Abstractions de modèles en automates temporisés pour la validation temporelle d'architectures embarquées

Karen Godary-Dejean, Isabelle Augé-Blum

Abstractions de modèles en automates temporisés pour la validation temporelle d'architectures embarquées

Karen Godary, Isabelle Augé-Blum

Laboratoire CITI, INSA Lyon
Bât L. de Vinci, 21 av. Jean Capelle, 69621 Villeurbanne Cedex
karen.godary@insa-lyon.fr

RÉSUMÉ. *La fiabilité des applications distribuées temps réel critiques doit être garantie par des techniques formelles de validation, comme le model-checking. Cependant, ces méthodes ont souvent des problèmes d'explosion combinatoire. Cet article propose des abstractions efficaces pour la modélisation et la validation temporelle de l'architecture TTA (Time-Triggered Architecture) avec des automates temporisés (UPPAAL).*

ABSTRACT. *The reliability of critical real time distributed applications must be guaranty by formal techniques of validation, as the model-checking. However these techniques often lead to combinatory explosion problems. This paper proposes efficient abstractions of the timed automata model of TTA (Time-Triggered Architecture) in a temporal validation context.*

MOTS-CLÉS : *abstraction de modèle, validation temporelle, automates temporisés, architecture embarquée*

KEYWORDS: *model abstraction, temporal validation, timed automata, embedded architecture*

1. Introduction

Dans le domaine automobile, des systèmes distribués temps réel tolérants aux fautes se rencontrent de plus en plus pour le *X-by-wire* : applications distribuées, temps réel et critiques comme par exemple le freinage. Ces systèmes se doivent d'offrir des temps de réponse bornés, même en présence de perturbations liées à l'environnement. Des architectures spécifiques ont été conçues, comme par exemple l'architecture TTA *Time-Triggered Architecture* (Kopetz, 1998) qui est l'objet de notre étude. TTA est constituée de plusieurs couches : une couche applicative, une couche *FTlayer* dédiée à la redondance d'applications et une couche de communication basée sur le protocole TTP/C (*Time-Triggered Protocol for class C application*) (TTT 2003). Ce protocole se base sur une vue globale du système partagée par tous les nœuds (garantie par des mécanismes spécifiques de synchronisation des horloges locales et de *membership*). La tolérance aux fautes est intégrée à l'architecture et assurée par différents principes : comportement déterministe dû à la technologie *time-triggered*, redondance matérielle et logicielle, mécanismes et composants (les *bus guardians*) spécifiques à la détection et à la tolérance aux fautes.

Dans le contexte étudié la criticité des applications nécessite des phases de validation de la fiabilité des architectures, en particulier du point de vue temporel. Les méthodes de validation utilisées actuellement comme la simulation, le test ou l'injection de fautes ne sont dans ce cadre pas suffisantes. Un autre type de validation plus adapté est nécessaire, plus formel et plus exhaustif. En particulier, le *model checking* permet la vérification de propriétés en parcourant l'ensemble des états possibles du système. Cette technique a été choisie dans le cadre d'une méthodologie de validation temporelle adaptée à l'architecture TTA, proposée dans (Godary, 2004). La première phase de cette méthodologie consiste dans le choix d'un formalisme de modélisation adapté à TTA et en la conception d'un modèle de l'architecture. Le formalisme de modélisation choisi sont les TSA, *Timed Safety Automata*, extension des automates temporisés. La deuxième étape de la méthodologie consiste en la vérification de propriétés sur ce modèle par *model-checking* en utilisant l'outil UPPAAL (Larsen *et al.*, 1997b). Dans le cadre de la validation temporelle de TTA, les propriétés vérifiées sont les pires temps d'exécution des services de l'architecture.

Le problème du *model checking* réside dans l'explosion combinatoire de l'espace d'états du système. Pour pallier à ce problème, de nombreuses optimisations du processus d'analyse ont déjà été développées et implémentées. Cependant dans le cas de systèmes complexes comme TTA, elles se révèlent insuffisantes. Cet article propose une méthode complémentaire d'optimisation de l'analyse : l'abstraction du modèle lui-même, effectuée avant de débuter le processus d'analyse afin de réduire l'espace des états possibles à parcourir. Ces abstractions sont basées sur une connaissance approfondie du système modélisé et sur une étude de la syntaxe spécifique du formalisme de modélisation : les TSA.

Cet article présente donc, après une introduction du contexte, trois abstraction adaptées au modèle TSA de l'architecture TTA : la réduction des entrelacements, la

fusion d'automates puis la réduction du nombre d'horloges. Ces règles d'abstractions sont illustrées par des expérimentations dans le contexte de la validation d'un service de l'architecture TTA : la réintégration.

2. Contexte

2.1. *TSA - Timed Safety Automata - définition*

Les *Timed Safety Automata* (TSA) (Henzinger *et al.*, 1994) (Bengtsson *et al.*, 2004), sont une extension des automates temporisés classiques (Alur *et al.*, 1994). Les TSA permettent la gestion du temps par l'introduction d'horloges et de contraintes temporelles sur les états et sur les transitions. Les TSA permettent également la gestion des variables. Un modèle TSA est un réseau de plusieurs automates, qui communiquent par variables globales partagées ou par synchronisation synchrone (sur rendez-vous). Ainsi, une transition peut également être associée à la réception ou à l'émission d'un signal de synchronisation. Une définition formelle des TSA est donnée ci-dessous, inspirées de (Bengtsson *et al.*, 2004).

Soit M un modèle TSA, composé d'un réseau de p automates. M est alors la composition parallèle de ces automates : $S = A_0||A_1||...||A_{p-1}$. Soit un ensemble fini C de variables représentant les horloges du système, et V l'ensemble des variables du système. Les *updates* (actions des transitions) sont un ensemble (noté U) d'opérations sur les variables et de réinitialisation des horloges. Une *guard* est un ensemble de contraintes sur les variables et les horloges du système, qui conditionne le tir des transitions. L'ensemble des *guards* est noté $G(C)$. Enfin, la *synchronisation* d'une transition représente ses actions de synchronisation. L'ensemble des signaux de synchronisation du système est noté S, et soient deux transitions s_i et s_j synchronisées sur le signal $s \in S$, on note alors $s_i \cap s_j = s$.

