusieurs implémentations (basées sur des protocoles conçus pour fonctionner ensemble) sont capables d'interagir et que, lors de leur interaction, elles rendent les services prévus dans la ou les spécifications correspondantes. Contrairement au cas du test de conformité, il n'existe pas de cadre méthodologique normalisé pour le test d'interopérabilité. Les tests d'interopérabilité sont généralement dérivés manuellement et basés sur une définition de l'interopérabilité qui peut être différente d'un test à l'autre. Tout cela a pour conséquence une définition souvent floue de la notion d'interopérabilité, et une non-interopérabilité effective en mode opérationnel des implémentations malgré les tests.

L'objectif des travaux présentés dans cette thèse était donc de préciser les notions relatives au test d'interopérabilité en terme d'architectures de test, de définitions de la notion d'interopérabilité et de méthodes de génération de test. Dans la pratique, les tests d'interopérabilité sont principalement appliqués pour des contextes où deux implémentations sont interconnectées. De ce fait, les contributions de cette thèse adressent majoritairement ce contexte qualifié de contexte de test d'interopérabilité one-to-one. Cependant, quelques études ont également été réalisées pour le test d'interopérabilité de N ($N > 2$) implémentations, appelé contexte de test d'interopérabilité multi-implémentations.

Contexte du test d'interopérabilité one-to-one

Dans cette thèse, nous avons présenté une approche formelle du test d'interopérabilité de deux implémentations. Ce contexte, appelé contexte one-to-one, constitue la majeure partie de l'activité actuelle de test d'interopérabilité : en effet, la plus grande partie des tests d'interopérabilité actuels consiste à vérifier soit l'interopérabilité de deux implémentations, soit l'interopérabilité d'une implémentation avec un deuxième

système qui peut être composé de plusieurs implémentations. L'étude de ce contexte peut être décomposée en plusieurs phases. Ces phases correspondent à la classification des architectures de test d'interopérabilité possibles en contexte one-to-one, à la définition formelle de la notion d'interopérabilité de deux systèmes, et à la définition d'une méthode de génération de tests d'interopérabilité basée sur ces définitions formelles.

Une classification des architectures de test d'interopérabilité possibles en contexte one-to-one avait été réalisée auparavant. Les architectures de test d'interopérabilité y sont classifiées en fonction des types d'interfaces auxquelles les testeurs ont accès et en fonction du mode d'accès aux implémentations testées. Dans cette thèse, nous avons reprécisé cette classification en se focalisant plus particulièrement sur les éléments à prendre en compte dans la définition formelle de la notion d'interopérabilité et dans la génération de tests associée.

Sur la base de ces différentes architectures de test d'interopérabilité, nous avons proposé plusieurs définitions formelles de la notion d'interopérabilité, appelées *critères d'interopérabilité*. Ces critères d'interopérabilité définissent la notion d'interopérabilité grâce à deux propriétés sur les systèmes testées. La première propriété est la capacité des implémentations à interagir, c'est-à-dire que chacune des implémentations doit envoyer des messages corrects (par rapport à la spécification) et que ces messages doivent être reçus par l'implémentation homologue. La seconde propriété est la capacité des implémentations à rendre le service prévu dans leur spécification lors de cette interaction.

Nous avons également étudié les propriétés de ces critères d'interopérabilité. En particulier, nous avons comparé les critères d'interopérabilité en terme de capacité de détection de la non-interopérabilité. Cette comparaison permet de guider le choix du critère à utiliser pour la génération de tests. Elle a également permis de mettre en évidence l'équivalence entre deux critères : le critère d'interopérabilité globale totale et le critère d'interopérabilité bilatérale totale qui correspondent à deux contextes de test d'interopérabilité distribué (respectivement avec et sans coordination entre les testeurs).

