Ministère de l'Enseignement Superieur et de la Recherche Scientifique
Université des Sciences et de la Technologie HOUARI Boumediene
U.S.T.H.B

Résumé de mémoire de Magister

en Informatique

Option

Informatique Mobile

Thème

Vérification distribuée des systèmes temps réel

Présenté par

Fatiha OUAZAR

Devant le jury composé de :
Président : ....
Rapporteur : ....
Examinateur : ....
Invité : ....

Promotion 2006 - 2007

Table des matières

Résumé

Les systèmes temps réel sont des systèmes dotés d'un comportement qui est contraint par le temps, un tel système doit réagir correctement avec son environnement non seulement au regards des informations échangées mais également aux instants auxquels ces interactions se réalisent. En effet la vérification de tels systèmes nécéssite un modèle temporel formel pour la modélisation tel que les réseaux de Petri temporels (Time Petri Net,TPN), une logique temporelle temporisée pour exprimer les propriétés à vérifier sur ce modèle et un modèle checker pour decider si la propriété est vérifiée sur le système ou non. Cependant la vérification par model-checking est souvent confrontée au problème de l'explosion combinatoire due à la taille du système qui est exponentielle en nombre d'état et en nombre d'horoges du système.

Pour pallier à ce problème,plusieurs approches peuvent être suivies,telles que la représentation des systèmes par des structures de données plus compactes (BDD), les symétries,la distribution...

Dans ce travail nous nous interéssons à la technique de vérification à la volée du model-checking basée sur une platforme distribuée, en utilisant la méthode des zones sur les réseaux de Petri temporels pour la génération de l'espace d'état. Ceci permet d'augmenter la mémoire principale de stockage et de distribuer la charge de calcul sur une grappe d'ordinateurs.

Mots clés : réseaux de Petri temporel, model-checking, model-checking à la volé, vérification, vérification distribuée, zones.

1. Introduction générale

Les systèmes temps réel se distinguent des autres systèmes informatiques par le fait qu'ils soient soumis à des contraintes temporelles. Il est donc impératif d'avoir des outils et techniques de modélisation et de vérification qui puissent garder un haut degré de performance et de correction pour un système temps réel. Il ne faut pas perdre de vue que dans un tel système le temps représente un facteur important. Par conséquent il doit être pris en considération d'une manière explicite durant la phase de modélisation et de vérification **[Gsh02]**.

Il existe plusieurs outils et techniques de modélisation des systèmes (files d'attente, chaines de Markov,...), dont plusieurs ont pu prouver leurs puissance et performances.

Parmi les outils courament utilisés pour modéliser les systèmes, les réseaux de Petri (RdP) occupent une place priviligiée. Le formalisme des RdP a été introduit dans les années soixante par C.A.Petri **[Pet62].** Les réseaux de Petri sont un outil trés utilisé car ils ont montré une grande puissance d'expression et de calcul pour la modélisation et la vérification de nombreux et différents types de systèmes, et ils fournissent des méthodes d'analyse avec lesquelles on peut vérifier certaines propriétés de systèmes (bornitude, vivacité,...).

Cependant, les RdPs ne prennent pas en considération le caractère temporel des systèmes. Ils se contentent d'effectuer une modélisation et une vérification du comportement logique d'un système hors son contexte temporel. Afin de palier à cela, et pour pouvoir profiter de la puissance des RdPs, des extensions temporelles des RdPs ont été proposées. Le but principal de telles extensions est de pouvoir effectuer une modélisation et une vérification du système sur le plan logique du séquencement des évènement, et surtout sur le plan temporel primordial dans le cadre de la représentation des systèmes temps-réel. Parmis ces extensions, nous trouvons les réseaux de Petri temporels (Time Petri Nets).

L'idée fondamentale des réseaux de Petri temporels est d'associer des intervalles de temps [a,b] à chaque transition. Une transition ne peut être tirée que si elle est sensibilisée de façon continue au moins a unités de temps. Si une transition est sensibilisée continuellement pendant b unités de temps , elle doit être tirée immédiatement car elle a atteint son délai maximum de sensibilisation. De plus,la durée de tir d'une transition est toujours nulle **[Boy01].**

La large utilisation des réseaux de Petri temporels a été concrétisée par la conception de plusieurs outils qui les implémentent. On retrouve par exemple les outils ***ROMÉO, TINA***, etc.

Les propriétés générales telles que la bornitude, la vivacité, le non blocage ... renseignent le modélisateur sur le comportement général du système, celles-ci doivent être complétées par l'analyse des propriétés spécifiques du système modélisé.

Plusieurs axes de recherche ont été développés visant à mettre en place des outils permettant une description formelle des propriétés spécifiques des systèmes informatiques. La logique temporelle initiée par EMERSON & AL **[Eme, Pra96],** est un outil formel qui procure une syntaxe sûre, précise et sans ambiguïté des propriétés qualitatives et quantitatives.

La logique temporelle se distingue selon deux axes:

— Soit on présente l'ensemble des évènement comme un arbre où les successeurs d'un état sont obtenus par les instances d'évènements possibles en cet état; on parle alors de logique temporelle arborescente (CTL Computation Tree Logic).
— Soit on considère l'ensemble des éxécutions comme des séquences distinctes; on parle alors de logique temporelle linéaire (LTL Linear Temporal Logic)

Pour pouvoir exprimer des propriétés spécifiques des systèmes temps-réel c-à-d des propriétés quantitatives, des extensions temporelles des différentes logiques temporelles ont été proposées, telle que la logique TCTL (Timed CTL) auquelle nous nous interéssons.

Aprés la description des propriétés spécifiques, viens l'étape vérification de ces propriétés sur le système.

Cette étape consiste à verifier si celui-ci satisfait ses spécifications, c'est-à-dire de s'assurer que les propriétés qu'il possède correspondent aux propriétés attendues.

Parmi les techniques de vérification: les approches par model-checking, se sont largement développées ces dernières années **[Sch01].** Cela suppose le comportement du système à vérifier soit modélisé et que la propriété de correction attendue énoncée sous la forme d'une formule de logique temporelle, en suite on peut utiliser un model-checker afin de vérifier si le modèle satisfait ou non la propriété.

Néanmoins, la vérification et la validation des systèmes temps réel souffrent

de l'éxplosion combinatoire liée à la taille du système qui est exponentielle en nombre d'états du système et en nombre d'horloge de ce système.

Plusieurs approches peuvent etre suivies pour pallier à ce problème, parmi ces approche, les techniques de réduction telles que la vérification à la volée, qui vérifie les propriétés en meme temps que l'exploration, la vérification modulaire **[Pet05]**, la représentation symbolique telle que les diagrammes de decision, nous citons les approches distribuées, aussi celles qui évitent de présenter tous les états du système en prenant en compte certaines propriétés du modèle telles que la symétrie **[Jor99]**.

Nous nous intéressons à la vérification des systèmes temps réel en utilisant des modèles temporisés (RdP temporels et automates temporisés) pour la modélisation de ce type de systèmes, et d'effectuer une vérification sur une plate-forme distribuée pour éviter le problème de l'explosion combinatoire.

Nous utilisons une technique basée sur l'idée de faire coopérer plusieurs stations (site) ayant chacune son propre processeur et sa propre mémoire. Cette technique permet d'effectuer des traitements parallèles et d'exploiter les espaces mémoire de l'ensemble des machines.

La technique offre deux principaux avantages:

- pouvoir augmenter la taille des modèles étudiés, en partitionnant l'éspace d'états entre les mémoires des différentes stations.

- minimiser le temps global de la procédure de vérification en distribuant les algorithmes de génération et de vérification.

Dans ce contexte, nous allons étudier les deux étapes principales de la vérification:

- La première étape consiste à générer de façon parallèle et distribuée l'espace d'états du système à vérifier.

- La seconde étape consiste à vérifier à la volée par des algorithmes distribués les propriétés spécifiques du système modélisé exprimées en logique temporelle TCTL .

