Une Approche Formelle pour Combiner les Tests Fonctionnels et les Tests de Charge

Moez Krichen*, Afef Jmal Maâlej**

*Faculté de CSIT, Université Al-Baha, Arabie Saoudite
Laboratoire ReDCAD, Université de Sfax, Tunisie
moez.krichen@redcad.org
www.redcad.org/members/mkrichen
**Laboratoire ReDCAD, Université de Sfax, Tunisie
afef.jmal@redcad.org
www.redcad.org/members/afef.jmal

Résumé. Dans ce travail, nous proposons une plate-forme de test basée sur modèle ayant pour but de combiner des tests fonctionnels et des tests de charge. Un modèle adéquat permettant de représenter cette corrélation est proposé. Il s'agit du modèle des automates temporisés étendus avec des entrées/sorties et des variables entières partagées. De plus, nous proposons une technique pour dériver des tests à base d'horloges analogiques à partir de la spécification du système sous test.

1 Introduction

Les méthodes formelles sont des techniques permettant de raisonner rigoureusement, à l'aide de logique mathématique, sur des programmes informatiques, afin de démontrer leur validité par rapport à une certaine spécification. Dans ce contexte, le test à base de modèles Tretmans (1999); Krichen et Tripakis (2006) consiste à utiliser une spécification décrite par un modèle formel abstrait comme référence afin de générer des tests automatisés, ce qui a comme principal avantage d'améliorer la qualité du processus de test en réduisant le coût et les délais d'exécution des tests.

Afin de considérer la dépendance entre le changement des comportements et des actions produites par un système et la charge exercée Maâlej et al. (2012, 2013); Maâlej et Krichen (2015), nous proposons une solution qui offre un cadre mathématique rigoureux dans un domaine où les aspects empiriques ont tendance à se perpétuer Maâlej et Krichen (2016). Pour ce faire, nous étendons la plate-forme de test proposée dans un travail précédent par Krichen et Tripakis (2009, 2004) pour le test de conformité des systèmes temps réels Krichen (2012, 2010).

En effet, nous adoptons la solution proposée dans le contexte des architectures orientées service, ayant en particulier des exigences temps réel fermes où il faut respecter des échéances temporelles et où la réponse du système dans les délais est essentielle. Des exemples de tels systèmes sont les transactions en bourse, les réservations de vol dans une agence de voyage, l'airbag dans la voiture, le système anti-blocage des roues (ABS), etc.

Le reste de ce papier est origanisé comme suit : La Section 2 donne un aperçu conernant les méthodes formelles, leurs principes et leur importance. La Section 3 introduit quelques notions préliminaires à savoir : (a) les systèmes à transitions étiquetées et temporisées, (b) les automates temporisés étendus et (c) les automates temporisés étendus avec entrées et sorties. La Section 4 introduit la notion des automates temporisés étendus avec des entrées/sorties et des variables entières partagées qui permet de modéliser la classe des sysèmes considérés. La Section 5 donne un aperçu concernant la technique de génération de tests adoptée. La Section 6 conclut l'article.

2 Motivation : Les méthodes formelles

Une des raisons de la présence d'erreurs dans les systèmes provient du fait que les personnes qui rédigent les cahiers des charges sont très rarement celles qui le construisent. De plus, le cahier des charges est très souvent écrit en langage naturel, ce qui est une source d'ambiguïtés, d'imprécisions voire de contradictions. Une autre raison de la présence d'erreurs est la complexité des systèmes considérés.

Par conséquent, il apparaît primordial de disposer de méthodes de conception rigoureuses et de techniques de validation automatiques et efficaces. Celles-ci doivent être utilisées dès le début de la phase de conception. En effet, plus une erreur sera détectée tard, plus elle risque d'être coûteuse. En plus, un tel travail ne peut être réalisé que dans un cadre mathématique bien défini permettant d'éviter tous les problèmes issus d'un langage non mathématique.

Dans ce contexte, les méthodes formelles visent à offrir un cadre mathématique permettant d'écrire de manière précise les systèmes et programmes que nous voulons construire. Ce cadre formel permet de lever les ambiguïtés existantes au niveau du cahier des charges, conséquence de l'utilisation du langage naturel Kitouni (2013).

Les systèmes de plus en plus complexes sont caractérisés par un indéterminisme dans leurs comportements et passent par des étapes d'abstraction très nombreuses, ce qui nécessite des représentations complexes et distinctes des comportements internes. Ces deux aspects donnent des spécifications indéterministes avec des actions ou opérations internes. Il est primordial de proposer des modèles qui puissent exprimer cela et leur utilisation à des fins de validation des systèmes réels correspondants. Les modèles de représentation des systèmes doivent alors pouvoir exprimer et capturer l'indéterminisme ainsi que la gestion des actions internes.

L'automatisation des différentes approches de validation nécessite des spécifications formelles. Le test basé sur les modèles nécessite l'utilisation de structures déterministes. De ce fait la propriété de déterminisme d'un modèle est souvent mise en avant pour son utilisation.

