TABLE DES MATIÈRES

Chapitre 3 : Les Principaux Protocoles MAC Adaptés aux Communications Temps Réel

Chapitre 4 : Environnements Temps Réel Répartis

Chapitre 5 : Une méthodologie de Validation Basée sur une Analyse d'Ordonnançabilité

Chapitre 6 : Analyse des Performances

Introduction Générale

L'évolution technologique de ces dernières années, en particulier, dans les réseaux locaux de communication, a permis le développement à grande échelle des systèmes distribués temps réel dans de nombreux domaines du secteur industriel (aéronautique, militaire, transport, industries des procédés, télécommunications ...). Ces applications très variées par leurs caractéristiques (taille, environnement, contraintes, etc.) vont des centrales nucléaires aux simples distributeurs de billets ou caisses enregistreuses. Mais ces applications possèdent un point commun : elles sont soumises à des contraintes temporelles.
L'augmentation de la complexité des applications impose de faire appel à des méthodes de conception de plus en plus élaborées. En même temps que des outils de développement, il est nécessaire de développer des outils qui servent à modéliser et à valider ces applications.

Partant du constat qu'il n'existe presque pas de méthodes d'analyse pour vérifier qu'un système atteint ses spécifications temps réel et afin de mesurer les paramètres de performance, il nous a paru important de réfléchir à ce problème et d'apporter une réponse même partielle. Dans ce travail, nous définissons les applications temps réel réparties comme des applications composées de tâches réparties sur différents sites, qui communiquent uniquement par échange de messages via un réseau local. Et nous proposons une méthodologie de modélisation, de simulation et de validation de ce type d'applications c'est-à-dire de vérification du respect des contraintes temporelles en fonction d'une part de l'algorithme d'ordonnancement utilisé (à priorité fixe ou dynamique) localement au niveau des sites et d'autre part du protocole de communication temps réel utilisé sur le réseau.

Notre méthodologie étant basée sur une analyse d'ordonnançabilité, nous avons jugé important de rappeler les études faites sur l'ordonnancement des tâches et des messages.

Ce mémoire présente six chapitres, les quatre premiers rappellent les bases essentielles à ce travail et synthétisent les travaux existants en matière d'algorithmes d'ordonnancement et de protocoles adaptés aux communications temps réel ; les deux derniers décrivent le travail proprement dit:

- le premier chapitre rappelle les concepts de base et les définitions essentielles des systèmes temps réel et des réseaux locaux temps réel,

- le deuxième chapitre a pour objectif de présenter les principales techniques d'ordonnancement monoprocesseur de tâches temps réel ainsi que les protocoles d'allocation de ressources intégrés à ces algorithmes pour minimiser le temps d'attente des tâches sur l'acquisition de ressources critiques,

- le troisième chapitre décrit les principaux réseaux locaux temps réel utilisés comme support de communication pour les applications temps réel considérées. Il met l'accent sur les protocoles de communication de la couche MAC car ils correspondent aux stratégies d'ordonnancement des messages,

- le quatrième chapitre présente, d'une part, quelques exemples de systèmes d'exploitation répartis réalisés dans le cadre de projets de recherche en mettant en évidence les concepts de chacun notamment l'aspect ordonnancement et, d'autre part, un outil et une méthode de validation d'applications temps réel réparties,

- le cinquième chapitre présente notre méthodologie qui réalise la validation temporelle des applications temps réel réparties en procédant à l'ordonnancement des tâches et des messages et en vérifiant le respect des contraintes temporelles. La méthodologie est générale dans le sens où elle s'applique à n'importe quel algorithme d'ordonnancement et à n'importe quel protocole de communication pourvu que ce dernier offre des délais de transmission bornés. Deux exemples d'application viennent illustrer le principe de la méthodologie. Le premier exemple repose sur le modèle producteur / consommateur, le second (exemple de l'ascenseur) est d'ordre pratique,

- le sixième chapitre présente l'outil développé MOSARTS et une étude de performance en définissant tous les paramètres mesurés ainsi que les résultats de simulation qui ont été obtenus pour quelques exemples.

En conclusion, nous montrons les apports de ce travail et présentons les axes de développement futurs.

Trois annexes complètent le contenu de ce mémoire. La première annexe présente brièvement le langage Tcl/Tk qui a servi à développer l'interface graphique de l'outil MOSARTS. La deuxième annexe présente MOSARTS à travers son interface (menus de saisie et menus résultats de simulation) et la dernière annexe vient compléter le chapitre 3 en décrivant les formats des trames des différents réseaux étudiés.

CHAPITRE 1

INTRODUCTION AUX SYSTEMES INFORMATIQUES TEMPS REEL

1. Introduction

Nous proposons, dans ce chapitre, une introduction aux systèmes informatiques temps réel ainsi qu'une présentation de la terminologie associée.

Ce chapitre comprend trois paragraphes:

- Le premier présente des généralités sur les systèmes temps réel,
- Le deuxième décrit brièvement les réseaux de communication temps réel,
- Le troisième précise notre cadre de travail par rapport au contexte général présenté.

2. Les systèmes temps réel

2.1. Présentation générale

Les systèmes temps réel sont, en général, constitués de deux parties: le système contrôlé, processus physique qui évolue avec le temps, appelé aussi *procédé* ou *partie opérative*, et le système contrôlant appelé aussi *partie commande* qui est un système informatique qui interagit avec le premier. Le système contrôlant prend régulièrement connaissance de l'état du procédé par le biais de capteurs, calcule la prochaine action à réaliser sur la base des dernières mesures puis applique une consigne au processus commandé par le biais d'actionneurs (voir figure 1.1).

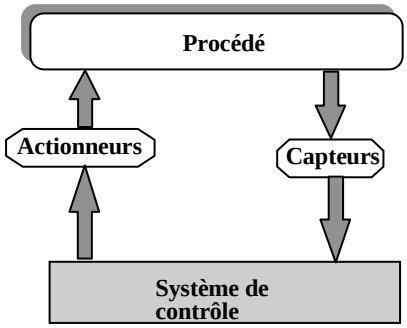


Figure 1.1: *Système temps réel*

Un système est qualifié de temps réel si sa correction ou sa validité ne dépend pas uniquement de la justesse des résultats obtenus mais aussi des instants de production de ces résultats [ZSR 90], car les différentes activités d'un tel système doivent nécessairement s'exécuter dans des laps de temps limités appelé *échéances*.
Le respect des contraintes temporelles est un aspect fondamental et spécifique aux systèmes temps réel, constituant la caractéristique principale qui les distingue des systèmes informatiques classiques (non temps réel).
D'une manière générale, nous distinguons trois types de systèmes temps réel:

- *les systèmes temps réel à contraintes strictes* qui sont des systèmes pour lesquels le non respect d'une échéance ne peut être toléré (exemple: la commande du moteur d'un avion) car cela peut engendrer une catastrophe,

- *les systèmes temps réel à contraintes relatives*, par opposition, tolèrent les dépassements des échéances,

- *les systèmes temps réel à contraintes mixtes* qui comprennent des programmes à contraintes strictes et d'autres à contraintes relatives.

Selon l'équipement contrôlé, les contraintes de temps peuvent être de divers ordres, de l'ordre:

- de la milliseconde: système radar,...

- de la seconde: système de visualisation,...

- de quelques minutes : chaîne de fabrication,...

- de l'heure: réaction chimique,...

2.2. Structure d'un système de contrôle

Le système de contrôle consiste en une structure matérielle munie d'un ensemble de logiciels permettant d'agir sur le système contrôlé. La structure matérielle correspond à l'ensemble des ressources physiques (processeurs, cartes d'entrée/sortie, ...) nécessaires au bon fonctionnement du système. L'architecture matérielle peut être:

- *monoprocesseur*: tous les programmes s'exécutent sur un seul processeur en parallélisme apparent,

- *multiprocesseur*: les programmes sont répartis sur plusieurs processeurs partageant une mémoire commune,

- *réparti*: les programmes sont répartis sur plusieurs processeurs dépourvus de mémoire commune et d'horloge commune. Ils sont reliés par un médium de communication par lequel ils peuvent communiquer par échange de messages.

Les systèmes temps réel répartis basés sur les réseaux locaux font partie de cette dernière catégorie. Ils sont très répandus principalement pour trois raisons: le débit élevé du réseau qui permet des commutations rapides et donc la prise en compte de beaucoup d'événements, l'existence de plusieurs processeurs qui leur confère une bonne résistance aux pannes et enfin la répartition topologique du procédé qui permet de répondre aux besoins des applications temps réel. Les programmes de l'application étant répartis sur différents processeurs (appelés aussi *sites*), la communication entre ces programmes constitue une activité importante dans les systèmes temps réel répartis.

La partie logicielle du système de contrôle comprend :

- d'une part, un système d'exploitation appelé *exécutif temps réel* chargé de fournir un ensemble de services (voir figure 1.2) que les programmes de l'application peuvent utiliser durant leur exécution. Il doit en particulier :
  - adopter une stratégie pour partager le temps processeur (notamment dans un système multitâches) entre les différentes activités en attente d'exécution en favorisant celles qui ont les délais critiques les plus proches,
  - gérer l'accès aux ressources partagées,
  - et offrir des mécanismes de synchronisation et de communication entre les activités du système de contrôle.
- d'autre part, afin de garantir un fonctionnement correct du procédé, des programmes sont implantés par l'utilisateur et traduisent les fonctions que doit réaliser le système de contrôle pour la prise en compte de l'état du procédé et sa commande. Ces programmes se présentent sous la forme d'un ensemble d'unités d'exécution appelées *tâches*.

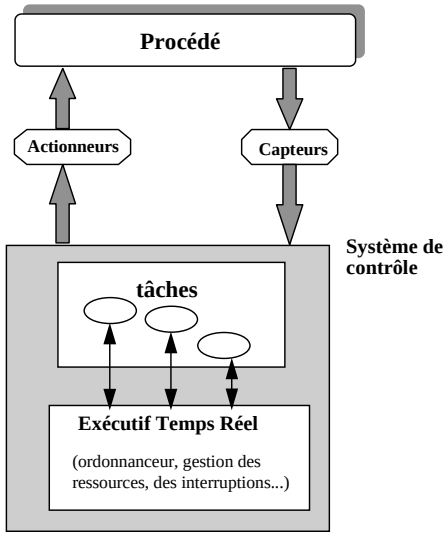


Figure 1.2: *Structure d'un noyau temps réel*

2.3. Les tâches

Les tâches peuvent être indépendantes comme elles peuvent coopérer par des mécanismes de synchronisation (postage/attente d'un événement) ou de communication de données (par boîte aux lettres ou par rendez-vous). Dans un cas comme dans l'autre, ces relations se traduisent, d'un point de vue comportemental, par des contraintes de précédence définissant ainsi un ordre dans l'exécution des tâches impliquées. De plus, plusieurs tâches peuvent partager une ou plusieurs ressources dont l'accès est exclusif. Ce qui veut dire que si plusieurs tâches demandent la même ressource critique, l'exécutif temps réel autorisera l'exécution d'une seule tâche afin d'assurer l'exclusion mutuelle. Par conséquent, les autres tâches seront bloquées en attente de cette ressource.

En plus de ces contraintes, les tâches doivent respecter des délais critiques imposés par les dynamiques de l'environnement à contrôler. D'un point de vue temporel, les paramètres suivants peuvent caractériser une tâche :

- *la date de réveil* : qui correspond à l'instant à partir duquel la tâche peut commencer son exécution ,
- *la durée d'exécution* : égale à la borne supérieure du temps processeur nécessaire à l'exécution de la tâche,
- *le délai critique* : qui est l'intervalle de temps, à partir de la date de réveil, durant lequel la tâche doit s'exécuter. En dehors de cet intervalle, son exécution devient inutile.
- *la période* : est l'intervalle de temps fixe qui sépare les arrivées successives d'une tâche. Ce paramètre concerne les tâches activées par un signal périodique provenant d'une horloge temps réel interne, c'est le cas par exemple de l'acquisition de données. Ce type de tâches est appelé *tâches périodiques*.

Les tâches dont la date de réveil n'est pas connue avant la mise en exploitation du système sont dites *tâches apériodiques*. En général, les tâches périodiques assurent le déroulement normal des fonctions de commande du procédé alors que les tâches apériodiques traitent les alarmes et les états d'exception.

2.4. La problématique de l'ordonnancement

Plusieurs tâches peuvent concourir pour l'obtention du processeur. L'attribution du processeur aux tâches c'est-à-dire leur exécution doit être planifiée pour garantir le respect de leurs contraintes temporelles. Cette fonction est gérée par un élément de l'exécutif temps réel appelé *ordonnanceur*. L'ordonnanceur doit régler les conflits des tâches concurrentes pour l'accès au processeur en utilisant une politique d'allocation qui favorise l'accès aux tâches les plus urgentes. Cette politique est dictée par un algorithme d'ordonnancement qui construit une séquence de l'ensemble des tâches à exécuter en veillant à respecter leurs contraintes temporelles et de synchronisation. Les études menées sur ce sujet [LL73, LW82] reposent sur la modélisation des tâches et la définition d'algorithmes d'ordonnancement.

2.5. Les algorithmes d'ordonnancement

Plusieurs critères peuvent définir un algorithme d'ordonnancement. Certains sont exposés ici mais la totalité des critères est présentée dans une synthèse faite par [CM 94] en vue de classer les algorithmes d'ordonnancement dans un contexte temps réel et réparti.

2.5.1.Préemptif/non Préemptif

Un algorithme d'ordonnancement est :

- *non préemptif*: si toute tâche qui commence son exécution doit être achevée avant qu'une autre tâche obtienne le processeur,

- *préemptif*: si une tâche peut être interrompue au profit d'une autre et être reprise ultérieurement.

Les algorithmes non préemptifs sont faciles à mettre en œuvre ; de plus, ils ont un faible coût puisque les changements de contexte sont minimisés et il n'y a pas lieu de gérer l'accès concurrent aux ressources critiques. Cependant les algorithmes préemptifs sont plus efficaces puisqu'ils offrent une exécution parallèle des tâches en ayant plus de liberté pour choisir une solution d'ordonnancement.

2.5.2.Hors ligne/en ligne

Un algorithme d'ordonnancement est dit :

- *hors ligne* : lorsque toutes les tâches ainsi que leurs caractéristiques temporelles sont connues avant le démarrage de l'application. La construction d'une séquence d'exécution des tâches respectant les contraintes temporelles est possible et l'ordonnanceur se réduit à une entité appelé *répartiteur* qui ne fait que lancer les tâches dans l'ordre dicté par la séquence. Il est évident qu'une telle approche ne peut prendre en compte des tâches dont la date d'activation n'est pas connue au départ.

- *en ligne* : lorsque la séquence d'exécution est construite au fur et à mesure uniquement avec les tâches prêtes à l'instant courant ; ce qui permet la prise en compte et l'exécution des tâches apériodiques.

2.5.3.Conduits par la priorité

L'ordonnanceur tel qu'il a été défini précédemment est l'une des fonctions principales de l'exécutif temps réel. Il est chargé d'attribuer le processeur à chaque tâche avant son échéance en utilisant un algorithme d'ordonnancement. On dit qu'un algorithme d'ordonnancement est conduit par la priorité s'il exécute à chaque instant la tâche la plus prioritaire. La priorité est ou bien affectée par le concepteur de l'application en fonction de la criticité des tâches ou encore assignée de manière automatique en fonction d'un paramètre temporel caractérisant les tâches telles que la période ou l'échéance qui reflètent l'urgence et non l'importance de la tâche au sens applicatif.

Une classification de ces algorithmes est basée sur le type de la priorité instaurée entre les tâches, qui peut être *fixe* ou *dynamique*. Un algorithme à priorité fixe affecte, à chaque tâche, une priorité qui reste constante durant l'exécution de cette dernière. Un algorithme à priorité dynamique recalcule la priorité des tâches durant leur exécution.

3. Les réseaux de communication temps réel

On qualifie de communication temps réel toute communication soumise à des contraintes de temps. Les réseaux qui supportent ce type de communication sont dits *réseaux temps réel*. En comparaison avec les réseaux classiques, leur étendue géographique est limitée, leur architecture de communication est réduite et les fonctions utilisées pour respecter les contraintes de temps sont à la charge des différents sites connectés au réseau. Ils sont dits à accès multiple (FIP, CAN, CSMA/DCR, etc.) car les sites accèdent en concurrence au médium de communication et c'est la technique implantée sur chacun d'eux souvent appelée *protocole de communication* qui règle les conflit d'accès. Beaucoup de travaux sont consacrés à l'adaptation de ces protocoles en vue de prendre en compte les contraintes de temps des messages [MZ 95, Mam 97].

3.1. Architecture

L'architecture la plus générale d'un réseau de communication est une architecture à sept couches basée sur le modèle ISO [ISO 84]. Néanmoins, dans le cadre des réseaux locaux temps réel, une architecture de communication limitée à trois couches suffit (voir figure 1.3): la couche physique, la couche liaison de données elle même décomposée en deux sous-couches LLC et MAC et la couche application. Seules les deux premières couches sont normalisées.
La couche physique gère les connexions physiques pour la transmission de bits à travers le médium de communication. La sous-couche LLC fournit les fonctions de liaison de données, de gestion des erreurs et de contrôle de flux. La sous-couche MAC gère l'accès au médium à l'aide d'un protocole de communication. La couche application fournit les services nécessaires à la gestion des tâches et du contrôle.
Dans un réseau local temps réel, un taux de perte de messages est toujours toléré ; de plus un message qui n'est pas transmis avec succès au bout d'un temps fixe est considéré comme étant perdu [MZ 95]. Le but, dans un système de communication temps réel, n'est pas de minimiser le délai moyen de transmission des messages comme dans les réseaux classiques mais de maximiser le pourcentage de messages transmis dans les délais.

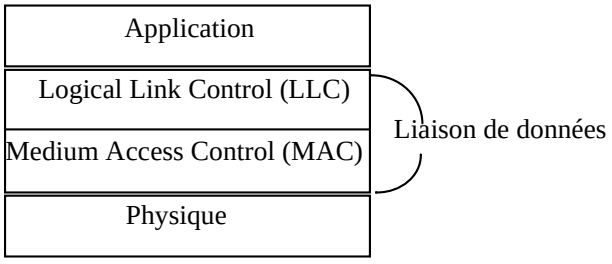


Figure 1.3: *Architecture d'un réseau de communication temps réel*

3.2. Les messages temps réel

Les messages venant des applications et soumis au service MAC peuvent être des messages temps réel (leur transfert est soumis à des contraintes temporelles strictes) ou non temps réel (pas de contraintes temporelles associées au transfert).

Nous pouvons distinguer deux types de messages:

- *les messages synchrones* (ou périodiques) sont caractérisés par un intervalle d'arrivée constant appelé période. Ils sont souvent utilisés pour l'acquisition de données délivrées par un capteur ou bien pour la communication entre tâches périodiques,

- *les messages asynchrones* (ou apériodiques) sont souvent utilisés pour véhiculer une information de type alarme ou une communication entre tâches apériodiques. Contrairement aux messages synchrones, les messages asynchrones arrivent à des instants non fixés.

Les tâches périodiques d'un site émettent le plus souvent des messages synchrones destinés à des tâches distantes périodiques. De plus, elles peuvent émettre des messages asynchrones. Par contre les messages émis par des tâches apériodiques sont toujours asynchrones.

3.3. Ordonnancement des messages temps réel

Les techniques du niveau MAC ont pour objectif de résoudre les conflits d'accès au médium entre les sites. Les stratégies existantes ne sont pas toujours basées sur le degré d'urgence des messages à transmettre mais restent utilisées dans les systèmes temps réel parce qu'elles assurent des délais de transfert bornés. L'adaptation de la couche MAC en vue de prendre en compte les contraintes de temps reste encore au stade de la recherche [Mam 97].

L'accès au médium peut être réalisé de différentes manières, chacune correspondant à une classe de protocoles de communication:

- *par multiplexage temporel* : chaque site dispose d'une tranche de temps dans un cycle répétitif (exemple, TDMA),

- *par compétition* : chaque site émet quand il le désire et doit coopérer avec les autres sites en vue de résoudre les situations de conflit (collision), exemple CSMA/DCR,

- *par consultation* : un site particulier autorise le site qui doit émettre (cas de FIP) ou bien un message particulier appelé *jeton* parcourt les sites en autorisant à émettre celui qui le détient à un moment donné.

4. Cadre du travail

Dans cette étude, nous considérons l'architecture d'un système réparti composé de plusieurs sites connectés à un réseau local doté d'un protocole de communication qui garantit un délai d'accès borné. Le système étant dépourvu de mémoire commune, le seul moyen de communiquer est l'échange de messages. Nous nous intéresserons aux applications implantées sur ce type de systèmes c'est-à-dire des applications composées de tâches réparties sur différents sites communiquant par échange de messages, que nous appellerons *applications temps réel réparties*. Ces applications sont bâties autour d'un réseau temps réel à délai d'accès borné. Le réseau est supposé fiable c'est-à-dire que tout message émis par une tâche d'un site est reçu, sans erreurs, par la tâche d'un autre site au bout d'un temps fini. De plus, chaque site dispose d'un ordonnanceur local classique pour les tâches comme ceux connus dans l'environnement monoprocesseur et d'un protocole de communication pour la gestion d'accès au réseau. La définition des caractéristiques temporelles des tâches et des messages suppose l'existence d'une horloge globale, dans le système, qui permet d'avoir un référentiel temporel global. Autrement dit, lorsque nous traitons des dates sur des sites différents (par exemple, dates de réveil de tâches distantes), celles-ci sont exprimées par rapport à une horloge globale. Nous supposons qu'il n'y a pas de dérive d'horloges locales et que l'horloge globale reste synchronisée avec le temps réel. Les mécanismes permettant de régler le problème de synchronisation des horloges ne seront pas considérés ici ; aussi le lecteur intéressé pourra consulter [Mam 97, FM 96, HMT 94, Lam 78, Ray 92].

La conception de telles applications passe inévitablement par la modélisation et la validation afin de vérifier qu'elles sont conformes aux exigences spécifiées dans le cahier des charges. En particulier, le respect des contraintes temporelles globales d'une application temps réel répartie implique le respect, à la fois, des contraintes temporelles des tâches et de celles des messages échangés entre ces tâches.
Notre objectif étant de valider temporellement une application temps réel composée de tâches périodiques à contraintes strictes qui échangent des messages périodiques (ou synchrones) à contraintes strictes, nous avons développé une méthodologie de validation basée sur l'ordonnancement des tâches et des messages.

Nous supposons connus :

- l'architecture de l'application c'est-à-dire l'ensemble des tâches qui la compose, leur placement sur les différents sites, leurs caractéristiques temporelles (date de réveil, durée d'exécution, délai critique et période), les ressources critiques locales à chaque site, ainsi que les messages échangés spécifiés à l'aide de leurs tailles en nombre d'octets,
- les caractéristiques physiques du réseau considéré : débit, taille du paquet, etc.,
- le protocole de communication,
- et l'algorithme d'ordonnancement des tâches combiné avec un protocole d'allocation de ressources.

La méthodologie comporte trois étapes (voir figure 1.4):

- *la modélisation de l'application* : consiste à caractériser les tâches et les messages à l'aide d'attributs temporels fournis en entrée pour quelques uns et déduits pour d'autres notamment en ce qui concerne les messages.

- *la prise en compte de la précédence* : la synchronisation (postage/attente d'un événement) et la communication (par boîte aux lettres ou par rendez-vous) entre tâches d'un même site sont modélisées à l'aide de relations de précédence locale entre ces tâches. L'échange de messages entre tâches de sites différents est modélisé à l'aide de relations de précédence qui impliquent la tâche qui émet, le message échangé et la tâche qui reçoit le message. L'ensemble de ces relations est représentée à l'aide d'un graphe (dit *de précédence global*) orienté où les arcs correspondent aux relations de précédence et les sommets aux entités: tâches ou messages. Cette étape consiste à modifier les paramètres temporels de manière à transformer la configuration de tâches et de messages avec relations de précédence en une configuration de tâches et de messages indépendants.

- *l'ordonnancement* : les tâches de la configuration obtenue à l'étape 2 sont ordonnancées, sur leurs sites de localisation, selon l'algorithme d'ordonnancement local et les messages sont ordonnancés selon le protocole de communication du réseau. Les attributs temporels étant calculés dans la situation la plus pessimiste, il s'agira de vérifier si les tâches et les messages respectent bien leurs échéances respectivement sur les sites et sur le réseau.

Une analyse des performances, faite dans les conditions les plus défavorables, nous donne un certain nombre de résultats concernant les temps de réponse des tâches et des messages, les taux d'utilisation des processeurs et du médium, etc. Ces résultats nous permettent de retenir l'algorithme d'ordonnancement local et le réseau qui donnent les meilleures performances pour une application donnée.

Vu que la méthodologie proposée est basée sur la notion d'ordonnancement des tâches et sur les techniques de communication pour le transfert des messages, nous parlerons dans la suite des techniques d'ordonnancement monoprocesseurs (chapitre 2) ensuite des protocoles de communication de la couche MAC (chapitre 3) avant de détailler le principe de cette méthodologie au chapitre 5 et de conclure, au chapitre 6, par une analyse des performances faite à partir de l'outil MOSARTS qui implante l'analyse proposée.