Nous avons ensuite étudié les approches de génération de test pouvant être dérivées de ces critères d'interopérabilité. En particulier, nous avons défini une méthode de génération de cas de test d'interopérabilité à partir d'un objectif de test basée sur l'équivalence entre le critère d'interopérabilité globale totale et le critère d'interopérabilité bilatérale totale. En effet, le critère d'interopérabilité globale totale correspond à la définition de l'interopérabilité actuellement la plus utilisée en pratique, mais la génération de test basée sur un tel critère rencontre généralement un problème d'explosion combinatoire dû à la vue globale du système composé des deux implémentations (et au calcul de l'interaction des spécifications que cette vue entraîne). Or, le critère d'inter-

opérabilité bilatérale totale ne s'appuie pas sur une telle vue du système sous test. La génération de tests basée sur ce critère d'interopérabilité permet ainsi d'éviter le problème d'explosion combinatoire cité ci-dessus. De plus, d'après l'équivalence entre critères bilatéral total et global total, les cas de test générés permettent de détecter les mêmes situations de non-interopérabilité que les cas de test qui seraient générés avec une approche globale.

De plus, cette approche bilatérale est complétée de façon à pouvoir vérifier l'exécution des entrées par l'implémentation réceptrice. En effet, les entrées ne sont pas des évènements observables. Pour vérifier leur exécution, nous proposons une méthode basée sur le calcul des sorties dépendances causales de l'entrée dont on veut vérifier la prise en compte par l'implémentation.

Enfin, notre méthode de génération de tests d'interopérabilité a été appliquée sur des exemples de protocoles. Les cas de test générés ont été comparés aux cas de test générés avec l'approche globale. Cette application a permis de confirmer que la méthode bilatérale permet bien d'éviter le problème d'explosion combinatoire rencontré avec l'approche globale. Cette application a également permis de confirmer que les cas de test d'interopérabilité générés avec la méthode bilatérale pouvaient détecter les situations de non-interopérabilité qui seraient détectées avec une approche globale, malgré l'absence de synchronisation entre testeurs.

Les cas de test d'interopérabilité générés par la méthode proposée dans cette thèse suggèrent une approche distribuée du test d'interopérabilité. Dans une telle approche, un testeur est connectée à chacune des implémentations et est dédié à l'observation/contrôle des évènements exécutés par cette implémentation. L'approche de génération de test proposée permet de générer des cas de test d'interopérabilité distribuée. De plus, il n'est pas nécessaire de dériver de procédure de synchronisation entre les testeurs pour l'exécution des cas de test générés.

Le travail présenté dans cette thèse concerne la phase de spécification des tests, c'est-à-dire les phases de l'activité de test d'interopérabilité de deux implémentations qui vont de la spécification des systèmes à tester et des objectifs de test jusqu'à la génération des cas de test abstraits. Comme suite à ce travail, il conviendrait d'étudier les phases suivantes de l'activité de test, c'est-à-dire la traduction de ces cas de test abstraits en cas de test exécutables, et l'exécution de ces cas de test sur les implémentations à tester. Une telle étude permettrait entre autres de confirmer le lien entre notre méthode de génération de test et l'approche distribuée du test d'interopérabilité. Cette étude permettrait également d'adresser les problèmes que peuvent poser les opérations de traduction de cas de test d'interopérabilité abstraits en cas de test d'interopérabilité exécutables.

Contexte du test d'interopérabilité multi-implémentations

Dans cette thèse, nous nous sommes également intéressés au contexte du test d'interopérabilité de N ($N > 2$) implémentations, également appelé contexte multi-implémentations. Deux points ont été plus particulièrement étudiés dans ce chapitre : l'influence de la configuration des testeurs (architecture de test) et des connexions entre implémentations (topologie) sur le test d'interopérabilité de N systèmes, et la définition formelle de l'interopérabilité de N implémentations.

Tout d'abord, nous avons réalisé une classification des architectures de test d'interopérabilité utilisables dans le contexte du test d'interopérabilité multi-implémentations. Comme dans le cas du contexte one-to-one, les architectures de test d'interopérabilité multi-implémentations sont classifiées en fonction de l'accessibilité des interfaces des implémentations, mais également en fonction du mode d'observation des N implémentations durant leur interaction.

