TGSE : Un outil générique pour le test

Ismail Berrada, Patrick Felix

TGSE : Un Outil Générique pour le Test *

Ismaïl Berrada — Patrick Félix

LaBRI - CNRS - UMR 5800 Université Bordeaux 1
351 cours de la libération 33405 Talence cedex, France.
{ berrada, felix}@labri.fr

* *Ce travail a été partiellement supporté par le projet RNRT Avérroes et le projet européen Marie Curie RTN TAROT (MCRTN 505121).*

RÉSUMÉ. *L'automatisation de la génération des tests devient une nécessité devant l'accroissement de la complexité des protocoles à tester. La simulation graphique de l'exécution d'un cas de test vient compléter l'étape de génération par une étape d'analyse des séquences générées.*
Dans cet article, nous proposons un modèle générique permettant de traiter différents types et architectures de test : le test de conformité, le d'interopérabilité, de composants, et le test dans le contexte. Ce modèle appelé 'systèmes communicants' (CS), supportant des contraintes temporelles et les données, est basé sur la définition d'une topologie de communication régissant les synchronisations possibles entre différentes entités communicantes. Nous montrons le caractère générique des CSs pour la spécification et la génération de test pour le test actif et passif. Ce modèle est mis en oeuvre dans l'outil TGSE (émulation, simulation et génération de test).

ABSTRACT. *This paper follows two main lines of research. The first line is related to the study of models for the description of systems. For this line, we introduce the model of Communicating Systems (CS), which defines a set of common resources, a set of entities, and a topology of communication. The second line concerns testing methodologies adapted to protocol testing. For this line, we give a formal definition of a generic algorithm of generation (GAG). We demonstrate that the CS model with a GAG supports various 1) test architectures, 2) test types: conformance, interoperability, embedded, component testing..., and 3) test approaches: passive and active testing. The paper presents also the main characteristics of the TGSE tool (Test Generation, Simulation, and Emulation). TGSE is composed of a test case generator based on the CS model and implementing a GAG, a graphic simulator of the execution of a sequence generated by TGSE and a real-time emulator of communicating specifications. In its current version, TGSE supports the passive and active testing of one or several components with data and temporal constraints.*

MOTS-CLÉS : *Systèmes communicants, Modélisation, Test de protocoles.*

KEYWORDS: *Communicating systems, Modeling, Protocol testing.*

1. Introduction

Le test est une étape importante dans le cycle de développement de tout système (logiciel, protocole de communication ou matériel). Cette étape consiste à soumettre le système considéré à toute une série d'épreuves (expériences) pour voir si son comportement est bien celui attendu.

Le contexte actuel du test offre une multitude de types de test : test de conformité, dans le cas d'une seule entité, test d'interopérabilité, dans le cas de plusieurs entités communicantes, test dans le contexte, dans le cas d'une entité communicante dans un environnement. A cette multitude de types, s'ajoute une grande diversité dans les modèles de base considérés : les systèmes de transitions étiquetés (LTS) [BRI 88], les systèmes de transitions à entrée/sortie (IOLTS) [TRE 96, FER 97] ou temporisés (TLTS) [BRI 04], les machines à états finies (FSM) [SEO 03], ou étendues (EFSM)[HIG 99], les automates à entrée/sortie temporisés (TIOA)[SPR 01]... Deux approches majeures sont alors utilisées pour tester : le test actif [CAR 00, CLA 97, DSS 98, HIG 99, KOU 00, SPR 01, BER 04, LAR 03, TRI 04, DSS 03, KOU 03], la dérivation se fait à partir d'une spécification (exprimée sous la forme d'une ou plusieurs entités communicantes) et le test passif [CAV 04] qui vérifie la validité d'une trace d'exécution d'une implémentation (la trace est une exécution valide).

De notre point de vue, cette diversité dans les types, les modèles et les approches traduit la spécificité des besoins. En effet, les différents types de test sont une conséquence de la composition des systèmes à tester et des architectures de test : la conformité considère seulement une entité tandis que l'interopérabilité et le test dans le contexte considèrent plusieurs entités communicantes interagissant selon une architecture de test donnée. La multitude des modèles de base se justifie par des besoins différents de spécification : les systèmes considérés qui ont des comportements événementiels, peuvent manipuler des données et être soumises à des contraintes temporelles. Que l'on soit dans le cadre du test actif où la dérivation d'un test se fait à partir de la spécification, ou bien dans le cadre du test passif où l'absence d'observabilité et de contrôlabilité ne permet qu'une validation des traces de l'implémentation, nous sommes confrontés au même problème. Il s'agit d'un problème d'accessibilité d'un état ou d'une transition au sein d'un graphe.

Motivations. Dans cet article, nous traitons le test des protocoles de communication temps-réel dans l'optique d'une approche permettant d'appréhender les différents types de test au sein d'un même outil. La première motivation est la modélisation des communications et le partage des données entre différentes entités d'un système. Les algèbres de processus se présentent alors comme un bon modèle, très expressive. Cependant, l'utilisation des algèbres de processus présente une difficulté de définition d'algorithmes applicables dans un outil de génération de test. Nous introduisons alors le modèle générique des systèmes communicants inspiré de [MAD 04]. Ce modèle définit un ensemble d'entités (composantes) communicantes, des ressources communes (données et paramètres partagés) aux différentes composantes et une topologie

de communication qui explicite les différentes synchronisations possibles dans un état global du système.

La deuxième motivation de cet article est réduire l'écart qui existe entre les différents types de test. Notre but est de montrer qu'il est possible d'appliquer la même stratégie de génération pour différents types de test. Nous définirons alors un algorithme générique de génération. Nous montrerons que le modèle des systèmes communicants muni d'un algorithme générique de génération supporte les différents types, approches et architectures de test.

Finalement, en présentant l'outil TGSE (Émulation, Simulation et Génération de Test), basé sur le modèle des systèmes communicants, nous montrons la faisabilité de notre approche. Nous présenterons les différentes techniques de génération implémentées dans TGSE.

Dans la suite de cet article, nous introduisons dans la section 2 le modèle des systèmes communicants et l'algorithme générique de génération dans le section 3. Puis, dans la section 4, nous montrons le caractère générique de ce modèle. Ensuite nous donnons dans la section 5 quelques éléments sur la mise en oeuvre de ce modèle au sein de l'outil TGSE. La section 6 est consacrée à une étude de cas sur le protocole CSMA/CD. La dernière section sera réservée à la conclusion et aux perspectives.

