Spécification Formelle de Systèmes de Test

Ousmane KONÉ
LaBRI, Université Bordeaux I
351, Cours de La Libération, 33405 Talence-FRANCE
e-mail : kone@labri.u-bordeaux.fr

Résumé. Les normes ISO9646-4, ISO9646-5 recommandent aux laboratoires de test de décrire les Moyens de Réalisation effective des Tests. Cet article présente une approche pour répondre à ces recommandations, en utilisant les TDF (Techniques de Description Formelle). Le langage de test TTCN, proposé par l'ISO permet de spécifier des Suites de Test Abstraites, mais TTCN ne permet pas de décrire des architectures de test. Pour cela, la TDF Estelle est proposée dans cet article.
Nous avons proposé, dans des travaux antérieurs, un Système de Testeurs Concurrents, utilisable pour le Test d'Interopérabilité. Dans cet article, ce système sert comme illustration et est formellement décrit en Estelle. Un Système de Test étant défini, il est nécessaire de le valider en montrant son pouvoir de test. Pour cela, nous utilisons l'outil de validation VEDA, qui intègre un simulateur Estelle.

1. Introduction

La norme ISO9646 est destinée au processus de normalisation du test ISO (Interconnection de Systèmes Ouverts). Elle définit entre autres les architectures de test, des recommandations pour la spécification de suites de test abstraites, des recommandations pour les moyens de réalisation effective de test. Un nombre important de travaux ont été entrepris dans ces différents domaines du test ISO : génération automatique de suites de test abstraites en TTCN[5, 10], réalisation d'architectures de test[29, 30]. Mais l'étude faite dans [8] permet d'identifier des problèmes essentiels qui restent non résolus dans le processus de dévéloppement (et en particulier de test) de protocoles. Une des conclusions que l'on peut retenir de [8, 13] est la reconnaissance par l'ISO et le CCITT du rôle important que jouent les méthodes formelles dans le test. Ce besoin de formalisation, comme exprimé dans [13] concerne entre autres les préoccupations suivantes :

1. l'établissement d'une théorie et d'un cadre de travail permettant de formaliser la conformité et le test d'implantations spécifiées dans des techniques de description formelle (e.g. Estelle, Lotos, SDL)

2. la description formelle d'entités intervenant dans un test (e.g. testeur inférieur, testeur supérieur).

Des dévéloppements ont déja eu lieu en vue de satisfaire la préoccupation 1[3, 11] et nous avons proposé, dans un cadre théorique, une approche pour générer des tests pour des systèmes concurrents, ce qui s'applique aux systèmes OSI devant interopérer[6, 24].

Rappel de définition : Le *test d'interopérabilité*, pour un protocole de niveau N donné, consiste à vérifier que le comportement global d' implantations de niveau N rend bien le service N correspondant, alors que le comportement de chacune des implantations est conforme au protocole N. Pendant le colloque CARI'92 [23], nous avons montré que les méthodes existantes de test d'interopérabilité étaient confrontées au problème d'explosion combinatoire du graphe décrivant la communication globale des entités à tester. Puis nous avons proposé dans [6] un système de test (testeur canonique d'interopérabilité) permettant de s'affranchir de ce problème. Par contre, les Moyens de Réalisation (la mise en oeuvre) de ce testeur n'avaient pas été proposés. Il est donc important de fournir une description plutôt opérationnelle de ce testeur, plus "proche" d'un fonctionnement réel[1].

Dans cet article donc, nous nous intéressons à la préoccupation 2 (précédente), à savoir comment spécifier de manière formelle des architectures de test et leurs réalisations, afin de répondre aux normes ISO9646-4[18], ISO9646-5[19] qui recommandent aux centres de test la description des Moyens de Réalisation de Test (MRT). Plusieurs écrits concernant les systèmes de test distribués - aussi bien pour le test de conformité que pour le test d'interopérabilité - décrivent plûtot l'architecture générale du système. Par exemple, dans [27] ou dans [2], seul le rôle des différents composants du système de test est expliqué. Pourtant, il est intéressant de savoir comment (et surtout est-il possible, de manière automatique, de) générer ces composants. Seul la connaissance de ces composants ou en tout cas la description explicite des interactions qu'ils ont entre eux permet d'établir ce qui est effectivement testé par un système, et comment il s'y prend.

Le langage TTCN [17] permet de décrire des suites de test abstraites; mais une description en TTCN est indépendante de toute réalisation partic-

[1] Cela sera fait dans cet article.

ulière de test. Comme indiqué dans [1], les TDF (Techniques de Description Formelles) sont de bonnes candidates pour décrire les tests. Nous proposons de faire la description de systèmes de test en Estelle [20](un exemple de description formelle est fourni en annexe), ce langage offrant la possibilité de spécifier différentes configurations de systèmes (distribués). Décrire les éléments d'une architecture de test en Estelle n'est pas une idée nouvelle puisque dans [9], la faisabilité de telles descriptions était déjà évoquée. Cette pratique offre les avantages suivants :

- une description formelle est par principe concise et non ambigüe. C'est du reste ce qui justifie la préoccupation 2 précédemment évoquée ;
- le langage Estelle étant un moyen d'aide à la conception de systèmes distribués (VEDA [12],ESTIM [31]), les architectures de test peuvent alors être conçues et validées par simulation [21] ;
- il est envisageable de faire de la génération automatique de systèmes de test réels. Par exemple, l'outil PET-DINGO du NIST [32] permet de faire de la génération d'implantations distribuées à partir de spécifications Estelle.