3 Notions préliminaires

Nous soulignons tout d'abord que le choix des automates temporisés comme formalisme de spécification dans notre travail est justifié par leur pouvoir expressif. En particulier, les automates temporisés offrent des avantages pour décrire, d'une façon appropriée, des contraintes fonctionnelles et temporelles, pour permettre la synchronisation comme la concurrence. En outre, des algorithmes de vérification ont été étendus à ces modèles Alur et al. (1993); Henzinger et al. (1994); Larsen et al. (1995) et des outils de model-checking ont été développés Yo-

vine (1997); Larsen et al. (1997) et appliqués avec succès sur des exemples issus du milieu industriel Bengtsson et al. (1996); Tripakis et Yovine (1995).

3.1 Systèmes à transitions étiquetées et temporisées

Les *Systèmes à Transitions Étiquetées et Temporisées* (TLTS pour Timed Labelled Transition Systems) permettent d'écrire la dynamique des systèmes combinant trois types d'actions : l'action "passage du temps", les actions observables et les actions internes. Les TLTS se prêtent bien à l'expression de la sémantique des automates temporisés.

Formellement, soient R l'ensemble de nombres réels positifs, Q l'ensemble de nombres rationnels positifs et N l'ensemble de nombres entiers positifs. Étant donné un ensemble fini d'*actions* Ac, l'ensemble $(\mathsf{Ac} \cup \mathsf{R})^*$ de toutes les *séquences temps réel* à longueur finie sur Ac sera notée $\mathsf{RT}(\mathsf{Ac})$. $\epsilon \in \mathsf{RT}(\mathsf{Ac})$ correspond à la séquence vide. Étant donné $\mathsf{Ac}' \subseteq \mathsf{Ac}$ et $\rho \in \mathsf{RT}(\mathsf{Ac})$, $\mathsf{P}_{\mathsf{Ac}'}(\rho)$ désigne la *projection* de ρ sur $\mathsf{Ac}' \cup \mathsf{R}$, obtenu en "éliminant" de ρ toutes les actions ne figurant pas dans $\mathsf{Ac}' \cup \mathsf{R}$. De même, $\mathsf{DP}_{\mathsf{Ac}'}(\rho)$ désigne la projection (discrète) de ρ sur Ac'. Par exemple, si $\mathsf{Ac} = \{a, b\}$, $\mathsf{Ac}' = \{a\}$ et $\rho = a\,1\,b\,2\,a\,3$, alors $\mathsf{P}_{\mathsf{Ac}'}(\rho) = a\,3\,a\,3$ et $\mathsf{DP}_{\mathsf{Ac}'}(\rho) = a\,a$. Le temps passé dans une séquence ρ, noté par $\mathsf{duration}(\rho)$, est la somme de tous les délais dans ρ, par exemple, $\mathsf{duration}(\epsilon) = 0$ et $\mathsf{duration}(a\,1\,b\,0.5) = 1.5$.

Dans ce qui suit, nous considérons un ensemble donné d'actions Ac, partitionné en deux ensembles disjoints : un ensemble d'*actions d'entrées* $\mathsf{Ac}_{\mathsf{in}}$ et un ensemble d'*actions de sorties* $\mathsf{Ac}_{\mathsf{out}}$. Les actions dans $\mathsf{Ac}_{\mathsf{in}} \cup \mathsf{Ac}_{\mathsf{out}}$ sont appelées actions *observables*. Nous supposons aussi qu'il existe une action *non observable* $\tau \notin \mathsf{Ac}$. Soit $\mathsf{Ac}_\tau = \mathsf{Ac} \cup \{\tau\}$.

Un TLTS sur Ac est un tuple $(S, s_0, \mathsf{Ac}, T_d, T_t)$, tels que :

— S est un ensemble d'*états* ;
— s_0 est l'état initial ;
— T_d est un ensemble de *transitions discrètes* de la forme (s, a, s') tel que $s, s' \in S$ et $a \in \mathsf{Ac}$;
— T_t est un ensemble de *transitions temporisées* de la forme (s, t, s') tel que $s, s' \in S$ et $t \in \mathsf{R}$.

Les transitions temporisées doivent être déterministes, c'est à dire, $(s, t, s') \in T_t$ et $(s, t, s'') \in T_t$ implique $s' = s''$. T_t doit aussi satisfaire les conditions suivantes : $(s, t, s') \in T_t$ et $(s', t', s'') \in T_t$ implique $(s, t + t', s'') \in T_t$; $(s, t, s') \in T_t$ implique que pour tout $t' < t$, il existe $(s, t', s'') \in T_t$.