2. Modèle des Systèmes Communicants

Le comportement d'un protocole de communication peut être décrit par un modèle formel comme les systèmes communicants. Un système communicant (CS) définit un ensemble de ressources communes, un ensemble d'entités et une topologie de communication. Les entités représentent des processus. Elles sont modélisées par des automates temporisés étendus à entrée/sortie (ETIOAs). La topologie de communication décrit les différentes synchronisations possibles entre les entités du CS. Les ressources communes représentent les différentes données partagées par les entités du CS. Par la suite, $\mathbb{R}$ dénotera l'ensemble des réels et $\mathbb{R}^+$ dénotera l'ensemble des réels positifs.

2.1. *Automate temporisé étendu à entrée/sortie*

2.1.1. *Horloges et Contraintes.*

Une horloge est une variable qui mémorise le passage du temps. Elle peut être réinitialisée et inspectée à tout moment pour calculer le temps écoulé depuis la dernière réinitialisassion. Dans le modèle d'Alur-Dill [ALU 94], les horloges évoluent à la même vitesse, elles sont évaluées dans $\mathbb{R}^+$ et la seule réinitialisassion permise est de la forme $x := 0$. Pour un ensemble C d'horloges, un ensemble P de paramètres et un ensemble V de variables, l'ensemble des contraintes d'horloge $\Phi(C, P, V)$ est défini par la grammaire suivante :

$$\phi := \phi_1 \; |\phi_2| \; \phi_1 \wedge \phi_2, \; \phi_1 := x \leq f(P,V), \; \phi_2 := f(P,V) \leq x$$

avec x une horloge de C et $f(P,V)$ une expression linéaire de P et V.

Définition 1 (ETIOA) *Un automate temporisé étendu à entrée/sortie (ETIOA) est un 10-uplet $M = (S, L, C, P, V, V_0, Pred, Ass, s_0, T)$ avec :*

– *S est un ensemble fini des états.*

– *s_0 est l'état initial.*

– *L est un alphabet fini d'actions, $L = L_i \cup L_o \cup I$.*

– *C est un ensemble fini d'horloges incluant une horloge globale h.*

– *P est un ensemble fini de paramètres.*

– *V est un ensemble fini de variables.*

– *V_0 est un ensemble de valeurs initiales pour les variables de V.*

– *$Pred = \Phi(C, P, V) \cup \tilde{P}[P, V]$, $\tilde{P}[P, V]$ est un ensemble d'inégalités linéaires sur V et P.*

– *$Ass = \{x := 0 \mid x \in C\} \cup \{v := f(P, V) \mid v \in V\}$ est un ensemble de mises à jour sur les horloges et les variables.*

– *$T \subseteq S \times L \times Pred \times Ass \times S$ est un ensemble de transitions.*

L'alphabet L d'un ETIOA est divisé en trois parties : L_i l'alphabet des symboles d'entrée, L_o l'alphabet des symboles de sortie et I l'alphabet des actions internes. Pour $t \in T, t = (s, a, pred, ass, s')$ est la transition de l'état s à l'état s' sur l'occurrence du symbole a. $pred \subseteq Pred$ est une contrainte sur C et V et $ass \subseteq Ass$ est l'ensemble des mises à jour sur C et V.

Exemple 2 *Un exemple d'ETIOA est donné dans la figure suivante.*

– *$S = \{s_0, s_1, s_2, s_3\}$ et s_0 l'état initial.*

– *$L = \{!a, ?b, !c, ?d\}$, $C = \{x, y\}$, $P = \{\beta, \lambda\}$, $V = \{v1\}$ et $V_0 = \{\beta\}$.*

– *$Pred = \{y \geq \lambda, x \leq 1, v1 \leq 4\}$.*

– *$Ass = \{x := 0, y := 0, v1 := v1 + 1\}$.*

– *La transition t de source s_2 et de destination s_3 est : $t = (s_2, !c, \{x \leq 1\}, \{v1 := v1 + 1\}, s_3)$.*

REMARQUE. —

Pour un ETIOA $M = (S, L, C, P, V, V_0, Pred, Ass, s_0, T)$,

– Lorsque $P = \emptyset$ et $V = \emptyset$, on retrouve alors la définition usuelle d'un automate temporisé (TIOA) et M sera noté simplement (S, L, C, s_0, T).

– Lorsque $C = \emptyset$, $P = \emptyset$ et $V = \emptyset$, on retrouve alors la définition usuelle d'un automate (IOA) et M sera noté simplement $M = (S, L, s_0, T)$.

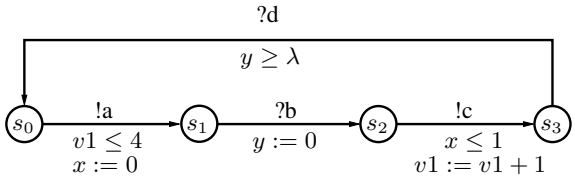


Figure 1. *Exemple d'un ETIOA.*

2.2. *Topologies de communication et Systèmes Communicants*

Une topologie de communication **Top** d'un ensemble de processus est un modèle de synchronisation des différents processus. Elle décrit les configurations dynamiques des processus et les synchronisations possibles dans une configuration donnée. La définition de **Top** est inspirée de celle d'Arnold [ARN 94, MAD 04]. Elle définit un ensemble d'actions globales, un ensemble d'ensembles d'actions et un Traducteur sous forme d'automate.

Définition 3 (Topologie) *Une topologie de communication Top d'un ensemble de n processus est un 3-uplet (G, I, Tr), avec G un ensemble fi ni d'actions globales, $I = \{I_i\}_{1 \leq i \leq n}$ un ensemble fi ni d'ensembles d'actions et Tr (traducteur) un automate $Tr = (S_{tr}, L_{tr}, s_{0tr}, \rightarrow_{tr})$ tel que les éléments de L_{tr} sont des vecteurs $\overrightarrow{v}$ à $n+1$ éléments et $\forall \overrightarrow{v} \in L_{tr}$, $\overrightarrow{v} = (a_g, a_1, ..., a_n)$ avec $a_g \in G$ et $\forall i \in [1, n]$, $a_i \in I_i \cup \{idle\}$.*

Un vecteur $\overrightarrow{v} = (a_g, a_1, ..., a_n)$ de L_{tr} décrit l'action a_i que le processus i, $i \in [1, n]$, doit effectuer. La synchronisation des différentes actions donne lieu à l'action globale a_g. Lorsque un vecteur $\overrightarrow{v} = (a_g, idle, ..., a_i, ..., idle)$ définit une seule action, le processus i exécute seul l'action a_i et change d'état. Pour une topologie $Top = (G, I, Tr)$, lorsque le nombre d'états du traducteur Tr est égale à 1, Top est appelée topologie **statique** et dans ce cas les éléments de L_{tr} sont appelés des *vecteurs de synchronisation*.

