Lot 3.2
Test

Techniques de test d'interopérabilité pour les systèmes temporisés

Description : Ce document présente une technique de génération de séquences de test d'interopérabilité pour les systèmes temporisés.

Auteur(s) : Ismail BERRADA, Richard CASTANET, Patrick FELIX.

Référence : AVERROES / Sous-Projet 3 / Fourniture 3.2.1 / V1.0

Statut : Validé

Version : 1.0

Réseau National des Technologies Logicielles
Projet subventionné par le Ministère de la Recherche et des Nouvelles Technologies

CRIL Technology, France Télécom R&D, INRIA-Futurs, LaBRI (Univ. Bordeaux 1 - CNRS), LIX (Ecole Polytechnique, CNRS), LORIA, LRI (Univ. Paris Sud - CNRS), LSV (ENS de Cachan – CNRS)

Liste des versions et révisions

Version / Révision	Date	Objet
1.0	20 novembre 2003	Version initiale

DOCUMENTS DE REFERENCE

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T.A.Henzinger, P.Ho, X.Nicollin, A.Olivero, J.Sifakis et S.Yovine, « The algorithmic analysis of hybrid systems », *Theoretical Computer Science*, 138(1) :3-34, 1995.

[2] R. Alur et D. Dill, « A theory of times automata », *Theoretical Computer Science*, 126:183-235, 1994.

[3] R. Alur, R. Kurshan, and M. Viswanathan. Membership problems for timed and hybrid automata, 19th IEEE Real-Time Systems Symposium, 1998.

[4] R. Alur, T.A. Henziger, and M.Y. Vardi, « Parametric real-time reasoning », 25th ACM Symposium on Theory of Computing, 1993.

[5] R. Anido, A. Cavalli, T. Macavei, L. P. Lima, M. Clatin, et M. Phalippou, « Engendrer des tests pour un vrai protocole grâce à des techniques éprouvées de vérification », CFIP'96, Rabat, Maroc, Octobre 1996.

[6] M. Benattou, L. Cacciari, R. Pasini and O.Rafiq. Principe and Tools for testing Open Distributed Systems. *Pro. of IWTCS'99*, Kluwer 1999, p. 77-92.

[7] I.Berrada, R.Castanet et P.Félix. « Formal approach for real-time test génération », *Workshop On Testing Real-Time and Embedded Systems*, Pise, Italie, Septembre 2003 (soumis).

[8] B. Berthomieu et M. Diaz, « Modeling and verification of Time Dependent Systems Using Time Petri Nets », IEEE Transactions on Software Engineering, 17(3), 1991.

[9] S.Bloch, H.Fouchal, E.Petitjean, et S.Salva, « Some issues on testing real time systems », *Int. Journal of Computer and Information Science*, n°2, Vol.4, Decembre 2001.

[10] G.V Bochmann, G. Das, R. Dssouli et M. Dubuc, « Fault models in testing », *IWTS*, 1991.

[11] A.Bouajjani, J.C.Fernandez, N.Halbwachs, C.Ratel, P.Raymond, « Minimal state graph generation », Science of Computer Programming, 18(3) June 1992.

[12] E. Brimksma, « A theory for the derivation of tests », In S. Aggarwal and K. Sabnani, editors. Proceedings of the 8th International Symposium on Protocol Specification, Testing and Verification. IFIP, North-Holland, 1988.

[13] W. Buehler. Introduction to ATM Forum Test Specifications, Version 1.0. *ATM Forum Technical Committee*, Testing SUbworking Group, af-test-0022.000.

[14] L. Cacciari and O. Rafiq. Contraintes temporelles dans le test réparti d'applications non temps-réel. *In CFIP'2000, Ingenierie des protocoles, Hermes*, October 2000.

[15] L. Cacciari and O. Rafiq. Controllability and Observability in distributed testing. *Information and Software Technology*, 41:767-780, 1999.

[16] R. Cardell-Oliver. Conformance Tests for Real Time Systems with Timed Automata Specification. Formal Aspects of Computing Journal, 12(5), 2000, 350-371.

[17] R. Castanet, C. Chevrier, O. Koné, B. Le Saec, « An adaptative test sequences generation method for the users needs », *8th IWPTS*, Evry, France, September 1995.

[18] R. Castanet et O. Koné, « Test generation for interworking systems », Computer Communication, Vol. 23, Issue 7, March 2000.

[19] D.Clarke et I. Lee, « Automatic test generation for the analysis of a real time system : case study » 3rd IEEE Real time Technology and Applications Symposium, 1997.

[20] M. Clatin, R. Groz, M. Phalippou, and R. Thummel, « Two approaches linking a test generation tool with verification technique », IWPTS'95, Ivry, France, 1995.

[21] E.A. Emerson, Handbook of *Theoretical Computer Science* (volume B), chapter "Temporal and Modal Logic », Ed. J. Van Leeuuwen, pages 995-1072, Elsevier, 1990.

[22] A. Ennouaary, R. Dssouli, A. Elqortobi, « Génération des tests temporisés », CFIP'97, Liège, Belgique.

[23] A. Ennouary, R. Dssouli, F. Kendek, et A. Elqortobi, « Timed test cases generation based on state characterization technique », 19 th IEEE RTSS'98.

[24] J.C. Fernandez, « An implementation of an efficient algorithm for bisimulation equivalence », *Science of Computer Programming*, 13(2-3), Mai 1990.

[25] J.C. Fernandez, C. Jard, T. Jéron, L. Nedelka et C. Viho, « Using On-the-Fly Verification Techniques for the Generation of Test Suites », *8th International Conference on Computer-Aided Verification, LNCS, 1102*, Springer Verlag, p. 348-359, août 1996.

[26] Robert W. Floyd. Algorithm 97 (shortest path),*Communications of the ACM*,18(3):165-172, 1964.

[27] J. Gadre, C. Rohrer, C. Summers, and S. Stmington. A COS Study of OSI Interopérability. *Computer Standards and Interfaces*, 9:217-237, 1990.

[28] G. Gonenc, « A method for the design of fault detection experiments », IEEE Transaction on Computers, 19(6), 1970.

[29] M.J.C. Gordon and T.F. Melham, « Introduction to HOL », Cambridge University Press, 1993.

[30] Anders Hessel, Kim G. Larsen, Brian Nielson, Paul Pettersson, and Arne Skou. Time-optimal Real-time Test Case generation using UPPAAL, To Appear.
[31] T.Higashino, A. Nakata, K.Taniguchi et R.Cavalli, « Generating test cases for a timed I-O Automaton model », *TestCom'99*, Budapest, Hongrie, septembre 1999.
[32] D. Hogrefe. Conformance Testing based on Formal Methods. *In FORTE 90, FOrmal Description Techniques* (ed. J. Quemada, A. Fernandez).
[33] IEEE. ANSI/IEEE 802.3, ISO/DIS 8802/3. IEEE Computer Press, 1985.
[34] Melania Ionescu. Une stratégie de test de télécommunications. *Ph.D Thesis*, University Every Val d'Essonne, 21 Juin 2001.
[35] ISO. Conformance testing methodology and framework, ISO 9646, 1991.
[36] ISO/TC97/SC21. OSI Conformance Testing Methodology and Framework - Parts 1-5,ISO , 1991.
[37] L. Kaiser. « Intéropérabilité temporelle à l'aide d'automates temporisés à entrées-sorties ». Mémoire de DEA, Université Henry Poincaré, Nancy, 1996.
[38] L. Kaiser. « Contribution à l'analyse des TIOSMs pour la vérification de propriétés temporelles de systèmes complexes », thèse INPL, 2001
[39] L. Kaiser et O. Kone, « Une méthode de vérification d'interopérabilité temporelle », *RENPAR'9*, 1997.
[40] A. Khoumsi. « A method for testing the conformance of real time systems », IEEE Int. Symp. on Formal Techniques in Real Time and Fault Tolerant Systems. Oldenbourg, Germany, September 2002
[41] A. Khoumsi. M. Akalay, R. Dssouli, A. Ennouary, L. Granger, « An approach for testing real time protocols », TestCom, Ottawa, Canada, 2000.
[42] P. Laurençot, « Intégration du temps dans le test de protocole de communication », thèse, Université Bordeaux I. Janvier 1999.
[43] P. Laurençot, O. Koné, and R. Castanet. « On the fly test generation for real time protocols », *International Conference on Computer Communications and Networks*, Louisiane, U.S.A., 1998.
[44] D. Lee et al. Passive Testing and Appilcations to Network Management. *ICNP'97 Internationnal Conference on Network Protocols*, Atlanta, Georgia, October 28-31, 1997.
[45] G. Luo, Rachida Dssouli, Gregor v. Bochmann, Pallapa Venkataram and Abderrazak Ghedamsi. Generating Synchronizable Test Sequences based on Finite State Machine with Distributed Ports. *IFIP TC6/WG6.1 Sixth International Workshop on Protocol Test Systems*, Pau, France 28-30 September, 1993.
[46] D. Mandrioli, S. Morasca and A. Mordenti. « Generating test cases for real time systems from logic specifications », *ACM Transaction on Computer Systems*, 13(4):365-398 , 1995.
[47] T. Mori, K. Tokuda, H. Tada, M. Higuchi, T. Higashino. « A method to generate conformance test sequences for FSM with timer system call », *FORTE 2001*, pp 301-318, August 2001.
[48] K. Myungchul, K. Gyuhyeong, D.C. Yoon. Interopérability Testing Methodology and Guidelines. Digital Audio-Visual Council, System Intergration TC, DAVIC/TC/SYS/96/06/006, 1996.
[49] S. Naito et M. Tsunoyoma, « Fault detection for sequential machines by transition tour », *IEEE Fault Tolerant Computing Conference*, 1981.
[50] X. Nicollin et J. Sifakis, et S. Yovine, « Compiling real-time specification into extended automata », *IEEE TSE Special Issue on RealTime Systems*, septembre 1992.
[51] R. Paige et R. Tarjan, « Tree partition refinement algoritms », *SIAM J.Computation*, n°6, 16, 1987.
[52] C. Paulin-Mohring. The Coq project, http://coq.inria.fr, 1999.
[53] L.C. Paulson. The Isabelle reference manual. Technical report 283, University of Cambridge, Computer Laboratory, 1993.
[54] E. Petitjean, « Etude des méthodes de test sur les systèmes temporisés », thèse, Université de Reims, novembre 2000.
[55] M. Phalippou, « Relation d'implantation et hypothèse de test sur des automates à entrées et sorties », thèse, Université Bordeaux I, 1994
[56] Projet RNRT Calife, « Automates temporisés », Fourniture 1.1, 12 avril 2000.
[57] Projet RNRT Calife, Bibliothèques Coq et Isabelle-HOL pour les systèmes de transitions et les p-automates, Fourniture 5.4, 21 juillet 2000.
[58] O. Rafiq and R. Castanet. From Conformance Testing to Interopérability Testing. *Pro. 3rd IWPTS*, October-November 1990, Washington D.C.
[59] D. Rouillard, « Application de techniques de preuve assistée pour la spécification, la vérification et le test », thèse, Université Bordeaux I, 2002.
[60] S. Salva, H. Fouchal et E. Petitjean, « Une méthode de test des systèmes temporisés orientée objectif de test », *RENPAR'14*, Hammamet, Tunisie, Avril 2002.
[61] P. Schnoebelen and all. « Vérification de logiciels, techniques et outils du model-checking », Vuibert, 1999.
[62] J. Springintveld, F. Vaandrager, et P.R. D'Argenio, « Testing timed automata », Theoretical Computer Science, 252 (1-2):225-257, March, 2001.

[63] A. Tanenbaum, « Computer Networks », Prentice Hall, Third Edition, 1996.
[64] T. Walter, I. Schieferdecker and J. Grabowski. Test Architecture for Distributed Systems - State of Art and Beyond. *IFIP TC6, 11th International Workshop on Testing of Communication Systems (IWTCS'98)*, Tomsk, Russia, August 31 - September 2, 1998.
[65] Marc F. Witteman and Ronald C. van Wuijtswinkel. ATM Broadband Network Testing Using the Ferry Principle. *IFIP TC6/WG6.1 Sixth International Workshop on Protocol Test Systems*, Pau, France 28-30 September, 1993.
[66] H.X.Zeng, and Rayner, D. The impact of Ferry Concept on Protocol Testing. *Protocol Specificqtion, Testing and Verification V*, North-Holland publishers, p. 519-531, 1986.
[67] H.X.Zeng, S.T. Chanson and B.R. Smith. On Ferry Clip Approches in Protocol Testing, *Computer Networks and ISDN Systems*, Vol. 17(2):77-88, 1989.
[68]

Table des matières

1. Introduction

Un système distribué est un système exploitant une architecture physique consistant en multiples, et autonomes éléments sans mémoire partagée communiquant à travers un réseau. Les éléments du système distribué peuvent être situés sur différents sites. Un système temporisé (ou temps réel) est caractérisé par la nécessité de respecter des contraintes temporelles imposées par son environnement : il évolue en réagissant aux événements provenant de son environnement.

Les domaines d'application des systèmes temporisés étant le plus souvent de nature critique, des efforts importants ont été faits dans des méthodes, techniques et outils pour leur conception afin de produire des implantations de qualité (sûrs).

