REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

N° d'ordre : ……

UNIVERSITE DE M'SILA

FACULTE DES MATHEMATIQUES ET DE L'INFORMATIQUE
Département d'Informatique

MEMOIRE de fin d'étude

Présenté pour l'obtention du diplôme de master

Domaine : Mathématiques et Informatique

Filière : Informatique

Spécialité : Technologie information et communication

SUJET

Spécification et vérification des systèmes temps réel

Réalisé par :

MILI Soufyane

Dirigé par : BOURAHLA .M

Promotion : 2016 /2017

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

N° d'ordre : ……

UNIVERSITE DE M'SILA

FACULTE DES MATHEMATIQUES ET DE L'INFORMATIQUE
Département d'Informatique

MEMOIRE de fin d'étude

Présenté pour l'obtention du diplôme de master

Domaine : Mathématiques et Informatique

Filière : Informatique

Spécialité : Technologie information et communication

SUJET

Spécification et vérification des systèmes temps réel

Réalisé par :

MILI Soufyane

Dirigé par : BOURAHLA .M

Promotion : 2016 /2017


بسم الله الرحمن الرحيم
الحمد لله

Dédicace

Je dédie ce modeste travail :

A mon père et ma belle maman.

Et mon seul frère et sœurs.

A mes grands-parents.

A tous mes amis et mes collègues (TIC).

A tous ceux qui m'aiment et que j'aime.

Remerciements

En préambule à ce mémoire je remercie ALLAH qui m'aide et me donne la patience et le courage durant ces langues années d'étude.

Quelle meilleure opportunité que cette thèse qui marque achèvement des études universitaires, pour exprimer les plus vifs remerciements à nos promoteurs Monsieur BOURAHLA Mustapha pour nous avoir encadrées et guidées tout au long de notre projet.

Nos plus vifs remerciements vont aussi à tous les membres de jury pour avoir accepté d'honorer par leur jugement notre travail.

Sans oublier nos parents qui ont sacrifié jusqu'aujourd'hui, aussi leurs encouragements incessants tout le long de notre parcours. Ainsi que toutes personnes qui, de près ou de loin, a contribué à la réalisation de ce projet.

TABLE DE MATIERES

Table de figure :

Liste des tableaux :

INTRODUCTION GENIRALE

Introduction générale

En informatique, on parle d'un système temps réel lorsque ce système est capable de contrôler (ou piloter) un procédé physique à une vitesse adaptée à l'évolution du procédé contrôlé.

Les systèmes informatiques temps réel se différencient des autres systèmes informatiques par la prise en compte de contraintes temporelles dont le respect est aussi important que l'exactitude du résultat, autrement dit le système ne doit pas simplement délivrer des résultats exacts, il doit les délivrer dans des délais imposés.

Les systèmes informatiques temps réel sont aujourd'hui présents dans de nombreux secteurs d'activités :l'industrie de production par exemple, au travers des systèmes de contrôle de procédé (usines, centrales nucléaires) et les salles de marché au travers du traitement des données boursières en « temps réel » ,l'aéronautique au travers des systèmes de pilotage embarqués (avions, satellites) et l'automobile avec le contrôle de plus en plus complet des paramètres moteur, de la trajectoire, du freinage, etc.

Et dans le secteur de la nouvelle économie au travers du besoin, toujours croissant, du traitement et de l'acheminement de l'information (vidéo, données, pilotage à distance, réalité virtuelle, etc.).

Le développement de systèmes temps réel nécessite donc que chacun des éléments du système soit lui-même temps réel, c'est-à-dire permettre de prendre en compte des contraintes temporelles et la priorité de chacune des taches. Un système d'exploitation conçu pour prendre en compte ces contraintes est appelé système d'exploitation temps réel. Ce son des systèmes critique à cause de ça on doit vérifier leur conception.

Dans notre travail on a spécifié le protocole FDDI (Fiber Distributed Data Interface) a fin d'avoir un modèle pour le simuler et le vérifier formellement avec l'outil UPPAAL, Ainsi avec ce travail nous pouvons montrer commet modéliser, simuler et vérifier formellement des systèmes temps réel.

Ce mémoire est organisé comme suit , Après une introduction géniale , On a présenté dans le première chapitre ,les systèmes temps réel leurs caractéristique ,les taches, l'ordonnancement et leurs algorithmes. Ainsi , l'ordonnancement dans le cas de partage des ressources.

Dans le deuxième chapitre on trouve deux parties, la première partie est consacrée aux formalismes de spécification des systèmes temps réel en particulier les automates temporisée et les logiques temporelles temporisés pour spécifier les propriétés des systèmes temps réel.

La vérification formelle des systèmes temps réel est basée sur la technique du modèle-checking (vérification par modèle).

Dans la deuxième partie on a présenté l'outil UPPAAL.

Notre contribution est présentée dans le troisième chapitre qui est basée sur la modélisation du protocole FDDI. Ce modèle est simulé par vérifier son comportement et a la fin vérifier formellement certaine propriétés.

Ce mémoire est conclu par une conclusion générale et des perspectives.

Chapitre 1

LES SYSTEMES TEMPS REEL

1 Chapitre 1 : LES SYSTEMES TEMPS REEL

1.1. Définition

Un système informatique valide est un système qui remplit son contrat et donc n'a pas un comportement autre que celui attendu. Dans le cas d'un système effectuant des calculs numériques, le contrat stipule notamment que les résultats des calculs doivent être justes. Dans [Sta88], une définition des systèmes temps réel est donné en référence à la validité d'un système informatique :

« *In real-time computing the correctness of the system depends not only on* »

« *the logical results but also on the time at which the results are produced.* »

En plus d'une validité du système classique liée à la justesse des calculs effectués, des propriétés de validité liées au temps sont introduites.

Ces systèmes sont généralement des systèmes réactifs dont le comportement dépend de l'environnement. On a alors un ou plusieurs capteurs permettant d'obtenir une approximation de l'environnement et de son évolution. Le système, selon cette approximation, décide de l'action ou plutôt de la réaction qu'il doit entreprendre. Une réaction peut ne plus être adaptée après une évolution de l'état de l'environnement et donc être valide pendant un temps limité.

Il faut alors suivre les évolutions de l'environnement pour réadapter la réaction. Généralement, l'environnement est en évolution continue et ne pouvant avoir une mesure continue de l'état de l'environnement, on définit alors un rythme de fonctionnement des capteurs donnant un échantillonnage raisonnable de cette évolution. De même, le calcul de la réaction ne pouvant se faire en temps nul, on donne alors une échéance à ce temps de calcul. Le but est que malgré le temps pris par le calcul de la réaction, cette réaction reste cohérente par rapport à l'état de l'environnement.

Les définitions des propriétés temps réel sont alors généralement données à travers un temps de réponse du système, le temps maximum pour produire la réaction du système. De même que les calculs effectués par un système doivent être justes pour produire des résultats justes, on préfère formuler les propriétés temps réel sous une autre forme. S'il existe une échéance pour ce temps de calcul, c'est que le temps de réaction du système n'est pas nul et que la mesure de l'environnement en entrée n'est valide que pendant un certain intervalle de temps. Tout résultat d'un calcul effectué à partir d'une mesure de l'environnement est donc valide pendant un intervalle de temps d'autant plus limité que le temps de calcul est long. Les limites sur le temps de réponse du système dépendent en fait de la validité du résultat des

calculs dans le temps. La définition d'un système temps réel sur laquelle repose les travaux de cette thèse est la suivante :

La validité d'un système temps réel dépend de la validité temporelle de ses données.

Par validité temporelle des données, on entend que les différentes valeurs prises par les variables du système sont temporellement valides. Un exemple de validité temporelle est d'exiger qu'une donnée soit fraîche et donc qu'elle soit mise à jour fréquemment. Une valeur peut aussi être définie comme temporellement valide uniquement si elle est basée sur des valeurs temporellement valides d'autres variables. Pour la valeur d'une variable, cette validité temporelle dépend alors de la sémantique de la variable, de la signification des différentes valeurs qu'elle prend (par exemple une température, une vitesse, une position, etc), et des liens avec les autres variables du système.

La validité d'un système ayant de telles propriétés dépend bien sûr des mécanismes qui réalisent ce système. Il s'agit cependant d'exprimer ces propriétés indépendamment des mécanismes. Une définition formelle de la validité temporelle de la valeur d'une variable est donnée par la suite.

1.2. Base de données temps réel

Dans le domaine des bases de données, il est possible de donner des propriétés temps réel sur les transactions mais on s'intéresse aussi à la validité temporelle des valeurs stockées dans la base de données. Dans [XSS+ 96], des intervalles de validités sont associées à chaque donnée de la base. Les échéances des transactions ne sont pas fixées à l'avance. Elles dépendent des données utilisées et de l'instant où ces données sont lues. En effet, une donnée lue par une transaction en fin de son intervalle de validité implique une échéance courte pour cette transaction. Il s'agit alors d'effectuer une analyse d'ordonnancement permettant de respecter ces échéances. Dans [ABRW93], l'intervalle de validité d'une valeur est définie par rapport au décalage avec l'environnement. La valeur doit être une mesure de l'environnement dans un état récent. De même, pour le résultat d'un calcul, le décalage avec l'environnement autorisé pour ce résultat est le minimum des décalages autorisés pour les entrées du calcul. De ces intervalles de validité, un ordonnancement du système est déduit sous la forme d'un ensemble de transactions périodiques synchronisées. Nous cherchons à exprimer des propriétés sur les valeurs semblables à celles qui viennent d'être présentées. On s'intéresse notamment, comme dans [ABRW93], aux liens entre les validités temporelles de chaque valeur des variables selon les liens entre les valeurs de ces variables. On souhaite cependant donner une sémantique plus riche à ces propriétés. La validité temporelle du résultat d'un

calcul dépend des entrées mais dépend aussi de la signification de ce calcul. On peut ainsi avoir un décalage avec l'environnement plus important que celui autorisé pour les entrées d'un calcul selon les cas.

1.3. Conception d'un système temps réel

- Description et expression des contraintes temps réel
- Représentation simultanée de l'évolution logique et temporelle du système
- Prédiction et estimation des temps de réponse
- Sélection de l'architecture, du matériel et du logiciel
- Le matériel conditionne les coûts de production
- Le logiciel conditionne les coûts de développement
- Compromis et équilibre matériel-logiciel [MB 17**]**.

1.4. Classes des systèmes temps-réel

1.4.1. Temps-réel strict/dur (hard real-time)

Le non respect d'une contrainte de temps a des conséquences graves (humaines, économiques, écologiques) : besoin de garanties.

Ex : applications de contrôle dans l'avionique, le spatial, le nucléaire, contrôle de production de matériaux toxiques [PI 09].

1.4.2. Temps-réel souple/mou (soft real-time)

On peut tolérer le non respect occasionnel d'une contrainte de temps (garanties probabilistes)

Ex : applications multimédia grand public (vidéo à la demande, TV) [PI 09].

1.5. Exemples de systèmes temps réel

Préface d'un livre sur le temps réel souple :

"*This book is about real-world programming ... So real-world programs (and real-world programmers) are all around us. What characterizes all of these real-world applications is a critical dependence on time.*" [GAL 95]

- ❖ Transports : métro, aéronautique (avions, satellites, spacial), trains, automobile, ...
- ❖ Multimédias : décodeurs numériques openTV, décodeurs TNT, MPEG, jeux vidéo. films d'animation.

- ❖ Services téléphoniques : téléphone mobile, auto-commutateur. Supervision médicale, écologique.
- ❖ Système de production industriel : centrale nucléaire, chaîne de montage, usine chimique.
- ❖ Robotique (ex : PathFinder) [2]**.**

1.6. Caractéristiques d'un système temps réel

- Un système d'exploitation temps réel doit s'affranchir des incertitudes sur le temps
- Un tel système possède la caractéristique d'être déterministe
- Il apporte les services suivants :
  - Communication,
  - Synchronisation,
  - Gestion et ordonnancement des tâches,
  - Gestion de la mémoire,
  - Gestion des interruptions et des entrées/sorties physiques,
  - Entrées/sorties logiques et gestion des périphériques,
  - Gestion du temps.

1.7. Les tâches des systèmes temps réel

- Aujourd'hui, la plupart des systèmes sont du type multi-tâches.
- Une tâche (processus) est un processus élémentaire exécuté par le processeur
- Elle est constituée de trois zones :
  - code : contient les instructions du programme,
  - données : contient les données globales du programme,
  - pile : contient la pile du programme, pour les données temporaires.
- Chaque tâche est caractérisée par un contexte. Il contient les différents registres, le nom du processus, et son état.

1.8. Ordonnancement d'un système temps réel

Ensemble des règles définissant l'ordre d'exécution des calculs sur le processeur [PI 09].

Pourquoi ordonnancer ?

- **Parce que ça a un impact sur le respect des contraintes de temps**

Exemple :

Tâche T1 : arrivée en 0, temps d'exécution 4, échéance 7

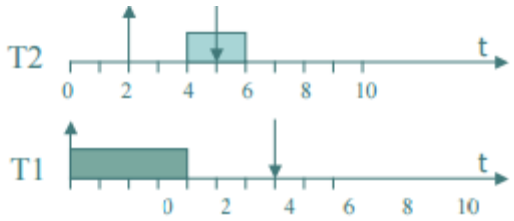
Tâche T2 : arrivée en 2, temps d'exécution 2, échéance 5

Ordonnancement O1: premier arrivé, premier servi, non préemptif

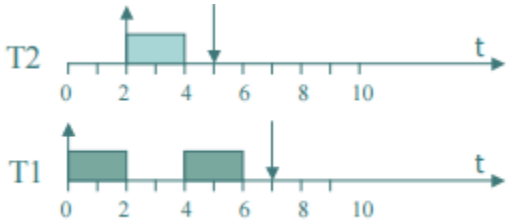
Ordonnancement O2: préemptif à priorité, T2 plus prioritaire que T1

Échéance des tâches respectées avec O2, pas avec O1

Ordonnancement O1 : T2 rate son échéance



Ordonnancement O2 : toutes les échéances sont respectées



T=2 : préemption de T1 au profit de T2 [PI 09].

Différents types d'ordonnanceurs :

- préemptif (avec réquisition) : L'ensemble des Unix et Windows NT
- non-préemptif : Windows 3.11

1.9. Classification Ordonnancement

1.9.1. Ordonnancement Hors-ligne / En-ligne

Ordonnancements Hors-ligne

Un ordonnancement hors-ligne (*off-line* en anglais) signifie que la séquence d'ordonnancement est prédéterminée à l'avance : dates de début d'exécution des tâches, de préemption/reprise éventuelles. En pratique, l'ordonnancement prend la forme d'un *plan hors-ligne* (ou *statique*), exécuté de façon répétitive (on parle aussi d'ordonnancement *cyclique*, qui

définit le *cycle majeur*) : à chaque top d'horloge (on parle de cycle *mineur*) une tâche particulière du cycle majeur courant est exécutée jusqu'à terminaison ou jusqu'au top d'horloge suivant[4].

Figure 1.1 Ordonnancements Hors-ligne

* **Evaluation**

Intérêts :

- Mise en œuvre simple.
- Facile de détecter les dépassements d'échéances (surcharge).

Limitations :

* Rigidité (pas toujours applicable) [PI 09].

Ordonnancements En-ligne

Un ordonnancement en-ligne correspond au déroulement d'un algorithme qui tient compte des tâches effectivement présentes dans la *file d'ordonnancement* (*run-queue* en anglais) lors de chaque décision d'ordonnancement. Les ordonnanceurs en-ligne peuvent reposer sur la construction de plans d'ordonnancements en cours de fonctionnement, auquel cas ils sont dits à plan *dynamique* [*KSSR96*]. Mais plus couramment, ces ordonnanceurs sont fondés sur la notion de priorité.

* **Evaluation**

* Intérêts : flexibilité
* Inconvénients :
  - surcoûts de mise en œuvre
  - plus difficile de détecter les surcharges [PI 09].

1.9.2. Ordonnancement non préemptif / préemptif

Non préemptif

On n'interrompt jamais l'exécution d'une tâche en cours au profit d'une autre tâche.

Préemptif

La tâche en cours peut perdre involontairement le processeur au profit d'une autre tâche (jugée plus urgente)

Préemptif => besoin d'un exécutif multitâche[PI 09].

Remarques : Orthogonal par rapport à la classification enligne / hors-ligne

- Un ordonnancement hors-ligne peut être préemptif
- Un ordonnancement en-ligne peut être non préemptif

1.9.3. Ordonnancement conduits par la priorité

- Ordonnancements préemptifs
- C'est la tâche de plus haute priorité qui s'exécute [PI 09].

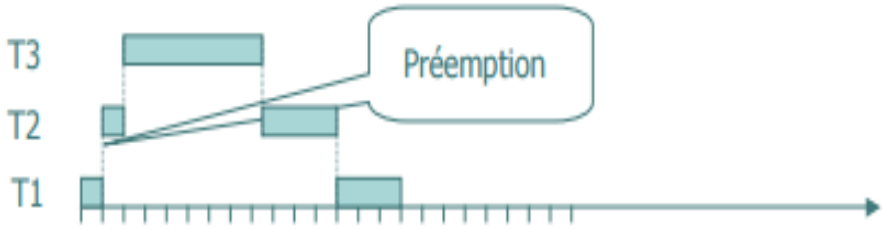
1.9.4. Priorités fixes (statiques) / priorités dynamiques

- ✓ Fixes **:** Indépendants du temps, fixés a priori
- ✓ Dynamiques **:** Evoluent avec le temps [PI 09].

Exemple

prio(T3) > prio(T2) > prio(T1) (ici, priorités fixes)

T1, T2 et T3 arrivent respectivement aux dates 1, 2, et 3



1.10. Différentes lois d'ordonnancement

- Premier Arrivé, Premier Servi (PAPS ou FIFO)

Supposons trois tâches, T1, T2 et T3 à exécuter. Leurs durées de traitement valent respectivement 6, 3 et 4 UT (Unité de Temps) [MB 17].