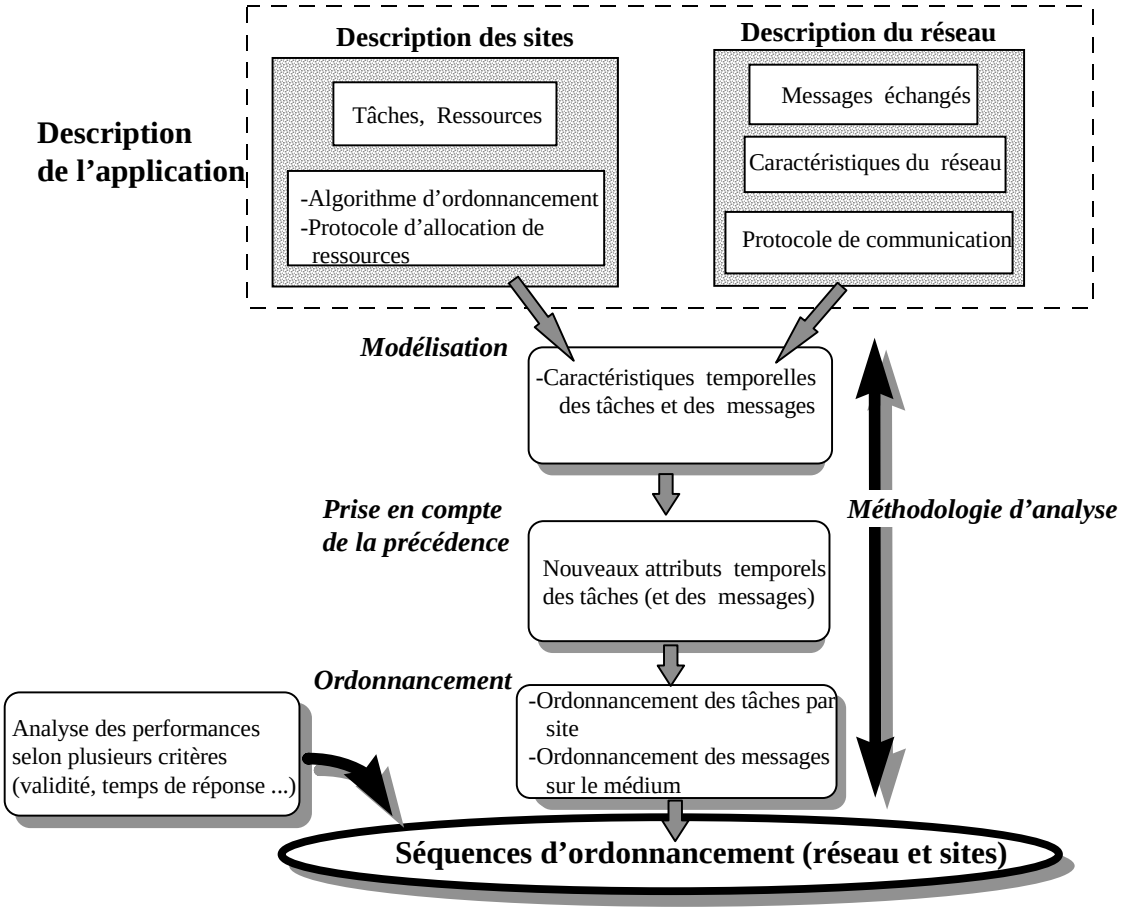


Figure 1.4 : *Différentes étapes de la méthodologie proposée*

CHAPITRE 2

ORDONNANCEMENT DES TACHES

1. Introduction

L'objectif de ce chapitre est de présenter les principaux algorithmes d'ordonnancement des tâches dans un contexte monoprocesseur. Les tâches peuvent interagir pour communiquer des données ou prendre en compte des événements. Dans un cas comme dans l'autre, leur exécution parallèle doit être soumise à des règles de synchronisation. Celles-ci pourront alors se matérialiser par un ordre dans l'exécution des tâches impliquées, que nous appellerons *précédence* entre tâches.

La compétition pour l'accès à une ressource critique est un problème classique et est connu sous le nom d'exclusion mutuelle, pour lequel il existe une variété de solutions plus ou moins complexes. Dans le domaine du temps réel, il s'agit non seulement de garantir l'accès exclusif aux ressources critiques (qui est le rôle de l'exécutif temps réel) mais aussi et surtout de minimiser le temps d'attente pour l'accès à ces ressources. Il existe des protocoles dits *d'allocation de ressources* intégrés aux algorithmes d'ordonnancement et capables de réduire le temps d'attente des tâches bloquées pour l'obtention de ressources afin que celles-ci ne dépassent pas leurs échéances.

Sachant que les algorithmes d'ordonnancement s'appliquent aux configurations de tâches périodiques, nous pourrons appréhender les tâches apériodiques avec ces algorithmes si nous les transformons, par exemple, en tâches périodiques particulières appelées *serveurs*. Les tâches que nous considérons, qu'elles soient périodiques ou apériodiques, sont toutes à contraintes strictes c'est-à-dire qu'elles doivent absolument s'exécuter avant leur échéance sous peine de conséquences pouvant être graves.

Avant de détailler toutes ces notions, nous présentons d'abord un modèle de tâches qui nous servira de base dans la suite de ce mémoire.

2. Modélisation des tâches

Sur chaque site, les tâches peuvent être périodiques ou apériodiques. Du point de vue temporel, une tâche périodique T_i est caractérisée par quatre paramètres (voir figure 2.1):

- r_i: la date de première activation,
- C_i: la durée maximale d'exécution sur le processeur du site,
- D_i: le délai critique maximal que T_i est tenue de respecter,
- P_i: la période qui est l'intervalle de temps maximal séparant deux instances successives de T_i.

Nous avons nécessairement les relations suivantes: $0 \leq C_i \leq D_i \leq P_i$.

La première exécution d'une telle tâche sera demandée à l'instant r_i pour une durée maximale de C_i unités de temps et doit s'achever avant r_i+D_i. La $k$ième exécution de cette tâche sera demandée à l'instant $r_i+(k-1)P_i$ pour une durée maximale de C_i et doit se terminer avant $r_i+D_i+(k-1)P_i$. Lorsque la période P_i est égale au délai critique D_i, la tâche est dite à *échéance sur requête*.

Étant réveillée aléatoirement, une tâche apériodique est caractérisée par le triplet (r_i, C_i, D_i) où:

- r_i: la date initiale de réveil de la tâche,
- C_i: sa durée maximale d'exécution,
- D_i: son délai critique.

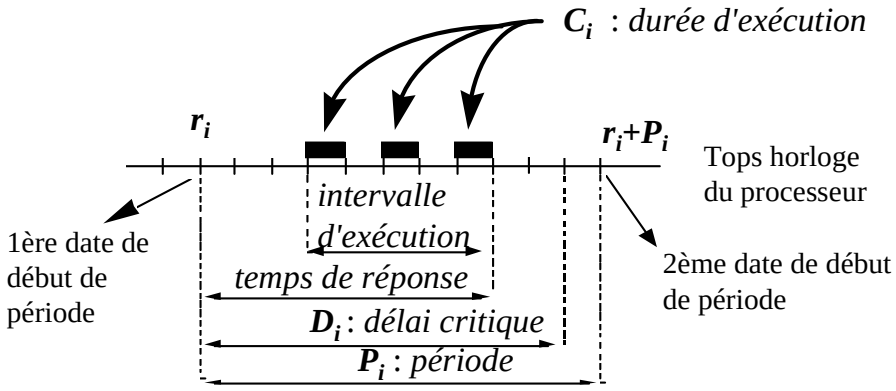


Figure 2.1: *Les caractéristiques temporelles d'une tâche périodique*

En général, les tâches peuvent interagir entre elles. Nous distinguons alors trois types d'interaction:

- *la synchronisation locale*: qui correspond à une simple synchronisation (postage et/ou attente d'un événement) ou communication de données (par le mécanisme de boîte aux lettres ou par rendez-vous si l'attente est bloquante) entre tâches d'un même site. Ce type de synchronisation peut être traduit par une précédence entre tâches définissant un ordre dans leur exécution.

- *la synchronisation globale* : qui correspond à l'échange de messages entre tâches de sites distincts. La tâche qui émet un message est dite *tâche émettrice*, celle qui le reçoit est dite *tâche réceptrice*. Ce type de synchronisation n'a pas de sens lorsque toutes les tâches résident sur le même site.

- *le partage de ressources* qui traduit l'utilisation de ressources critiques par deux ou plusieurs tâches du même site. C'est le problème de l'exclusion mutuelle résolu au niveau de l'exécutif temps réel à l'aide de mécanismes tels que les sémaphores, moniteurs, etc.

3. Représentation d'une séquence

L'ordonnanceur a pour fonction de fournir une séquence valide d'exécution des tâches. Une séquence d'exécution est dite valide si toutes les tâches se sont exécutées dans le respect de leurs contraintes temporelles, de précédence et de ressources.
Pour représenter une séquence d'exécution, nous avons choisi le diagramme de Gantt. Il permet de visualiser les diverses contraintes temporelles des tâches (r_i, C_i, D_i, P_i) et la séquence d'exécution. Sur un axe gradué en fonction du temps, l'évolution de l'état d'activité des tâches est représentée selon les notations suivantes:

- date de réveil:

- date de fin d'échéance: 

- échéance sur requête: 

- allocation du processeur pour la tâche pendant un quantum de temps égal à une période d'horloge: 

Un axe gradué est ajouté pour représenter l'allocation du processeur aux tâches. Lorsqu'aucune tâche n'est élue, l'intervalle est occupé par un zéro (0) ou par le symbole vide (∅). De plus, un axe gradué associé à chaque ressource est tracé en vue de mettre en évidence les différents instants d'utilisation des ressources. Les notations que nous reprenons et qui sont proposées dans [Bab 96] sont les suivantes (voir l'exemple de la figure 2.2):

- demande d'une ressource (par exemple, par la tâche A): A/

- attribution de la ressource à une tâche (par exemple, à A): A

- libération de la ressource (obligatoirement par la tâche en section critique): /*

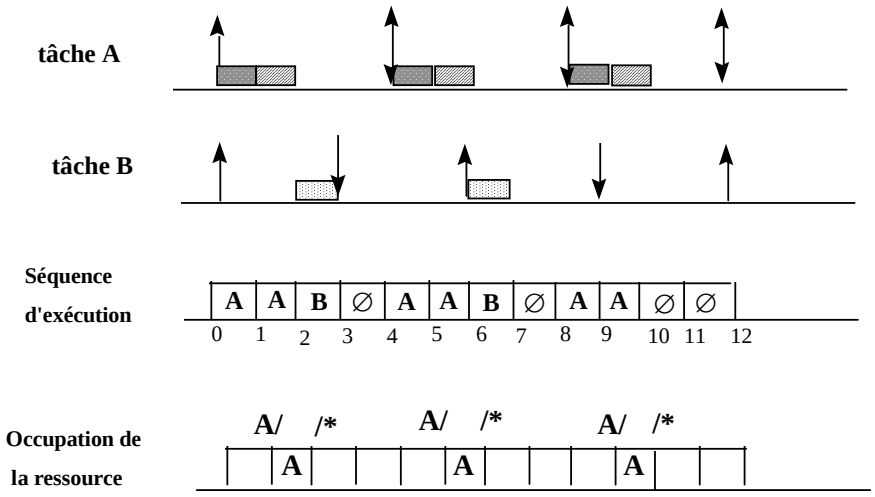


Figure 2.2: *Exemple de séquence d'exécution obtenue pour 2 tâches A et B de caractéristiques: $r_A=0$ $D_A=4$ $P_A=4$ $C_A=2$; $r_B=0$ $D_B=3$ $P_B=6$ $C_B=1$; priorité(A)>priorité(B); intervalle de validation [0,PPCM(4,6)]=[0,12]*

L'inconvénient de ce type de représentation est de ne pas faire apparaître les durées système (temps de changement de contexte, de prise en compte d'une interruption, etc.). Ces durées seront considérées comme négligeables ou intégrées dans les durées d'exécution des tâches.

3.1. Durée de validation

Il faut choisir un intervalle de temps englobant une séquence de tâches périodiques et apériodiques. Cet intervalle de temps doit être assez long pour que la séquence obtenue permette de valider l'ensemble de la configuration de tâches. Il est délimité par les instants *Début* et *Fin* avec:

Début = Min(r_i) si r_i est la date de première activation de toute tâche T_i

Fin correspond à l'instant où l'on trouve un contexte d'exécution, pour l'ensemble des tâches, identique au contexte à l'instant *Début*:

Fin = Max(r_i) + PPCM(P_i) si toutes les tâches sont périodiques

ou

Fin = Max($(r_i + D_i)_{apériodiques}$, $(r_i)_{périodiques}$) + 2 PPCM$(P_i)_{périodiques}$ s'il y a des tâches apériodiques [LM 80]

Si la séquence est valide sur l'intervalle [*Début*, *Fin*], alors elle est valide sur un temps infini [LM 80].

3.2. Condition nécessaire de validation

Pour une configuration, nous définissons le facteur d'utilisation *U* comme étant le pourcentage d'utilisation du processeur par les tâches périodiques: $U = \Sigma(C_i/P_i)$ et la

charge initiale *CH* comme le pourcentage d'utilisation du processeur par l'ensemble des tâches (périodiques et apériodiques): $CH = \Sigma(C_i / D_i)$. Il n'est évidemment pas possible d'utiliser le processeur au delà de ses capacités c'est-à-dire dépasser un taux d'occupation de 100%: c'est la condition nécessaire pour une configuration de tâches périodiques : $U \leq 1$.

4. Classification des tâches

Comme nous l'avons vu précédemment, les tâches d'un site peuvent se synchroniser c'est-à-dire respecter un ordre d'exécution ou précédence. Nous distinguons alors :

- *les tâches indépendantes* qui ne présentent pas de relation de précédence,
- *les tâches dépendantes localement* qui sont des tâches suivant ou précédant d'autres tâches (parce qu'elles se synchronisent ou communiquent entre elles) ; autrement dit entre deux tâches du même site est définie une relation de précédence locale.

Remarquons que le partage de ressources peut être vu comme un autre type de dépendance entre tâches.

Le problème d'ordonnancement est résolu différemment selon la classe de tâches considérée.

5. Les tâches indépendantes

Pour réaliser l'ordonnancement, une des méthodes consiste à attribuer une priorité à chaque tâche. L'élection de la tâche la plus prioritaire consiste d'abord à trier les tâches actives selon leurs priorités. La plupart des exécutifs temps réel intègrent la notion de priorité et de file ordonnée de tâches pour gérer ce type d'ordonnancement. Nous distinguons deux types de priorités. La *priorité fixe* est déterminée à l'activation de la tâche et reste constante tout au long de l'exécution. La *priorité dynamique*, par contre, évolue au cours de l'exécution de la tâche.

5.1. Les tâches périodiques

Pour l'ordonnancement des tâches périodiques, nous présentons des exemples d'algorithmes à priorité fixe et à priorité dynamique, tous préemptifs autrement dit l'ordonnanceur pourra élire une nouvelle tâche à chaque fois que la tâche en cours a fini de s'exécuter ou est bloquée ou encore à chaque fois qu'une nouvelle tâche plus prioritaire que la tâche courante demande à s'exécuter.

5.1.1. Les algorithmes d'ordonnancement à priorité fixe

Pour de tels algorithmes [LL 73], les priorités sont fixes, attribuées une fois pour toutes. On affecte une priorité à chaque tâche du système que celle-ci gardera pendant toute la vie de l'application. Les algorithmes à priorité fixe les plus répandus sont les suivants:

- L'algorithme *Rate Monotonic* (RM): la plus grande priorité est associée à la tâche de plus petite période. Pour les tâches de même période, l'affectation d'une priorité est faite de manière aléatoire.

Cet algorithme est optimal[1] parmi les algorithmes à priorité fixe pour des tâches à échéance sur requête.

Exemple

Soit l'exemple de deux tâches *A* et *B* dont les caractéristiques sont données dans le tableau 2.1. La priorité de *A* est supérieure à celle de *B* car la période de *A* est inférieure à celle de *B*.

Si on déroule la séquence d'exécution sur un intervalle de temps égal à [0, 0+PPCM(6,8)=24], on s'aperçoit que la séquence est bien valide, c'est-à-dire que l'exécution de chaque tâche est réalisée dans le respect de son échéance comme le montre la figure 2.3.

Condition d'ordonnançabilité

Une configuration de tâches est ordonnançable s'il existe un algorithme d'ordonnancement capable de produire un ordonnancement de ces tâches respectant leurs échéances. De nombreuses études [LL 73] ont pour finalité l'établissement de conditions nécessaires et/ou suffisantes d'ordonnancement de tâches car elles permettent de vérifier aisément et *a priori* une telle politique d'ordonnancement.
Avec RM comme algorithme d'ordonnancement et pour une configuration de *n* tâches périodiques indépendantes, une condition suffisante d'ordonnançabilité est donnée par [LL 73].

Théorème [LL 73]:

Toute configuration de n tâches périodiques à échéance sur requête est fiablement ordonnancée par RM si : $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n(2^{1/n}-1)$.

Lorsque *n* est grand, $n(2^{1/n}-1)$ converge vers *Ln2*= 0,69.

Tableau 2.1: *Caractéristiques temporelles des tâches*

Tâches	*ri*	*Ci*	*Di*	*Pi*
A	0	2	6	6
B	0	3	5	8

[1] un algorithme est optimal à l'intérieur d'une classe d'algorithmes s'il permet d'ordonnancer toute application ordonnançable par un autre algorithme de la classe considérée.

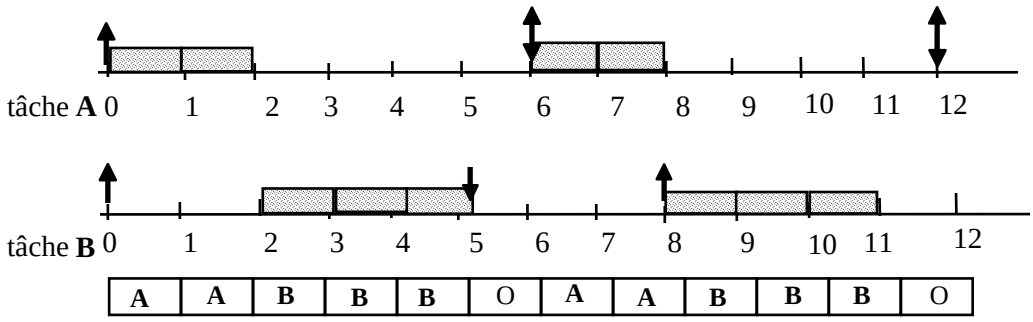


Figure 2.3: *Séquence valide obtenue avec RM pour la configuration du tableau 2.1*

Exemple

Soient trois tâches indépendantes T_1, T_2 et T_3 dont les caractéristiques temporelles sont données dans le tableau 2.2.

Calculons $U_i=C_i/P_i$ et $U_i^* = i(2^{1/i}-1)$.

L'ensemble $\{T_1, T_2, T_3\}$ est ordonnancé par *RM* étant donné que le test d'ordonnançabilité est vérifié, U_3 cumulé=0,50<U_3^*=0,779.

Supposons que T_2 nécessite une plus grande durée d'exécution, par exemple C_2=61 comme dans le tableau 2.3. La condition d'ordonnançabilité n'étant pas vérifiée, on ne peut rien conclure quant à l'ordonnançabilité de cette configuration de tâches car même dans ce cas l'ordonnancement peut être valide (la condition est suffisante mais pas nécessaire).

♦ L'algorithme *Deadline Monotonic* (DM): associe la plus forte priorité à la tâche de plus petit délai critique. Pour les tâches de même délai critique, l'affectation de priorités se fait de manière aléatoire. En échéance sur requête, l'algorithme est équivalent à RM.

Exemple

Reprenons l'exemple de tâches dont les caractéristiques sont données dans le tableau 2.1. Le diagramme de Gantt tracé pour l'algorithme DM (voir figure 2.4) est différent de celui de la figure 2.3 car la priorité de *A* est inférieure à celle de *B* vu que le délai critique de *A* est supérieur à celui de *B*.

Tableau 2.2: *Exemple de configuration de tâches*

Tâches	P_i	C_i	D_i	U_i	U_i *cumulé*	U_i*
T_1	80	20	80	0,25	0,25	<1,0
T_2	100	15	100	0,15	0,40	<0,828
T_3	300	30	300	0,10	0,50	<0,779

Tableau 2.3: *Autre exemple de configuration de tâches*

Tâches	P_i	C_i	D_i	U_i	U_i *cumulé*	U_i*
T_1	80	20	80	0,25	0,25	<1,0
T_2	100	61	100	0,61	0,860	>0,828
T_3	300	30	300	0,10	0,96	>0,779

Condition d'ordonnançabilité
DM est optimal pour des tâches périodiques indépendantes. Une condition suffisante d'ordonnançabilité pour des tâches quelconques est donnée par: $CH = \sum (C_i / D_i) < 1$.

Dans l'exemple du tableau 2.1, *CH* est égal à 0,93 c'est-à-dire plus petite que 1. Ce qui implique que la configuration de tâches est ordonnançable avec DM. Ceci se vérifie bien dans la figure 2.4.

La connaissance de *CH*, dans le cadre de tâches indépendantes, permet de connaître la validité de la séquence. Cette propriété de DM permet aussi de savoir simplement et rapidement à tout instant si la configuration peut accepter une surcharge ou non. Un algorithme d'acceptation de tâche en ligne peut être alors implanté.

5.1.2. Les algorithmes d'ordonnancement à priorité dynamique

Dans les politiques d'ordonnancement à priorité dynamique, chaque tâche verra évoluer sa priorité au cours de la vie de l'application. On distingue principalement deux types d'algorithmes:

♦ L'algorithme *Earliest Deadline* (ED): ordonne les tâches selon les échéances croissantes. Cet algorithme est optimal pour un système sans partage de ressources.

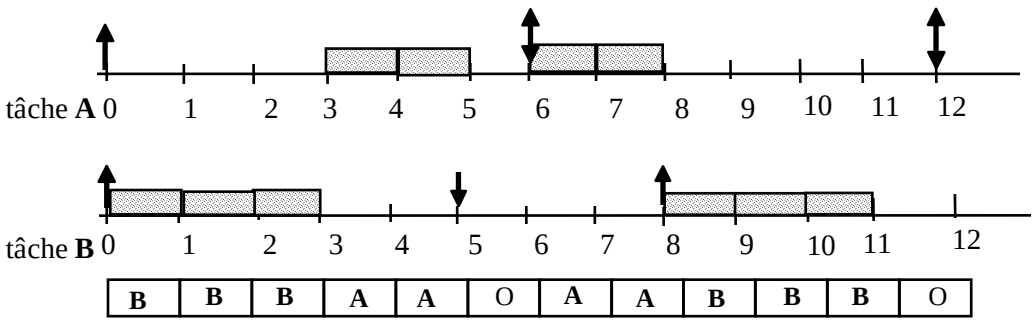


Figure 2.4*: Séquence valide obtenue avec DM pour la configuration du tableau 2.1*

Exemple

Soit la configuration de trois tâches présentées dans le tableau 2.4. A chaque instant, les tâches sont classées dans l'ordre croissant des échéances. Dans la figure 2.5, initialement (t=0), priorité(C) > priorité(A) > priorité (B) car échéance (C)< échéance(A) < échéance(B) sachant que l'échéance d_i est égale à D_i+r_i. C'est donc C qui est activée en premier. A l'instant 1, C se termine et c'est A qui s'exécute car son échéance est égale à 4 alors que celle de B est égale à 8. A l'instant 3, lorsque A se termine, B commence à s'exécuter. A l'instant 4, C se réveille, son échéance étant égale à 7, elle est plus petite que celle de B qui est égale à 8, B est donc préemptée par C.
A l'instant 5, C se termine et B reprend son exécution. A l'instant 6, l'échéance de B (égale à 8) est plus petite que celle de A (égale à 10) ; ce qui implique que la tâche courante B continue son exécution même si la tâche A est réveillée à cet instant. De même, la tâche A continue à s'exécuter à l'instant 8 même si les tâches B et C sont réveillées car son échéance est plus petite.

Condition d'ordonnançabilité

Dans le cas de tâches périodiques et apériodiques indépendantes, une condition suffisante d'ordonnançabilité est la suivante:

Toute configuration de n tâches quelconques est ordonnancée par ED si: $\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$.

Tableau 2.4: *Exemple de configuration de tâches pour l'algorithme ED*

Tâches	r_i	C_i	D_i	$d_i=D_i+r_i$	P_i
A	0	2	4	4	6
B	0	3	8	8	8
C	0	1	3	3	4

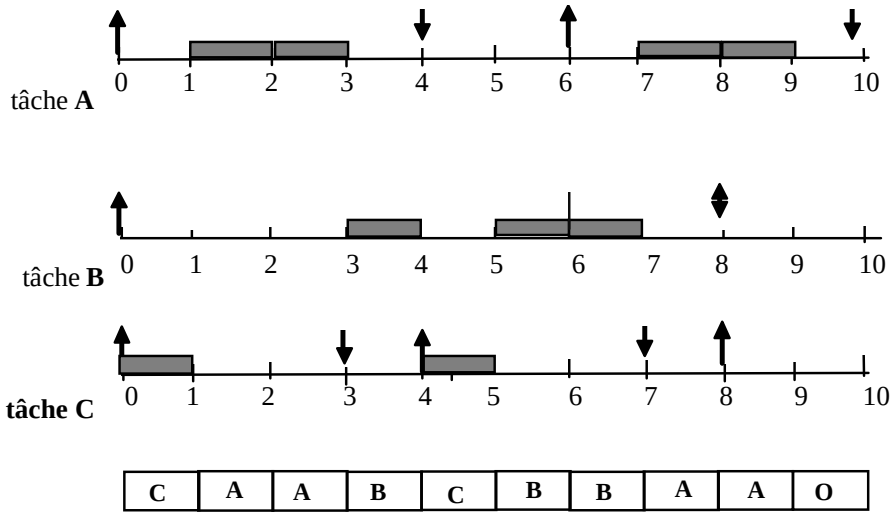


Figure 2.5: *Séquence valide obtenue avec ED pour l'exemple du tableau 2.4*

Si les tâches sont périodiques indépendantes et à échéance sur requête, la condition suivante [LL73] est nécessaire et suffisante pour ED.

> **Condition nécessaire et suffisante:** *toute configuration de n tâches périodiques à échéance sur requête est ordonnancée par ED si et seulement si son facteur d'utilisation U vérifie:* $U=\sum_{i=1}^{n}\frac{C_i}{P_i}\leq 1$.

♦ L'algorithme *Least Laxity* (LL): il associe la plus forte priorité à la tâche qui possède le plus petit temps creux dynamique. Le temps creux dynamique est l'intervalle de temps qui sépare la fin d'exécution d'une tâche de son échéance c'est-à-dire : $\delta_i = d_i(t)-C_i(t)-t$ où $d_i(t)$ est l'échéance de la tâche T_i à l'instant t, $C_i(t)$ son temps d'exécution restant au même instant.

Exemple

Soit la configuration de tâches présentées dans le tableau 2.5. Initialement (t=0), δ_A= 4-2-0=2, δ_B=8-3-0=5 et δ_C=4-1-0=3, ce qui donne *A* plus prioritaire que *C* qui est plus prioritaire que *B*. A t=2, *A* se termine. Comme δ_C=4-2-2=0 et δ_B =8-3-2=3 C s'exécute avant *B*. A t= 4, *C* arrive et *B* continue de s'exécuter car δ_B=8-2-4=2 et δ_C=8-1-4=3 (voir figure 2.6).
A t=6 *B* se termine, *A* et *C* sont prêtes à s'exécuter. Comme δ_A=10-2-6=2 et δ_C=8-1-6=1, c'est *C* qui prend le processeur. Le même principe s'applique à tout instant où il y a une demande concurrente du processeur.