Le simulateur Estelle, VEDA [22, 12] sera utilisé parce qu'il supporte la description en Estelle(*) [7] de la communication synchrone entre IOSM (automates à entrée-sortie que nous utilisons comme formalisme sous-jacent, voir section 2). Cependant, on pourra remarquer que Estelle(*) ne permet pas de modéliser la synchronisation multiple que requiert le système de test modélisé dans cet article. Mais nous verrons que l'introduction d'un module Estelle supplémentaire permet de pallier à ce problème.
La section 2 fait un rappel du fonctionnement général du Système de Testeurs Concurrents (STC) proposé dans [6]. La section 3 présente l'approche utilisée pour modéliser ce fonctionnement avec la TDF Estelle. Dans la section 4, une méthode pour automatiser la génération du STC, de même que l'expérimentation de ce dernier sous VEDA sont présentées. Enfin, un exemple de spécification du STC est fourni en annexe. Cet exemple est une description opérationnelle du Système de Testeurs Concurrents présenté dans la section 2.

2. Système de Testeurs Concurrents

Cette section a pour but de rappeler le fonctionnement du Système de Testeurs Concurrents initialement proposé dans [6], et qui sert d'illustration

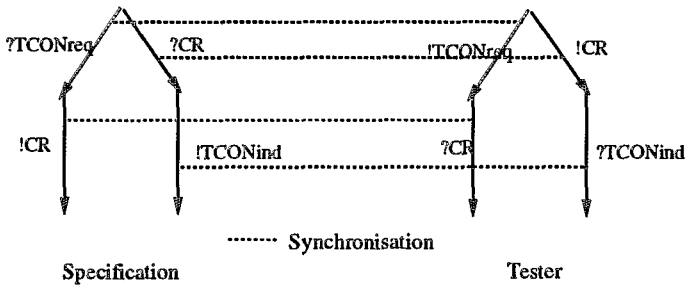

Figure 1. Exemple de spécification $\mathcal{S}$ et son testeur

dans cet article.
Les entités que nous manipulons (spécifications, implantations, testeurs) sont modélisées par des automates à entrée-sortie appelés IOSM (Input Output State Machine). La communication (qui est synchrone dans cet article) entre deux IOSM M_1, M_2 est modélisée par une IOSM notée $M_1 \parallel M_2$ et qui est le résultat de la composition parallèle (en d'autres termes la communication globale) entre les deux IOSM. Des exemples de communication globale seront fournis par la figure 2.
Testeur associé à une spécification : Pour une spécification $\mathcal{S}$ donnée, le testeur $T(\mathcal{S})$ (ou $\mathcal{T}$) correspondant doit être apte à détecter toute exécution non autorisée par cette spécification. Pour le test de conformité, un tel testeur peut être calculé suivant la démarche qui suit :

1. Afin de prendre en compte les comportements non déterministes de la spécification $\mathcal{S}$, celle-ci est d'abord transformée en son équivalent déterministe, soit $\mathcal{S}'$.

2. Pour obtenir le testeur, les transitions de $\mathcal{S}'$ sont ensuite inversées (puisqu'une émission de l'entité de protocole devient une réception du testeur, et vice versa).

La figure 1 fournit un exemple simple.
Dans cet exemple, aprés avoir émis TCONreq, la réception de toute interaction différente de CR doit conduire le testeur à un état d'erreur noté *fail*. Pour une question de lisibilité, les transitions conduisant à l'état d'erreur ne sont pas représentées sur les figures.