Ensuite, nous nous sommes intéressés au lien entre le choix d'une topologie interconnectant les N implémentations et le test d'interopérabilité de ces N implémentations. Plus particulièrement, nous avons défini les paramètres à prendre en compte pour choisir la topologie à utiliser et avons comparé ces paramètres avec les pratiques actuelles du test d'interopérabilité de N implémentations. Nous avons également défini une méthode de modélisation de la topologie pour les définitions formelles de la notion d'interopérabilité de N implémentations et décrit quelques topologies de base avec ce modèle. Cette modélisation des topologies possibles et les liens entre les modèles des différentes topologies permettent d'aider au choix d'une topologie pour le test d'interopérabilité, mais également à organiser une campagne de tests d'interopérabilité.

Nous avons ensuite proposé des définitions formelles de la notion d'interopérabilité adaptée aux architectures de test d'interopérabilité du contexte multi-implémentations. Comme dans le contexte one-to-one, nous avons appelé ces définition des *critères d'interopérabilité*. Nous avons appliqué ces critères d'interopérabilité multi-implémentations sur des exemples de protocoles et avons comparé leur pouvoir de détection de la non-interopérabilité. les résultats de cette comparaison suggèrent la possibilité d'utiliser une approche similaire à celle du contexte one-to-one.

Enfin, nous avons donné des pistes à étudier pour développer une approche de génération de cas de test d'interopérabilité multi-implémentations. Le problème est ici de déterminer dans quelle mesure les dépendances entre évènements exécutables par plusieurs implémentations peuvent être utilisées pour générer des cas de test d'interopérabilité sans se baser sur une vue globale du SUT composé des N implémentations. Une approche multi-unilatérale (approche similaire à l'approche bilatérale du

contexte one-to-one) comporte une étape de dérivation d'objectifs de test unilatéraux à partir d'un objectif de test global. Or, une telle étape nécessite des calculs de dépendances multiples entre évènements très complexes. Cependant, une fois la dérivation des objectifs de test unilatéraux réalisée, une telle approche permettrait une approche distribuée sans synchronisation du test d'interopérabilité multi-implémentations.

Glossaire

$=_{\triangle}$: "par définition"
API : Application Programming Interface
ATS : Abstract Test Suite (suite de tests abstraits)
BCG : Binary Coded Graphs (format et API associée pour représenter des IOLTS dans la boîte à outils Cadp)
CADP : Construction and Analysis of Distributed Processes (ensemble d'outils logiciels de construction et d'analyse de processus distribués)
CEFSM : Communicating Extended Finite State Machine (machine communicante étendue à états finis)
CTG : Complete Test Graph (graphe de test complet)
EFSM : Extended Finite State Machine (machine étendues à états finis)
ETS : Executable Test Suite (suite de tests exécutables)
ETSI : European Telecommunications Standards Institute
FSM : Finite State Machine (machine à états finis)
IAP : Interface Access Point
IEEE : Institute of Electrical and Electronics Engineers
IETF : Internet Engineering Task Force
IOLTS : Input-Output Labelled Transition System (Systèmes de transitions étiquetées à entrées et sorties)
iop : interopérabilité
IOSM : Input-Output State Machine (autre appellation pour IOLTS)
ISO : International Organization for Standardization (Organisation internationale de normalisation)
ETSI : European Telecommunications standards Institute
ITU : International Telecommunication Union
IUT : Implementation Under Test (Implémentation sous test)
LI : Lower Interface
FDT : Formal Description technique
LTS : Labelled Transition System (Systèmes de transitions étiquetées)
M-iop : Multi-implementations interoperability
MSC : Message Sequence Charts

PCO : Point of Control and Observation (point de contrôle et d'observation)
PO : Point of Observation (point d'observation)
RI : Refence Implementations (implémentation de référence, dont le comportement est connu)
SDL : Specification and Description Language
SUT : System Under Test (Système sous test)
TC : Test Case (cas de test)
TC : Test Component (composant de test, partie du Test System)
TCP : Test Coordination Procedure
TS : Test System
TGV : Test Generation from transitions systems using Verification techniques (outil de génération de test de conformité)
TP : Test Purpose (objectif de test)
UI : Upper Interface

Références de l'auteur

Alexandra DESMOULIN, César VIHO. **Quiescence Management improves Interoperability Testing**. *17th IFIP International Conference on Testing of Communicating Systems (Testcom 2005)*. Montréal, Canada, 2005.