> **Condition nécessaire et suffisante d'ordonnançabilité:** *toute configuration de n tâches périodiques indépendantes à échéance sur requête est fiablement ordonnancée par LL si et seulement si son facteur d'utilisation U vérifie:* $U\leq 1$.

Il a été démontré que l'algorithme qui engendre le moins de préemptions parmi les algorithmes présentés ci-dessus est ED [Hen 75]. C'est donc ED qui utilise le mieux le temps CPU.

Tableau 2.5: *Exemple de configuration de tâches pour l'algorithme LL*

Tâche	r_i	C_i	D_i	P_i
A	0	2	4	6
B	0	3	8	8
C	0	1	4	4

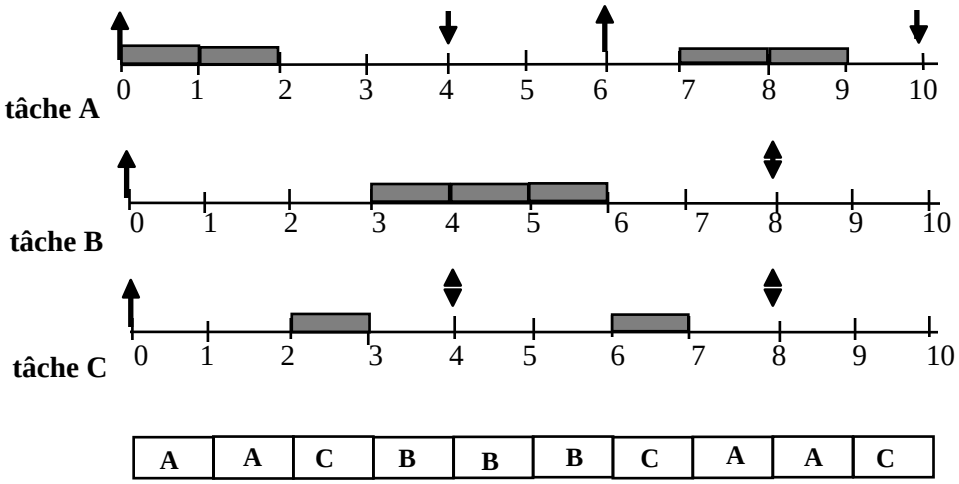


Figure 2.6: *Séquence valide obtenue avec LL pour l'exemple du tableau 2.5*

5.2. Les tâches apériodiques

Dans le contrôle de procédé, des événements aléatoires en provenance du procédé ou des événements internes sans aucune synchronisation avec le système peuvent survenir à tout instant. Celui-ci doit reconnaître la nature d'un événement et déclencher l'exécution du service lié à cet événement. La prise en compte d'un tel événement se fait grâce à une tâche apériodique. Le problème est alors d'ordonnancer ce type de tâches.

Une première solution consiste à ordonnancer les tâches apériodiques, en second plan, au sein de la configuration des tâches périodiques. On commence par ordonnancer l'ensemble des tâches périodiques afin de déduire les temps creux où seront insérées les tâches apériodiques.

Une autre solution consiste à ordonnancer les tâches apériodiques sur le même plan que les tâches périodiques. Une tâche périodique appelée *serveur* est dédiée à l'ordonnancement des tâches apériodiques.

Les caractéristiques temporelles du serveur sont celles d'une tâche classique:

- r_s: la date de réveil,
- C_s: la durée maximale d'exécution réservée à l'exécution des traitements non périodiques,
- P_s: la période évaluée de sorte à prendre en compte l'ensemble des requêtes non périodiques,

calculées en fonction des caractéristiques des tâches apériodiques à servir.

Il existe principalement deux types de serveurs [SSL 89] qui diffèrent dans la manière de prendre en compte les tâches apériodiques.

5.2.1. Le serveur à scrutation

Si aucune requête non périodique n'a été mémorisée avant la date d'activation du serveur, celui-ci aura *un temps d'exécution nul* pendant la période courante (voir figure 2.7).

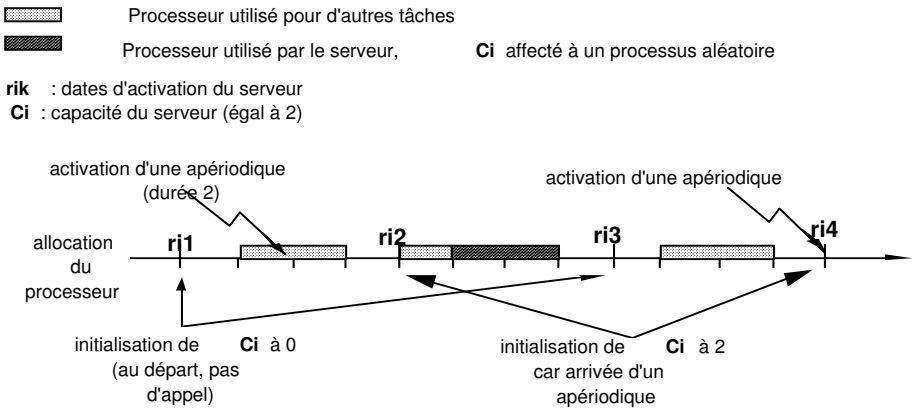


Figure 2.7: *Le serveur à scrutation*

Si, par contre, une requête a été mémorisée, le serveur utilisera son temps d'exécution C_s à sa prochaine activation pour exécuter le traitement associé. Si le traitement n'est pas terminé sur une période, il sera poursuivi pendant la (les) période(s) suivante(s). Le temps de réponse (temps écoulé entre la date de réveil et la date de fin d'exécution) des tâches apériodiques peut être important surtout si elles surviennent juste après la fin d'exécution du serveur. Au niveau de l'analyse de l'ordonnançabilité, ce serveur est considéré comme une tâche périodique.

5.2.2. Le serveur ajournable

Pour remédier à l'incompatibilité des rythmes d'arrivée, ce serveur maintient son temps d'exécution même si aucune requête non périodique n'est survenue (voir figure 2.8). Dans le cas où une requête survient, le serveur exécute le traitement associé dans la mesure où son temps n'est pas épuisé ; il n'attend pas la prochaine activation pour la servir.

5.2.3. Autres méthodes

Il existe une méthode dite de la *pseudo période* qui s'applique avec l'algorithme d'ordonnancement RM car elle consiste à associer à chaque tâche apériodique une pseudo-période correspondant à l'intervalle minimal séparant deux arrivées successives de la tâche associée. Les tâches apériodiques sont donc ordonnancées au même niveau que les tâches périodiques. L'inconvénient de cette méthode est la difficulté d'estimation de la pseudo-période.

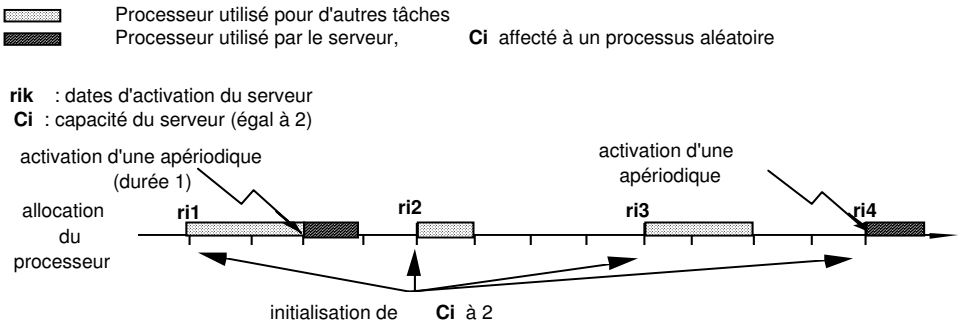


Figure 2.8: *Le serveur ajournable*

Une autre approche utilise une routine de garantie déclenchée, à chaque arrivée d'une tâche apériodique, qui vérifie si cette tâche peut être exécutée en respectant son échéance sans toutefois mettre en péril les échéances des tâches périodiques et des tâches apériodiques préalablement acceptées. Ce principe est utilisé dans le projet Spring [SR 91].
Si la routine de garantie ne réussit pas à trouver un tel ordonnancement et si l'environnement est distribué, un ordonnanceur réparti est lancé afin de trouver un site moins chargé susceptible de garantir cette tâche.

6. Les tâches dépendantes

6.1. Prise en compte des relations de précédence

Les relations de précédence expriment des contraintes d'exécution entre les tâches. Nous imposons l'égalité des périodes entre toute paire de tâches liées par une relation de précédence parce que ces tâches doivent s'exécuter le même nombre de fois. Pour cela, durant leur période, chaque tâche qui s'exécute une fois sera suivie par une seule exécution de la tâche appartenant à la même relation de précédence. Ce qui garantit la synchronisation entre les deux tâches.
L'objectif des algorithmes de prise en compte des relations de précédence [Bab 96] est de transformer la configuration de tâches avec relations de précédence en une configuration de tâches indépendantes. L'idée est donc d'affecter des priorités (selon la stratégie de l'algorithme d'ordonnancement utilisé) aux tâches en respectant les relations de précédence de manière à ce qu'une tâche ne s'exécute jamais avant la tâche la précédant, sans toutefois contredire les ordres de priorité entre tâches de périodes différentes.

Si la tâche A précède la tâche B avec les caractéristiques temporelles suivantes: $A=(r_A, C_A, D_A, P_A)$ et $B=(r_B, C_B, D_B, P_B)$ alors les transformations générales à effectuer sur les paramètres temporels afin de rendre A et B indépendantes sont les suivantes:

Règles de précédence [Bab 96]

> A précède $B \Rightarrow$
>
> - $r_B \geq r_A$
> - Si une tâche est périodique, l'autre l'est obligatoirement et $P_A=P_B$
> - Priorité(A)>Priorité(B)

Ces transformations se traduisent différemment selon l'algorithme d'ordonnancement utilisé.

6.1.1. Précédence avec l'algorithme RM

Lorsque les tâches sont périodiques et de même période, l'algorithme RM leur affecte des priorités de manière arbitraire. La prise en compte des relations de précédence dans ce cas se traduit par:

r_i*=Max{r_i, (r*$_{prédécesseur}$)} pour $T_{prédécesseur}$ tâche précédant T_i

Si *A* précède *B* avec P_A=P_B alors Priorité(*A*)>Priorité(*B*) selon RM

Exemple

Soit le graphe de précédence des tâches de la figure 2.9 tel que *A* précède *B* et *C* et *D* précèdent *E*. *A* et *B* sont de période 5 ; *C*, *D* et *E* sont de période 6. Les priorités des tâches doivent respecter les règles de précédence et celles de RM comme dans le tableau 2.6.

6.1.2. Précédence avec l'algorithme DM

L'idée est de modifier les échéances [Bla 76] de manière à affecter à une tâche un délai critique D_i inférieur à ceux de ses successeurs dans le graphe de précédence. On est alors sûr que les tâches s'exécuteront dans l'ordre souhaité tout en les considérant comme tâches indépendantes et sans modification de l'ordre de priorité défini par DM.

Les transformations sont alors les suivantes:

r_i*=Max{r_i, (r*$_{prédécesseur}$)} avec $T_{prédécesseur}$ tâche précédant T_i

D_i*=Min{D_i, (D*$_{successeur}$)} pour $T_{successeur}$ tâche suivant T_i

Si *A* précède *B* avec P_A=P_B alors Priorité(*A*)>Priorité(*B*) selon DM

6.1.3. Précédence avec l'algorithme ED

La modification des contraintes doit permettre d'aboutir à une configuration qui, lors de l'ordonnancement, vérifiera les relations de précédence ; c'est-à-dire qui imposera aux tâches, dans le graphe de précédence, de commencer après leurs prédécesseurs. Les échéances sont alors modifiées de manière à associer à une tâche une échéance d_i (d_i=r_i+D_i) strictement inférieure à celles de ses successeurs [CSB 90] étant donné que le calcul de priorité est basé sur cette échéance.

On obtient alors:

r_i*=Max{r_i, (r*$_{prédécesseur}$+ $C_{prédécesseur}$)}

d_i*=Min{d_i, (d*$_{successeur}$- $C_{successeur}$)}

Tableau 2.6 : *Priorités des tâches de l'exemple de la figure 2.9*

Tâche	*A*	*B*	*C*	*D*	*E*
Priorité[2]	5	4	2	3	1

[2] La valeur la plus grande est associée à la tâche la plus prioritaire

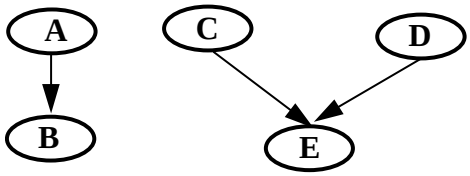


Figure 2.9: *Exemple de graphe de précédence*

Exemple

Soient les caractéristiques temporelles des tâches (voir tableau 2.7) ainsi que leurs relations de précédence représentées par le graphe de la figure 2.10. La prise en compte de la précédence nous donne des tâches indépendantes avec les nouvelles caractéristiques temporelles du tableau 2.8.

Tableau 2.7: *Exemples de contraintes temporelles de tâches **avant** transformation due à la prise en compte des relations de précédence présentées figure 2.10*

Nom Tâche	r_i *date de départ*	C_i *temps d'exécution*	D_i *délai critique*	$d_i=r_i+D_i$ *échéance*	P_i *période*
A	0	1	12	12	12
B	0	2	11	11	12
C	0	1	11	11	12
D	0	2	9	9	12
E	0	1	8	8	8
F	5	4	5	10	-

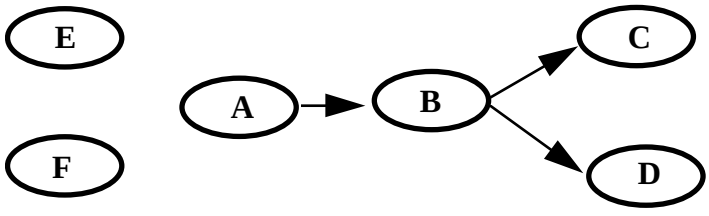


Figure 2.10: *Graphe de précédence des tâches présentées dans le tableau 2.7*

Tableau 2.8:*Exemples de contraintes temporelles de tâches **après** transformation due à la prise en compte des relations de précédence présentées figure 2.10*

Nom Tâche	r_i* *date de départ*	C_i *temps d'exécution*	d_i* *échéance*	P_i *période*
A	0	1	5	12
B	1	2	7	12
C	3	1	11	12
D	3	2	9	12
E	0	1	8	8
F	5	4	10	-

6.2. La prise en compte du partage de ressources

Les tâches peuvent interagir entre elles en partageant des ressources communes. Lorsque ces dernières sont critiques, il faut garantir qu'à tout instant une ressource critique n'est allouée qu'à une seule tâche. Le problème de l'exclusion mutuelle est résolu, par l'exécutif temps réel, par divers mécanismes tels que les sémaphores, les moniteurs, etc. Cependant deux problèmes restent à résoudre: *l'interblocage* et *l'inversion de priorité*.

L'interblocage : pour mettre en évidence ce phénomène, traitons l'exemple suivant : soient deux tâches T_1 et T_2 qui se partagent deux ressources c_1 et c_2. Supposons que T_1 a une priorité supérieure à celle de T_2, et analysons la séquence suivante :

> A l'instant t_1, T_2 se réveille en premier, demande c_2 et l'obtient,
> A l'instant t_2, T_1 se réveille, étant plus prioritaire elle préempte T_2 et s'exécute,
> A l'instant t_3, T_1 demande c_1 et c_2, elle obtient c_1 mais se bloque en attente de c_2 détenue par T_2. T_2 reprend l'exécution et demande c_1 détenue par T_1 qui est en attente de c_2 détenue par T_2: c'est le blocage mutuel.

L'inversion de priorités apparaît lorsqu'une tâche prioritaire *A* est retardée par une tâche moins prioritaire *B* parce que *A* demande une ressource déjà détenue par une tâche *C* moins prioritaire (que *A* et *B*). Ce phénomène est illustré à travers l'exemple suivant:

Exemple

Soient T_1, ..., T_n des tâches à priorités décroissantes avec T_1 et T_n qui partagent une ressource commune.
A l'instant t_1: T_n commence l'exécution et entre en section critique,
A l'instant t_2: T_1 interrompt T_n car la priorité de T_1 est supérieure à celle de T_n,
A l'instant t_4: T_1 demande la ressource occupée par T_n et se bloque, T_n reprend son exécution,
A l'instant t_5: T_2 demande le processeur qu'elle obtient car la priorité de T_2 est plus grande que celle de T_n, donc T_n est interrompue avant la libération de la section critique (voir figure 2.11).

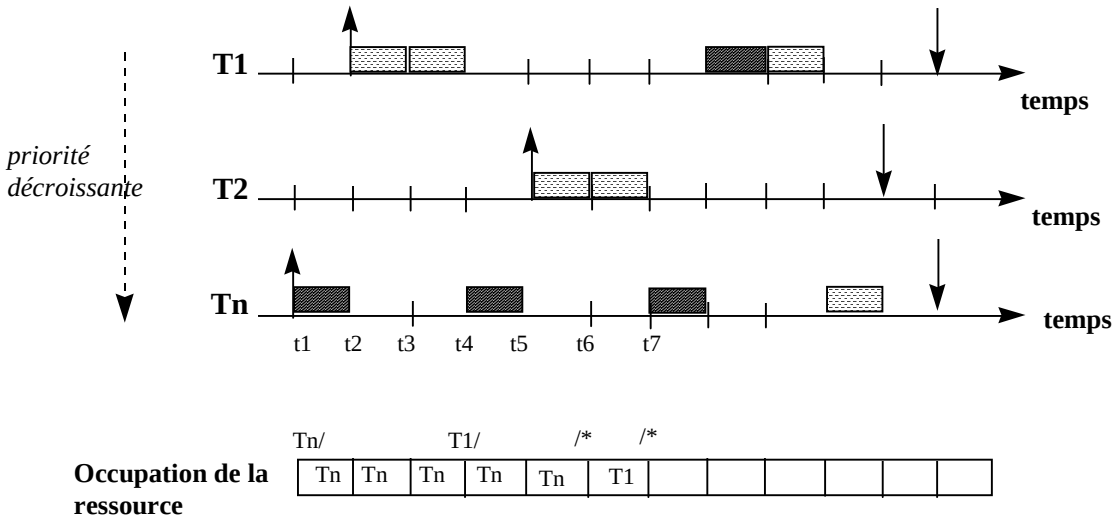


Figure 2.11: *Exemple d'inversion de priorité*

T_2, tout comme les autres tâches comprises entre T_1 et T_n qui ne partagent pas la ressource, pourra s'exécuter jusqu'à la fin avant que T_n ne reprenne son exécution et libère la section critique pour qu'enfin T_1 reprenne l'exécution. Nous assistons, dans ce cas, à une inversion de priorités car T_1 a été bloquée par des tâches moins prioritaires.

Les solutions au problème de l'inversion de priorités existent et sont appelées *protocoles d'allocation de ressources*. Elles sont intégrées aux algorithmes d'ordonnancement et adoptent des stratégies pour réduire le temps de blocage d'une tâche en attente d'une ressource détenue par une tâche moins prioritaire. Nous présentons trois protocoles d'allocation de ressources différents, nous décrivons leurs principes et nous montrons comment calculer le temps de blocage dans chaque cas. Nous supposerons, au chapitre 5, que nous savons calculer ce temps.

6.2.1. Le protocole à priorité héritée

Ce protocole (en anglais, *Priority Inheritance Protocol*) permet à la tâche en section critique d'hériter de la plus haute priorité parmi les tâches bloquées. Autrement dit, lorsqu'une tâche T de faible priorité bloque des tâches T_1, T_2, ..., T_k de plus forte priorité alors T s'exécute avec une priorité égale à Max {p(T_1), p(T_2), ..., p(T_k)} où p(T_i) représente la priorité de T_i [Raj91]. C'est ainsi que dans l'exemple de la figure 2.11, lorsque T_1 se bloque et que T_n reprend le calcul, T_n doit hériter de la priorité de T_1. Autrement dit, T_n sera ininterruptible jusqu'à la libération de la section critique, après quoi T_1 pourra reprendre son calcul (voir figure 2.12).

Théorème

Avec le protocole à priorité héritée [SRL90], s'il existe n tâches de priorité inférieure à la priorité de T alors T sera bloquée dans le pire des cas, pendant la durée de n sections critiques.

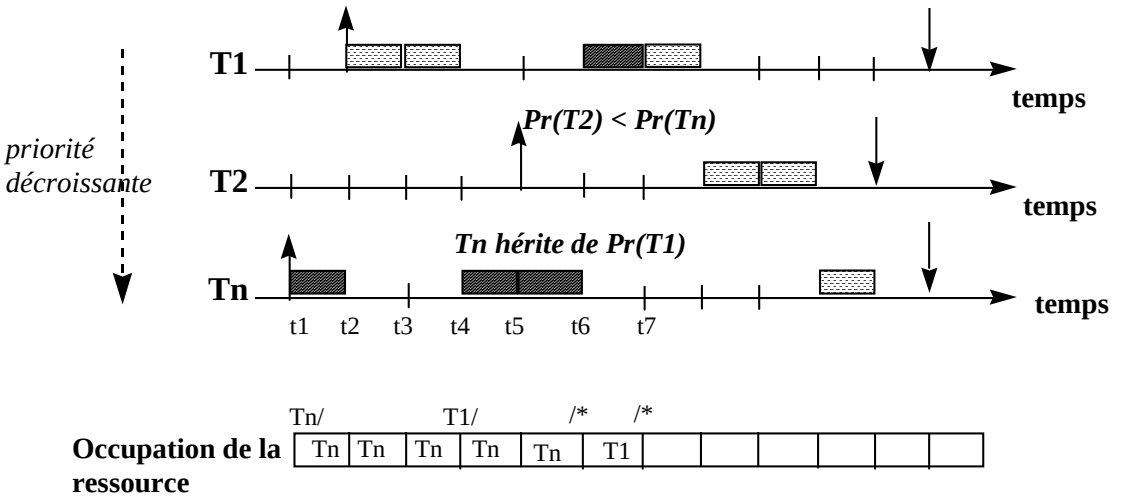


Figure 2.12: *Résolution de l'inversion de priorité par le protocole à priorité héritée*

En d'autres termes, s'il existe *m* sémaphores sur lesquels la tâche *T* peut se bloquer alors *T* peut être bloquée pendant la durée d'au plus *m* sections critiques.

Algorithme de calcul du temps de blocage maximal *R* [Mar 94]

Soit une tâche *T*:

1. Considérer les tâches moins prioritaires que *T*

2. Sélectionner celles qui possèdent des ressources communes avec *T*, soit *l* ce nombre

3. Considérer toutes les ressources dont a besoin *T*, soit *s* ce nombre

4. Prendre $\text{Max}(l,s) * \max_{i=1,s}$(durée des sections critiques pour utiliser la ressource c_i)

Fin.

Exemple

Considérons la configuration de tâches d'un site donné telle qu'elle est représentée dans la figure 2.13. Soient deux ressources critiques c_1 et c_2 disponibles sur ce site, A_1, A_2 et A_3 partagent la ressource c_1 et A_3, A_4 et A_5 la ressource c_2. Les caractéristiques temporelles et les priorités des tâches sont données dans le tableau 2.9.

Les priorités Pr_i sont affectées selon l'algorithme RM avec respect des contraintes de précédence locale. Toutes les valeurs sont exprimées en ms.
Comme dans [Kai 82], nous modélisons les tâches demandant l'allocation d'une ressource c_j comme suit.
Chaque tâche $T_i=(r_i, C_i, D_i, P_i)$ peut être découpée en trois parties α,β,γ.

α: séquence d'instructions précédant l'appel à la ressource,
β: séquence d'instructions de la section critique et
γ: séquence d'instructions suivant la libération de la ressource.

Nous noterons les durées de chaque séquence par $C^{\alpha}(c_j)$, $C^{\beta}(c_j)$, $C^{\gamma}(c_j)$ avec $C^{\alpha}(c_j)+C^{\beta}(c_j)+C^{\gamma}(c_j)=C_i$. Pour l'exemple de la figure 2.13, les durées de ces séquences, en particulier les temps d'utilisation des ressources par chaque tâche, sont données dans le tableau 2.10. Remarquons que A_3 est la seule tâche qui utilise les deux ressources simultanément puisque $C^{\alpha}(c_1)= C^{\alpha}(c_2)$ d'après le tableau 2.10.

Tableau 2.9: *Caractéristiques temporelles des tâches de la figure 2.13 (r_i=0 pour toutes les tâches)*

Tâche	C_i	D_i	P_i	Pr_i
A_1	10	50	60	5
A_2	8	150	200	2
A_3	10	100	100	3
A_4	9	50	60	4
A_5	5	200	200	1

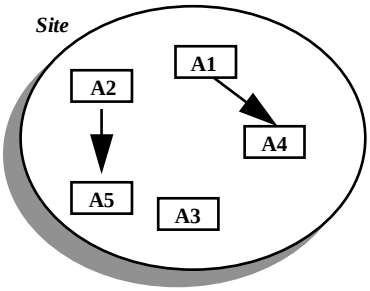


Figure 2.13: *Configuration d'une application monoprocesseur*

Tableau 2.10 : *Les durées des sections critiques pour les tâches de la figure 2.13*

Tâche	$C^{\alpha}(c_1)$	$C^{\beta}(c_1)$	$C^{\gamma}(c_1)$	$C^{\alpha}(c_2)$	$C^{\beta}(c_2)$	$C^{\gamma}(c_2)$
A_1	3	4	3	10	0	0
A_2	2	4	2	8	0	0
A_3	4	2	4	4	2	4
A_4	9	0	0	3	3	3
A_5	5	0	0	1	3	1

Notons que si $C^{\beta}(c_j)=0$ (pas d'utilisation de ressource), alors $C^{\gamma}(c_j)=0$.

Calculons la durée maximale de blocage R_{A3} pour la tâche A_3 par exemple dans le cas du protocole à priorité héritée. Pour cela, appliquons l'algorithme de calcul de R_{A3}:

⊗ trouvons les tâches moins prioritaires que A_3: A_2 et A_5 puisqu'elles ont une période plus grande,

⊗ parmi ces tâches, considérons uniquement celles qui partagent des ressources communes avec A_3: A_2 partage c_1 et A_5 partage c_2 (l=2),
⊗ le nombre de ressources communes est s=2,
⊗ pour utiliser c_1, la durée maximale est $C^{\beta}(c_1)=4$; pour utiliser c_2, la durée maximale est $C^{\beta}(c_2)=3$,
⊗ R_{A3}=max(l,s)*max(durées des sections critiques)=max(2,2)*max(4,3)=8

L'inconvénient de ce protocole est qu'il n'évite pas l'interblocage.