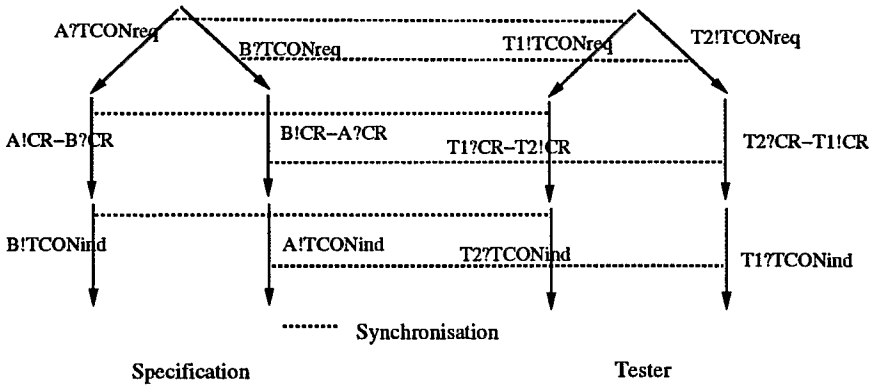

Figure 2. Extrait du comportement global

Etant données deux implantations $\mathcal{I}_1$, $\mathcal{I}_2$ dont les spécifications respectives sont $\mathcal{S}_1$, $\mathcal{S}_2$, le test d'interopérabilité de ces implantations (voir définition en introduction) consiste à vérifier que (a) $\mathcal{I}_1 \parallel \mathcal{I}_2$ se comporte globalement comme $\mathcal{S}_1 \parallel \mathcal{S}_2$, et que (b) $\mathcal{I}_1$ (resp. $\mathcal{I}_2$) se comporte individuellement comme $\mathcal{S}_1$ (resp. $\mathcal{S}_2$).
Supposons que les entités de protocole soient décrites par la spécification $\mathcal{S}$ de la figure 1. On a donc $\mathcal{S}_1 = \mathcal{S}_2 = \mathcal{S}$. Par conséquent, on a également $T(\mathcal{S}_1) = T(\mathcal{S}_2) = T(\mathcal{S})$.

D'après la figure 2, on remarque que la synchronisation de $T(\mathcal{S}_1)$ et $T(\mathcal{S}_2)$, soit $T(\mathcal{S}_1) \parallel T(\mathcal{S}_2)$, est bien un testeur pour le comportement global $\mathcal{S}_1 \parallel \mathcal{S}_2$ (Dans cette figure, les interactions de l'entité de protocole 1 (resp. 2) sont préfixées de **A** (resp. **B**) et celles du Testeur 1 (resp. 2) correspondant sont préfixées de **T1** (resp. **T2**) ; $?a{-}!a$ (*ou* $!a{-}?a$) désigne une synchronisation des entités communicantes sur l'interaction a.).
On remarque que l'interaction utilisée par les testeurs pour se synchroniser (en l'occurrence CR) doit être la même que celle utilisée par les entités de protocole (ceci constitue une forme de synchronisation multiple entre les testeurs et les entités de protocole). Pour cela, (i) les testeurs observent l'interaction échangée entre les entités de protocole afin de se synchroniser sur cette interaction ; (ii) le fait d'accéder à cette interaction permet à chaque testeur d'avoir une observation totale de chaque entité (voir figure

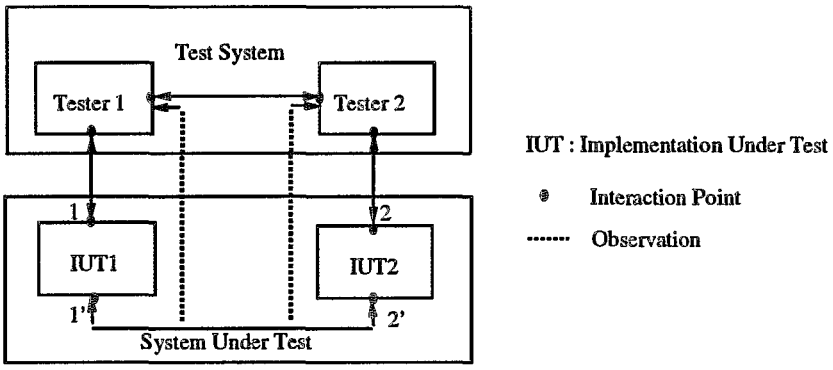

Figure 3. Système de test concurrent

3), et par conséquent de vérifier que cette dernière n'émet pas d'interaction non spécifiée.
L'approche de description proposée par cet article sera illustrée par une description opérationnelle et formelle du Système de Testeurs Concurrents précédent.

3. Technique de modélisation

3.1. Communication Synchrone (ou par Rendez-Vous)

Dans le système concurrent sous test, présenté dans [6] (voir figure 3), toutes les communications sont synchrones (par Rendez-Vous). La version normalisée d'Estelle [20] n'autorise que des communications par files FIFO entre modules. Une sémantique de communication synchrone a donc été proposée[2] pour Estelle (Estelle* [7]) et implantée par les simulateurs ESTIM [31] et VEDA [33]. Deux clauses supplémentaires sont introduites:

1. `IP!interaction` exprime une demande de synchronisation en émission au point d'interaction `IP`,

2. `IP?interaction` exprime une demande de synchronisation en réception au point d'interaction `IP`.

[2]Pour l'instant, elle n'est pas encore admise en tant que norme internationale.

Soit IP le point d'interaction où une interaction donnée, soit b est échangée ; alors la transition (1, !b, 2) s'écrit :

```
FROM 1 to 2
      IP!b
      begin end;
```

Pour que deux entités puissent se synchroniser sur l'interaction b, il est nécessaire qu'il y ait une demande de synchronisation en émission et une demande de synchronisation en réception, ce qui correspond par exemple à la transition (3, ?b, 4) qui s'écrit :

```
FROM 3 to 4
      IP?b
      begin end;
```

Pour que la synchronisation puisse effectivement avoir lieu, il est nécessaire que :

- les points d'interaction IP où la synchronisation a lieu doivent être connectés,
- les deux transitions précédentes doivent être tirables.