Nous utilisons une notation standard concernant les TLTS. Pour $s, s', s_i \in S$, $\mu, \mu_i \in \mathsf{Ac}_\tau \cup \mathsf{R}$, $a, a_i \in \mathsf{Ac} \cup \mathsf{R}$, $\rho \in \mathsf{RT}(\mathsf{Ac}_\tau)$ et $\sigma \in \mathsf{RT}(\mathsf{Ac})$, nous avons :

— Les transitions générales :
  — $s \xrightarrow{\mu} s' \stackrel{Def}{=} (s, \mu, s') \in T_d \cup T_t$;
  — $s \xrightarrow{\mu} \stackrel{Def}{=} \exists s' : s \xrightarrow{\mu} s'$;
  — $s \not\xrightarrow{\mu} \stackrel{Def}{=} \nexists s' : s \xrightarrow{\mu} s'$;
  — $s \xrightarrow{\mu_1 \cdots \mu_n} s' \stackrel{Def}{=} \exists s_1, \cdots, s_n : s = s_1 \xrightarrow{\mu_1} s_2 \xrightarrow{\mu_2} \cdots \xrightarrow{\mu_n} s_n = s'$;
  — $s \xrightarrow{\rho} \stackrel{Def}{=} \exists s' : s \xrightarrow{\rho} s'$;
  — $s \not\xrightarrow{\rho} \stackrel{Def}{=} \nexists s' : s \xrightarrow{\rho} s'$.
— Les transitions observables :
  — $s \stackrel{\epsilon}{\Rightarrow} s' \stackrel{Def}{=} s = s'$ or $s \xrightarrow{\tau \cdots \tau} s'$;

— $s \overset{a}{\Rightarrow} s' \overset{Def}{=} \exists s_1, s_2 : s \overset{\epsilon}{\Rightarrow} s_1 \overset{a}{\rightarrow} s_2 \overset{\epsilon}{\Rightarrow} s'$;
— $s \overset{a}{\not\Rightarrow} \overset{Def}{=} \nexists s' : s \overset{a}{\Rightarrow} s'$;
— $s \overset{a_1 \cdots a_n}{\Longrightarrow} s' \overset{Def}{=} \exists s_1, \cdots, s_n : s = s_1 \overset{a_1}{\Rightarrow} s_2 \overset{a_2}{\Rightarrow} \cdots \overset{a_n}{\Rightarrow} s_n = s'$;
— $s \overset{\sigma}{\Rightarrow} \overset{Def}{=} \exists s' : s \overset{\sigma}{\Rightarrow} s'$;
— $s \overset{\sigma}{\not\Rightarrow} \overset{Def}{=} \nexists s' : s \overset{\sigma}{\Rightarrow} s'$.

Une séquence de la forme $s_0 \overset{\mu_1}{\rightarrow} s \overset{\mu_2}{\rightarrow} \cdots \overset{\mu_n}{\rightarrow} s'$ est appelée un *essai* (run) et une séquence de la forme $s_0 \overset{a_1}{\Rightarrow} s \overset{a_2}{\Rightarrow} \cdots \overset{a_n}{\Rightarrow} s'$ est un *essai observable*.

3.2 Automates temporisés étendus

Nous utilisons les automates temporisés Alur et Dill (1994) avec délais (deadlines) pour modéliser l'urgence Krichen et Tripakis (2009). Un *automate temporisé étendu sur* Ac est un tuple $A = (Q, q_0, X, \mathsf{I}, \mathsf{Ac}, \mathsf{E})$, tels que :

— Q est un ensemble fini de *places* ;
— $q_0 \in Q$ est la place initiale ;
— X est un ensemble fini d'*horloges* ;
— I est un ensemble fini de variables entières ;
— E est un ensemble fini d'*arcs (edges)*.

Chaque arc est un tuple $(q, q', \psi, r, inc, dec, d, a)$, tels que :

— $q, q' \in Q$ sont les places source et destination ;
— ψ est la *condition (guard)*, une conjonction de contraintes de la forme $x \# c$, tels que $x \in X \cup \mathsf{I}$, c est un entier constant et $\# \in \{<, \leq, =, \geq, >\}$;
— $r \subseteq X \cup \mathsf{I}$ est un ensemble d'horloges et de variables entières à *réinitialiser (reset)* à zéro ;
— $inc \subseteq \mathsf{I}$ est un ensemble de variables entières (disjoint de r) à *incrémenter* de un ;
— $dec \subseteq \mathsf{I}$ est un ensemble de variables entières (disjoint de r et inc) à *décrémenter* de un ;
— $d \in \{\mathsf{lazy}, \mathsf{delayable}, \mathsf{eager}\}$ est le *délai* ;
— $a \in \mathsf{Ac}$ est l'action.

Un exemple d'automate temporisé étendu $A = (Q, q_0, X, \mathsf{I}, \mathsf{Ac}, \mathsf{E})$ sur l'ensemble d'actions $\mathsf{Ac} = \{a, b, c, d\}$ est présenté dans la Figure 1 tels que :

— $Q = \{q_0, q_1, q_2, q_3\}$ est un ensemble de places ;
— q_0 est la place initiale ;
— $X = \{x\}$ est un ensemble fini d'horloges ;
— $\mathsf{I} = \{i\}$ est un ensemble fini de variables entières ;
— E est l'ensemble d'arcs dessinés dans la figure.

La Figure 1 utilise la notation suivante :

— "$x := 0$" signifie réinitialiser l'horloge x à 0 ;
— "$i := 0$" signifie réinitialiser la variable entière i à 0 ;
— "$i + +$" signifie incrémenter i de 1 [1] ;
— "$i - -$" signifie décrémenter i de 1 [2].

1. Il est aussi possible d'utiliser la notation usuelle "$i := i + 1$".
2. Il est aussi possible d'utiliser la notation usuelle "$i := i - 1$".