Leur cycle de développement fait apparaître les étapes suivantes :

- Spécification, modélisation,
- Validation, vérification et preuve,
- Réalisation,
- Test.

La première étape mettra en évidence les contraintes fonctionnelles, comportementales et temporelles du système étudié : les réseaux de Petri temporisés et les automates temporisés sont parmi les formalismes de spécification les plus connus pour décrire de tels systèmes. Puis, lors de la deuxième étape, on s'assurera des bonnes qualités la modélisation et de la solution à implanter. La troisième étape aboutit alors à une implantation : même si des méthodes et des outils existent pour vérifier des spécifications et des modélisations temps réel, il est nécessaire de tester l'implantation pour s'assurer de différents aspects :

- tests de conformité qui vérifient l'adéquation d'une implantation à une spécification,
- tests de robustesse qui vérifient que l'implantation peut fonctionner dans un environnement présentant des dysfonctionnements,
- tests de performance qui permettent d'établir des configurations optimales (valeur maximale et/ou minimale de certains paramètres) pour un fonctionnement normal, voire efficace.
- tests d'interopérabilité qui vérifient la capacité d'au moins deux entités à inter opérer dans un contexte réel.

Le processus du test peut s'organiser selon les étapes suivantes (norme ISO 9646 [32]) :

- génération d'une suite générique de test,
- adaptation de la suite générique de test sur une architecture particulière (locale, distribuée, coordonnée) afin d'obtenir une suite abstraite de test,
- transformation de la suite abstraite de test en une suite de test exécutable propre à une implantation sous test particulière et à son environnement,
- exécution du test sur l'implantation et émission d'un verdict sur la conformité.

L'exécution d'un test peut conduire à trois sortes de verdict :

- « Réussi » (Pass) où les résultats du test sont en accord avec les résultats attendus,
- « Echec » (Fail) où une partie des résultats du test est en désaccord avec les résultats attendus,
- « Non concluant » (Inconclusive) lorsque les résultats obtenus, bien que cohérents avec la spécification, sortent du cadre du test effectué, et n'appartiennent pas aux résultats attendus.

On rappelle dans une première partie les principes des techniques de génération de séquences de test dans pour le test de conformité. Mais quand différents systèmes sont intégrés ou interconnectés avec d'autres, le système global doit être aussi testé afin de s'assurer que les composants inter opèrent correctement : tester différents systèmes afin de montrer qu'ils peuvent, ou non, interagir correctement correspond au test d'interopérabilité. La partie suivante nous présente l'état d'art, non exhaustif mais assez riche, des travaux relatifs au test des systèmes distribués. Nous commencerons par établir les définitions de l'interopérabilité. Les différents architectures sont exposées par la suite. Finalement, nous présenterons les travaux relatifs à la génération de séquences de test d'interopérabilité. Et dans une dernière partie, nous aborderons une étude de cas : CSMA/CD.

2. Quelques rappels sur le test de conformité

2.1. Problématique du test de conformité

Le test de conformité d'un système permet de s'assurer que le comportement observable d'une implantation est conforme à sa spécification : on soumet alors une suite d'interactions au système tout en observant son comportement. Différents types de test existent et peuvent s'organiser de la façon suivante :

- Boîte noire : Aucune connaissance sur le comportement interne du système n'est formulée, seuls les évènements observables (échangés entre le système et son environnement) sont pris en compte grâce à des points de contrôle et d'observation (PCO).
- Boîte blanche : Le comportement interne du système est connue en sa totalité.
- Boîte grise : En plus des évènements observables, le testeur accède aux messages de communications entre les parties du système. Ce accès se fait à travers des points d'observation (PO), ou des points de contrôle et d'observation (PCO).

L'architecture de test dépend du type de test que l'on veut réaliser sur une implémentation (test de conformité, interopérabilité, test boite noire,...), des éléments composant le système (test d'une seule entité ou de plusieurs), et de l'organisation en couches des protocoles du système. Plusieurs architectures ont été proposées reprenant une ou plusieurs de ces trois caractéristiques. Associées à ces architectures , des méthodes de test sont proposées par les organismes de normalisation. Par exemple, la « méthode locale » exploite l'architecture décrite dans la Figure 1 avec des points de contrôle et d'observation (PCO). Ce style d'architecture repose sur le principe du test de type « boîte noire » : durant le test, aucune information interne à l'entité à tester est connue du testeur.

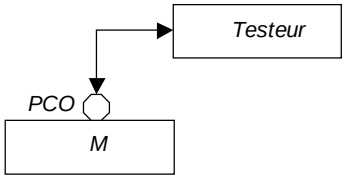

Figure 1. *Un exemple d'architecture de test de conformité*

Prenons l'exemple où M est le système avec les spécifications suivantes (Figure 2) où $!e$ et $?e$ représentent des communications entre le système M et son environnement :

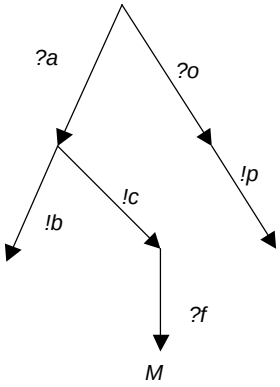

Figure 2. *Spécification du système M*

Une connaissance de la spécification du système M permet d'obtenir des idées de test à exécuter : on peut s'assurer qu'il est possible d'avoir, après la réception de l'événement a, l'envoi du message c suivi de la réception de f par M. Une séquence de test correspondante est $?a_1.!c.!e_2$. Et on souhaite bien naturellement automatiser cette recherche de séquences de test. Dans le cas de systèmes temporisés, une séquence de test temporisée est une séquence de la forme $(e_1, \tau_1).(e_2, \tau_2) \ldots (e_n, \tau_n)$, où e_i représente un événement d'entrée (resp. de sortie) $?a_i$ (resp. $!a_i$) et τ_i représente l'instant d'exécution de cet événement. Comme de telles implantations sont à priori non déterministes, différents instants d'exécutions sont possibles. Pour une séquence de test temporisée, on devra calculer les intervalles de temps qui

correspondent à des instants d'exécution corrects. Nous présentons sommairement quelques méthodes permettant de générer automatiquement de telles séquences de test dans le cas non temporisé et temporisé.

2.2. Architectures de Test pour le Test de Conformité des Protocoles

Les architectures pour le test de conformité des protocoles peuvent être classées comme suivant :

Architectures Standard OSI [36]. OSI définit un système sous test (SUT), système qui utilise les protocoles standards OSI depuis la couche physique jusqu'à la couche application, et une implémentation sous test (IUT), une implémentation d'un protocole OSI. L'interface entre une IUT et la couche adjacente dans SUT est appelée point de contrôle et d'observation (PCO). Une IUT possède un ou deux PCOs situées en bas ou en haut de l'IUT (dans un accès à distance à l'IUT, ce point est situé en bas). Le système de test (TS) qui communique avec l'IUT grâce aux PCOs, se compose d'une ou deux composantes, testeur supérieur (UT) et/ou testeur inférieur (LT). La communication entre le UT et le LT se formule à travers une procédure de coordination de test (TCP). Le canal de communication entre l'IUT et le TS peut être local ou distant et supposé fiable. OSI distingue quatre types d'architectures de test, pour le test de conformité des protocoles : l'architecture centralisée, l'architecture distante, l'architecture répartie et l'architecture coordonnée, qui sont définies essentiellement en termes de coordination entre les testeurs dans le processus de test. Dans l'architecture centralisée, deux PCOs sont définis avec le UT et le LT, mais peuvent être vus comme un port unique vu que l'IUT et le TS se trouvent sur le même site. Notons que le PCO attaché à l'UT n'accède pas réellement à l'IUT mais plutôt au SUT. Dans l'architecture répartie, un UT et un LT sont définis. Le LT est local au système de test et le UT est atteint à distance à travers le service de communication adjacent grâce à la TCP. Le testeur ne possède alors qu'un PCO (celui du LT). L'architecture coordonnée est semblable à l'architecture répartie, mais utilise un protocole de coordination (TMP), semblable au TCP, entre l'UT et l'LT. L'architecture distante correspond à l'architecture distribuée, mais seul le testeur inférieur est utilisé et par conséquent un seul PCO est exploité.

Architecture de Type Ferry [64][67]. Cette architecture est un support d'implémentation des architectures OSI. Elle définit un accès à un protocole grâce à des fonctionnalités supposées être implémentées dans le SUT. Dans le cas d'un accès aux bornes supérieure et inférieure de l'IUT dans la pile SUT, l'architecture est appelée *Ferry Clip*. Au niveau SUT, une fonctionnalité appelée *ferry clip passive* permet l'accès et le contrôle de l'IUT. Le système de test implémente une fonctionnalité appelée *ferry clip active* qui communique avec *ferry clip passif* grâce à un *service ferry transfert* et un *protocole de contrôle ferry*. Les UT et LT sont dans le même site et testent à distance l'IUT. Le *ferry clip* offre donc une médiation entre l'IUT et SUT. Cette architecture a été expérimentée dans [64]. Ce dernier élargit les fonctionnalités exigées sur le SUT et le ST : rajout de fonctionnalités de transfert des informations de contrôle en plus des informations de données au *ferry clip passif*, et l'addition d'un protocole de gestion des suites de test au TS.

Architecture Multi-port [64][44]. Cette architecture généralise les architectures OSI, dans le contexte d'une IUT communicant avec plusieurs entités ISO simultanément. Dans ce contexte, plusieurs UT et LT associés aux différentes interfaces, contrôlent et observent l'IUT. Le système de test [64] possède une fonctionnalité de contrôle qui permet de lancer l'ensemble des LT et de coordonner les différents testeurs dans le cas local (Les PCOs sont sur le même site que le SUT). Dans le cas d'interfaces distribuées, [44] introduit une méthode qui généralise la synchronisation entre deux testeurs à plusieurs testeurs.

2.3. Test de systèmes non temporisés

Cette partie rappelle les principales méthodes formelles non temporelles sur la génération automatique de test. Elles utilisent en général une description des protocoles par des systèmes à transitions (automates, systèmes de transitions étiquetées) ou par des techniques de description formelles, ayant un automate sous-jacent. Les différentes méthodes existantes qui se basent sur les automates peuvent être réparties en trois catégories présentées dans les paragraphes suivants.

2.3.1. Méthodes utilisant les automates d'états finis

Cette méthode est basée sur la structure de l'automate d'états finis. Une suite d'entrées est appliquée au système de communication à tester, et les sorties obtenues sont analysées. La conformité est établie par la comparaison des traces à celles de la spécification. Une erreur de sortie est le révélateur d'une ou plusieurs fautes d'implantation. Les principales erreurs possibles, identifiées dans un 'modèle de fautes' [10], sont les suivantes :

- Faute de sortie si pour une transition donnée, à la réception de l'entrée correspondante, l'implantation émet une sortie différente de celle qui est spécifiée.
- Faute de transfert si pour une entrée donnée reçue dans un état de l'automate, l'implantation atteint un état différent de celui spécifié.
- Transition manquante ou additionnelle pour des fautes multiples qui correspondent à une combinaison de fautes de transfert et de sortie.

Plusieurs techniques sont utilisées pour créer des séquences de tests qui permettent de détecter les erreurs spécifiées par le modèle de fautes. Les techniques les plus utilisées sont la méthode du « Tour de transition » [48], la méthode W, la méthode de la Séquence de Distinction (Distinguishing Sequence) [26], et la méthode d'Entrées-Sorties Uniques (UIO : Unique Input Output). Par contre, elles ne peuvent s'appliquer que sur une spécification ayant de bonnes propriétés (par exemple : automate minimal, fortement connexe,...).

2.3.2. Méthodes utilisant la théorie des testeurs canoniques

Dans cette méthode, le but est de comparer l'implantation à une spécification [12][55]. La démarche est la suivante :

- Une spécification S est considérée,
- Une propriété (appelée relation d'implantation) $\mathcal{R}$ est définie pour exprimer la notion de conformité avec S. $\mathcal{R}(I,S)$ signifie que I est une implantation conforme à S,
- Un testeur T, dit canonique, est défini pour tester $\mathcal{R}(I,S)$. T doit être capable de détecter si I viole la relation $\mathcal{R}$. Pour cela, T et I sont exécutés en parallèle, et l'échec du test correspond à un blocage de l'exécution.

Le testeur canonique T est fabriqué à partir de la spécification décrite sous la forme d'un système de transitions. Cette création s'appuie dans un premier temps sur l'utilisation d'un graphe de refus [20]. Ce dernier est un système de transitions déterministe, où les nœuds correspondent aux actions interdites à la spécification.

2.3.3. Méthodes orientées techniques de vérification

Les systèmes de communication sont habituellement modélisés par des automates d'états finis étendus sous-jacents dans des langages de spécification comme SDL. Des outils liés à ces types de langages ont été développés et commercialisés (TVEDA, ObjectGeode...). Ces outils permettent de construire le graphe de comportement global du système avec l'inconvénient connu d'une explosion combinatoire due essentiellement à la grandeur des domaines des variables de la spécification.