Un automate temporisé étendu A du formalisme TSA est noté $A = (E, l_0, I, T)$:

– E un ensemble fini d'états, et $l_0 \in E$ l'état initial ;

– $I : E \longrightarrow G(C)$ une fonction qui assigne les invariants aux états ;

– $T \in E \times G(C) \times S \times U \times E$ est l'ensemble des transitions ;

L'invariant d'un état $e \in E$ sera noté I_e, et une transition $t \in T$ pourra être notée $t(G_t, s_t, U_t)$, avec $G_t \in G(C)$ sa *guard*, $s_t \in S$ sa *synchronisation* et $U_t \in U$ son *update*. Ses états source et destination seront notés respectivement Src_t et $Dest\_t$. Un exemple d'automate est représenté à gauche de la figure 1. Cet automate possède deux états $E = \{e0, e1\}$, avec $e0 \in E$ l'état initial. L'état $e1$ possède un *invariant* ($x < 20$), contrainte sur l'horloge $x \in C$ du système. La transition entre $e0$ et $e1$ possède une *guard* avec une contrainte temporelle sur l'horloge x, tandis que la transition entre $e1$ et $e0$ possède une *update* (réinitialisation de x à la valeur 10).

Enfin, les TSA offrent deux possibilités de représentation de l'urgence par l'utilisation de deux attributs sur les états : *urgent* et *committed*. Dans les deux cas, le temps

ne pourra pas s'écouler dans les états marqués comme tels. La différence entre *urgent* et *committed* est qu'un état *committed* devra être quitté immédiatement, tandis qu'aucune condition supplémentaire n'est appliquée aux états *urgents*. Cette différence n'est perceptible que lorsque deux transitions sont concurrentes : une transition sortant de la place *committed* sera exécutée en priorité, tandis que celle qui sort d'une place *urgent* n'a aucune priorité par rapport aux autres transitions exécutables au même instant.

2.2. *Processus d'analyse de l'outil UPPAAL*

Le processus d'analyse de l'outil UPPAAL permet la vérification "à la volée" de propriétés exprimées en logique temporelle par un parcours de l'ensemble de l'espace des états du système. Le *model checking* sur les automates temporisés devient décidable en utilisant une relation d'équivalence (Alur *et al.*, 1994) et qui permet une représentation finie du temps continu sous forme de régions et du graphe d'états du système par un graphe des régions. Inspiré de ces travaux, UPPAAL implémente un *model checking* symbolique qui permet de construire le graphe de zones (Möller, 2002, Bengtsson *et al.*, 2004), représentation abstraite du graphe des régions. Les zones permettent de représenter efficacement le comportement temporel du système, sous la forme d'un ensemble de contraintes temporelles sur les horloges du système. L'espace des états du système sera donc représenté sous la forme d'un graphe d'états symboliques : le graphe de zones.

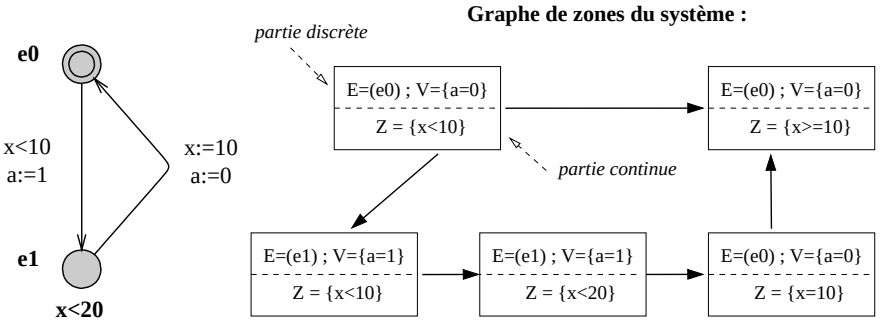

Figure 1. *Exemple d'un modèle TSA et de son graphe de zones*

Un état symbolique (appelé simplement état par la suite) est constitué de deux parties : la partie discrète, non temporisée, et la partie continue. La partie discrète comprend un vecteur $E = (e_0, .., e_{n-1})$ représentant l'état courant de chacun des n automates du système, ainsi qu'un vecteur $V = (v_0, .., v_{m-1})$ représentant les valeurs de ses m variables. La partie continue est une zone Z, c'est-à-dire un ensemble de contraintes sur l'ensemble $C_0 = C \cup \{0\}$ des horloges du système. La figure 1 donne un exemple simple de modèle et son graphe de zones associé. Un état symbolique représente l'ensemble des états du système possédant la même partie discrète et dont la partie continue est inclue dans la zone de cet état.

2.3. *Explosion combinatoire et rédution de la complexité*

L'exhaustivité de la technique de *model checking* induit le problème de l'explosion combinatoire de l'espace d'états du système. Le processus d'analyse est donc souvent limité par la complexité en place mémoire nécessaire pour le stockage du graphe d'états. La place mémoire totale utilisée dépend de deux critères : la place mémoire de stockage d'un état donné, et le nombre d'états total du graphe. La taille mémoire de stockage d'un état dépend des paramètres du systèmes stockés dans cet état : complexité linéaire dans le nombre de variables du système et le nombre d'automates (stockage de l'état de chacun des automates du modèle), et complexité de la partie continue de l'état. Dans le contexte des systèmes modélisés en automates temporisés, l'espace d'états est représentés sur un graphe des régions dont le nombre d'états est exponentiel au nombre d'horloges du système et à la constante maximale présente dans les contraintes sur les horloges (Alur *et al.,* 1994). La représentation symbolique des régions en zones (Möller, 2002, Bengtsson *et al.,* 2004) permet de représenter l'espace des états de façon plus compacte, avec une complexité mémoire de stockage d'une zone dépendante du nombre de contraintes sur les horloges (dans le pire cas polynômial dans le nombre d'horloges, mais en moyenne de complexité moindre). D'autres optimisations existantes et implémentées dans UPPAAL, permettent une amélioration de l'implémentation des états en mémoire, ainsi que du nombre d'états stockés (Larsen *et al.,* 2003) (Behrmann *et al.,* 2002) (Bengtsson, 2001) (Larsen *et al.,* 1997a). L'établissement exact de la complexité du processus d'analyse de UPPAAL est donc très difficile, il reste cependant dépendant des facteurs énoncés précédemment.