Une topologie offre la possibilité de modéliser des communications entre un, deux ou plusieurs processus : unicast, multiscast et broadcast. Elle peut être utilisée, dans certains cas, comme une sorte de contrôleur sur les actions permises par les différents processus dans une configuration donnée du système global.

Définition 4 (Système Communicant) *Un système communicant CS est un 5-uplet $(SP, SV, SV_0, (M_i)_{1 \leq i \leq n}, Top)$ avec :*

- *SP est un ensemble de paramètres partagés.*
- *SV est un ensemble de variables partagées.*
- *SV_0 est un ensemble de valeurs initiales pour les variables de SV.*

– $Top = (G, \{I_i\}_{1 \leq i \leq n}, Tr)$ *est une topologie.*

– $M_i = (S_i, L_i, C_i, P_i, V_i, V_{0i}, Pred_i, Ass_i, s_{0i}, \rightarrow_i)$ *est un ETIOA tel que :* $\forall i \in [1, n]$, $I_i \subseteq L_i$.

Un système communicant (CS) définit un ensemble de ressources communes, un ensemble d'entités et une topologie de communication. Les entités représentent des processus. Elles sont modélisées par des ETIOAs. La topologie de communication décrit les différentes synchronisations possibles entre les entités du CS. Nous avons supposé dans la définition des entités que $\forall i \in [1, n]$, $I_i \subseteq L_i$. Ceci permet d'avoir des topologies partielles qui ne définissent que les synchronisations permises (dans la section suivante, nous donnerons des exemples de telles topologies). Les ressources communes représentent les différentes données partagées du CS. Nous nous restreignons à des données de types variables et paramètres. Les paramètres (resp. les variables) peuvent être lus (resp. lues et modifiées) par les entités du CS [1]. C'est une communication à mémoire partagée. Ceci ne restreint pas le modèle vu que la communication par messages peut être modélisée par des paramètres et des variables partagées. La sémantique d'un CS est définie en terme d'un ETIOA. Pour simplifier, nous supposons que les noms des différents paramètres et variables des différentes entités du système sont différents et différents de ceux du CS.

Définition 5 (Sémantique) *La sémantique d'un système communicant* $S = (SP, SV, SV_0, (M_i)_{1 \leq i \leq n}, Top)$*, avec* $M_i = (S_i, L_i, C_i, P_i, V_i, V_{0i}, Pred_i, Ass_i, s_{0i}, \rightarrow_i)$ *et* $Top = (G, I, (S_{tr}, L_{tr}, s_{0tr}, \rightarrow_{tr}))$*, est défi nie par un ETIOA* $\zeta(S) = (S, L, C, P, V, V_0, Pred, Ass, s_0, \rightarrow)$*, tel que :*

– $S = \{s = (s_{tr}, s_1, ..., s_n) \mid s_{tr} \in S_{tr}, \forall i \in [1, n], s_i \in S_i\}$

– $s_0 = (s_{otr}, s_{01}, ..., s_{0n})$.

– $L = G$, $C = C_1 \cup ... \cup C_n$, $P = SP \cup P_1 \cup ... \cup P_n$.

– $V = VP \cup V_1 \cup ... \cup V_n$, $V_0 = VP_0 \cup V_{01} \cup ... \cup V_{0n}$.

– $Pred = Pred_1 \cup ... \cup Pred_n$, $Ass = Ass_1 \cup ... \cup Ass_n$.

– $\rightarrow = _\delta \{(s_{tr}, s_1, ..., s_n) \xrightarrow{a,pred,ass} (s'_{tr}, s'_1, ..., s'_n) \mid \exists \overrightarrow{v} = (a, a_1, ..., a_n) \in L_{tr}, s_{tr} \xrightarrow{\overrightarrow{v}}_{tr} s'_{tr}, \forall i \in [1, n], (((a_i = idle) \wedge (s_i = s'_i)) \parallel ((a_i \neq idle) \wedge (s_i \xrightarrow{a_i, pred_i, ass_i}_i s'_i))), pred = pred_1 \wedge ... \wedge pred_n, ass = ass_1 \wedge ... \wedge ass_n\}$.

La sémantique d'un CS S est définie en terme de l'ETIOA $\zeta(S)$. L'alphabet de $\zeta(S)$ est les actions globales G de Top. Un état de $\zeta(S)$ est constitué d'un état de Top et des états de $(M_i)_{i \in [1,n]}$. Une transition $(s_{tr}, s_1, ..., s_n) \xrightarrow{a,pred,ass} (s'_{tr}, s'_1, ..., s'_n)$ de $\zeta(S)$ est conditionnée par l'existence d'une transition de Top de s_{tr} à s'_{tr} sur un vecteur ayant comme action globale a.

1. Les paramètres et les variables partagées peuvent apparaître dans la définition d'une transition d'une entité.

Ainsi, la sémantique d'un CS autorise la possibilité de synchronisation avec d'autres CSs, ce qui permet une définition hiérarchique des CSs et rend le modèle des CSs un modèle générique pour la modélisation des spécifications.

3. Méthodologie : Algorithme Générique de Génération.

La majorité des algorithmes de génération de test sont basés sur une recherche en profondeur d'un état ou d'une transition cible dans le graphe d'accessibilité. Il est alors possible de définir un algorithme générique de génération pour les différents types de test. Dans cette partie, nous montrons comment définir un tel algorithme.

Définition 6 *Un système communicant sous test (CSUT) est un système communicant* $S = (SP, SV, SV_0, (M_i)_{1 \leq i \leq n}, Top)$*, tel qu'il existe au moins une entité* M_i*,* $i \in [1, n]$*, défi nissant un ou plusieurs états étiquetés par* $ACCEPT$*.*

Les états d'un CSUT étiquetés par $ACCEPT$ définissent les comportements cibles à tester. Notre définition de CSUT ne considère que des états étiquetés par $ACCEPT$, mais il est possible de définir des transitions étiquetées par $ACCEPT$. Ce dernier cas n'est pas traité dans cet article, mais l'approche reste la même. Notons par $\mathcal{CSUT}$ l'ensemble de tous les CSUTs.