Le système TestGen [5], développé à l'INT, construit le graphe d'accessibilité du système, en utilisant le simulateur d'ObjectGeode. Une minimisation et une déterminisation sont ensuite effectuées (par exemple suppression des actions internes) avec l'aide des outils *Aldébaran* sur la base d'une équivalence de traces. Des séquences de type UIO sont extraites du graphe réduit pour obtenir des séquences de test.

Le système TVEDA, développée à FT R&D, est basé sur des heuristiques de sélection de tests à partir du développement partiel du graphe d'accessibilité. Un observateur permet de sélectionner les transitions pour lesquelles on veut produire un test.

2.3.4. Méthodes orientées objectif de test

Cette méthode est très différente des deux précédentes car son but est de limiter le problème de l'explosion combinatoire en créant une séquence de test à partir d'un « Objectif de Test ». Celui-ci correspond à une suite d'événements qui doivent ensuite figurer dans la séquence de tests pour vérifier la propriété voulue. Ces objectifs de test peuvent être créés à la main, ou générés automatiquement [25][17][43][20]. Parmi ces travaux, il faut plus particulièrement citer l'outil TGV, développé conjointement par l'IRISA et Vérimag, qui a été à l'origine du produit Testcomposer de Telelogic.

Un objectif de test OT se définit comme un automate déterministe et acyclique avec un ensemble non vide d'états particuliers noté $Accept(OT)$. $Accept(OT)$ définit un ensemble d'états acceptants qui permettent d'indiquer si l'objectif de test a été satisfait.

La démarche pour obtenir cette séquence de tests consiste à parcourir en parallèle l'objectif de test OT et la spécification jusqu'à trouver une séquence adéquate en utilisant une technique de produit synchronisé. Une séquence est adéquate lorsqu'il existe un chemin qui permet d'arriver dans un état inclus dans $Accept(OT)$. L'obtention de cette séquence peut se faire de différentes manières : par simulation, par parcours en largeur ou en profondeur de la spécification...

2.4. Test de systèmes temporisés

2.4.1. Aperçu sur différents travaux pour le test de systèmes temporisés

Plus récemment, les chercheurs se sont intéressés au test de systèmes répartis et au test de systèmes temporisés. L'applicabilité des anciennes méthodes et le développement de nouvelles méthodes sont en cours dans ces deux domaines.

La grande majorité des travaux de recherche en génération de tests temporisés sont basés sur le graphe de régions. Plusieurs approches utilisent le graphe de régions souvent en le réduisant [13][38][60] ou utilisent des modèles avec une référence à un module d'horloge, par exemple [40] qui utilise le modèle ioFSA avec 2 types d'événements Set et Exp sur l'horloge ou encore [23][44].

Dans [44], la génération de cas de test est obtenue à partir de formules logiques (le temps s'exprime avec deux constructeurs : *future* et *past*). Une horloge unique est utilisée et le domaine temporel est discret. Ce modèle ne permet pas de représenter des systèmes temps réel complexes et la suite de test générée ne contient que des valeurs entières du temps.

Dans [19], la spécification d'un système est basée sur un graphe contraint. A partir d'un modèle de fautes, les auteurs définissent des critères de test et génèrent des cas de test en accord avec le critère. Cette approche a quelques limitations : le graphe contraint utilisé comme modèle ne permet d'exprimer que des délais entre deux événements successifs et la couverture de fautes n'est pas complète.

Dans [13], une horloge implicite est utilisée, le temps est discret et le modèle utilisé est du type système à transitions temporisé. Dans ce cas, le modèle ne prend en compte que des domaines sur le temps et non des événements arrivant à des moments précis et le nombre de cas générés peut être très grand.

Dans [41], les auteurs prennent comme modèle les automates temporisés avec un temps discret. Le modèle est transformé en un automate sans temps, avec deux événements spéciaux sur les horloges (*set* et *expire*). L'avantage de la méthode est la limitation de l'explosion combinatoire, mais il est possible de générer des tests qui contiennent des séquences non exécutables.

Dans [23], les auteurs proposent de construire un automate de grille à partir du graphe des régions et utilisent la méthode Wp pour la génération de test, assurant ainsi une bonne couverture de la spécification, mais le nombre de tests générés peut être très grand.

Dans [62], les auteurs proposent une approche formelle très intéressante pour la génération de test à partir d'un modèle d'automate temporisé. Cette méthode peut conduire à engendrer un très grand nombre de tests.

Dans [9], les auteurs proposent d'élargir les régions du graphe des régions et construisent dynamiquement le graphe des régions selon des propos de test. Cette approche est intéressante car elle permet de limiter l'explosion combinatoire.

Dans [38], l'approche est voisine de celle que nous avons utilisée. La génération de cas de test est obtenue par un produit synchronisé entre une spécification temporisée et un propos de test tous les deux modélisés par un automate temporisé.

Dans [30], les auteurs proposent une méthode de génération basée sur une traçabilité « Must/May » qui permet de prendre en compte les séquences de test possibles et celles qui sont réellement exécutables.

2.4.2. Approche orientée « Graphe des régions »

Cette approche permet, à partir de la spécification du système sous forme d'un automate temporisé de générer une suite générique de test. Elle exploite directement l'automate des régions associé (automate non temporisé) en lui appliquant une technique de génération de test issue du domaine non temporisé. Cette approche a été exploitée dans [22][54]. Comme contrainte, on impose à l'automate temporisée à entrée/sortie de la spécification d'être déterministe.

La séquence de test obtenue dépend :

- Du chemin choisi dans l'automate des régions,
- Des durées choisies sur chaque transition de ce chemin.

Pour le premier point, toute approche issue du domaine non temporisé peut apporter une solution. Les approches qui ont déjà été proposées sont les suivantes :

- Une technique de « tour de transition »,
- Une approche basée sur la notion de caractérisation d'états[23],
- Une approche issue de [23] s'inspirant du testeur canonique synchrone intégrant également la notion d'objectif de test.

Pour le deuxième point, il est possible d'avoir plusieurs stratégies : choix aléatoire, s'approcher de la frontière de la région temporelle, rester au plus près du milieu de la région temporelle, …

Minimisation

L'inconvénient majeur de cette approche est l'explosion combinatoire liée à la taille de l'automate des régions. Afin de traiter des modélisations de taille plus importante, il peut être utile d'exploiter une phase de minimisation des systèmes manipulés, tout en préservant les propriétés du système. Dans la cadre des automates temporisés, plusieurs orientations apportent une solution à ce problème :

- définir différemment le graphe des régions afin de faire intervenir uniquement des régions d'horloge « utiles ». Cela peut se traduire par une nouvelle définition de la relation d'équivalence ~ sur les valuations d'horloge en tenant compte de la forme des contraintes. Le résultat est un graphe des régions « plus proche » de l'automate temporisé de départ. On trouvera une présentation de cette technique dans [54].
- minimiser l'automate des régions. L'algorithme de Paige et Tarjan de raffinement de partition [51] a été adapté aux systèmes de transitions étiquetées représentés de façon explicite pour le calcul de la bisimulation forte [24]. Cet algorithme a été combiné avec une représentation symbolique des ensembles d'états et de la relation de transitions étiquetées dans [11] : ce qui permet d'obtenir la minimisation, sans passer par l'expression explicite du système de transitions. Un algorithme analogue a été étudié dans le cadre des systèmes temporisés.

2.4.3. Approche orientée « Objectif de test »

Cette approche est basée sur l'expression d'un objectif de test et le produit synchronisé de la spécification et de l'objectif, permettant ainsi d'obtenir une séquence de test temporisée[42]. Le calcul du produit synchronisé utilise une technique liée au graphe d'états d'un réseau de Petri temporisé. La séquence est obtenue sans construire la totalité du graphe d'états (limitant ainsi l'explosion combinatoire) et de plus le graphe est construit dynamiquement (on ne le stocke pas). La construction pas à pas de la séquence de test nécessite un calcul à chaque pas d'un ensemble d'inéquations temporelles. Cette technique est aussi utilisée dans [38] mais avec des règles de synchronisation un peu différentes.

Dans [9], les auteurs proposent d'élargir les régions du graphe des régions et construisent dynamiquement le graphe des régions selon des propos de test. Cette approche est intéressante car elle permet aussi de limiter l'explosion combinatoire.

2.4.4. Approche orientée « Assistant de preuve »

Nous donnons dans cette partie les idées d'une approche utilisant les assistants de preuve : cette approche, exploitée dans le cadre du test de conformité de systèmes temporisés [58], peut-être étendue pour le test d'interopérabilité.

Une preuve constructive

Les assistants de preuve tels que *Coq* [52], *Isabelle* [53], *HOL* [29] manipulent la logique d'ordre supérieur dans laquelle il est possible de formaliser des modèles d'automate. La notion d'exécution d'un automate peut également être introduite comme une suite fine ou infinie de transitions[57].

Le problème de produire un cas de test satisfaisant une propriété ϕ donnée, revient alors à résoudre un énoncé de la forme :

$$s \xrightarrow[S]{x^?} t \wedge \phi(x^?) \qquad (1)$$

où la variable existentielle $x^?$ représente une exécution (encore inconnue) dans l'automate S et dont s et t sont respectivement les états origine et destination.

Le symbole $^?$ précise que x est une variable inconnue qui pourra donc être instanciée au cours de la preuve.

Si l'on parvient à démontrer cet énoncé, la valeur obtenue pour $x^?$ constituera un cas de test. Nous sommes ainsi ramenés à un problème de vérification classique, à la différence près que nous devons fournir une preuve constructive afin de pouvoir interpréter le résultat comme un cas de test.

Comment exprimer la propriété désirée ?

Les exécutions d'un automate sont représentées sous la forme de listes potentiellement infinies de transitions. Les propriétés qu'une exécution doit satisfaire peuvent donc être définies à l'aide des opérateurs sur les listes formalisés dans l'assistant de preuve. On pourra par exemple utiliser :

- Les opérateurs issus de logique temporelle ou modale [21], par exemple LTL: « Always » (□), « Eventually » (◊), « next » (O), « infinitely often » (□ ◊), …
- Les fonctions usuelles sur les listes : append, member, len, omega.
- les opérateurs d'expressions rationnelles : $L_1.L_2$, L^+ , L^* , L^Ω, …

Ces opérateurs sont généralement accompagnés de règles dérivées des définitions (règles d'égalité, d'introduction, d'élimination) qui facilitent leur manipulation. Dans le cadre de la génération de test, nous nous intéressons plus particulièrement aux règles dites d'introduction qui donnent les conditions suffisantes à la satisfaction d'une propriété. Par exemple :

$$\frac{x = x_1.x_2 \wedge \phi(x_2)}{\Diamond \phi(x)}$$

affirme que pour satisfaire la propriété $\Diamond \phi$, il suffit de trouver un découpage de l'exécution x en une exécution x_1 quelconque et une exécution x_2 qui satisfait ϕ.

L'application de telles règles impose des contraintes sur la forme de l'exécution $x^?$ recherchée. Cela permet d'instancier une portion de l'exécution, mais ne suffit néanmoins pas à construire une exécution complète. Il faut donc également étudier la première partie de la conjonction (1) c'est à dire rechercher dans l'automate les portions manquantes de l'exécution.

Construction d'une exécution

L'assistant doit faciliter cette tâche en fournissant des mécanismes efficaces sur les automates :

- La *simulation* : l'utilisateur peut faire évoluer pas à pas l'automate en avant ou en arrière en précisant quelle est l'action (ou la liste d'actions) à exécuter. Il s'agit de l'opération primitive qui permet de construire une exécution. La simulation est toujours possible car elle s'appuie sur la règle d'introduction des transitions du modèle considéré. L'assistant présente à chaque pas, des obligations de preuve à satisfaire. Ces obligations peuvent correspondre, par exemple dans le cas des p-automates, à la satisfaction des gardes ou des invariants. Selon la nature de l'automate, ces obligations peuvent être évacuées automatiquement.
- La *recherche d'exécution* : étant donnés deux états, le système est en mesure de proposer une liste d'exécutions possibles qui permettent de relier ces états. Dans l'idéal, la procédure est capable de tenir compte des portions déjà instanciées. Cette recherche peut être effectuée par un outil extérieur (par exemple Hytech). Le résultat est ensuite intégré dans l'assistant de preuve, soit sous la forme d'un axiome ou après une phase de validation.

3. Le test d'interopérabilité de systèmes temporisés

3.1. Définitions de l'Interopérabilité

L'aptitude à interopérer pour un ensemble d'équipements d'un système distribué est une condition nécessaire pour la bonne coopération du système ou pour l'interfonctionnement des différents processus d'applications supportées par les équipements. Il est dit alors que deux ou plusieurs équipements interopèrent si le comportement de l'entité composée par ces différents équipements coïncide avec le comportement attendu. Cette aptitude à interopérer comporte :

- Une aptitude à interopérer au niveau spécifications. Les différentes spécifications des composantes peuvent interopérer.
- Une aptitude à interopérer au niveau implémentions. Les différentes spécifications des composantes sont supposées interopérer, ainsi que leurs implémentations.