T_1	T_1	T_1	T_1	T_1	T_1	T_2	T_2	T_2	T_3	T_3	T_3	T_3

=> Cet algorithme n'est absolument pas efficace en temps réel. Toute tâche créée doit attendre que les précédentes soient terminées.

- Tour de rôle ou tourniquet (round robin) :

Les tâches sont mises dans une file d'attente et sont activées périodiquement. Cette période se nomme un quantum de temps [MB 17].

T_1	T_2	T_3	T_1	T_2	T_3	T_1	T_2	T_3	T_1	T_3	T_1	T_1

=> Avec cette méthode, les processus sont tous exécutés. Cependant, aucune notion d'importance n'est prise en compte.

- Priorité :

Cette méthode d'ordonnancement permet de résoudre le problème évoqué précédemment.

Lorsqu'une tâche est prête, elle est placée dans une file d'attente en fonction de sa priorité.

T_2	T_2	T_2	T_1	T_1	T_1	T_1	T_1	T1	T_1	T_3	T_3	T_3

=> L'inconvénient de cette méthode est qu'une tâche peut s'exécuter au détriment d'une autre d'égale priorité.

- Mixte :

Pour assurer une équité entre des tâches de même priorité on associe souvent la méthode de priorité avec celle du tourniquet [MB 17].

T_2	T_2	T_2	T_1	T_3	T_1	T_3	T_1	T_3	T_1	T_3	T_1	T_1

- L'algorithme HPF

Pour utiliser cet algorithme, il faut commencer par déterminer la priorité de chaque tâche.

Ceci se fait en utilisant la méthode d'analyse monotone par taux (Rate Monotonic). Cette méthode permet d'affecter, hors ligne, une priorité statique aux tâches.

Critères retenus pour les tâches [MB 17]:

Les tâches doivent être périodiques : la priorité de chaque tâche est inversement proportionnelle à la fréquence d'apparition de la tâche.

Les tâches peuvent être préemptées mais on néglige le temps de commutation et d'ordonnancement.

Les tâches sont indépendantes les unes des autres (pas de synchronisation entre tâches), la capacité des tâches est connue.

Si on prend une priorité statique basée sur la période prio(Ti) = 1/Pi

C'est à dire la tâche la plus prioritaire est celle de plus petite période.

C'est optimal pour la classe des algorithmes à priorité statique pour des configurations Cfp de n tâches à échéances sur requête.

Le test d'acceptabilité (Condition suffisante) est le suivant :

Avec les tâches définies plus haut, le test d'acceptabilité n'est pas vérifié

$2/6 + 2/8 + 3/12 \leq 3*(21/3 - 1)$

$0{,}84 \leq 0.78$ n'est pas vérifié

On obtient l'ordre de priorité suivant : A, B et C (A est la plus prioritaire).

L'ordonnancement des tâches est donc le suivant [MB 17] :

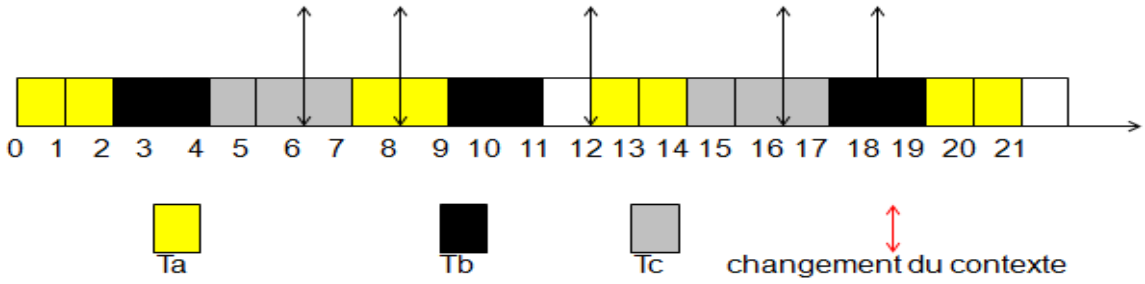


Figure 1.2 L'ordonnancement des tâches de l'algorithme HPF

Il existe une variante à cette analyse : l'analyse Deadline Monotonic. Avec cette méthode, la priorité d'une tâche est inversement proportionnelle à son échéance.

- Rate Monotonic

C'est un algorithme qui est basé sur des priorités statiques et qui est optimal. Une tâche dispose d'une priorité fixe qui est inversement proportionnelle à la période. Nous présentons les tests de faisabilité dans le cas préemptif et non préemptif[1].

- Priorité de la tâche fonction de sa période. Priorité constante
- La tâche de plus petite période est la tâche la plus prioritaire

• Pour un ensemble de n tâches périodiques à échéance sur requête Tpi (r0, Ci, Pi), un test d'acceptabilité est (condition suffisante) :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{1/n} - 1)$$

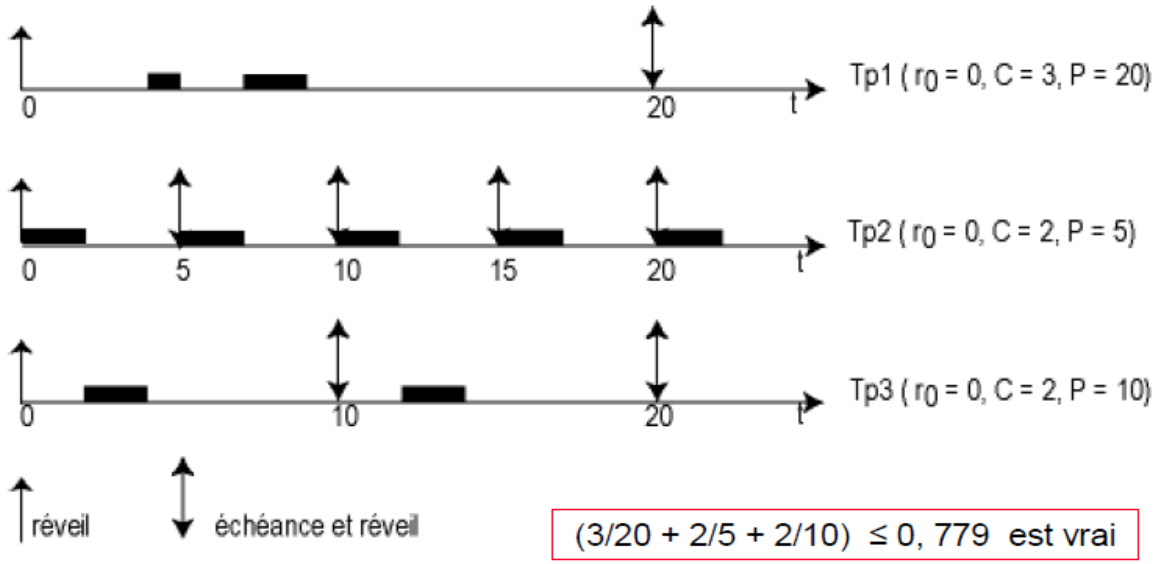


Figure 1.3 : Un test d'acceptabilité de l'algorithme Rate Monotonic

✓ Rate Monotonic préemptif :

Une condition suffisante a été définie par [LL 73]. L'ordonnancement Rate Monotonic d'un ensemble de n tâches périodiques est faisable si le facteur d'utilisation U vérifie la relation suivante :

$$U = \sum_{i=1}^{n} Ci / Pi$$

$$U \leq n * (2^{1/n} - 1)$$

✓ Rate Monotonic non préemptif

Pour Rate Monotonic non préemptif, un travail a été effectué sur la recherche d'une condition d'ordonnançabilité [CAR96]. Celle-ci est basée sur l'étude du partage de ressources des algorithmes préemptifs, et en particulier des techniques à priorité héritée [SRL 90]. Ainsi, les deux tests suivants présentent deux manières de vérifier l'ordonnancement :

1ère test :

$$\forall i \ 1 \le i \le n \quad \sum_{j=1}^{i} Cj/Pj + Bi/Pi \le i * (2^{1/i} - 1)$$

2ème test:

$$\sum_{i=1}^{n} Ci/Pi + \max_{1 < i \le n} \{ Bi/Pi \} \le n * (2^{1/n} - 1)$$

Avec

$$Bi = \max_{i+1 \le j \le n} \{ Cj \}$$

- Earliest Deadline First

L'algorithme Earliest Deadline First est basé sur des priorités dynamiques. La priorité maximale est accordée à la tâche dont l'échéance est la plus proche.

Nous avons trouvé des tests de faisabilité pour le cas préemptif et le cas non préemptif que nous présentons ci-après. Dans les deux cas, il faut distinguer les tâches à échéance sur requête et les tâches quelconques [LL73].

- EDF préemptif

Pour des tâches à échéance sur requête une condition nécessaire et suffisante a été donnée par [LL73]. Celle-ci donne une borne sur le taux d'utilisation :

$$\sum_{i=1}^{n} \frac{C_i}{R_i} \le 1$$

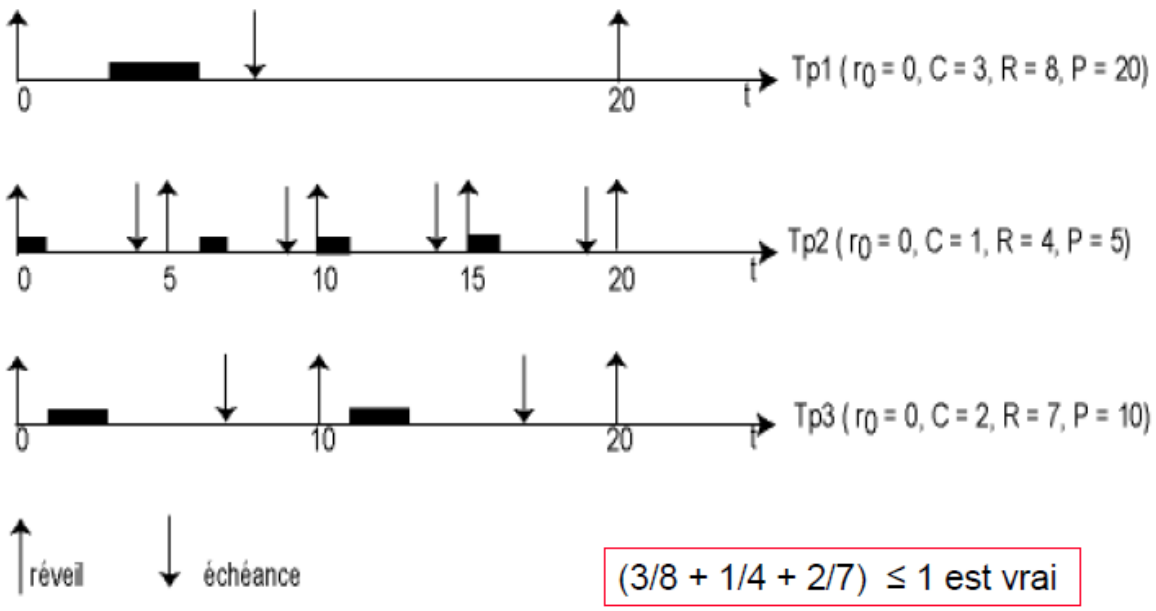


Figure 1.4 : Un test d'acceptabilité de l'algorithme EDF préemptif

- EDF non Préemptif

Nous présentons ici un test qui permet l'obtention des conditions nécessaires et suffisantes d'ordonnançabilité pour les tâches à échéance sur requête. Ce test est défini dans [JSM91]. Par contre, aucun test pour l'algorithme EDF non préemptif avec des tâches quelconques n'a pu être trouvé.

Pour bien comprendre le test de faisabilité pour des tâches à échéance sur requête, il est nécessaire de rappeler ce qu'est le pire temps de déphasage dont nous avons déjà parlé à l'occasion du test de Rate Monotonic non préemptif. Ce pire temps de déphasage se produit lorsque la requête d'une tâche arrive une unité de temps avant l'arrivée des requêtes des taches plus prioritaire. Le phénomène est appelé : « inversion de priorité ».

1.11. Classification des algorithmes d'ordonnancement

- A contraintes strictes ou souples
- Respect des contraintes garanties à 100% ou tolérance
- Statiques ou dynamiques
- Ordonnancement fixe ou réévaluation en ligne
- Préemptifs ou non préemptifs
- Tâches pouvant être interrompues avant terme ou pas
- Centralisés ou répartis

1.12. Partage de ressources

Accéder à une ressource c'est éventuellement devoir attendre qu'elle se libère = borne sur le temps de blocage (noté $\mathcal{B}i$) **[3]**.

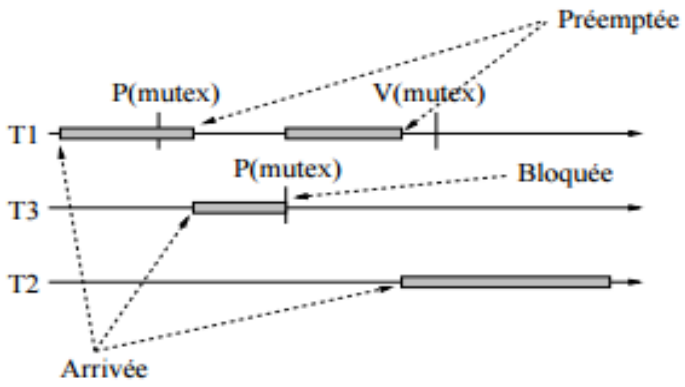


Figure 1.5 Partage de ressource

• Problèmes :

L'inversion de priorités : comment réduire la durée de l'inversion de priorité ?

⇒ protocoles d'héritage de priorité. Comment finement évaluer $\mathcal{B}i$?

• Caractéristiques des protocoles : calcul de $\mathcal{B}i$, nombres de ressources accessibles, complexité, interblocage possible ou non, etc[3].

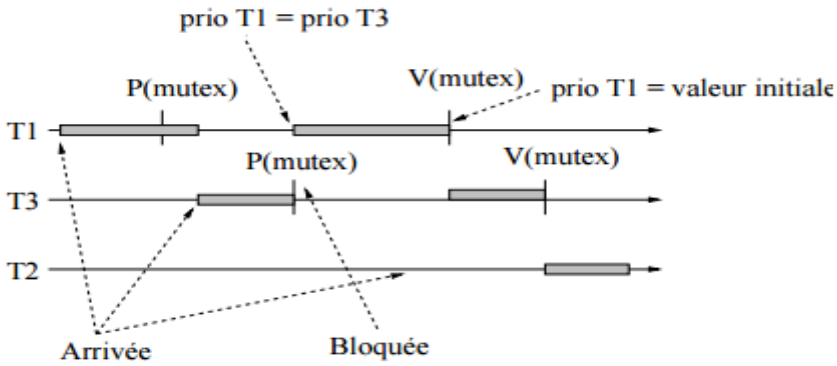


Figure 1.6 Partage de ressource avec priorités

• Héritage simple (ou Priority Inheritance Protocol ou PIP) :

Une tâche qui bloque une autre plus prioritaire qu'elle, exécute la section critique avec la priorité de la tâche bloquée.

Une seule ressource : sinon interblocage possible.

$\mathcal{B}i$ = somme des sections critiques des tâches moins prioritaires que i.

• PIP ne peut pas être utilisé avec plusieurs ressources ⇒ interblocage.

On implante généralement PCP (Priority Ceiling Protocol) [SHA 90] (ex. VxWorks).

• Deux variétés de PCP : OCPP et ICPP

Exemple d'Immediate Ceiling Priority Protocol ou ICPP :

Priorité plafond d'une ressource = priorité statique maximale de toutes les tâches qui utilisent la ressource.

Priorité dynamique d'une tâche = maximum (priorité statique de la tâche, priorité plafond de toutes les ressources allouées).

$\mathcal{B}_i$ = plus grande section critique.

• Soit n tâches périodiques synchrones à échéance sur requête, ordonnées de façon décroissantes selon leur priorité (avec $\mathcal{B}_n = 0$ donc).

• Prise en compte du temps de blocage dans :

** Critère d'ordonnançabilité RM :

$$\forall\, i, 1 \leq i \leq n : \sum_{k=1}^{i-1} \frac{C_k}{P_k} + \frac{C_i + B_i}{P_i} \leq i(2^{\frac{1}{i}} - 1)$$

Critère d'ordonnançabilité EDF/LLF :

$$\forall\, i, 1 \leq i \leq n : \sum_{k=1}^{i-1} \frac{C_k}{P_k} + \frac{C_i + B_i}{P_i} \leq 1$$

• Prise en compte du temps de blocage $\mathcal{B}_i$ dans le calcul du temps de réponse d'un ensemble de tâches périodiques synchrones à échéances sur requêtes, ordonnancées priorité fixe préemptif :

$$r_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{r_i}{P_j} \right\rceil C_j$$

Où $hp(i)$ est l'ensemble des tâches de plus forte priorité que i.

• Résolution par la méthode itérative :

$$w_i^{n+1} = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^n}{P_j} \right\rceil C_j$$

Attention :

Avec B_i, le calcul du temps de réponse devient une condition suffisante mais non nécessaire. C'est une solution pessimiste.

Chapitre 2

SPÉCIFICATION ET VÉRIFICATION FORMELLE DES SYSTÈMES TEMPS RÉEL

2 Chapitre II : SPECIFICATION ET VERIFICATION FORMELLE DES SYSTEMES TEMPS REEL

Introduction

Dans ce chapitre 2 qui intitulée spécification et vérification formelle des systèmes temps réel ,qui dévisie par deux partie , nous allons voir dans premiere parti les automates communicants temporisés et la logique temporelle temporisée (TCTL) et model checking temporisé , Dans le deuxieme parti la modélisation avec language UPPAAL.

2.1 Partie 1 : Spécification et Vérification:

2.1.1 Les automates communicants temporisés

- Automates Temporisés
  - Introduction

Les automates temporisés ont été introduits par R. Alur et D. Dill dans les années90 [AD90]. Il s'agit d'automates classiques munis d'horloges qui évoluent de manière continue avec le temps. Des conditions sur la valeur de ces horloges permettent d'intégrer des contraintes temps-réel dans la description du comportement de ces modèles. Depuis leur introduction, de nombreuses variantes d'automates temporisés ont été proposées, ici nous nous contentons d'une d´définition simple de ces modèles et nous mentionnons juste quelques variantes classiques.