Un autre paramètre important de la complexité du processus d'analyse est le nombre d'états du graphe de zones. Dans le contexte des systèmes concurrents, ce paramètre est possède une complexité pire exponentielle dans le nombre de composants du système. La concurrence de ces transitions est une cause importante de la multiplication des états lors de l'entrelacement des transitions (cf section 3). La réduction des entrelacements est un problème connu dans le domaine des systèmes concurrents. De nombreuses techniques dites d'ordre partiel (Wolper *et al.,* 1993) (François *et al.,* 1996), (Ribet *et al.,* 2002) permettent d'éviter le parcours de plusieurs chemins d'exécution lorsqu'ils sont équivalents du point de vue de la propriété à vérifier. Ces techniques ne s'appliquent cependant pas aisément aux systèmes temps réel, et il existe peu de travaux sur les automates temporisés. (Krimm *et al.,* 2000) propose une solution adaptée aux automates temporisés communiquant par buffers. Dans le cadre des automates temporisés communiquant par rendez-vous (comme les TSA), une méthode est proposée dans (Bengtsson *et al.,* 1998) puis (Minea, 1999) basée sur la considération de la désynchronisation des horloges locales des automates en dehors des transitions simultanées (émission/réception d'un signal de synchronisation). Dans le cas de ces dernières, une resynchronisation des régions temporelles est effectuée pour garantir la fiabilité du comportement. Cette techniques est basée sur une sémantique spécifique et nécessite une implémentation spéciale de l'algorithme de *model checking*. De même, certaines techniques d'ordre partiel pourraient dans certaines conditions être appliquées aux systèmes temporisés, au prix toujours d'une

modification importante de l'algorithme d'analyse. De le cadre de la modélisation et la validation en TSA avec l'outil UPPAAL, nous proposons dans cet article des règles d'abstractions appliquées statiquement avant le processus d'analyse, sans modification de l'algorithme de *model checking*. Ces abstractions permettent de réduire les éléments du modèle : réduction du nombre de transitions concurrentes, du nombre d'automates ou du nombre d'horloges, et par là même la complexité de l'analyse.

2.4. *Modèle initial de l'architecture TTA*

La première étape de modélisation d'un système mène à la conception d'un modèle du système proche des spécifications de celui-ci limitant les erreurs de modélisation en concevant un modèle expressif et facilement compréhensible. Par exemple, la conception d'un protocole spécifié par couches se modélisera aisément en associant un automate par couche. Ainsi, chacun des noeuds du modèle initial est modélisé par un ensemble de cinq automates, un par couche (application, *FTlayer* et contrôleur TTP/C), un pour le système d'exploitation (ordonnancement des tâches) et un *bus guardian*. Le reste du système est représenté par trois automates, l'un permettant l'initialisation des variables du modèle et le démarrage synchronisé de tous les automates ; le deuxième permettant la gestion du temps global du modèle et enfin le `MEDL` permettant la gestion de l'ordonnancement global des messages. Pour la validation de TTA, ce modèle est complété par deux automates : un générateur de fautes et un observateur pour la vérification de la borne temporelle. Le modèle initial est donc constitué de $5N + 5$ automates (N étant le nombre de nœuds du système). De même, les horloges du système peuvent être comptabilisées en fonction de N : $2N + 3$ horloges. Le nombre de variables est plus complexe à compter, il dépend du nombre de nœuds mais aussi d'autres paramètres du système. Ces paramètres sont cependant dépendants de l'ordonnancement dans TTP/C, ils sont donc statiques pour un scénario donné, le nombre de variables du modèle initial est donc $k_1 N + k_2$, avec k_1 et k_2 des constantes.

La richesse du modèle initial entraîne une explosion de l'espace des états du système, dont la complexité dépend directement du nombre d'éléments du modèle. La validation du modèle initial atteignait les limites de l'ordinateur (pentium III 500MHz, 500Mo) du point de vue consommation mémoire. Ce modèle va servir de base à l'illustration des règles d'abstractions proposées dans cet article.

3. Réduction des entrelacements

3.1. *Entrelacements : définition*

Les systèmes étudiés dans notre contexte impliquent la modélisation de la concurrence. Celle-ci est représentée par des entrelacements lorsque plusieurs transitions sont tirables simultanément. Ce problème est une des bases du domaine des systèmes parallèles (Mateescu, 1998). La sémantique des entrelacements dans notre contexte est illustrée par un exemple (non temporisé pour simplifier) sur la figure 2 : soit un sys-

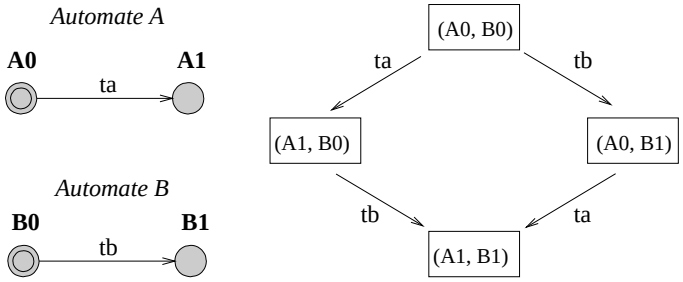

Figure 2. *Automates concurrents et entrelacement de leur espace d'états*

tème (à gauche) composé de deux automates concurrents A et B. Les deux transitions ta et tb sont concurrentes : elles peuvent toutes deux être tirées à partir de l'état initial du système (automates respectivement en $A0$ et $B0$). Le graphe d'états de ce système, donné à droite de la figure 2, comporte donc les deux séquences de transitions ta, tb et tb, ta qui aboutissent alors au même état final $(A1, B1)$, les transitions ta et tb étant indépendantes l'une de l'autre. Deux transitions sont indépendantes (Wolper *et al.*, 1993) si leur ordre d'exécution n'influence pas le comportement du système. Cette situation est typique des entrelacements, et peut être très lourde dans le cadre de processus de validation : $n!$ chemins d'exécution sont générés lorsque n transitions sont concurrentes, alors qu'un seul chemin est nécessaire pour l'analyse de ce système.