On distingue alors, le test d'interopérabilité dont l'objectif est l'interopérabilité des implémentations, de la vérification d'interopérabilité dont l'objectif est l'interopérabilité des spécifications. Le test d'interopérabilité consiste en la confrontation de deux ou plusieurs implémentations, puis la comparaison du comportement produit avec le comportement attendu. Ce concept d'interopérabilité a été introduit pour palier l'insuffisance du test de conformité à garantir l'interopérabilité et le bon fonctionnement des systèmes communicants. On peut encore citer les raisons suivantes :

- Les standards des protocoles restent ambigus malgré l'effort de formalisation: Ils permettent souvent des options et des choix qui peuvent se contredire. Les implémentations basées sur différents choix peuvent ne pas interopérer.
- L'effort de vérification n'est pas complet : seulement des aspects particuliers sont vérifiés.
- Les suites de test de conformité ne sont pas exhaustives : ils ne permettent pas de garantir la conformité aux standards. Les considérations de coût limitent la taille des suites de test, ainsi la couverture de test n'est pas complète.

Il est clair qu'en présence de l'exhaustivité des tests, le test d'interopérabilité perd sa motivation. En pratique, l'exhaustivité est en générale impossible (ressources limitées), et par conséquent le test d'interopérabilité est une phase importante. Avant de clôturer cette discussion sur la définition de l'interopérabilité, il est utile de citer d'autres définitions qui nous semblent intéressantes. ASD (Advanced Studies Department) du COS (Corporation for Open Systems) définit quatre types d'interopérabilité par rapport à deux critères [26] :

Le moyen utilisé pour montrer l'interopérabilité : par preuve (contexte théorique), ou par test (contexte pratique). Ce critère distingue la vérification d'une spécification et le test d'une implémentation.

Le nombre de couches considérés dans le système : une ou plusieurs couches.

Ces deux critères combinés définissent quatre types de d'interopérabilité:

- Interopérabilité théorique en couche.
- Interopérabilité théorique en système.
- Interopérabilité testable en couche.
- Interopérabilité testable en système.

Les notions d'interopérabilité en couche et en système sont les suivantes : 'en couche' est en relation avec les protocoles du même système (cas de protocoles OS (Open System)). Cette interopérabilité tente de tester/vérifier l'interopérabilité de deux composantes (chacune ayant été testée/vérifiée) à travers leur fonctionnalité au niveau du service partagé. 'En système' est en relation avec des composantes de deux ou plusieurs systèmes. Le but est de tester/vérifier l'interopérabilité des protocoles d'un système avec un autre. Notons que [34] définit le test horizontal pour interopérabilité testable en système, le test vertical pour interopérabilité testable en couche, et définit le test en diagonal pour l'appel de procédure dans le cas des protocoles.

Les travaux portant sur l'interopérabilité des systèmes comportent deux grands axes : définitions de l'architecture de test, et méthodes de générations de séquences de test d'interopérabilité.

3.2. Architecture de Test.

3.2.1. Test d'Interopérabilité uniquement.

Architectures Passives et Actives. Deux approches principales dans le contexte de test d'interopérabilité OSI ont été proposées : test actif et test passif [26][32]. La différence entre les deux approches, est que dans le test passif, le rôle du testeur est de vérifier le comportement entre l'IUT et le SUT (rôle observateur). Le test actif, par contre, permet un contrôle du SUT, une génération d'erreurs contrôlée et d'avoir plus de détails sur la communication. Le Forum ATM [13] définit une architecture qui consiste à connecter des testeurs aux différents IUTs, et de placer des POs ou des PCOs entre IUT-Testeurs ou IUT-IUT. Le genre de test, où seuls des POs sont utilisés et placés entre IUT-Testeurs [48], est appelé interopérabilité pur. L'intérêt de ce genre d'architecture est la facilité de mise en oeuvre, et dans certaine situation, cette architecture se présente comme l'unique solution [44].

Architectures OSI Distribuées [13][6][15]. Les architectures d'OSI traitent seulement le cas d'un protocole isolé dans le SUT. Dans le cas des systèmes distribués, Rafiq et al. [13][6][15] définissent des architectures équivalentes à celles d'OSI pour le test de conformité. Dans la méthode centralisée, le système de test est réduit à un seul testeur composé de plusieurs testeurs attachés aux différents ports du système réparti. Dans les autres méthodes, le système de test consiste en plusieurs testeurs. S'il n'existe pas de coordination entre les testeurs, chacun d'eux interagit avec le PCO lui correspondant indépendant des autres : on parle de méthode à distance. Si la coordination entre les testeurs est prise en charge par le processus de test, la méthode est dite répartie. Si cette coordination doit suivre un processus normalisé, la méthode est dite coordonnée. Cette dernière correspond à un cas particulier de la méthode répartie. La difficulté que pose le portage des architecture OSI vers les systèmes distribués est en terme de coordinations entre les différentes entités du TS

qui peuvent être physiquement réparties. L'étude de ces méthodes dans [6][15] établit que la méthode centralisée est la plus puissante des quatre en termes de contrôle et d'observation, et la méthode à distance est la plus faible. Par ailleurs, les auteurs établissent dans [13] les contraintes temporelles qui garantissent une équivalence entre la méthode centralisée et la méthode répartie dans la cas du test d'applications n'imposant pas de contraintes temporelles de fonctionnement à leur environnement. Notons, en fin, que ces architectures sont conçues pour un test de type boîte noire, vu que le système distribué est considéré comme une seule entité.

3.2.2. Test de conformité et d'Interopérabilité.

Pour pouvoir combiner le test de conformité et d'interopérabilité des systèmes répartis, l'architecture doit avoir un accès aux communications entre les différentes composantes. De ce fait, le test est de type boîte grise.

Architecture à cheval [58]. La première architecture de test d'interopérabilité est proposée par O. Rafiq et R. Castanet. Elle consiste en un testeur, composé de plusieurs composantes, centrale pour le test d'un système réparti. Le contrôle et l'observation des SUTs sont assurés par des PCOs entre TS et SUTs. L'architecture définit aussi des points d'observations POs implantés entre les SUTs.

Architecture Générique [64]. Une architecture générique pour la conformité, l'interopérabilité, la performance et la qualité de service est présentée dans [64]. Cette architecture propose une boîte à outils permettant d'exprimer les propriétés spécifiques d'une application ou un système réparti à tester. Elle se compose de plusieurs instances pour différents types de composantes :

- Implémentation sous test (IUT), i.e, une partie du système à tester.
- Interface de composante (IC) permet d'interfacer les différentes IUTs. Les ICs sont différentes des PCOs, du fait qu'ils n'accèdent pas directement aux IUTs (cas de composants imbriqués).
- Composant de test (TC) associé à une IUT, permet de formuler des verdicts après une coordination avec les autres TCs.
- Composant de contrôle (CC) permet d'établir l'environnement spécifique pour l'exécution des tests. Il peut être utilisé par un des TCs pour contrôler l'exécution du test et ainsi produire un verdict.
- Point de contrôle et d'observation (PCO), permet d'observer les communications et de contrôler ou guider les IUTs.
- Lien de communication (CL) pour décrire les communications possibles entre différentes composantes (TCs, IUTs,...).
- Système sous test (SUT) composé d'ICs et IUTs.

Les différents composants et propriétés d'un système peuvent être exprimés grâce à cette boîte à outils.

3.3. Une technique de génération de test d'interopérabilité de systèmes temporisés

3.3.1. Une approche orientée objectif de test

Cette recherche de séquences de test peut être automatisée en appliquant un algorithme classique d'accessibilité sur le comportement global du système. Reprenons notre exemple précédent : le comportement global de $M=M_1//M_2$ est fourni dans la figure (Figure 3) et l'application d'un algorithme d'accessibilité sur le graphe correspond au comportement global fournit, par exemple, les séquences : $?o_1.!q_2$, $?a_1.!e_2.?f_1$, et $?a_1.?g_2.!h_2$.

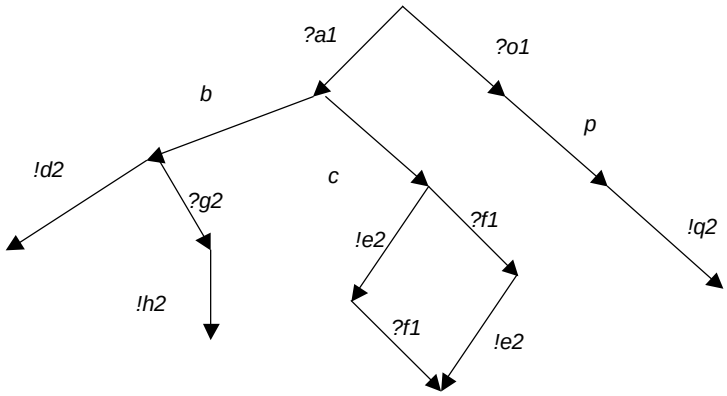

Figure 3. *Comportement global de M*

Le nombre de séquences de test que l'on peut ainsi obtenir étant important, se pose alors la question du choix des séquences de test « pertinentes ». Il est donc important d'avoir un algorithme d'accessibilité permettant de sélectionner les séquences de test : cette recherche de séquences dans le comportement globale peut alors être guidée par un objectif de test. L'objectif de test défini dans la figure suivante (Figure 4) permettrait d'obtenir uniquement les séquences $?a_1.!f_1.?e_2$ et $?a_1.?e_2.!f_1$ dans le cas d'un test de type boîte noire.

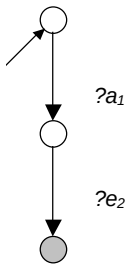

Figure 4. *Un objectif de test*

Dans le cadre des systèmes temporisés de taille réelle, les techniques algorithmiques basées sur la construction du système global sont confrontées à deux types potentiels d'explosion combinatoire résultant de la composition parallèle et des aspects temps réel : explosion dans l'espace des états de contrôle et explosion dans l'espace des valeurs des horloges et variables. Parmi les méthodes permettant d'éviter l'explosion combinatoire, citons :

- Stratégies de réduction du système global,
- Calcul « à la volée » de séquences de test sans calculer le système global.

Notre approche décrite dans 3.2 expose une technique de génération automatique de séquences de test temporisées du deuxième type.

3.3.2. Le modèle de spécification

Le modèle de spécification utilisé pour décrire le système étudié et les objectifs de test est celui des P-automates, modèle de référence du projet Averroès et du projet Calife [56]. Un p-automate est un tuple $<\Sigma, V, L, l_0, T, I, S, P>$ où :

- Σ est l'alphabet des actions,
- V est l'ensemble des variables dont les types peuvent être arbitrairement complexes,
- L est un ensemble fini de places,
- $l_0 \in L$ est la place initiale,
- P est un ensemble fini de paramètres,
- S est l'horloge universelle,
- I est une application qui étiquette chaque place l de L avec un invariant (ie une contrainte sur les variables et S)
- T est un ensemble de transitions de la forme (l,a,θ,l'), où l est la place de départ, l' la place d'arrivée, a une action et θ est une relation de mise à jour (ie une contrainte entre les valeurs des

variables avant le franchissement de la transition et les valeurs des variables, θ peut inclure une garde pour la transition*)*.

Notations : nous noterons <Σ(M), V(M), L(M), l_0(M), T(M),I(M),S,P(M)> le P-Automate associé au système *M*.

Dans le cadre de la génération de séquences de test, nous utiliserons une version restreinte des P-Automates [56] où :

- les paramètres sont instanciés (et deviennent donc des constantes),
- les types des variables que nous utiliserons sont *Time* (horloge), *nat* (entier) et *bool* (booléen).

3.3.3. Principes

Considérons le système temporisé *M* composé des deux systèmes M_1 et M_2 (Figure 5).

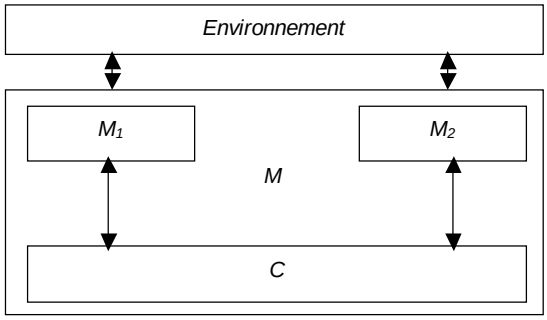

Figure 5. *Un exemple de système composé*

La figure suivante présente un d'architecture avec des points d'observation (PO) et des points de contrôle et d'observation (PCO). Ce style d'architecture avec la présence de points d'observation internes au système correspond à un test de type « boîte grise » dans le sens où on se propose d'observer quelques aspects à l'intérieur du système : cette observation concerne les aspects de communication entre les deux entités M_1 et M_2. Lorsque le système le permet, l'utilisation d'un testeur unique (Figure 6) évite les problèmes classiques de synchronisation de testeurs que l'on retrouve dans le cas de test distribué. Cette architecture suppose que le testeur puisse communiquer directement avec les entités M_1 et M_2 (ce qui n'est pas forcement possible dans le cas de test de composants imbriqués).