Définition 7 *Pour un* $S \in \mathcal{CSUT}$*, un état* $s = (s_{tr}, s_1, ..., s_n)$ *de* $\zeta(S)$ *et* $\rho = t_0...t_n$ *une suite de transitions dans* $\zeta(S)$ *de l'état initial :*

– s *est dit un état acceptable de* $\zeta(S)$ *s'il existe* $i \in [1, n]$ *tel que* s_i *est un état étiqueté par* $ACCEPT$*.*

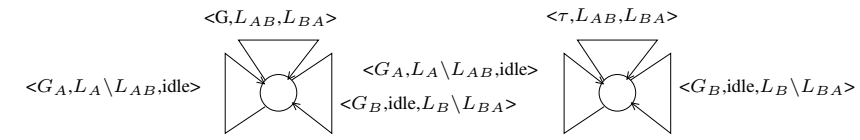
– ρ *est dit un chemin acceptable de* $\zeta(S)$*, si*

1) ρ *est un chemin exécutable.*

2) L'état destination de la transition t_n *est un état acceptable de* $\zeta(S)$*.*

Un état s de l'ETIOA $\zeta(S)$, sémantique de S, est un état acceptable de $\zeta(S)$ si l'un des états des entités qui le compose est un état étiqueté par $ACCEPT$. Un chemin $\rho = t_0...t_n$ dans $\zeta(S)$ de l'état initial est un chemin acceptable de $\zeta(S)$ si 1) l'état s_n de la transition $t_n = (s_{n-1}, a, pred, ass, s_n)$ est un état acceptable de $\zeta(S)$ et 2) ρ est un chemin exécutable (faisable), c'est à dire, les différentes contraintes sur les transitions du ρ sont toutes vérifiées. L'exécutabilité d'un chemin est traité dans [CAV 04, CAL 04]. L'activité de test consiste alors à générer tous les chemins acceptables.

Définition 8 *Un algorithme générique de génération (GGA) pour* $\mathcal{CSUT}$ *est un algorithme qui calcule, pour tout* $S \in \mathcal{CSUT}$*, tous les chemins acceptables (s'ils existent) de* $\zeta(S)$*.*



(a) Topologie TopS (b) Topologie TopS'

Figure 2. *Différentes topologies*

Un algorithme gga est GGA pour $\mathcal{CSUT}$, si gga appliqué à $\zeta(S)$ retourne un ensemble $PATH(S)$ contenant tous les chemins acceptables de $\zeta(S)$. Des exemples d'algorithmes GGA sont présentés dans [CAV 04, ZAI 99, CAL 04]. Signalons que l'algorithme Hit-or-Jump [ZAI 99] ne traite pas les aspects temporels des systèmes et considère des transitions $ACCEPT$ au lieu des états $ACCEPT$.

Finalement, un algorithme gga ne dépend pas d'un CSUT, il est exhaustif dans le sens que tout chemin acceptable est retourné par gga et en fin, le choix entre états ou transitions acceptables dépend du critère de couverture adopté.

4. CS : Un Modèle Générique pour le Test

Un modèle de spécification d'un protocole de communication doit permettre la possibilité de modéliser le comportement propre du protocole et son comportement vis-à-vis de son environnement. Le but de cette section est de montrer le caractère générique des CSs pour la spécification et la génération de test.

Pour deux ensembles L_1 et L_2, notons par $L_1 \setminus L_2$ l'ensemble des éléments propres de L_1 : $L_1 \setminus L_2 = \{a \,|\, a \in L_1 \wedge a \notin L_2\}$. Dans le reste de cette section, nous considérons deux spécifications communicantes S_A et S_B, partageant les paramètres SP et les variables SV, SV_0 étant les valeurs initiales des variables. Nous modélisons S_A (resp. S_B) par l'ETIOA $A = (S_A, L_A, C_A, P_A, V_A, V_0, Pred_A, Ass_A$ $, s_0, T_A)$ (resp. $B = (S_B, L_B, C_B, P_B, V_B, V_0', Pred_B, Ass_B, s_0', T_B)$). L_{AB} (resp. L_{BA}) dénotera l'ensemble des événements de L_A (resp. L_B) qui synchronisent avec un événement de L_B (resp. L_A). Par exemple, si $L_A = \{?begin, ?end, !busy\}$ et $L_B = \{!end, ?busy, !CD\}$ alors $L_{AB} = \{?end, !busy\}$ et $L_{BA} = \{!end, ?busy\}$. Pour simplifier, nous supposons que pour tout $a \in L_{AB}$ il existe un unique $b \in L_{BA}$ qui synchronise avec a.

4.1. *CS comme Modèle de Spécification*

Le but de cette partie est de montrer comment décrire une spécification dans le modèle CS.

Considérons la spécification S composée des spécifications S_A et S_B. Une modélisation en CS de S est la suivante : $CS_1 =< SP, SV, SV_0, < A, B >, TopS >$, avec $TopS$ l'automate de la Figure 2 (a). $TopS$ est une topologie statique. Le vecteur $< G, L_{AB}, L_{BA} >$ est pour dénoter l'ensemble des vecteurs de la forme $< g_{ab}, a, b >$ tel que $a \in L_{AB}$ synchronise avec $b \in L_{BA}$ et leur synchronisation donne lieu à une action globale observable g_{ab}. Un exemple de g_{ab} peut être a si a est une émission et b si b est une émission (l'action visible d'une émission et une réception est une émission). De même, le vecteur $< G_A, L_A \backslash L_{AB}, idle >$ est pour dénoter l'ensemble des vecteurs de la forme $< g_a, a, idle >$ tel que $a \in L_A \backslash L_{AB}$. Dans le vecteur $< g_a, a, idle >$, l'ETIOA A performe l'action a donnant lieu à une action globale observable g_a. Quant à L'ETIOA B, il reste dans le même état ($idle$). L'ensemble G_A correspond, en général, à l'ensemble $L_A \backslash L_{AB}$. Finalement, $TopS$ permet d'appliquer chaque vecteur de synchronisation (si c'est possible) dans un état global de S.

Si on considère que les synchronisations des événements L_{AB} avec les événements de L_{BA} sont inobservables (comme c'est le cas pour l'architecture de test boîte noire) alors la modélisation de S en CS est $CS_2 =< SP, SV, SV_0, < A, B > , TopS' >$, avec $TopS'$ l'automate de la Figure 2 (b). Le vecteur $< \tau, a, b >$ de $< \tau, L_{AB}, L_{BA} >$, considère que la synchronisation de a avec b donne lieu à une action interne τ. D'une façon générale, on peut modéliser la synchronisation en actions internes d'une partie des événements de synchronisations (comme c'est le cas pour une architecture de test).

Ainsi, nous avons décrit une spécification à deux composantes en CS en tenant compte de l'observabilité des actions du système. La même approche s'applique à une spécification à plusieurs composantes.

4.2. *CS comme Modèle pour la Génération de Test*

Pour le test des protocoles, deux approches majeures ont été utilisées : le test actif et le test passif. Dans le test actif, la dérivation se fait à partir de la spécification. Cette dérivation peut ne concerner qu'une partie de la spécification dans le but de limiter l'explosion combinatoire des états, comme c'est le cas pour la technique de génération orientée objectif de test. Ce genre de test peut considérer une ou plusieurs entités communicantes [CAR 00, CLA 97, DSS 98, HIG 99, KOU 00, MAN 95, SPR 01, BER 04, LAR 03, TRI 04]. En revanche, le test passif considère des traces d'exécution d'une implémentation, qui peuvent contenir des valeurs pour les variables et les horloges, et vérifie leur validité par rapport à la spécification. Dans les travaux relatifs au test passif [CAV 04], les auteurs considèrent une seule entité.