3.2. *Détection et réduction des entrelacements*

La réduction des entrelacements s'est révélées dans notre cas indispensable à la terminaison du processus d'analyse. Cette section propose une règle de réduction des entrelacements applicable aux TSA, qui pourrait être la base d'une implémentation de cette technique dans l'outil UPPAAL. En reprenant l'exemple de la figure 2, on pourra définir un **entrelacement** dans un système d'automates temporisés par la conséquence des trois situations suivantes :

– plusieurs automates sont dans un état connu simultanément (au même instant) ; sur la figure 2, ces états sont $A0$ et $B0$; on nommera ces états des états **simultanés**.

– de ces états simultanés partent des transitions **concurrentes**, c'est-à-dire qu'elles peuvent s'exécuter au même instant ; sur la figure 2, ce sont ta et tb ;

– enfin, ces transitions sont **indépendantes**, c'est-à-dire que leur ordre d'exécution n'influence pas le comportement du système.

La première étape de l'optimisation des entrelacements consiste à détecter les états simultanés. Ces états sont difficilement détectables par une analyse statique du modèle. Les TSA permettent cependant la détection d'un sous-ensemble de ces états : ceux qui résultent du tir d'une transition associée à l'émission d'un signal. Il est ainsi possible de déterminer statiquement un ensemble d'états simultanés, que nous appel-

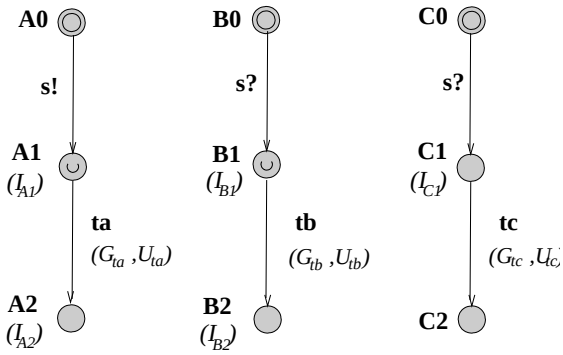

Figure 3. *Exemple de détection de réduction d'un entrelacement*

lerons E_s. Ainsi dans la figure 3, on a $E_s = \{A1, B1, C1\}$ car ces états sont atteints simultanément par le tir des transitions associées à l'émission et à la réception du signal de synchronisation s (ici émis en *broadcast*)

L'étape suivante devient alors la détection des transitions concurrentes. Les transitions tirables à partir de ces états simultanés sont toutes potentiellement concurrentes. Un élément spécifique des TSA permet de détecter aisément un sous-ensemble de transitions concurrentes : l'urgence. En effet, les états marqués *urgent* ou *committed* doivent être quittés immédiatement, sans écoulement du temps. Ainsi, si un sous-ensemble $E_{s,u}$ de E_s est marqué par l'un de ces mots-clé, l'ensemble des transitions sortantes de ces états (noté $T_{s,u}$) sera alors un ensemble de transitions concurrentes. Dans l'exemple de la figure 3, les états $A1$ et $B1$ sont marqués *urgent*, les transitions ta et tb sont alors détectées comme concurrentes et on a $E_{s,u} = \{A1, B1\}$ et $T_{s,u} = \{ta, tb\}$. L'état $C1$ est "normal" (non urgent), il est donc exclu et sera traité normalement. Une extension pourrait l'inclure dans le traitement si tc est tirable immédiatement, et donc en partie concurrente avec les autres.

Il reste enfin à tester l'indépendance des transitions de $T_{s,u}$. La relation d'indépendance de deux transitions dépend de la sémantique du formalisme utilisé. Pour les TSA, nous la définirons comme suit : pour que deux transitions soient indépendantes, il faut que les variables et les horloges modifiées lors du tir de l'une des transitions, c'est-à-dire modifiées dans son *update*, ne concernent pas celles impliquées dans les conditions de tir de l'autre, c'est-à-dire dans sa *guard* ou dans les *invariants* de ses états source et destination. Ainsi, en considérant les deux transitions ta et tb de l'exemple précédent, il faut que :

$$U_{ta} \cap (G_{tb} \cup I_{B1} \cup I_{B2}) = \emptyset \qquad et \qquad U_{tb} \cap (G_{ta} \cup I_{A1} \cup I_{A2}) = \emptyset$$

Il est possible de généraliser l'indépendance de deux transitions ti et tj :

$$U_{ti} \cap (G_{tj} \cup I_{Src\_tj} \cup I_{Dest\_tj}) = \emptyset$$

$$U_{tj} \cap (G_{ti} \cup I_{Src\_ti} \cup I_{Dest\_ti}) = \emptyset$$

Une fois l'indépendance des transitions prouvée, une situation d'entrelacement a alors été détectée. Il est dans ce cas nécessaire d'intégrer un traitement de cette situation afin d'éviter l'exploration de tous les chemins concurrents. La technique proposée ici devra être intégrée directement à l'algorithme de *model checking*. Une première phase d'analyse statique du modèle permettrait la détection des états simultanés et des transitions concurrentes indépendantes. Ces transitions devront être marquées par un marquage spécifique. Il serait alors nécessaire d'intégrer un traitement spécial de ces transitions dans l'algorithme d'exploration du graphe d'états du système, permettant de réduire l'entrelacement à un seul chemin en exécutant ces transitions dans un ordre arbitraire.