Nous nous interessons surtout à la vérification de la sous classe de propriétés TCTL vérifiable à la volée sur les réseaux de Petri temporels sur une plate forme distribuée.

Dans le reste de ce document nous présentons le principe des réseaux de Petri temporels dans la section 2, la technique de verification par model-checking dans la section 3, la section 4 portera sur notre travail qui est la génération et la verification distribuées et à la volé des propriétés TCTL, la section 5 expose les résultats obtenus et nous terminerons par une conclusion générale et quelques perspectives.

2. Les réseaux de Petri temporels

Les réseaux de Petri temporel sont utilisés pour modéliser les systèmes où le temps joue un rôle critique. On les obtient en associant, à chacune des transitions, un intervalle de temps [a,b]. Ce type de réseaux exprime nativement la notion de Délai. En explicitant les débuts et fins d'actions, ils peuvent exprimer la notion de Durée.

Un réseau de Petri temporel est défini comme étant un réseau de Petri ordinaire où on associe à chaque transition deux dates : min et max ($0 \leq \text{min} \leq$ max, max est éventuellement infini) **[Ber 01].** L'intervalle ainsi obtenu, [min, max], est relatif à la date θ où la transition t a été sensibilisée. Dans ce cas, la transition t ne peut être franchie ou tirée avant la date min+θ, et elle doit être tirée au plus tard à la date max+θ, sauf bien sûr si elle est désensibilisée par une autre transition. La durée de tir d'une transition est supposée nulle **[Bbd 03].**

Formellement un réseau de Petri temporel (TPN) est un tuple (P,T,Pré,Post,M_0,IS) où (P,T,Pré,Post,M_0) est un réseau de Petri ordinaire.

IS : $T \rightarrow Q^+ \times (Q^+ \cup \infty)$ est une fonction d'intervalle statique qui associe à chaque transition t du réseau un intervalle à bornes rationnelles IS (t)= [α,β], avec $0 \leq \alpha \leq \beta$ (β peut être infini).

La fonction IS associe à chaque transition du réseau un intervalle à bornes rationnelles IS(t). La plus petite borne de cet intervalle est appelée date de tir au plus tôt de t (notée SMin (t)). La plus grande borne est appelée date de tir au plus tard de t (notée SMax(t)).

Dans un réseau temporel, franchir une transition t sensibilisée n'est permis que dans l'intervalle de tir qui lui est associé. Cet intervalle est relatif à la date de sensibilisation de la transition. Initialement, et si t est sensibilisée par le marquage initial, l'intervalle de tir de t correspond à son intervalle statique IS (t). Mais avec l'évolution du réseau, l'intervalle de temps associé à cette transition va également évoluer : il est décalé, vers l'origine des temps, d'une quantité égale à la durée écoulée depuis la sensibilisation de la transition. Cet intervalle « dynamique » est exprimé avec une application **I**. Elle fait correspondre à chaque transition un nouvel intervalle de temps **I**(t), dans lequel elle peut être tirée. Les bornes de cet intervalle sont désignées comme étant les dates au plus tôt DMin (t), et au plus tard DMax (t) de franchissement de t **[Bbd 03]**.

La sémantique des TPN peut être donnée en terme de systèmes de transitions temporisés (TTS) **[Lar95]** qui sont des systèmes de transitions classiques avec deux types d'étiquettes: des étiquettes discrètes pour les évènements et les étiquettes réelles positives pour l'écoulement du temps.

La sémantique d'un TPN T= (P, T, Pré, Post, M_0, **IS**) est un système de transitions temporisé ST=(Q, q_0,→) avec:

- Q = $\mathbb{N}^P \times (\mathbb{R}_{\geq 0})^n$,avec n =|T |; v $\in (\mathbb{R}_{\geq 0})^n$ est une valuation telle que la valeur v_i représente le temps écoulé depuis le dernier instant où la transition t_i a été sensibilisée.

- q_0 =(M_0, 0).

- →∈Q× (T ∪$\mathbb{R}_{\geq 0}$) × Q est la relation de transition composée des transitions discrètes et continues suivantes:

- la relation de transition discrète est définie comme suit:

$$\forall t_i \in T,(M,v)\rightarrow^{t_i}(M',v')\text{ssi}\begin{cases} M \geq Pre(\bullet,t_i) \wedge M' = M - Pre(\bullet,t) + Post(\bullet,t) \\ Dmin(t_i) \leq v_i \leq Dmax(t_i) \\ v'_k = \begin{cases} 0 & si \uparrow enabled(t_k, M, t_i) \\ v_k & sinon \end{cases} \end{cases}$$

- La relation de transition continue est définie comme suit:

$$\forall d \in \mathbb{R}_{\geq 0},(M,v)\rightarrow^{e(d)}(M,v')\text{ssi}\begin{cases} \forall k \in [1..n], (M \geq Pre(\bullet,t_k) \Rightarrow v'_k \leq Dmax(t_k) \\ v' = v + d \end{cases}$$

↑enabled (t_k, M, t_i) est vrai si t_k est sensibilisée par le tir de la transition t_i à partir du marquage M, et faux dans le cas contraire. Une transition t_k est nouvellement sensibilisée par le tir d'une transition t_i à partir du marquage M si " elle n'est pas sensibilisée par M- Pre (p, t_i) et elle est sensibilisée par M'= M- Pre(p, t_i)+ Post(p, t_i)" **[Ber91]**. De plus la transition t_k est nouvellement sensibilisée par son propre tir **[Fra03]**. Formellement:

↑enabled (t_k, M, t_i) = (M- Pre(•, t_i)+ Post(•, t_i) ≥ t_k) ∧ (M- Pre (•, t_i)< t_k)∨ (t_k=t_i).

Une exécution d'un TPN *T* est un chemin dans S_T à partir de q_0. L'ensemble des exécutions de *T* est notée [*T*]. Un marquage M est accessible dans *T* ssi il existe

une exécution $(M_0, 0) \Rightarrow^{\sigma} (M, v)$ dans [T]. L'ensemble des marquages accessibles de T est reach (T). Si l'ensemble reach (T) est fini, le réseau T est alors borné.

L'état d'un réseau temporel est un couple E = (M,I), où M est un marquage du réseau et I est la fonction intervalle de tir qui :

- Associe, à chaque transition t sensibilisée, son intervalle de tir.
- Elle assigne l'intervalle vide pour les transitions non sensibilisées.

L'état initial est défini par $E_0 = (M_0,I_0)$ où M_0 est le marquage initial. I_0 est la fonction qui associe à une transition sensibilisée t son intervalle de tir statique **IS** (t).

La définition des états permet de déterminer les conditions de franchissement d'une transition t. t est franchissable dans un état E (M,I) à une date relative θ si :

1) t est sensibilisée par le marquage M (condition standard des réseaux de Petri ordinaires: $M \geq Pre(t)$).

2) θ appartient à l'intervalle de tir de t, I (t) = [DMin,DMax].

3) Aucune autre transition n'est tirable avant t. En d'autres termes:

$\forall k \in T, M \geq pre(k) \Rightarrow \theta \leq DMax(k)$ **[Ber 01]**.

La première condition est celle autorisant le tir dans les RdPs ordinaires, la deuxième et troisième condition résultent de l'obligation de tirer les transitions dans leurs intervalles de tir.

Le tir d'une transition t, à une date θ depuis l'état E = (M,I), conduit vers un état E' = (M',I') déterminé comme suit :

1) M' = M - Pré (t) + Post (t) (comme dans les réseaux de Petri ordinaires).

2) Le nouvel intervalle de tir I'(k), pour toute transition t_k est défini par :

a) Si t_k n'est pas sensibilisée par M' alors I' (t_k) = Φ.

b) Si t_k est sensibilisée par M et n'est pas en conflit avec t alors :

I'(t_k) = [Max (0, DMin (t) - θ), DMax (t) -θ], si DSmax(t_k) est fini.