6.2.2. Le protocole à priorité plafond

Ce protocole (en anglais, *Priority Ceiling Protocol*) a été proposé pour limiter la durée maximale de blocage à la durée d'une section critique et pour prévenir l'interblocage. Pour éviter les blocages, on définit une nouvelle notion: *la priorité plafond* d'une ressource, qui est le maximum des priorités des tâches pouvant accéder à la ressource. Une tâche ne peut entrer en section critique que si sa priorité est strictement supérieure à la priorité plafond des ressources alors utilisées.

Exemple

Soit une configuration de trois tâches *A*, *B* et *C* qui utilisent les ressources c_1, c_2 et c_3 comme suit :

- la tâche *A* utilise les ressources c_1 et c_3
- la tâche *B* utilise les ressources c_1, c_2 et c_3 et
- la tâche *C* utilise les ressources c_1 et c_2.

On suppose que priorité(*A*) > priorité(*B*) > priorité(*C*). Les priorités plafonds des ressources notées PP sont alors :

PP(c_1)= Max priorités (*A*, *B*, *C*) = priorité(*A*)
PP(c_2) = Max priorités (*B*, *C*) = priorité(*B*)
PP(c_3) = Max priorités (*A*, *B*) = priorité(*A*).

Dans l'exemple de la figure 2.11, si la ressource est partagée uniquement par T_1 et T_n, la priorité plafond de cette ressource sera égale à : Max(priorité(T_1), priorité(T_n))= priorité(T_1). Afin d'empêcher les tâches de priorité intermédiaire telle que T_2 de s'exécuter, T_n hérite, à l'instant t_4, de la priorité plafond de la ressource partagée c'est à dire de la priorité de T_1 dans ce cas (voir figure 2.14).

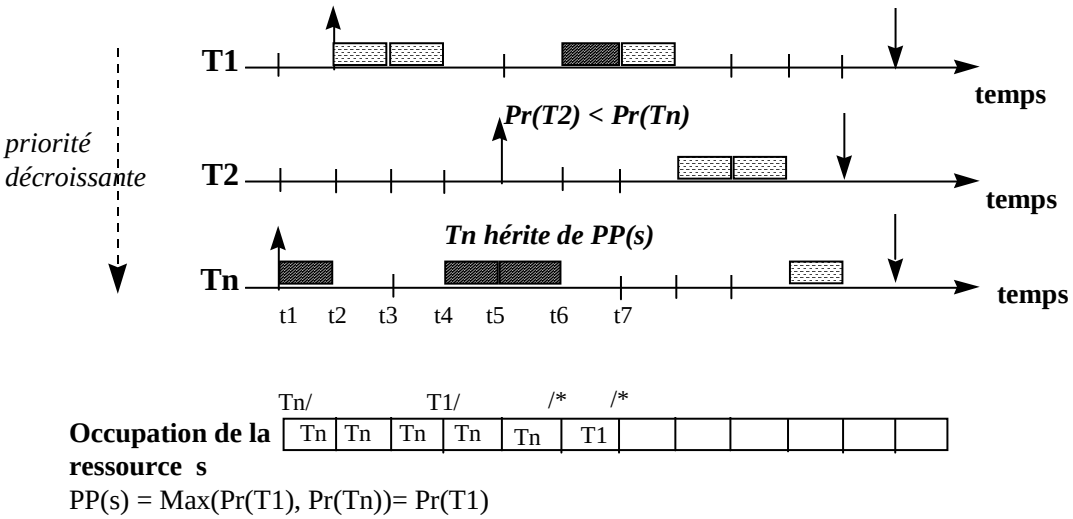


Figure 2.14: *Résolution de l'inversion de priorité par le protocole à priorité plafond*

Théorème [SRL 90]

> *Le protocole à priorité plafond évite des situations d'interblocage et garantit que toute tâche ne peut avoir sa priorité inversée pour plus de la durée de la section critique.*

Algorithme de calcul du temps de blocage maximal *R* [TH 95]

Soit une tâche *T*:

1. Considérer les tâches moins prioritaires que *T*

2. Sélectionner celles qui possèdent des ressources communes avec *T*

3. Considérer uniquement les ressources dont la priorité plafond est ≥ priorité (*T*), soit *s* ce nombre

4. Prendre $\max_{i=1,s}$(durée des sections critiques pour l'utilisation de la ressource c_i)

Fin.

Si *T* est la tâche la moins prioritaire, alors le temps de blocage est nul car il n'y a pas de tâche moins prioritaire qui peut la bloquer.

Exemple

Reprenons l'exemple de la figure 2.13 avec les mêmes caractéristiques décrites dans l'exemple précédent (§6.2.1). En utilisant le protocole à priorités plafond, calculons R_{A3} pour la tâche A_3:
Pour cela, supposons que les priorités des tâches respectent les règles de l'algorithme RM et les contraintes de précédence (voir tableau 2.9) et calculons les priorités plafond des ressources c_1 et c_2.
PP(c_1)=max priorités(A_1, A_2, A_3)=1,
PP(c_2)=max priorités(A_3, A_4, A_5)=2,

Appliquons l'algorithme de calcul de la durée maximale de blocage:

⊗ trouvons les tâches les moins prioritaires que A_3: A_2 et A_5

⊗ considérons parmi ces tâches, celles qui partagent des ressources avec A_3: A_2 partage c_1 et A_5 partage c_2,

⊗ prenons uniquement les ressources dont la priorité plafond est supérieure ou égale à la priorité de A_3: c_1 et c_2 puisque PP(c_1)=5, PP(c_2)=4 et priorité(A_3)=3,

⊗ R_{A3}=max($C^{\beta}(c_1)$, $C^{\beta}(c_2)$)=max(4,3)=4

Si l'algorithme d'ordonnancement utilisé est ED, le protocole à priorité plafond est dit *dynamique* car la priorité plafond d'une ressource est réévaluée à chaque instant, étant donné qu'elle dépend des priorités des tâches qui, elles, changent avec le temps.

Les deux protocoles d'allocation de ressources présentés jusqu'à présent se basent sur la priorité des tâches, ce qui signifie qu'ils s'adaptent mieux aux algorithmes à priorité fixe tels que RM ou DM. Le protocole présenté dans le paragraphe suivant s'adapte aussi aux algorithmes à priorité dynamique.

6.2.3. Le protocole d'allocation de la pile (PAP)

Ce protocole (en anglais, *Stack Resource Policy*) a été proposé par Baker [Bak 91]. Il améliore le protocole à priorité plafond en permettant:

- d'avoir plusieurs exemplaires de la même ressource,
- d'utiliser un ordonnancement dynamique tel que ED,
- de réduire le nombre maximal de changements de contexte.

La modification principale apportée au protocole à priorité plafond initial concerne la définition d'une nouvelle notion appelée *niveau de préemption* d'une tâche qui correspond à une seconde priorité qui doit être obligatoirement fixe et qui sert à définir la condition d'autorisation de préemption. En effet, une tâche T_j ne peut préempter une tâche T_i que si le niveau de préemption de T_j noté $\pi(T_j)$ est strictement supérieur à celui de T_i. La distinction entre priorité et niveau de préemption est faite en vue d'utiliser un ordonnancement à priorité dynamique.

Chaque ressource c a une valeur plafond courante notée $\lceil c \rceil$ et vérifiant la condition suivante : soit T qui doit accéder à la ressource c déjà allouée à d'autres tâches moins prioritaires. Si T est la tâche active ou, si T peut préempter la tâche active alors $\pi(T) \leq \lceil c \rceil$.

Une définition possible de $\lceil c \rceil$ est la valeur maximale des niveaux de préemption de toutes les tâches bloquées ou pouvant être bloquées pour l'accès à cette ressource c dont seules v unités sont accessibles :

$$\lceil c \rceil_v = \text{Max}\ (\{0\} \cup \{\pi(T) / v < \mu(T)\})$$

où $\mu(T)$ est le nombre maximal d'unités de la ressource c requises par la tâche T pendant son exécution.

Par extension, la valeur plafond courante du système est :

$\bar{\Pi}$ = Max $\{\lceil c_i \rceil / \ i=1,m\}$ *m* étant le nombre de ressources

Le protocole PAP impose à l'exécution d'une requête de ne pas débuter la tâche tant que $\bar{\Pi} > \pi(T)$. Ainsi, dès qu'une tâche commence son exécution, toutes les ressources qu'elle nécessite sont accesibles à sa demande.

Caractéristiques principales

- aucune tâche ne peut être bloquée en attente de ressources après son début d'exécution,
- il n'y a pas d'interblocage,
- comme pour le PPP, une tâche peut être bloquée (c'est-à-dire son exécution retardée) par une tâche moins prioritaire pour la durée d'au plus une section critique,
- la tâche de plus grande priorité parmi les tâches bloquées (c'est-à-dire dont l'exécution est retardée) sera débloquée dès l'instant où la tâche active aura libéré toutes ses ressources. Dans le cas où plusieurs tâches sont bloquées c'est la tâche qui est bloquée le plus longtemps qui sera débloquée,
- l'algorithme de calcul du temps de blocage est identique à celui de PPP excepté que la priorité plafond d'une ressource se calcule par rapport aux niveaux de préemption des tâches,
- Une différence importante entre les techniques du PPP et du PAP est que le protocole PPP garantit seulement qu'une tâche bloquée une fois ne le sera plus jamais. Par opposition, le protocole PAP garantit que si une tâche est bloquée, elle l'est avant de commencer son exécution.

Exemple

Soient 3 tâches T_1,T_2 et T_3 de priorités respectives 1, 2 et 3 confondues avec leurs niveaux de préemption.

$\lceil c \rceil_v$ = Max($\{0\} \cup \{\pi(T) / v<\mu(T)\}$) avec *v* le nombre d'unités de *c* disponibles et μ le nombre d'unités qu'une tâche *T* peut demander. *v* et μ sont au plus égaux à 1. $\bar{\Pi}$ = Max($\lceil c \rceil_1$)=0 initialement car toutes les ressources sont disponibles.

Le tableau 2.11 donne les priorités plafond des ressources c_1 et c_2 et les tableaux 2.12 et 2.13 donnent respectivement les caractéristiques des tâches et les durées d'exécution de ces ressources. L'influence de ce protocole sur l'ordonnancement des tâches est illustrée dans la figure 2.15. Comme le montre cette figure, $\bar{\Pi}$ est initialement égal à 0. T_1 demande et obtient la ressource c_1 car la priorité de T_1 est supérieure à $\bar{\Pi}$, $\bar{\Pi}$ passe alors à la valeur 2. Lorsque T_2 arrive, elle ne peut s'exécuter car sa priorité est égale à $\bar{\Pi}$. Au contraire quand T_3 se réveille, elle préempte T_1 car sa priorité est plus grande que $\bar{\Pi}$. Mais ce n'est que lorsque T_3 demande et obtient c_2 que $\bar{\Pi}$ passe à la valeur 3 car la mise à jour de $\bar{\Pi}$ ne se fait qu'à l'obtention ou à la libération d'une ressource. A la libération de c_2 par T_3, $\bar{\Pi}$

reprend sa valeur précédente, c'est-à-dire 2. $\bar{\Pi}$ ne repassera à 0 que lorsque c_1 sera libérée. Le même principe s'applique pour T_2 lors de l'acquisition de c_1 et de c_2.

Tableau 2.11: *Les priorités plafond des ressources c_1 et c_2*

c	v	$\mu(T_1)$	$\mu(T_2)$	$\mu(T_3)$	$\lceil c \rceil_0$	$\lceil c \rceil_1$
c_1	1	1	1	0	2	0
c_2	1	0	1	1	3	0

Tableau 2.12 : *Les caractéristiques temporelles des tâches*

Tâche	r_i	C_i	D_i	P_i
T_1	0	6	10	10
T_2	1	3	11	11
T_3	2	3	12	12

Tableau 2.13 : *Les durées des sections critiques*

Tâche	$C^{\alpha}(c_1)$	$C^{\beta}(c_1)$	$C^{\gamma}(c_1)$	$C^{\alpha}(c_2)$	$C^{\beta}(c_2)$	$C^{\gamma}(c_2)$
T_1	0	5	1	6	0	0
T_2	1	1	1	2	1	0
T_3	3	0	0	1	1	1

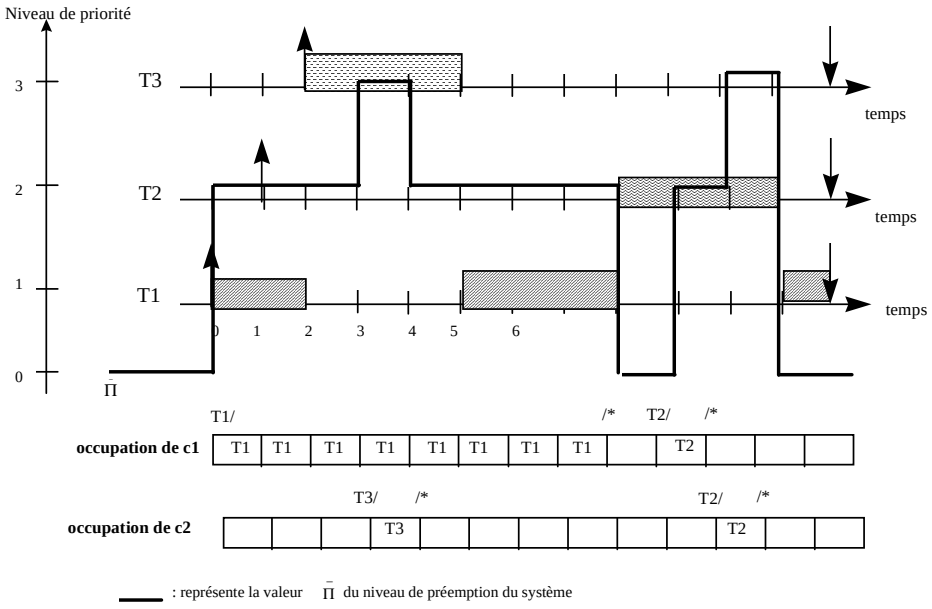


Figure 2.15: *Comportement des tâches avec un protocole à pile*

7. Conclusion

A travers ce chapitre, nous avons passé en revue les principaux algorithmes d'ordonnancement connus dans l'environnement monoprocesseur tout en mettant en évidence les modifications à opérer sur les contraintes temporelles en vue de prendre en compte les relations de précédence. De plus, des protocoles d'allocation de ressources ont été présentés afin d'être intégrés aux algorithmes d'ordonnancement et réduire le temps de blocage d'une tâche demandant une ressource critique déjà occupée.

Dans ce chapitre, nous avons distingué les tâches indépendantes et les tâches dépendantes. Selon la nature périodique ou apériodiques des tâches indépendantes, nous avons présenté les méthodes d'ordonnancement appropriées. Dans notre classification telle qu'elle est présentée dans le tableau 2.14, les tâches sont dépendantes lorsqu'elles sont liées par des relations de précédence locale ou bien lorsqu'elles partagent des ressources critiques. Nous avons, d'une part, montré les règles à suivre en vue de prendre en compte la précédence locale en fonction de l'algorithme d'ordonnancement considéré et exposé, d'autre part, le principe des différents protocoles d'allocation de ressources permettant de réduire le temps de blocage généré dans une situation d'inversion de priorités. L'environnement qui vient d'être décrit avec :

* les différents types de tâches,
* les méthodes supposées intégrées dans l'exécutif temps réel pour l'ordonnancement des tâches et la gestion de l'allocation des ressources

n'est rien d'autre que le contexte de tout site qui accueille les tâches de l'application temps réel répartie. Que cela soit en utilisant la méthode des pseudo-périodes ou la méthode des serveurs, les tâches apériodiques sont ordonnancées au même niveau que les tâches périodiques. A un niveau d'abstraction plus élevé, toutes les tâches de l'application répartie sont considérées comme périodiques.

Le chapitre suivant présente quelques protocoles de communication qui s'adaptent bien aux messages temps réel.

Tableau 2.14 : *Tableau récapitulatif*

Tâches		Méthodes/ Règles/ Protocoles
indépendantes	périodiques	RM, DM, ED
	apériodiques	serveurs, pseudo-période
dépendantes	Prise en compte de la précédence	transformation des paramètres temporels selon RM, DM ou ED
	Partage de ressources	PHP ou PPP avec RM ou DM PAP avec ED

CHAPITRE 3

LES PRINCIPAUX PROTOCOLES MAC ADAPTES AUX COMMUNICATIONS TEMPS REEL

1. Introduction

Le respect des délais de transfert des messages constitue la principale caractéristique qui permet de distinguer les communications temps réel des autres types de communication. Aussi, les applications temps réel utilisent-elles des réseaux qui offrent un mode de communication permettant de garantir une certaine qualité de service au niveau temporel pour l'échange des messages entre tâches.

Comme dans [Mam 97], La qualité de service se définit grâce à un certain nombre de paramètres tels que :

- *le délai maximal de transfert* : qui correspond à la durée maximale de transmission d'un message d'un site à un autre.
- *la gigue maximale* : correspondant à la variation maximale du délai de communication.
- *le taux de perte maximal* : lorsque le réseau est surchargé, il peut décider d'écarter certains messages. Selon que l'application sous-jacente tolère ou non la perte de messages, le taux de perte maximal peut être faible (voire nul) ou plus important.
- *le taux d'erreur maximal* : le taux d'erreurs n'est jamais nul même dans un réseau qualifié de fiable. Néanmoins pour chaque application, il existe un seuil que le réseau ne doit pas dépasser si l'on veut garantir la qualité de service requise.

Les messages échangés par les tâches sont spécifiés à l'aide des paramètres suivants :

- *la criticité* : lorsque les messages doivent respecter des délais de transfert, ils sont dits à contraintes strictes car le non respect de ces délais mène à la dégradation du système, voire même à la catastrophe. Les messages sont dits à contraintes relatives lorsque le système tolère leur dépassement de temps en temps.

- *la longueur* : c'est le nombre d'octets ou de bits constituant le message. La trame constituée d'un nombre fixe de bits est l'unité d'allocation transmise sur le réseau. Le message étant de taille quelconque, il peut correspondre à un nombre entier de trames. La taille de la trame varie d'un réseau à un autre.

- *le type* : un message est qualifié de périodique (ou de synchrone) lorsque l'intervalle séparant deux productions successives du même message est fixe. Il est dit apériodique (ou asynchrone) lorsqu'il peut arriver à n'importe quel moment. Le succès du transfert des messages apériodiques est d'autant plus possible que des informations telles que les lois d'arrivée et les échéances sont connues à l'avance.

- *la destination* : un message émis peut être adressé à un seul site, à un groupe de sites ou diffusé à tous les sites.

Une variété de protocoles à accès multiple a été proposée et évaluée tels que les protocoles basés sur un jeton temporisé (FDDI [ACZD 92, ISO 94]) ou basé sur la technique de compétition CSMA en introduisant des mécanismes déterministes (CSMA/DCR et DOD/CSMA-CD [LR 94], CSMA/CA de CAN [TBW 95]). Une synthèse des protocoles de communication dans les réseaux locaux temps réel est présentée dans [MZ 95] et [Ell 91]. Le premier [MZ 95] réalise une classification en se basant sur deux processus: l'arbitrage d'accès et le contrôle de transmission, le second [Ell 91] définit les propriétés attendues et les critères de comparaison des protocoles de communication. [ARS 91] définissent un modèle de communication qui tient compte des caractéristiques temporelles des messages et proposent un protocole à fenêtre généralisé.
De nombreux travaux ont été développés dans le cadre des réseaux locaux à accès multiple pour adapter ce type de réseaux à la prise en compte des contraintes de temps de messages en proposant des algorithmes d'ordonnancement de messages au-dessus de la couche MAC, souvent inspirés des algorithmes d'ordonnancement des tâches temps réel [ZR 87, KSY 84]. Une synthèse de ces travaux est présentée dans [Mam 97]. [SS 96b], par exemple utilise l'algorithme RM non préemptif sur chaque site FDDI pour ordonnancer les messages dans le but d'améliorer la capacité de transmission du trafic synchrone. Les messages sont ordonnés dans une file de transmission par ordre décroissant des périodes.

Dans la suite de ce chapitre, nous exposerons les principales techniques d'ordonnancement des messages temps réel implantées au niveau de la couche MAC en supposant que ces techniques suffisent pour l'ordonnancement des messages car même en ignorant les algorithmes d'adaptation, certains réseaux peuvent garantir un délai de communication borné, principale propriété des réseaux temps réel. Une classification des protocoles, basée sur ces techniques d'accès au médium de communication, est présentée en illustrant chaque classe à l'aide d'un ou de deux exemples de protocole.

Nous nous intéresserons uniquement aux messages périodiques à contraintes temps réel strictes puisque les tâches qui échangent ces messages sont aussi considérées périodiques à contraintes strictes. Les messages sont sans contrainte de gigue et sont caractérisés par des délais critiques confondus avec leurs périodes. Le réseau est supposé fiable avec un taux d'erreurs minimal.
Dans la suite, les termes *paquet* et *trame* seront sciemment confondus, il en sera de même pour les termes *station* et *site*. Dans notre terminologie, un message est l'entité d'information envoyée par une tâche application, qui peut correspondre alors à un ensemble de trames au niveau de la couche MAC.

2. Classification des protocoles MAC

Contrairement aux réseaux à commutation de paquets (ATM), les réseaux à accès multiple (FDDI, CAN, FIP, etc.) sont caractérisés par des étendues géographiques limitées. Dans ce type de réseaux, les sites connectés au réseau accèdent au médium en utilisant une technique MAC implantée sur les différents sites. Le médium étant une ressource partagée par les sites, l'accès se fait soit par compétition, soit par consultation (à l'aide d'un jeton par exemple) selon le protocole MAC utilisé par le réseau pour la transmission des paquets.

La classification des protocoles MAC temps réel proposée par [MZ 95] distingue deux processus pour la gestion de la communication :

- *un processus d'arbitrage d'accès*: qui détermine quand un site a le droit d'utiliser le médium pour émettre, et
- *un processus de contrôle de transmission*: qui détermine combien de temps un site a le droit d'utiliser le médium.

Dans le réseau FDDI, par exemple, qui est un anneau à jeton circulant, le processus d'arbitrage d'accès consiste à obtenir le droit d'accès au médium qui se matérialise par l'obtention du jeton et le processus de contrôle de transmission concerne l'émission, par un site qui détient le jeton (le droit d'accès), pour une durée correspondant à une bande passante fixée au démarrage du réseau.
L'étude des différents protocoles MAC fait apparaître que l'un des deux processus joue un rôle prépondérant. Par exemple, le protocole IEEE 802.5 développe plus son processus d'arbitrage d'accès en utilisant un arbitrage par priorités tandis que le protocole FDDI développe plus son processus de contrôle de transmission en définissant une bande passante synchrone égale à un temps fixe alloué à chaque site pour la transmission des messages synchrones.

3. Les techniques d'accès

La présentation classique des protocoles MAC est basée sur les techniques d'accès au médium en mettant en œuvre les processus d'arbitrage d'accès et de contrôle de transmission définis dans [MZ 95].
Les principales techniques d'accès sont illustrées à travers la figure 3.1 comme dans [Car 96].
L'accès aléatoire est une technique basée sur un principe de compétition puisqu'il s'agit, pour chaque site, d'émettre lorsqu'il le souhaite pourvu que le bus soit libre.

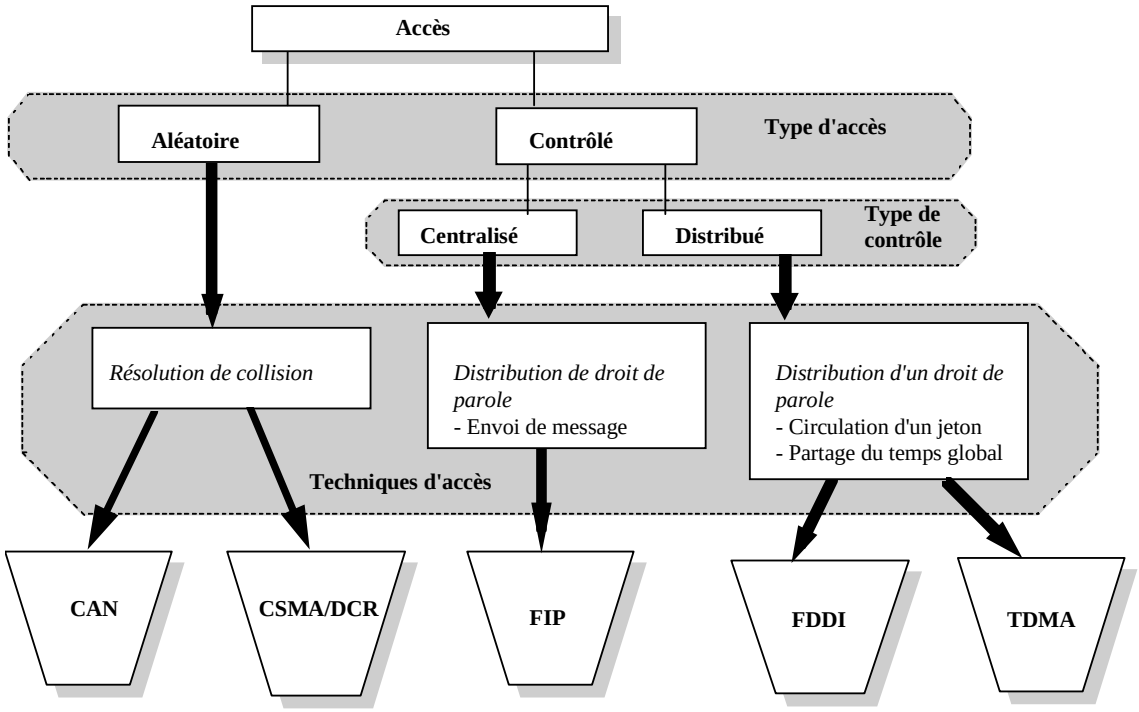


Figure 3.1: *Les techniques d'accès*

L'accès à contrôle centralisé suppose l'existence d'un site de contrôle qui distribue un droit de parole aux sites. C'est le cas par exemple de FIP où un site de contrôle envoie à chaque site un message l'autorisant à utiliser le médium.
La technique d'accès à contrôle distribué suppose la coopération des sites en vue de déterminer celui qui a le droit d'utiliser le médium. La technique du jeton circulant est un exemple de mécanisme de coopération explicite des sites (cas de FDDI) et la technique du partage du temps global en intervalles de temps alloués aux différents sites est un exemple de coopération implicite (cas de TDMA).