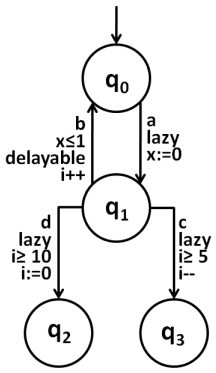

FIG. 1 – *Exemple d'un automate temporisé étendu*

Un automate temporisé étendu A définit un TLTS infini désigné par L_A. Ses états sont des triplets $s = (q, v_X, v_\mathsf{I})$, tels que $q \in Q$, $v_X : X \rightarrow \mathsf{R}$ est une *évaluation d'horloge* et $v_\mathsf{I} : \mathsf{I} \rightarrow \mathsf{N}$ est une *valeur de variable entière*. $\vec{0}_X$ est la valeur assignant 0 à chaque horloge de A. $\vec{0}_\mathsf{I}$ est la valeur assignant 0 à chaque variable entière de A. S_A est l'ensemble de tous les états et $s_0^A = (q_0, \vec{0}_X, \vec{0}_\mathsf{I})$ est l'état initial.

Les transitions discrètes sont de la forme :

$$(q, v_X, v_\mathsf{I}) \xrightarrow{a} (q', v'_X, v'_\mathsf{I})$$

tels que $a \in \mathsf{Ac}$ et il existe un arc

$$(q, q', \psi, r, inc, dec, d, a)$$

tels que (v_X, v_I) satisfait ψ et (v'_X, v'_I) est obtenu en :

— réinitialisant à zéro toutes les horloges et les variables entières dans r ;
— incrémentant les variables entières dans inc de un ;
— décrémentant les variables entières dans dec de un ;
— laissant toutes les autres variables inchangées.

Les transitions temporisées sont de la forme :

$$(q, v_X, v_\mathsf{I}) \xrightarrow{t} (q, v_X + t, v_\mathsf{I})$$

tels que $t \in \mathsf{R}, t > 0$ et il n'existe aucun arc

$$(q, q'', \psi, r, inc, dec, d, a)$$

tels que :

— soit $d = \mathsf{delayable}$ et il existe $0 \leq t_1 < t_2 \leq t$ tels que $(v_X + t_1, v_\mathsf{I}) \models \psi$ et $(v_X + t_2, v_\mathsf{I}) \not\models \psi$;
— ou $d = \mathsf{eager}$ et $(v_X, v_\mathsf{I}) \models \psi$.

Les arcs de type lazy n'affectent pas les sémantiques. Ils signifient qu'un arc n'est ni "différé" (delayable), ni "pressé" (). Plus précisément, les arcs de type lazy ne sont pas capables de bloquer la progression du temps, tandis que les arcs de type delayable et eager le sont. Nous ne considérons pas des arcs de type delayable avec des conditions de la forme $x < c$ comme il n'y a pas de moment *dernier (latest)* lorsque la condition est encore vraie. De même, nous ne permettons pas des arcs de type eager avec des conditions de la forme $x > c$ comme il n'y a pas de moment *plus tôt (earliest)* lorsque la condition devient vraie. Un état $s \in S_A$ est *accessible (reachable)* s'il existe $\rho \in \mathsf{RT}(\mathsf{Ac})$ tel que $s_0^A \xrightarrow{\rho} s$. L'ensemble des états accessibles de A est noté $\mathsf{Reach}(A)$.

3.3 Automates temporisés étendus avec entrées et sorties

Un *automate temporisé étendu avec entrées et sorties* (ETAIO pour Extended Timed Automata with Inputs and Outputs) est un automate temporisé étendu sur l'ensemble partitionné d'actions

$$\mathsf{Ac}_\tau = \mathsf{Ac}_{\mathsf{in}} \cup \mathsf{Ac}_{\mathsf{out}} \cup \{\tau\}.$$

En effet, l'implémentation et le testeur vont interagir : le fait d'utiliser des entrées et des sorties permet de distinguer les éléments contrôlables par le testeur (les entrées de l'implémentation) des éléments observables par le testeur (les sorties de l'implémentation) qu'il ne peut qu'observer de façon passive.

Pour plus de clarté, nous allons inclure explicitement les entrées et les sorties dans la définition d'un ETAIO A et écrire

$$(Q, q_0, X, \mathsf{I}, \mathsf{Ac}_{\mathsf{in}}, \mathsf{Ac}_{\mathsf{out}}, \mathsf{E})$$

au lieu de

$$(Q, q_0, X, \mathsf{I}, \mathsf{Ac}_\tau, \mathsf{E}).$$

Un ETAIO est dit *observable* si aucun de ses arcs n'est étiqueté par τ.