Alexandra DESMOULIN, César VIHO. **Formalizing Interoperability for Test Case Generation Purpose**. *IEEE ISoLA Workshop on Leveraging Applications of Formal Methods, Verification, and Validation*. Columbia, MD, Etats-Unis, 2005.

Alexandra DESMOULIN, César VIHO. **Formalizing Interoperability testing : Quiescence Management and Test Generation**. *25th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2005)*. Taipei, Taiwan, octobre 2005.

Ariel SABIGUERO, Anthony BAIRE, Alexandra DESMOULIN, Annie FLOCH, Frédéric ROUDAUT, César VIHO. **Towards an IP-oriented testing framework - The IPv6 Testing Toolkit**. *TTCN-3 User Conference 2006*. Berlin, Germany, Mai 2006.

Alexandra DESMOULIN, César VIHO. **Interoperability test generation : formal definitions and algorithm**. *Huitième Colloque Africain sur la Recherche en Informatique (CARI 2006)*. Cotonou, Bénin, Novembre 2006.

Alexandra DESMOULIN, César VIHO. **Interoperability test generation : formal definitions and algorithm**. *Edition spéciale CARI'06 de la revue ARIMA*. 2007.

Alexandra DESMOULIN, César VIHO. **A New Method for Interoperability Test generation**. *19th IFIP International Conference on Testing of Communicating Systems and 7th International Workshop on Formal Approaches to Testing of Software(Testcom/Fates 2007)*. Talinn, Estonie, Juin 2007.

Alexandra DESMOULIN, César VIHO. **Automatic Interoperability Test Case Generation based on Formal Definitions**. *Proceedings of 12th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2007)*. Berlin, Allemagne, Juillet 2007.

Bibliographie

[APRS93] Noriyasu Arakawa, Marc Phalippou, Nathalie Risser, and Terunao Soneoka. Combination of conformance and interoperability testing. In *FORTE '92 : Proceedings of the IFIP TC6/WG6.1 Fifth International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols*, pages 397–412, Amsterdam, The Netherlands, 1993. North-Holland Publishing Co.

[BBB+96] 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.

[BCKZ02] Cédric Besse, Ana R. Cavalli, Myungchul Kim, and Fatiha Zaïdi. Automated generation of interoperability tests. In *TestCom '02 : Proceedings of the IFIP 14th International Conference on Testing Communicating Systems XIV*, page 169, Deventer, The Netherlands, The Netherlands, 2002. Kluwer, B.V.

[BFS05] A. Belinfante, L. Frantzen, and C. Schallhart. Tools for test case generation. In M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors, *Model-Based Testing of Reactive Systems*, number 3472 in Lecture Notes in Computer Science No 3472, chapter 14. Springer Verlag, 2005.

[BLPZ98] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella. Diagnosis of active systems. In *Proceedings of the European Conference on artificial Intelligence (ECAI)*, pages 274–278, 1998.

[BP94] Gregor V. Bochmann and Alexandre Petrenko. Protocol testing : review of methods and relevance for software testing. In *ISSTA '94 : Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis*, pages 109–124, New York, NY, USA, 1994. ACM Press.

[BPGQ02] Sergiy Boroday, Alexandre Petrenko, Roland Groz, and Yves-Marie Quemener. Test generation for cefsm combining specification and fault coverage. In Ina Schieferdecker, Hartmut König, and Adam Wolisz, editors, *TestCom*, volume 210 of *IFIP Conference Proceedings*, pages 355–372. Kluwer, 2002.