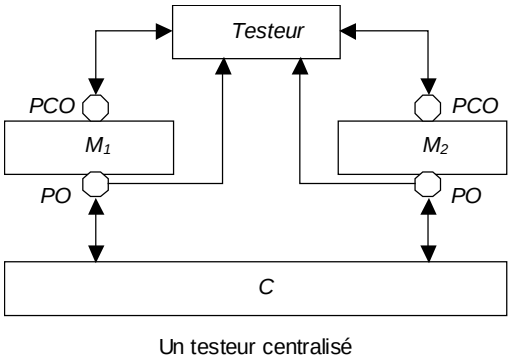

Figure 6. *Deux exemples d'architecture de test d'interopérabilité*

Prenons l'exemple où M_1 et M_2 sont les systèmes avec les spécifications suivantes (Figure 7) où $!e_i$ et $?e_i$ représentent des communications entre le système M_i et l'environnement du système *M*, alors que $!e$ et $?e$ représentent des communications internes au système *M* entre M_1, M_2 et le contexte *C*.

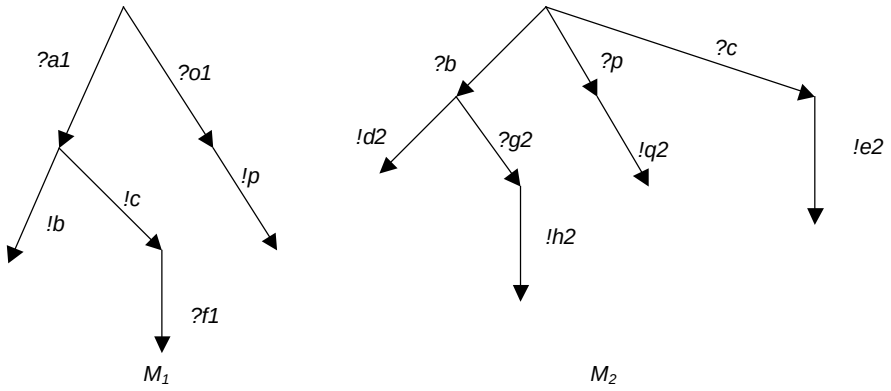

Figure 7. *Spécifications des systèmes M_1 et M_2*

Supposons que nous souhaitons tester une propriété temporelle P qui définit un certain comportement attendu. Dans notre approche, cette propriété est un objectif de test modélisé avec un P-automate que nous notons M_{OT}. Une séquence de test d'interopérabilité pour cette propriété est un comportement temporisé qui satisfait à la fois M_1, M_2 et M_{OT}. Une définition formelle du produit synchronisé de P-automates est définie dans [56] : nous étendons cette définition pour prendre en compte les interactions entre les systèmes M_1 et M_2, ainsi que la « synchronisation » avec l'objectif de test M_{OT}.

L'algorithme mis en œuvre calcule directement les séquences de test à la volée, pendant l'exploration de la concurrente spécification définie par M_1, M_2, et M_{OT}. Une séquence de test est obtenue quand un état acceptant de M_{OT} est atteint. Durant les calculs à la volée, les intervalles de temps caractérisant le succès ou l'échec du comportement du système étudié sont aussi calculés [6]. Pendant l'exécution du test, les instants d'exécution des interactions sont comparés à ces intervalles et permettent au système de test d'élaborer le verdict approprié : une erreur est détectée si la réaction de l'implantation est produite à l'extérieure de l'intervalle prévu.

3.3.3.1. Synchronisation de deux P-Automates

Soient $M_i=<\Sigma(M_i), V(M_i), L(M_i), l_0(M_i), T(M_i),I(M_i),S,P(M_i)>_{i\in\{1,2\}}$ deux P-Automates. Le système composé de ces deux P-Automates, que l'on notera $M_1//M_2$, est le système $M=<\Sigma(M), V(M), L(M), l_0(M), T(M),I(M),S,P(M_1)\cup P(M_2)>$, où les ensembles de place et de transitions sont les plus petits ensembles obtenus à partir des règles suivantes :

Règle R_0 : $l_0(M)=(l_0(M_1),l_0(M_2))\in L(M)$.

Règle R_1 : une transition est tirée dans M_1, mais pas dans M_2 :

$$\frac{(l_1,l_2)\in L(M) \wedge (l_1,a,\theta,l'_1)\in T(M_1) \wedge (l_2,a,\theta_2,l'_2)\notin T(M_2)}{(l'_1,l_2)\in L(M) \wedge ((l_1,l_2),a,\theta_1,(l'_1,l_2))\in T(M)}$$

Règle R_1' : Symétrique de R_1 (une transition est tirée dans M_2, mais pas dans M_1).

Règle R_2 : Une transition est tirée dans M_1 et M_2 :

$$\frac{(l_1,l_2)\in L(M) \wedge (l_1,a,\theta_1,l'_1)\in T(M_1) \wedge (l_2,a,\theta_2,s'_2)\in T(M_2)}{(l'_1,l'_2)\in L(M) \wedge ((l_1,l_2),a,\theta_1\wedge\theta_2,(l'_1,l'_2))\in T(M)}$$

3.3.3.2. Synchronisation contrainte par un objectif de test

Soient $M_i=<\Sigma(M_i), V(M_i), L(M_i), l_0(M_i), T(M_i),I(M_i),S,P(M_i)>_{i\in\{1,2\}}$ deux P-Automates modélisant deux STRE interagissant. Soit $OT=<\Sigma(OT), V(OT), L(OT), l_0(OT), T(OT),I(OT),S,P(OT)>_{i\in\{1,2\}}$ un P-Automate représentant un objectif de test. La synchronisation de M1 et M2 contrainte par l'objectif de test OT, que l'on notera $M_1/_{[OT]}/M_2$, est le système :

$M=<\Sigma(M), V(M), L(M), l_0(M), T(M),I(M),S,P(M_1)\cup P(M_2)\cup P(OT)>$,

où les ensembles d'états et de transitions sont les plus petits ensembles obtenus à partir des règles suivantes :

Règle R_0 : $l_0(M)=(l_0(M_1),l_0(M_2),l_0(OT))\in L(M)$.

Règle R_1 : une transition est tirée uniquement dans M_1 :

$$\frac{(l_1,l_2,l_{OT})\in L(M) \wedge (l_1,a,\theta_1,l'_1)\in T(M_1) \wedge (l_2,a,\theta_2,l'_2)\notin T(M_2) \wedge (l_{OT},a,\theta_{OT},l'_{OT})\notin T(OT)}{(l'_1,l_2,s_{OT})\in L(M) \wedge ((l_1,l_2,l_{OT}),a,\theta_1,(l'_1,l_2,l_{OT}))\in T(M)}$$

Règle R_1' : Symétrique de R_1 (une transition est tirée uniquement dans M_2).
Règle R_2 : Une transition est tirée dans M_1 et M_2, mais pas dans OT :

$$\frac{(l_1,l_2,l_{OT})\in L(M) \wedge (l_1,a,\theta_1,l'_1)\in T(M_1) \wedge (l_2,a,\theta_2,l'_2)\in T(M_2) \wedge (l_{OT},a,\theta_{OT},l'_{OT})\notin T(OT)}{(l'_1,l'_2,l_{OT})\in L(M) \wedge ((l_1,l_2,l_{OT}),a,\theta_1\cup\theta_2,(l'_1,l'_2,l_{OT}))\in T(M)}$$

Règle R_3 : Une transition est tirée dans M_1 et OT, mais pas dans M_2 :

$$\frac{(l_1,l_2,l_{OT})\in L(M) \wedge (l_1,a,\theta_1,l'_1)\in T(M_1) \wedge (l_2,a,\theta_2,l'_2)\notin T(M_2) \wedge (l_{OT},a,\theta_{OT},l'_{OT})\in T(OT)}{(l'_1,l_2,l'_{OT})\in L(M) \wedge ((l_1,l_2,l_{OT}),a,\theta_1\cup\theta_{OT},(l'_1,l_2,l'_{OT}))\in T(M)}$$

Règle R_3' : Symétrique de R_3 (une transition est tirée uniquement dans M_2 et OT, mais pas dans M_1).
Règle R_4 : Une transition est tirée dans M_1, M_2 et OT :

$$\frac{(l_1,l_2,l_{OT})\in L(M) \wedge (l_1,a,\theta_1,l'_1)\in T(M_1) \wedge (l_2,a,\theta_2,l'_2)\in T(M_2) \wedge (l_{OT},a,\theta_{OT},l'_{OT})\in T(OT)}{(l'_1,l'_2,l'_{OT})\in L(M) \wedge ((l_1,l_2,l_{OT}),a,\theta_1\cup\theta_2\cup\theta_{OT},(l'_1,l'_2,l'_{OT}))\in T(M)}$$

3.3.4. Faisabilité d'un chemin

La génération de séquences de test pour un chemin suppose qu'on est capable décider de l'atteignabilité de certains états de ce chemin. Dans cette partie, on considère le problème de faisabilité suivant : *Si un chemin donné d'un automate admet des exécutions, alors quelles sont les contraintes que doivent vérifier chaque exécution.* Dans cette section, on formalise ce problème en introduisant le concept de valeurs atteignables (RV), puis on propose une solution [7] par le biais du calcul des ces valeurs atteignables. Cette méthode d'analyse d'accessibilité pour des automates temporisés est adaptable aux P-Automates.

3.3.4.1. **Modèle**

Les automates temporisés [2] ont été introduits comme un formalise pour spécifier le comportement des systèmes temporisés. Dans cette section, le modèle de calcul, ainsi que certaines définitions sont présentées.

Notons par $\mathbb{R}$ l'ensemble des réels, $\mathbb{R}^+$ l'ensemble des réels positifs, et $\overline{\mathbb{R}}$ l'ensemble $\mathbb{R}^+ \cup \{\infty\}$. L'ordre partiel $\leq$, l'opération d'addition $+$ et l'opération de soustraction $-$ sur $\mathbb{R}$ sont étendues à $\overline{\mathbb{R}}$ comme suit : pour tout $t \in \overline{\mathbb{R}}$, $t \leq \infty$, $t + \infty = \infty + t = \infty$, et $\infty - t = \infty$. Un intervalle I de $\overline{\mathbb{R}}$ est un ensemble de $\overline{\mathbb{R}}$ noté par $[a,b]$. On écrit, $inf(I) = a$ et $sup(I) = b$. $\overline{I}$ est le complémentaire de I dans $\overline{\mathbb{R}}$. Pour $E = (E_{ij})_{i\in I, j\in J}$, $E' = (E'_{ij})_{i\in I, j\in J}$ and $\tau \in \mathbb{R}$: $\tau \times E = (\tau \times E_{ij})_{i\in I, j\in J}$, $E + E' = (E_{ij} + E'_{ij})_{i\in I, j\in J}$. Pour un ensemble fini Q, $Card(Q)$ est le nombre d'éléments de Q.

Horloges, Contraintes et Valuations. Une horloge est une variable qui permet d'enregistrer le passage du temps. Toutes les horloges évoluent de la même vitesse et les seules opérations possibles sur les horloges sont des remises à zéro : $x := 0$. Pour un ensemble d'horloges C, $\Phi(C)$ est l'ensemble des contraintes ϕ définies par la grammaire: $\phi := x \leq c \mid c \leq x \mid \phi_1 \wedge \phi_2$

avec x une horloge de C et c est une constante de $\mathbb{Q}$. Une valuation d'horloges ν pour l'ensemble C est une fonction de C dans $\mathbb{R}^+$ qui affecte une valeur réelle à chaque horloge.

Définition 1 (Automate Temporisé). Un automate temporisé (TA) A est un tuple (S, L, C, S_{init}, T) avec,

- S est un ensemble fini de sommets,

- S_{init} est l'état initial,
- L est un ensemble fini d'évènements,
- C est un ensemble fini d'horloges,
- $T \subseteq S \times L \times 2^C \times \Phi(C) \times S$ est l'ensemble de transitions. $t_i = (s, a, \phi, \lambda, s')$ représente un arc entre *s* et s' de symbole *a*. ϕ est un ensemble de contraintes sur t_i, et $\lambda \subseteq C$ est l'ensemble des horloges à remettre à zéro. Pour une transition t_i et une horloge x, $C(t_i)$ est l'ensemble d'horloges appartenant à t_i, et $SC(t_i, x)$ est la contrainte sur *x*: $\phi(x)$.

Définition 2 (TA avec Horloge Globale). Un automate temporisé avec une horloge globale (TAGC) est un automate temporisé, ayant une horloge *h* sans reset dans toutes transitions n'allant pas vers l'état initial.

Définition 3 (Exécution temporisée). Soit A un TAGC. Un chemin $\rho = t_1..t_n$ de A est une suite de transitions formant un chemin dans le graphe associé à A. Une exécution temporisée (TE) E de ρ est une matrice à deux dimensions de taille $n \times Card(C(\rho))$ [1] qui vérifie :

- Règle 1: $\forall x \in C$, $\forall i \in \mathbb{N}$, $i < n$: $E(t_i, x)$ est la valuation de *x* en t_i.
- Règle 2: $E(t_i, x) \in SC(t_i, x)$.
- Règle 3: $E(t_i, x) \leq E(t_{i+1}, x)$ si *x* n'est pas mise à 0 en t_i .
- Règle 4: Pour toute horloge *y*
  - Si *x* et *y* sont mises à 0 en t_i, alors $E(t_{i+1}, x) - E(t_i, x) = E(t_{i+1}, y) - E(t_i, y)$.
  - Si seulement *y* est mise à 0 en t_i, alors $E(t_{i+1}, x) - E(t_i, x) = E(t_{i+1}, y)$.
  - Si seulement *x* est mise à 0 en t_i, alors $E(t_{i+1}, x) = E(t_{i+1}, y) - E(t_i, y)$.
  - Si *x* et *y* sont mises à 0 en t_i, alors $E(t_{i+1}, x) = E(t_{i+1}, y)$.