Pour simplifier, appelons *test à une composante* le test d'une seule spécification (test de conformité) et *test à plusieurs composantes* le test de plusieurs spécifications (test d'interopérabilité, test dans le contexte, test de composants,...). Dans le reste de cette section, nous considérons que gga est un GGA. Le but de cette partie est de montrer que l'activité de test revient à modéliser un CS, en reportant les particularités

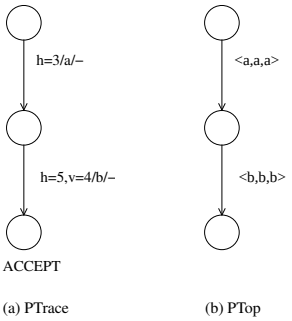


Figure 3. *Test passif*

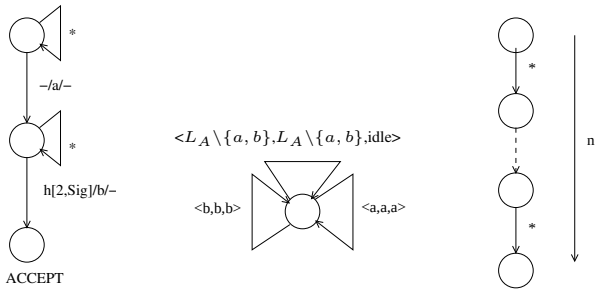
des tests dans la topologie de communication, et à appliquer gga pour valider une trace (test passif) ou générer des traces (test actif).

4.2.1. *Test passif.*

Supposons que la trace modélisée par l'ETIOA $PTrace$ de la Figure 3 (a) est une trace d'une implémentation de la spécification S_A. Cette trace exprime que l'implémentation a effectuée l'action $a \in L_A$ à l'instant 3, puis l'action $b \in L_B$ à l'instant 5 telle que la valeur de la variable partagée $v \in SV$ vaut 4. Vérifier la validité de cette trace consiste à modéliser un CS $CS_3 =< SP, SV, SV_0, < A, PTrace >, PTop >$, avec $PTrace$ l'ETIOA de la Figure 3 (a) et $PTop$ l'automate de la Figure 3 (b).

La topologie $PTop$ est partielle et ne définit que les synchronisations sur les événements de $PTrace$. Nous avons étiqueté l'état après l'action b de $PTrace$ par $ACCEPT$. Ceci a pour but de rendre CS_3 un CSUT et de pouvoir appliquer gga. Dès lors, gga permet de décider si $PTrace$ est une trace valide de A : si gga retourne un ensemble vide ($PATH(CS_3) = \emptyset$) alors $PTrace$ n'est pas valide, sinon c'est une trace valide.
REMARQUE. —

1) D'une façon générale, la construction de $PTop$ dépend fortement de $PTrace$, elle ne doit définir que les synchronisations sur les événements de $PTrace$ et dans le même ordre.
2) La trace $PTrace$ est considéré comme une entité de CS_3 sans aucune distinction par rapport aux autres entités. Ceci permet d'étendre la forme des traces que le test passif considère à des traces sous forme d'ETIOAs définissant des états $ACCEPT$.
3) L'exemple du test passif de la spécification S_A est un test à une composante, mais l'approche reste la même dans le cas du test passif à plusieurs composantes. La difficulté, dans ce cas, est de réordonner des différentes traces des composantes pour en construire une trace globale. Nous pensons que les techniques d'estampillage, et



(a) Objectif TP1 (b) Topologie PTop1 (c) Topologie TPop1'

Figure 4. *Test actif à une composante (1)*

spécialement la technique présentée dans [JAR 99], peuvent être utilisées. Ce sujet dépasse le cadre de cet article et mérite plus d'investigation.

4.2.2. *Test actif.*

Pour un CSUT S, les chemins générés par l'algorithme gga peuvent être utilisés pour dériver des cas de test qui couvrent par exemple tous les états de S. Ceci revient à définir tous les états de S comme des états acceptables. Nous nous intéressons alors qu'à la technique de dérivation orientée objectif de test.

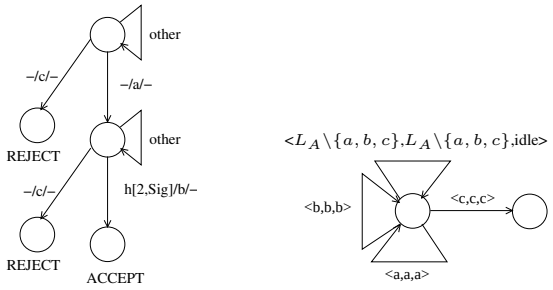
Définition 9 *Un objectif de test (OT) est un ETIOA* $(S, L, C, P, V, V_0, Pred, Ass, s_0, \rightarrow)$ *ayant deux ensembles d'états* $ACCEPT$ *et* $REJECT$ *caractérisant les comportements désirables à tester.*

Un OT définit une partie de la spécification à tester. Nous modélisons un OT par un ETIOA équipé de deux ensembles d'états $ACCEPT$ et $REJECT$. $TP1$ de la Figure 4 (a) est un exemple d'OT pour la spécification S_A. $TP1$ teste que l'implémentation peut effectuer l'action a puis l'action b entre $[2, Sig]$ selon l'horloge globale h ($Sig \in SP$ est un paramètre partagé de S_A). La boucle étoile sur les deux premiers états de $TP1$ est pour dénoter tout l'alphabet L_A des événements de S_A. Nous assumons ici que $a, b, c \in L_A$.

Test à une composante.
On s'intéresse au test actif de la spécification S_A avec OT $TP1$. Une modélisation possible de ce test en CS est $CS_4 =< SP, SV, SV_0, < A, TP1 >, TPop1 >$, avec $TPop1$ la topologie de la Figure 4 (b). Les vecteurs $< L_A \setminus \{a, b\}, L_A \setminus \{a, b\}, idle >$ est pour dénoter un évolution séparée de la spécification sur des événements autres que a et b. Une application directe de l'algorithme gga permet alors de trouver les comportements vérifiant l'OT $TP1$.