Ce mécanisme de synchronisation correspond précisement à celui qui a été utilisé dans [6] pour le modèle d'automates à entrée sortie (les IOSM), et dans cet article, nous ne considérons que des communications synchrones.

3.2. Coordination et détection d'erreurs

Dans le STC, il existe deux mécanismes dont la mise en oeuvre nous paraît intéressante à décrire:

1. (mécanisme 1) le premier consiste en la coordination des testeurs à l'aide de l'interaction échangée entre les IST (Implantations Sous Test);
2. (mécanisme 2) le deuxième consiste en la détection des erreurs aussi bien à l'interface de service qu'au point d'interaction entre les IST.

Le mécanisme 2 (détection d'erreurs) ne pose pas de problème particulier, à condition que les testeurs accèdent, non seulement à l'interface de service (c'est trivialement le cas), mais aussi aux points d'interaction entre les IST. Cette condition étant réalisée, chaque testeur (de conformité) entre en état d'erreur si l'interaction en provenance de l'IST observée, est erronnée. Des observateurs associés aux modules "Testeur" sont conçus pour signaler que ces derniers sont entrés en état d'erreur, ce qui provoquera l'arrêt du test.

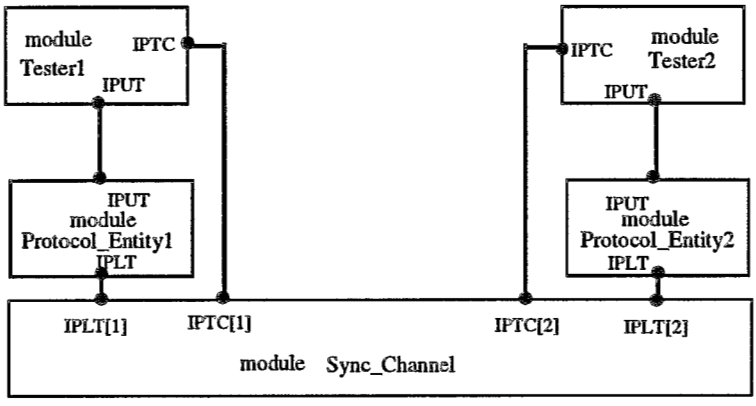

Figure 4. Modèle de simulation du STC

En ce qui concerne le mécanisme 1, rappelons que l'on utilise l'interaction échangée entre les IST pour coordonner les testeurs. Cette coordination doit se faire "en même temps" que l'interaction est échangée entre les IST (voir figure 2). Ceci est une forme de synchronisation multiple entre les IST et les Testeurs. Différentes solutions sont présentées dans [25] pour décrire ce mécanisme. Il est clair que Estelle ne permet pas d'exprimer la synchronisation multiple. Mais cela peut être simulé en introduisant un module intermédiaire, jouant un rôle de synchronisation (mécanisme 1), tout en permettant de réaliser le mécanisme 2. Ce module, Sync_Channel (Synchronous Channel[3], voir figure 4) aura globalement le comportement décrit par la figure 5. Ce module a un comportement "symétrique", c'est à dire qu'il a un fonctionnement similaire vis à vis des entités de protocole 1 et 2. La figure 5 décrit ce fonctionnement pour une interaction de synchronisation reçue de l'entité de protocole 1 :

1. Une (demande de synchronisation en) émission, soit **Pdu**, est reçue au point d'interaction IPLT[1], en provenance de l'entité de protocole 1

2. cette interaction est envoyée au Testeur 1 pour vérifier sa validité. Si l'interaction est erronnée, le Testeur devra entrer en état d'erreur

[3]Nous traduirons cela par *Canal Synchrone*, en français

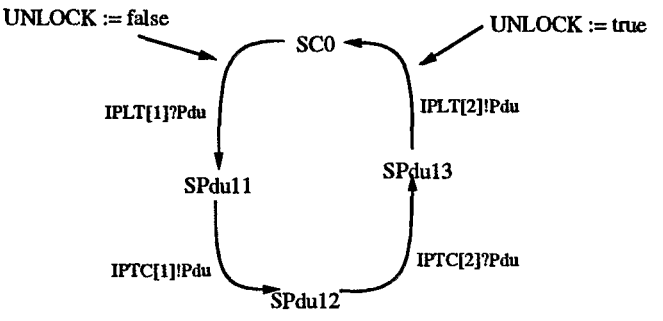

Figure 5. Fonctionnement du Canal Synchrone

(arrêt du test), sinon

3. l'interaction **Pdu** est indiquée au Testeur 2 comme interaction de coordination. Le Testeur 2 se "synchronise" (ou se coordonne) en émettant **Pdu**, puis

4. l'interaction est envoyée à l'entité de protocole 2, car elle lui est en fait destinée.