[BT00] Ed Brinksma and Jan Tretmans. Testing transition systems : An annotated bibliography. In Franck Cassez, Claude Jard, Brigitte Rozoy, and Mark Dermot Ryan, editors, *Modeling and Verification of Parallel Processes, 4th Summer School, MOVEP 2000, Nantes, France, June 19-23, 2000*, volume 2067 of *Lecture Notes in Computer Science*, pages 196–205. Springer, 2000.

[CK94] Richard Castanet and Osmane Koné. Deriving coordinated testers for interoperability. In *Proceedings of the IFIP TC6/WG6.1 Sixth International Workshop on Protocol Test systems VI*, pages 331–346, Amsterdam, The Netherlands, 1994. North-Holland Publishing Co.

[CL98] M.-O. Cordier and C. Largouet. Diagnosing discrete-event systems : an experiment in telecommunication networks. In *4th International Workshop on Discret Event Systems*, pages 130–137, 1998.

[CLRZ99] Ana R. Cavalli, David Lee, Christian Rinderknecht, and Fatiha Zaïdi. Hit-or-jump : An algorithm for embedded testing with applications to in services. In *FORTE XII / PSTV XIX '99 : Proceedings of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX)*, pages 41–56, Deventer, The Netherlands, 1999. Kluwer, B.V.

[Con] Connectathon. `http://www.connectathon.org`.

[DK03] Sarolta Dibuz and Péter Krémer. Framework and model for automated interoperability test and its application to rohc. In Dieter Hogrefe and Anthony Wiles, editors, *TestCom 2003*, volume 2644 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 2003.

[DV05] Alexandra Desmoulin and César Viho. Quiescence management improves interoperability testing. In Ferhat Khendek and Rachida Dssouli, editors, *TestCom*, volume 3502 of *Lecture Notes in Computer Science*, pages 365–379. Springer, 2005.

[DV07a] A. Desmoulin and C. Viho. Automatic interoperability test case generation based on formal definitions. In *12th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, Berlin, Germany, July 2007.

[DV07b] A. Desmoulin and C. Viho. A new method for interoperability test generation. In *19th IFIP International Conference on Testing of Communicating Systems (Testcom) and 7th International Workshop on Formal Approaches to Testing of Software (Fates)*, Tallinn, Estonia, June 2007.

[EFTSY04] Khaled El-Fakih, Vadim Trenkaev, Natalia Spitsyna, and Nina Yevtushenko. Fsm based interoperability testing methods for multi stimuli

model. In Roland Groz and Robert M. Hierons, editors, *TestCom*, volume 2978 of *Lecture Notes in Computer Science*, pages 60–75. Springer, 2004.

[ETS] ETSI. Etsi's plugtests. `http://www.etsi.org/plugtests/`.

[Fer89] Jean-Claude Fernandez. Aldébaran : A tool for verification of communicating processes. Technical Report Spectre C14, LGJ-IMAG Grenoble, 1989.

[FJJV97] Jean-Claude Fernandez, Claude Jard, Thierry Jéron, and César Viho. An experiment in automatic generation of test suites for protocols with verification technology. *Science of Computer Programming*, 29(1-2) :123–146, 1997.

[FS03] Maximilian Frey and Bernd-Holger Schlingloff. Conformance of distributed systems. In Dieter Hogrefe and Anthony Wiles, editors, *Testing of Communicating Systems, 15th IFIP International Conference, TestCom 2003, Sophia Antipolis, France, May 26-28, 2003, Proceedings*, volume 2644 of *Lecture Notes in Computer Science*, pages 163–179. Springer, 2003.

[FTW04] Lars Frantzen, Jan Tretmans, and Tim A. C. Willemse. Test generation based on symbolic specifications. In Jens Grabowski and Brian Nielsen, editors, *FATES*, volume 3395 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2004.

[GHN93] Jens Grabowski, Dieter Hogrefe, and Robert Nahm. Test Case Generation with Test Purpose Specification by MSCs. In *In : SDL'93 - Using Objects (Editors : O. Faergemand, A. Sarma), North-Holland, October 1993*, October 1993.