Dans notre contexte, aucune modification n'a été apportée à l'algorithme d'analyse. Les situations d'entrelacements ont été détectées grâce à une connaissance approfondie du modèle, et ont été évitées en utilisant la priorité intrinsèque des états *committed* sur les états *urgent*. Ainsi, la situation d'entrelacement illustrée dans la figure 3 a été traitée en modifiant l'un des états $A1$ ou $B1$ en le rendant *committed*. Les transitions ta et tb ne sont alors plus concurrentes, celle dont l'état source est *committed* étant prioritaire.

Cette technique permet ainsi de supprimer un certain nombre d'entrelacements, dans le cas particulier d'états marqués *urgent* à la suite d'une transition de synchronisation. Cette situation, qui de prime abord semble très spécifique, est un cas relativement fréquent dans la modélisation des systèmes concurrents temps-réel en TSA. Elle peut par exemple traduire la modélisation d'une série d'actions instantanées, ou d'une action complexe, suite à un signal précis (un top d'horloge ou un évènement extérieur). Il est de plus possible d'étendre ces définitions à des cas plus larges. Par exemple, en rajoutant à l'ensemble des transitions concurrentes celles issus d'états non marqués *urgent*. Une deuxième idée serait de considérer des suites de transitions concurrentes. Par exemple, en reprenant la figure 3, on peut imaginer que les états $A2$ et $B2$ soient également marqués *urgent*. Ces états seront alors également simultanés. Une analyse plus poussée de cette situation d'entrelacement, spécifique au formalisme TSA, serait intéressante. Elle devra être accompagnée par une étude de l'efficacité de cette abstraction sur des modèles de système différents, dans l'optique d'une évaluation de la pertinence du développement de cette abstraction au sein de l'outil UPPAAL.

4. Fusion d'automates

4.1. *Résultats et analyse*

Trois fusions différentes ont été effectuées à partir du même modèle initial de l'architecture avec $N = 4$ nœuds (cf. section 2.4). Afin de pouvoir étudier l'impact direct de ces fusions, elles ont été modélisées séparément en différentes versions du modèle. Les performances en temps d'exécution et en place mémoire du processus de

validation ont été mesurées pour chacune de ces versions. Elles sont reportées dans le tableau 1, ainsi que le nombre d'automates N_{AT} du modèle considéré.

Version	N_{AT}	**Temps**	**Mémoire**
Modèle de référence	25	$25.00s$	19156KB
Fusion des couches applicative et *FTlayer*	21	$17.32s$	17524KB
Fusion de l'initialisation et du temps global	24	$18.49s$	17588KB
Fusion du générateur de fautes et de l'observateur	24	$19.86s$	17548KB
Les 3 fusions cumulées	19	$16.49s$	17428KB

Tableau 1. *Résultats pour la fusion*

Le tableau 1 montre que les différentes fusions d'automates permettent une amélioration des performances de l'analyse en temps d'exécution et en espace mémoire. Les trois fusions cumulées permettent dans ce cas (modèle de l'architecture à 4 nœuds) un gain mémoire de 9% et un gain de temps de 34%. Ces gains correspondent à l'économie de stockage de l'état de l'automate supprimé dans le vecteur d'états, ainsi qu'aux opérations effectuées sur ce vecteur (cf. section 2.3). De plus, la fusion d'automates à ce stade du modèle permet un gain dans le calcul de la composition parallèle du modèle. Les gains associés aux différentes fusions ne sont cependant pas identiques : la fusion des automates `Appli` et `FTlayer` est plus efficace que les autres. Cette différence vient du fait que ces automates sont des automates spécifiques aux nœuds du système, ce qui répercute la fusion sur tous les nœuds (N fois le gain). Les autres fusions ne seront par contre bénéfiques qu'une seule fois. Cet écart est d'autant plus grand que le nombre de nœuds modélisés est important.

4.2. *Définition formelle*

Les fusions d'automates effectuées sont basées sur deux concepts : la fusion d'automates dits "à états stables", et la fusion d'automates séquentiels.

4.2.1. *Automates à états stables*

Un automate à état stable est un automate ne contenant qu'un seul état, dans lequel il revient après chaque tir de transition, comme les automates A et B représentés sur la figure 4. Il est alors possible de fusionner ces états, E_A et E_B sur la figure, en un seul état stable E associé à toutes les transitions des automates A et B. On obtient alors un seul automate fusionné, représenté à droite de la figure 4. L'état E de ce nouvel automate correspond aux deux états E_A et E_B, et ses transitions sont l'ensemble des transitions des deux autres automates. Le cas particulier de deux transitions synchronisées (t_2 et t_3 sur la figure) peut être traité en fusionnant les deux transitions, c'est-à-dire en fusionnant leurs *updates*.

Cette transformation respecte les règles suivantes :

– Soit $S = A||B||P_0...||P_n$ un modèle TSA,

– Soient $A = (E_A, e_A, I_A, T_A)$ et $B = (E_B, e_B, I_B, T_B)$ deux automates de S,

– Si $|E_A| = |E_B| = 1$, c'est-à-dire $E_A = \{e_A\}$ (resp. B), $I_A = \emptyset$ (resp. B) et $\{s_a \cap s_b = \emptyset,\ \forall t_a \in T_A,\ \forall t_b \in T_B\}$, c'est-à-dire si A et B sont à états stables sans synchronisation,

– alors les deux automates A et B peuvent être fusionnés en un automate $C = (E_C, e_C, I_C, T_C)$ avec $E_C = \{e_C\}$, $T_C = T_A \cup T_B$ et $I_C = I_A \cup I_B$.