L'introduction de ce fonctionnement est une solution permettant de simuler le mécanisme 1. Si l'on fait de la simulation interactive, le fonctionnement du module Sync_Channel est satisfaisant puisque nous guidons, manuellement, le séquencement des interactions. Par contre, si nous faisons de la simulation automatique, nous sommes obligés d'automatiser le fait que le séquencement des interactions est soumis aux règles de communication sychrone : en l'occurrence, lorsqu'une entité émet une demande de synchronisation vers l'entité paire, le Rendez-Vous impose que cette entité se suspende jusqu'à ce que la demande de synchronisation soit acceptée par l'entité paire. Pendant le "cycle" du module Sync_Channel (figure 5) , il faut "verrouiller" (UNLOCK = true/false) les interfaces de communication de l'entité, de sorte qu'aucune autre de ses transitions ne soit tirable.

Note : observateurs actifs

La solution adoptée pour le "verrouillage" précédent se met en oeuvre en utilisant la variable que nous appelons **UNLOCK**. Cette solution est relativement triviale si tous les modules concernés (Canal Synchrone, Testeurs)

peuvent accéder à cette variable (rappelons que celle-ci est modifiée par le module Canal Synchrone, et lue par les modules Testeur). En Estelle, le "partage" des variables entre modules est possible uniquement entre modules père et fils. Un tel "lien de parenté" n'existe pas dans la configuration du STC où tous les modules sont des "activités indépendantes".
Par contre, dans l'environnement VEDA, les observateurs accèdent à toutes les variables. De plus, ces observateurs peuvent être actifs, c'est à dire qu'ils peuvent agir sur le système observé. Dans notre cas, l'action des observateurs se limitera à assurer que, à chaque instant (global), les testeurs connaissent la valeur courante de la variable UNLOCK. Pour cela, chaque Testeur possèdera une variable UNLOCK qui contiendra, grâce aux observateurs, la valeur de la variable UNLOCK du Canal Synchrone (voir la spécification des observateurs en annexe).

4. Génération et Expérimentation du STC

4.1. Génération semi-automatique du STC

La description du STC est constituée d'une partie fixe qui définit l'architecture et d'une partie variable qui définit le corps [4]des modules.

- La partie fixe comprend la définition des entêtes de modules et des canaux de communication. Ceux-ci sont à configurer (manuellement) en déclarant les interactions du protocole qui y sont échangées. De manière systématique, les canaux entre Testeurs et IST (points d'interaction IPUT) véhiculent les Primitives de Service, tandis que les autres canaux (points d'interaction IPLT, IPTC)véhiculent les Unités de Données de Protocole.
- La génération automatisée de cette partie variable peut se faire en suivant la démarche suivante:

1. **Testeurs :** calculer les testeurs en appliquant la méthode indiquée dans la section 2 (Calcul du testeur de conformité, voir [6]).
   Renommer chaque interaction du testeur en la préfixant de l'interface de test correspondante (IPUT,IPTC).
   Pour prendre en compte le "verrouillage" de l'IST, précédemment évoqué, on ajoute le traitement suivant : toute transition réalisant une interaction avec l'IST (de telles transitions correspondent à celles dans lesquelles figure le point d'interaction IPUT) doit contenir dans sa garde la clause **"PROVIDED (UNLOCK)"**, ce qui permet d'assurer que le testeur ne peut interagir avec l'IST que si cette

[4]Nous empruntons la terminologie Estelle, soit *body*. Le corps d'un module définit l'ensemble de ses transitions.

dernière n'est pas "en Rendez-Vous" avec l'IST paire.

2. Canal Synchrone : Pour toute interaction **Pdu** déclarée au point d'interaction IPLT, générer un ensemble (une séquence) de transitions tel que la figure 5 le décrit.

A titre d'illustration, la description du STC de la section 2, est présenté en annexe.

4.2. Expérimentation sous VEDA

L'expérimentation consiste en une simulation de l'exécution du STC. Cette simulation est réalisée dans un contexte de test par mutation. Le *test par mutation* [4, 14] consiste à tester des prototypes d'implantations dans lesquels l'on a volontairement introduit des erreurs, le but étant de savoir si le test considéré peut effectivement détecter ces erreurs. De telles implantations que l'on appelle *mutants*, sont généralement obtenues en introduisant une ou plusieurs erreurs dans une spécification de référence. Par exemple, toute interaction non autorisée dans un état donné de la spécification est dite inopportune : de telles erreurs peuvent être simulées en introduisant des interactions non autorisées, dans une spécification de référence (qui devient ainsi une IST non valide). Toute exécution exhaustive du STC est censée détecter les interactions inopportunes émises par les IST.
Remarque : Différents types d'erreur d'interopérabilité (donc de mutants) sont possibles. Le lecteur trouvera une classification de telles erreurs dans [26]. Nous illustrons dans ce qui suit, des cas de fonctionnement mis en oeuvre par simulation.
Pendant une simulation de test sous VEDA, deux types de terminaison peuvent se produire.

1. **Terminaison normale de l'exécution :**
   toutes les transitions du graphe de communication globale ont été exécutées (le parcours exhaustif a été entièrement réalisé). On revient à l'état initial, et la campagne de test s'arrête.

2. **Terminaison dûe à une situation de blocage[5] :**
   Les situations de blocage peuvent être dûes à la détection d'une erreur d'implantation, ou encore à un problème d'incompatibilité.