Notation 1. Pour un chemin $\rho = t_1.t_2...t_n$ d'un TAGC A, $\rho_i = t_1.t_2...t_i$ est un sous chemin de ρ. $ES(\rho)$ est l'ensemble des exécutions de ρ. Si pour tout i dans $[1, n]$, a_i est l'évènement associé à t_i, alors: $(d_1, a_1).(d_2, a_2)...(d_n, a_n)$, avec $d_i \in \overline{\mathbb{R}}$, est l'exécution E définie par:

- Pour tout i dans $[1, n]$, $E(t_i, h) = d_i$.
- Pour tout i dans $[1, n]$, pour toute horloge x de $C(t_i)$ mise à 0 pour la dernière fois en t_k, $E(t_i, x) = d_i - d_k$.

Dans la suite, on suppose que $\rho = t_1..t_n$ est un chemin d'un TAGC *A* donnée de l'état initial, et n est son état final.

3.3.4.2. **Valeurs atteignables**

Définition 4. Soit $RV(t_i, x)_\rho$ un sous ensemble de $SC(t_i, x)$. $RV(t_i, x)_\rho$ est l'ensemble des valeurs atteignables de l'horloge x dans la transition t_i si: 1) Pour toute valeur $\tau \in RV(t_i, x)_\rho$, il existe une exécution $E \in ES(\rho)$ telle que: $E(t_i, x) = \tau$, 2) Pour toute exécution $E \in ES(\rho)$, $E(t_i, x)$ appartient à $RV(t_i, x)_\rho$.

Les valeurs atteignables (RV) d'une contrainte dans une transition dépend du chemin choisi. Le problème de faisabilité consiste à calculer des RV pour chaque horloge en chaque transition du chemin. En absence de confusion sur le chemin ρ, $RV(t_i, x)_\rho$ est noté $RV(t_i, x)$.

[1] $C(\rho)$ est l'ensemble des horloges utilisées dans ρ.

Corollaire 1. L'état n est un état atteignable de ρ ssi *pour tout* i *dans* [1,n], RV(t_i, h) n'est pas vide. Dans ce cas, ρ est dit faisable.

Lemme 1. Pour tout i dans $[1,n]$, pour toute horloge x de $C(t_i)$, $RV(t_i,x)$ est un intervalle de $\overline{\mathbb{R}}$.

3.3.4.3. Calcul des Valeurs Atteignables de l'Horloge h

Définition 5 (Intervalles Hors Portée). L'intervalle hors portée $h(t_i)$, d'une transition t_i du chemin ρ, est l'intervalle de temps dans lequel la transition est tirée[2] en dehors de cet intervalle. Si la transition est franchie dans cet intervalle, alors au moins une contrainte de ρ a été violée. Formellement, si pour $I=[a,b]$ et $J=[c,d]$ deux intervalles et $I \oplus J$ est l'intervalle défini par: $I \oplus J = [a+c, b+d]$, alors :

- $h(t_0) = \overline{[0,0]} =]0,\infty[$.
- $h(t_i) = \overline{\bigcap_{x \in C(t_i)} \overline{h}(t_{k_x}) \oplus SC(t_i,x)}$.

avec *x* remise à 0 pour la dernière fois en $t_{k_x}(k_x < i)$, et $\overline{h}(t_i)$ est le complémentaire de $h(t_i)$ dans $\overline{\mathbb{R}}$.

Définition 6 (Suite Bien Formée). Soit $(I_i)_{i\in[1,n]}$ une suite d'intervalles non vides de $\overline{\mathbb{R}}$, $I_i = [inf(I_i), sup(I_i)]$. $(I_i)_{i\in[1,n]}$ est une suite bien formée (WF) si pour tout i dans $[1,n-1]$, $inf(I_i) \le inf(I_{i+1})$, $sup(I_i) \le sup(I_{i+1})$. $(I_i)_{i\in[1,n]}$ accepte une WF sous suite $(J_i)_{i\in[1,n]}$ si pour tout i dans $[1,n]$, $J_i \subset I_i$, et $(J_i)_{i\in[1,n]}$ est WF.

Lemme 2. ρ est faisable ssi il existe une suite d'intervalles $(I_i)_{i\in[1,n]}$ telle que :

1. $(I_i)_{i\in[1,n]}$ est une suite WF.
2. Pour tout i dans $[1,n]$, $I_i \subset \overline{h}(t_i)$.
3. Pour tout i dans $[1,n]$, pour tout x de x $C(t_i)$ mise à 0 pour la dernière fois en t_j, $j<i$:
   - Pour tout τ_j dans I_j, il existe τ_i dans I_i tel que $\tau_i - \tau_j \in SC(t_i,x)$.
   - Pour tout τ_i dans I_i, il existe τ_j dans I_j tel que $\tau_i - \tau_j \in SC(t_i,x)$.

De plus, si pour tout i dans $[1,n]$, $I_i \subset \overline{h}(t_i)$ est le plus grand intervalle qui vérifie la troisième condition, alors $(I_i)_{i\in[1,n]}$ est la suite $(RV(t_i,h))_{i\in[1,n]}$.

Corollaire 2. Si ρ est faisable alors la suite des complémentaires des intervalles hors portée $(\overline{h}(t_i))_{i\in[1,n]}$ admet une sous suite WF.

Algorithme. Étant donnée un TAGC A et un chemin $\rho = t_1..t_n$ de A partant de l'état initial, l'algorithme du calcul de la suite $(RV(t_i,h))_{i\in[1,n]}$ est exposé dans la Fig. L'entrée de l'algorithme est un chemin $\rho = t_1..t_n$. Il utilise un système initialement vide S_ρ et un vecteur de *n* variables τ. L'algorithme distingue trois étapes : une étape de vérification dans laquelle la suite $(\overline{h}(t_i))_{i\in[1,n]}$ est calculée. Si cette suite n'admet pas une suite WF, alors le chemin est infaisable (corollaire 2). La deuxième étape consiste en la

[2] selon l'horloge h.

construction d'un système de contraintes, où chaque contrainte correspond à une relation entre quelques horloges conséquence du lemme 2. Enfin, une étape de résolution du système construit S_ρ. La méthode de résolution de ce système est exposée par la suite.

Analyse de Complexité : Nouvel algorithme de génération des estampilles. L'algorithme (Figure 8) introduit un système linéaire d'inéquations. Dans [3], les auteurs donnent un algorithme efficace de complexité $o(n \times k^2)$, pour la génération d'une séquence temporisée qui traverse chaque transition d'un chemin. L'algorithme calcule le plus court chemin $shortest_G(0,i)$ du nœud 0 au nœud i dans le graphe de contraintes G associé au système S_ρ (voir Annexe A). Cette solution peut être modifiée pour résoudre en partie le problème de faisabilité. En effet, dans notre cas, le plus court chemin valué $shortest_G(0,i)$ correspond à $\inf(RV(t_i,h)) : \inf(RV(t_i,h)) = shortest_G(0,i)$. Maintenant, si on considère le graphe de contraintes G' obtenu à partir de G en inversant les arcs, alors le même algorithme appliqué à G' donne $shortest_{G'}(0,i) = shortest_G(i,0) = \sup(RV(t_i,h))$.

Théorème 2. Étant donne un chemin ρ à partir de l'état initial d'un TAGC A, ayant n transitions et k horloges: Le calcul des RV pour l'horloge h dans chaque transition peut se faire en temps $o(n \times k^2)$.

Entrée: Un chemin temporisé ρ.
Sortie: Les valeurs atteignables de l'horloge h.
Structure de données: Un système S_ρ et un vecteur τ de n variables τ_i.
Début
Phase 1 : Vérification de la faisabilité de ρ.
/* S_ρ est initialement vide */
/* Vérifier si $(\overline{h}(t_i))_{i\in[1,n]}$ admet une sous suite WF */
Calcul de la suite $(\overline{h}(t_i))_{i\in[1,n]}$;
Si $(\overline{h}(t_i))_{i\in[1,n]}$ n'admet pas une sous suite WF Alors **exit**;
Pour $i := 1$ à n Faire
$SC(t_i,h) := \overline{h}(t_i)$;
Fin Pour
Phase 2 : Remplissage de S_ρ.
/* Addition de nouvelles contraintes (Lemme 2) */
Pour $i := n$ à 1 Faire
Si $i \neq 1$ Ajouter $\tau_{i-1} \leq \tau_i$ à S_ρ;
Ajouter $inf(SC(t_i,h)) \leq \tau_i \leq sup(SC(t_i,h))$ à S_ρ;
Pour $x \in C(t_i)$ Ajouter à S_ρ
/* x est mise à 0 pour la dernière fois en t_j */
$inf(SC(t_i,x)) \leq \tau_i - \tau_j \leq sup(SC(t_i,x))$;
Fin Pour
Fin Pour
/* Étape de résolution */
Phase 3 : Résoudre S_ρ.
Résoudre S_ρ;
Fin

Figure 8 Calcul de $(RV(t_i,h))_{i\in[1,n]}$

3.3.4.4. Calcul des valeurs atteignables de chaque horloge x

La génération des exécutions avec cumul temporel minimale et maximale est intéressante pour plusieurs raisons. Ces exécutions permettent en particulier tester la réaction à des évènements en avance ou tardifs dans le temps. De plus, il est généralement souhaitable, pour le test de régression, que le temps d'exécution des tests soit suffisamment court, ceci pour améliorer le temps du chargement lors des modifications des versions d'implémentation [30]. Comme ces exécutions ne sont pas uniques, dans cette section, nous identifions deux variantes pour chaque exécution avec cumul temporel minimal (resp. maximal) : exécutions avec cumul temporel minimal(resp. maximal) selon les transitions traversées et exécutions avec cumul temporel minimal (resp. maximal) selon toutes les horloges d'une transition. Enfin, nous exploitons ces résultats pour présenter une méthode de calcul des $(RV(t_i,x))_{i\in[1,n]}$, pour une horloge $x \neq h$.

Exécutions avec cumul temporel Minimal et Maximal selon les Transitions Traversées. Une des façons pour atteindre un état s au plus tôt (resp. tard) est de traverser chaque état entre l'état initial et l'état s le plus tôt (resp. tard) possible. Ce constat est formulé ci-dessous.

Lemme 3. Supposons que ρ est faisable.

1. Si pour tout i dans $[1,n]$, $RV(t_i,h)$ est bornée, alors il existe une unique exécution $E_S \in ES(\rho)$ telle que: pour tout i dans $[1,n]$, $E_S(t_i,h) = sup(RV(t_i,h))$.
2. Il existe une unique exécution $E_F \in ES(\rho)$ telle que: pour tout i dans $[1,n]$, $E_F(t_i,h) = inf(RV(t_i,h))$.

Exécutions avec cumul temporel Minimal et Maximal selon toutes les Horloges d'une Transition. Comme l'exécution minimale (resp. maximale) n'est pas unique, nous présentons par la suite une autre version.

Lemme 4. Supposons que ρ est faisable. Pour tout j dans $[1,n]$:

1. Si pour tout i dans $[1,j]$, $RV(t_i,h)_\rho$ est bornée, alors il existe une exécution $E \in ES(\rho)$ telle que: pour tout x in $C(t_j)$, $E(t_j,x) = sup(RV(t_j,x)_\rho)$
2. Il existe une exécution $E' \in ES(\rho)$ telle que: pour tout x dans $C(t_j)$, $E'(t_j,x) = inf(RV(t_j,x)_\rho)$.

Faisabilité Forte et Faible.

Définition 7. Un chemin faisable $\rho = t_1.t_2...t_n$ est dit fortement faisable si pour tout $i \in [1,n]$, et pour toute exécution $E' \in ES(\rho_i)$, il existe une exécution $E \in ES(\rho)$ qui prolonge E'. Dans le cas contraire, ρ est dit faiblement faisable.

Typiquement, ρ est fortement faisable si: pour tout i dans $[1,n]$, et pour toute exécution E' dans $ES(\rho_i)$, il existe E dans $ES(\rho)$ telle que: pour tout j dans $[1,i]$, pour toute x dans $C(t_j)$, $E(t_j,x) = E'(t_j,x)$.

La faisabilité forte permet de décider à l'avance si le comportement courant du système pourra atteindre le comportement souhaité. Ainsi, durant l'application du test sur une implémentation réelle, le testeur pourra décider d'arrêter le déroulement du test, à tout moment, et sans attendre la terminaison du cas du test. Le lemme 5 donne une condition nécessaire et suffisante pour la faisabilité forte.

Lemme 5. Soit $\rho = t_1.t_2...t_n$ un chemin faisable. ρ est fortement faisable ssi pour tout i dans $[1,n]$, et pour toute horloge x dans $C(t_i)$, t_i est franchis dans $RV(t_i,x)_\rho$.