– Si $\exists t_a \in T_A, t_b \in T_B$ tels que $s_a \cap s_b = s$, c'est-à-dire deux transitions de A et B sont synchronisées, alors on fusionne leurs *updates* en une seule transition : $t_c \in T_C$ telle que $u_c = u_a \cup u_b$ et $s_c = \emptyset$.

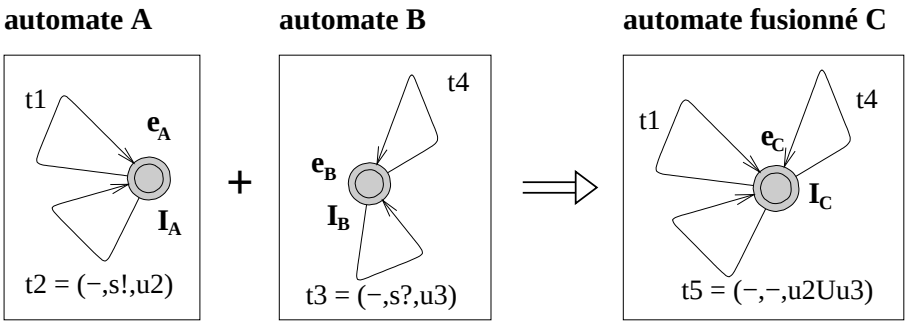

Figure 4. *Fusion d'automates à états stables*

4.2.2. *Automates séquentiels*

Une deuxième sorte de fusion est la fusion d'automates séquentiels. Deux automates sont considérés séquentiels lorsque l'exécution de l'un des deux se termine lorsque celle de l'autre commence, comme montré informellement figure 5. Le problème réside dans la détection de cette séquentialité.

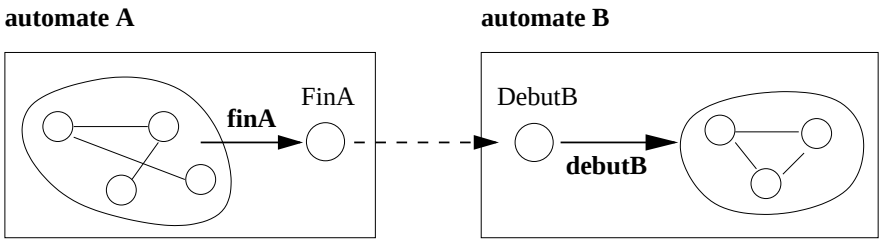

Figure 5. *Fusion d'automates séquentiels*

Un cas possible est la détection de la séquentialité par l'utilisation de la synchronisation : l'exécution de l'automate B est initiée par un signal envoyé par la dernière

transition de l'automate A. Cette situation est simple à détecter, mais risque d'être relativement peu fréquente dans un modèle.

Une deuxième méthode de détection de la séquentialité est basée sur la connaissance approfondie du fonctionnement du système : la faute générée par l'automate générateur de fautes déclenche un mécanisme dans l'automate du contrôleur (ici la réintégration) dont le début déclenche l'exécution de l'observateur. Sur la figure 5, les pointillés peuvent représenter un comportement intermédiaire, tant que la transition `finA` est toujours avant `debutB`. Cette séquentialité est par contre plus difficile à détecter sur un modèle complexe, et difficile à prouver. Une technique pourrait cependant être utilisée dans ce but : la tau-bisimulation (Pinchinat, 1993, Tripakis *et al.*, 2001). Cette méthode permet de masquer les éléments du modèle considérés comme non pertinents, tout en conservant les informations indispensables à l'analyse du modèle. Dans le contexte de la détection d'automates séquentiels, seul l'ordre d'exécution des transitions est nécessaire, le délai écoulé entre ces transitions peut par contre être abstrait. Les transitions qui correspondent à des transitions non pertinentes sont abstraites et remplacées par une transition particulière τ. Une transition τ peut correspondre soit à l'écoulement d'un délai temporel (tau-bisimulation temporelle, (Tripakis *et al.*, 2001)), soit au tir d'une transition non indispensables (tau-bisimulation ou équivalence observationnelle (Pinchinat, 1993)) à la détermination de la séquentialité de deux transitions. La figure 6 montre un système équivalent par tau-bisimulation au système de la figure 5, c'est-à-dire à la composition parallèle des automates de la figure 5. Ces deux systèmes sont équivalents sur le plan comportemental (non temporel), ils sont dits bisimilaires.

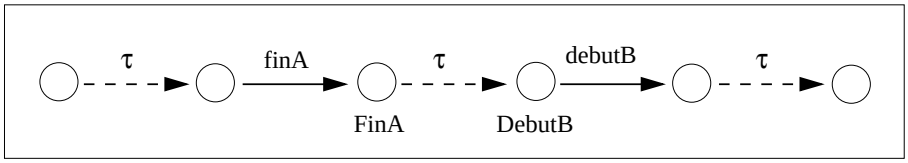

Figure 6. *Automate tau-bisimilaire au système de la fi gure 5*

Les techniques de bisimulation ne seront pas développées plus dans cet article. L'important est de savoir qu'il est possible de déterminer, étant donné deux transitions t et t', qu'elles sont séquentielles : on notera $t << t'$ si t est exécutée dans tous les cas avant t'. Dans notre contexte, la fusion d'automates peut s'effectuer s'il existe un automate avec une transition de fin qui s'exécutera toujours avant la transition de début d'un autre automate. En définissant une transition de fin comme une transition aboutissant dans un état final, et une transition de début comme une transition partant de l'état initial.

– Soit $S = P_0||...||P_n$ un modèle TSA,

– Soit Sf l'ensemble des automates possédant une transition de fin, avec T_f l'ensemble de ces transitions,

– Soit Sd l'ensemble des automates possédant une transition de début, avec T_d l'ensemble de ces transitions,

– Si $\exists t_i \in T_f, t_j \in T_d$ tels que $t_i << t_j$ (et $i \neq j$),

– Alors les automates P_i et P_j sont séquentiels et peuvent être fusionnés en un automate P_{ij} équivalent à la somme des deux automates, avec simplement l'état final de p_i fusionné à l'état initial de P_j.