   • Erreur d'implantation : Lorsque le Système de Test Concurrent

[5] Après un blocage, la trace d'exécution mémorisée par le simulateur peut être reexécutée et analysée, ce qui permet de connaître l'origine du blocage.

détecte un comportement erroné, il conduit à un état d'erreur. Dans ce cas, l'observateur 1 (resp. observateur 2) associé au testeur 1 (resp. testeur 2) indique ce fait tout en se conduisant, lui aussi, à un état d'erreur (error state dans VEDA). Cela provoque un arrêt de la campagne de test. Rappelons que l'état d'erreur (noté état *fail*) d'un testeur est un "état puits" à partir duquel le testeur ne peut plus évoluer.

• Erreur d'incompatibilité : Considérons le cas de fonctionnement suivant, où les IST considérées sont conformes : l'IST1 reçoit une PDU de l'IST2. Cette dernière étant conforme, le Système de Test ne détecte pas d'erreur. Par contre, supposons que l'IST1 juge cette PDU inacceptable[6] (voire invalide). Elle peut alors réagir en émettant une interaction indiquant le rejet de la PDU. Elle peut également ne pas réagir, c'est à dire qu'elle ignore la PDU. Dans ce dernier cas, l'IST1 reste en attente de la PDU, tandis que l'IST2 la considère comme reçue. On se trouve dans ce cas dans une situation de blocage. Une telle situation est signalée par le simulateur ("**deadlock**").
Cette situation est simulable en utilisant des paramètres incompatibles entre eux. Mais remarquons qu'une situation similaire se produit lorsque l'on utilise des spécifications non valides[7]. Il est trivial que si des situations de blocage sont potentiellement présentes dans un protocole, des implantations conformes à ce dernier vont probablement les exhiber pendant l'interfonctionnement. Toutefois, d'après ce qui précède, l'expérience montre que si la spécification de protocole, et une étude de compatibilité entre les implantations, n'ont pas été validées au préalable, des problèmes d'interopérabilité peuvent survenir.

5. Conclusion

L'environnement que nous avons présenté dans cet article présente de l'intérêt pour la mise en oeuvre de test dans le contexte ISO. Les concepts essentiels que l'on peut retenir sont :

[6]Par exemple, dans le protocole Transport [15], les valeurs de champs d'adresse peuvent varier d'une implantation à l'autre ; il arrive donc que pour des implantations données, les champs d'adresse soient incompatibles entre eux.

[7]Ce cas arrive lorsque les spécifications considérées sont des cas d'école trop simplifiés, et non vérifiés.

- la méthode conduisant à la dérivation et la description formelle du STC ; ceci repond à la norme ISO9646-4[18] préconisant la description explicite des Moyens de Réalisation de Test. Une description formelle du STC, en Estelle, est fournie en annexe ;

- la simulation du test avec un outil de validation, en l'occurrence VEDA. L'approche de test par mutation est proposée pour montrer la validité du STC, c'est à dire son aptitude à détecter les éventuelles erreurs d'interopérabilité, ce qui illustre, entre autres, les résultats (théoriques) établis dans [24]. Une génération d'implantation réelle du STC est ensuite envisageable à l'aide d'outils tels que PET DINGO [32].

Ceci étant, les concepts précédents n'auront (davantage) de poids que s'ils intègrent les cas de communication asynchrone, les contraintes liées aux données, les contraintes liées au temps physique.

REFERENCES

1. G.v.Bochmann, R.Dssouli, B.Sarikaya. Architectures et sélection de test, dans Ingéniérie des protocoles. Ed. R.Castanet, O. Rafiq. Eyrolles 1988.
2. G.Bonnes. Services IBM d'interopétation OSI. dans Ingéniérie des Protocoles. Eyrolles 1988. R.Castanet, O.Rafiq (Ed)
3. E.Brinksma. A theory for the derivation of tests. Proc. 8th IFIP symposium on Protocol Specification, Testing and Verification. Atl. City . 1988.
4. T.A.Budd et al. Theoritical and empirical studies on using program mutation to test functional correctness of programs. 7th ACM Symphosium on principles of programming languages, Las Vegas, January 1980.
5. R.Castanet. Validation et Test de protocoles de communications. Rapport interne LaBRI 90-6 1990.
6. R.Castanet, O.Koné. Deriving Coordinated Testers for Interoperability. Proc. 6th International Workshop on Protocol Testing Systems, Pau, France, September 1993.
7. J.P. Courtiat. Introducing a Rendez-Vous Mechanism in Estelle : Estelle*. In The Formal Description Technique Estelle. North Holland 1989. M.Diaz, J.P.Ansart, J.P.Courtiat, P. Azema, V.Chari (Ed).