Signalons qu'à partir de ce même OT $TP1$, on peut avoir plusieurs modélisations en CS, mettant en jeu des topologies différentes. En effet, ceci est possible dans la défi-

nition de la topologie qui procure plus d'expressivité sur le comportement attendu par OT. Un exemple typique de l'expressivité de la topologie est le suivant : étant donné que les chemins générés par gga pour CS_4 sont de taille (nombre de transitions) arbitraire, nous désirons générer que les chemins de taille inférieure à n. Ce souhait ne peut pas être formulé par un OT qui ne permet pas de compter les occurrences des événements. Considérons maintenant le CS $CS_5 =< SP, SV, SV_0, < A, TP1 >, TPop1' >$, avec $TPop1'$ la topologie de la Figure 4 (c). L'étiquette étoile dans $TPop1'$ dénote les vecteurs $< a, a, a >, < b, b, b >$, et $< L_A \backslash \{a, b\}, L_A \backslash \{a, b\}, idle >$ (une transition étoile est l'ensemble de transitions sur ces vecteurs de synchronisation). Appliquer gga à CS_5 générera des chemins vérifiant $TP1$ et de taille inférieure à n vu qu'on dépasse pas n états dans $TPop1'$.



(a) Objectif TP2 (b) Topologie PTop2

Figure 5. *Test actif à une composante (2)*

Pour conclure la partie test à une composante, prenons l'OT $TP2$ de la Figure 5 (a). $TP2$ teste les mêmes fonctionnalités que $TP1$ mais interdit l'apparition de c dans les deux premiers états. L'étiquette *other* dans $TP2$ dénote les événements $L_A \backslash \{c\}$. La définition des états $REJECT$ est en effet une manière d'interdire la synchronisation sur un ensemble d'événements. Cette interdiction peut être formulée au niveau de la topologie au lieu de l'objectif de test. Dans ce cas, on peut utiliser $TP1$ au lieu de $TP2$. En effet, le test actif de S_A et l'OT $TP2$ peut être modélisé par le CS $CS_6 =< SP, SV, SV_0, < A, TP1 >, TPop2 >$, avec $TPop2$ la topologie de la Figure 5 (b). Dans $TPop2$, lorsque la synchronisation sur c se produit, le système communicant évolue dans un état bloquant et donc lors de l'application de gga à CS_6, gga est obligé de dépiler cette synchronisation. Finalement, l'algorithme gga appliqué à CS_6 donne les chemins vérifiant $TP2$.

Test à plusieurs composantes

On s'intéresse au test actif de la spécification S composée des deux spécifications S_A et S_B et l'OT $TP1$ de la Figure 6 (a). Pour simplifier, nous assumons ici que $a \in L_A$, $a \notin L_{AB}$ et $b \in L_{AB} \cap L_{BA}$. Une modélisation en CS de ce test est $CS_7 =< SP, SV, SV_0, < A, B, TP1 >, Top >$, avec Top la topologie de la Figure 6 (b). Le vecteur $< b, b, b, b >$ considère que A, B et $TP1$ se synchronisent sur b. Le

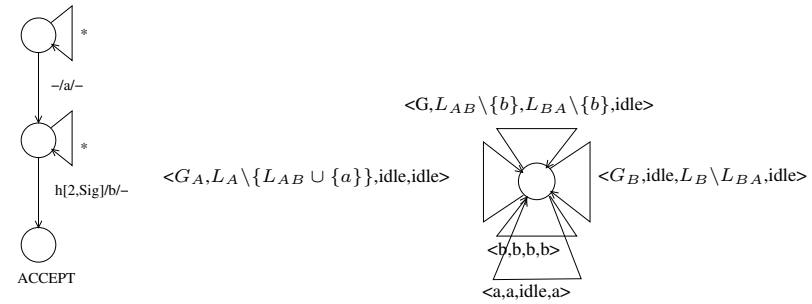


Figure 6. *Test actif à plusieurs composantes*

vecteur $< a, a, idle, a >$ considère que seulement A et $TP1$ se synchronisent sur a. Dès lors, l'application de gga permet de générer les chemins qui vérifient $TP1$.

Finalement, dans le cas du test actif orienté OT d'une spécification définissant des composantes symétriques, à l'exemple de la spécification TCP et la spécification de CSMA/CD à plusieurs émetteurs, l'OT peut contenir des événements (a_j) qui ne sont pas des événements de synchronisations entre composantes mais apparaissent dans plusieurs composantes. Pour spécifier qu'un événement de OT correspond à une composante spécifique i, deux solutions sont envisageables sans utiliser le modèle CS : 1) dupliquer les ETIOAs où les événements (a_j) apparaissent puis indexer les différents événements par l'indice de chaque composante ou 2) indexer les événements (a_j) par l'indice de la composante i $(a_j(i))$ et définir un algorithme de génération tenant compte des indices des événements dans OT. Dans le modèle CS, ce problème ne se pose pas vu que cette correspondance peut être formulée dans les vecteurs de la topologie. Ceci évite d'indexer et de dupliquer des composantes.

Ainsi, nous avons montré que notre modèle CS est un modèle générique dans le sens qu'il permet la modélisation des différents types de communications et l'incorporation de l'architecture du test dans la description et un modèle unificateur des différents types de test, offrant plus d'expressivité pour tester les protocoles de communications.

5. TGSE : Un Outil de Générique de Génération

Dans cette section, nous décrivons l'implémentation des CSs dans l'outil TGSE (Test génération, simulation and emulation) développé au LaBRI. TGSE [CAL 04] est un ensemble de logiciels regroupant les différentes activités du test. Il est composé d'un générateur de séquences de test basé sur le modèle CS, d'un simulateur à travers la plate-forme Calife permettant de simuler graphiquement l'exécution d'une séquence générée par TGSE et un émulateur temps réel des différentes spécifications

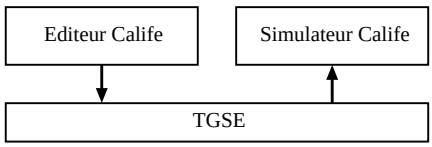
communicantes. Le manque d'espace nous oblige à ne pas inclure les détails d'implémentation de TGSE, et le lecteur intéressé peut se référer au rapport interne [CAL 04]. Nous ne présentons ici que la génération de test dans TGSE .

5.1. *Interface TGSE.*

Le projet RNRT Calife et son successeur Averroès est un projet académique et industriel regroupant un ensemble de partenaires (France Telecom R&D, CRIL Technologie, Labri, LSV, Loria, LRI). Le but de ce projet est de définir une plate-forme générique (Open Source) permettant d'interfacer des outils de vérifications et de générations de test. Dans le cadre de ce projet, la participation du LaBRI est d'intégrer l'outil TGSE dans la plate-forme. La plate-forme Calife comporte un éditeur et un simulateur. L'éditeur fournie un interface agréable et facile à manipuler les différents types d'automates (Temoporisés, hybrides, étendus,...). Son simulateur permet de simuler graphiquement l'exécution d'un ensemble d'automates.