Dans les techniques d'accès de la couche MAC, l'ordonnancement de messages peut nécessiter de définir des priorités aux messages ou bien d'imposer, aux sites, un temps d'accès borné. En tenant compte de cet aspect, nous obtenons les familles de protocoles de la figure 3.1. Chacune d'elles est illustrée par un exemple de protocole qui sera décrit en détails dans la suite du chapitre.

4. Protocoles à accès aléatoire

4.1. Le protocole CAN

Le réseau CAN (Controller Area Network) [ISO 94] a été introduit initialement pour gérer la communication dans les véhicules. CAN est un bus à diffusion. Les sites ne possèdent pas d'adresses et accèdent au bus en utilisant la technique CSMA/CA (Carrier Sense Multiple Access /Collision Avoidance). Chaque site comporte un contrôleur CAN (type Intel 82527 ou Philips 82C200) qui assure l'interface avec le bus.
Le support de transmission est constitué d'une paire de fils torsadés. Les débits normalisés vont de 125Kb/s à 1Mb/s. Plus le bus est étendu, plus le débit est faible (voir tableau 3.1).

Tableau 3. 1 : *Débit et distance maximale*

Débit	Distance maximale
1Mb/s	40m
500 Kb/s	130m
250Kb/s	270m
125Kb/s	530m

Comme tout réseau local, CAN est composé de trois couches dont seules les couches physique et liaison de données sont normalisées [ISO 94]. Chaque paquet peut être transmis périodiquement, sporadiquement ou à la demande.
Un paquet contient au plus 8 octets d'information utile (voir structure du paquet dans l'annexe 3) et des champs de contrôle dont notamment un identificateur de 11 bits. L'identificateur est utilisé d'une part pour filtrer les paquets à la réception et d'autre part pour assigner une priorité au paquet qui sert à contrôler l'accès au bus lorsque plus d'un site est émetteur. En effet, si un site parmi un ensemble de sites émetteurs émet un bit 0 alors tous les sites verront 0 (dit *bit dominant*) sur le bus. Inversement, les sites verront 1 uniquement lorsque tous les sites émetteurs transmettront un bit 1 (dit *bit récessif*). Le bus se comportant comme un ET logique, seul le paquet de plus grande priorité (ayant le plus petit identificateur) sera envoyé car les sites qui émettent des bits récessifs se retirent de la contention lorsqu'ils détectent un bit dominant sur le bus. On dit que CAN est fondé sur un accès au médium avec priorité et avec résolution de collisions non destructive.

CAN utilise un mécanisme de *bitstuffing*, servant à signaler les erreurs aux différents sites, qui consiste à insérer un bit dit *de bourrage* de signe contraire à celui de la séquence de 5 bits identiques après laquelle il est inséré. Parmi les 47 bits composant les champs de contrôle, 34 bits peuvent contenir des bits de bourrage en plus des 8 octets de donnée. Le nombre total de bits de bourrage est alors donné, comme dans [TBW 95], par la formule suivante : $\left\lfloor \frac{34+8n}{5} \right\rfloor$ bits de bourrage avec n égal à 8 octets au maximum.
Pour un débit de 1Mb/s lorsque la durée de transmission d'une trame comportant 8 octets de données est égale à 130µs, elle correspond à la taille maximale de la trame (130bits) c'est-à-dire à un nombre maximal de bits de bourrage (19).

Exemple

Soient trois sites S_1, S_2 et S_3 qui ont des messages de priorités respectives 111, 010 et 011 à émettre.
Durant la première tranche de l'intervalle de temps réservé à l'arbitrage de priorités, chaque site diffuse le bit le plus significatif (1 pour S_1, 0 pour S_2 et 0 pour S_3). Dans la figure 3.2, S_1 est éliminé car il envoie 1 et reçoit 0, ce qui veut dire qu'il y a au moins un site qui veut émettre un message plus prioritaire. Dans la figure 3.3, durant la deuxième tranche de temps S_2 et S_3 envoient 1. Le bit lu sur le bus étant identique à celui qui est émis, aucun des deux ne se retire et le processus continue avec les bits suivants. La figure 3.4 représente la troisième tranche de temps durant laquelle S_3 est éliminé et S_2 gagne l'accès au bus.

Autrement dit, le site qui a envoyé tous ses bits et qui, à chaque fois, a lu la même valeur de bit sur le bus que celle qu'il a émise déduit qu'il est le vainqueur. La priorité étant propre à chaque message et chaque site n'émettant qu'un seul message à la fois, le vainqueur est unique.

Notons que les messages qui arrivent durant une phase d'arbitrage de priorités ne seront considérés que durant la phase suivante.

Remarque

Si l'on choisit les identificateurs des paquets inversement proportionnels à leurs périodes en associant des identificateurs différents aux paquets de même période alors l'ordonnancement des messages à l'aide de CAN correspondra directement à l'algorithme d'ordonnancement des tâches RM car les identificateurs sont directement utilisés comme priorités des messages.

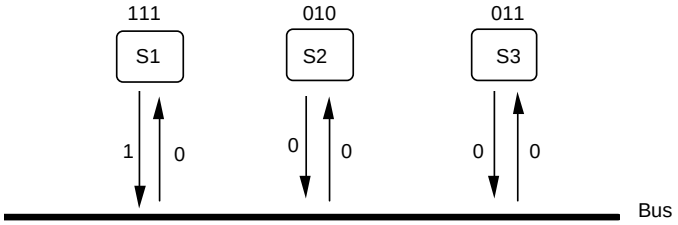


Figure 3.2: *Tous les sites envoient leur bit MSB*

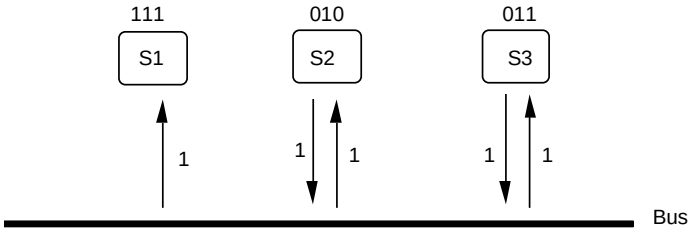


Figure 3.3: *S_1 se retire de la compétition en laissant S_2 et S_3 continuer le processus d'arbitrage de priorités*

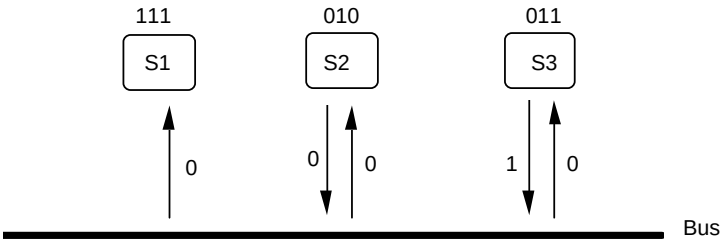


Figure 3.4: *S_3 émet 1 et reçoit 0, il se retire de la compétition, S_2 est donc le vainqueur*

4.2. Le protocole CSMA/DCR

4.2.1.Principe

Le protocole CSMA/DCR (Carrier Sense Multiple Access/ Deterministic Collision Resolution) reprend le principe de base de la méthode CSMA/CD normalisée [IEEE 85] et s'utilise sur un réseau composé de sites connectés sur un bus. Pour un débit de 10Mb/s, la durée de transmission d'un paquet C_p vaut 1228µs pour 1518 octets d'information utile (voir format de la trame dans l'annexe 3).
Utilisant le principe de compétition, chaque site est libre d'émettre à tout moment pourvu que le bus soit libre. Lorsqu'un site émet un paquet, il écoute le support en même temps qu'il émet. S'il ne détecte pas de collision durant la transmission, il conclut que le paquet est émis avec succès, sinon il attend un temps aléatoire avant de tenter une autre émission. Ce protocole est dit déterministe car il résout le problème de collisions en autorisant tous les sites à émettre dans un ordre défini grâce à un découpage dichotomique basé sur les adresses des sites. Ainsi pendant toute la phase de résolution de conflit, appelée *époque*, tous les sites vont pouvoir émettre dans un ordre fixé et avec un délai maximum connu suivant la position de chacun. Le groupe qui conserve le droit d'émettre est dit "gagnant" et l'autre "perdant". Le tableau 3.2 et la figure 3.5 illustrent bien cette situation à travers un exemple de 8 sites dont 5 sont émetteurs.
Dans le tableau 3.2, les sites 0, 1, 3, 6 et 7 émettent des paquets et détectent la collision.
La résolution de conflit étant basée sur une solution dichotomique, les sites 4 à 7 se retirent. Les sites restants (0 à 3) sont autorisés à émettre. Les sites 0, 1 et 3 émettent et la collision persiste. Le découpage se poursuit jusqu'à isoler un seul émetteur qui pourra émettre. Le processus est répété afin de permettre à chacun des sites en conflit d'envoyer son paquet.

Tableau 3.2 - *Délai d'émission du site 6 : cas quelconque*

temps / **N° site**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	C	C	C	E	6	6	6	6	6	6	6
1	C	C	C	6	E	6	6	6	6	6	6
2			6	6	6		6	6	6	6	6
3	C	C	6	6	6	E	6	6	6	6	6
4		6	6	6	6	6		V	6	6	6
5		6	6	6	6	6		V	6	6	6
6	C	6	6	6	6	6	C	6	C	E	6
7	C	6	6	6	6	6	C	6	C		E
Temps d'émission du site 6	<	-	-	-	-	-	-	-	-	-	>

Les notations sont : **C** : collision ; **E** : émission réussie ; **V** : pas d'émission ; 6 : pas d'autorisation d'émettre.

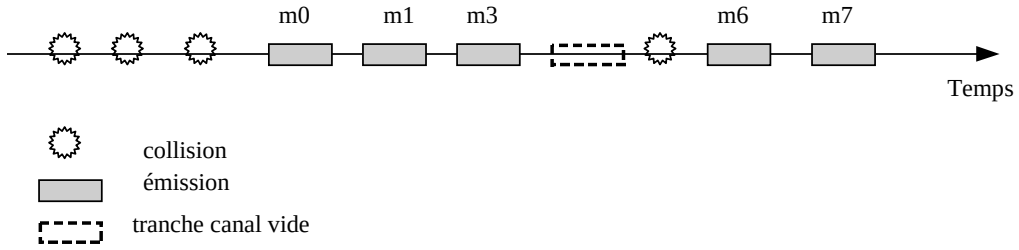


Figure 3.5: *La durée de l'époque dans le cas du tableau 3.2*

4.2.2. Calcul de l'époque

Si les n sites du réseau sont émetteurs, l'époque est égale à $nC_p + (n\text{-}1)T_c$ avec C_p étant la durée de transmission d'un paquet et T_c la tranche canal. Si seulement x sites ($x<n$) sont émetteurs alors l'époque se réduit à l'expression suivante selon [LR94]:

$$xC_p + \theta_x^n T_c \quad \text{avec} \quad \theta_x^n = \min\{n\text{-}1,\ x + [x\text{Log}(n/x)]\}.$$

T_c, la tranche canal, est égale à deux fois le temps de propagation, environ 50 µs. Si un groupe de sites n'a rien à émettre et a été sélectionné lors du découpage dichotomique alors le médium sera occupé par une phase canal vide (équivalente à une durée de T_c).

5. Protocoles d'accès à contrôle centralisé: exemple de FIP

FIP (*Factory Instrumentation Protocol*) [Tho 93, NFa 90] est un réseau de terrain qui assure le transfert d'informations entre les équipements (capteurs, actionneurs, régulateurs, unités de commande, etc.) reliés à un bus. Le support de transmission est une paire torsadée blindée ou une fibre optique. La vitesse de transmission 1Mb/s est la vitesse standard. Comme le montre le tableau 3.3, plus le débit est faible plus le temps de retournement est important. Le temps de retournement est le temps qui sépare la fin de réception d'une trame du début de l'émission de la trame suivante. La taille maximale d'un paquet est de 1069 bits (information utile et champs de contrôle) s'il est de type *RP_DAT* avec 128 octets comme information utile (voir format de la trame dans l'annexe 3).

Il existe deux types de service de transmission: la transmission de variables (service MPS: services périodiques/apériodiques industriels) et la transmission de messages (service MMS: service de messagerie).

Tableau 3.3 : *Débits et temps de retournement*

Débit	***tr*, Temps de retournement**
1Mb/s	$10\mu s \le tr \le 70\mu s$
2,5 Mb/s	$4\mu s \le tr \le 28\mu s$
31,25 Kb/s	$320\mu s \le tr \le 22{,}4ms$

Les variables de nature périodique ou apériodique représentent les données échangées pour le contrôle d'un procédé industriel. Les messages permettent de gérer la configuration du système de contrôle et sont de nature plutôt apériodique.
Selon la nature des données échangées, il existe deux types d'adressage:

- *Adressage de variables*: à une variable on associe un identificateur unique codé sur 16 bits. Pour un identificateur donné, il n'existe qu'un seul producteur et plusieurs consommateurs. Ces derniers doivent se reconnaître car leurs adresses ne figurent pas dans la trame.
- *Adressage de messages*: l'échange est multi-point ou point à point. Le message contient l'adresse de l'émetteur et sera accompagné de l'adresse du récepteur si l'échange est point à point.

L'ordonnancement des échanges de variables et de messages est défini et mis en œuvre dans un contrôleur central appelé *arbitre de bus*. Il consiste à construire, hors ligne, une table de scrutation qui sera suivie, en ligne, par l'arbitre de bus pour déterminer la variable (ou le message) à laquelle (ou auquel) il faut réserver le médium.
Les variables de la table de scrutation sont triées dans l'ordre de leur périodicité. En utilisant cette table, l'arbitre de bus diffuse l'identificateur de la variable à échanger qui indique au producteur d'émettre la valeur de la variable. Celle-ci sera copiée par tous les consommateurs concernés : c'est le modèle producteur-distributeur-consommateurs. S'il reste du temps libre, FIP échange des variables apériodiques et/ou des messages.
Certains sites jouent le rôle d'arbitre de bus de secours mais un seul arbitre de bus est actif dans le réseau afin d'avoir une vue globale du système et gérer les variables et messages de manière centralisée. La duplication de cette fonction constitue un mécanisme de tolérance aux pannes.

Exemple

Soient les 6 variables périodiques du tableau 3.4 scrutées par l'arbitre de bus [Let 92] où la somme des temps de transfert des variables est égal à 1,35ms.

Une répartition possible de ces variables en fonction des périodicités associées est donnée dans la figure 3.6 où un macrocycle est une juxtaposition de cycles élémentaires et où un cycle élémentaire est une tranche de temps égale au PGCD des périodes des variables périodiques. Le macrocycle est égal au PPCM des périodes.

Dans le cas d'un trafic périodique, l'arbitre de bus envoie des demandes de transfert *ID_DAT* pour chaque variable dans le réseau. Le producteur *P* (voir figure 3.7) répondra par la diffusion de cette variable, à l'aide de la trame *RP_DAT* qui sera lue par les consommateurs *C*. Lorsque l'arbitre de bus reçoit *RP_DAT*, il diffuse une nouvelle trame *ID_DAT* afin de permettre la mise à jour d'une nouvelle variable ; et ainsi de suite pour toutes les variables.

Tableau 3.4 : *Exemple de table de scrutation*

Variable	Périodicité	Type de variable	Temps de transfert[3] de la variable (µs)
A	5	INT_8	154
B	10	INT_16	162
C	15	OSTR_32	402
D	20	SFPOINT	178
E	20	UNS_32	178
F	30	VSTR_16	274

Après scrutation des variables périodiques, l'arbitre de bus traite des demandes de transfert apériodique. S'il n'y a pas de demande, il émet des identificateurs de bourrage jusqu'à la fin du cycle élémentaire pour indiquer aux autres sites que le réseau fonctionne toujours. Un identificateur de bourrage est un identificateur que ne produit aucun site. Le trafic apériodique et la messagerie ne font pas l'objet de notre étude. Le lecteur intéressé pourra consulter [Tho 93, SC95].

[Son 96] modélise la couche MAC du réseau FIP à l'aide de TDMA et montre comment FIP peut gérer, dans ce cas, non seulement des messages périodiques à contraintes strictes mais aussi des messages apériodiques à contraintes relatives. Ce résultat peut être intéressant lorsque nous savons que la configuration commercialisée de FIP supporte uniquement un trafic périodique hors ligne.

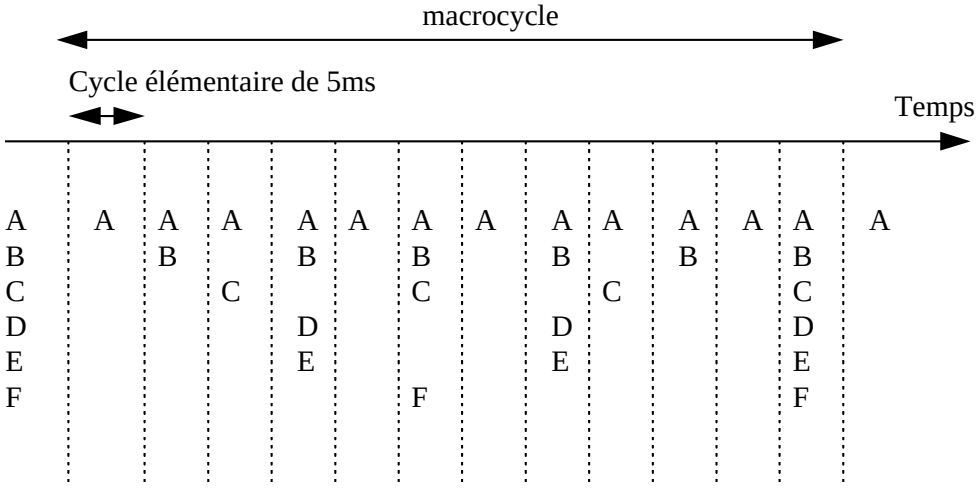


Figure 3.6: *Transfert de variables durant un macrocycle*

[3] Le temps de transfert d'une variable est la somme du temps d'envoi de la requête, du temps de réception de la réponse et du temps de retournement multiplié par deux

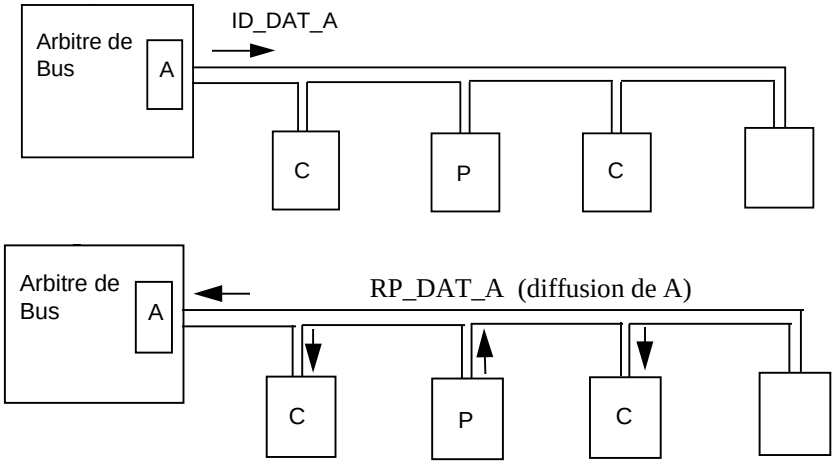


Figure 3.7: *Transfert de variables périodiques*

6. Protocoles d'accès à contrôle distribué

6.1. Le protocole FDDI

Le réseau FDDI (*Fiber Distributed Data Interface*) est un réseau à structure de boucle ou anneau, qui utilise une fibre optique multimode comme support de transmission. La technique d'accès est dite à jeton temporisé.

FDDI est un réseau à contrôle d'accès par jeton proche de la norme "anneau à jeton 802.5" [IEEE 802.5]. Il est basé sur la considération de deux types de trafic: le trafic *synchrone* soumis à des contraintes temporelles et le trafic *asynchrone* qui désigne un trafic non contraint temporellement. Il présente des performances élevées: longueur maximale entre les sites les plus éloignées : 200 km, nombre de sites ou connections sur le réseau : 1000, débit nominal : 100 Mbit/s. Le délai maximal de propagation sur l'anneau du signal est fixé par la norme à 1,677 ms. La durée de transmission d'un paquet est de 360µs pour 4490 octets d'information utile au maximum (voir format de la trame dans l'annexe 3).

Le trafic de base est le trafic synchrone et le principe du protocole est d'allouer à chaque site une durée pré-définie qui représente le temps maximal pendant lequel il peut transmettre du trafic synchrone chaque fois qu'il reçoit le jeton. Le protocole de gestion du jeton repose sur une durée déterminée à partir du temps de rotation du jeton. Le temps maximal que peut mettre le jeton pour faire un tour de l'anneau est un paramètre du réseau appelé *TTRT* (*Target Token Rotation Time*) fixé au démarrage du réseau. Cette valeur est utilisée pour charger un temporisateur, appelé *TRT* (*Token Rotation Timer*), qui contrôle la saisie du jeton pour la transmission des trames en attente. Chaque site *i* dispose d'une fraction α_i de la transmission exprimée en pourcentage du *TTRT*. La somme des allocations de tous les sites du réseau est inférieure à 100%.

A chaque fois qu'un site reçoit le jeton, il peut émettre pendant un temps $H_i = \alpha_i TTRT$ des trames synchrones. Les trames asynchrones ne peuvent être envoyées que lorsque le site qui reçoit le jeton voit celui-ci arriver en avance, c'est-à-dire si le jeton lui revient au bout d'un temps inférieur à *TTRT*. Le temps alloué à l'émission asynchrone est donné par un compteur *THT* (*Token Hold Timer*):*THT = TTRT - TRT*. En utilisant le paramètre *AT* (arrivée tardive), la gestion de ce réseau est faite à partir des deux algorithmes décrits ci-après conduisant à un fonctionnement représenté sur la figure 3.8**.**

A l'obtention du jeton

THT :=*TRT*
Si *AT* = 0 **alors**
    TRT := *TTRT*
Finsi
émission synchrone
Si *AT* = 0 et *THT* > 0 **alors**
    émission asynchrone pendant *THT*
Finsi
AT := 0

Arrivée tardive du jeton (détectée)
Si *TRT* = 0 **alors**
    AT := 1
    TRT := *TTRT*
Finsi

L'exemple de la figure 3.8 suppose que *TTRT*=100µs et que le site considéré dispose d'une bande passante synchrone *H*=25%*TTRT*=25µs.
Pour un site donné, seul le trafic synchrone (de durée $H_i = \alpha_i TTRT$) est certain d'être transmis et le trafic asynchrone (de durée *THT*) n'est transmis que si le jeton tourne en moins de *TTRT*.

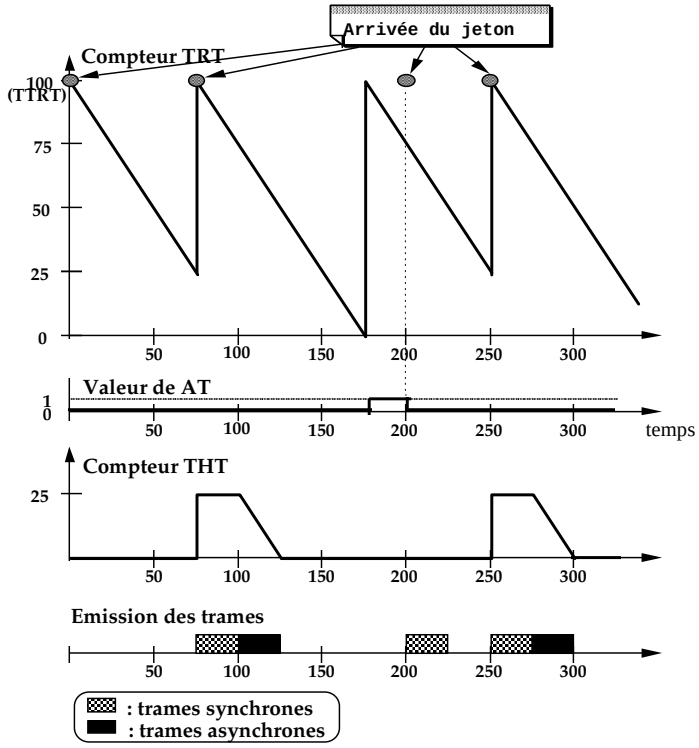


Figure 3.8 - *Exemple de fonctionnement d'un réseau FDDI : trames synchrones et asynchrones*

Le protocole, tel qu'il est décrit, maintient un temps entre deux visites consécutives d'un site par le jeton inférieur à 2**TTRT* si le trafic asynchrone est prévu et est inférieur à *TTRT* si le mode est uniquement synchrone [SJ 87]. [ACZD 92] généralisent ce résultat

à v (v≥2) visites consécutives d'un site bornées par $v.TTRT$, davantage affiné dans [ZB 95].

Le choix de la bande passante

Garantir les échéances des messages synchrones dépend principalement d'une allocation appropriée de la bande passante synchrone. Pour cela, plusieurs schémas d'allocation de la bande passante ont été décrits dans [ACZD 92]:

- *allocation complète de la longueur* pour un site i: $H_i=C_i$ avec H_i: bande synchrone et C_i la durée de transmission d'un message synchrone.
- *allocation de partitions égales*: $H_i = \frac{TTRT - \partial}{n}$ avec $TTRT$: paramètre du système, ∂ le délai de propagation et n le nombre total de sites.
- *allocation proportionnelle*: $H_i = \frac{C_i}{P_i}(TTRT - \partial)$, la bande synchrone est proportionnelle au taux de la longueur sur la période.
- *allocation proportionnelle normalisée*: $H_i = \frac{C_i / P_i}{U_s}(TTRT - \partial)$ avec $U_s = \sum_{i=1}^{n} \frac{C_i}{P_i}$ on parle de normalisation par rapport au facteur d'utilisation des messages synchrones.
- *allocation locale* : $H_i = \frac{C_i}{\lfloor P_i / TTRT \rfloor - 1}$, le jeton visitera le site i au moins $\lfloor P_i / TTRT \rfloor - 1$ fois durant une période P_i.

Remarquons que la deuxième technique d'allocation qui consiste à allouer équitablement le médium n'est pas réaliste dans le sens où les sites émettent des messages avec des tailles et des périodes différentes c'est-à-dire qu'ils ont des besoins différents en termes de bande passante.

[ZS95] proposent un schéma optimal pour l'allocation de la bande passante synchrone en allouant et en désallouant la bande passante synchrone à un site sans toutefois changer celles assignées aux autres sites.

6.2. Le protocole TDMA

Le réseau TDMA (*Time Division Multiple Access*) est un réseau en structure d'anneau qui possède un générateur de trames. Les trames sont allouées aux différents sites de manière cyclique. Pour un débit de 1 Mb/s, la durée de transmission d'un paquet est de 100 µs et la durée d'un cycle est de 16 tranches de temps.