I' (t_k) = [Max (0, DMin (t) - θ), ∞ [, sinon.

c)I' (t_k) = IS (t_k) sinon.

Autrement dit, les transitions non sensibilisées par le nouveau marquage reçoivent des intervalles vides. Les transitions nouvellement sensibilisées par M' (i.e. non sensibilisées par M-pré (t) et sensibilisées par M') reçoivent leurs intervalles de tir statiques. Les autres transitions (i .e celles qui étaient sensibilisées par M et n'ont pas été désensibilisées par le tir de t) voient leurs intervalles de tir décalés, vers l'origine du temps, de la valeur θ.

Deux transition t et t' sont en conflit pour un marquage M si toutes deux sont sensibilisées par M, mais, pour au moins une place p, M(p) < Pre(p,t) + Pre(p,t').

2.1. La méthode des classes d'états

Une classe d'état est constituée par l'ensemble des états accessibles depuis l'état initial, en tirant tous les échéanciers réalisables (s,u) ayant comme support la séquence s. Une classe d'état est caractérisée par cette séquence de tir s, et par le marquage M accessible par cette séquence à partir du marquage initial, on la note C = (M,D), où :

- M est le marquage accessible depuis M_0 par la séquence s (commun à tous les états de la classe).
- D est le domaine de tir de la classe, défini par l'union des intervalles de tir des états constituant la classe.

2.1.1. Calcul du graphe des classes

Le graphe des classes est calculé par application successive de la fonction de calcul des successeurs à partir de la classe d'état initiale C_0= (M_0, D_0) avec D_0= $\{\alpha_k \leq \theta_k \leq \beta_k \ / \ t_k \in \text{enabled}(M_0)\}$. La convergence est assurée en mémorisant les classes d'états générées et en arrêtant l'analyse des successeurs d'une classe si celle-ci a déjà été précédemment analysée. les détails de cette méthode sont dans le mémoire

3. Le model-checking

Les techniques de vérification par model-checking reposent sur le calcul et l'exploration de l'espace d'états du modèle. Dans le cas des systèmes temporisés en temps dense, cet espace d'états est infini et des techniques d'abstraction doivent être utilisées afin d'obtenir une représentation finie de l'espace d'états Les automates temporisés et les réseaux de Petri temporels constituent deux modèles trés appropriés à la modélisation de systèmes temps réel. Nous nous intéressons plus particulièrement aux réseaux de Petri temporels. Ils constituent un modèle permettant d'exprimer facilement les comportements de systèmes parallèles et distribués (synchronisation,...). De plus comparativement aux automates temporisés, les techniques de model-checking et en particulier le model-checking de propriétés temporelles quantitatives ont été peu étudiées.

3.1. La méthode des classes d'états

La méthode d'analyse présentée dans la section précédente (Les TPN:Time Petri Nets) permet pour les réseaux de Petri temporels une analyse d'accessibilité semblable à celle permise pour les réseaux de Petri par la technique du graphe des marquages. Cette technique a été utilisée dans de nombreux travaux universitaires ou industriels, et a été intégrée à plusieurs outils d'analyse de systèmes.

Les limites intrinsèques de la méthode ne doivent toutefois pas être perdues de vue. Une première limite est qu'il ne peut être énoncé de condition nécessaire et suffisante pour la propriété de bornitude pour les réseaux temporels, une seconde limite est que le nombre de classes d'états d'un réseau de Petri temporel peut être trés grand ce qui pose le problème de l'explosion combinatoire du nombre d'états.

3.2. La méthode des zones

Le graphe des classes donne l'ensemble des marquages du réseau et préserve son langage non temporisé. C'est-à-dire qu'il fournit l'ordre d'apparition des évènements (tir de transition). Il permet donc de vérifier des propriétés de type logique temporelle linéaire. Cependant, nous pouvons remarquer que si la classe C' est le successeur par la transition t de la classe C, cela n'implique pas forcement que tous les états de C ont un successeur par t de C' mais uniquement qu'il existe au moins un état de C qui possède un successeur par t dans C'. C'est pourquoi nous ne pouvons pas vérifier des propriétés de type logique temporelle arborescente (CTL) sur le graphe des classes. De plus le graphe des classes ne fournit pas d'informations quantitatives sur les instants auxquels les tirs des transitions se sont produits.

Le graphe des classes ne permettent pas dans l'état d'envisager la vérification de propriétés temporelles quantitatives, nous nous sommes intéressés à une autre technique d'exploration de l'espace d'états: la méthode des zones proposée dans **[GRR03]**.

3.2.1. Définition d'une zone

Soit C un ensemble d'horloges. Une zone est un ensemble convexe de valuations d'horloges représentant un ensemble de contraintes de la forme $x_i - x_j \leq c_{ij}$, $x_i \leq c_{i0}$, $x_i \geq c_{0j}$. Avec $c_{ij}, c_{i0}, c_{0j} \in \mathbb{Z}$.

De même que pour les classes d'états de Berthomieu , une zone peut être encodée par une DBM. Une zone est une forme particulière de polyèdre qui peut être représentée par une DBM efficacement (Difference Bounded Matrice), en utilisant une horloge supplémentaire x_0 qui est toujours égal à 0. Cette représentation (DBM) permet une représentation plus facile et des algorithmes moins complexes par rapport à un polyèdre général.

Si Z est une zone, nous notons z_{ij} la contrainte sur la différence de l'horloge

x_i-x_j c-à-d. Z représente la conjonction de contraintes atomiques: $\forall$ x_i, $x_j \in$ C, (x_i-$x_j \leq z_{ij}$).

3.2.2. Définition d'une DBM

Une DBM est une structure de données qui permet de représenter efficacement les zones.

Pour avoir une forme unifiée de contraintes d'horloges on introduit une horloge de référence x_0 avec la valeur 0 (zéro). Alors $C_0 = C \cup \{x_0\}$.

Pour construire le DBM qui représente une zone D, on commence par énumérer toutes les horloges qui appartiennent à C_0 à partir de 0,...,n et l'indexe de l'horloge ajoutée est x_0 . Chaque horloge est indiquée par une ligne de la matrice (DBM), la ligne est utilisée pour sauvegarder la borne inférieure de la différence entre une horloge et les autres horloges, ainsi la colonne correspondante est utilisée pour les bornes supérieures. Les éléments de la matrice sont calculés selon trois étapes :

— Pour toute contrainte de D: x_i-$x_j \leq$ n , D_{ij}=(n, $\leq$).
— Pour toute différence d'horloges x_i-x_j qui n'est pas bornée dans D , $D_{ij} = \infty$, tel que ∞ est une valeur spéciale qui indique qu'il n y a pas de borne.
— Ajouter les contraintes implicites telle que toutes les horloges sont positives, c-à-d: x_0 - $x_i \leq 0$, et telle que la différence entre une horloge et elle-même est toujours 0, c-à-d: x_i - $x_i \leq 0$.

3.3. Etats symboliques:

De la même manière que dans le graphe des classes, on donne une définition d'un regroupement d'états de la sémantique d'un réseau de Petri temporel.

3.3.1. Définition d'un état symbolique

Un état symbolique Z d'un réseau de Petri temporel ***N*** est un couple **Z** =(M,Z) où M est un marquage et Z un polyèdre convexe de $\mathbb{R}^{+|enabled(t)|}$ appelé zone.
Une zone est caractérisée par un ensemble de contraintes de la forme:

$$\begin{cases} -z_{ij} \leq x_i - x_j \leq z_{ij} \\ -a_i \leq x_i \leq b_i \end{cases} \qquad \forall t_i, t_j \in \text{enabled(M)}$$

Les variables x_i représentent les valuations des horloges associées aux transitions du réseau de Petri temporel ***N***.

À la différence d'une classe d'état, un état symbolique représente les valuations réelles des horloges du réseau de Petri temporel. L'idée sous jacente est de pouvoir utiliser les informations temporelles contenues dans la zone pour déterminer les séquences de transitions possibles en fonction de la date d'entrée dans l'état symbolique.