[GLM01] H. Garavel, F. Lang, and R. Mateescu. An overview of cadp 2001. Technical Report 0254, INRIA, 2001.

[GR98] Roland Groz and Nathalie Risser. Eight years of experience in test generation from fdts using tveda. In *FORTE X / PSTV XVII '97 : Proceedings of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE X) and Protocol Specification, Testing and Verification (PSTV XVII)*, pages 465–480, Osaka, Japan, 1998. Chapman & Hall, Ltd.

[GRSS90] Jay Gadre, Chris Rohrer, Catherine Summers, and Susan Symington. A cos study of osi interoperability. *Comput. Stand. Interfaces*, 9(3) :217–237, 1990.

[Hee98] L. Heerink. *Ins and outs in refusal testing*. PhD thesis, University of Twente, Institute for programming Research and Algorithmics, May 1998.

[HLSG04] Ruibing Hao, David Lee, Rakesh K. Sinha, and Nancy Griffeth. Integrated system interoperability testing with applications to voip. *IEEE/ACM Trans. Netw.*, 12(5) :823–836, 2004.

[IRI] IRISA. Tipi activities - interoperability events. `http://www.irisa.fr/tipi/activity_en.htm`.

[ISO89] ISO. Information processing systems – Open Systems Interconnection – Estelle – A formal description technique (FDT) based on extended state transition model. Technical Report IS9074, ISO, 1989.

[ISO94] ISO. Information Technology - Open Systems Interconnection Conformance Testing Methodology and Framework - Parts 1-7. *International Standard ISO/IEC 9646/1-7*, 1994.

[IT02] ITU-T. Specification and Description Language (SDL). Recommendation Z.100, ITU-T, August 2002.

[JCTG96] J. C. Fernandez, C. Jard, T. Jeron, and G. Viho. Using on-the-fly verification techniques for the generation of test suites. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102, pages 348–359, New Brunswick, NJ, USA, / 1996. Springer Verlag.

[JJ05] Claude Jard and Thierry Jéron. Tgv : theory, principles and algorithms : A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. *Int. J. Softw. Tools Technol. Transf.*, 7(4) :297–315, 2005.

[JJTV99] Claude Jard, Thierry Jéron, Lénaick Tanguy, and César Viho. Remote testing can be as powerful as local testing. In Jianping Wu, Samuel T. Chanson, and Qiang Gao, editors, *FORTE*, volume 156 of *IFIP Conference Proceedings*, pages 25–40. Kluwer, 1999.

[Jér01] Thierry Jéron. Le test de conformité : état de l'art. Rapport pour l'AEE (Architecture Electronique Embarquée), 2001.

[Kan98] S. Kang. Relating interoperability testing with conformance testing. In *Global Telecommunications Conference. GLOBECOM 98*, Sydney, NSW, Australia, 1998.

[KC00] Ousmane Koné and Richard Castanet. Test generation for interworking systems. *Computer Communications*, 23(7) :642–652, 2000.

[KK96] Myungchul Kim and Gyuhyeong Kim. Interoperability testing methodology and guidelines. In *Digital Audio-Visual Council, system integration TC*, volume DAVIC/TC/SYS/96/06/006, New York, 1996.

[KOS+00] Toshihiko Kato, Tomohiko Ogishi, Hiroyuki Shinbo, Yutaka Miyake, Akira Idoue, and Kenji Suzuki. Interoperability testing system of tcp/ip based systems in operational environment. In Hasan Ural, Robert L. Probert, and Gregor von Bochmann, editors, *TestCom*, volume 176 of *IFIP Conference Proceedings*, pages 143–. Kluwer, 2000.

[KSM96] James D. Kindrick, John A. Sauter, and Robert S. Matthews. Improving conformance and interoperability testing. *StandardView*, 4(1) :61–68, 1996.

[LY96] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - A survey. In *Proceedings of the IEEE*, volume 84, pages 1090–1126, 1996.