Un site ne peut émettre que dans la trame qui lui est allouée. Lorsque la trame a fait un tour, elle est purgée par le générateur.

Il existe trois politiques différentes d'allocation:

- une trame par site et par cycle,
- plusieurs trames contiguës par site et par cycle,
- plusieurs trames non contiguës par site et par cycle.

Entre deux cycles consécutifs, il existe un délimiteur de trames (voir figure 3.9 et format de la trame dans l'annexe 3) qui aide l'émetteur et le récepteur à se synchroniser.
Une implémentation de ce protocole est le protocole TTP [KG 94] défini dans le cadre du système d'exploitation MARS. Dans ce protocole, le temps global est divisé en une séquence de tranches de temps. Chaque tranche de temps est pré-assignée à un site.

[SS 96a] ont étudié les performances du réseau TDMA en considérant le trafic périodique et le trafic apériodique qui arrive suivant une loi poissonienne, tout en supposant un tampon à *K* places au niveau de l'émetteur, *K* étant différent du nombre *b* de tranches de temps allouées. Ils montrent comment le choix des paramètres *K* et *b* peut être pertinent pour obtenir de bonnes performances.

7. Conclusion

Une classification des différents réseaux à accès multiple a été présentée et pour chaque classe, un réseau de communication a été décrit. Les caractéristiques principales des réseaux étudiés sont récapitulées dans le tableau 3.11. Il n'y a pas d'algorithmes d'adaptation pour la prise en compte des contraintes de temps des messages. Seuls les protocoles de communication sont utilisés dans la méthodologie proposée au chapitre 5 qui, dans ce cas, doivent fournir un délai borné pour l'accès au médium ; ce qui a motivé le choix des réseaux à étudier dans ce chapitre.

Le chapitre suivant présente, d'un côté, quelques exécutifs temps réel répartis à travers leurs configurations matérielle et logicielle afin de montrer le type d'environnement qui pourrait supporter les applications temps réel réparties que nous avons décrites. D'un autre côté, il montre ce qui existe en matière d'outils et de méthodes de validation temporelle d'applications temps réel réparties.

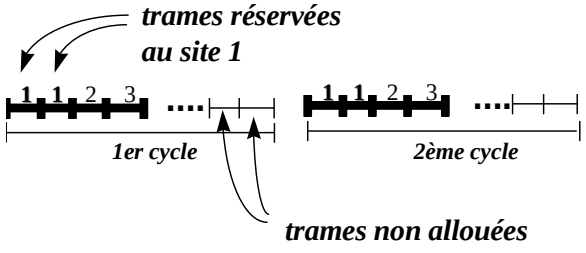


Figure 3.9: *Les cycles de communication dans TDMA*

Tableau 3.11 : *Les caractéristiques générales des réseaux à accès multiple*

Protocoles à contrôle centralisé ex: *FIP*	Réservation statique ex: *TDMA*	Protocoles à compétition ex: *CSMA/CD, CSMA/DCR*	Protocoles à jeton ex: *Token ring, Token bus, FDDI*
Gestion centralisée: l'arbitre de bus est une entité unique qui gère l'accès au bus.	Gestion centralisée: une unité génératrice de trames affecte initialement une ou plusieurs tranches de temps à chaque station.	Gestion complètement décentralisée pour l'accès au canal (bus).	le jeton est une marque unique qui autorise ou non une station à émettre.
L'arbitre de bus invite les stations à émettre (attente de sélection)	une station peut émettre dans la tranche de temps qui lui est réservée.	chaque station est libre d'émettre à tout moment: l'accès est permanent au canal (pas d'attente de sélection).	seul le détenteur du jeton peut émettre.
Architecture type bus	architecture type boucle simple ou double	architecture type bus	architecture anneau (physique ou logique, simple ou double)
connexion multi-point (mode d'échange par diffusion)	connexion point à point	connexion multi-point	connexion point à point pour le jeton et connexion point à point ou multi-point pour le transfert des données
pas de collisions	pas de collisions	collisions	pas de collisions
pas de retransmission: les données sont ignorées	possibilité de retransmission dans les tranches de temps réservées	bonne gestion des erreurs de communication: plusieurs stratégies de retransmission existent pour la résolution des conflits	procédures de régénération de jeton (en cas de perte) et de destruction (en cas de duplication). La couche liaison ne gère pas les retransmissions des données: c'est le rôle des couches supérieures

la notion de priorité n'est pas intégrée	on peut définir des priorités statiques au départ ayant pour conséquence l'allocation de plus de tranches de temps aux stations prioritaires	possibilité de définir des priorités statiques ou dynamiques pour les stations (ex: *CSMA/DCR*)	un mécanisme de priorités est défini pour allouer le support de communication aux trames prioritaires (IEEE 802.5)
FIP est statique: pas de retrait ni d'insertion dynamiques de stations	*TDMA* est statique	l'insertion ou le retrait d'un équipement n'a aucune incidence sur le protocole	l'ajout et le retrait de stations se fait à l'aide de procédures particulières (mais la détection de panne et la reconfigu--ration de l'anneau sont automatiques dans *FDDI*)
se prête bien au trafic périodique et apériodique	se prête bien au transport de petits messages répétitifs	*CSMA/DCR* peut fonctionner en mode général (=*CSMA/CD* en faible charge) et en mode périodique (=*TDMA* à forte charge).	*FDDI* offre un service de transfert synchrone pour le trafic périodique et un service de transfert asynchrone pour le trafic aléatoire

CHAPITRE 4

ENVIRONNEMENTS TEMPS REEL REPARTIS

1. Introduction

Ce chapitre se compose de deux parties. Dans la première, nous examinons quelques exemples d'exécutifs temps réel répartis à travers les concepts de base qu'ils mettent en oeuvre et leurs stratégies d'ordonnancement et de communication. Cette étude a pour but de déterminer le type d'applications temps réel réparties, en termes d'architectures matérielle et logicielle, que nous pouvons analyser temporellement à l'aide de notre méthodologie. Nous présentons donc les trois systèmes les plus connus : MARS, SPRING et CHORUS.

La deuxième partie est consacrée, quant à elle, aux outils et méthodes de validation d'applications temps réel réparties. Nous commençons par la description de PERTS qui est un outil d'aide à la validation d'applications temps réel (réparties ou non), basé sur une analyse d'ordonnançabilité. PERTS partage la même philosophie que notre méthodologie de validation. Nous développons ensuite une méthode d'analyse d'ordonnançabilité d'applications temps réel réparties, qui utilise FDDI comme réseau de communication.

2. Quelques exécutifs temps réel répartis

Jusqu'à présent, nous avons décrit les applications temps réel répartis en termes de tâches et de messages la constituant sans toutefois connaître les environnements temps réel qui peuvent supporter ce type d'applications. Le but de ce paragraphe est de présenter quelques exécutifs répartis connus dans le domaine du temps réel.

2.1. Le projet MARS

Le projet MAintainable Real-time System (MARS) [Kop 86, DRSK 89] a été développé à Technische Universitat Berlin. Démarré en 1980, le premier prototype est né en 1984 mettant en œuvre les concepts fondamentaux de MARS. Un second prototype développé à l'université de Vienne a été fonctionnel dès 1988. Il apporte une solution au problème des systèmes temps réel distribués à contraintes strictes, tolérant aux fautes.

2.1.1. Architecture matérielle de MARS

L'architecture du système est définie par des éléments de base appelés *grappes* (en anglais, *clusters*). Chaque grappe consiste en un ensemble de composants connectés par un bus de communication temps réel synchrone appelé *bus MARS*. Un composant n'est rien d'autre qu'un ordinateur indépendant sur lequel s'exécutent des tâches temps réel et une copie du noyau temps réel. Le concept de grappe a été introduit afin de gérer les réseaux complexes de composants (voir figure 4.1).

2.1.2. Le système d'exploitation de MARS

Une copie du système d'exploitation est installée sur chaque composant et s'exécute localement de manière autonome. Le système d'exploitation consiste en un petit noyau et un ensemble de tâches système (voir figure 4.2).

Trois fonctions principales sont réalisées par le noyau:
* l'ordonnancement des tâches à contraintes strictes,
* le maintien d'une horloge temps réel globale et
* la gestion de messages.

2.1.2.1. L'ordonnancement des tâches

Les tâches du système sont de deux catégories:

- *Les tâches temps réel critiques*: qui sont des tâches cycliques pouvant recevoir, traiter et émettre des messages dans des délais stricts. Elles peuvent être des tâches système ou des tâches application.

- *Les tâches temps réel non critiques*: qui sont généralement des tâches acycliques dont l'exécution est effectuée pendant les périodes où le processeur n'a pas de tâche critique à exécuter.

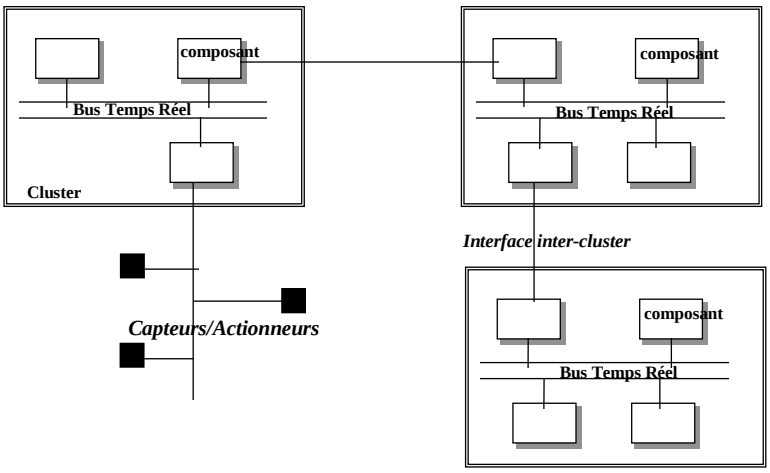


Figure 4.1: *L'architecture du système MARS*

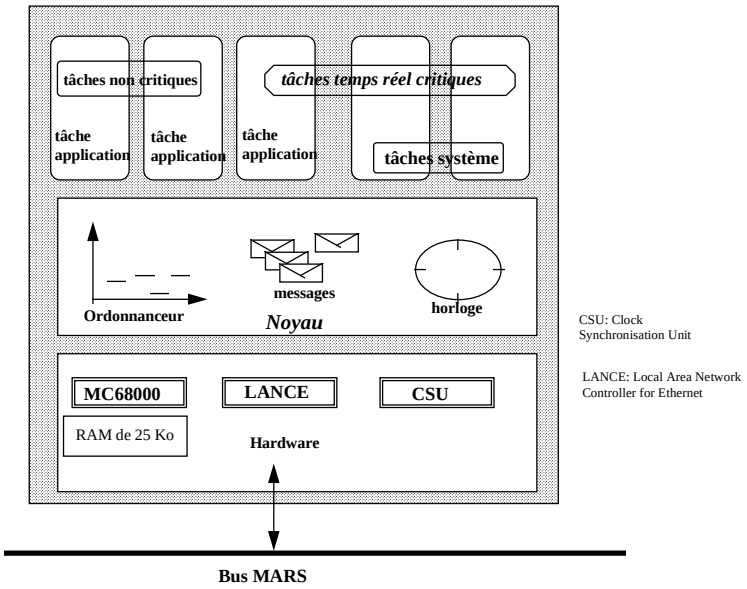


Figure 4.2: *Structure d'un composant MARS*

Pour la prise en compte des interruptions, seule l'horloge temps réel peut interrompre le processeur. Périodiquement, la routine de traitement de l'interruption horloge teste l'état de tous les périphériques afin de prendre en compte les événements sporadiques. Les événements détectés durant un cycle d'ordonnancement seront traités durant le prochain cycle de manière à ce que l'ensemble des tâches à ordonnancer soit statique et connu à l'avance.

D'un point de vue temporel, toute tâche périodique est caractérisée par une période d'activation, une durée maximale d'exécution et le composant où elle s'exécute. Le concepteur doit spécifier tous ces attributs lors de la description de l'application.

Pour la gestion de la communication, un réseau du type TDMA est utilisé où des tranches de temps (en anglais, *slots*) sont assignées aux composants selon une stratégie d'ordonnancement du type « Tourniquet ».

Une transaction est une séquence de tâches (généralement réparties sur différents composants) déclenchée suite à un stimulus et qui produit une réponse. La contrainte temporelle de bout en bout d'une telle transaction s'exprime par son *temps de réponse maximal* (Mart) qui comprend :

- le temps de latence séparant le lancement de la première tâche qui réagit à ce stimulus,
- les durées d'exécution des tâches impliquées,
- les temps de communication entre ces tâches,
- le temps séparant la fin de la dernière tâche de la réponse.

L'ordonnanceur dispose en entrée des durées maximales d'exécution, du nombre maximal de ressources nécessaires pour chaque tâche, des tâches critiques de chaque composant (site) et du temps maximal séparant deux activations successives de chaque transaction. Le résultat est une séquence d'ordonnancement mettant en évidence les tâches critiques des différents sites avec leurs instants d'activation, les instants de préemption et les périodes d'activation de chaque tâche. L'ordonnanceur hors ligne est basé sur une stratégie de parcours en profondeur [Ric 83]. En effet, une fonction

heuristique est utilisée pour construire un arbre en recherchant les délais les plus courts pour les tâches qui font partie d'une même transaction afin de respecter le temps de réponse maximal de la transaction. Un chemin, dans l'arbre, correspond à une séquence d'ordonnancement dont le coût n'excède pas Mart. Chaque nœud *N* de l'arbre correspond à une tâche vérifiant la fonction f(N)=g(*N*)+h(*N*) où g(*N*) est le coût du chemin issu de la racine à *N* et h(*N*) est une estimation du coût du chemin « le moins cher » depuis *N* jusqu'à une solution. h(*N*) est la fonction heuristique qui doit tenir compte des durées maximales des tâches, de leurs durées de communication et de la période d'inactivité du processeur.

2.1.2.2. *Gestion du temps*

Chaque composant comporte une horloge temps réel de résolution 1 µs appelée CSU (Clock Synchronisation Unit). La synchronisation d'horloge consiste :

- d'une part, en une synchronisation interne qui consiste à garder les horloges de tous les composants d'une même grappe décalées au maximum d'une constante connue (10 µs),

- d'autre part, en une synchronisation externe qui ajuste les horloges de la grappe avec le temps physique.

La synchronisation interne est un recalage d'horloges qui se fait grâce à des messages estampillés échangés périodiquement.

La synchronisation externe se fait grâce à un composant particulier qui a accès au temps réel et qui mesure la différence entre ce temps et le temps maintenu dans la grappe. Une tâche spéciale diffuse alors la correction à porter sur les horloges locales indépendamment de la synchronisation interne.

2.1.2.3. L'échange de messages

Deux classes de messages sont échangés dans MARS:

- Les messages non critiques qui sont émis dans des tranches de temps inutilisées. Ils servent à transmettre des données asynchrones notamment des données « archive ».
- Les messages périodiques critiques véhiculent des informations d'état nécessaires au bon fonctionnement des systèmes d'exploitation sur les différents composants, *appelés messages d'état*. Ce sont des messages datagramme comportant une date de validité exprimée dans un référentiel temporel global. Ils sont ordonnancés de manière statique en affectant un message à chaque tranche de temps de manière à ce qu'il ne soit émis que dans la tranche de temps qui lui est réservée.

2.1.3. Conclusion

On note quelques particularités de MARS:

- les redondances physique (duplication des composants qui exécutent les mêmes tâches) et logique (transmission multiple des messages d'état) constituent un mécanisme de tolérance aux pannes,

- le comportement de MARS est déterministe: les tâches périodiques critiques ainsi que leurs caractéristiques temporelles sont connues à l'avance. Elles sont ordonnancées hors ligne en assignant des tranches de temps aux tâches émettrices de messages avant de déclencher l'ordonnanceur en ligne. Les événements aléatoires sont détectés périodiquement par une procédure de traitement d'interruption horloge et sont pris en compte dans le prochain cycle d'ordonnancement statique. Aucune dynamicité n'est permise.

- les contraintes temporelles, de précédence et d'accès aux ressources sont prises en compte grâce à l'algorithme d'ordonnancement hors ligne qui explore tous les chemins possibles dans un arbre de recherche comportant toutes les combinaisons (ordonnancements) possibles afin de construire une séquence d'exécution. Notons que l'algorithme de recherche utilisé est assez bien connu dans le domaine de l'intelligence artificielle et a pour inconvénient majeur son explosion combinatoire ; ce qui peut donner lieu à un temps de réponse non acceptable dans un domaine tel que celui du temps réel.

2.2. Le projet SPRING

Le développement du système Spring [SR 91] a été motivé par le fait que les systèmes temps réel existants ne sont rien d'autres qu'une version optimisée des systèmes d'exploitation en temps partagé. Le nouveau paradigme de Spring est de connaître *a priori* le comportement des tâches et leurs caractéristiques afin de garantir les échéances pour toutes les tâches à contraintes strictes et de minimiser le retard pour les autres tâches.

2.2.1.Classification et caractérisation des tâches

Spring utilise trois types de tâches:

- *les tâches critiques* correspondent à des tâches qui doivent respecter leurs échéances. Dès la phase de conception, il y a lieu de vérifier que ces tâches respectent leurs échéances dans une situation de charge maximale car dans le cas contraire, une catastrophe peut survenir.

- *les tâches essentielles* correspondent aux tâches nécessaires au bon fonctionnement du système. Elles possèdent des contraintes temporelles qui peuvent provoquer une dégradation du système si elles ne sont pas respectées sans que cela ne conduise à une catastrophe. Ce type de tâches sont traitées en ligne dans cette approche.

- *les tâches non essentielles* peuvent avoir ou non des contraintes temporelles. Elles s'exécutent lorsque les tâches critiques et essentielles n'ont plus besoin du processeur.

Le gestionnaire de tâches supporte des *tâches* et des *groupes de tâches*. Un groupe de tâches est un ensemble de tâches ayant des relations de précédence les unes avec les autres et partageant un délai critique de groupe (délai critique de bout en bout). A la

date d'activation, le compilateur est supposé capable de déterminer la durée maximale d'exécution et les besoins en ressources pour chaque tâche du groupe.

2.2.2.Architecture de SPRING: Springnet

Springnet est un système distribué composé d'un ensemble de multiprocesseurs reliés à un réseau de communication du type CSMA/CD. Chaque multiprocesseur contient des *processeurs application*, des *processeurs système* et un *sous-système d'E/S* (voir figure 4.3).

Les processeurs application exécutent des tâches application de haut niveau qu'il faut garantir c'est-à-dire dont il faut respecter les échéances.

Le processeur système se charge de l'ordonnancement et de l'exécution des tâches système afin de ne pas perturber les tâches application dont il faut à tout prix garantir l'exécution.

Le sous-système d'E/S est une partie du noyau qui gère les E/S non critiques et/ ou lentes et les capteurs rapides.

Les fonctions du noyau sont: la gestion des tâches, l'ordonnancement, la gestion mémoire et la communication entre tâches.

2.2.3.L'ordonnancement des tâches

L'ordonnancement est une partie intégrante du noyau. Il est composé de 4 niveaux, le premier s'exécute sur le processeur application et les autres sur le processeur système.

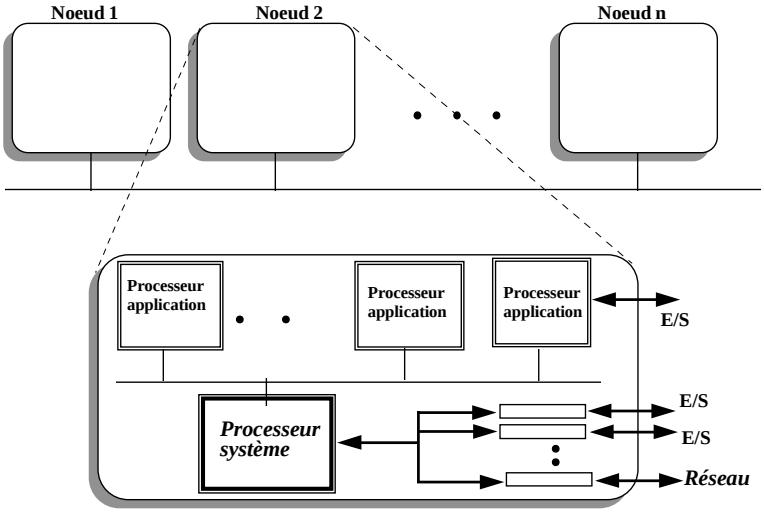


Figure 4.3: *Le système distribué Springnet*

- au niveau le plus bas, il y a plusieurs répartiteurs: un type de répartiteur s'exécute sur les processeurs application et un autre type sur les processeurs

système. Le répartiteur des processeurs application se contente d'activer la tâche prête en tête de file. La file contient les tâches à garantir dans un ordre approprié pour chaque processeur application. Le répartiteur du processeur système permet l'exécution périodique des tâches système et l'exécution des tâches sporadiques lorsque ces dernières n'influent pas sur les tâches à garantir.

- un ordonnanceur local de niveau 2 qui tente, grâce à une routine de garantie, de voir si l'ajout d'une tâche ou d'un groupe de tâches ne remet pas en cause le respect des contraintes temporelles, étant donné une liste de tâches que l'on est sûr de garantir. L'ordonnanceur n'ordonnance pas uniquement les tâches essentielles mais aussi les non essentielles dans les tranches de temps libres. Un arbre de recherche est utilisé pour construire des séquences d'ordonnancement valides. Au fur et à mesure que les tâches sont insérées dans l'arbre en construisant un ordonnancement partiel, les tâches sont insérées dans la file des tâches prêtes selon l'ordre croissant de leurs délais critiques. Pour rajouter une tâche dans l'arbre, l'algorithme doit déterminer si l'ordonnancement étendu avec n'importe quelle tâche parmi *N* tâches en attente d'ordonnancement reste valide. Si tel est le cas, une fonction heuristique est évaluée pour chacune des *N* tâches. La tâche de plus petite valeur heuristique est choisie pour étendre l'ordonnancement courant. La fonction heuristique utilisée est Min_D+Min_S qui permet de sélectionner la tâche dont l'échéance (grâce à Min_D) et le début d'exécution (grâce à Min_S) sont les plus proches.

- un ordonnanceur distribué de niveau 3 qui essaie de trouver un site sur lequel pourrait s'exécuter une tâche ou un groupe de tâches lorsqu'il est impossible de la (ou le) garantir localement. La stratégie de migration est basée sur l'algorithme dit *souple*, de [Sta 89], qui consiste à envoyer la tâche au site le moins chargé appelé *hôte* à la connaissance du site demandeur et diffuse en même temps des requêtes, avec l'adresse du site hôte, aux différents sites afin qu'ils répondent au site hôte en lui envoyant leurs charges. Si ce dernier ne peut pas garantir la tâche, il peut la propager au site qui a la charge la plus petite selon les réponses reçues.

2.2.4.Conclusion

Garantir les contraintes temporelles est le point central de cette approche qui distingue deux cas:

- la garantie *a priori* des tâches critiques,

- la garantie en ligne des tâches essentielles.

Les avantages d'un tel ordonnancement sont:

- ♦ le système a une vue globale de l'ensemble des tâches, c'est-à-dire qu'il connaît exactement les tâches qui ont été garanties, la séquence des tâches à garantir et les tâches qui s'exécutent sans contraintes temporelles.

- ♦ il n'y a pas de conflit pour l'obtention de ressources. Les tâches qui demandent la même ressource sont ordonnancées à des instants différents.

- ♦ Le répartiteur (implanté sur le processeur application) et la routine de garantie (implantée sur le processeur système) s'exécutent en parallèle. Le répartiteur travaille sur un ensemble de tâches que l'on sait pouvoir garantir alors que la routine de garantie travaille sur un ensemble composé de l'ensemble courant de tâches (que l'on est sûr de garantir) et d'une tâche qui vient d'arriver.

- ♦ Si la routine de garantie conclut que la tâche qui vient d'être invoquée ne peut être garantie, celle-ci peut recevoir des cycles oisifs au niveau du processeur et en parallèle être affectée, grâce à l'ordonnanceur distribué, à un autre site susceptible de la garantir.

Cette approche est très différente de celle de Mars dans le sens où l'ordonnancement est dynamique ; toutefois l'utilisation d'un réseau CSMA/CD ne permet pas de borner les délais de transfert ; ce qui est contraignant pour des applications réparties qui nécessitent des communications temps réel.

2.3. Le système CHORUS

Chorus est un système d'exploitation réparti [Roz 91, Cho 92, Zim 81 *et al.*] basé sur une technologie micro-noyaux. Il a été conçu par l'équipe Chorus de l'INRIA de 1980 à 1988 et est actuellement commercialisé par la société Chorus Systèmes.

Il se présente sous la forme d'un exécutif de petite taille, écrit en C pour faciliter sa portabilité et enrichi par un ensemble de serveurs système. Chorus est disponible sur des architectures multiprocesseurs faiblement couplées (hypercube par exemple) et récemment sur des architectures fortement couplées (à architecture symétrique).

Le micro-noyau est le composant minimal présent sur chaque site et réalisant essentiellement:

- le contrôle d'exécution des processus,
- la gestion des communications entre processus distants et locaux,
- la gestion de la mémoire virtuelle.

Des modules externes au noyau s'exécutent sous la forme de serveurs locaux ou distants. Ces serveurs qui composent le système d'exploitation réparti sont spécialisés (gestion de fichiers, gestion de processus, gestion de noms, ...) et coopèrent en utilisant les mécanismes de coopération offerts par le noyau.

2.3.1. Les objets dans CHORUS

Le noyau manipule un certain nombre d'objets élémentaires définis ci-dessous:

2.3.1.1. L'acteur

L'acteur (*actor*) est un processus Unix sans la partie exécution qui, elle, se trouve dans les sous-processus (processus légers). Il existe deux types d'acteurs: *les acteurs utilisateurs* et *les acteurs superviseurs*. Chaque acteur utilisateur a son propre espace

d'adressage virtuel. L'ensemble des acteurs superviseurs partagent le même espace d'adressage virtuel que le noyau. Ces derniers requièrent une attention particulière dans leur développement car ils peuvent accéder à toutes les données du noyau.

2.3.1.2. L'activité

L'activité (*thread*) est l'unité d'exécution ou processus léger séquentiel. Une activité est attachée à un acteur qui définit son environnement d'exécution. Les activités d'un même acteur peuvent partager l'environnement offert par ce dernier, c'est-à-dire le même espace d'adressage et les mêmes ressources, leur permettant ainsi de se synchroniser (à l'aide de sémaphores) et de communiquer entre elles.