Étant donné un ensemble d'entrées $\mathsf{Ac}' \subseteq \mathsf{Ac}_{\mathsf{in}}$, un ETAIO A est dit *d'entrée activée (input-enabled)* en ce qui concerne Ac' si il peut accepter n'importe quelle entrée dans Ac' à n'importe quel état :

$$\forall s \in \mathsf{Reach}(A)\,.\,\forall a \in \mathsf{Ac}' \;:\; s \xrightarrow{a}\,.$$

Un ETAIO A est simplement dit "input-enabled" lorsque $\mathsf{Ac}' = \mathsf{Ac}_{\mathsf{in}}$. A est dit lazy-*input* en ce qui concerne Ac' si les délais sur toutes les transitions étiquetées avec des actions "input" dans Ac' sont de type lazy. Il est dit lazy-input si il est lazy-input en ce qui concerne $\mathsf{Ac}_{\mathsf{in}}$. Il est à noter que input-enabled n'implique pas lazy-input en général.

A est dit *déterministe* si

$$\forall s, s', s'' \in \mathsf{Reach}(A)\,.\,\forall a \in \mathsf{Ac}_\tau \;:$$

$$s \xrightarrow{a} s' \wedge s \xrightarrow{a} s'' \Rightarrow s' = s''.$$

A est dit *non-bloquant* si

$$\forall s \in \mathsf{Reach}(A)\,.\,\forall t \in \mathsf{R}\,.\,\exists \rho \in \mathsf{RT}(\mathsf{Ac}_{\mathsf{out}} \cup \{\tau\}) \;:$$

$$\mathsf{duration}(\rho) = t \wedge s \xrightarrow{\rho} .$$

Cette condition garantit que A ne bloquera pas le temps quel que soit l'environnement.

L'ensemble de *traces temporisées* d'un ETAIO A est défini comme étant

$$\mathsf{TTr}(A) = \{\rho \mid \rho \in \mathsf{RT}(\mathsf{Ac}_\tau) \wedge s_0^A \xrightarrow{\rho}\}.$$

L'ensemble de *traces temporisées observables* de A est défini comme étant

$$\mathsf{OTTr}(A) = \{\mathsf{P}_{\mathsf{Ac}}(\rho) \mid \rho \in \mathsf{RT}(\mathsf{Ac}_\tau) \wedge s_0^A \xrightarrow{\rho}\}.$$

Le TLTS défini par un ETAIO est dit un *LTS temporisé de type input-output (timed input-output LTS)* (TIOLTS). Dans ce qui suit, sauf si mentionné, tous les ETAIO considérés sont définis par rapport aux mêmes ensembles $\mathsf{Ac}_{\mathsf{in}}$, $\mathsf{Ac}_{\mathsf{out}}$ et l'action inobservable τ. En ce qui concerne les ETAIO, un TIOLTS donné L est noté

$$(S, s_0, \mathsf{Ac}_{\mathsf{in}}, \mathsf{Ac}_{\mathsf{out}}, T_d, T_t)$$

au lieu de

$$(S, s_0, \mathsf{Ac}_\tau, T_d, T_t).$$

Les deux opérateurs $\mathsf{TTr}(\cdot)$ et $\mathsf{OTTr}(\cdot)$ sont étendus d'une façon naturelle pour le cas de TIOLTS.

4 Automates temporisés étendus avec des entrées/sorties et des variables entières partagées

Nous proposons dans ce qui suit notre approche qui permet de modéliser la notion de charge d'une part, et d'adapter, d'autre part, les ETAIO pour mettre en évidence la communication entre plusieurs instances à travers l'accès aux variables partagées. Dans ce contexte, nous proposons et illustrons quelques patrons de modélisation Maâlej et Krichen (2016).

Afin de modéliser la concurrence entre les ETAIO, nous appliquons leur composition parallèle. De plus, il est possible que ces ETAIO synchronisent et communiquent entre eux à travers l'accès et la mise à jour des valeurs d'une ou de plusieurs ressources partagées (par exemple, nombre de places à réserver). Formellement, il s'agit d'utiliser des variables globales partagées. Pour des raisons de simplicité, nous considérons des variables entières.

En effet, soit n un entier positif tel que $n \geq 2$. Nous considérons n ETAIO $(A_i)_{1 \leq i \leq n}$ tels que :

$$A_i = (Q^i, q_0^i, X^i, \mathsf{I}, \mathsf{Ac}_{\mathsf{in}}^i, \mathsf{Ac}_{\mathsf{out}}^i, \mathsf{E}^i).$$

C'est à dire l'ensemble des variables entières I est partagé entre tous les ETAIO considérés $(A_i)_{1 \leq i \leq n}$ et aucun élément de Q^i, X^i, $\mathsf{Ac}_{\mathsf{in}}^i$ et $\mathsf{Ac}_{\mathsf{out}}^i$ n'est partagé avec les autres ETAIO $(A_j)_{j \neq i}$.