A chaque transition d'un automate temporisé, est associe une *garde* (une contrainte sur la valeur des horloges) d´écrivant quand la transition peut être exécuté, et un ensemble d'horloges qui doivent être remises a zéro lors du franchissement de la transition. Chaque état de contrôle contient un invariant (une contrainte sur les horloges) qui peut restreindre le temps d'attente dans l'´etat et donc, d'une certaine manière, forcer l'exécution d'une transition d'action. Cette notion d'invariant a donc pour principale fonction d'introduire de la vivacité dans le modèle, comme peuvent le faire des conditions d'équité (du type Büchi ou Müller).

Le domaine de temps $\mathbb{T}$ que nous considérons est $\mathbb{R}+$ mais la plupart des résultats présent´es ici ne sont pas modifiés lorsque l'on considère $\mathbb{N}$.

Un automate est une machine à état finie. Il peut être représenté par un graphe. Une définition formelle d'un automate est la suivante :

Définition

Automate. Un automate est défini par un quintuplet A = (N,$\mathcal{L}$, →,$\mathcal{I}$,Q) *avec :*

- N *:* ensemble fini d'états ;

- $\mathcal{L}$ *: un* alphabet, un ensemble fini de symboles ;

- →: (N × $\mathcal{L}$) × N : la relation de transition ;

– I : l'ensemble des états initiaux ;

– Q : l'ensemble des états accepteurs.

Dans la représentation par un graphe, chaque nœud est un état et les arcs définissent la relation de transition du système. Un automate est donc un système de transitions étiqueté. A partir de l'état initial, on utilise un mot c'est-à-dire une séquence finie de symboles de l'alphabet. Ce mot détermine avec la relation de transition l'enchaînement des états pris par le système, les symboles du mot étant lus les uns après les autres. Le mot est reconnu si tout au long de la lecture du mot il existe une transition correspondant au symbole lu. De plus, l'état final du système doit faire partie des états accepteurs. L'automate donné figure 1 reconnaît les mots constitués de a et de b et finissant par un b.

Il existe plusieurs types d'automates. On cite les automates de Büchi [Tho90] qui sont une extension à des mots infinis. Pour cela, la sémantique des états accepteurs est modifiée, un mot est accepté si l'exécution passe infiniment souvent dans les états accepteurs. L'ensemble de ces états étant fini, c'est équivalent au passage infiniment souvent par l'un des états accepteurs.

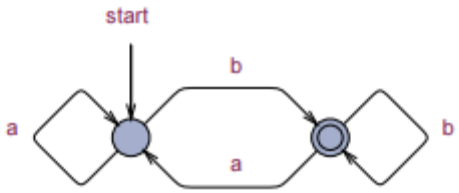


Figure 2.1 Un exemple d'automate.

Les automates temporisés sont le type d'automate qui nous intéresse le plus. On introduit la forme la plus courante définie par [AD94]. Le temps est introduit par l'utilisation d'un ensemble d'horloges qui sont des variables positives, prenant leurs valeurs dans $\mathbb{R}+$.

Définition 1

Automate temporisé. Un automate temporisé est défini par un sextuplet A = (N,$\mathcal{L}$,C,T,I,Q) où :

– N : un ensemble fini d'états ;

– $\mathcal{L}$: un alphabet, un ensemble fini de symboles ;

– C : un ensemble fini d'horloges ;

– I : l'ensemble des états initiaux ;

– $T \subset N \times N \times \mathcal{L} \times 2^{C} \times \Phi(C)$: l'ensemble des transitions ;

– Q : l'ensemble des états accepteurs.

L'ensemble 2^{C} définit les horloges remises à zéro lors de la prise de la transition. Pour un ensemble d'horloges C, Φ(C) définit l'ensemble des conditions δ :

$\delta := z \sim c \mid \delta_1 \wedge \delta_2$

Où $c \in \mathbb{Q}$ est un rationnel, $z \in \mathbb{C}$, $\sim \in \{<, \leq, =, , \neq, \geq, >\}$. Pour une condition δ et un vecteur de valeurs u prises par les horloges de C, on note δ(u) le prédicat indiquant que les valeurs de u vérifient δ.

Définition 2

Système de transitions défini par un automate temporisé. Un automate temporisé définit un système de transitions où un état du système est défini par un couple (n,u) donnant l'état courant n et les valeurs u des horloges. On note $u' = [r]u + c$ l'affectation d'horloge qui ajoute c aux valeurs u prises par les horloges de C, excepté les horloges de r qui sont remises à zéro. La relation de transition → d'un automate temporisé est définie par $\langle n,u\rangle \rightarrow \langle n', u'\rangle$ si :

$\exists r \subseteq \mathbb{C}, \exists a \in L, \exists \delta \in \Phi(\mathbb{C}) : \langle n,n',a,r,\delta\rangle \in T$ et $\exists d \in \mathbb{R}^{+} : \delta([\emptyset]u + d) \wedge u' = [r]u + d$

L'évolution des horloges se fait au même rythme que l'évolution du temps qui est implicite. On peut donc avoir une mesure du temps global avec une horloge jamais remise à 0. Une transition existe lorsque le passage du temps permet d'arriver dans un état où les valeurs des horloges vérifient la condition associée à une transition. L'étiquette associée à cette transition est alors lue et on change alors de nœud en remettant à jour l'ensemble des horloges spécifié par cette transition. Un automate temporisé est déterministe si les contraintes sur les horloges et les étiquettes des différentes transitions quittant un état sont mutuellement exclusives. Un exemple d'automate temporisé est donné figure 2.

D'autres modèles d'automates temporisés existent, notamment celui utilisé dans l'outil de vérification UPPAAL [LPY97]. Par rapport à la définition utilisée ici, les nœuds des automates d'UPPAAL peuvent avoir un invariant sur les horloges comme propriété.

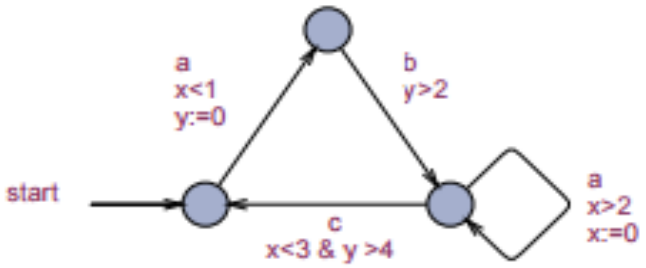


Figure 2.2 Un exemple d'automate temporisé

Types de mise à jour	Sans contraintes diagonales	Avec contraintes diagonales
x := c; x := y	PSpace complet	PSpace complet
x := x + 1		Indécidable
x := y + c		
x := x − 1	Indécidable	
x :< c	PSpace complet	PSpace complet
x :> c		Indécidable
x :~ y + c		
y + c <: x :< y + d		
y + c <: x :< z + d	Indécidable	

Table 2.1 Résultats de décidabilité.

Cet invariant est utilisé pour forcer un changement d'état avant que l'invariant ne soit plus respecté. De plus, il est possible de mettre plusieurs automates en concurrence ou de les synchroniser par rendez vous.

DÉCIDABILITÉ. Le problème de la satisfiabilité pour les automates dont la relation de transition est défini par la définition 2 est décidable et PSpace complet. Cette vérification s'appuie sur l'utilisation de zones ou régions définies par un ensemble de contraintes sur les horloges, la prise de transition définit un nouvel ensemble de contraintes et une nouvelle zone. Pour du model checking, on construit l'intégralité de ces régions qui est fini mais exponentiel par rapport au nombre d'horloges. On obtient cependant un graphe abstrait de l'ensemble des états atteignables dans un automate temporisé. Cette méthode est détaillée dans [BY04]. L'outil de model checking Uppaal utilise cette méthode pour vérifier des automates temporisés. Les propriétés qui sont vérifiées sont spécifiées dans une logique temporelle proche de CTL.

D'autres types d'automates temporisés existent, différenciés par la définition de la relation de transition et notamment par les mises à jour des horloges et le type de conditions sur ces horloges. On peut utiliser des contraintes diagonales, c'est-à- dire évaluant la différence

$x - y \sim c$ entre deux horloges x et y par rapport à c et où $\sim \in \{<, \leq, =, \neq, \geq, >\}$. Il est aussi possible de mettre à jour une horloge avec n'importe quelle valeur de manière déterministe, par exemple $z := x + c$ affecte la valeur de l'horloge x plus une constante à l'horloge z, ou indéterministe, par exemple $z :< c$ affecte une valeur entre 0 et c à l'horloge z. La figure 3 reprend le tableau donné dans [FPY02] et résume les résultats de décidabilité pour la satisfiabilité de l'automate. Dans chaque cas, il s'agit de construire une partie du graphe des régions abstrayant les états atteignables. Les différentes mises à jour données sont ajoutées à la remise à zéro.

EXPRESSIVITÉ. L'expressivité des différents modèles est aussi donnée par [FPY02]. Le modèle le moins expressif est celui d'un automate classique défini par la définition 1 et dont la relation de transition est déterministe. On regroupe ensuite les modèles décidables ayant des mises à jour des horloges déterministes. Ils sont équivalents aux automates temporisés classiques définis par [AD94]. En terme d'expressivité, on trouve ensuite les automates ayant des mises à jour indéterministes mais libres de contraintes diagonales puis ceux avec ce type de contraintes. Pour finir, on a les classes indécidables. Finalement, on utilise des automates temporisées pour réaliser des analyses d'ordonnancement [FPY02]. Cette technique est implantée dans l'outil Times. On peut notamment modéliser le déclenchement d'une tâche selon l'état d'un automate.

2.1.2 Le model-checking

- Historique

Avec l'arrivée des premiers ordinateurs et des premiers programmes séquentiels déterministes, des études sur la vérification ont débute avec des techniques de preuves manuelles de systèmes séquentiels. La logique de Hoare [Hoa69] pour la correction et la terminaison de programmes a vu le jour, sous la forme de règles d'inférences permettant de vérifier un programme séquentiel. La technique proposée était innovante, élégante et "compositionnelle" (la correction d'un programme est déterminée par la correction de ses composants).

Très vite la question s'est posée de la vérification de systèmes concurrents. Amir Pnuelli en 1977 [Pnu77] à le premier introduit la logique temporelle linéaire permettant de décrire le comportement d'un système concurrent dans le temps de manière "intemporelle"

(qualitativement) et surtout de manière formelle. Ses travaux permettent d'unifier la spécification formelle des systèmes séquentiels et concurrents dans un même formalisme.

La vérification de programmes concurrents était néanmoins toujours effectuée manuellement dans les travaux de Pnuelli. En 1978 le langage CSP [Hoa85], algèbre de processus permettant de décrire formellement les systèmes concurrents a été introduite par Hoare. En 1980 et 1981, Emerson et Clarke ont introduit la logique temporelle arborescente CTL (qui pouvait être caractérisée par des points fixes) et un algorithme de vérification automatique base sur les points fixes qu'ils ont nomme model-checking [CE82]. Ils ont ainsi développe le premier model-checker baptise EMC. En 1982, Sifakis et Queille ont aussi introduit la notion de model-checking [QS82] indépendamment des travaux effectues par Clarke et Emerson.

Par la suite, en 1986, le model-checking a été étudie pour LTL [MP92a], et un algorithme de vérification base sur la méthode des tableaux a été développe. Il s'agit en effet de transformer une formule LTL en automate de Buchi par décomposition de la formule en sous-formules en forme normale conjonctive. Se sont ensuivis des travaux sur des logiques plus expressives telles que CTL* [EH83] partant du constat que la logique LTL ne pouvait exprimer certaines formules CTL et réciproquement. Enfin, une logique plus expressive : la logique modale μ-calcul [Koz82] s'est vue décrite comme une unification des logiques temporelles, et lui a valu le nom de "théorie du tout".

- Principes du *model-checking*

Etant donne un système réel et une exigence de la spécification, le *model-checking* s'intéresse au problème de savoir si le système satisfait bien a l'exigence. La vérification automatique par *model-checking* s'effectue en trois étapes illustrées sur la figure 2.3 :

1. Etape 1 : Modélisation du système réel en modèle formel,
2. Etape 2 : Formalisation de l'exigence (écrite en langage naturel) en logique temporelle,
3. Etape 3 : Vérification par un algorithme.

Modélisation de systèmes complexes. Les systèmes réels étant complexes, une manière d'y faire face consiste à décomposer le système en plusieurs sous-systèmes. C'est à dire représenter le comportement du système global en plusieurs automates concurrents. Ensuite seulement, il s'agit de reconstruire le comportement du système complet en effectuant un produit synchronisé [Niv79,AN82], afin d'obtenir la dynamique du système global. Dans le cas des systèmes complexes, ce produit synchronisé peut générer une explosion combinatoire.

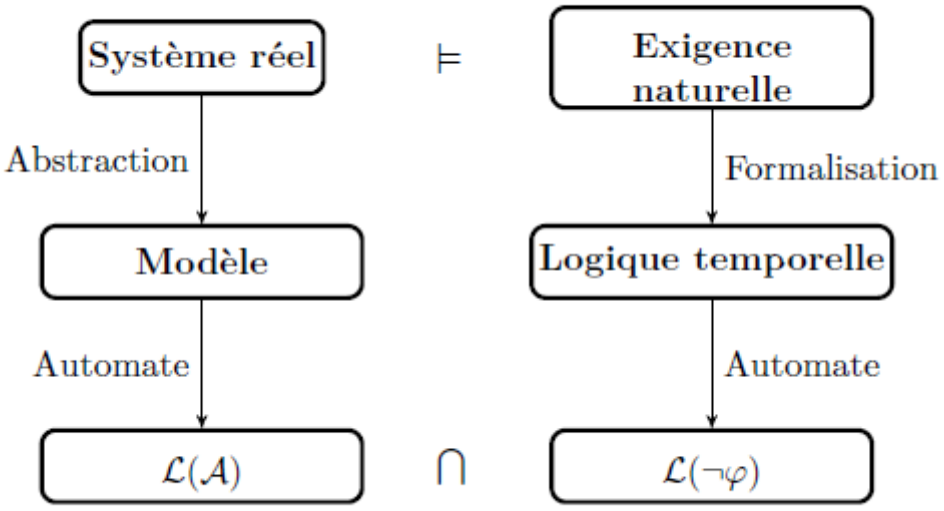


Figure 2.3 Etapes du *model-checking*

Remarque.

Il est à noter que la méthode du model-checking commence par effectuer une abstraction du système réel. Par conséquent une erreur du système réel ne peut être détectée dans le système, seulement si elle a été modélisée. Cette méthode à l'avantage d'être exhaustive, et permet donc de mettre en évidence la totalité des erreurs d'un système. De plus, un contre-exemple peut être généré lorsque la propriété à vérifier est falsifiée.

Etape 1 : modélisation du système par des Structure de Kripke.

Les systèmes sont en général modélisés sous forme d'automates. Toutefois, le model-checking consiste à vérifier des propriétés sur les états. Pour cela un modèle simplifié basé sur les propositions atomiques est utilisé :

Les structures de Kripke.

Une structure de Kripke est un automate particulier composé par un sex-tuplet $\mathcal{K}$ = (∑ ,A, T, I,AP, l) où :

- ∑ représente l'ensemble des états du système
- A représente l'ensemble des actions
- Tr ⊆ ∑ × A × ∑ représente l'ensemble des transitions
- I ⊆ ∑ contient l'ensemble des états initiaux
- AP est l'ensemble des propositions atomiques (c'est à dire des énoncés prenant les valeurs vraies ou fausses)

• l : S→2^{AP} est la fonction d'étiquetage qui associe à un état des propositions atomiques

Cette représentation permet de vérifier simplement des propriétés sur les états du système considéré

Exemple.

L'automate de la figure 2.4 représente un système de digicode qui garantit l'accès uniquement sur la suite d'actions ABA. La structure de Kripke K1 de ce système est définie par $K_1 = (\sum_1 A_1, Tr_1, I_1, AP_1, l_1)$:

- $\sum_1 = \{S_0, S_1, S_2, S_3\}$
- $I_1 = \{S0\}$
- $A_1 = \{A,B,C\}$
- $AP_1 = \{OPEN\}$
- Tr_1 est l'ensemble des 9 transitions
- $l_1(S_0) = \{OPEN\}$

La structure de Kripke est bien adaptée pour la vérification de propriétés d'états. Sur cet exemple, le nombre d'états et de transitions sont finis mais le système peut potentiellement avoir un comportement infini.

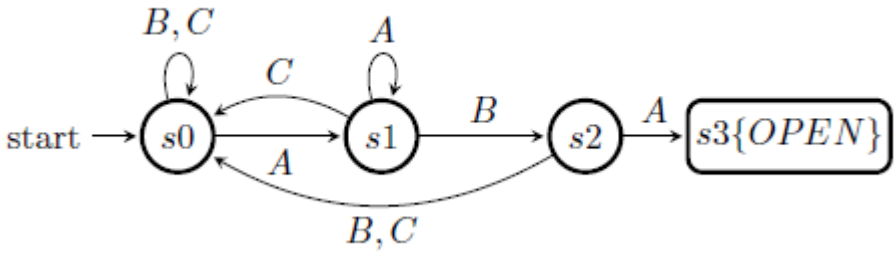


Figure 2.4 Structure de Kripke du digicode

Etape 2 :

Formalisation d'exigences avec les logiques temporelles. Même si nous travaillons sur des systèmes à états finis, nous cherchons souvent à vérifier des propriétés sur des exécutions potentiellement infinis, puisque l'on souhaite que la propriété reste vraie quel que soit le comportement du système.

De nombreuses logiques temporelles ont été développées comme citées précédemment. Parmi ces dernières trois fondamentales se distinguent : la logique linéaire (LTL) décrivant des propriétés sur des scénarios d'exécution, la logique arborescente (CTL) et la logique modale (μ-calcul) qui est la logique la plus expressive. Bien entendu certaines propriétés peuvent s'exprimer à l'aide de quantificateurs avec la logique de premier ordre qui est très

expressive. Le développement de divers langages de logique temporelle permet toutefois d'utiliser une syntaxe simple et l'expressivité adéquate aux propriétés que l'on souhaite vérifier. Par exemple, la logique LTL permet d'exprimer des propriétés sur des traces d'exécution. La logique CTL permet de formuler des propriétés sur un ensemble d'embranchement de traces. Par contre, la logique LTL ne peut exprimer certaines propriétés telle que : il y aura un nombre pair d'événement a. La logique CTL ne peut exprimer l'équité. Ainsi les logiques temporelles permettent d'exprimer un large panorama de propriétés parmi lesquelles : l'accessibilité, l'invariance, la sûreté, la vivacité, l'équité.