Une transition est dite franchissables dans l'état symbolique (M,Z) s'il existe une valuation v $\in$ Z telle que la transition est franchissable. L'état successeur est alors calculé en appliquant la sémantique des réseaux de Petri temporels.

On a étendu les deux types de transitions de la sémantique d'un réseau de Petri temporel (transitions discrètes et transitions continues) aux états symboliques.

3.3.2. Successeur discret d'un état symbolique

Soit **Z** = (M, Z) un état symbolique et t_f une transition franchissable. L'état symbolique successeur de **Z** par franchissement d'une transition t_f , noté $post_{t_f}$(**Z**), est défini par:

$$\text{post}_{tf}(\mathbf{Z})=\begin{cases} M - Pre(\bullet, t) + Post(t, \bullet) \\ Z \cap \{x_f \geq \alpha_f\})[x_r \leftarrow 0] \end{cases}$$

où x_r={$x_i \in$X |↑enabled(t_i,M, t_f)}.

ainsi $post_{t_f}$(***Z***) *contient tous les états de la sémantique accessible à partir de* ***Z*** *en franchissant la transition* t_f.

3.3.3. Successeur temporel d'un état symbolique

Soit Z = (M, Z) un état symbolique. L'état symbolique obtenu par écoulement de temps, noté Post (***Z***), est défini par:

Post (***Z***) =(M, Z $\cap$ { $x_i \leq \beta_i$ | $t_i \in$ enabled(M)}).

Post (***Z***) *contient tous les états de la sémantique accessibles à partir de Z en laissant le temps s'écouler*.

$t_i \in$ enabled(M): t_i sensibilisée par le marquage M.

En utilisant les états symbolique on peut calculer les états atteignables du système en appliquant l'algorithme suivant:

Algorithm 1 Algorithme de calcul du graphe des zones:

$\mathbf{Z}_0$ = (M, Z_0)
Waiting $\leftarrow$ {$\mathbf{Z}_0$}
Passed $\leftarrow\emptyset$
Tant que Waiting $\neq \emptyset$ faire
  Z = pop(Waiting) // prendre le premier élément de la pile
  Si **Z** $\notin$Passed alors
    Pour tout t $\in$ enabled (**Z**) faire
      Z′ = post (post$_t$(**Z**)) // successeur temporel
      Waiting $\leftarrow$ Waiting $\cup$ { **Z**′}
    Fin pour
    Passed$\leftarrow$ Passed $\cup$ { **Z**}
  Fin si
Fin Algorithme

3.4. Technique de vérification par model-checking

Le but de la vérification est d'assurer qu'un système satisfait un certain nombre de propriétés. Pour cela il est nécessaire de modéliser formellement le comportement de ce système : Rappelons que le système est modélisé par un RdP temporel, le graphe des zones décrit l'évolution de ce système. Une fois le système décrit, et les propriétés souhaitées spécifiées en logique temporelle quantitatives (TCTL) l'algorithme dit model-checking permet de répondre automatiquement à la question :«est-ce que le système satisfait les propriétés souhaitées ?».

3.4.1. Model-Checking à la volée

La vérification à la volée consiste à vérifier les propriétés attendues du système durant la génération de l'espace d'états. Cette technique n'explore qu'une partie de l'espace d'états ce qui permet de réduire le problème de l'explosion combinatoire.

Pour pouvoir utiliser un model checker il faut modéliser formellement le système et spécifier les propriétés à vérifier à l'aide d'une logique temporelle adéquate.

La logique temporelle temporisée qui permet d'exprimer les propriétés quantitatives d'un système temps réel est la logique temporelle temporisée TCTL.

3.4.2. CTL et la temporisation (TCTL)

La logique CTL ne permet de décrire que des propriétés qualitatives. Une façon d'introduire le temps dans cette logique est de borner la portée des opérateurs temporels. TCTL (Timed Computation Tree Logique) est une logique temporisée,

extension de CTL, qui permet d'exprimer des propriétés sur des systèmes temps réels dits temporisés c.à.d des systèmes dont le comportement est conditionné par le passage du temps.

3.4.3. Syntaxe de TCTL

Soit AP un ensemble de propositions atomiques; les formules de TCTL sont définies par les règles suivantes :

S1 : chaque proposition atomique P est une formule de TCTL.

S2 : si f et g sont des formules de TCTL alors (f$\vee$g) et ($\neg$f) sont des formules de TCTL .

S3 : si f est une formule de chemin alors E f U$\sim$c g et A fU$\sim$c g sont des formules de TCTL, avec $\sim\in${<,<=, =,>,>=} et c$\in \mathbb{N}$. c peut être un intervalle de temps .

3.4.4. Sémantique de TCTL

Pour une séquence d'exécution (chemin) π = (s_0 ,s_1 ,...) du système, on note π(t) l'état du système à l'instant t sur cette séquence π et suc(s) l'ensemble des chemins issus de s.

La relation de satisfaction des formules de TCTL est définie par :

s$\models$ p $\Leftrightarrow$ p$\in$L(s)

s$\models$ f$\vee$g $\Leftrightarrow$s$\models$ f ou s$\models$ g

s$\models$ $\neg$ f $\Leftrightarrow\neg$ s$\models$ f

s$\models$ **E** f **U**$\sim$c g $\Leftrightarrow\exists$ $\pi \in$ suc(s); $\exists$ t$\sim$c tel que π(t) $\models$ g et $\forall$t'(0$\leq$ t'<t), π(t')$\models$ f

s$\models$ **A** f**U**$\sim$c g$\Leftrightarrow$ $\forall$ $\pi\in$suc(s); $\exists$ t$\sim$c tel que π(t) $\models$ g et $\forall$t'(0<=t'<t), π(t') $\models$ f

On définit également quelques abréviations classiques:

-**EF**$\sim$c f pour **E**(true **U**$\sim$c f), signifie qu'il existe une exécution pour laquelle f devient vraie avant l'instant c (potentialité bornée).

-**AF**$\sim$c f pour **A**(true **U**$\sim$c f);

-**EG**$\sim$c f pour $\neg$**AF**$\sim$c $\neg$f , signifie qu'il existe une exécution pour laquelle f reste vraie jusqu'à l'instant c (quasi_invariance borné).

-**AG**$\sim$c f pour $\neg$**EF**$\sim$c$\neg$f ;

On peut également utiliser des intervalles de temps pour la spécification des propriétés.

4. Génération distribuée de l'espace d'états

Le problème de la vérification par model-checking est l'explosion combinatoire de l'espace d'états, afin d'éviter ce problème plusieurs techniques peuvent être utilisées. Nous nous intéressons à l'approche distribuée qui permet de faire

coopérer plusieurs machines pour générer l'espace d'états et réaliser la vérification; dans ce cas, l'espace d'états et la charge de calcul sont repartis entre les différentes machines.

L'étude que nous venons de faire, permet de repartir les différents états d'un système temps réel et d'assurer la vérification par la méthode du model-checking à la volée d'une classe de propriétés exprimées en logique temporelle TCTL.

Dans cette section nous nous intéressons à la génération distribuée de l'espace d'états en se basant sur l'algorithme des zones et à la vérification des propriétés exprimées en TCTL sur les réseaux de Petri temporels.

4.1. Génération distribuée de l'espace d'états

Plusieurs outils ont été développés dans le cas de la génération distribuée de l'espace d'états d'un modèle (RdP, AT,...) **[Bou07] [Bhv01]** sur plusieurs stations de travail (machines); partitionner cet ensemble d'états sur différentes machines peut se faire en utilisant une fonction de hachage H: V $\rightarrow$ S tel que:
V est l'ensemble de tous les états du modèle.
S = {0,.., N-1} est l'ensemble des identifiant de machines (stations).
L'approche que nous avons utilisée pour la distribution est inspirée de **[Bhv01]**.