Le TIOLTS suivant généré par le produit parallèle des ETAIO spécifié juste aprés

$$L^P = (S^P, s_0^P, \mathsf{Ac}_{\mathsf{in}}^P, \mathsf{Ac}_{\mathsf{out}}^P, T_d^P, T_t^P)$$

$$(A_i)_{1 \leq i \leq n}$$

est défini comme suit :

$$s_0^P = ((q_0^1, \cdots, q_0^n), (\vec{0}_{X^0}, \cdots, \vec{0}_{X^n}), \vec{0}_{\mathsf{I}})$$

$$\mathsf{Ac}_{\mathsf{in}}^P = \bigcup_{1 \leq i \leq n} \mathsf{Ac}_{\mathsf{in}}^i,\ \mathsf{Ac}_{\mathsf{out}}^P = \bigcup_{1 \leq i \leq n} \mathsf{Ac}_{\mathsf{out}}^i$$

et S^P, T_d^P et T_t^P sont les ensembles les plus petits tels que :

— $s_0^P \in S^P$;

— Pour $s^P = ((q^1, \cdots, q^n), (v_{X^0}, \cdots, v_{X^n}), v_{\mathsf{I}}) \in S^P$ et $\delta \in \mathsf{R}$:

$$\forall 1 \leq i \leq n : (q_i, v_{X^i}, v_I) \xrightarrow{\delta} (q_i, v_{X^i} + \delta, v_I) \in T_t^i$$

$$\Rightarrow s^{'P} = ((q^1, \cdots, q^n), (v_{X^0} + \delta, \cdots, v_{X^n} + \delta), v_{\mathsf{I}}) \in S^P$$

et

$$s^P \xrightarrow{\delta} s^{'P} \in T_t \tag{1}$$

— Pour $s^P = ((q^1, \cdots, q^n), (v_{X^0}, \cdots, v_{X^n}), v_{\mathsf{I}}) \in S^P$, $1 \leq i \leq n$ et $a_i \in {\mathsf{Ac}_\tau}^i$: [3]

$$(q_i, v_{X^i}, v_I) \xrightarrow{a_i} (q_i', v'_{X^i}, v_I') \in T_d^i$$

$$\Rightarrow s^{'P} = (q^{'p}, v_X^{'p}, v_{\mathsf{I}}') \in S^P \quad \wedge \quad s^P \xrightarrow{a_i} s^{'P} \in T_d$$

tels que

$$q^{'p} = (q^1, \cdots, q^{i-1}, q'^i, q^{i+1}, \cdots, q^n)$$

et

$$v_X^{'p} = (v_{X^0}, \cdots, v_{X^{i-1}}, v'_{X^i}, v_{X^{i+1}} \cdots, v_{X^n}) \tag{2}$$

Il convient de préciser à ce niveau qu'il est possible de définir la composition parallèle de n copies $(A_i)_{1 \leq i \leq n}$ du même ETAIO A. Dans ce cas, nous supposons qu'il est possible de distinguer les ensembles des entrées et sorties des différentes instances (en attribuant un identifiant unique pour chaque instance par exemple). Bien évidemment, les n instances partagent l'ensemble de variables entières de l'ETAIO A. Le TIOLTS obtenu est noté L_n^P.

5 Génération de test

Nous adaptons dans cette partie l'algorithme de génération de test de Tretmans Tretmans (1999). L'algorithme construit un test sous la forme d'un arbre. Un nœud dans l'arbre est un ensemble d'états S de la spécification et représente la "connaissance" du testeur de l'état du test actuel. L'algorithme étend le test en ajoutant des successeurs à un nœud feuille, comme illustré dans la Figure 2 (Tretmans (1999). Pour toute sortie *illégale* a_i (sortie qui ne peut pas

3. ${\mathsf{Ac}_\tau}^i = \mathsf{Ac}_{\mathsf{in}}^i \cup \mathsf{Ac}_{\mathsf{out}}^i \cup \{\tau\}$

avoir lieu à partir de n'importe quel état dans S), le test produit un Fail. Pour toute sortie légale b_i, le test évolue vers le nœud S_i, qui est l'ensemble des états où peut être la spécification après l'émission de b_i. S'il existe une entrée c qui peut être acceptée par la spécification pour n'importe quel état dans S, alors le test peut décider d'émettre cette entrée. Au niveau de n'importe quel nœud, l'algorithme peut décider d'arrêter le test et étiqueter ce nœud comme Pass.

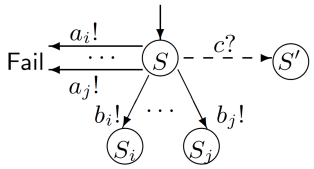

FIG. 2 – *Schéma de génération de test générique*

Les tests à base d'horloge analogique ne peuvent pas être directement représentés comme un arbre fini. En effet, il y a a priori un ensemble infini de délais observables possibles au niveau d'un nœud donné. Pour remédier à ceci, nous utilisons l'idée de Tripakis (2002). Nous représentons un test à base d'horloge analogique comme un *algorithme*. Ce dernier réalise essentiellement une construction par sous-ensembles sur l'automate de la spécification, durant l'exécution du test. De ce fait, notre méthode de test peut être classifiée comme *à la volée* ou *en ligne*, signifiant que le test est généré au même moment auquel il est exécuté. Plus précisément, le testeur va maintenir un ensemble d'états S du TIOLTS $L^P_{S,n}$. S sera mis à jour à chaque fois qu'une action est observée ou qu'un délai quelconque s'écoule. Étant donné qu'un délai donné n'est pas connu à priori, il doit être une entrée pour la fonction de mise à jour. Nous définissons les opérateurs suivants :