Remarque :

Dualité fini/infini. Pour pouvoir vérifier des comportements infinis (les propriétés) sur des systèmes finis (structure de Kripke) la méthode consiste alors à calculer les composantes fortement connexes maximales. En effet lorsqu'une composante fortement connexe est atteinte,alors, une infinité de fois, un même comportement peut s'effectuer. Dans la suite, les principaux algorithmes existant pour les logiques temporelles LTL, CTL et le μ-calcul sont brièvement présentés. Le lecteur intéressé par les algorithmes pour d'autres logiques temporelles telles que CTL pourra consulter l'ouvrage [CGP00].

Etape 3 **:** Algorithme de vérification. Il existe deux principales questions concernant la vérification :

- Le model-checking qui cherche à démontrer si un système satisfait la propriété donnée.
- La satisfiabilité qui cherche à démontrer s'il existe un modèle permettant de satisfaire la propriété.

Ces deux problèmes sont complémentaires. La satisfiabilité est très importante pour déterminer la sémantique des langages temporels et leur domaine d'expressivité et de validité. A partir de là,étant donné un automate et une propriété, il suffit de savoir si le modèle en question fait parti des modèles permettant de satisfaire la propriété, pour obtenir le diagnostic final.

On présente maintenant les principes utilisés pour la vérification de la logique linéaire et la logique arborescente. Comme énoncé précédemment, l'objectif est de vérifier des propriétés sur des exécutions infinies du système fini. Pour cela, la méthode classique consiste à associer le calcul des composantes fortement connexes avec une analyse d'accessibilité.

Pour le cas de la logique LTL, il s'agit de récrire la formule à l'aide de la méthode dite "du tableau" de manière à obtenir un automate de Büchi, en décomposant successivement la formule.

Pour le cas de la logique CTL, la méthode dite du marquage est utilisée : elle consiste à décomposer la formule générale en sous-formules. Pour chaque sous-formule, il s'agit de

marquer les états qui la vérifie. Ce procédé est réitéré sur un ensemble de sous-formules. Au final les sous-formules représentent la formule globale et la méthode du marquage se termine. Cette méthode permet de vérifier des propriétés écrites sous forme de point fixe [Tar55].

- Obstacle au model-checking.

Bien qu'élégante, la méthode du model-checking se confronte à un problème fondamental : le problème de l'arrêt qui découle de la théorie du calcul et de la complexité [Sip96], qui s'adonne à la question suivante : "qu'est ce qui est calculable efficacement et qu'est ce qui ne l'est pas ?" indépendamment de toute technologie. Pour cela if faut remonterdans les années 40 et les travaux innovants de Alan Turing (avec la célèbre machine à penser), Alonzo Church et Kurt Gödel. Le problème peut être défini de la manière suivante : étant donnée une machine de Turing [Tur37] (qui est un modèle de calcul3 permettant de définir la sémantique d'exécution d'un programme) avec une bande infinie, et la propriété : est ce que la machine termine un jour (sur un espace blanc par exemple représenté par le caractère 'b') , il a été démontré que ce problème connu sous le nom de problème de l'arrêt ou encore "the halting problem" est indécidable ; c'est à dire qu'il n'existe pas de fonction caractéristique répondant systématiquement par "vrai" ou "faux" excepté pour les cas triviaux. Cela est le fruit du théorème de Rice. De tels théorèmes mettent à néant les espoirs de vérification automatique. De manière générale, vérifier si un programme fait exactement ce que l'on souhaite est un problème indécidable.

Néanmoins, ce problème concerne les machines possiblement infinies. Le fait de se restreindre à des machines moins expressives avec les structures de Kripke permettent d'introduire la vérification automatique, en ne prenant en compte, que des machines à états finis.

En résumé, étant donné un programme, il n'existe aucun programme permettant de décider systématiquement dans des délais raisonnables si un programme boucle ou non, quelque soit la puissance de calcul de l'ordinateur. Même l'accessibilité d'un état de contrôle n'est pas décidable.

La machine ne pourra donc pas remplacer totalement l'activité humaine. Toutefois, dans le cas où un problème reste décidable, le phénomène de l'explosion combinatoire empêche une vérification exhaustive du programme.

Pour pouvoir dépasser ces limites, il faut faire une abstraction du modèle de conception en prenant des modèles moins réalistes (d'où le nom model-checking) par exemple en éliminant des informations inutiles : les modèles finis, dans lesquels les états et les transitions du système sont énumérables.

- L'expression des propriétés.

Un autre point délicat lors de l'application du model-checking, est la nécessité d'exprimer la propriété à vérifier dans une logique temporelle telle que LTL ou CTL, qui reste une étape délicate et demandant beaucoup d'expertise. C'est pourquoi la méthodologie proposée par l'ENSIETA consiste à se restreindre à des classes d'accessibilité plus faciles à démontrer, et qui peuvent facilement être modélisées sous formes d'observateurs. La vérification par les observateurs se ramène ainsi à vérifier, si un état correct est atteint un jour (état de succès) :happy, ou bien si un état non désiré n'est jamais atteint (état de rejet) : (¬unhappy). Dans la suite on se restreint à cette classe de propriétés.

- Discussion sur le model-checking

La méthode générale du model-checking qui consiste à explorer l'ensemble des états du système global après composition de ses sous-composants, présente l'avantage d'être exhaustive ; encontre-partie, elle n'est vraiment efficace que pour certains systèmes dont la complexité est raisonnable. Dans le cas de systèmes complexes, une explosion combinatoire est obtenue sur le nombre d'états. Or, les systèmes embarqués mettent en oeuvre l'asynchronisme, la concurrence et le non déterminisme (résultant de l'asynchronisme de cette concurrence). Ces critères augmentent d'autant plus les possibilités combinatoires, par les nombreux entrelacements engendrés par les événements des processus concurrents.

Le model-checking dans le milieu industriel. Dans le cas du génie logiciel, le model-checking est appliqué sur des "modèles" de conception (par exemple une spécification SDL) à partir desquels le code est généré par des outils certifiés, préservant les propriétés à vérifier. Ce mémoire se positionne au niveau de la conception détaillée (figure 2.3) et prend comme point de départ, les modèles de conception réalisés dans le langage SDL par l'équipe de spécification formelle de l'entreprise CS-SI.

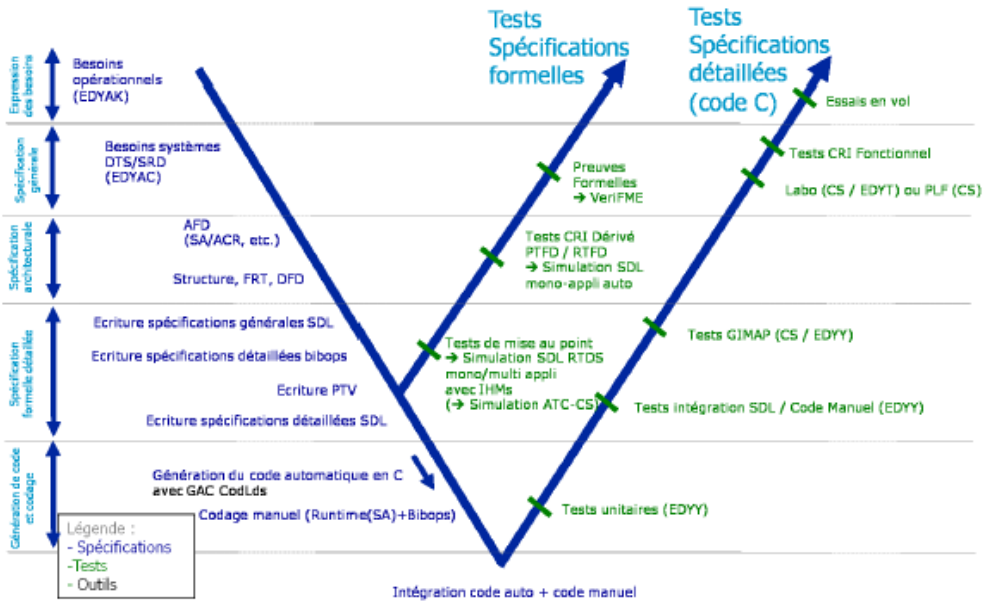


Figure 2.5 Positionnement du mémoire dans le cycle en V

Néanmoins, le model-checking atteint rapidement ses limites, lorsque l'on veut valider des systèmes de taille industrielle. Pour pallier à cette contrainte, de nouvelles techniques basées sur les avancées du model-checking ont été réalisées, en particulier le model-checking "à la volée", les méthodes d'abstraction et les Diagrammes de Décisions Binaires : les BDDs [Bry92].

- 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 ?» [OF07].

- 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 [OF07].

- 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 [OF07].

2.1.3 La logique temporelle temporisée

La "Timed Computation Tree Logic" étend la CTL en intégrant une notion de temps borné.

Ainsi des formules peuvent par exemple être vérifiées dans un état pendant un certain tempst borné avant qu'elles ne soient invalidées. La logique TCTL permet, grâce à cette notion, de vérifier des propriétés des systèmes à temps réel où les séquences d'exécutions ne sont pas finies.

Cette logique est très souvent associée aux automates temporisés.

- Etant donné un système décrit sous la forme d'un réseau d'automates temporisés, on souhaite pouvoir énoncer (et ensuite vérifier) des propriétés sur ce système. Parmi ces propriétés, on trouvera des propriétés simplement temporelles, mais aussi des propriétés temps réel qui mettent en jeu l'information quantitative sur les délais séparant les actions.
- Pour énoncer formellement de telles propriétés temps réel, plusieurs approches sont possibles. Le plus simple est d'exprimer la propriété en termes d'atteignabilité (ou de non-atteignabilité) de certains ensembles de configurations de l'automate.
- Pour des propriétés plus compliquées, on peut aussi utiliser la technique des automates observateurs : étant donné une propriété *p* et un réseau *R*, on ajoute à *R* un nouvel automate *Ap* se synchronisant sur certaines actions de *R*. Vérifier *p* se ramène alors à tester l'atteignabilité de certains états du produit $R \parallel Ap$. cette méthode est souvent utilisée dans les outils UPPAAL et HYTECH.
- Une autre possibilité est d'utiliser une logique temporelle temporisée, c'est à dire l'extension ou l'adaptation d'une logique temporelle par des primitives permettant d'exprimer des conditions sur les durées et les dates. Cette approche est sûrement plus satisfaisante dans la mesure où elle fournit un langage logique pour spécifier les propriétés temporisées.

  - 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∨ g) et (¬f) sont des formules de TCTL . S3 : si f est une formule de chemin alors E f U ∼ c g et A fU ∼ c g sont des formules de TCTL, avec ∼∈ {<,<=, =,>,>=} et c∈ N. c peut être un intervalle de temps[OF07] .

  - Sémantique TCTL

Pour le modèle M(G) du système de transition $(Q,q_0,-\rightarrow)$ de l'automate temporisé G, on définit la relation de satisfaction d'une formule TCTL en un état q = (s, v),q ∈Q par induction:
– q |= p si et seulement si p ∈ P(s) i.e la proposition p est associée à s.
– q |= x ≺ c si et seulement si v(x) ≺ c i.e la valuation de x respecte cette contrainte d'horloge.

– q |= x−y ≺ c si et seulement si v(x)−v(y) ≺ c i.e la valuation de x moins celle de y respecte cette contrainte d'horloge.

– q |= ¬φ si et seulement si ¬(s |= φ) i.e s'il est faut que le sommet s puisse vérifier la formule φ.

– q |= φ1 ∨φ2 si et seulement si (s |= φ1)∨(s |= φ2)

– q |= ∃φ1 ∪≺c φ2 si et seulement si il existe au moins une séquence (q0,q1,...) avec q = q0 tel que quelque soit (i, j), 0 ≤ i ≤ j, qi |= φ1 et qj |= φ2 avec un temps entre q0 et qj satisfaisant la relation ≺ c. Autrement dit, s'il existe une séquence où φ1 est toujours vraie jusqu'à un état, situé dans un temps respectant la contrainte ≺ c où φ2 est vraie.

– q |= ∀φ1 ∪≺c φ2 si et seulement si pour toutes les séquences (q0,q1,...) avec q = q0 tel que quelque soit (i, j), 0 ≤ i ≤ j, qi |= φ1 et qj |= φ2 avec un temps entre q0 et qj satisfaisant la relation ≺ c. Autrement dit, si pour toutes les séquences φ1 est toujours vraie jusqu'à un état, situé dans un temps respectant la contrainte ≺ c où φ2 est vraie.

Remarque **:** Ici, l'état q d'un automate temporisé est un couple (s, v) alors que dans la logique CTL l'état d'un automate se réduisait à un sommet de son graphe.

Remarque **:** L'opérateur (suivant) n'existe pas en TCTL. En effet, cette logique travaillant sur du temps borné et réel, il peut y avoir une infinité d'état entre deux états d'une séquence. De ce fait la notion de suivant n'a plus de sens.

Expressivité (tiré de [Cha03])

Propriété	Code	Formule
Sûreté	S1	$\forall\Box\neg(SC1 \wedge SC2)$
	S2	$\forall\Box\neg Dbordement$
Vivacité	V1	$\forall\Box(req \Rightarrow \exists\Diamond sat)$
	V2	$\forall\Box init$
Vivacité Borné	VB1	$\forall\Box(req \Rightarrow \exists\Diamond_{<5} sat)$
	VB2	$\forall\Box\exists\Diamond_{<3} vert$

Table 2.2 Propriétés temps réel en TCTL.

2.2 PARTIE 2 : UPPAAL

UPPAAL [BEN96, PET00, DAV03] est un outil pour la modélisation, la validation et la vérification des systèmes temps réel. Il est approprié pour les systèmes qui peuvent être modélisés par des automates temporisés ou par des automates hybrides linéaires de classe LHS[OLI94a]. Autrement, UPPAAL est approprié pour les systèmes qui peuvent être modélisés sous la forme de processus non-déterministes avec une structure de contrôle finie et des horloges avec des valeurs réelles, communiquant par des canaux ou des structures de données partagées.

Nous présentons dans ce volet UPPAAL, sa syntaxe et sa sémantique. Nous illustrons, par la suite, sur quelques exemples de comportements.

L'outil UPPAAL

UPPAAL [BEN98] est un ensemble d'outils pour la vérification automatique des propriétés de sûreté et de vivacité bornée, des systèmes temps réel. Il est construit avec l'architecture Client/Serveur (Figure 2.6). La machine UPPAAL est le serveur, elle est développée en C++. L'interface graphique utilisateur (GUI) est le client, développé en JAVATM. La communication s'effectue via des protocoles internes. Ainsi, la conception d'UPPAAL offre la possibilité d'exécuter le serveur et l'interface GUI sur deux machines différentes. La machine UPPAAL est constituée de plusieurs outils tel que checkta (Syntax Checker) et verifyta (Model Checker). L'interface graphique utilise des outils tel que atg2ta et hs2ta. La collaboration entre ces outils est illustrée par la(figure 2.8) [BEN95].

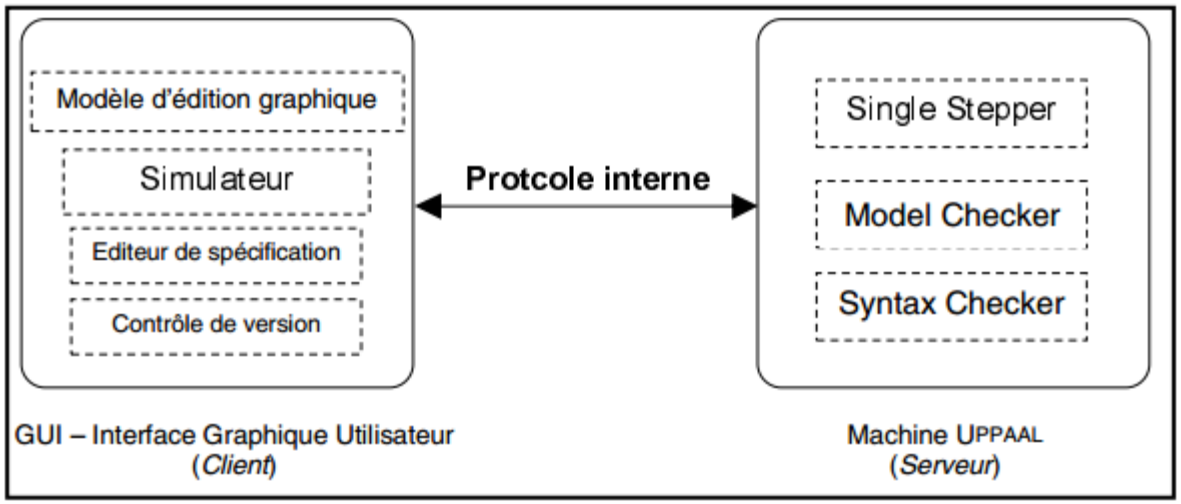


Figure 2.6 Vu d'ensemble d'UPPAAL

Depuis son développement UPPAAL [DAV03] a connu des améliorations en vue de supporter des fonctionnalités désirées qui sont de plus en plus complexes, et d'offrir la

rapidité et la flexibilité. La nouvelle architecture est constituée par des couches qui peuvent se communiquer.