Algorithme. Nous supposons que les horloges utilisées sont h, x_1, x_2, ... , $x_{Card(C(\rho))}$. L'algorithme du calcul des RV pour chaque horloge est exposé dans Figure 9. Au début de l'algorithme,

$H_m[k][j]$ contient la borne inférieure de la contrainte sur *x* (ie. $\inf(SC(t_k, x_j))$), et $H_m[k][j]$ la borne supérieure sur la contrainte sur *x* (ie. $\sup(SC(t_k, x_j))$). Dans la première phase, l'algorithme calcule les RV de l'horloge globale *h*. Pour chaque horloge *x* dans la transition t_i, mise à zéro pour la dernière fois à l'état initial, son RV est celui de l'horloge globale *h* (ie. $RV(t_i, x) := RV(t_i, h)$). H_m et H_M sont alors remises à jour.

La seconde phase consiste en une boucle sur les transitions à partir de t_1. Pour chaque transition courante t_i, on vérifie si l'ensemble des horloges à remettre à 0 est vide. Si cet ensemble est vide alors on incrémente l'indice de la transition. Sinon, on calcule le maximum et minimum délais entre cette transition et toute autre transition d'indice supérieur. Ceci est fait grâce au calcul des RV de h pour les chemins ρ_1 et ρ_2. ρ_1 et ρ_2 sont alors définis tels que $ES(\rho_1)$ et $ES(\rho_2)$ contiennent toutes les exécutions qui atteignent simultanément, respectivement, le maximum et le minimum des RV des horloges de la transition courante t_i ($ES(\rho_1)$ et $ES(\rho_2)$ ne sont pas vides selon lemme 4). Une fois que les RV de h pour ρ_1 et ρ_2 sont calculés, pour chaque horloge x_k dans $C(t_j)$, telle que j est plus grand que i, et x_k est mise à 0 pour la dernière fois dans t_i, on a alors $inf(RV(t_j, x_k)_\rho) := inf(RV(t_j, h)_{\rho_2}) - inf(RV(t_i, h)_{\rho_2})$, et $sup(RV(t_j, x_k)_\rho) := sup(RV(t_j, h)_{\rho_2}) - sup(RV(t_i, h)_{\rho_2})$.

Entrée : Un chemin faisable $\rho = t_1 ... t_n$ ayant n états et k horloges.
Sortie : Calcul des RV pour chaque horloge.
Structure de données : H_m et H_M deux matrices de taille $n \times k$.
Variables temporaires : Deux chemins $\rho_1 = t_1 ... t_n$ et $\rho_2 = t_1 ... t_n$
Début :
Phase 1 : Remplissage de H_m et H_M avec les RV des horloges mises à 0 dans l'état initial.
Calcul $(RV(t_i, h)_\rho)_{i \in [1,n]}$;
Pour $i := 1$ à n Faire
Pour tout $x_k \in C(\rho)$, Si x_k est mise à 0 pour la dernière fois à l'état initial Alors
$H_m[i][k] := inf(RV(t_i, h))$; $H_M[i][k] := sup(RV(t_i, h))$;
Phase Deux: Remplissage de H_m et H_M avec les autres RV.
Pour $i := 1$ à n Faire
S'il y a des resets horloges dans t_i Alors
Pour ρ_1 : $\forall x_k \in C(\rho)$, $SC(t_i, x_k) := [H_m[i][k], H_m[i][k]]$;
Pour ρ_2 : $\forall x_k \in C(\rho)$, $SC(t_i, x_k) := [H_M[i][k], H_M[i][k]]$;
/* La faisabilité de ρ_1 et ρ_2 est assurée par lemme 4 */
Calcul $(RV(t_i, h)_{\rho_1})_{i \in [1,n]}$;
Calcul $(RV(t_i, h)_{\rho_2})_{i \in [1,n]}$;
Pour $j := i$ à n Faire
Pour tout $x_k \in C(\rho)$ Faire
/* Mise a jour de H_m et H_M */
Si $x_k \in t_j$ est mise à 0 pour la dernière fois en t_i Alors:
$H_m[j][k] := inf(RV(t_j, h)_{\rho_2}) - inf(RV(t_i, h)_{\rho_2})$;
$H_M[j][k] := sup(RV(t_j, h)_{\rho_1}) - sup(RV(t_i, h)_{\rho_1})$;
Fin;

Figure 9 Algorithme du calcul des RV.

Théorème 3. Étant donne un chemin ρ à partir de l'état initial d'un TAGC A, ayant n transitions et k horloges: Le calcul des RV pour toutes les horloges du chemin peut se faire en temps $o((n \times k)^2)$.

3.3.4.5. **Adaptation aux P-Automates**

Traduction d'un P-automate vers un TAGC...

3.3.5. Algorithme de génération des séquences de test

Étant données deux spécifications (S) S_1, S_2 et un objectif de test (OT) T_p, on cherche à calculer un chemin qui respecte les spécifications et l'objectif de test. Le produit synchrone défini précédemment permet le calcul d'un chemin qui coïncide avec les événements de S et de OT, mais le chemin obtenu peut être 'non faisable'. Dans cette section, nous présentons un algorithme qui permet le calcul d'un chemin qui rencontre les spécifications S_1, S_2 et l'objectif de test T_p sur les événements et sur les contraintes temporelles. On définit alors les types de données suivants :

- $State$: un triplet (s_1, s_2, s_3).
- $ListState$: une liste d'éléments de type $State$.
- $ListEnt$: une liste d'éléments de type entier.
- $StateEven$: un doublet (e, μ), où e est un $State$, et μ un événement.
- $ListEven$: une liste d'éléments de type $StateEven$.
- $StackEven$: une pile d'éléments de type $StateEven$.
- $StateSucc$: un doublet (e, l), où e est un State, et l une $ListEven$.
- $StackSucc$: une pile d'éléments de type $StateSucc$.

Opérations sur les données :

- $Add(State, StateSet)$: Ajoute $State$ à $StateSet$ s'il n'existe pas.
- $Head(Stack)$: Renvoie le sommet ($StateEven$ ou $StateSucc$) de la pile $Stack$ ($StackEven$ ou $StackSucc$).
- $Pop(Stack)$: dépile l'élément du sommet ($StateEven$ ou $StateSucc$) de la pile Stack ($StackEven$ ou $StackSucc$).
- $Push(Stack, state)$: empile l'élément $state$ ($StateEven$ ou $StateSucc$) dans la pile $Stack$ ($StackEven$ ou $StackSuc$).
- $Successors(State)$: renvoie la $ListEven$ contenant les successeurs possibles de $State$ en appliquant l'une des règles du produit synchrone. Un élément de $ListEven$ est un couple $(State_1, \mu)$, tel que $(State, \mu, State_1)$, est une transition obtenue par l'une des règles de synchronisation.
- $Feasible(StackEven, StateEven)$: retourne vrai si le chemin constitué par l'empilement de $StackEven$ dans la pile $StackEven$ est faisable. Pour déterminer si un chemin est faisable, on applique les algorithmes du chapitre précédent.
- $Visited(ListSate, State)$: retourne 0 si $State$ n'appartient pas à $ListState$, sinon son indice.
- $IncVisited(ListEnt, Indice)$: incrémente la valeur stockée dans $ListEnt[Indice]$.
- $nVisited(ListEnt, Indice, nb)$: retourne vrai si la valeur de $ListEnt[Indice]$ vaut nb.

Algorithme. L'algorithme de calcul d'un chemin faisable de la synchronisation des spécifications et de l'objectif de test est présenté dans Figure 10. Les entrées de l'algorithme sont un objectif de test, deux spécifications et un entier représentant le nombre maximal d'apparition d'un nœud dans le chemin recherché. L'algorithme utilise les variables suivantes : e_1 et e_2 pour mémoriser l'état actuel (du produit S_1, S_2 et T_p) et l'état suivant (par l'application de l'une des règles de synchronisation). La liste $Visited$ est utilisée pour

enregistrer les états déjà rencontrés, et la liste $Occur$ pour le nombre d'occurrences de chacun de ces états. SP contient le chemin parcouru et $Paths$ les chemins possibles.

L'algorithme comporte deux phases : phase d'initialisation des différentes variables et phase de recherche du chemin faisable. Partant d'un état courant, les différentes possibilités par application de l'une des règles de synchronisation sont stockées dans $Paths$. Le choix d'une possibilité n'est effectué (donc stocké dans SP) qu'après vérification de faisabilité du chemin obtenu, ainsi que du nombre d'occurrences du nouveau nœud dans ce dernier (ne doit pas dépasser n). L'arrêt de l'algorithme se produit lors du franchissement d'un état acceptant ou le parcours de toutes les possibilités d'évolutions.

Entrée : Deux spécifications S_1, S_2, un objectif de test T_P et entier n.

Sortie : Un chemin faisable SP.

Structure de données : $State\ \ e_1, e_2$; $ListState\ \ Visited$; $ListEnt\ \ occur$; $ListEven\ \ l$; $StackEven\ \ SP$; $StackSucc\ \ Paths$; $int\ \ indice$;

Début

Phase 1 : Initialisation des données.

/* Initialisation */

$Visited := \emptyset$; $SP := \emptyset$; $Paths := \emptyset$;

$e_1 := (s_{init}(S_1), s_{init}(S_2), s_{init}(T_P))$;

$l := Successors(e_1)$;

$Add(e_1, Visited)$;

$IncVisited(occur, 0)$;

$Push(SP, (e_1, \emptyset))$;

$Push(Path, (e_1, l))$;

Phase 2 : Recherche du chemin synchrone.

Tant que $(Paths \neq \emptyset)$ Faire

$(e_1, l) := Head(Paths)$;

Si $(l \neq \emptyset)$ Alors

$(e_2, \mu) := Head(l)$;

$Pop(l)$;

$indice := Visited(Visited, e_2)$;

Si $indice \neq 0$ où $nVisited(occur, indice, n)$ Alors

Si $Feasible(SP, (e_2, \mu))$ Alors

$Push(Paths, (e_2, Successors(e_2)))$;

$Push(SP, (e_2, \mu))$;

$Add(e_2, Visited)$;

$IncVisited(occur, Indice)$;

Si $(e_2 \in Accep(S_1 \times S_2 \times T_P))$ Alors retourner SP;

Fin Si;

Fin Si;

Sinon

$Pop(SP)$; $Pop(Paths)$;

Fin Si;

Fin Tant que;

Retourner $\emptyset$;

Fin;

Figure 10 Calcul d'un chemin faisable du produit synchrone

4. Une étude de cas : CSMA/CD

4.1. Introduction

CSMA/CD ([32][63]) (*Carrier Sense Multiple Access with Collision Detection*) est un protocole d'accès multiple par écoute de la porteuse et détection des collisions. Ici on s'intéresse au système *CSMA_CD* composé uniquement de deux entités $Sender_{i=1,2}$ et une entité *Receiver* reliées au canal *Bus* : $CSMA\_CD=Sender_1//Sender_2//Bus//Receiver$ (Figure 11).

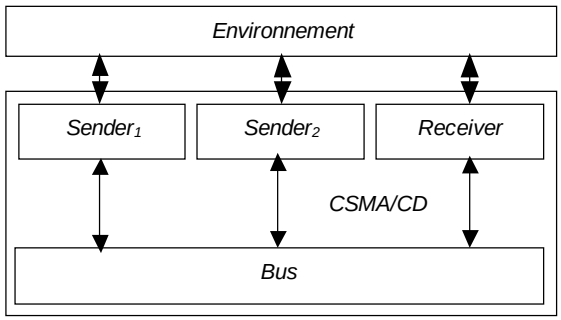

Figure 11. *Le système CSMA_CD*

On se propose de tester l'interopérabilité entre les entités $Sender_{i=1,2}$ et l'entité *Receiver*. Le test d'interopérabilité de cet exemple a les caractéristiques suivantes :

- Architecture de test centralisé : un unique testeur.
- Test de type boîte grise : des points d'observation internes au système *CSMA_CD* seront accessibles depuis le testeur.
- Test non imbriqué : le testeur communique directement avec les entités *Sender* et *Receiver*.
- Le contexte des entités à tester n'est pas vide : il est constitué du *Bus*.
- Pas de communication directe entre les entités *Sender* et *Receiver*

4.2. Architecture de test

On utilisera une architecture de test centralisé où le testeur communique directement (test non imbriqué) avec les entités $Sender_{i=1,2}$ et l'entité *Receiver*. On se propose d'observer quelques aspects à l'intérieur du système : cette observation porte uniquement sur les aspects de communication entre les différentes entités à tester du système avec l'entité *Bus* (test de type « boîte grise »).

Le test d'interopérabilité de ce système consiste à soumettre une suite d'interactions directement aux entités $Sender_{i=1,2}$ et *Receiver*, tout en observant leurs comportements afin de s'assurer qu'elles interagissent conformément à la spécification du système *CSMA_CD*.