$$\mathsf{dsucc}(S,a) = \{s' \mid \exists s \in S \ : \ s \xrightarrow{a} s'\}$$

$$\mathsf{tsucc}(S,t) =$$

$$\{s' \mid \exists s \in S \ . \ \exists \rho \in \mathsf{RT}(\{\tau\}) \ : \ \mathsf{duration}(\rho) = t \wedge s \xrightarrow{\rho} s'\}$$

où $a \in \mathsf{Ac}^n$ et $t \in \mathsf{R}$. $\mathsf{dsucc}(S,a)$ contient tous les états qui peuvent être atteints par un état quelconque dans S exécutant l'action a. $\mathsf{tsucc}(S,t)$ contient tous les états qui peuvent être atteints par un état quelconque dans S via une séquence ρ qui contient des actions non observables et prend exactement t unités de temps Maâlej et Krichen (2016).

Le test opère comme suit. Il commence à l'état $S_0 = \mathsf{tsucc}(\{s^P_{n,0}\}, 0)$ où $s^P_{n,0}$ est l'état initial de $L^P_{S,n}$. Étant donné un état courant S : Si la sortie a est reçue t unités de temps après avoir entré S, alors S est mis à jour à $\mathsf{dsucc}(\mathsf{tsucc}(S,t),a)$. Si jamais l'ensemble S devient vide, le test annonce Fail. À n'importe quel point, pour une entrée b, si $\mathsf{dsucc}(S,b) \neq \emptyset$, le test peut décider d'émettre b et met à jour son état en conséquence.

6 Conclusion

Dans cet article, nous avons introduit les briques de base pour combiner des tests fonctionnels et de charge. Pour cela, nous avons proposé une approche générique de test à base d'automates temporisés étendus avec des entrées/sorties et des variables entières partagées. Dans d'autres travaux Krichen et al. (2018a), nous avons tenté d'appliquer cette approche pour le test de conformité des compositions de services Web sous des conditions de charge variées, tout en considérant l'environnement d'exécution de ces applications.

Dans le futur nous pourrons appliquer des techniques de raffinement Bensalem et al. (2007) ou de sélection Lahami et al. (2015) pour simplifier l'étape de génération de tests en terme de temps de calcul et espace de stockage. Nous pourrons également appliquer des techniques de placement des testeurs Krichen et al. (2018b); Maâlej et al. (2018); Lahami et al. (2012) utilisés afin de tenir compte des ressources disponibles au niveau des diffèrents nœuds du systeme. Il est envisageable aussi d'adopter le standard de test TTCN3 Lahami et al. (2012) et d'utiliser des services cloud pour la génération et l'exécution des tests Lahami et al. (2019); Lahami et al. (2018). Finalement, il est également possible de considérer des aspects de securité Krichen et al. (2020) et de les combiner avec des aspects de conformité.

Références

Alur, R., C. Courcoubetis, et D. L. Dill (1993). Model-checking in dense real-time. *Inf. Comput. 104*(1), 2–34.

Alur, R. et D. L. Dill (1994). A theory of timed automata. *Theoretical computer science 126*(2), 183–235.

Bengtsson, J., W. O. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, et W. Yi (1996). Verification of an audio protocol with bus collision using UPPAAL. In *Proceedings of the 8th International Conference on Computer Aided Verification (CAV)*, Volume 1102 of *LNCS*, New Brunswick, NJ, USA, pp. 244–256. Springer.

Bensalem, S., M. Krichen, L. Majdoub, R. Robbana, et S. Tripakis (2007). A simplified approach for testing real-time systems based on action refinement. In *Proceedings of ISoLA Workshop on Leveraging Applications of Formal Methods, Verification and Validation*, Volume RNTI-SM-1 of *Revue des Nouvelles Technologies de l'Information*, Poitiers-Futuroscope, France, pp. 191–202. Cépaduès-Éditions.

Henzinger, T. A., X. Nicollin, J. Sifakis, et S. Yovine (1994). Symbolic model checking for real-time systems. *Inf. Comput. 111*(2), 193–244.

Kitouni, I. (2013). *Modèles et Algorithmes pour le Test des Systèmes Temps Réel*. Ph. D. thesis, PhD thesis, Université de Mentouri, Constantine.

Krichen, M. (2010). A formal framework for conformance testing of distributed real-time systems. In *Principles of Distributed Systems - 14th International Conference, OPODIS 2010, Tozeur, Tunisia, December 14-17, 2010. Proceedings*, pp. 139–142.

Krichen, M. (2012). A formal framework for black-box conformance testing of distributed real-time systems. *IJCCBS 3*(1/2), 26–43.

Krichen, M., M. Lahami, O. Cheikhrouhou, R. Alroobaea, et A. J. Maâlej (2020). *Security Testing of Internet of Things for Smart City Applications : A Formal Approach*, pp. 629–653. Cham : Springer International Publishing.

Krichen, M., A. J. Maâlej, et M. Lahami (2018a). A model-based approach to combine conformance and load tests : an ehealth case study. *IJCCBS 8*(3/4), 282–310.