[Mor00] Pierre Morel. *Une algorithmique efficace pour la génération automatique de test de conformité*. PhD thesis, Université de Rennes I, 2 2000.

[MRW04] Scott Moseley, Steve Randall, and Anthony Wiles. In pursuit of interoperability. *Int. J. IT Standards and Standardization Res.*, 2(2) :34–48, 2004.

[Pet00] Alexandre Petrenko. Fault model-driven test derivation from finite state models : Annotated bibliography. In Franck Cassez, Claude Jard, Brigitte Rozoy, and Mark Dermot Ryan, editors, *Modeling and Verification of Parallel Processes, 4th Summer School, MOVEP 2000, Nantes, France, June 19-23, 2000*, volume 2067 of *Lecture Notes in Computer Science*, pages 196–205. Springer, 2000.

[Pha94] Marc Phalippou. *Relations d'implantation et hypothèses de test sur des automates à entrées et sorties*. PhD thesis, Université de Bordeaux I, 9 1994.

[PYvBD96] Alexandre Petrenko, Nina Yevtushenko, Gregor von Bochmann, and Rachida Dssouli. Testing in context : framework and test derivation. *Computer Communications*, 19(14) :1236–1249, 1996.

[Rep94] ETSI Technical Report. Methods for testing and specification (mts) ; interoperability and conformance testing a classification scheme. Technical Report ETR 130, European Telecommunications Standardisation Institute, 1994.

[SKKC04] S. Seol, M. Kim, S. Kang, and S. T. Chanson. Interoperability test generation and minimization for communication protocols based on the multiple stimuli principle. *IEEE Journal on selected areas in Communications*, 22(10) :2062– 2074, december 2004.

[SKKR03] S. Seol, M. Kim, S. Kang, and J. Ryu. Fully automated interoperability test suite derivation for communication protocols. *Comput. Networks*, 43(6) :735–759, 2003.

[TB02] J. Tretmans and E. Brinksma. Côte de Resyste – Automated Model Based Testing. In M. Schweizer, editor, *Progress 2002 – 3^{rd} Workshop on Embedded Systems*, pages 246–255, Utrecht, The Netherlands, October 24 2002. STW Technology Foundation.

[TB03] J. Tretmans and E. Brinksma. Torx : Automated model based testing. In A. Hartman and K. Dussa-Zieger, editors, *Proceedings of the First European Conference on Model-Driven Software Engineering*, Nurnberg, Germany, December 2003.

[Tel] Telelogic/Verilog. ObjectGeode 4.2. http ://www.telelogic.com.

[TKS03] Vadim Trenkaev, Myungchul Kim, and Soonuk Seol. Interoperability testing based on a fault model for a system of communicating fsms. In Dieter Hogrefe and Anthony Wiles, editors, *Testing of Communicating Systems, 15th IFIP International Conference, TestCom 2003, Sophia Antipolis, France, May 26-28, 2003, Proceedings*, volume 2644 of *Lecture Notes in Computer Science*, pages 226–242. Springer, 2003.

[Tör99] Maria Törö. Decision on tester configuration for multiparty testing. In Gyula Csopaki, Sarolta Dibuz, and Katalin Tarnay, editors, *Testing of Communicating Systems : Method and Applications, IFIP TC6 12^{th} International Workshop on Testing Communicating Systems, September 1-3, 1999, Budapest, Hungary*, volume 147 of *IFIP Conference Proceedings*, pages 109–130. Kluwer, 1999.

[Tre92] J. Tretmans. *A Formal Approach to Conformance Testing*. PhD thesis, University of Twente, Enschede, The Netherlands, 1992.

[Tre96] Jan Tretmans. Conformance testing with labelled transition systems : implementation relations and test generation. *Comput. Netw. ISDN Syst.*, 29(1) :49–79, 1996.

[Tre99] Jan 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, London, UK, 1999. Springer-Verlag.