5. Réduction d'horloges

5.1. *Réduction par analyse statique*

Un autre type d'abstraction considéré concerne le paramètre du modèle le plus influent pour la complexité : le nombre d'horloges du système. Plusieurs formes d'abstractions ont été appliquées au modèle initial pour réduire ce nombre. Le premier type de réduction d'horloge est lié à l'une des fusions précédentes : la fusion des automates générateur de fautes et observateur. Ces deux automates possédant chacun une horloge, leur fusion simple donne un automate `Fault` avec deux horloges. Cependant, la séquentialité de ces deux automates implique que les horloges seront utilisées l'une après l'autre. Il est ainsi possible de n'en utiliser qu'une sur les deux, et de supprimer l'autre. Cette abstraction permet un gain en temps et en espace mémoire en fonction du nombre d'horloges (N_C), comme le montre le tableau 2 (ligne 3) : gain de 24% en temps et plus de 9% en mémoire pour l'architecture à 4 nœuds. Ce gain correspond à la réduction du nombre de contraintes dans les zones du graphes (celles correspondant à l'horloge supprimée), et permet un gain de place direct ainsi qu'une réduction du temps de calcul des opérations sur les zones.

Version	N_C	**Temps d'exécution**	**Place mémoire**
Modèle initial	11	$25.00s$	19156KB
Réduction pour les *bus guardians*	8	$11.30s$	11988KB
Réduction pour le générateur de fautes et l'observateur	10	$18.97s$	17364KB

Tableau 2. *Résultats pour la réduction d'horloges*

Cette première méthode de réduction d'horloge a été effectuée "à la main". Elle est inspirée de (Daws *et al.*, 1996), qui présente des algorithmes de détection des horloges redondantes ou inutiles par une analyse statique du modèle. Cette méthode peut se révéler précieuse dans le contexte de la modèlisation d'un système distribué, dont les différents composants sont souvent modélisés séparément avec chacun leur horloge locale, ou lorsque les composants possèdent différentes couches temporisés ou différents timers. Cette modélisation génère de nombreuses horloges différentes, qui peuvent se révéler redondantes. Cette méthode à été étendue à la gestion des réseaux d'automates et implémentée dans l'outil KRONOS (Daws *et al.*, 1998) et dans un

module externe de l'outil UPPAAL (Behrmann *et al.*, 2002). Une analyse statique basée sur des principes similaires pourrait également être appliquée pour la réduction du nombre de variables du système.

5.2. *Réduction basée sur des hypothèses spécifiques*

Le second type de réduction des horloges est basée sur une connaissance précise du système et sur l'établissement d'hypothèses réalistes. Ces abstractions sont alors spécifiques au modèle étudié et ne peuvent pas être généralisées. Par exemple, l'une des abstractions appliquées dans notre cas consiste en la suppression des horloges locales de *bus guardians* pour les remplacer par une horloge globale. La notion locale de temps est donc abstraite au profit d'une notion de temps global qui suppose que toutes les visions locales sont identiques. Cela suppose donc que les horloges locales sont synchronisées entre elles. Dans TTA ces horloges sont censées être synchronisées, par un service de synchronisation accompagné d'un service de gestion de la fenêtre d'accès au bus. Pour être fiable, cette abstraction doit donc reposer sur l'hypothèse de la validation de ces services, ce qui est le cas pour TTA : les services ont été validés formellement par preuve de théorèmes (Pfeifer *et al.*, 1999).

L'influence de cette dernière abstraction sur les performances du processus de validation est donnée dans le tableau 2, ligne 2. De la même façon que pour les fusions, cette abstraction est plus efficace que la réduction d'horloges de l'automate `Fault`. En effet, la fusion des horloges locales des automates `BG` permet de remplacer N horloges (N est le nombre de nœuds du système) par une seule horloge globale. Cette abstraction fournie ainsi dans le cas d'une architecture à 4 nœuds un gain de 55% en temps d'exécution, et de 37% en place mémoire, et peut monter a plus de 70% de gain lorsque le nombre de nœuds augmente.

6. Conclusion

Cet article présente une méthode d'optimisation du processus de validation temporelle de l'architecture TTA. L'utilisation de la technique de *model checking* pour l'analyse exhaustive du système entraîne en effet un problème d'explosion combinatoire. La méthode proposée ici a pour but l'abstraction du modèle du système avant la phase d'analyse afin d'en réduire la complexité. Cet article propose trois types d'abstractions spécifiques au formalisme de modélisation TSA implémenté dans l'outil UPPAAL : la réduction des entrelacements, la fusion d'automates et la réduction du nombre d'horloges. Des expérimentations ont montrées que ces abstractions se révélaient efficaces, offrant des gains en temps et en espace mémoire pouvant aller jusqu'à plus de 50%.

Les règles d'abstractions définies ici sont spécifiques au formalisme TSA, ainsi qu'au système modélisé. Or les TSA et UPPAAL sont de plus en plus utilisés dans le domaine de la modélisation et la validation de systèmes distribués temps réel (Lindahl *et al.*, 2001, Bengtsson *et al.*, 2002). La généralisation de ces abstractions à la

validation d'autres systèmes est donc un domaine intéressant à considéré. Certaines abstractions reposent sur des hypothèses liées au système et à son environnement. Ces abstractions sont donc difficilement, voir impossibles à généraliser au cas d'autres systèmes. Cependant, l'étude des abstractions proposée dans cet article peut fournir des pistes à l'automatisation de la plupart de ces abstractions, et pourrait offrir une aide efficace à la conception et la validation temporelle des systèmes temps-réel avec UPPAAL. Dans ce cadre, il s'agira également d'étudier la complexité de cette étape d'abstraction automatique du modèle, qui sera à rajouter à celle du processus d'analyse.