8. J.P.Favreau, G.v.Bochmann, P.Mondin-Monval. Open Issues in Protocol Development. in Réseaux et Informatique Répartie n.2 HERMES 1991
9. J.P.Favreau. Mise en oeuvre et Test de Protocoles écrits en Estelle. Thèse de Doctorat. Université de Bordeaux I, n. 2153, Juin 1986.
10. B. Forghani. B. Sarikaya. Automatic dynamic behaviour generation in TTCN format from ESTELLE specification.
2nd International Workshop on Protocol Test Systems. Berlin October 1989.
11. S.Fujiwara. G.v.Bochmann. Testing non-deterministic state machines with fault coverage. In Proc. 4th International Workshop on Protocol Testing Systems. Leidschendam. The Netherlands. October 1991.
12. R.Groz. Vérification de propriétés logiques des protocoles et systèmes répartis par observation de simulations.
Thèse de Docteur de l'Université de Rennes I, no 194, Janvier 1989.
13. Discussion Summary on Formal Methods on Conformance Testing, ISO/IEC JTC1/SC21/WG1 S 58, Seoul 1990.
14. F.Guo, R.Probert. E-MPT Protocol Testing : Preliminary Experimental Results. 3rd IWPTS, Mc Lean, New Jersey, October 1990.
15. Systèmes de traitement de l'information - Interconnexion de systèmes ouverts - Protocole de transport en mode connexion,
Norme Internationale, ISO 8073, 1988 (F)-AFNOR.
16. Information Processing Systems. Open Systems Interconnection.Estelle (a formal description technique based on extended state transition model), ISO IS 9074, 1989.
17. ISO/IEC 9646-3, Information Technology - Open Systems Interconnection-Conformance testing methodology and framework-Part 3: The Tree and Tabular Combined notation(TTCN)
18. ISO/IEC 9646-4 : 1991, Information Technology - Open Systems Interconnection-Conformance testing methodology and framework-Part 4: Test Realization.
19. ISO/IEC 9646-5 : 1991, Information Technology - Open Systems Interconnection-Conformance testing methodology and framework-Part 5: Requirements on test laboratories and clients for the Conformance Assessment Process.
20. Information Processing Systems. Open Systems Interconnection.Estelle (a formal description technique based on extended state transition model), ISO IS 9074, 1989.
21. C.Jard. Valider les protocoles en simulant. dans Ingéniérie des Proto-

coles. Eyrolles 1988. R.Castanet, O.Rafiq (Ed)
22. C.Jard, J.F. Monin, Development of Véda, a prototyping tool for distributing algorithms. IEEE Trans. on Software Engineering, March 1988
23. O.Koné. Méthodes de Test de Protocoles. Actes du 1^0 Colloque Africain sur la Recherche en Informatique. Yaoundé, Octobre 1992.
24. O.Koné, R.Castanet. A case of concurrent canonical tester derivation, Technical Report LaBRI,1993
25. O.Koné, R.Castanet. Formal Description of Test Systems, Technical Report LaBRI,1993.
26. O.Koné. Interconnexion de Systèmes Ouverts. Test d'Interopérabilité, Test avec Contraintes de Temps Physique.
Thèse de Doctorat de l'Université de Bordeaux I. Février 1994.
27. R.J.Linn. An Evaluation of the ICST Test Architecture after Testing class 4 Transport. in PSTV IV. North Holland 1985. Y.Yemini, R.Strom, S.Yemini(Ed)
28. R.Molva. Conception et Réalisation d'un Observateur d'Architectures Multicouches dans des Réseaux d'Ordinateurs. Thèse de Doctoirat, Université Paul Sabatier, toulouse, Octobre 1986.
29. J.S. Nightingale. Protocol Testing using a Reference Implementation. PSTV, North Holland 1982. C.Sunshine (Ed)
30. O.Rafiq. R.Castanet. From Conformance Testing to Interoperability Testing. Proc. 3rd IWPTS. October-November 1990. Washington D.C.
31. P. de Saqui-Sannes. Prototypage d'un environnement de validation de protocoles: Application à l'approche Estelle. Thèse de Doctorat. Université Paul Sabatier. Toulouse, Févirer 1990.
32. R.Sijelmassi, B.Strausser. NIST Integrated Tool Set for Estelle. IFIP Conference on Formal Description Technique FORTE'90. Madrid, Spain. 1990.
33. VEDA, Protocol Design Verification. Administrator Guide, Reference Manual, Release 2.0. VERILOG S.A.(France), October 1991.