[UK99] Andreas Ulrich and Hartmut König. Architectures for testing distributed systems. In Gyula Csopaki, Sarolta Dibuz, and Katalin Tarnay, editors, *Testing of Communicating Systems : Method and Applications, IFIP TC6 12^{th} International Workshop on Testing Communicating Systems, September 1-3, 1999, Budapest, Hungary*, volume 147 of *IFIP Conference Proceedings*, pages 93–108. Kluwer, 1999.

[VB94] Gert Vermeer and Hans Blik. Interoperability testing : Basis for the acceptance of communication systems. In *Proceedings of the IFIP*

TC6/WG6.1 Sixth International Workshop on Protocol Test systems VI, pages 315–330, Amsterdam, The Netherlands, 1994. North-Holland Publishing Co.

[VBT01] César Viho, Sébastien Barbin, and Lénaick Tanguy. Towards a formal framework for interoperability testing. In Myungchul Kim, Byoungmoon Chin, Sungwon Kang, and Danhyung Lee, editors, *FORTE*, volume 197 of *IFIP Conference Proceedings*, pages 53–68. Kluwer, 2001.

[vdBRT04] 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.

[VTKB92] Louis Verhaard, Jan Tretmans, Pim Kars, and Ed Brinksma. On asynchronous testing. In Gregor von Bochmann, Rachida Dssouli, and Anindya Das, editors, *Protocol Test Systems*, volume C-11 of *IFIP Transactions*, pages 55–66. North-Holland, 1992.

[WP94] 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.

[WSG98] Thomas Walter, Ina Schieferdecker, and Jens Grabowski. Test architectures for distributed systems : State of the art and beyond. In Alexandre Petrenko and Nina Yevtushenko, editors, *IWTCS*, volume 131 of *IFIP Conference Proceedings*, pages 149–174. Kluwer, 1998.

[WWY04] Zhiliang Wang, Jianping Wu, and Xia Yin. Towards interoperability test generation of time dependant protocols : a case study. In *Global Telecommunications Conference, GLOBECOM '04, Dallas, Texas, Etats-Unis*, volume 2, pages 589–594, 2004.

Table des figures

Résumé

Cette thèse s'intéresse au domaine du test d'interopérabilité de protocoles. Il s'agit de vérifier que plusieurs composants de réseaux sont capables de communiquer correctement tout en fournissant le service prévu dans leur spécification.
Nous proposons une définition formelle de la notion d'interopérabilité via des critères d'interopérabilité qui précisent les conditions dans lesquelles des composants peuvent être considérés interopérables.
Pour le contexte le plus utilisé du test d'interopérabilité de deux composants, nous proposons une méthode de génération automatique de tests. Cette méthode évite le problème d'explosion combinatoire généralement rencontré lors de la génération de tests d'interopérabilité.
Un des problèmes spécifiques au domaine du test d'interopérabilité est celui de la vérification du traitement effectif par un composant d'un message qui lui a été envoyé. Nous proposons une solution basée sur les dépendances causales entre messages.
Pour le contexte du test d'interopérabilité de multiple (plus de deux) composants, nous proposons une classification des architectures de test possibles, des définitions de critères d'interopérabilité, une méthodologie pour le choix des topologies d'interconnexion à utiliser et quelques éléments pour la dérivation des tests.

Mots-clefs : test, protocoles, critères d'interopérabilité, génération de tests, approche formelle

Abstract

This thesis deals with the domain of protocol interoperability testing. The objective is to verify that different network components are able to communicate correctly while providing the service described in their respective specification. We propose a formal definition of the interoperability notion using interoperability criteria that precise conditions to be verified by components to be considered interoperable.
We propose an automatic test generation method for the most used context of two components interoperability testing. This method avoids the state space explosion problem that interoperability test generation generally faces.
A specific problem of the interoperability testing domain is the verification of the actual reception of a message that was sent to a component. We propose a solution based on causal dependencies between messages.
For the interoperability of multiple components (more than two), we propose a classification of the possible testing architectures, definitions of interoperability criteria, a methodology for the choice of interconnection topologies to be used and some elements for the test derivation.

Keywords : test, protocol, interoperability criteria, test generation, formal approach