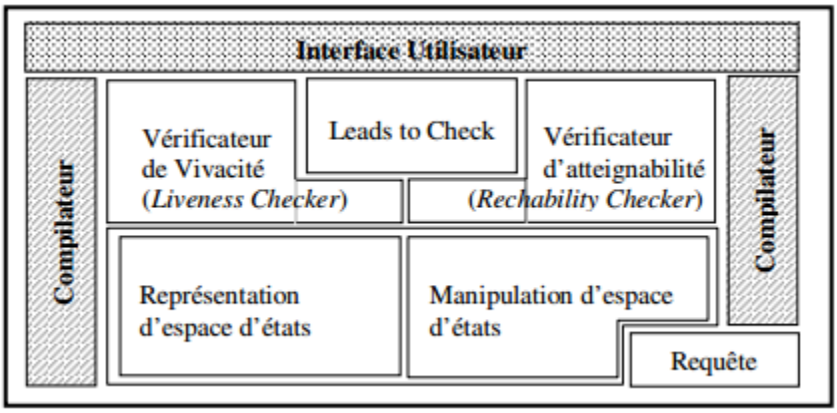


Figure 2.7 Architecture d'UPPAAL

• Fonctionnalité :

atg2ta : Ce programme est responsable de la transformation graphique des descriptions d'autographe [BOU96b] en format textuel.

Hs2ta : Ce programme est capable de transformer des systèmes hybrides simples en réseaux d'automates temporisés. Les systèmes considérés doivent être une sous classe de LHS□, où la vitesse des variables est indépendante de l'état du système.

Checkta : c'est un programme assurant la vérification syntaxique de la modélisation d'un système donnée en format textuel. La description doit être sous la forme d'automates temporisés. S'il s'agit de système hybride linéaire, il doit être d'abord transformé en système temporisé en utilisant hs2ta.

Verifyta : Ce programme est le noyau de la vérification dans UPPAAL. Il reçoit en entrée la description du système et une propriété à vérifier. Il répond par "oui" ou "non". Il permet aussi de générer une trace qui confirme ou viole la propriété à vérifier.

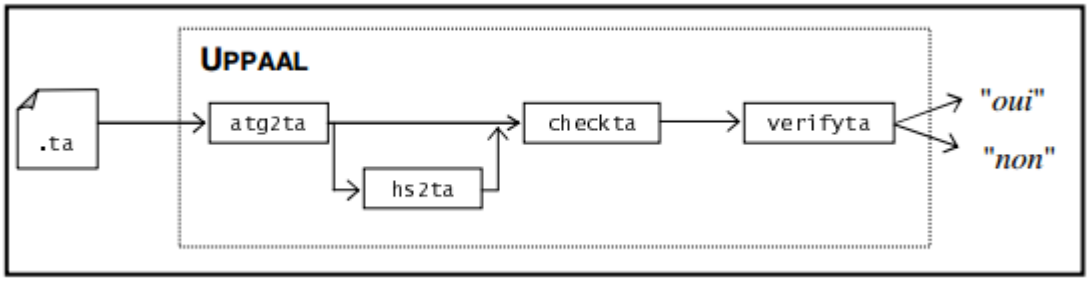


Figure 2.8 Aperçu d'UPPAAL

Pour UPPAAL [LAR97], un système temps réel typique est un réseau non-déterministe de processus séquentiels communiquant les uns avec les autres via des canaux. UPPAAL utilise les automates à états-fini étendus avec des horloges et des variables de données pour décrire les processus et les réseaux d'automates relatifs au système temps réel traité.

La base du modèle d'UPPAAL est la notion des automates temporisés, développée par Alur et Dill [ALU94], comme extension des automates à états-finis classiques avec les variables d'horloges et de données. Pour avoir une modélisation plus expressive et pour la rendre plus facile, les automates temporisés sont étendus avec des types plus généraux des variables de données tel que les variables booléennes et entières. Le but est le développement d'un langage de modélisation le plus proche que possible d'un langage de programmation des systèmes temps réel de haut niveau. Clairement, ceci va créer des problèmes de décidabilité. Cependant, on peut exiger que le domaine de valeurs des variables de données doive être fini afin de garantir la terminaison de la procédure de vérification.

Les transitions de l'automate temporisé, dans UPPAAL, sont étiquetées par trois types d'étiquettes (Figure 2.13) : (a) les gardes, exprimés sur les valeurs d'horloges et les variables entières qui doivent être satisfaites dans l'ordre pour les transitions qui vont être franchies ; (b) une action de synchronisation qui est exécutée quand les transitions sont franchies ; et finalement (c) les attributions (assignement) des variables entières. Tous ces trois types sont optionnels.

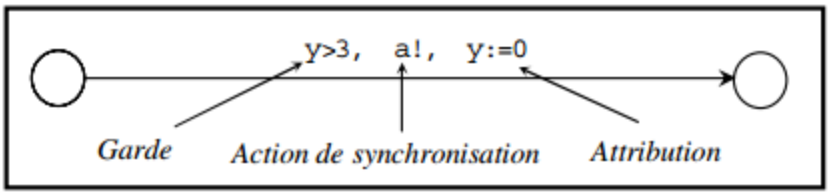


Figure 2.9 : Etiquettes d'une transition dans UPPAAL

En plus, les nœuds de contrôles peuvent être étiquetés avec des invariants, qui sont des conditions exprimant des contraintes sur les valeurs d'horloges pour demeurer dans un nœud de contrôle particulier.

Les gardes expriment les conditions sur les variables d'horloges et des variables entières qui peuvent être satisfaites dans l'ordre de franchissement des transitions. Formellement, les gardes sont la conjonction des contraintes temporisées et des contraintes de données. Une contrainte d'horloge est de la forme : $x \sim n$ ou $x - y \sim n$, où n est un nombre naturel et $\sim \in \{ \leq, \geq, =, >, < \}$. Une contrainte de données est de la même forme, $i \sim j$ ou $i - j \sim k$, sauf que k est un entier arbitraire. La garde d'une transition est par défaut vrai. Dans la (figure 2.11) , la

transition entre A0 et A1 n'est franchie que si la valeur de l'horloge est supérieure ou égale à 3.

La remise à zéro d'une horloge ou d'une variable de données sur une transition est une affectation de la forme w := e, sachant que w est une horloge ou variable de donnée et e une expression. L'opération de remise à zéro d'une horloge est de la forme x := n, où n est un nombre entier. La remise à zéro des variables de données est de la forme i := c * i + c', où c,c' sont des constantes entières (sachant que c et c' peuvent être nuls ou négatifs).

- Canaux, synchronisation et urgence :

Un modèle UPPAAL consiste en un réseau d'automates temporisés (étendu). Les automates peuvent communiquer via des variables entières (dans UPPAAL elles sont globales) ou en utilisant des canaux de communication, permettant l'envoi et la réception d'un message. La communication sur un canal apparaît comme étant la synchronisation entre deux processus. a! (Envoie) et a? (Réception) dénotent qu'un processus synchronise avec un autre. L'absence de l'action de synchronisation dénote une transition (de non synchronisation) interne. Pour empêcher un automate de s'attarder dans une situation, où deux composants sont capables de synchroniser, il faut déclarer le canal comme étant urgent. Pour des raisons d'efficacité, les transitions sont étiquetées avec des actions de synchronisation sur des canaux urgents, et elles doivent être privées des gardes sur les horloges pour que les transitions de deux processus puissent être franchies simultanément et ne pas bloquer un système synchronisé avec un autre.

- Emplacement comité (committed location) :

Soit un émetteur S qui émet un message m à deux destinataires R1 et R2 (Figure 2.10). La synchronisation entre ces trois processus ne peut pas être directement exprimée en UPPAAL. Elle ne peut être effectuée qu'entre deux processus uniquement. Cependant, l'émission sera modélisée sous la forme d'une séquence de synchronisation entre deux processus, où S synchronise avec R1 sur m1 puis avec R2 sur m2. Pour assurer l'atomicité de la synchronisation. On marque le nœud intermédiaire comme comité (indiqué par C).

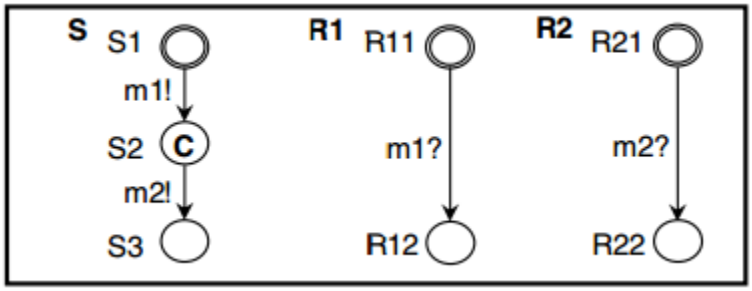


Figure. 2.10 Transmission de communication et Emplacement comité

• Invariant :

Pour renforcer la progression dans le temps, les nœuds de contrôles doivent être étiquetés par des invariants, qui expriment dans l'ordre les contraintes sur les valeurs d'horloges pour le nœud de contrôle, afin qu'il puisse y résider..

- Sémantique

Formellement, les états du modèle UPPAAL sont de la forme (l, v), où pour chaque composant du réseau d'automate, l est un vecteur de contrôle indiquant le nœud de contrôle en cour, et pour chaque horloge et variable entière v est un assignement indiquant la valeur en cours.

- Transition de délai :

Le temps doit évoluer, sans l'affectation des vecteurs des nœuds de contrôles et en incrémentant les valeurs des horloges avec la durée du temps écoulé, aussi longtemps qu'aucun invariants des nœuds de contrôle n'est violé.

- Transition d'action :

Si deux transitions complémentaires de deux composants différents sont rendues actives, dans un état, alors elles peuvent synchroniser.

- Canaux urgents :

Aucun délai n'est permis dans un état, où deux composants doivent synchroniser sur un canal urgent. Ainsi, dans la Figure 2.15, si un canal a est urgent, alors le temps ne doit pas s'écouler pendant 3,5 secondes à partir de l'état initial ((A0,B0), x = 0, y = 0, n = 0) puisque la synchronisation sur a est possible dans l'état ((A0,B0), x = 3, y = 3, n = 0). 2.4.4. Exemples Nous présentons dans ce qui suit quelques exemples pour la modélisation graphique et textuelle des systèmes dans UPPAAL. Exemple général Nous illustrons la modélisation graphique et textuelle avec UPPAAL sur un exemple général.

2.2.1 Modélisation Graphique

La (figure 2.11) présente un système constitué par deux composants modélisés par les automates A et B. Ces deux automates synchronise via le canal a.

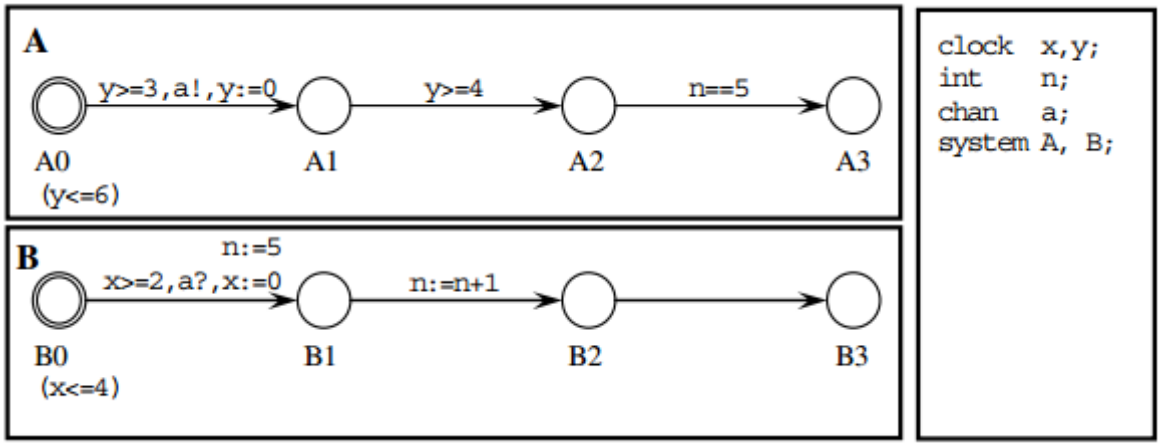


Figure 2.11 Exemple d'un modèle UPPAAL

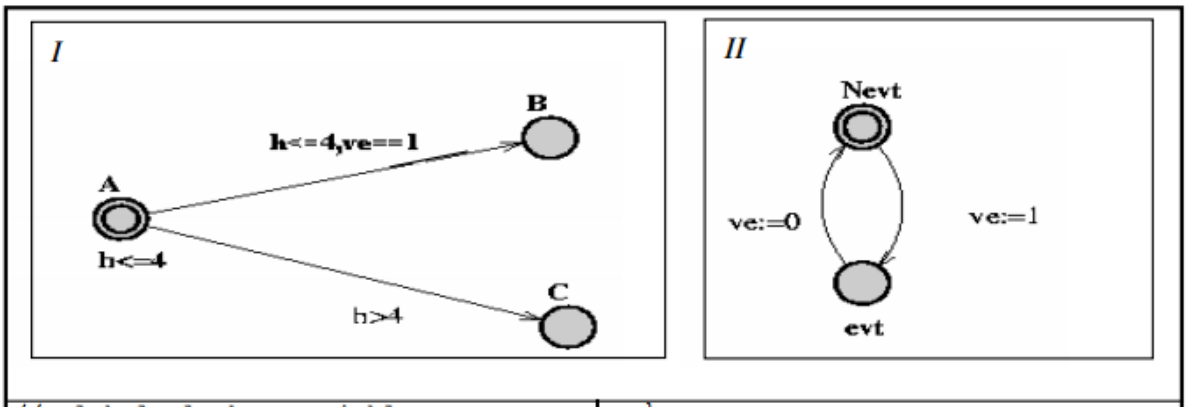
2.2.2 Modélisation Textuelle

L'exemple (Figure. 2.12) suivant illustre la traduction du modèle de la figure 2.2.6 en description textuelle dans UPPAAL.

```
// Déclaration Globale

clock x, y ;
int n ;
chan a ;

// Description des composants

process A {
  state A0 { y<=6 }, A1, A2, A3 ;
  init   A0 ;
  trans A0 -> A1 {guard y>=3 ;
      sync  a !;
      assign      y :=0 ;     },
      A1 -> A2 {
        Guard n==5 ; } ; }

process B {
   state B0 { x<=4 }, B1, B2, B3 ;
   commit  B1 ;
   init    B0 ;
   trans  B0 -> B1 {guard x>=2 ;
      sync  a ?;
      assign n :=5,x :=0 ; },
      B1 -> B2 {Assign n :=n+1 ;},
      B2 -> B3  {} ; }

// Description du Système
system A, B ;
```

Figure. 2.12 : Description textuelle en UPPAAL.

Exemple du TimeOut La modélisation graphique et textuelle de l'exemple du Timeout est donnée dans la figure 2.13.




```
// global clocks, variables,
constants and channels.
clock h;
int ve;

// Process Control
process Control{
state A{h<=4}, B, C;
init A;
trans A -> B{guard ve==1; },
A -> C{guard h>4;
}
// Process Event
process Event{
state Nevt, evt;
init Nevt;
trans Nevt -> evt{assign ve:=1; },
evt -> Nevt{assign ve:=0; };
}
// System definition.
system Control,Event;
```

Figure 2.13 Modélisation graphique et textuelle du Timeout

22 Vérification dans UPPAAL

UPPAAL est capable de vérifier les propriétés d'atteignabilité, de sûreté, de vivacité simple, de vivacité bornée, et de non-blocage.

- Logique temporelle temporisée :

Les propriétés qui peuvent être analysés sont de la forme :

$$\phi ::= \forall \ \Box \beta \mid \exists \Diamond \beta \mid \exists \Box \beta \mid \forall \Diamond \beta \quad \beta ::= a \mid \beta 1 \wedge \beta 2 \mid \neg \beta$$

Où a est une formule atomique qui peut être une contrainte atomique d'horloge ou de donnée.

3 Model-Checking

UPPAAL utilise une structure de donnée appelée Diagramme de différence d'horloge (Clock Différence Diagram), CDD [BEH99, LAR99, LAR03]. Cette structure est semblable aux BDD, sauf qu'elle projeté une représentation et une manipulation efficace du sous-ensemble convexe de l'espace durant la vérification des automates temporisés.

Plusieurs techniques de vérification ont été testées et utilisées dans UPPAAL telle que l'atteignabilité compositionnelle en arrière CBR (Compositional Backwards Reachability) [NIE99]. Cette technique utilise l'analyse de dépendance et de compositionnabilité pour améliorer l'efficacité de la vérification symbolique du modèle État/Évènement. Deux nouveaux algorithmes de vérification d'atteignabilité sont proposés dans [BEH02]. Le premier algorithme pour des machines classiques (Figure 2.14). Il est plus rapide que ceux qui l'ont précédé. En fait, il permet d'éliminer la duplication des états. Le deuxième algorithme

est destiné pour les machines parallèles et distribuées (Figure 2.15). Dans cet algorithme l'espace d'état est distribué sur un nombre d'état utilisant une fonction de hachage.

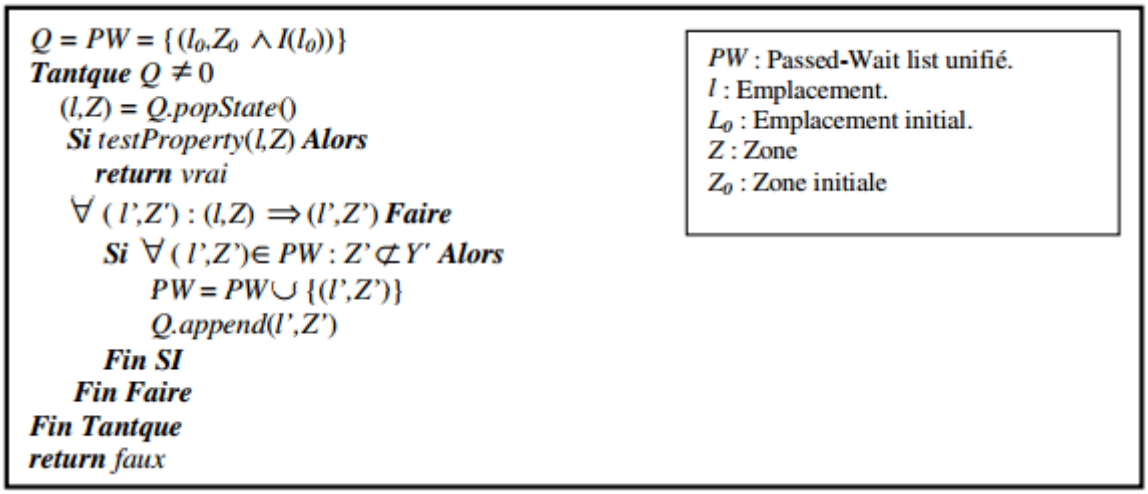


Figure. 2.14 : Algorithme de vérification d'atteignabilié.