Chaque activité a son propre environnement d'exécution: pile, compteur ordinal et registres. La priorité d'une activité dépend de sa propre priorité et de celle de l'acteur dont elle dépend. Les activités sont les unités d'ordonnancement du noyau qui les gère en fonction de leurs priorités: le noyau garantit l'exécution de l'activité la plus prioritaire.

2.3.1.3. La Porte

Le noyau offre un mécanisme de communication uniforme indépendant de la localisation des activités: l'IPC (*InterProcess Communication*). La communication consiste en un échange de messages entre activités d'acteurs différents à travers les portes de ces derniers quelle que soit leur localisation (acteurs sur le même site ou sur des sites différents, voir figure 4.4).

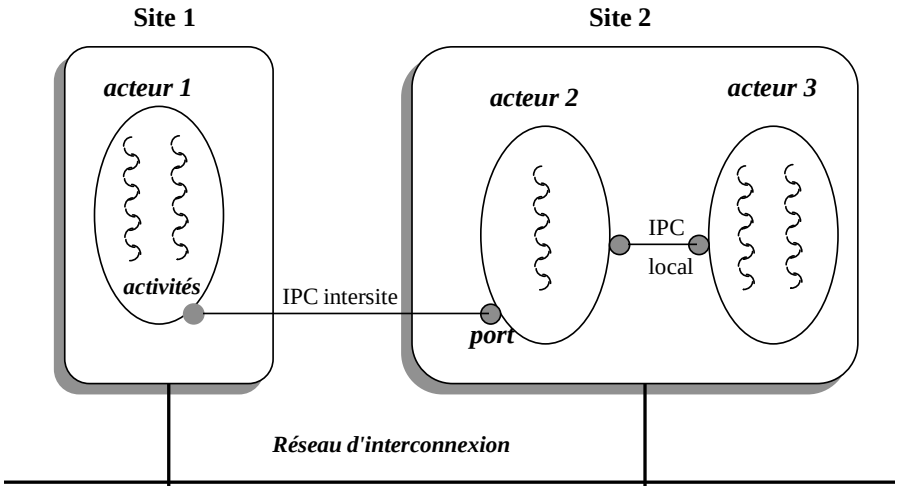


Figure 4.4: *Le mécanisme IPC*

2.3.2.L'ordonnancement des tâches

L'ordonnanceur associe à chaque activité une priorité fixe qui est un entier compris entre 0 et 255. Si sa priorité est inférieure à 127, l'activité est considérée comme un processus temps réel ; si elle est supérieure, c'est un processus non temps réel et donc sans contrainte temporelle. L'ordonnanceur assure qu'à tout moment la tâche prête activée sur un processeur est celle de priorité la plus petite.

Le traitement des demandes d'interruption est réalisé par des gestionnaires d'interruption qui s'exécutent comme des tâches superviseur. Comme plusieurs événements peuvent être associés à un même niveau d'interruption, plusieurs gestionnaires sont rattachés à cette interruption, chacun correspondant à une priorité.

Ainsi, l'ordonnanceur est fondé sur l'emploi de priorités fixes attribuées en général de manière empirique et ne tient pas compte de l'urgence d'une tâche qui est fonction du délai critique. Des travaux ont été entrepris sur CHORUS par [KS 98] en vue d'introduire un nouvel ordonnanceur du type *Earliest Deadline* et d'un mécanisme de stabilisation appelé *régisseur* tel qu'il a été décrit par J. Delacroix dans [Del 94].

L'ordonnanceur à échéance ordonnance les tâches de façon à ce que la tâche élue soit celle de plus proche échéance parmi l'ensemble des tâches prêtes.

Le régisseur est une unité de contrôle locale au site, qui ne s'exécute que lorsque le système est surchargé. Il intervient pour privilégier les tâches de plus haute importance vis à vis de l'application. Un algorithme réparti a été développé sur CHORUS par [KS 98] appelé *gestionnaire d'importance réparti* (voir figure 4.5) en vue de dialoguer avec les autres sites et de déterminer le site qui se surcharge le moins en acceptant l'exécution d'une tâche qui ne peut être acceptée localement.

Le noyau offre un ensemble de services de base et plusieurs versions sont offertes: système orienté temps réel (CHORUS/Mix, CHORUS/Fusion, CHORUS/ClassiX), système orienté objets (CHORUS/COOL).

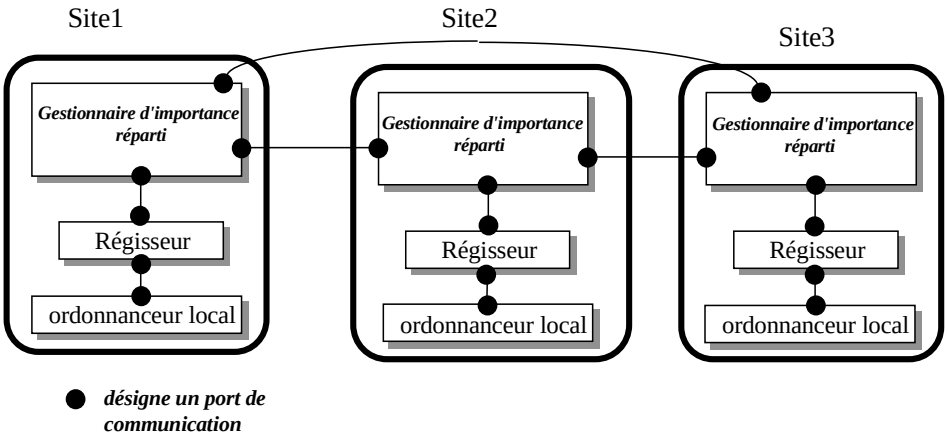


Figure 4.5: *Gestionnaire d'importance réparti implanté sur CHORUS*

3. Outils et méthodes de validation

En matière d'outils et de méthodes de validation d'applications temps réel réparties, très peu de choses existe. Nous décrivons d'abord PERTS un outil d'aide à la validation d'applications temps réel (réparties ou non) ensuite une méthode de validation par analyse d'ordonnançabilité.

3.1. L'outil PERTS

PERTS est un outil graphique [LRD 93 *et al.*] d'aide à la validation d'applications monoprocesseur ou réparties en analysant leur ordonnançabilité. Il permet de visualiser la séquence d'exécution sur chaque processeur et d'évaluer les caractéristiques temporelles d'une application en termes de charge, de temps d'utilisation des ressources et de temps de réponse.

PERTS est composé de trois programmes:

* *tgedit*: un éditeur de tâches qui construit le graphe de précédence de toutes les tâches de l'application,
* *rgedit*: un éditeur de ressources qui définit le ou les processeurs utilisés et
* *sched*: qui est l'ordonnanceur proprement dit.

3.1.1. Éditeur de tâches

Cette interface permet d'abord d'affecter un nom à l'application à créer ensuite de construire toutes les tâches en définissant leurs caractéristiques temporelles. Celles-ci incluent des paramètres classiques tels que : le nom, la date de réveil, le délai critique, la période, la durée d'exécution mais aussi: l'importance de la tâche, son type (soft ou hard), le processeur sur lequel elle s'exécute, ses sections non préemptibles ou optionnelles, les ressources nécessaires avec les instants de début et fin d'utilisation, etc. Les relations de précédence entre les tâches sont construites graphiquement (voir figure 4.6) et pour chaque relation, il y a lieu de définir des informations telles que la plus grande quantité d'information échangée entre une tâche et ses successeurs directs, la durée minimale entre la fin d'exécution de la tâche source et le début d'exécution de la tâche destinataire, etc.

3.1.2. Éditeur de ressources

Plusieurs arborescences peuvent être construites, la racine correspond à un processeur et les nœuds aux ressources disponibles (mémoire, etc.) (voir figure 4.7). Chaque ressource est définie par plusieurs paramètres tels que le nom, le nombre d'exemplaires, les temps d'acquisition et de libération, etc.

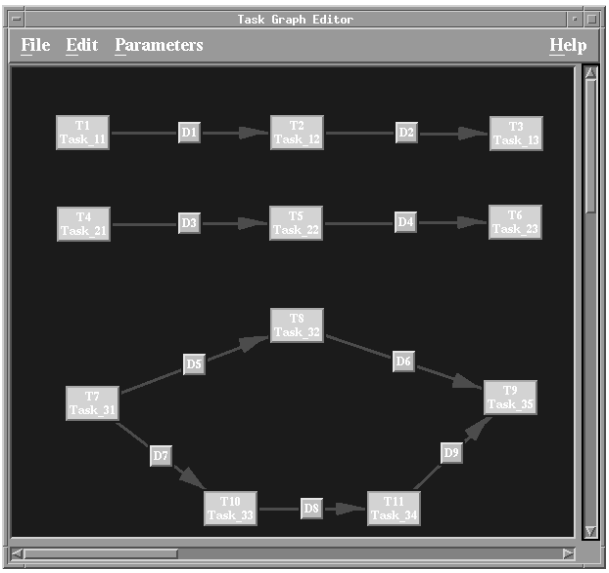


Figure 4.6: *Un exemple de configuration de tâches*

3.1.3. L'ordonnanceur

Ce programme nous offre la possibilité de choisir entre RM et DM comme algorithme d'ordonnancement au niveau des processeurs, associé à un protocole d'allocation de ressources tels que le protocole à priorité plafond ou le protocole à pile. L'analyse d'ordonnançabilité est faite pour chaque site (processeur) en fonction de l'algorithme d'ordonnancement et du protocole d'allocation de ressources.

A partir des informations fournies par l'utilisateur sur les tâches et les ressources, des informations sont calculées et affichées comme par exemple: le résultat de l'ordonnançabilité, le taux d'utilisation des tâches périodiques, des apériodiques, le taux global d'utilisation des ressources, etc. (voir figure 4.8). A ce niveau, la modification d'un paramètre temporel pour une tâche déclenche une réexécution automatique de l'ordonnanceur et un affichage des nouveaux paramètres calculés.
Pour chaque tâche, des informations peuvent être visualisées sous forme de courbe (le taux d'utilisation avec le taux de blocage, le taux d'exécution de la tâche et le taux d'exécution de tâches plus prioritaires, etc.). Un diagramme de Gantt montrant la séquence d'exécution des tâches peut être visualisé mettant en évidence, à l'aide de couleurs différentes, des instants pertinents comme par exemple: la durée d'exécution d'une tâche, la durée d'utilisation d'une ressource donnée, l'instant de réveil, l'instant de fin d'exécution, l'instant de blocage, l'instant de préemption, l'instant de demande ou de libération de la section critique, l'instant de début ou fin d'une section non préemptible, etc.

L'utilisateur n'a pas la possibilité de définir le réseau qu'il veut utiliser. Il n'a pas connaissance du réseau utilisé par défaut car aucune information n'est précisée. L'outil calcule le taux d'utilisation du réseau qui constitue la seule information fournie.
Il est possible de définir une transaction en choisissant, sur le graphe de précédence des tâches, une tâche initiale et une tâche finale appartenant au même chemin. Le taux d'utilisation de toutes les tâches impliquées dans ce chemin est calculé et affiché.

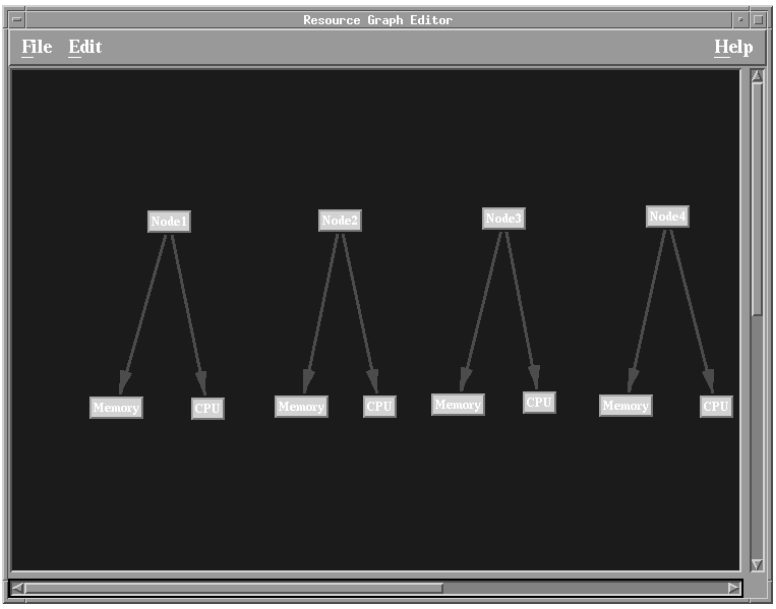


Figure 4.7: *Exemple de ressources par site*

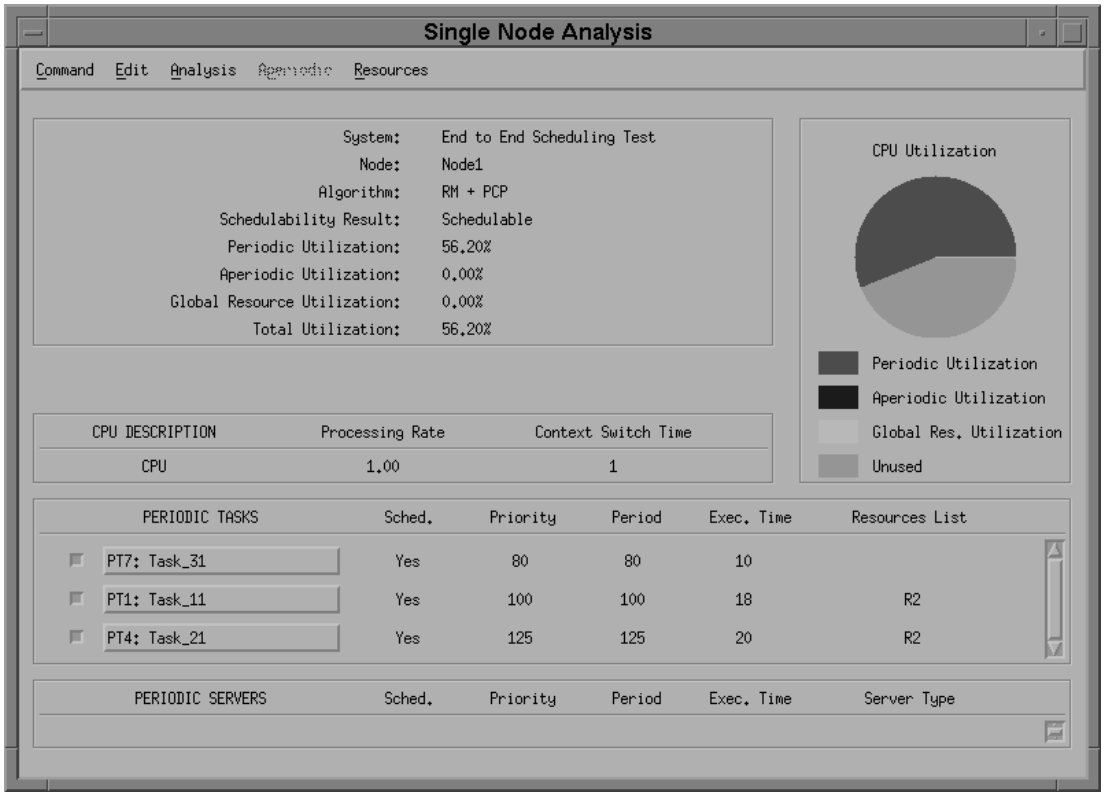


Figure 4.8: *Informations relatives à l'ordonnançabilté des tâches d'un site*

3.2. Une méthode de validation d'applications temps réel réparties

Dans la méthode proposée par [KLR 94, SRS 94], un système réparti est défini comme un ensemble de stations reliées par un médium de communication via lequel les tâches qui résident sur les stations s'échangent des messages. L'ensemble des stations et du médium de communication est vu comme un ensemble de sous-systèmes. Afin d'étudier l'ordonnançabilité des tâches dans un environnement réparti, la notion de *délai critique de bout en bout* est définie. Elle est égale à la somme des délais critiques des sous-systèmes constituant le système réparti. Le principe se base sur le fait que : si chaque sous-système est ordonnançable en appliquant les techniques connues dans l'environnement monoprocesseur, alors le délai critique de bout en bout est aussi respecté. Dans ce cas, le système est dit valide du point de vue temporel.

3.2.1. Hypothèses générales

- chaque site utilise RM comme algorithme d'ordonnancement local des tâches,
- le protocole à priorité plafond est intégré à RM afin de minimiser le temps de blocage dû au partage de ressources locales critiques,
- les tâches T_i sont caractérisées par les paramètres temporels classiques (r_i, C_i, D_i, P_i) décrits au chapitre 2,
- la prise en compte de la précédence locale entre tâches est faite en scindant toutes les tâches de même priorité (c'est-à-dire de même période) en une seule,
- les tâches sont périodiques et échangent des messages périodiques. Les tâches apériodiques sont traitées par des serveurs périodiques.

Afin d'illustrer le principe de cette méthode, nous présentons un exemple d'application répartie implantée sur un réseau FDDI, qui a été décrit dans [KLR 94, SRS 94].

3.2.2. Description de l'exemple d'application

Soit un système réparti composé de quatre stations N_1, N_2, N_3 et N_4 connectées au réseau FDDI sur lesquelles est implantée une application temps réel. Les stations N_1, N_2 et N_3 sont dédiées au contrôle de robot alors que N_4 possède une console opérateur qui sert pour les commandes d'affichage.
Comme le montre la figure 4.9, sur les stations N_1, N_2 et N_3 cinq tâches indépendantes sont définies :

- A_1 pour le contrôle du robot
- A_2 et A_3 des tâches de mesure
- A_4 et A_5 pour la commande du système.

Sur la station N_4, il existe trois tâches :

- B_2 affiche les données envoyées par la station N_1
- B_1 et B_3 partagent une ressource critique et affichent des données en provenance respectivement de N_2 et de N_3

Les caractéristiques temporelles des tâches sont données dans les tableaux 4.1 et 4.2.

Tableau 4.1 : *Caractéristiques temporelles des tâches des stations N_1, N_2 ou N_3*

Tâches	P_i	C_i	D_i
A_1	40	6	40
A_2	50	20	50
A_3	100	20	100
A_4	200	30	100
A_5	400	24	400

Tableau 4.2 : *Caractéristiques temporelles des tâches de la station N_4*

Tâches	P_i	C_i	D_i
B_1	80	20	80
B_2	200	61	200
B_3	300	30	300

A_2 de N_1 lit des données à partir d'un capteur pendant 50ms et transmet un Mégatbit de données chaque 50ms, qui sont traitées par N_4 pendant 200ms pat la tâche B_2. De même, les tâches B_1 et B_3 traitent chacune un Mégabit de données reçues respectivement des tâches A_2 de N_2 et de N_3.

Contrainte de temps

Le délai critique de bout en bout égal à l'intervalle de temps qui sépare l'instant de lecture, par A_2, de la donnée à partir du capteur de la station N_1 de l'instant de son affichage sur la station N_4 est de 0,5s.

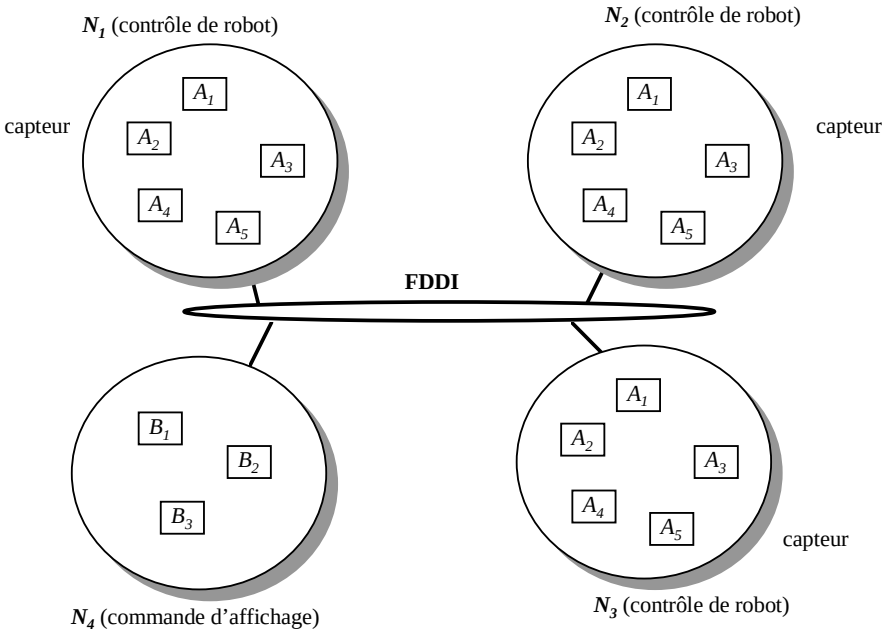


Figure 4.9 : *Exemple d'application temps réel répartie*

3.2.3. Description du réseau utilisé

Le réseau local utilisé est le réseau FDDI de débit 100Mb/s qui utilise le protocole à jeton comme méthode d'accès. Les stations sont connectées autour d'un anneau. Une station est autorisée à émettre lorsqu'elle obtient le jeton. Le temps que met un jeton pour parcourir un anneau inactif (aucune station n'émet) est appelé *Walk Time* (W_T). Le temps maximal que peut mettre le jeton pour faire un tour de l'anneau est un paramètre du réseau donné par *TTRT* (*Target Token Rotation Time*).
Chaque station émet uniquement en synchrone pour un temps maximal autorisé appelé *capacité synchrone* à chaque fois que le jeton est obtenu. La capacité synchrone de chaque station est un pourcentage de (*TTRT*-*WT*). Dans un mode synchrone, le protocole maintient un temps entre deux visites consécutives d'une station par le jeton, inférieur à *TTRT*.
Chaque station N_i peut émettre, une fois, chaque *TTRT* pour une capacité synchrone de $H_i = \frac{U_i}{U}(TTRT - W_T)$ avec :
U_i: la bande pasante effectivement utilisée par N_i et
$U = U_1 + ... + U_n$ s'il existe *n* stations dans le réseau ayant utilisé effectivement des bandes passantes
($TTRT - W_T$) : la bande passante totale pour un tour complet de l'anneau.

Données numériques
W_T= 1ms, *TTRT* = 8ms, débit= 100Mb/s
A chacune des stations N_1, N_2 et N_3 est alloué 30% du temps ($TTRT - W_T$), par contre 10% seulement sont alloués à N_4.

3.2.4. Analyse temporelle

Afin d'illustrer le principe de cette analyse, on s'intéressera uniquement à la tâche A_2 de N_1. Le principe est le même pour les tâches A_2 des stations N_2 et N_3.
Les données lues à partir du capteur de N_1 doivent être affichées au bout de 0,5 s (délai critique de bout en bout). Ces données utilisent trois sous-systèmes: CPU(N_1), réseau FDDI et CPU(N_4). Le principe consiste à associer un délai critique *D* à chaque sous-système: si la somme des délais critiques des sous-systèmes est inférieure au délai critique de bout en bout, les contraintes temporelles du système réparti sont respectées donc il est ordonnançable.

3.2.4.1. Analyse du sous-système N_1

Le but est de montrer que la configuration de tâches de N_1 est ordonnançable avec RM. La condition suffisante d'ordonnançabilité relative à l'algorithme RM n'étant pas vérifiée (thèorème de [LL 73]), [LRK 94] utilisent la méthode du test d'acceptation qui s'applique aux systèmes monoprocesseurs pour montrer que la configuration de tâches, notamment A_2, est ordonnançable. Nous ne décrirons pas ici le test d'acceptation mais le lecteur intéressé pourra consulter [KLR 94] et [SRS 94].

3.2.4.2. Analyse du sous-système réseau

Il s'agit de voir si la capacité synchrone allouée est suffisante pour gérer le trafic généré par la station N_1. L'idée est de traiter le trafic de messages généré par chaque station comme une tâche du réseau.

A_2 sur N_1 génère 1 Mb de données chaque 50ms. Comme le débit du réseau est de 100Mb/s, la transmission de 1 Mb se fait en 1/100= 0,01s= 10ms. Nous venons de définir une tâche réseau φ_1 qui occupe le réseau pendant 10ms chaque 50ms (φ_1: C=10, P= 50). Le temps réservé aux autres stations est calculé comme suit :
Sachant que W_T est égal à 1ms, que *TTRT* est égal à 8ms et que 30% de la bande passante est réservée à N_1: 30%(*TTRT*-W_T)= 2,1ms= H_1, le temps réservé aux autres stations est: 8-2,1 = 5,9ms pour chaque tour de l'anneau fait en 8ms.

Le temps non disponible pour N_1 sera traité comme le temps réservé à une autre tâche réseau φ_2 dont la durée d'exécution C est de 5,9ms et la période P de 8ms.
Les tâches du réseau sont donc :

- φ_1: C_1=5,9 et P_1=8 de grande priorité selon l'algorithme RM
- φ_2: C_2 = 10 et P_2=50 de faible priorité selon l'algorithme RM

A l'aide du test d'acceptation qui normalement s'utilise dans l'environnement monoprocesseur, on montre que ces tâches sont ordonnançables sur le réseau.

3.2.4.3. Analyse du sous-système N_4

En appliquant le test d'acceptation encore une fois, on montre que les tâches de la station N_4, notamment B_2, sont ordonnançables.

3.2.4.4. Analyse du délai critique de bout en bout

- A_2 de période 50, chargée de capturer les données sur N_1, est ordonnançable sur N_1
- φ_2 de période 50, chargée de transférer les données de N_1, est ordonnançable sur le réseau
- B_2 de période 100, chargée de traiter les données reçues, est ordonnançable sur N_4

Le temps de réponse total est de 300ms (50+50+100), il est inférieur à 500ms qui est le délai d'affichage (ou encore le délai critique de bout en bout). La contrainte de temps spécifiée dans le cahier des charges de l'application est bien respectée.

3.2.5. Conclusion

Nous venons de voir, à travers cet exemple, que la technique d'ordonnancement monoprocesseur RM associée au test d'acceptation comme condition d'ordonnançabilité peut s'appliquer au réseau. L'idée de partitionner le système en sous-systèmes indépendants est intéressante car l'étude de l'ordonnancement des tâches est faite alors de manière séparée. Néanmoins, cette méthode ne tient pas compte de la technique d'ordonnancement des messages dans le réseau et elle ne peut pas s'appliquer, *a priori*, à un algorithme d'ordonnancement à priorité dynamique (ED par exemple). De plus, le traitement du trafic de messages comme des tâches à ordonnancer sur le réseau en déduisant leurs périodes à partir du paramètre *TTRT* est un raisonnement étroitement lié au comportement du réseau FDDI, qui ne peut s'appliquer à d'autres réseaux notamment ceux qui sont basés sur le principe de compétition comme CAN et CSMA/DCR.