Étant donné un réseau de Petri temporel qui modélise un système temps réel. La première étape à faire dans le cadre d'une vérification par le model-checking est la construction de l'espace d'états à partir de ce modèle.

L'approche de construction consiste à explorer les différents états que l'on peut atteindre à partir des états initiaux. Cette étape est appelée génération d'espace d'états.Afin de construire rapidement l'espace d'états du système à vérifier, nous distribuons la tache d'exploration sur plusieurs machines (un réseau de stations). Dans ce cas, chaque machine est responsable pour la génération d'une partie de l'ensemble des états du système.

La génération distribuée est alors effectuée par plusieurs machines. Dans une telle opération, chaque machine génère une partie de l'espace d'états en calculant les successeurs des états qu'elle détient. Une fonction de hachage est utilisée pour distribuer l'ensemble des états entre les différentes machines. Une machine peut envoyer alors, selon la fonction d'attribution (de hachage), un état à une autre machine. L'opération est initiée par la machine qui détient l'état initial.

On distingue alors deux type d'arcs: arcs locaux qui relient les états de la même station; et les arcs traversants qui relient deux états qui appartiennent à deux dif-

férentes stations, ce dernier type permet la transmission de messages entre stations.

4.1.1. Algorithme de distribution

Nous proposons un algorithme inspiré de **[Bhv01]**. Cet algorithme permet une génération distribuée de l'espace d'états.

On considère un ensemble de N machines: w_0, w_1,..., w_{N-1} et un système représenté par un réseau de Petri temporel *N*. soit S_0 son état initial tel que S_0= (M_0, Z_0) avec M_0 est le marquage initial et Z_0 la zone initiale. Et une fonction succ de succession permettant de calculer pour un état quelconque l'ensemble de ses états successeurs.

Chaque machine wi explore un ensemble d'états Si. Elle construit alors une partie {S_i, T_i} du graphe des zones. L'ensemble Si est calculé à partir d'une fonction de hachage qui permet d'attribuer les états aux différentes machines.

Chaque machine wi stocke dans sa mémoire locale les informations concernant les états et les transitions de sa partie {S_i, T_i}.

Les états visités et explorés, par une machine wi, sont stockés dans des ensembles disjoints : waiting (états visites non encore explorés) et passed (états explorés: on a déjà généré leurs états successeurs).

Une machine wi peut envoyer et recevoir des messages par invocation des primitives de communication. Il est à noter qu'il y a deux principaux types de messages. Le premier type sert à échanger les états et les transitions (et les zones) (messages de données). L'autre sert pour la terminaison (message de contrôle).

Le calcul est initié par une machine initiatrice. Cette machine doit avoir un index qui correspond à h (S_0). Elle explore alors l'état initial S_0= (M_0, Z_0).

L'algorithme est composé d'une boucle principale, qui assure les différentes actions permettant de calculer tous les états atteignables. Ce même algorithme est exécuté par toutes les machines.

L'algorithme distribué d'exploration est le suivant :

Algorithm 2 Algorithme de la machine i

```
Début
passed:= ∅
termi:=faux
  Si h(S0) = i alors
    waiting:= S0
  Fsi
 Tant que ¬termi faire
  Si waiting ≠ ∅ alors
    Prendre (S) de waiting
    waiting:= waiting \ S /// S= (M, Z)
    passed := passed ∪ {S}
    Pour tout S' succ(S) faire
     Si h(S') = i alors
       passed := passed ∪ {S'}
     Sinon envoyer S' à   h(S')
     Fsi
    FinPour
  Sinon
    termi:= vrai
  Fsi
 Fait
Fin algorithme
```

4.1.2. Gestion de la terminaison:

Le principe utilisé pour la détection de la terminaison est le suivant:

La vérification à la volée consiste à effectuer la vérification au moment de la génération des configurations et cette méthode permet d'arrêter l'analyse dès que la valeur de vérité de la formule est connue. Donc si une machine détecte la valeur de vérité de la formule, elle envoie un message de type END_CHECK au coordinateur, ce dernier diffuse un message d'arrêt à toutes les machines pour arrêter le calcul.

4.2. Vérification d'une partie de TPN-TCTL à la volée

Une vérification à la volée explore l'espace d'états d'un modèle en même temps que la propriété est vérifiée. À la différence des méthodes générant la totalité de l'espace d'états, cette méthode permet d'arrêter l'analyse dès que la valeur de vérité de la propriété est connue. Cette méthode a montré son efficacité pour l'analyse de systèmes temps réel, son efficacité a été prouvée sur les automates

temporisés avec des outils tel qu'UPPAAL bien que la classe de propriétés vérifiables soit moindre que pour les autre méthode de vérification, **[Gar05]** des études récentes ont montré que dans la plupart des cas, elles étaient suffisantes pour l'analyse des systèmes.

Nous proposons dans cette partie une solution basée sur la méthode du calcul distribué de l'espace d'états (zones) (présentée en dessous), l'idée générale pour la vérification à la volée de propriétés TCTL est : utiliser le graphe des zones pour calculer de manière distribuée l'espace d'états du système et vérifier la propriété au fur et à mesure du calcul (technique de vérification à la volée).

Par la suite nous présentons les algorithmes qui permettent de vérifier les propriétés TCTL en parallèle avec la génération distribuée de l'espace d'états.

La génération distribuée de l'espace d'états permet de repartir les états entre les différentes machines qui permet donc de pallier au problème de l'explosion combinatoire de l'espace d'états dans le cas de systèmes temps réel de grande taille. Ce qui implique la distribution des algorithmes de vérification.

La vérification distribuée consiste à repartir les taches de vérification entre plusieurs machines c-à-d faire coopérer plusieurs machines pour exécuter les algorithmes de vérification.

Dans ce qui suit nous présentons les algorithmes de vérification d'un sous ensemble de propriétés quantitatives exprimées en logique temporelle TCTL. De même nous nous intéressons à une vérification à la volée c-à-d : vérifier au moment de la génération distribuée ce qui permet d'arrêter la génération dès que la propriété sera vérifiée.

4.3. Les algorithmes

4.3.1. Le model-checking de $\exists\varphi\mathbf{U}_I\psi$

Le but de l'algorithme est de détecter une exécution (s_0,v_0),...,(s_n,v_n) telle que pour tous les états (s_i,v_i) $\models\varphi$ et (s_n,v_n) $\models\psi$ dans l'intervalle de temps spécifié I.

Algorithm 3 Algorithme *check EU(s, φ, ψ, I)*

Visited := $\emptyset$
s_0 = post(M_0, **0**) /// **0** = Z_0
retourner *check* EU_{aux}*(s, φ, ψ, I)*

Algorithm 4 Algorithme *check* $EU_{aux}(s, \varphi, \psi, I)$

(M, Z):= s
si M $\models \psi \wedge Z \cap I \neq \emptyset$ *alors*
retourner true
diffuser (terminaison) /propriété vérifiée alors arrêt de la génération/
sinon si M $\neg \models \varphi$ *alors*
retourner false
diffuser(terminaison) /propriété non vérifiée /
sinon
Z':= Z $\cap$ *[0, Imax)*
Si Z' = $\emptyset$ *alors*
retourner false
diffuser(terminaison) /propriété non vérifiée /
Sinon
Visited := visited $\cup$ *{s}*
Pour tout t $\in$ *firable (M, Z') faire*
s' := post(post$_t$*(M, Z')) /successeur temporel du successeur discret/*
si h(s')=i alors
si s'$\notin$ *visited alors*
si check $EU_{aux}(s', \varphi, \psi, I)$ *alors*
retourner true
diffuser(terminaison) /propriété vérifiée /
fsi
fsi
sinon
envoyer(check $EU_{aux}(s', \varphi, \psi, I)$ *à h(s')*
fsi
Fin pour
retourner false
diffuser (terminaison) // il n'ya pas de successeurs qui verifient ψ
Fsi
Fsi
Falgorithme