Krichen, M., A. J. Maâlej, M. Lahami, et M. Jmaiel (2018b). A resource-aware model-based framework for load testing of WS-BPEL compositions. In *Enterprise Information Systems - 20th International Conference, ICEIS 2018, Funchal, Madeira, Portugal, March 21-24, 2018, Revised Selected Papers*, pp. 130–157.

Krichen, M. et S. Tripakis (2004). Black-box conformance testing for real-time systems. In *Proceedings of SPIN Workshop*, Volume 2989 of *LNCS*, Barcelona, Spain, pp. 109–126. Springer.

Krichen, M. et S. Tripakis (2006). Interesting properties of the real-time conformance relation. In *Theoretical Aspects of Computing - ICTAC 2006, Third International Colloquium, Tunis, Tunisia, November 20-24, 2006, Proceedings*, pp. 317–331.

Krichen, M. et S. Tripakis (2009). Conformance testing for real-time systems. *Formal Methods in System Design 34*(3), 238–304.

Lahami, M., F. Fakhfakh, M. Krichen, et M. Jmaiel (2012). Towards a TTCN-3 test system for runtime testing of adaptable and distributed systems. In *Testing Software and Systems - 24th IFIP WG 6.1 International Conference, ICTSS 2012, Aalborg, Denmark, November 19-21, 2012. Proceedings*, pp. 71–86.

Lahami, M., M. Krichen, et R. Alroobaea (2018). Towards a test execution platform as-a-service : Application in the e-health domain. In *2018 International Conference on Control, Automation and Diagnosis (ICCAD)*, pp. 1–6.

Lahami, M., M. Krichen, et R. Alroobaea (2019). Tepaas : test execution platform as-a-service applied in the context of e-health. *IJAACS 12*(3), 264–283.

Lahami, M., M. Krichen, H. Barhoumi, et M. Jmaiel (2015). Selective test generation approach for testing dynamic behavioral adaptations. In *Testing Software and Systems - 27th IFIP WG 6.1 International Conference, ICTSS 2015, Sharjah and Dubai, United Arab Emirates, November 23-25, 2015, Proceedings*, pp. 224–239.

Lahami, M., M. Krichen, M. Bouchakwa, et M. Jmaiel (2012). Using knapsack problem model to design a resource aware test architecture for adaptable and distributed systems. In *Testing Software and Systems - 24th IFIP WG 6.1 International Conference, ICTSS 2012, Aalborg, Denmark, November 19-21, 2012. Proceedings*, pp. 103–118.

Larsen, K. G., P. Pettersson, et W. Yi (1995). Model-checking for real-time systems. In *Proceedings of the 10th International Symposium on Fundamentals of Computation Theory (FCT)*, Volume 965 of *LNCS*, Dresden, Germany, pp. 62–88. Springer.

Larsen, K. G., P. Pettersson, et W. Yi (1997). Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT) 1*(1), 134–152.

Maâlej, A. J., M. Hamza, M. Krichen, et M. Jmaïel (2013). Automated significant load testing for ws-bpel compositions. In *Proceedings of the 6th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, Luxembourg, pp. 144–153. IEEE

Computer Society.

Maâlej, A. J. et M. Krichen (2015). Study on the limitations of WS-BPEL compositions under load conditions. *Comput. J. 58*(3), 385–402.

Maâlej, A. J. et M. Krichen (2016). A model based approach to combine load and functional tests for service oriented architectures. In *Proceedings of the 10th International Workshop on Verification and Evaluation of Computer and Communication System (VECoS)*, Volume 1689 of *CEUR Workshop Proceedings*, Tunis, Tunisia, pp. 123–140. CEUR-WS.org.

Maâlej, A. J., M. Krichen, et M. Jmaïel (2012). Conformance testing of ws-bpel compositions under various load conditions. In *Proceedings of the 36th IEEE Annual International Computer Software and Applications Conference*, Izmir, Turkey, pp. 371. IEEE Computer Society.

Maâlej, A. J., M. Lahami, M. Krichen, et M. Jmaïel (2018). Distributed and resource-aware load testing of WS-BPEL compositions. In *Proceedings of the 20th International Conference on Enterprise Information Systems, ICEIS 2018, Funchal, Madeira, Portugal, March 21-24, 2018, Volume 2.*, pp. 29–38.

Tretmans, J. (1999). Testing concurrent systems : A formal approach. In *Proceedings of the 10th International Conference on Concurrency Theory*, Volume 1664 of *LNCS*, pp. 46–65. Springer.

Tripakis, S. (2002). Fault diagnosis for timed automata. In *Proceedings of the 7th International Symposium on Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT)*, Volume 2469 of *LNCS*, Oldenburg, Germany, pp. 205–224. Springer.

Tripakis, S. et S. Yovine (1995). Verification of the fast-reservation protocol with delayed transmission using kronos. Technical report, Verimag, Grenoble, France.

Yovine, S. (1997). KRONOS : A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer (STTT) 1*(1-2), 123–133.

Summary

In this work, we propose a model-based test platform aiming to combine functional and load tests. An appropriate model to represent this correlation is proposed. The proposed formalism is based on the model of extended timed automata with inputs/ outputs and shared integer variables. In addition, we propose a technique to derive analog clock based tests from the specification of the system under test.