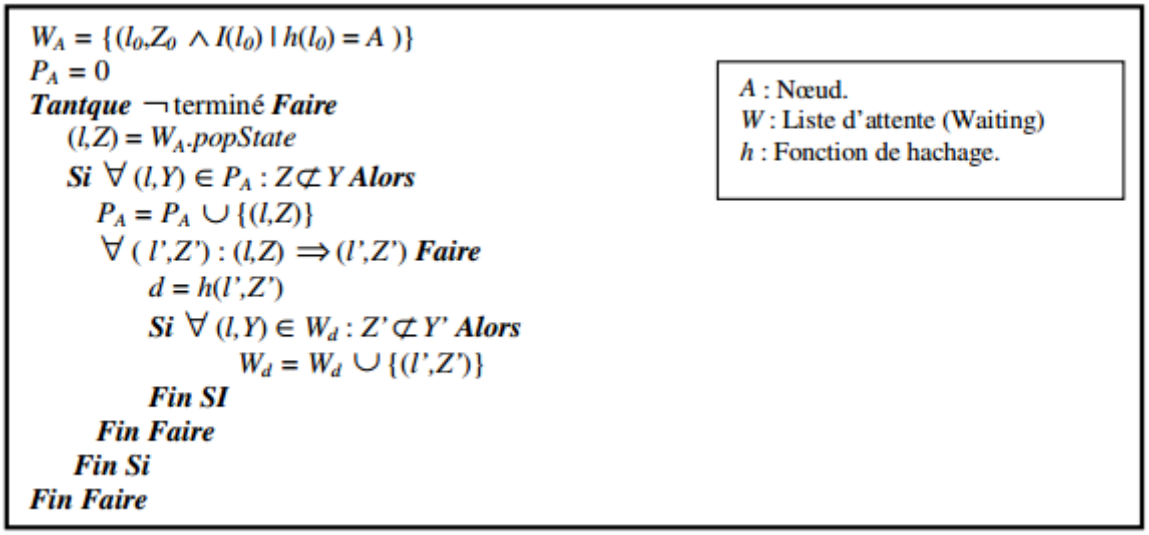


Figure. 2.15 Algorithme de vérification distribuée et parallèle d'atteignabilité

Notons qu'UPPAAL donne à l'utilisateur le choix entre les points suivants, avant de procéder à la vérification d'un système.

- Recherche
  - ✓ En profondeur
  - ✓ En largeur

- Réduction d'espace d'état
  - ✓ Sans réduction
  - ✓ Agressive
  - ✓ Conservative

- Représentation de l'espace d'état

- ✓ DBM o Structure de Donnée Compacte
- ✓ Sous-Approximation
- ✓ Sur-Approximation

❖ Réduction d'horloge

Conclusion

UPPAAL est un outil de modélisation et de vérification des systèmes temps réel. Il offre une interface graphique et un simulateur facilitant l'assistance à la vérification pour l'utilisateur. Il permet de modéliser les systèmes temporisés ainsi que les systèmes hybrides linéaires. Il permet aussi d'assurer la vérification des propriétés d'atteignabilité, de vivacité simple et borné, et de non-blocage.

Chapitre 3

CONTRIBUTION

3 CHAPITRE 3 : CONTRIBUTION

FDDI (Fiber Distributed Data Interface)

3.1 Introduction

Le besoin croissant en bande passante a créé le développement de nouveaux standards qui se sont imposés. FDDI fait partie de ces standards. FDDI suit la norme ISO 9314 (ANSI X3T9.5) qui a été standardisé dans le milieu des années 1980. Ce type de réseau est fréquemment utilisé comme Backbone pour des réseaux locaux ou leurs interconnexions. Ce document décrit cette norme en quelques pages aussi bien du point de vue technique que du point de vue économique, ainsi que les évolutions …[5].

3.2 Définition

FDDI (Interface de Données Distribuées sur Fibre) est un type de réseau Token Ring. L'implémentation et la topologie FDDI est différente de celles d'une architecture de réseau local Token Ring d'IBM. L'interface FDDI est souvent utilisée pour connecter différents bâtiments au sein d'un campus universitaire ou d'une structure d'entreprise complexe. Les réseaux FDDI fonctionnent par câble en fibre optique. Ils allient des performances haute vitesse aux avantages de la topologie en anneau avec passage de jeton. Les réseaux FDDI offrent un débit de 100 Mbits/s sur une topologie en double anneau. L'anneau extérieur est appelé anneau primaire et l'anneau intérieur c'est anneau secondaire[6].

3.3 Pourquoi FDDI ?

FDDI répond à trois besoins simples [7]:

- Le besoin d'interconnexion des réseaux locaux par des réseaux fédérateurs.
- Des débits élevés évitant ainsi tout goulot d'étranglement.
- Raccordement de stations à haut débit (Visioconférence, Vidéo, Son en temps réel …).

3.4 Avantages du FDDI

- Technologie éprouvée
- Fort débit
- Conserve la structure existante (Ethernet, Token Ring)

  ⇒ Surcoût d'installation faible

- Existe aussi en version TP-DDI

  ⇒ sur paire torsadée

- Contraintes de temps prises en compte dans V 2

3.5 Les caractéristiques de FDDI

Débit nominal	100 Mbit /s
Type de trafic	Synchrone / Asynchrone
Distance	200 Km de longueur de fibre, soit 100 Km de distance Max
Diamètre de l'anneau	31 Km sous forme de boucle
Distance maximale entre les nœuds (stations)	2 Km
Taille des trames	£ 4 500 octets
Transmission	Bande de base et codage des données 4B/5B – NRZI
Méthode d'accès	Jeton temporisé sur boucle
Architecture	Double anneau, reconfiguration en cas de défaillance de l'un des anneaux
Topologie	Double anneau en fibre optique utilisant la technique du jeton
Support physique	• Fibre optique multimode 62,5/125 • Monomode
Module de gestion	Intégré au réseau : Station management (SMT)
Nombre de stations	500 à 1000 stations suivant la classe

Table 3.1 Les caractéristiques [7]

3.6 . Les rôles de FFDI :

Backbone : . Interconnexion de réseaux locaux (réseau fédérateur)
. Interconnexion de serveurs (de stockage, de traitement, etc) [BC 99].

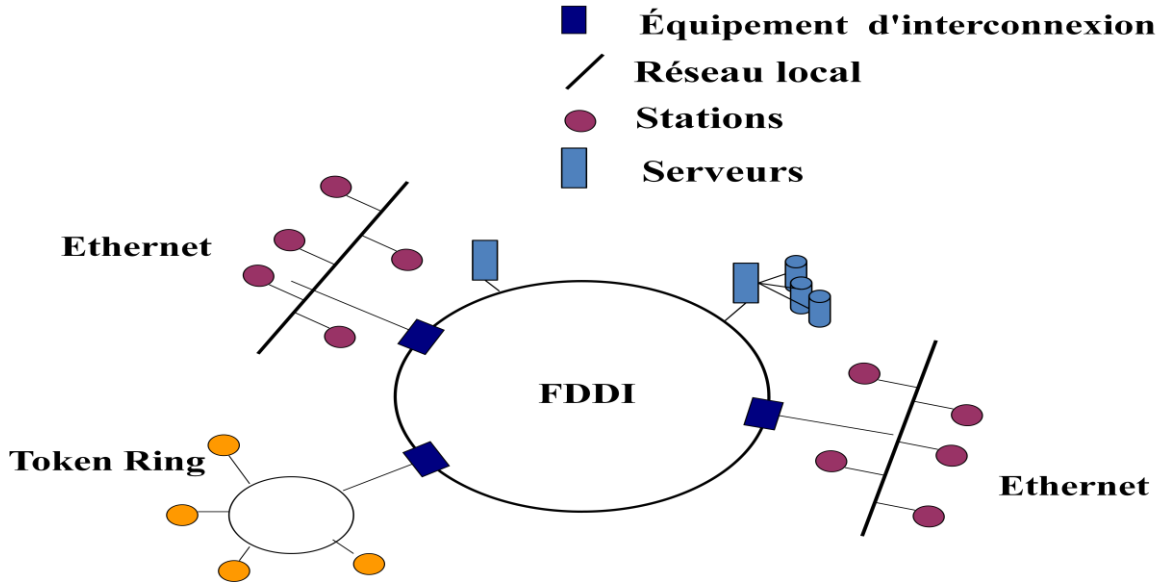


Figure 3.1 Principe FDDI

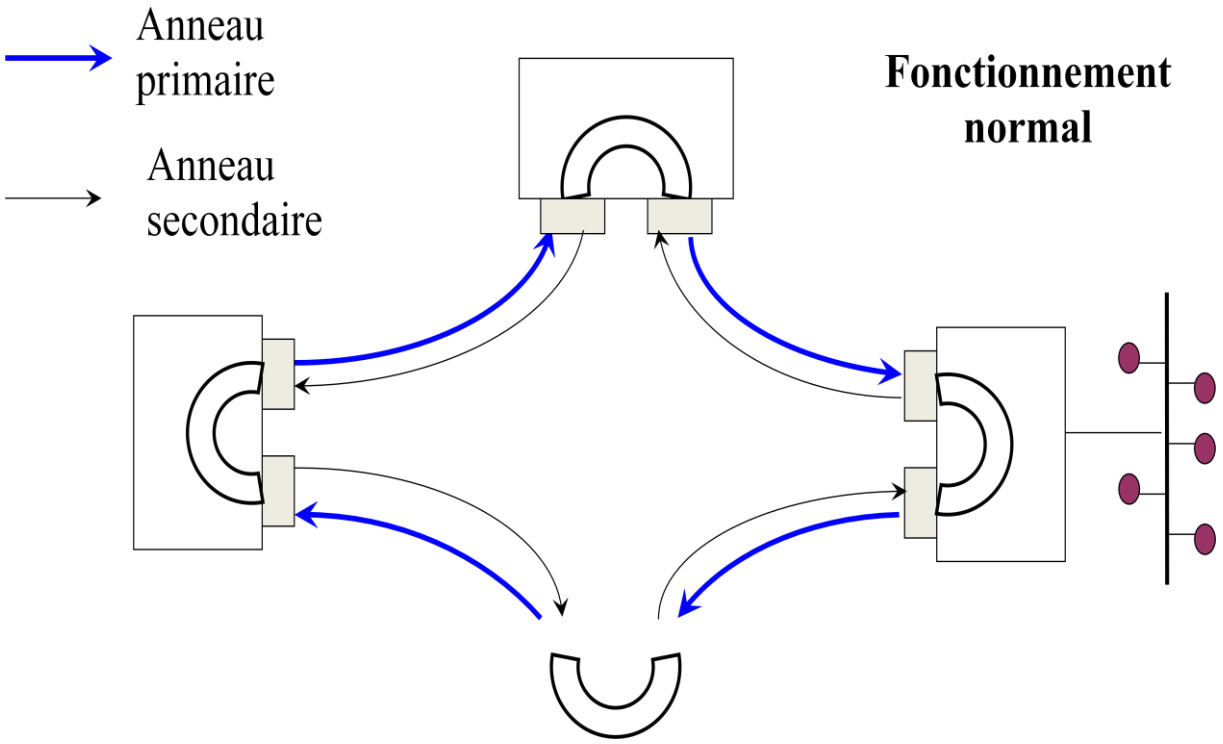


Figure 3.2 Principe FDDI en fonctionnement normal

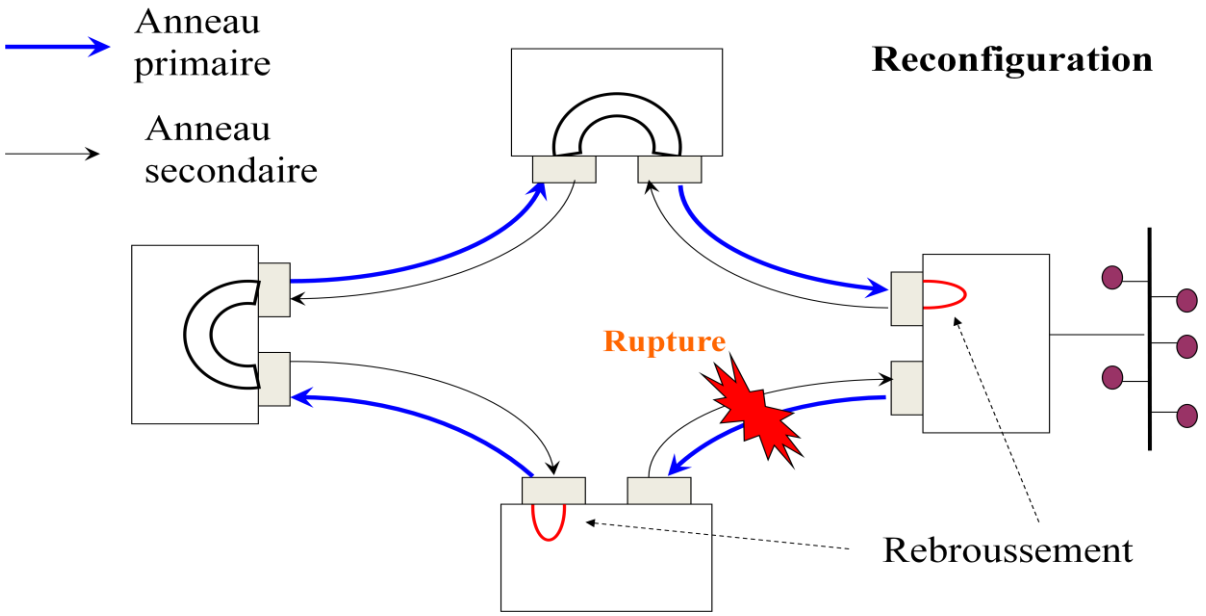


Figure 3.3 Principe FDDI en reconfiguration

3.7 La standard FDDI

3.7.1 Topologie

FDDI fonctionne selon une topologie logique en anneau. Les machines peuvent être interconnectées soit en étoile à la sortie d'un concentrateur, soit directement sur l'anneau (cette dernière possibilité est plutôt réservée aux serveurs et aux stations de travail rapides, vus les prix des adaptateurs correspondants) [7].

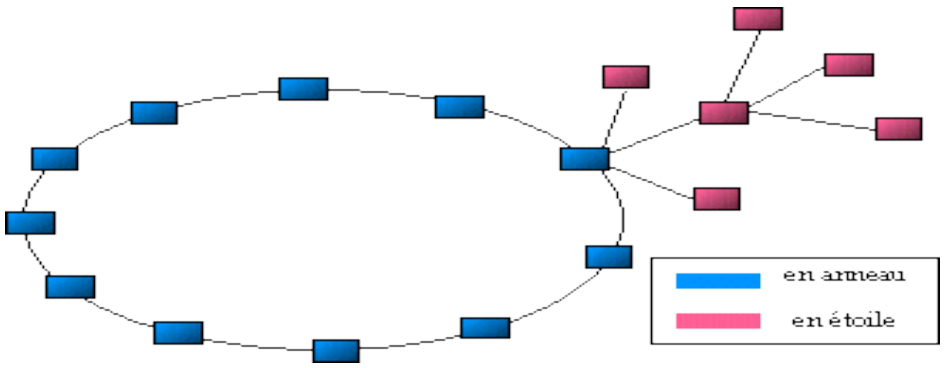


Figure 3.4 Topologie de FDDI

Les données circulent normalement sur l'anneau principal; en cas de défaillance, le trafic bascule automatiquement sur l'anneau secondaire (dit de secours). Certains constructeurs de matériels proposent des variantes qui mettent les deux anneaux à contribution; ce procédé permet de doubler la bande passante[7].