4.3.2. Le model-checking de $\exists \mathbf{G}_I \varphi$

Algorithm 5 check $\exists G(s, \varphi, I)$

Visited := $\emptyset$
s_0 = post(M_0, **0**) /// **0** = Z_0
retourner *check* $EG_{aux}(s, \varphi, I)$

Algorithm 6 *check EG_{aux}(s, φ, I)*

(M, Z):= s
si M $\neg \models \varphi$ alors
 retourner false
 diffuser (terminaison) //propriété non vérifiée //
sinon
 Visited := visited $\cup$ {s}
 Si I$\subseteq$ Z $\cap$ I alors
  retourner true
  diffuser (terminaison) //propriété vérifiée //
 sinon // chercher les successeurs s'ils vérifient φ
  Pour tout t $\in$ firable (s') faire
   s' := post(post$_t$(s)) / successeur temporel du successeur discret/
   si s'$\notin$ visited alors
    si h(s')=i alors
     si M si check EGaux(s',φ,I) alors
      retourner true
      diffuser (terminaison) //propriété vérifiée //
      Fsi
     Sinon envoyer(EGaux(s', φ, I) à h(s')
     Fsi
   fsi
  Fpour
 retourner false //pas de successeurs qui vérifient la propriete//
 diffuser (terminaison) //propriété non vérifiée alors arrêt de la génération//
 Fsi
Fsi
Falgorithme

4.3.3. Le model-checking de $\forall\varphi\mathbf{U}_I\psi$

Soit Z = (M, Z) un état symbolique pour lequel on veut tester la propriété $\forall\varphi\mathbf{U}_I\psi$.

Intuitivement, on peut itérativement décider de sa valeur de vérité de la façon suivante :

— S'il existe des évaluations de z plus grande que Imax alors la propriété est fausse.
— Si M $\models \psi$ et que toutes les évaluations de l'horloge z de Z sont dans I alors la propriété est vraie pour cet état.
— Si M $\neg \models \varphi$ alors la propriété est fausse.
— on calcule les successeurs temporels de Z .
— on supprime tous les états tels que ψ est vraie et z$\in$I et on note (M, Z') ce nouvel état symbolique. Ces états sont supprimés car ils vérifient tous la propriété : on a seulement besoin de tester les états qui n'ont pas encore validés ψ pour des valuations z <I_{min} .
— on calcule les successeurs par le franchissement de transitions.

Algorithm 7 check **AU(s,φ,ψ, I)**

Visited := $\emptyset$
prefix:= $\emptyset$
s_0 = post(M_0, **0**) /// **0** = Z_0/successeur temporel/
retourner *check* AU_{aux}*(s,* φ,ψ *I)*

Algorithm 8 check **AU(s,φ,ψ, I)**

Z' := Z
(M, Z):= s
$Z = Z_{(0,I_{min})} \cup Z_{(I_{min},I_{max})} \cup Z_{(I_{max},inf)}$
si M $\models \psi$ *alors*
si $Z_{(I_{max},inf)} \neq \emptyset$ *alors*
retourner false
diffuser (terminaison) //propriété non vérifiée /
Fsi
Z':= Z $\cap$ *[0,* I_{max}*)*
sinon si M $\neg \models \varphi$ *alors*
retourner false
diffuser (terminaison) //propriété non vérifiée //
sinon
Z' := timeNext(Z) // successeur temporel
Visited := visited $\cup$ *{s}*
Pour tout t $\in$ *firable (M, Z') faire*
Prefix:= prefix $\cup$ *s*
s' := discretNext(M, Z',t) //successeur discret
si h(s')=i alors
si s'$\in$ *prefix alors*
retourner false
diffuser (terminaison) //propriété non vérifiée //
Fsi
Si s'$\notin$ *visited alors*
si not check AUaux(s', φ*,* ψ*, I) alors*
retourner false
diffuser (terminaison) //propriété non vérifiée //
Fsi
Fsi
prefix := prefix $\setminus$ *s*
Sinon envoyer (check AUaux(s', φ*,* ψ*, I) à h(s')*
Fin pour
retourner true
diffuser (terminaison) //propriété vérifiée //
Fsi
Falgorithme

Nous avons présenté ici les algorithmes de vérification des propriétés $\exists\varphi\mathbf{U}_I\psi$, $\exists\mathbf{G}_I\varphi$, et $\forall\varphi\mathbf{U}_I\psi$. Les autres propriétés peuvent être déduites en utilisant les définitions suivantes:

Le model-checking de $\exists\mathbf{F}_I\varphi$

Par définition $\exists\mathbf{F}_I\varphi\equiv$ True$\mathrm{U}_I\varphi$.

Le model-checking de $\forall\ \mathbf{G}_I\varphi$

Par définition $\forall\ \mathbf{G}_I\varphi\equiv\ \neg\exists\mathrm{F}_I\varphi$.

Le model-checking de $\forall\mathbf{F}_I\varphi$
Par définition $\forall\mathrm{G}_I\varphi\equiv\forall$true $\mathrm{U}_I\varphi\equiv\neg\exists G_I\neg\varphi$

5. Applications et resultats

Afin d'analyser les performances et la complexité des algorithmes proposés dans la section précédante, nous avons procédé à l'implementation de ces derniers en utilisant le language C++. Ce dernier offre un mécanisme simple et efficace, permettant la communication et les envois des messages entre les processus, qui est le principe des *sockets*.

Dans cette section, nous allons discuter cette implementation en présentant les resultats des tests concernant les différents algorithmes distribués.

5.1. Génération distribuée du graphe des zones

Soit un réseau de Petri temporel TPN. Les configurations (états) successeures possibles pour une configuration (M,Z) peuvent être calculées à partir de la sémantique des réseaux de Petri temporels (transitions discrêtes et transitions continues). L'application de l'algorithme présenté dans la section 4 permet de générer d'une façon distribuées l'ensemble des configurations (graphe des zones) d'un réseau de Petri temporel. Cependant l'attribution des configurations aux machines se fait à l'aide d'une fonction de hachage comme sera présenté dans le paragraphe suivant.

5.2. Fonction d'attribution des configurations aux machines

Nous considérons une fonction de répartition h qui associe à chaque marquage d'une configuration (composé du marquage et de la zone) un entier compris entre

0 et N-1, N est le nombre de machine. Nous calculons la fonction de hachage de la façon suivante:

— Une fonction coef qui génère de façon aléatoire des entiers.
— On calcule pour chaque marquage un numéro n en fonction de la fonction aléatoire qui n'est pas unique.

5.3. Tests et résultats

Nous avons testé les algorithmes sur 3 machines ayant 512 mo de RAM chacune. Les algorithmes implémentés et testés sont les suivants:

— L'algorithme distribué de génération de l'espace d'états
— Les algorithmes distribués du model-checking de propriétés TCTL vérifiables à la volée.

5.3.1. Test de la génération et de la vérification distribuées de configurations (espace d'etats en utilisant les zones)

Concernant la génération de l'espace d'états, nous avons considéré les résultats suivants:

1. La répartition des états se fait selon la fonction de hashage, le nombre de configurations dans chaque machine est calculé. une fonction de hachage particulière a été utilisée pour le problème des philosophes qui a été modélisé par les TPN; afin d'avoir une repartition equilibrée.
2. Les performances de l'algorithme: l'avantage de la distribution est d'éviter le problème de l'explosion combinatoire de l'espace d'états, et comme nous avons utilisé une vérification à la volée qui permet d'arrêter la génération dès que la valeur de vérité de la formule est connue, en plus sur une platforme distribuée donc le temps d'éxécution est réduit énormément et le problème de l'explosion combinatoire est complètement évité ainsi q'un équilibre spacial et temporel est remarquable.