7. Bibliographie

Alur R., Dill D. L., « A theory of timed automata », *Journal of Theoretical Computer Science*, vol. 126, n° 2, p. 183-235, 1994.

Behrmann G., Bengtsson J., David A., Larsen K. G., Pettersson P., Yi W., « Uppaal Implementation Secrets », *Proc. of the 7th Int. Symp. on Formal Techniques in Real-Time and Fault-Tolerant Systems*, Springer-Verlag, Oldenburg, Germany, p. 3-22, September, 2002.

Bengtsson J., Reducing Memory Usage in Symbolic State-Space Exploration for Timed Systems, Technical Report n° 2001-009, Uppsala universitet, Institutionen för informationsteknologi, May, 2001.

Bengtsson J., Griffi oen W. O. D., Kristoffersen K. J., Larsen K. G., Larsson F., Pettersson P., Yi W., « Automated Analysis of an Audio Control Protocol Using UPPAAL », *Journal of Logic and Algebraic Programming*, vol. 52–53, p. 163-181, July-August, 2002.

Bengtsson J., Jonsson B., Lilius J., Yi W., « Partial Order Reductions for Timed Systems », *"Proceedings, Ninth International Conference on Concurrency Theory"*, vol. 1466 of *"Lecture Notes in Computer Science"*, "Springer-Verlag", p. 485-500, 1998.

Bengtsson J., Yi W., « Timed Automata : Semantics, Algorithms and Tools », *in* , W. Reisig, , G. Rozenberg (eds), *Lecture Notes on Concurrency and Petri Nets*, LNCS vol 3098, Springer–Verlag, 2004.

Daws C., Tripakis S., « Model checking of real-time reachability properties using abstractions », *In Tools and Algorithms for the Construction and Analysis of Systems TACAS'98*, number 1384 in *LNCS*, Lisbon, Portugal, 1998.

Daws C., Yovine S., « Reducing the number of clock variables of timed automata », *in* , I. C. S. Press (ed.), *Proc. of the 17th IEEE Real Time Systems Symposium, RTSS'96*, Washington, DC, USA, December, 1996.

François V., Pierre A., François M., « Covering Step Graph », *Proc. 17th International Conference in Application and Theory of Petri Nets (ICATPN'96)*, vol. 1091, Springer-Verlag, Osaka, Japan, p. 516-535, June, 1996.

Godary K., Validation temporelle de réseaux embarqués critiques et fi ables pour l'automobile, PhD thesis, INSA Lyon, Novembre, 2004.

Henzinger T. A., Nicollin X., Sifakis J., Yovine S., « Symbolic model checking for real time systems », *Journal of Information and Computation*, vol. 111, n° 2, p. 193-244, 1994.

Kopetz H., « The Time-Triggered Architecture », *Proc. of the First International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '98)*, April, 1998. Kyoto, Japan.

Krimm J.-P., Mounier L., « Compositional State Space Generation with Partial Order Reductions for Asynchronous Communicating Systems », *TACAS '00 : Proceedings of the 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, Springer-Verlag, London, UK, p. 266-282, 2000.

Larsen K. G., Larsson F., Pettersson P., Yi W., « Compact Data Structures and State-Space Reduction for Model-Checking Real-Time Systems », *Real-Time Syst.*, vol. 25, n° 2-3, p. 255-275, 2003.

Larsen K., Larsson F., Pettersson P., Yi W., « Efficient Verification of Real-Time Systems : Compact Data Structure and State-Space Reduction », *Proc. of the 18th ICCC Real Time Systems Symposiums (RTSS'97)*, San Francisco, California, USA, p. 14-24, December 3-5, 1997a.

Larsen K., Pettersson P., Yi W., « UPPAAL in a Nutshell », *Int. Journal on Software Tools for Technology Transfer*, vol. 1, n° 1, p. 134-152, december, 1997b.

Lindahl M., Pettersson P., Yi W., « Formal Design and Analysis of a Gearbox Controller », *Springer International Journal of Software Tools for Technology Transfer (STTT)*, vol. 3, n° 3, p. 353-368, 2001.

Mateescu R., Vérification des propriétés temporelles des programmes parallèles, PhD thesis, Institut National Plytechnique de Grenoble, 1998.

Minea M., Partial Order Reduction for Verification of Timed Systems, PhD thesis, Carnegie Mellon University, Pittsburg, PA 15213, December, 1999.

Möller O., Structure and Hierarchy in Real-Time Systems, Phd thesis, BRICS PhD school, University of Aarhus, February, 2002.

Pfeifer H., Schwier D., von Henke F. W., « Formal Verification for Time-Triggered Clock Synchronization », *in* , C. B. Weinstock, , J. R. (eds.) (eds), *Dependable Computing for Critical Applications 7*, vol. 12 of *Dependable Computing and Fault-Tolerant Systems*, IEEE Computer Society, p. 207-226, January, 1999.

Pinchinat S., Des bisimulations pour la sémantique des systèmes réactifs, PhD thesis, Institut National Polytechnique de Grenoble, 1993.

Ribet P.-O., Vernadat F., Berthomieu B., « On Combining the Persistent Sets Method with the Covering Steps Graph Method », *FORTE '02 : Proceedings of the 22nd IFIP WG 6.1 International Conference Houston on Formal Techniques for Networked and Distributed Systems*, Springer-Verlag, London, UK, p. 344-359, 2002.

Tripakis S., Yovine S., « Analysis of Timed Systems Using Time-Abstracting Bisimulations », *Formal Methods in Systems Design*, vol. 18, n° 1, p. 25-68, 2001.

TTT, *Time-Triggered Protocol TTP/C, High-Level Specification Document - edition 1.4.3*. November, 2003, http ://www.ttech.com.

Wolper P., Godefroid P., « Partial-Order Methods for Temporal Verification », *CONCUR '93 : Proceedings of the 4th International Conference on Concurrency Theory*, Springer-Verlag, London, UK, p. 233-246, 1993.