Les équipements disposent de connecteurs pour accéder à l'un ou l'autre des anneaux, voire au deux. Ces équipements sont répartis en trois classes[7]:

Classe A	*Les stations reliées aux deux anneaux simultanément. *DAS: Dual Attachement Station.
Classe B	*Les stations reliées à un seul anneau. *SAS: Single Attachement Station.
Classe C	Les concentrateurs FDDI

Table 3.2 Les classes de FDDI

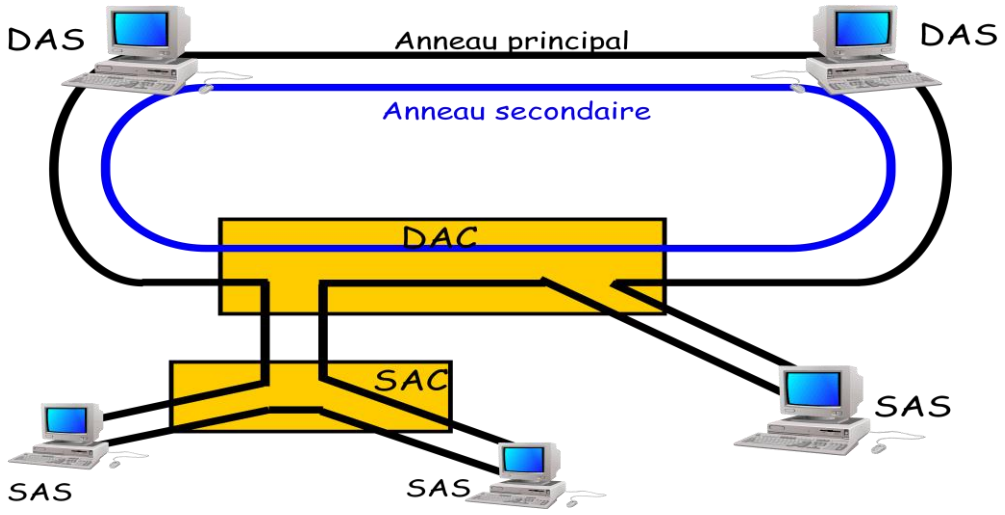


Figure 3.5 Les stations FDDI

3.7.2 Intégration au modèle OSI

FDDI divise les couches Physique et Liaison du modèle OSI en deux sous-couches

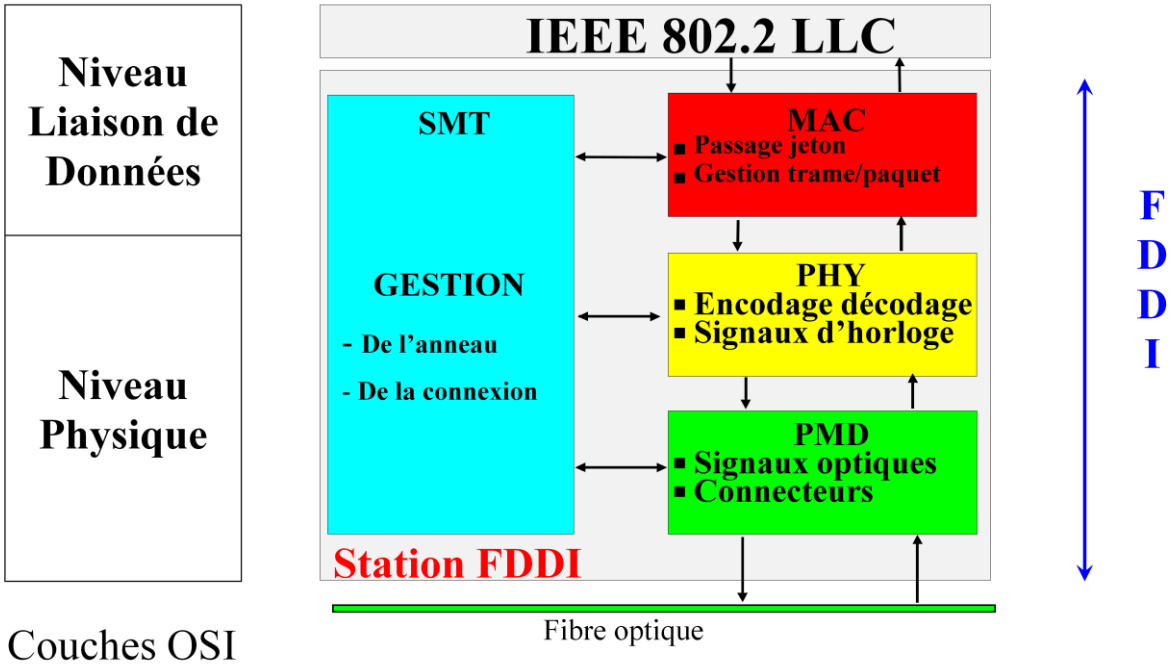


Figure 3.6 Modèle OSI.

Sous-couche PMD

- Physical Medium Dependent

Spécifie [9]:

- Le type de fibre (mono ou multimode)
- La longueur d'onde : 1300 nm
- La distance maximale entre stations
- Taux d'erreur : 10^{-12}
- Les organes d'émission[9].

Sous-couche PHY

- Physical Layer Protocol

Interface entre PMD et MAC

Définit :

- Le codage
- La synchronisation

- Codage : code 4B/5B
  - On code 4 bits sur 5 bits pour garantir des transitions
    - Au moins deux bits à "1"

- ❖ Pas plus de trois "0"

- 16 combinaisons → données
- les autres → état du réseau[9].

I	1	1	1	1	1	Idle (bourrage synchro)
H	0	0	0	1	0	Halt (Arrêt de l'activité)
Q	0	0	0	0	0	Quiet (Absence transitions)
J	1	1	0	0	0	Délimitation de la trame
K	1	0	0	0	1	Délimitation de la trame
L	0	0	1	0	1	Délimitation de la trame
T	0	1	1	0	1	Délimitation de la trame
R	0	0	1	1	1	« 0 » logique
S	1	1	0	0	1	« 1 » logique
V	0	0	0	0	1	Invalide
V	0	0	0	1	0	Invalide
V	0	0	0	1	1	Invalide
V	0	0	1	1	0	Invalide
V	0	1	0	0	0	Invalide
V	0	1	1	0	0	Invalide
V	1	0	0	0	0	Invalide

0	1	1	1	1	0	Data « 0000 »
1	0	1	0	0	1	Data « 0001 »
2	1	0	1	0	0	Data « 0010 »
3	1	0	1	0	1	Data « 0011 »
4	0	1	0	1	0	Data « 0100 »
5	0	1	0	1	1	Data « 0101 »
6	0	1	1	1	0	Data « 0110 »
7	0	1	1	1	1	Data « 0111 »
8	1	0	0	1	0	Data « 1000 »
9	1	0	0	1	1	Data « 1001 »
A	1	0	1	1	0	Data « 1010 »
B	1	0	1	1	1	Data « 1011 »
C	1	1	0	1	0	Data « 1100 »
D	1	1	0	1	1	Data « 1101 »
E	1	1	1	0	0	Data « 1110 »
F	1	1	1	0	1	Data « 1111 »

Table 3.4 Symboles de données et de contrôle code 4B/5B

- Ensuite, on code le 4B/5B avec le code NRZI

(transition lorsque le bit à transmettre est 1)

⇒ au moins 2 transitions par symbole

⇒ au plus 3 zéros consécutifs

⇒ Bonne récupération de l 'horloge

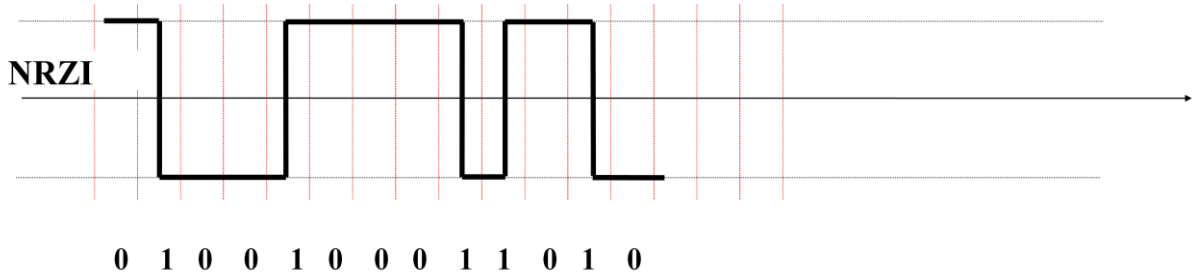


Figure 3.7 Code NRZI

- Synchronisation
  - Réseau plésiochrone (pas d'horloge unique)
    - ❖ Horloge 125 MHz ± 0.005 %
  - Récupération garantie par le codage
  - Utilisation de l'horloge interne pour transmission des données

- Horloge réception extraite des données entrantes
  - Dérive max entre 2 stations : 0.01 %

Buffer d'élasticité pour compenser la dérive[9].

Sous-couche MAC

- Medium Access Control

Définit :

- Le format des trames (4500 octets)
- Le format du jeton (ETR = Early Token Release)
- La méthode d 'accès à l'anneau[9].

Format des trames

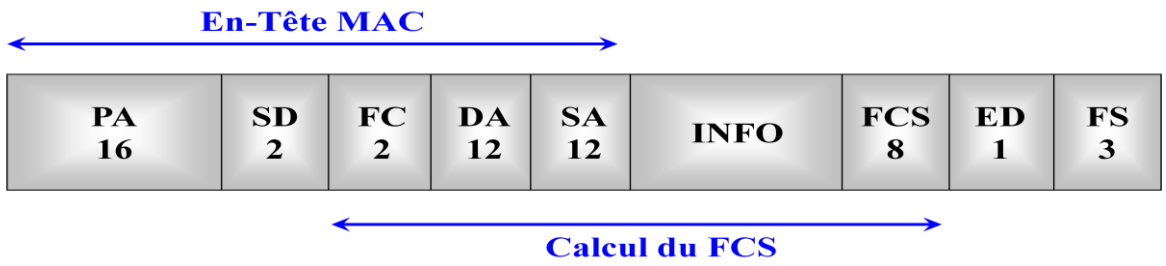


Figure 3.8 La trame FDDI

La trame FDDI comprend 4500 octets (dont 2 octets de PA).

Le fonctionnement est proche de la norme 802.5.

En raison des distances plusieurs trames peuvent circuler sur l'anneau[9].

PA = Préambule: au moins 16 symboles I (Idle).

Permet l'acquisition de la synchronisation bit

SD = *Starting Delimiter* : symboles 'J' et 'K'

soit 11000 & 10001

FC = *Frame Control* : décrit le type de trame et ses particularités.

- Synchrone ou asynchrone
- Longueur champ adresse
- Jeton

- Trame de type MAC
- trame de gestion SMT
- Trame LLC avec priorité éventuellement[9].

DA = *Destination Address* : indique le destinataire.

SA = *Source Address* : indique l'émetteur.

INFO = Informations : de 0 à 4478 octets d'information
Ce champ peut être vide ou contenir un nombre pair de symboles

FCS = *Frame Check Sequence* : séquence de contrôle.
Similaire à celui utilisé avec d'autre type de réseaux

ED = *Ending Delimiter* :

- un symbole 'T' dans le cas d'une trame
- deux symboles dans le cas d'un jeton

FS = *Frame Status* : indication des indicateurs reflétant la validité de la trame.
Comporte au moins trois symboles :

- E (erreur détectée)
- A (adresse reconnue)
- C (trame copiée)

Pour chacun de ces indicateurs, un "1" logique est représenté par un symbole S (Set) et un "0" logique par un symbole R (Reset) [9].

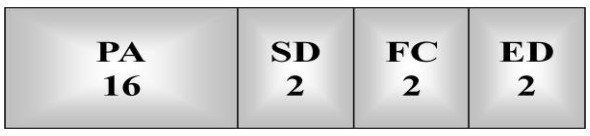


Figure 3.9 Format d'un jeton FDDI

- PA : Préambule au moins 16 symboles « I »
- SD : Starting Delimiter ; un « J » et un « K »
- FC : Frame Control - type de jeton
- ED : Ending Delimiter ; deux « T »

Sous-couche SMT

- Station Management
  - Contrôle de l'initialisation du système
  - Configuration du réseau
  - Insertion et retrait des stations
  - Traitement des erreurs
  - Collecte de statistiques[9].

Technique d'accès

Validation du temps de rotation cible du jeton

- Génération d'un jeton par le maître
- Circulation à vide pour validation du TTRT[9].

Remarque :

- Plus le TTRT est grand, plus le débit utile est important

Mais alors la QoS Temps réel se dégrade[9].

Allocation des crédits d'émission

- En fonction du TTRT :
  - Calcul par le maître du temps à allouer à chaque station
  - Prend en compte le Di demandé
  - Allocation par le maître de TSYN(i)
  - Crédit alloué à chaque station[9].

$$\sum_{i=1}^{N} TSYN(i) + T_{vide} = \beta \times T\,\mathrm{TR}T$$

3.8. Spécification formelle du protocole FDDI

Pour modéliser le protocole FDDI, on a choisi une configuration de 2 stations ST1 et ST2 et un anneau RING. Le modèle est composé des quatre canaux de communication pour la réception et l'envoi des messages entre l'anneau et les deux stations, il s'agit de tt1 et tt2 pour transmission des jetons et rt1 et rt2 pour la réception des jetons.

Le temps de transmission synchrone ou asynchrone SA ne doit pas dépassé 20 unités de temps. Le temps représentant le délai de maintenir le jeton est td et il égale 0 unité de temps.

Le temps de garder le jeton par une station est TRTT = 120 unités de temps. Voici le modèle du système en langage UPPAAL.

```
chan tt1, tt2, rt1, rt2;
const SA 20;
const td 0;
const TRTT 120;
system RING, ST1, ST2;
```

3.8.1 Le modèle de l'anneau

L'état initial de l'anneau est ring_to_1 pour donner le contrôle à la station 1 par l'envoi du message tt1. L'anneau restera dans cet état un temps qui ne dépasse pas td. Une fois la station 1 commencera la transmission, l'anneau restera dans son état ring_1 jusqu'à la réception du message rt1 ? Et il fait la même chose avec la station 2, puis la station 1 et ainsi de suite.

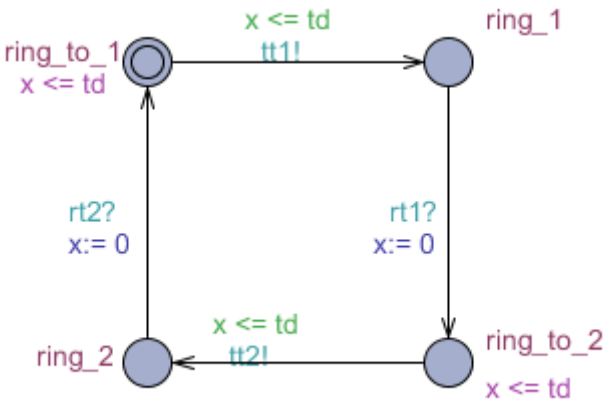


Figure 3.10 Le modèle graphique de l'anneau

Le modèle (le processus) de l'anneau selon le langage UPPAAL contient une horloge locale x, quatre états et quatre transitions.

```
process RING {
clock x;
state
  ring_to_1 {x <= td}, ring_1,
  ring_to_2 {x <= td}, ring_2;
init
  ring_to_1;
trans
  ring_to_1 -> ring_1 { guard x <= td; sync tt1!; },
  ring_1  -> ring_to_2 { sync rt1?; assign x:= 0; };
  ring_to_2 -> ring_2 { guard x <= td; sync tt2!; },
  ring_2  -> ring_to_1 { sync rt2?; assign x:= 0; };
}
```

3.8.2 Le modèle de la première station

Ce modèle contient trois horloges locales, x, y et z, six états et 8 transitions. Si la première station reçoit le message tt1 de modèle d'anneau, elle passe de l'état initial station_z_idle à l'état station_z_sync pour commencer la transmission synchrone pour une durée qui ne dépasse pas SA.

Si la station a terminé la transmission synchrone avant ce délai, elle peut effectuer une transmission asynchrone et ceci avant l'écoulement du temps TRTT. D'après ce modèle la station a un temps de 2*TRTT pour faire une transmission synchrone et une transmission asynchrone.

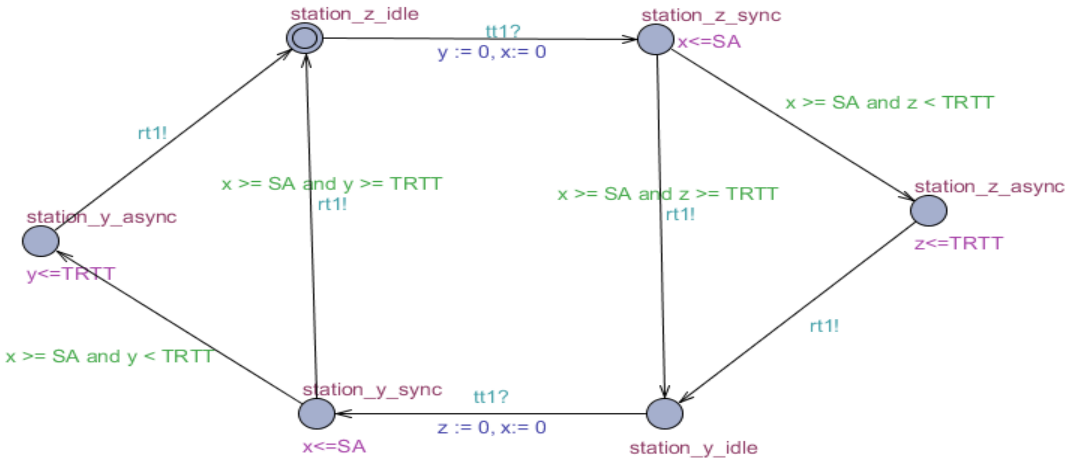


Figure 3.11 Modèle de la première station

Le modèle de la première station selon le langage UPPAAL :

```
process ST1 {
clock
  x, y, z;
state
  station_z_idle, station_z_sync{ x<=SA }, station_z_async{ z<=TRTT },
  station_y_idle, station_y_sync{ x<=SA }, station_y_async{ y<=TRTT };
init
  station_z_idle;
trans
  station_z_idle -> station_z_sync { sync tt1?; assign y := 0, x:= 0; },
  station_z_sync -> station_y_idle { guard x >= SA, z >= TRTT ; sync rt1!; },
  station_z_sync -> station_z_async { guard x >= SA, z < TRTT ; },
  station_z_async -> station_y_idle { sync rt1!; },
  station_y_idle -> station_y_sync { sync tt1?; assign z := 0, x:= 0; },
  station_y_sync -> station_z_idle { guard x >= SA, y >= TRTT ; sync rt1!; },
  station_y_sync -> station_y_async { guard x >= SA, y < TRTT ; },
  station_y_async -> station_z_idle { sync rt1!; };
}
```

3.8.3 Le modèle de la deuxième station

Le modèle de la deuxième station est similaire à celui de la première station, ici les canaux de communication avec le modèle de l'anneau sont tt2 et tr2.