5.3.2. Résultats de la vérification distribués des formules TCTL

Le principe de vérification utilisé est la vérification à la volée, donc dès que la valeur de vérité de la formule TCTL introduite est connue, la génération est arrêtée par tous les processus. Le test de la vérité des formules TCTL à la volée et dans le cas distribué montre des bonnes performances. Les machines coopèrent pour vérifier la formule. Une fois l'une des machines termine la vérification elle envoie au coordinateur le résultat et ce dernier diffuse un message d'arrêt à toutes les machines qui arrêtent le calcul. Les résultats sont présentés comme dans l'exemple suivant:

Les algorithmes ont été appliqués aux problème des philosophes qui est un problème trés connu en informatique. Un philosophe peut avoir les états suivants:

— penser pendant une période de temps (intervalle de temps associé à cet évènement)
— manger pendant une période de temps déterminée afin de permettre aux autres de manger.
— une fonction de hachage non aléatoire a été utilisé pour le problème des philosophes afin d'avoir une distribution équilibrée.

Le tableau suivant montre les premiers résultats, en analysant les résultats du tableau on déduit que l'aspect temporel affecte le résultat de la vérification:

Formule TCTL	*Nbre de philo*	*nbre de conf*	*Résultat*	*Temps(s)*	*NB proc*
$EtrueU_{[0\,5]}p_7$	*6*	*12*	*formule vérifiée*	*<0.01*	*3*
$Ep_0U_{[0\,2]}p_{70}$	*6*	*7*	*formule verifiée*	*<0.01*	*3*
$Ap_0U_{[0\,2]}p_{70}$	*6*	*1*	*formule non verifiée*	*<0.01*	*3*
$Ep_0U_{[0\,2]}p_{70}$	*8*	*81*	*formule vérifiée*	*<0.06*	*3*
$AtrueU_{[0\,8]}p_{22}$	*8*	*15*	*formule vérifiée*	*<0.01*	*3*
$AtrueU_{[0\,8]}p_{22}$	*10*	*22*	*formule vérifiée*	*<0.02*	*3*
$Ap1U_{[0\,8]}p_{29}$	*10*	*1*	*formule vérifiée*	*<0.00*	*3*
$Ep_0U_{[0\,2]}p_{10}$	*10*	*79*	*formule vérifiée*	*<0.04*	*3*
$Ep_0U_{[4\,5]}p_{13}$	*10*	*21*	*formule non verifiée*	*<0.01*	*3*
$Ep_0U_{[0\,2]}p_{13}$	*10*	*67*	*formule vérifiée*	*<0.04*	*3*
$Ep_0U_{[0\,2]}p_{13}$	*10*	*55*	*formule vérifiée*	*<0.08*	*1*
$Ep_0U_{[0\,5]}p_{19}$	*20*	*1039*	*formule vérifiée*	*0.46*	*3*
$Ep_0U_{[0\,5]}p_{58}$	*20*	*1012*	*formule vérifiée*	*0.46*	*3*
$Ep_0U_{[0\,5]}p_{19}$	*20*	*246*	*formule vérifiée*	*0.14*	*1*
$Ep_0U_{[0\,5]}p_{13}$	*20*	*189*	*formule vérifiée*	*0.33*	*1*
$Ep_0U_{[0\,5]}p_{13}$	*20*	*1324*	*formule vérifiée*	*0.62*	*3*
$Ep_0U_{[0\,5]}p_{43}$	*20*	*3756*	*formule vérifiée*	*5.80*	*1*
$Ep_0U_{[0\,5]}p_{43}$	*20*	*147*	*formule vérifiée*	*0.09*	*3*
$Ep_0U_{[0\,5]}p_{49}$	*20*	*7201*	*formule vérifiée*	*12.89*	*1*
$Ep_0U_{[0\,5]}p_{49}$	*20*	*427*	*formule vérifiée*	*0.17*	*3*
$Ep_0U_{[0\,5]}p_{13}$	*25*	*11527*	*formule vérifiée*	*11.08*	*3*
$Ep_0U_{[0\,5]}p_{58}$	*25*	*206*	*formule vérifiée*	*0.19*	*3*
$Ep_0U_{[0\,5]}p_{58}$	*25*	*8711*	*formule vérifiée*	*14.57*	*1*
$Ep_0U_{[0\,5]}p_{10}$	*25*	*14431*	*formule vérifiée*	*26.32*	*2*

Conclusion et perspectives

Les systèmes temps réel se distinguent des autres systèmes informatiques par le fait qu'ils soient soumis à des contraintes temporelles. Il est donc impératif d'avoir des outils et techniques de modélisation et de vérification qui puissent garder un haut degré de performance et de correction pour ce type de système. Par conséquent le temps dans un tel système doit être pris en considération d'une manière explicite durant la phase de modélisation et de vérification. Il existe plusieurs outils et techniques de modélisation des systèmes informatiques (files d'attente, chaines de Markov,...), dont plusieurs ont pu prouver leurs puissance et performances dans la modélisation et la vérification des systèmes. Parmi les outils courament utilisés pour modéliser les systèmes, les réseaux de Petri (RdP) occupent une place priviligiée. Les réseaux de Petri sont un outil trés utilisé car ils ont montré une grande puissance d'expression et de calcul pour la modélisation et la vérification de nombreux et différents types de systèmes, et ils fournissent des méthodes d'analyse avec lesquelles on peut vérifier certaines propriétés de systèmes (bornitude, vivacité,...).

Cependant, quand il s'agit d'un système temps-réel les RdPs atteignent trés vite leurs limites. Ceci est dû au fait que les RdPs ne prennent pas en considération le caractère temporel des systèmes. Ils se contentent d'effectuer une modélisation et une vérification du comportement logique d'un système hors son contexte temporel. Afin de palier à ce problème, et pour pouvoir profiter de la puissance des RdPs, des extensions temporelles des RdPs ont été proposées. Le but principal de telles extensions est de pouvoir éffectuer une modélisation et une vérification du système sur le plan logique du séquencement des évènement, et surtout sur le plan temporel primordial dans le cadre de la représentation des systèmes temps-réel.

Les propriétés générales telles que la bornitude, la vivacité, le non blocage ... reseignent le modélisateur sur le comportement général du système, celles-ci doivent être complétées par l'analyse des propriétés spécifiques du système modélisé.

Plusieurs axes de recherche ont été développés visant à mettre en place des outils permettant une description formelle des propriétés spécifiques des systèmes. La logique temporelle est un outil formel qui procure une syntaxe sûre, précise et sans ambiguïté des propriétés qualitatives et quantitatives. Cependant pour pouvoir exprimer des propriétés spécifiques des systèmes temps-réel c-à-d des propriétés quantitatives, des extensions temporelles des différentes logiques temporelles ont été proposées, telle que la logique TCTL (Timed CTL).

Aprés la description des propriétés spécifiques, viendra l'étape vérification de ces propriétés sur le système.Parmi les techniques de vérification: les approches

par model-checking, "vérification automatique" se sont largement développées ces dernières années. Cela suppose de modéliser le comportement du système à vérifier et d'énoncer la propriété de correction attendue sous la forme d'une formule de logique temporelle. En suite on peut utiliser un model-checker afin de vérifier si le modèle satisfait ou non la propriété.

Néanmoins, la vérification et la validation des systèmes temps réel souffrent de l'éxplosion combinatoire non seulement à la taille du système, mais aussi au nombre d'horloges.

Pour pallier ce problème, une nouvelle approche a été introduite qui consiste à éffectuer une vérification à la volée des proprietés TCTL (vérifiable à la volée) sur un ensemble de procéssus interconnectés et communiquant en échangeant des messages. Ce qui permet d'éffectuer une vérification à la volée sur une platforme distribuée.

Dans ce cadre, nous avons proposé un algorithme de génération distribuée de l'espace d'états d'un système temps réel modelisé par un RdPT en utilisant la méthode des zones pour la génération de l'ésoace d'états, nous avons proposé également des algorithmes de vérification à la volée et distribuée d'une sous classe de propriétés TCTL vérifiables à la volée.