L'entrée de TGSE est une description du CS sous une syntaxe simple permettant la définition des différentes entités du CS. Dans un soucis de réutilisation des composantes, chaque ETIOA du CS est défini dans un fichier séparé. Un fichier système décrit le chemin d'accès à chaque composante, ainsi que la topologie de communication. Dans sa version actuelle, TGSE implémente une topologie statique, mais supporte le test actif et passif à une ou plusieurs composantes. La sortie du TGSE est un fichier XML selon une DTD Calife définissant une séquence de test.

TGSE peut être utilisé aussi en mode graphique à travers Calife. Dans ce cas, la saisie des spécifications est à travers l'éditeur Calife qui permet la génération automatiquement les vecteurs de synchronisation, ceci en offrant le choix d'une synchronisation par rendez-vous, broadcast, la synchronisation binaire d'Uppaal, ou par labels identiques. L'appel a TGSE se fait à travers l'interface graphique et Calife génère les fichiers d'entrée pour TGSE. TGSE produit un cas de test qui sera simulé sous Calife. Le schéma suivant représente les communications entre Calife TGSE.



5.2. *Méthode de Génération*

TGSE implémente un algorithme gga de recherche en profondeur à la volée de l'automate sémantique du CS. La recherche est paramétrée par le nombre maximal d'apparition d'une transition dans la séquence générée. Le choix des transitions, des vecteurs de synchronisations et des automates qui performent les actions est paramétré par des variables pour chaque donnée, dont les valeurs sont RANDOM, pour un

choix aléatoire, et FIFO pour un respect de l'ordre d'apparition dans le définition du CS. L'algorithme gga calcule un chemin de l'état initial qui se termine sur un état $ACCEPT$. Durant la recherche en profondeur, plusieurs calculs sont effectués :

Étape 1 : Calcul des successeurs.

A partir d'un état courant s dans l'automate sémantique du CS, les vecteurs de synchronisations sont évalués d'une façon paramétrable pour calculer un état successeur s'. Une API $UpdateContext()$ est alors appelée.

*Étape 2 : Trace Symbolique [CAL 04] (***Module contexte***).*

L'API $UpdateContext()$ calcule la trace symbolique des nouvelles transitions tirées. Ce module définit aussi l'API $UpdateVarContext()$ qui permet la mise à jour du contexte lors du franchissement des transitions (les affectations des transitions).

*Étape 3 : Résolution des contraintes (***Module Contraintes***).*

Une fois la trace symbolique est calculée, une API $getSolution()$ est appelée. Dans le cas d'une trace paramétrée, $getSolution()$ appelle la procédure du **Module Paramètres** $getLpSolution()$ qui interagit avec l'outil de programmation linéaire $lp\_solve\ v4$ pour instancier les paramètres. Dans le cas contraire, un appel à l'API $getTimeExecution()$ du **Module Horloges** est effectué pour le calcul des exécutions en temps minimal et maximal [BER 04, CAL 04].

*Étape 4 : Calcul d'un cas de Test (***Module Trace***).*

Si lors du parcours, un état $ACCEPT$ est rencontré, la recherche se termine par un appel à l'API $writeTrace()$ qui décore le chemin obtenu par les différents verdicts. La sortie est en XML suivant une DTD Calife.

Signalons finalement qu'en cas ou aucun état $ACCEPT$ n'est rencontré, l'algorithme dfs est relancé automatiquement pour une nouvelle tentative (le lancement est paramétrable). De plus, il est possible de générer un cas de test minimal en nombre de transitions pour un nombre de tentatives donné.

6. Application : CSMA/CD

Le protocole CSMA/CD (Figure 7) se compose d'un bus (médium de communication) et d'un ou plusieurs émetteurs (stations émettrices). Nous ne modélisons pas ici les récepteurs. Lorsque deux ou plusieurs émetteurs envoient simultanément des données sur le bus, ce dernier détecte les collisions des trames et envoie le message CD en diffusion aux émetteurs ; ces derniers devront re-émettre plus tard. Par la suite, $Sender$ dénotera l'ETIOA représentant la spécification de l'émetteur (Figure 7 (b)) et Bus, l'ETIOA représentant la spécification de bus (Figure 7 (a)).

Nous avons expérimenté TGSE avec le protocole CSMA/CD à un bus et plusieurs émetteurs et l'objectif de test consistant en une émission de $!busy$ en diffusion par

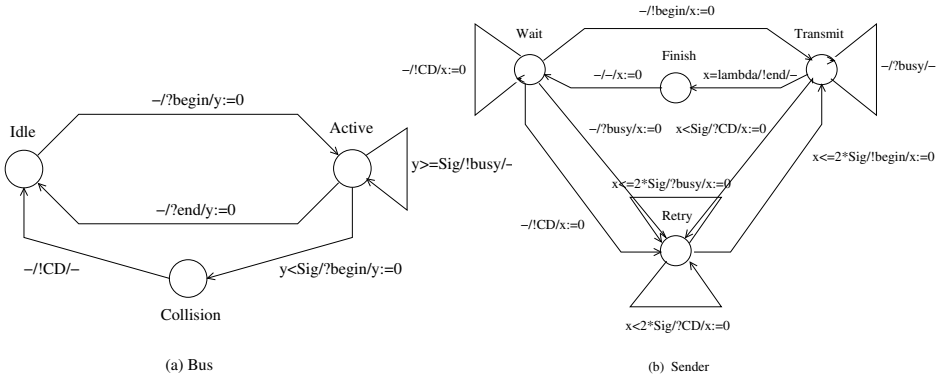


Figure 7. *Exemples de spécifi cations*

le bus suivie d'une demande d'émission $!begin$ par un émetteur à l'instant 5 selon l'horloge globale, sous un portable DELL INSPIRON 5100, de processeur intel P4 (2,4GHZ) et à 256Mo de RAM sous Mandrake 10.0 (TGSE a été aussi testé sous WindowsNT et RedHat). Lock représente le nombre de fois qu'une transition peut figurer dans le chemin, Taille du TC représente la taille moyenne d'un cas de test, Nb Sender le nombre des émetteurs considérés et Temps CPU est le temps moyen de génération.

Lock	Nb Sender	Taille du TC	Temps CPUs (s)
1	2	3	0.077
1	5	3	0.303
1	10	3	0.621
1	20	3	0.914
10^3	2	18	0.027
10^3	5	55	0.098
10^3	10	79	0.234
10^3	20	130	0.793

Malgré que le protocole CSMA/CD est de taille réduite, l'utilisation de plusieurs émetteurs augmente sa complexité. Les resultats obtenus sont très encouragants et des améliorations sont en cours.