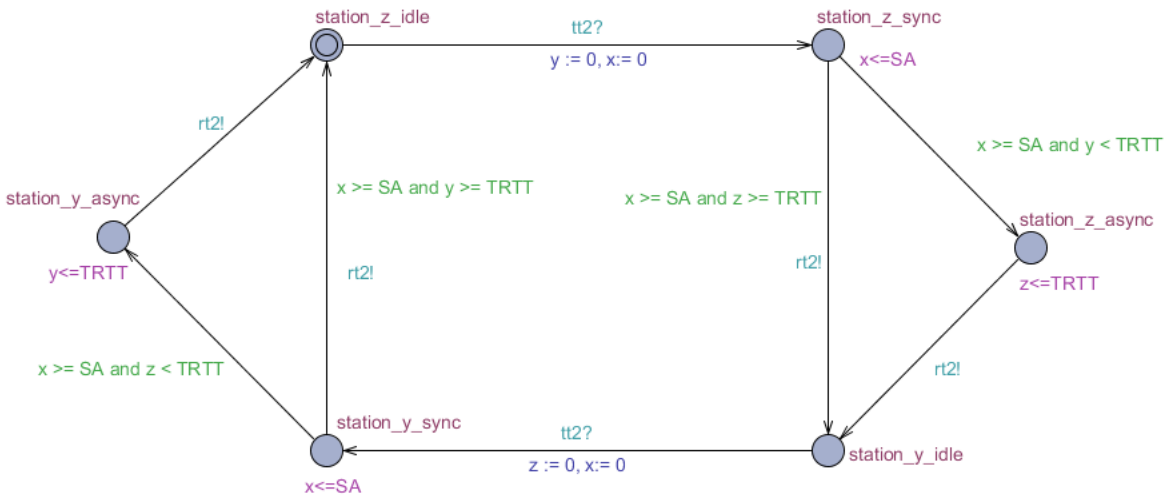


Figure 3.12: Modèle de la deuxième station

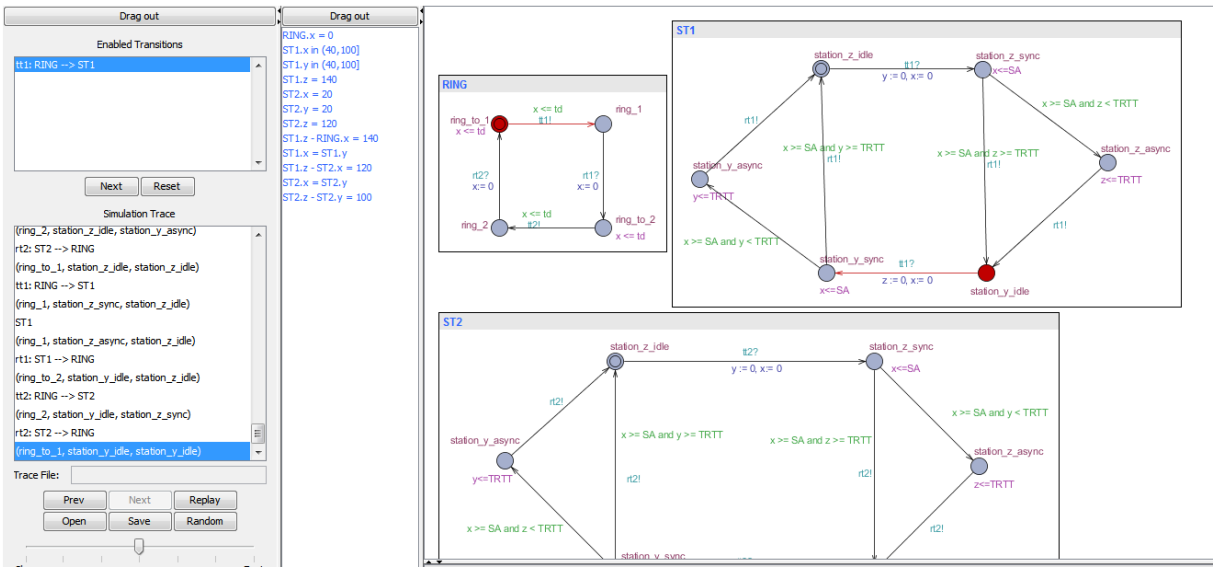
Le modèle en langage UPPAAL :

```
process ST2 {
clock
  x, y, z;
state
  station_z_idle, station_z_sync{ x<=SA }, station_z_async{ z<=TRTT },
```

```
  station_y_idle, station_y_sync{ x<=SA }, station_y_async{ y<=TRTT };
init
  station_z_idle;
trans
  station_z_idle -> station_z_sync { sync tt2?; assign y := 0, x:= 0; },
  station_z_sync -> station_y_idle { guard x >= SA, z >= TRTT ; sync rt2!; },
  station_z_sync -> station_z_async { guard x >= SA, z < TRTT ; },
  station_z_async -> station_y_idle { sync rt2!; },
  station_y_idle -> station_y_sync { sync tt2?; assign z := 0, x:= 0; },
  station_y_sync -> station_z_idle { guard x >= SA, y >= TRTT ; sync rt2!; },
  station_y_sync -> station_y_async { guard x >= SA, y < TRTT ; },
  station_y_async -> station_z_idle { sync rt2!; };
}
```

3.9 Simulation du modèle

L'outil UPPAAL permet de simuler un modèle d'un système temps réel pour vérifier son comportement. Notre modèle a été simulé en utilisant l'option de la simulation aléatoire et les résultats de simulation nous a confirmé le bon fonctionnement du système. Les deux figures suivantes nous montrent quelques résultats de simulation.



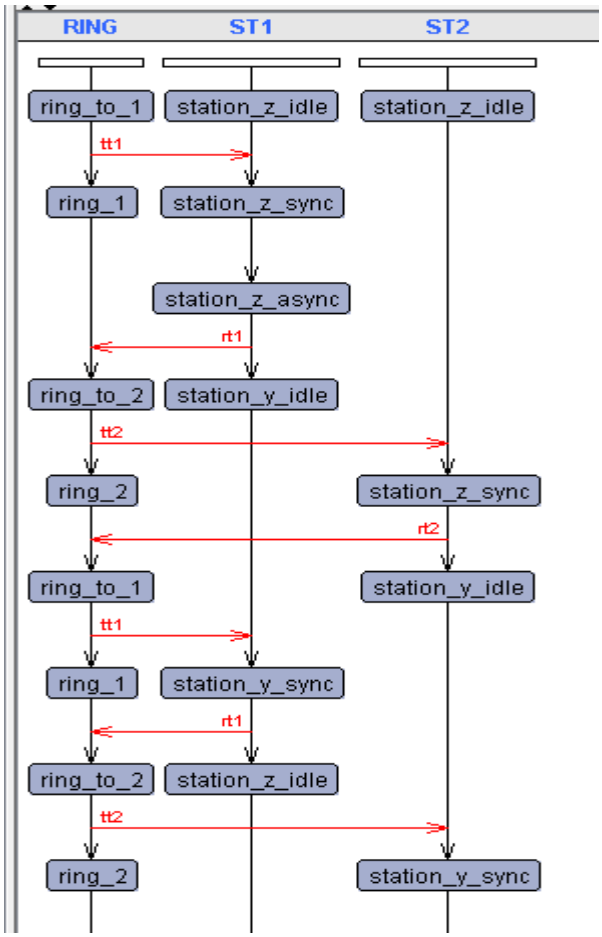


Figure 3.13 Résultats de simulation

3.10 Vérification formelle

Après la vérification du comportement du modèle par des simulations, nous avons réalisé la vérification (satisfaction) de certaines propriétés du système. L'outil UPPAAL intègre un outil très puissant pour le model-checking temps réel.

Comme exemple, nous avons vérifié la propriété que nous avons toujours une seule station en cours de transmission. L'expression formelle de cette propriété selon la logique temporelle est comme suit :

A[] not ((ST1.station_z_sync or ST1.station_z_async or ST1.station_y_sync or ST1.station_y_async) and (ST2.station_z_sync or ST2.station_z_async or ST2.station_y_sync or ST2.station_y_async))

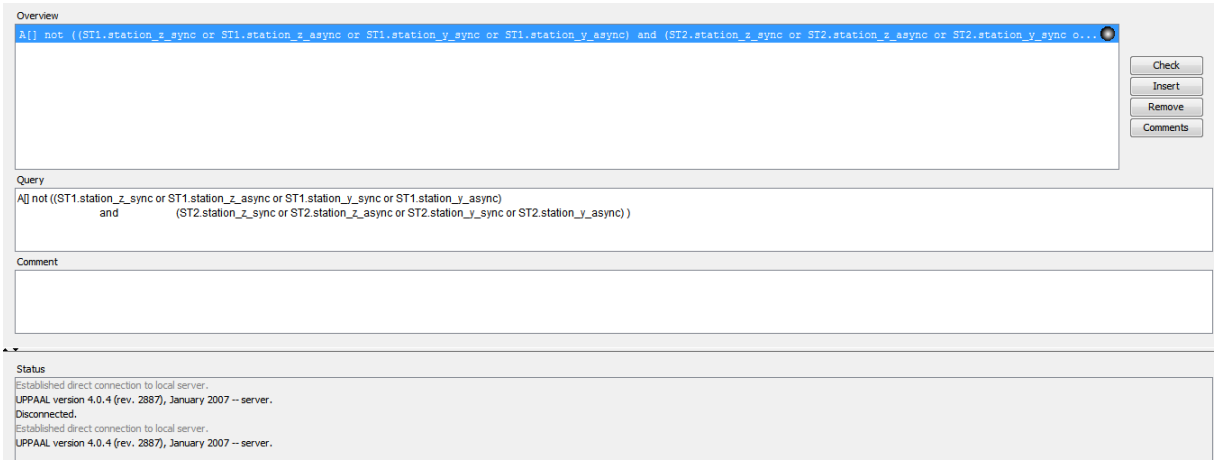


Figure 3.14 Vérification de la propriété.

Conclusion

Dans ce chapitre, nous avons montré comment modéliser un système temps réel avec l'outil UPPAAL, le simuler et puis vérifier des propriétés temporelles.

CONCLUSION GERERALE

Conclusion générale

Dans ce mémoire nous avons présenté une méthode pour la spécification et vérifications des systèmes temps réel en utilisent l'outil UPPAAL.

Il est connue que les systèmes temps réel sont des systèmes complexes a cause des contraintes temporelles ajouté comme des caractéristiques pour leur fonctionnement.

Nous avons montré en utilisant l'outil UPPAAL comment spécifier ces systèmes a l'aide du formalisme des automates temporisés et comment la vérifier par le simulation UPPAAL ainsi leur vérification formelle par la technique du model-checking intégrée dans l'outil UPPAAL.

Comme perspective, nous voulons aborder des systèmes plus complexes a fin de montrer l'efficacité de cette approche.

Bibliographie :

[AD90] R. Alur and D. L. Dill. Automata for modeling real-time systems. In Proc. 17th Int. Coll. Automata, Languages, and Programming (ICALP '90), Warwick University, England, July 1990, volume 443 of Lecture Notes in Computer Science, pages 322–335. Springer, 1990.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. Theoretical Computer Science, 126 :183–235, 1994. (Cité pages 33 et 35.)

[AN82] A. Arnold and M. Nivat. Comportements de processus. Colloque AFCET Les Mathématiques de l'Informatique, pages 35–68, 1982.

[ABRW93] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Data consistency in hard real-time systems. Technical report, 1993. (Cité page 26.)

[BC 99] Bernard Cousin , cours ,Le protocole FDDI ,Université Rennes I ,mars 1999

[Bry92] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. ACM Comput. Surv., 24(3) :293–318, 1992.

[BY04] J. Bengtsson and W. Yi. Timed automata : Semantics, algorithms and tools. pages 87–124. 2004. (Cité page 34.)

[CAR96] Francisco Vasques de Carvahlo ; Integration de Mecanismes d'Ordonnancement et de Communication dans la sous-Couche MAC de Reseaux Locaux Temps réel ; Thèse, Laboratoire d'Analyse et d'Architectures des Systèmes du CNRS, 1996 (N° ordre: 2340)

[CE82] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Logic of Programs, Workshop, pages 52–71, London, UK, 1982. Springer-Verlag.

[CGP00] E. Clarke, O. Grumberg, and D. Peled. Model checking. MIT Press, 2000.

[EH83] E.A. Emerson and J.Y. Halpern. "sometimes" and "not never" revisited : on branchingversus linear time (preliminary report). In POPL '83 : Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, pages 127–140, New York, NY, USA, 1983. ACM.

[FPY02] E. Fersman, P. Pettersson, and W. Yi. Timed automata with asynchronous processes : Schedulability and decidability. In TACAS '02 : Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pages 67–82. Springer-Verlag, 2002. (Cité pages 34 et 35.)

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming, volume 12. ACM, New York, NY, USA, 1969

[JSM91] K. Jeffay, D.Stanat, C. Martel ; On non preemptive Scheduling of periodic and Sporadic Tasks, in Proc. Of RTSS'91 – IEEE Real Time Systems Symposium, San Antonio, Texas, December 1991

[GAL 95] B. O. Gallmeister. POSIX 4 : Programming for the Real World . O'Reilly and Associates, January 1995.

[Koz82] D. Kozen. Results on the propositional μ-Calculus. In Proceedings of the 9th Colloquium on Automata, Languages and Programming, pages 348–359, London, UK, 1982.Springer-Verlag.

[KSSR96] H. Kaneko, J. A. Stankovic, S. Sen, and K. Ramamritham. Integrated scheduling of multimedia and hard real-time tasks. Technical Report UM-CS-1996-045, University of Massachusetts, Amherst, Computer Science, December, 1996.

[LAR03] Kim Guldstrand Larsen, Resource-Efficient Scheduling for Real Time Systems, Embedded Software, Third International Conference, {EMSOFT} 2003.

[LAR99] Klaus Havelund, Kim Guldstrand Larsen, Arne Skou: Formal Verification of a Power Controller Using the Real-Time Model Checker UPPAAL. ARTS 1999: 277-298

[LL73] C.Liu & J. Layland ; Scheduling Algorithms for multiprogramming in a hard Real time environment ; Journal of ACM, vol n°29, n°1, 1973, pp 46-61

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. International Journal on Software Tools for Technology Transfer, 1(1–2) :134–152, October 1997. (Cité page 33.)

[MB 17] Mustapha BOURAHLA ,Cours ,Caractéristiques des systèmes temps réel,cours master 2 GL,2016/2017

[MP92a] Z. Manna and A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems. Springer, 1992.

[Niv79] M. Nivat. Sur la synchronisation de processus. Revue technique Thomson-CSF, 11 :899–919, 1979.

[OF07] Fatiha.OUAZAR, Vérification distribuée des systèmes temps réel, Résumé de mémoire de Magister,2006-2007

[Pnu77] A. Pnueli. The temporal logic of programs. In SFCS '77 : Proceedings of the 18th Annual Symposium on Foundations of Computer Science, pages 46–57, Washington, DC, USA, 1977. IEEE Computer Society.

[PI 09] Isabelle PUAUT , Option STR Systèmes temps-réel. Master informatique – première année, 2009.

[QS82] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in cesar. In Proceedings of the 5th Colloquium on International Symposium on Programming, pages 337–351, London, UK, 1982. Springer-Verlag.

[Tur37] A. Turing. On computable numbers, with an application to the entscheidungsproblem. Proceedings of the London Mathematical Society, 42 :230–265, 1937.

[Tho90] W. Thomas. Automata on infinite objects. pages 133–191. MIT Press, 1990. (Cité page 32.)

[Tar55] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics, 5(2) :285–309, 1955.

[Sip96] M. Sipser. Introduction to the Theory of Computation : Preliminary Edition. PWS Publishing Co., Boston, MA, USA, 1996.

[Sta88] J.A. Stankovic. Misconceptions about real-time computing : a serious problem for next-generation systems. Computer, 21 :10–19, 1988. (Cité page 17.)

[SRL90] L. Sha, R.Rajkumar, J.Lehoczky ; Priority Inheritance Protocols; an approach to Real time Synchronization ; IEE Tr. on computers, 39(9), September 1990

[XSS+ 96] M. Xiong, R. Sivasankaran, J. A. Stankovic, K. Ramamritham, and D. Towsley. Scheduling transactions with temporal constraints : exploiting data semantics. In RTSS ’96 : Proceedings of the 17th IEEE Real-Time Systems Symposium, pages 240–253. IEEE Computer Society, 1996. (Cité page 26.)

Web graphies :

[1] http://beru.univ-brest.fr/~singhoff/ENS/UE_Appli_Info/CM/intro.pdf vu le 25/03/2017

[2] http://beru.univ-brest.fr/~singhoff/cheddar/publications/martin04.pdf vu le 3/03/2017

[3] http://ekladata.com/1QFnlCft_PVMIGe6qLG5MRqKeWY.pdf Vu le 05/03/2017

[4] http://david.decotigny.free.fr/rt/intro-ordo/intro-ordo004.html vu le 17/03/2017

[5] https://wapiti.telecomlille.fr/commun/ens/peda/options/st/rio/pub/exposes/exposesrio1997/FDDI/FDDI.htm Vu le 16/04/2017

[6] http://www.memoireonline.com/09/13/7361/m_Etude-dimplementation-dune-solution- VOIP-securisee-dans-un-reseau-informatique-dentrepr18.html Vu le 16/04/2017

[7] https://wapiti.telecom-lille.fr/commun/ens/peda/options/st/rio/_pub/exposes/_exposesrio1997/FDDI/FDDI.htm Vu le 17/04/2017

[9] http://yaju.free.fr/lic_pro/Cours_Perrenot/Hautdebit/1_FDDI_DQDB/For_FDDI.ppt vu le : 17/04/2017


الحمد لله

Résumé :

En informatique, on parle d'un système temps réel lorsque ce système est capable de contrôler (ou piloter) un procédé physique à une vitesse adaptée à l'évolution du procédé contrôlé.

Dans notre travail on a spécifié le protocole FDDI (Fiber Distributed Data Interface) a fin d'avoir un modèle pour le simuler et le vérifier formellement avec l'outil UPPAAL, Ainsi avec ce travail nous pouvons montrer commet modéliser, simuler et vérifier formellement des systèmes temps réel.

Mots clés : #système temps réel #spécification et vérification formelles #FDDI #UPPAAL

Abstract :

Formal Specification and verification are two methods to design systems as real time systems. The first is used to model systems , where its result will be used to apply verification techniques such as model checking.

In this work , we developed a method for specifying and verifying real-time systems using the tool UPPAAL.

key words : # Formal Specification ,# real-time Systems #UPPAAL #FDDI

ملخص :

في الإعلام الآلي نحن نتحدث عن النظام في الوقت الحقيقي حيث هذا النظام قادر على التحكم و ادارة عملية فيزيائية بسرعة مناسبة لعملية التي تسيطر عليها

في عملنا هذا نتخصص في البروتوكول FDDI ونقوم بالمحاكاة و التحقق رسميا باستعمال النموذج UPPAALو في هذا العمل نظهر كيف يمكن تصميم نماذج و مراقبة أنظمة الزمن الحقيقي.

الكلمات المفتاحية : #المحاكاة #الزمن-الحقيقي #UPPAAL #FDDI