Afin de valoriser notre travail, nous avons implémenté ces algorithmes pour prouver par les tests les avantages d'une vérification à la volée sur une platforme distribuée. Les tests montrent les avantages que peut apporter une vérification à la volée et distribuée par rapport aux tailles des systèmes qu'on peut vérifier ou par rapport au temps de calcul.

Comme perspectives il serait interessant de:

— Compléter notre vérificateur par l'implémentation d'un outil distribué de vérification à la volée en utilisant la méthode des zones sur les RdPT.
— Integrer notre vérificateur à l'outil Roméo.
— Enrichir notre vérificateur pour les RdPT non sauf en considérant la multisensibilisation.
— Enrichir notre travail pour le cas des RdPT contenant l'infini comme date de tir au plus tard des transitions en proposant une approximation adéquate.
— Proposer une vérification distribuée non à la volée pour permettre la vérification de toutes les propriétés TCTL.
— La définition des algorithmes de répartition à partir de l'analyse temporelle des différents états.

Références

[Alu94a] :ALUR R., DILL D., « A Theory of Timed Automata », Theoretical Computer Science, vol. 126, n"°2, p. 183-235, 1994.

[Bbd03] :B.Berthomieu, M.Boyer, M.Diaz «les réseaux de Petri :chapitre5 'les réseaux de Petri temporels'» Hermes, 2003

[Ber00] :BÉRARD B., DUFOURD C., « Timed Automata and Additive Clock Constraints »,Information Processing Letters, vol. 75, n"°1-2, p. 1-7, 2000

[Ber01] :B. Berthomieu «La méthode des classes d'états pour l'analyse des réseaux temporels», In modélisation des systèmes réactifs (MSR'01), pages 275-290, Toulouse, France, octobre 2001. Hermes.

[Bhv01] :M. Boyer « Contribution à la modélisation des systèmes à temps contraint et application au multimédia», Thèse de doctorat, Université Toulouse 3 Paul abatier, 2001.

[Bme83] :B.Berthomieu, M.Menasche «an enumerative approch for analysing time Petri nets" IFIP congress series, vol 9,p 41-46,1983

[Bou02] :P. Bouyer «Modèles et Algorithmes pour la Vérification des Systèmes Temporisés» phd thesis,Laboratoire Specification et Verification. ENS Cachan,France, apr 2002.

[Bou04a] :P. Bouyer« Forward Analysis of Updatable Timed Automata », Formal Methods in System Design, vol. 24, n"°3, p. 281-320, 2004.

[Bou05] :Patricia Bouyer, «foundation for timed system 2005».

[Bou06] :Patricia Bouyer, et Francois Larossinie, «chapitre1 : vérification par automate temporisé 2006».

[Bou07] :M.C Boukala, L. Petrucci «Toward distributed verification of Petri nets properties» VECo'S07, 1st international workshop on verification and evaluation of computer and communication systems May 2007

[Boy01] :M. Boyer « Contribution à la modélisation des systèmes à temps contraint et application au multimédia», Thèse de doctorat, Université Toulouse 3 Paul abatier, 2001.

[BRG05] :H. Boucheneb, H. Roux & G. Gardey « TCTL model checking of Time Petri Nets » 2005.

[Eme96] :E.A.Emerson,A.P.Sistla «Symetrie and Model-Checking »,In formal Methods and System Design 9,pp105-031, 1996

[Fle02] :Emmanuel Fleury, «automates temporisés avec mises à jour» ,2002.

[Gar05] :G. Gardey, «contribution à la vérification et au contrôle des systemes temps réel. Application aux réseaux de Petri temporels et aux automates temporisés», thèse de doctorat École centrale de Nantes et l'université de Nantes, Décembre 2005.

[GRR03] :G. Gardey, O.H. Roux and O.F. Roux ,«using zone graph method for computing the state space of a time Petri net». In In formal modeling and analysis of timed systems,(FORMATS'03), volume LNCS 2791. Springer-Verlag, September 2003.

[Gsh02] :A.Gu, K.G.Shin «Analysis of event-driven real time systems with time Petri nets, A translation approch»,IFIP conference proceedings, vol,219.Proceedings of the IFIP 17th world computer congress-Tc10 stream on distributed and parallel Embedded systems :design and analysis of distributed embedded systems,pages 31-40, 2002

[Had03] :S. Haddad, F.Vernadat «vérification et mise en oeuvre des réseaux de Petri». Vérification des propriétés spécifiques Paris,Hermes science publication, 2003

[Hen98] :HENZINGER T. A., KOPKE P. W., PURI A., VARAIYA P., « What's Decidable about Hybrid Automata ? », Journal of Computer and System Sciences, vol. 57, n"°1, p. 94-124, 1998.

[Jou00] :jean Pierre.Jouannaud : «cours vérification des systèmes réactifs temps réel» 2000.

[Kha97] :W. Khansa : « Réseaux de Petri p-temporels : contribution à l'étude des systèmes à évènements discrets » Thèse de doctorat, Université de Savoie, Annecy, France 1997.

[Kho06] :Ahmed Khoumsi, Mustapha Nour elfath «Méthode de transformation d'Automates temporisés avec invariants de localités».6e conférence de Modélisation simulation. Maroc. avril 2006.

[Lar98] :F. Laroussine, K.G.Larsen «CMC :A tool for compositional model-checking of real-time systems». Proc IFIP Joint Int. conf Formal description techniques & protocol specification, Testing and verification (Forte-PSTV'98), kluwer Academic publishers, 1998, P.439-456

[Lar05] :F. Laroussine, «Model-checking temporisé : Algorithmes efficaces et complexité». Theoretical computer science 2005

[Lar95] :K.G. Larsen.,P. Y1 W. Pettersson «Model-checking for real-time systems», Proc 10th International Conference on Fundamentals of Computation Theory (FCT'05),vol.965 de lecture Notes in computer science, springer,p. 62-88,1995.

[Lar97] :K.G. Larsen.,P. Y1 W. Pettersson «Uppal in a Nutshell», journal of software tools for technology transfer, vol 1, n°1-2, p. 134-152, 1997.

[Loh02] :C. LOHR, «contribution à la conception de systèmes temps réel en s'appuyant sur la technique de description formelle RT-LOTOS». Thèse de doctorat, institut national polytechnique de Toulouse. France 2002.

[Mer74] :P. Merlin «A study of the recoverability of computer systems» PHD. Thesis, univ of California Irvine, 1974

[Pet00] :P.K.Pettersson, G.Larsen «UPPAL 2k» Bulletin of the european association for theoretical computer science, vol 70, 2000 P 40-44

[Pet62] :C.A.Petri «kommunikation mit automaten» phd thesis, university of bonn 1962

[Ram74] :C. Ramchandani.«Analysis of asynchronous concurrent systems by timed Petri nets». PhD thesis. Massachusets Institute of technology. Cambridge, MA, 1974. Project MAC report MAC-TR-120.

[Ras05] :RASKIN J.-F., « An Introduction to Hybrid Automata », Chapitre Handbook of Net-worked and Embedded Control Systems, p. 491-518, Springer, 2005.

[Rob04] :ROBIN A., «Aux frontières de la décidabilité...», Master's thesis, DEA Algorithmique, Paris, 2004.

[Sch01] :P. Schnoebelen, B. Bérard, M.Bidoit, F.Laroussinie,A. Petit «Systems and software verification-model-checking techniques and tools», Springer, 2001.

[Yov93] :S. Yovine «Méthodes et outils pour la vérification symbolique des systèmes temporisés». thèse de doctorat, INP de Grenoble, France, 1993

[Yov97] :S. Yovine,«Kronos : A verification tool for real-time systems», journal of software tools for technology transfer, vol 1, n°1-2, p, 123-133, 1997

[Zou98] :M. Zouaoui «définition d'un langage de type logique temporelle pour la spécification et l'évaluation des performances» thèse de doctorat d'état, univ Pierre et Marrie Curie, Paris, 1998