7. Discussion

Dans cet article, nous avons traité le test des protocoles de communication temps-réel et à données. Le but était de réduire l'écart qui sépare les différents types et approches du test. Dans cet optique, nous avons présenté le modèle générique des systèmes communicants (CS) pouvant traiter différents types et modèles de test et

dont la sémantique est définie en terme d'automate temporisé étendue à entrée/sortie (ETIOA). Ce modèle définit un ensemble d'entités (composantes) communicantes (modélisées par des ETIOAs), des ressources communes (données et paramètres partagées) aux différentes composantes et une topologie de communication qui explicite les différentes synchronisations possibles dans un état globale du système. L'introduction de la topologie dans la spécification offre des mécanismes de modélisation, de synchronisation et de génération plus adaptés au test des protocoles.

Ce modèle a été mis en oeuvre dans l'outil TGSE (Émulation, Simulation et Génération de Test). La version actuelle de TGSE peut être utilisée pour le test actif orienté objectif de test (à une ou plusieurs composantes) et le test passif, mais ne supporte qu'un CS à topologie statique et dont l' ETIOA sémantique est événementiellement déterministe. Notre travail actuel s'oriente à enlever ces restrictions en étendant la topologie dans TGSE et investiguer les techniques de détérminisation à la volée dans le cas temporel à l'exemple de la technique présentée dans [TRI 04]. Une intention particulière est portée sur la définition et l'incorporation dans TGSE du silence des états à l'exemple de [TRI 04, BRI 04].

Remerciements.
Nous tenons à remercier les membres de l'action spécifique AS 32 menée par Ana Cavalli pour leur remarques constructives. Nous remercions également les élèves de l'ENSEIRB Dimitri Kandassa, Jamel Semeh, David Dogoh et Carine Beduz pour leur participations dans la réalisation de TGSE.

8. Bibliographie

[BRI 88] E. Brinskma. A Theory for the Derivation of Tests. In S. Aggarwal and Sabnani, eds. *Protocol Specifications, Test, and Verifications VIII*, 63-74. North-Holland, 1988.

[TRE 96] Jan Tretmans. Test Generation with Inputs, Outputs and Repetitive Quiescence, *Software - Concepts and Tools* 17(3) : 103-120 (1996).

[FER 97] J. Fernandez, C. Jard, T. Jéron, C Viho, An Experiment in Automatic Generation of Test Suites for Protocols with Verification Technology. *Sci. Comput. Program*. 29(1-2) : 123-146 (1997).

[BRI 04] Laura Brandán and Ed Brinksma. A test generation framework for quiescent real-time systems. *Proceedings of the 4rd International Workshop on Formal Approaches to Testing of Software, FATES2004*, Linz, Austria September 21 2004.

[SEO 03] S. Seol, M. Kim, S. Kang, J. Ryu. Fully automated interoperability test suite derivation for communication protocols, *Computer Networks* Volume 43, Pages 735 - 759, December 2003.

[CAR 00] Rachel Cardell-Oliver. Conformance Testing of Real-Time Systems with Timed Automata Specifications, *Formal Aspects of Computing*, 12(5) :350-371,2000.

[CLA 97] Duncan Clarke and Insup Lee. Automatic Test Generation for the Analysis of a Real-Time System : Case Study. In *3rd IEEE Real-Time Technology and Applications Sym-*

posium, 1997.

[DSS 98] A. En-Nouaary, R. Dssouli, F. Khenedek, and A. Elqortobi. Timed test cases generation based on state characterization technique, *In 19th IEEE Real Time Systems Symposium (RTSS'98)*, Madrid, Spain, 1998.

[HIG 99] T. Higashino, A. Nakata, K. Taniguchi, and A. Cavalli. Generating Test Cases for a Timed I/O Automaton model, *TESTCOM99*, Budapest, Hungary, September 1999.

[KOU 00] A. Koumsi, M. Akalay, R. Dssouli, A. En-Nouaary, L. Granger. An approach for testing real time protocols, *TESTCOM*, Ottawa, Canada, 2000.

[MAN 95] Dino Mandrioli, Sandro Morasca, and Angelo Morzenti. Generating Test Cases for Real-Time Systems from Logic Specifications, *ACM Transactions on Computer Systems*, 13(4) :365-398, 1995.

[SPR 01] Jan Springintveld, Frits Vaandrager, Pedro R. D'Argenio. Testing Timed Automata. *Theoretical Computer Science*, 252(1-2) :225-257, March 2001.

[BER 04] I. Berrada, R. Castanet, P. Félix. From the Feasibility Analysis to Real-Time Test Generation, *Studia Informatica Universalis* 2004.

[LAR 03] K. Larsen, M. Mikucionis, and B. Nielsenn. Real-time system testing on-the-fly. *In the 15th Nordic Workshop on Programming Theory (NWPT)*, 2003.

[TRI 04] M. Krichen and S. Tripakis. Black-box conformance testing for real-time systems. *In SPIN 2004* (2004), Spring-Verlag Heidelberg, pp. 109-126.

[CAV 04] Batiste Alcalde. Ana cavalli, Dongluo Chen, Davy Khuu, and David Lee. Network Protocol System Passive Testing for Fault Management : A Backward Checking Approach. *Proceeding of FORTE/PSTV'2004*, Madrid, Spain. LNCS 3235, September 2004.

[ALU 94] R. Alur and D. Dill. A theory of timed automata, *Theoretical Computer Science*, 126 :183-235, 1994.

[ARN 94] Arnold, A. Finite transiton systems. Semantics of communicating systems. *Prentice-Hall*, 1994

[MAD 04] Tomás Barros, Rabéa Boulifa and Eric Madelaine. Parameterized Models for Distributed java Objects. *Proceeding of FORTE/PSTV'2004*, Madrid, Spain. LNCS 3235, September 2004.

[CAL 04] Ismail Berrada, Richard Castanet and Patrick Félix. Techinques de Test d'Interopérabilité. Fourniture Calife, 2004.

[ZAI 99] Ana Cavalli, David Lee, Christian Rinderknecht and Fatiha Zaïdi. Hit-or-Jump : An algorithm for embedded testing with applications to IN services. *Proceeding of FORTE/PSTV'99*, Beijing, China. October 1999.

[JAR 99] Claude Jard, Thierry Jéron, Lénaïck Tanguy and César Viho. Remote testing can be as powerful as local testing. *Proceeding of FORTE/PSTV'99*, Beijing, China. October 1999.

[DSS 03] Abdeslam En-Nouaary, Rachida Dssouli : A Guided Method for Testing Timed Input Output Automata. TestCom 2003 : 211-225

[KOU 03] Ahmed Khoumsi, Thierry Jéron, Hervé Marchand. Test Cases Generation for Nondeterministic Real-Time Systems. FATES 2003 : 131-146