A. Spécification formelle du STC

Cette annexe fournit la spécification du STC et celle des observateurs qui lui sont associés. Le STC est configuré avec l'extrait d'une phase de connexion du protocole Transport (voir les figures 1, 2).

```
(***** SPECIFICATION DU STC   ******)

specification CONC_TEST_SYS systemactivity;
default individual queue;

channel UT_interface(user,provider);
  by user :
   TCONreq;
  by provider :
   TCONind;
channel TC_interface(user,provider);
  by user,provider :
   CR;
channel LT_interface(user,provider);
  by user, provider :
   CR;


module Tester_Entity_Type activity;
  IP
   IPUT : UT_interface(user) rendez_vous;
   IPTC : TC_interface(user) rendez_vous;
end;

module Protocol_Entity_Type activity;
  IP
   IPUT : UT_interface(provider) rendez_vous;
   IPLT : LT_interface(user) rendez_vous;
end;

module Sync_Chan_Entity_Type activity;
  IP
   IPLT : array[1..2] of LT_interface(provider) rendez_vous;
   IPTC : array[1..2] of TC_interface(provider) rendez_vous;
end;


body Tester_Entity_Body for Tester_Entity_Type ;
  var
   UNLOCK : BOOLEAN;


state s0,s1,s2,fail;
initialize to s0
   begin
    UNLOCK := true
   end;
trans
from s0 to s1
   PROVIDED(UNLOCK)
   IPUT!TCONreq
   begin end;

trans
from s0 to s2
   IPTC!CR
   begin end;

    to fail
   PROVIDED(UNLOCK)
   IPUT?TCONind
   begin end;
    to fail
   IPTC?CR
   begin end;

trans
from s1 to s0
   IPTC?CR
   begin end;
    to fail
```

```
    PROVIDED(UNLOCK)
    IPUT?TCONind
    begin end;

trans
from s2 to s0
    PROVIDED(UNLOCK)
    IPUT?TCONind
    begin end;
     to fail
    IPTC?CR
    begin end;

end;(* body Tester_Entity_Body *)


body Protocol_Entity_Body for Protocol_Entity_Type ;
state s0,s1,s2;
initialize to s0
    begin
    end;
trans
from s0 to s1
    IPUT?TCONreq
    begin end;
trans
from s0 to s2
    IPLT?CR
    begin end;
trans
from s1 to s0
    IPLT!CR
    begin end;
(* Erreur Inopportune *)
 trans
from s1 to s0
  IPUT!TCONind
    begin end;


trans
from s2 to s0
    IPUT!TCONind
    begin end;
end;(* body Protocol_Entity_Body *)


body Sync_Chan_Entity_Body for Sync_Chan_Entity_Type ;
  var
    UNLOCK : array[1..2] of  BOOLEAN;

state SC0,SCR11,SCR12,SCR13,SCR21,SCR22,SCR23;

initialize to SC0
begin
    UNLOCK[1] := true;
    UNLOCK[2] := true
end;
(* Sync_Chan Behavior for a CR *)

trans
from SC0 to SCR11
    IPLT[1]?CR
    begin
     UNLOCK[1] := false;
     UNLOCK[2] := false
    end;
trans
from SCR11 to  SCR12
    IPTC[1]!CR
    begin end;
```

```
trans
from SCR12 to SCR13
   IPTC[2]?CR
   begin end;
from SCR13 to SC0
   IPLT[2]!CR
   begin
    UNLOCK[1] := true;
    UNLOCK[2] := true
   end;

trans
from SC0 to SCR21
   IPLT[2]?CR
   begin
    UNLOCK[1] := false;
    UNLOCK[2] := false
   end;
trans
from SCR21 to  SCR22
   IPTC[2]!CR
   begin end;
trans
from SCR22 to SCR23
   IPTC[1]?CR
   begin end;
from SCR23 to SC0
   IPLT[1]!CR
   begin
    UNLOCK[1] := true;
    UNLOCK[2] := true
   end;

end;(* body Sync_Chan_Entity_Body *)


(*  ------------------------------------------------- *)

modvar
  Tester1,Tester2 : Tester_Entity_Type;
  Sync_Channel    : Sync_Chan_Entity_Type;
  IUT1,IUT2       : Protocol_Entity_Type;

initialize
  begin
   init Tester1      with Tester_Entity_Body;
   init Tester2      with Tester_Entity_Body;
   init Sync_Channel with Sync_Chan_Entity_Body;
   init IUT1         with Protocol_Entity_Body;
   init IUT2         with Protocol_Entity_Body;
   connect Tester1.IPTC to Sync_Channel.IPTC[1];
   connect Tester1.IPUT to IUT1.IPUT;
   connect Tester2.IPTC to Sync_Channel.IPTC[2];
   connect Tester2.IPUT to IUT2.IPUT;
   connect Sync_Channel.IPLT[1] to IUT1.IPLT;
   connect Sync_Channel.IPLT[2] to IUT2.IPLT;
  end;
end.







(***** SPECIFICATION DES OBSERVATEURS ********)
```

```
observation of CONC_TEST_SYS;
taskprobe
 Testo1 : Tester1;
 Testo2 : Tester2;
 SynCha : Sync_Channel;
observer obs;
state observant;
initialize to observant
  begin end;
trans
from observant to observant
  begin
   Testo1.UNLOCK := SynCha.UNLOCK[1];
   Testo2.UNLOCK := SynCha.UNLOCK[2];
   end;
end(obs);

observer obs1;
state observant,fail;
stateset error = [fail];
initialize to observant
  begin end;
trans
from observant to fail
  provided  Testo1.state = Testo1.fail
  begin
   writeln('+++ OK! Testeur 1 a detecte 1 erreur  +++');
  end;

end(obs1);
observer obs2;
state observant, fail;
stateset error = [fail];
initialize to observant
  begin end;
trans
from observant to fail
  provided  Testo2.state = Testo2.fail
  begin
   writeln('+++ OK! Testeur 2 a detecte 1 erreur  +++');
  end;

end(obs2);
end() .
```