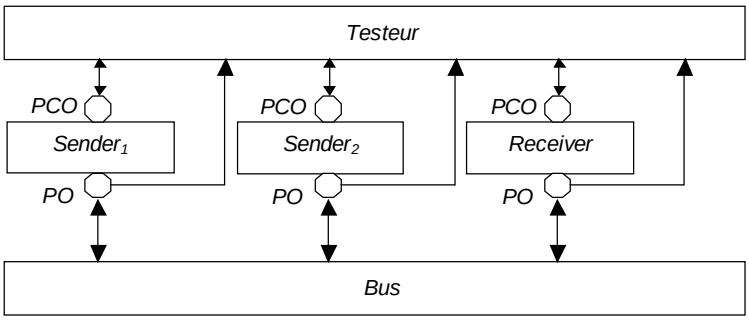

Figure 12. *Architecture de test pour CSMA_CD*

4.3. Modélisation

De nombreuses modélisations de CSMA/CD ont été proposées dans la littérature (Kronos[50], Calife…) en s'intéressant uniquement aux émetteurs : la variante que nous présentons se propose d'avoir aussi un récepteur afin de s'intéresser aux aspects d'interopérabilité entre émetteur et récepteur de CSMA/CD. Les figures suivantes présentent la spécification des systèmes $Sender_{i=1,2}$, *Receiver* et *Bus* où *Lambda* représente le temps d'émission d'une trame CSMA/CD et *Sig* représente le temps de propagation entre les deux stations les plus éloignées.

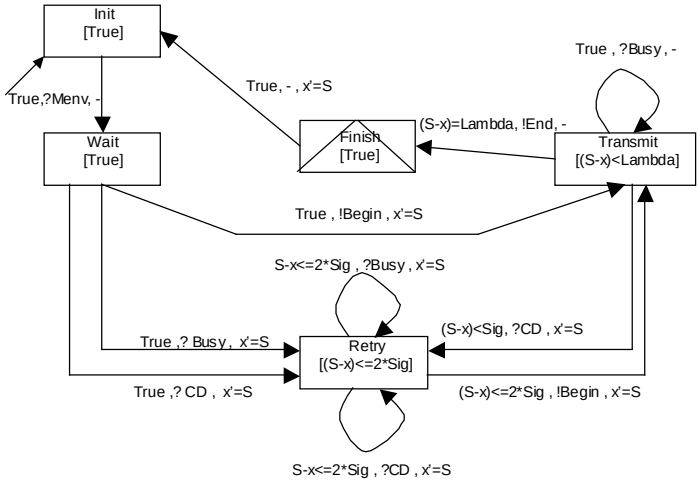

Figure 13 *Spécification de Sender*

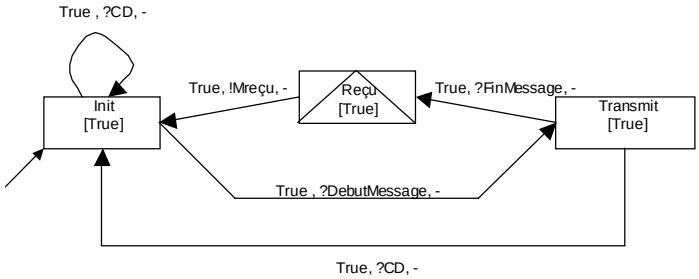

Figure 14 *Spécification de Receiver*

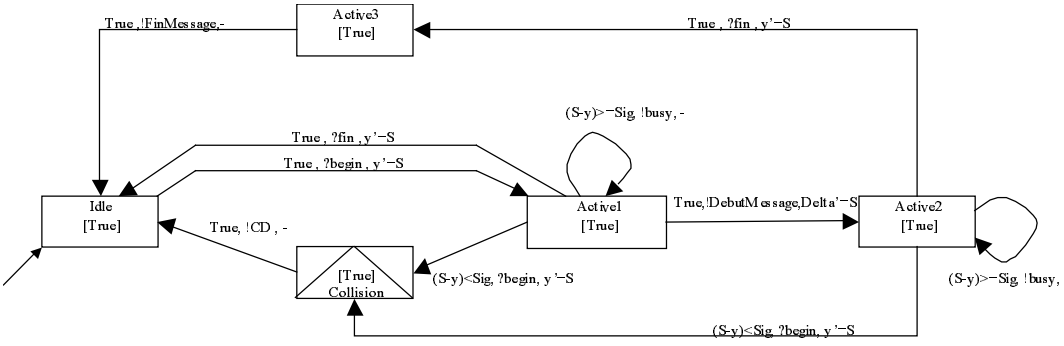

Figure 15 *Spécification de Bus.*

Les messages *?Menv* et *!Mrec* représentent des communications entre les entités à tester et l'environnement du système *CSMA_CD*, alors que *busy, cd, begin,* et *fin* représentent des communications internes au système *CSMA_CD* entre $Sender_i$ et *Bus*, et *DebutMessage* et *FinMessage* représentent des communications internes au système *CSMA_CD* entre *Receiver* et *Bus.*

4.4. Objectifs de test

Voici quelques exemples de propriétés à tester sur l'implantation :

1. Un message fourni à un *Sender* est restitué par le *Receiver*.
2. Un message fourni à un *Sender* à l'instant *t1* est restitué par le *Receiver* entre les instants *t1*+*a* et *t1*+*b*.
3. Deux messages fournis aux deux *Sender* sont restitués par le *Receiver*.
4. Deux messages fournis 'simultanément' aux deux *Sender*, sont restitués par le *Receiver*.
5. Deux messages fournis 'simultanément' aux deux *Sender*, sont restitués par le *Receiver* après avoir provoqué une collision.

Seule la propriété 5 doit faire l'objet d'un test de type boîte grise : on souhaite observer un événement (*CD*) entre le contexte (*Bus*) et une entité à tester (*Sender*) après l'envoi des deux messages aux *Sender*. La figure suivante (Figure 16) fournit des objectifs de test correspondant à ces propriétés. Ces objectifs de test sont exprimés sous forme de P-Automate où *S* est l'horloge et *C* une variable locale de type *Time* :

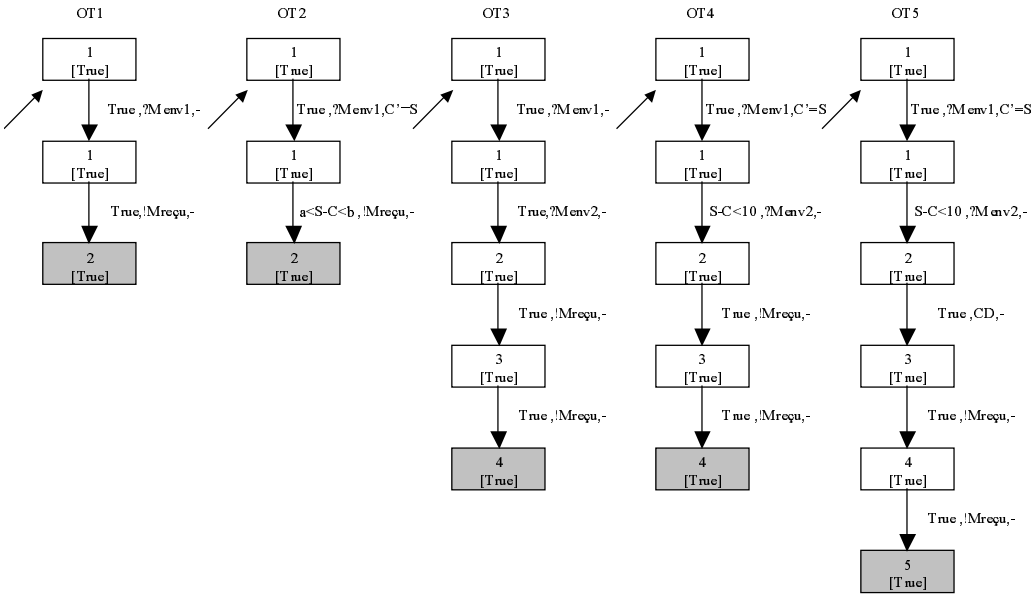

Figure 16. *Objectifs de test.*

4.5. Recherche d'une séquence de test

La figure suivante (Figure 17) nous donne le comportement global (une partie bien sûr !) de *CSMA_CD=Sender*$_1$*//Sender*$_2$*//Bus//Receiver*. Si on s'intéresse à l'objectif de test *OT5*, l'application d'un algorithme d'accessibilité sur le système *CSMA_CD//OT5* nous fournit, par exemple, la séquence de test représentée en gras dans la figure suivante (Figure 17).

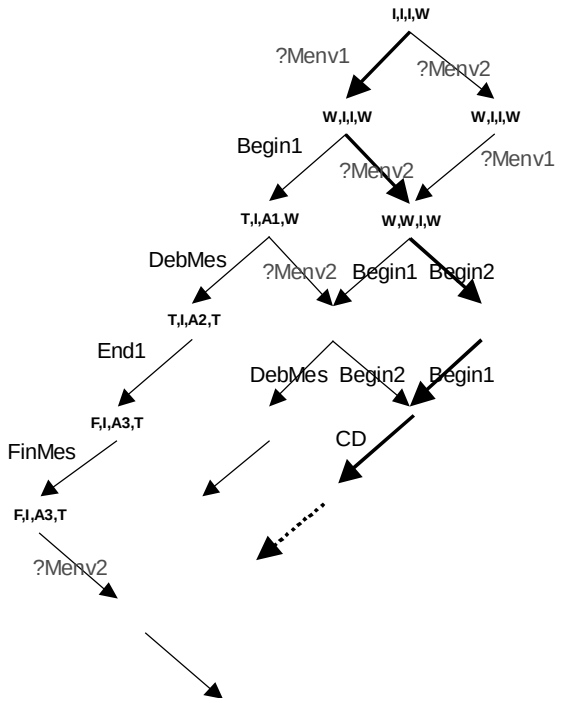

Figure 17. *Une partie du comportement global de CSMA_CD .*

5. Conclusion

La dernière étape, qui n'est pas traitée dans ce document, est la constitution d'un cas de test dérivé de la séquence à tester, en tenant compte de l'environnement de test, de l'architecture de test et en inversant les émissions et les réceptions d'événements du scénario de test (par exemple si la spécification contient une émission d'un événement, qui appartient au scénario, le cas de test doit contenir la réception de cet événement).

Une des tâches du sous-projet 3 est d'intégrer un outil de générations de séquences de tests d'interopérabilité dans la plate-forme Calife, et de valider cet outil à partir de cas d'études concrets basés sur des algorithmes du domaine des réseaux de télécommunications.

Ce document présente une technique de génération de test d'interopérabilité orienté 'objectif de test'. Cette technique fera l'objet d'une implémentation qui sera décrite dans une prochaine fourniture. Un prototype sera développé et des expérimentations sur des exemples de systèmes complexes seront aussi abordées. Il est envisagé d'effectuer des tests d'interopérabilité sur le protocole PGM (Pragmatic General Multicast) où la modélisation fait intervenir plusieurs composants (émetteur, récepteur, nœud) : après avoir testé la conformité de différents composants indépendamment, il s'agit de connaître leurs aptitudes à communiquer entre eux en satisfaisant les contraintes temporelles.

Annexe A. Résolution des systèmes d'inéquations à deux inconnues

Un système d'inéquations peut être représenté par un graphe où chaque nœud est une variable et chaque arc est une contrainte entre ses variables. Le système admet une solution si et seulement si le graphe ne contient pas de cycle négatif.

Définition 10. (Graphe des contraintes) Soit X un ensemble de variables et ϕ_x une conjonction de termes atomiques de la forme $x - x' \leq c$, avec $(x, x') \in X \cup \{0\}$ et $c \in Q$. Le graphe des contraintes associé à ϕ_x est un graphe orienté valué $G(\phi_x) = \{X \cup \{0\}, A\}$ avec:

- $X \cup \{0\}$ l'ensemble des sommets du graphe.
- $A \subset (\{X \cup \{0\} \times R \times (X \cup \{0\})\})$ l'ensemble des arcs du graphe. On note $x \xrightarrow{c} x'$ pour un arc de A.

$$x \xrightarrow{c} x' \in A \longleftrightarrow x - x' \leq c, \textit{est un terme de } \phi_x$$

Exemple 1 . Le système d'inéquations D_0 peut se mettre sous la forme de conjonction de termes de la forme $x - x' \leq c$, ce qui donne la forme D_1:

$$D_0 = \begin{cases} 3 \leq x \leq 6 \\ x - 3 \leq y \\ y \geq 5 \end{cases} \quad D_1 = \begin{cases} x - 0 \leq 6 \\ 0 - x \leq -3 \\ x - y \leq 3 \\ 0 - y \geq -5 \end{cases}$$

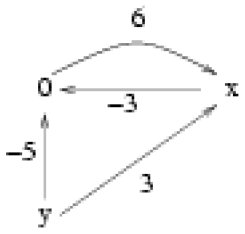

Figure 18 Graphe de contraintes G pour le système D_1

Aucun cycle négatif n'est présent dans le graphe G de la figure 4. Donc le système admet une solution. Au système D_1, on associe une matrice M . deux sommets du graphe.

$$M =$$

L'algorithme de FLOYD-WARSHALL, de recherche des plus courts chemins [26] permet à la fois de mettre un système sous forme normale et de tester si l'ensemble des contraintes admet une solution.