4. Conclusion

Dans ce chapitre, nous avons décrit trois systèmes d'exploitation répartis d'une part, et d'autre part, un outil et une méthode qui permettent d'étudier l'ordonnançabilité d'une application temps réel répartie.

Le premier système (MARS) a été choisi pour son déterminisme total : toutes les tâches à exécuter doivent être connues au préalable pour construire un ordonnancement valide et elles utilisent un réseau déterministe pour leurs communications, les tranches de temps allouées aux différents sites étant connues à l'avance.

Le second système (SPRING) offre un peu plus de souplesse du point de vue de l'ordonnancement et exploite davantage le parallélisme en considérant chaque site comme un système multiprocesseur sur lequel les processeurs application sont dédiés à l'exécution des tâches périodiques critiques et le processeur système est chargé de garantir des tâches de degré de criticité moindre (tâches essentielles) et de prendre en compte les événements aléatoires. Un ordonnanceur distribué est intégré dans chaque site et est lancé pour trouver un site susceptible d'accepter une tâche à chaque fois que celle-ci ne peut être garantie localement.

A travers l'amélioration proposée par l'équipe du CEDRIC pour l'ordonnancement distribué dans le système CHORUS, nous constatons que la tendance va vers la séparation de l'ordonnancement local (qui ne prend en compte que les tâches dont il est sûr de pouvoir garantir) de l'ordonnancement global qui lui, doit coopérer avec les autres sites en vue de trouver une solution (trouver un site qui peut garantir une tâche non garantie localement), s'il en existe, tout en veillant à équilibrer la charge des différents sites. A la différence de MARS et de SPRING qui recherchent un ordonnancement valide grâce à un algorithme de construction d'arbre basé sur une heuristique à définir et pouvant exploser rapidement, CHORUS implémente un algorithme d'ordonnancement local au niveau de chaque site et un algorithme global pour la migration éventuelle des tâches.

La méthodologie de validation que nous proposons dans le prochain chapitre s'appliquerait bien aux applications temps réel réparties comme celles supportées par CHORUS car elle suppose un système réparti dans lequel chaque site utilise un algorithme d'ordonnancement local et communique avec les autres sites à travers un réseau de communication. Notre travail consiste donc à proposer une méthodologie permettant de valider une application temps réel répartie en analysant son ordonnançabilité. C'est pourquoi, nous avons jugé utile d'étudier de près l'outil PERTS qui nous paraît s'inscrire dans cette optique, même s'il présente des insuffisances non négligeables. En effet, le modèle réseau, tel qu'il est présenté dans PERTS, n'est pas riche car il ne permet pas de fournir suffisamment d'informations relatives à la communication si l'on veut prendre en compte les caractéristiques physiques du réseau et définir un protocole de communication. De plus, le calcul des performances n'est pas fait ; seul le taux d'utilisation est calculé. En contrepartie, les modèles de tâche et de ressources sont très riches et permettent de fournir une quantité importante d'informations pour caractériser chaque tâche et chaque ressource.

Quant à la méthode de validation présentée et proposée par [KLR 94, SRS 94], elle est très limitée vu qu'elle s'applique uniquement à l'agorithme d'ordonnancement RM et au réseau FDDI.

Le prochain chapitre détaille les différentes étapes de notre méthodologie de validation qui repose sur un ordonnancement conjoint des tâches et des messages. Par rapport à PERTS, notre méthodologie considère un modèle réseau beaucoup plus riche qui prend en compte les caractéristiques physiques du réseau et calcule les durées de communication en se basant sur le protocole de communication du réseau. Au niveau des sites, nous modélisons les tâches à l'aide d'un nombre d'attributs temporels plus restreint. Néanmoins notre méthodologie sait gérer comme, PERTS, les relations de précédence et le partage de ressources entre tâches du même site. De plus, elle s'applique aux algorithmes d'ordonnancement à priorité dynamique tel que ED, chose que ne sait pas faire PERTS.
Par rapport à la méthode proposée par [KLR 94, SRS 94], nous n'imposons ni un réseau ni un algorithme d'ordonnancement particuliers. Dans le chapitre 5, nous montrons comment utiliser n'importe quel algorithme d'ordonnancement (RM, DM ou ED) et n'importe quel réseau de communication à délai d'accès borné à travers cinq exemples de réseaux temps réel (FIP, CAN, CSMA/DCR, FDDI et TDMA).

CHAPITRE 5

UNE METHODOLOGIE DE VALIDATION BASEE SUR L'ANALYSE D'ORDONNANÇABILITE

1. Introduction

Nous définissons une application temps réel répartie comme un ensemble de tâches réparties sur différents sites connectés à un réseau local où le seul moyen de communication est l'échange de messages. Elle est dite à contraintes strictes lorsque les tâches et les messages doivent respecter leurs échéances respectivement lors de leur exécution et leur transfert.

Le but de ce chapitre est de présenter la méthodologie proposée pour la validation temporelle de ce type d'applications. Cette méthodologie est assez générale dans le sens où nous pouvons l'appliquer avec n'importe quel algorithme d'ordonnancement (RM, DM ou ED) et n'importe quel protocole de communication (FIP, TDMA, CAN, CSMA/DCR, FDDI) pourvu qu'il fournisse un délai d'accès borné. Elle est principalement constituée de trois étapes:

1. *Modélisation de l'application* à l'aide d'attributs temporels définis pour les tâches et les messages. Un modèle temporel, pour les messages, similaire à celui des tâches est présenté. Le délai de transfert étant l'une des caractéristiques des messages, il sera calculé pour chaque protocole de communication présenté au chapitre 3.

2. *Prise en compte de la précédence* : un graphe de précédence global est présenté en mettant en évidence d'une part les relations de précédence entre tâches locales d'un site pour représenter les relations de synchronisation et de communication entre elles et d'autre part les relations de précédence entre tâches distantes pour représenter l'échange de messages via le médium. Notre objectif, dans ce cas, est de dériver les relations de précédence par modification des attributs temporels des tâches et des messages en vue d'obtenir des entités (messages et tâches) indépendantes.

3. *Ordonnancement des tâches et des messages* : les tâches et les messages dont les caractéristiques temporelles sont obtenues à l'étape 2 sont ordonnancés respectivement sur chaque site et sur le réseau. Un diagramme de Gantt est tracé pour chaque site et pour le réseau et montre la séquence d'ordonnancement obtenue dans la situation dite du pire cas. Il reste alors à vérifier, dans ce cas, si chaque tâche et chaque message respecte bien son échéance afin de déduire la validité de l'application et d'analyser ses performances à travers des critères mesurés.

Très peu de travaux traitent simultanément de l'ordonnancement des tâches et des messages. [KLR 94] généralisent l'algorithme RM pour étudier l'ordonnancement des tâches et des messages dans le réseau FDDI. [EVC 96] améliorent cet algorithme en considérant des sources multiples au niveau de chaque site. Néanmoins, cette méthode semble s'appliquer au cas particulier des réseaux FDDI. [TBW 95] dérivent une analyse d'ordonnançabilité qui borne le délai d'accès au médium, illustrée par deux protocoles d'accès. [SSB 97] ont développé un outil d'aide à la validation des applications temps réel réparties autour du réseau CAN, basée sur la vérification du respect des contraintes temps réel des messages. L'outil intègre deux méthodes, une qui simule le cas réel et l'autre qui calcule les temps de réponse dans le pire des cas selon l'analyse faite par Tindell et al. [TB 94]. La faiblesse de cet outil se résume dans le fait que l'activité des tâches se restreint à l'envoi et à la réception des messages et qu'aucun type d'interaction entre tâches n'est prévu. Par contre il est assez réaliste dans le sens où il intègre les erreurs de transmission selon différentes probabilités.

Après avoir énoncé les hypothèses générales de travail, les différentes étapes citées ci-dessus sont détaillées dans la suite de ce chapitre.

2. Le modèle général

2.1. Modèle structurel

Nous émettons d'abord quelques hypothèses générales qui supposent:

- l'existence d'une horloge globale, dans le système, qui sert à définir les paramètres temporels des tâches et des messages par rapport à un même référentiel temporel,

- l'existence d'un réseau fiable c'est-à-dire que les messages sont transmis, sans erreurs, au bout d'un temps fini. Toutefois nous pouvons considérer le cas d'erreurs dans le cas où nous savons borner le temps nécessaire à la détection et à la guérison de ces erreurs de manière à inclure ce temps dans la durée maximale de communication,

- que les architectures de communication considérées sont à trois couches seulement,

- que seuls les protocoles de communication à délai d'accès borné sont utilisés car ils fournissent un délai de transmission connu et borné pour les messages,

- que les tâches sont placées sur les différents sites de manière statique et qu'aucune migration de tâche n'est permise,

- que chaque site dispose d'un exécutif temps réel, en plus des tâches application, en particulier d'un ordonnanceur local de tâches et intègre les fonctions de gestion du réseau,

- qu'il n'y a pas de dérive d'horloges et que l'horloge globale reste toujours synchronisée avec le temps réel ; par conséquent il n'y a pas de messages de synchronisation d'horloge,

- que les temps d'ordonnancement et de changement de contexte, dans les sites, sont nuls,

- que le nombre de sites est connu.

2.2. Le modèle temporel de tâches

Le modèle de tâches que nous utiliserons, au niveau de chaque site, est celui défini dans le chapitre 2 ; autrement dit toute tâche T_i est caractérisée par quatre attributs temporels:

- r_i: la date de première activation,

- C_i: la durée maximale d'exécution sur le processeur du site,

- D_i: le délai maximal correspondant à respecter et

- P_i: la période qui est l'intervalle de temps maximal séparant deux instances successives de T_i.

Au niveau de chaque site, les tâches peuvent être liées par une relation de précédence que l'on qualifiera de *locale* et partager des ressources critiques. De plus, une relation de précédence dite *globale* est définie entre toute paire de tâches distantes dont l'une est émettrice de message et l'autre en est réceptrice.

Nos hypothèses concernant les tâches sont les suivantes:
- seules les tâches périodiques sont considérées. Ceci n'est pas restrictif car les tâches apériodiques peuvent être vues comme des serveurs périodiques traitant des événements apériodiques [HR 94],
- deux tâches liées par une relation de précédence locale ou globale ont la même période,
- chaque tâche émet des messages à la fin de son exécution et n'en reçoit qu'en début d'exécution. Ce type de tâches peut être obtenu facilement à partir de tâches ordinaires par un simple découpage,
- Les tâches émettrice et réceptrice de messages doivent inclure dans leurs durées d'exécution respectivement le temps de découpage et le temps de réassemblage des paquets au niveau de la couche application, si l'on ne veut pas créer de sous-tâches chargées de ces traitements comme dans [TBW 95].

Les algorithmes d'ordonnancement utilisés sont ceux présentés dans le chapitre 2. Afin de réduire le temps de blocage en attente de ressources, les protocoles d'allocation de ressources leur sont associés.
Dans toute la suite, nous parlerons de messages pour désigner les entités d'information échangées par les tâches application. Ces messages seront décomposés en un ensemble d'entités plus petites appelées *paquets* ou *trames* correspondant aux unités d'information manipulées par la couche MAC.

2.3. Le modèle temporel de messages

Les tâches périodiques d'un site émettent le plus souvent des messages périodiques (ou *synchrones*) lorsque ces derniers sont destinés à des tâches distantes périodiques. De plus, elles peuvent émettre de temps à autre des messages apériodiques (ou *asynchrones*) (voir figure 5.1). Par contre, les tâches apériodiques, étant donné que leur arrivée est aléatoire, émettent toujours des messages asynchrones.

Nous supposons ici que seuls les messages synchrones sont utilisés pour la communication entre tâches périodiques.

Un message synchrone *m* est caractérisé par (voir figure 5.2):

- r_m: l'instant d'insertion de *m* dans la file de transmission,

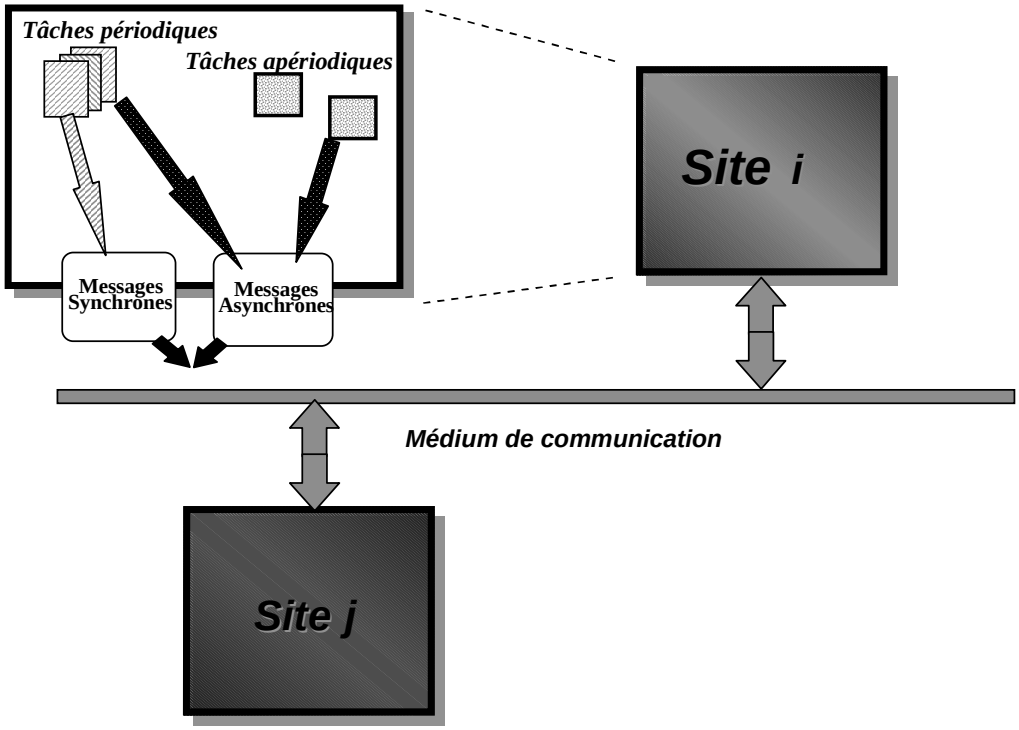


Figure 5.1: *Nature des tâches et messages d'une application temps réel répartie*

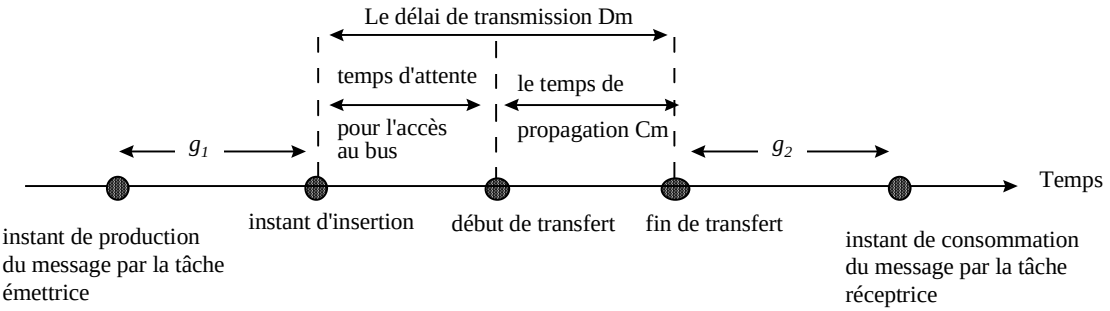


Figure 5.2: *Le délai de transmission d'un message*

- C_m: le temps de propagation sur le réseau, qui est proportionnel à la taille du message et qui correspond au nombre d'unités de temps nécessaires à la transmission du message à travers le médium de communication,

- D_m: le délai de transmission qui sépare le moment où le message est prêt à être émis de l'instant de réception de ce message par le site destinataire. Il correspond au temps de propagation additionné au temps d'attente dans la file de transmission.

- La période P_m peut être ajoutée comme paramètre temporel puisque *m* est généré périodiquement par la tâche émettrice.

Dans la suite, le nombre de messages émis par une tâche et les destinations des messages sont supposés connus. Chaque message possède une taille fixe et un délai critique confondu avec sa période. On associe à chaque message un tampon à une case pour son émission et un autre pour sa réception. De plus, afin de minimiser la gigue, les durées g_1 et g_2 définies dans la figure 5.2 sont supposées nulles.

3. Principe de la méthodologie

Le principe consiste à déterminer les paramètres temporels caractérisant une application temps réel en vue d'étudier l'ordonnançabilité des tâches et des messages la constituant.

Du point de vue ordonnancement, nous pouvons noter les caractéristiques fondamentales des tâches et des messages:

- les tâches et les messages sont définis avec quatre paramètres temporels,

- les tâches d'un site sont ordonnancées à l'aide d'un algorithme d'ordonnancement tandis que les messages sont ordonnancés à l'aide d'un protocole de communication.

Par conséquent, il s'agit d'abord de déterminer les caractéristiques temporelles des messages échangés puisqu'ils sont dotés de paramètres temporels qui varient en fonction du protocole de communication utilisé et des caractéristiques physiques du réseau. Ensuite, il y a lieu de modifier les caractéristiques temporelles (définies initialement) des tâches impliquées dans la communication afin de tenir compte de l'échange de messages. Nous verrons que la détermination des paramètres temporels

dépend étroitement du contexte local d'un site (type d'algorithme d'ordonnancement et le protocole d'allocation de ressources) et du réseau. Une fois les caractéristiques temporelles complètement déterminées, nous procédons à l'ordonnancement des tâches et des messages en vue de vérifier le respect des contraintes temporelles.

3.1. Modélisation de l'application

Considérons une tâche émettrice $\mathcal{E}$ sur un site N activée à l'instant r_e, avec un temps d'exécution C_e, un délai critique D_e et une période P_e : $\mathcal{E}=(r_e, C_e, D_e, P_e)$. A chaque cycle d'exécution, $\mathcal{E}$ peut envoyer un message m à une tâche $\mathcal{R}$ d'un site M. De manière similaire, $\mathcal{R}$ est caractérisée par une date d'arrivée, une durée d'exécution, un délai critique et une période c'est-à-dire $\mathcal{R}=(r_r, C_r, D_r, P_r)$. $\mathcal{E}$ et $\mathcal{R}$ étant liées par une relation de précédence globale, P_r est égale à P_e par hypothèse.

Les caractéristiques des tâches que nous considérons, à ce niveau, sont supposées être celles obtenues après prise en compte des relations de précédence locale (voir chapitre 2).

Le message m est caractérisé par (r_m, C_m, D_m, P_m) avec:

- r_m l'instant d'insertion dans la file de transmission,
- C_m le temps de propagation de m,
- D_m le délai de transmission allant du temps d'insertion dans la file à la réception du message par la destination et
- P_m la période égale à P_e car m est émis périodiquement par $\mathcal{E}$.

Nous devons avoir nécessairement les relations suivantes: $0 \leq C_m \leq D_m \leq P_m$.

Le but est de calculer les caractéristiques temporelles des messages en supposant que celles des tâches sont connues au départ c'est-à-dire définies par l'utilisateur ou déterminées automatiquement à l'aide d'un outil d'évaluation des paramètres.

Comme la période P_m est connue puisqu'elle est égale à celle de la tâche émettrice, il reste à déterminer les paramètres C_m, D_m et r_m. C_m et D_m sont deux paramètres qui dépendent étroitement du réseau utilisé et de la taille du message. En effet, C_m est le temps de transmission des paquets constituant le message m et D_m est le délai de communication entre la tâche émettrice qui insère un message dans la file de transmission et la tâche réceptrice capable de consommer le message.

3.1.1. Calcul du temps de propagation d'un message

3.1.1.1. Réseau FIP

Dans le cas de FIP, vu le type d'applications distribuées auxquelles nous nous intéressons, seul le trafic périodique est considéré. La transmission de toute variable périodique (selon la terminologie de FIP) se fait en 2 étapes: la diffusion d'une requête

ID_DAT par l'arbitre de bus ; le producteur de la variable spécifiée dans la requête répondra en diffusant la variable proprement dite à l'aide de *RP_DAT*. Le temps de transmission C_m de *m* est égal au temps d'émission de *ID_DAT* ajouté au temps d'émission de *RP_DAT*, plus deux fois le temps de retournement *tr*.

ID_DAT= 21 bits+5 octets=61 bits
RP_DAT=21 bits+3 octets +*n* octets = 45 bits +*n* octets
C_m=(*n**8+61+45)/*De* + 2**tr* si *De* est le débit du réseau

$$C_m = (n*8+106)/De + 2*tr \qquad (n \leq 128)$$

3.1.1.2. Réseau CAN

Comme les messages dans CAN sont confondus avec les trames de la couche MAC alors le temps de propagation d'un message est celui du paquet. Autrement dit, un message CAN ne peut pas transporter plus que 8 octets de données.

Un paquet contient 47 bits de contrôle et 8 octets d'information utile (voir structure de la trame dans l'annexe 3). De plus, des bits de bourrage peuvent être rajoutés. Leur nombre est au maximum égal à : $\left\lfloor \frac{34+8n}{5} \right\rfloor$.

Par conséquent si *De* est le débit du réseau, le temps de propagation est égal à :

$$C_m = 47+n*8+\left\lfloor \frac{34+8n}{5} \right\rfloor)/De \quad (n \leq 8)$$

3.1.1.3. Autres Réseaux

Pour les réseaux FDDI, CSMA/DCR et TDMA, un message émis par une tâche de l'application peut avoir une taille qui excède le nombre maximal d'octets pouvant être transporté par une trame. Un découpage en paquets est donc inévitable, il sera fait au niveau de la couche application avec insertion des paquets dans la file de transmission dans l'ordre FIFO. Si un message *m* est constitué de *p* paquets, la taille d'un paquet variant d'un réseau à un autre, la transmission de *m* nécessite *p* fois C_{pi} où C_{pi} correspond à la durée de transmission d'un paquet *i*:

$$C_m = \sum_{i=1}^{p} C_{pi}$$

Si le réseau est CSMA/DCR de débit *De*, le paquet se compose de 144 bits (18 octets) de contrôle et de 1518 octets d'information utile au maximum (voir trame dans l'annexe 3). 144 est le nombre de bits de contrôle obtenu en supposant que la taille du champ d'adresse d'une station émettrice ou réceptrice est égale à 2 octets.

$$C_p = (n*8+144)/De \qquad (n \leq 1518)$$

Si le réseau est FDDI de débit *De*, le paquet contient 168 bits (21 octets) de contrôle et 4490 octets d'information utile au maximum (voir trame dans l'annexe 3). 168 est le

nombre de bits de contrôle obtenu en supposant que la taille du champ d'adresse d'une station émettrice ou réceptrice est égale à 2 octets.

$$C_p = (n*8+168)/De \qquad (n \leq 4490)$$

Si le réseau est TDMA de débit *De*, le paquet contient 28 bits de contrôle et 8 octets d'information utile au maximum (voir trame dans l'annexe 3).

$$C_p = (n*8+28)/De \qquad (n \leq 8)$$

Exemple

Une tâche qui veut envoyer 20 octets de données par exemple, devra émettre :

– 3 messages distincts m_1, m_2 et m_3 de tailles respectives 8 octets, 8 octets et 4 octets dans CAN.

$C_{m1} = C_{m2} = 130\mu s$ et $C_{m3} = 92\mu s$ si De=1Mb/s

– Un message *m* dans FIP ne doit pas excéder une taille de 128 octets. Par conséquent, la quantité de données 20 octets peut correspondre à un seul message : $C_m = 286\mu s$ si De=1Mb/s et tr=10µs

– Un seul message est émis par la tâche lorsque les réseaux utilisés sont FDDI, TDMA et CSMA/DCR quelle que soit la taille du message car, dans ce cas, la fragmentation du message est possible et le message correspondra à plusieurs trames au niveau de la couche MAC.

Dans FDDI, si le débit *De* est de 100Mb/s alors : $C_m = 1*C_p = 3{,}28\mu s$.

Dans CSMA/DCR, si le débit est de 10Mb/s alors : $C_m = 1*C_p = 30{,}4\mu s$. Comme la durée de transmission du paquet est plus petite que la tranche canal, un certain nombre d'octets de bourrage est rajouté dans le paquet (trame) afin d'atteindre T_c. Dans ce cas, $C_m = 50{,}4\mu s$ si le nombre d'octets rajouté est de 25.

Dans TDMA, si le débit est de 1Mb/s alors : $C_m = 3* C_p = 244\mu s$ sachant que les deux premiers paquets contiennent chacun 8 octets de données (taille maximale du champ de données) et que le troisième n'en contient que 4.

3.1.2. Calcul du délai de transmission d'un message

Comme cela est expliqué dans [MZ 95], D_m doit être au moins égal au temps d'attente dans la file de transmission rajouté au temps de propagation du message via le médium de communication (voir figure 5.2). Afin de calculer ce délai, nous considérons la situation la plus défavorable qui suppose une charge maximale du réseau pour inclure un temps d'attente maximal avec une possibilité de contention. Les protocoles de communication présentés dans le chapitre 3 serviront d'illustration de ce calcul.

Calculer le délai de transmission D_m d'un message *m* revient à évaluer le temps d'attente de *m* dans la file de transmission car le temps de propagation C_m est facile à déterminer connaissant la taille de *m* (§ 3.1.1). Il s'agit donc de construire la file de transmission du site émetteur afin de déduire le nombre *s* de paquets qui peuvent précéder le message *m* et par conséquent retarder son envoi. Le principe est de considérer tous les messages émis par le site émetteur du message *m* car ils sont tous susceptibles de précéder *m* dans la file de transmission. Le nombre de fois qu'un message *m'* peut précéder le message *m* est donné à l'aide du rapport de leurs périodes c'est-à-dire $\left\lceil \frac{P_m}{P_{m'}} \right\rceil$. Le nombre de paquets dans la file est obtenu en remplaçant chaque message par le nombre de paquets qui le constituent. La taille du paquet varie d'un réseau à un autre.

Exemple

Soit l'exemple d'application de la figure 5.3, qui sera traité tout au long de ce chapitre:

Les caractéristiques temporelles des tâches sont données dans le tableau 5.1. Concernant les messages, seules les tailles en nombre d'octets sont connues. Elles correspondent aux valeurs parenthésées accompagnant les messages dans la figure 5.3. Toutes les valeurs du tableau 5.1 sont exprimées en ms. Les périodes sont affectées aux tâches de manière à ce qu'elles soient les mêmes pour les tâches liées par une relation de précédence locale ou globale.

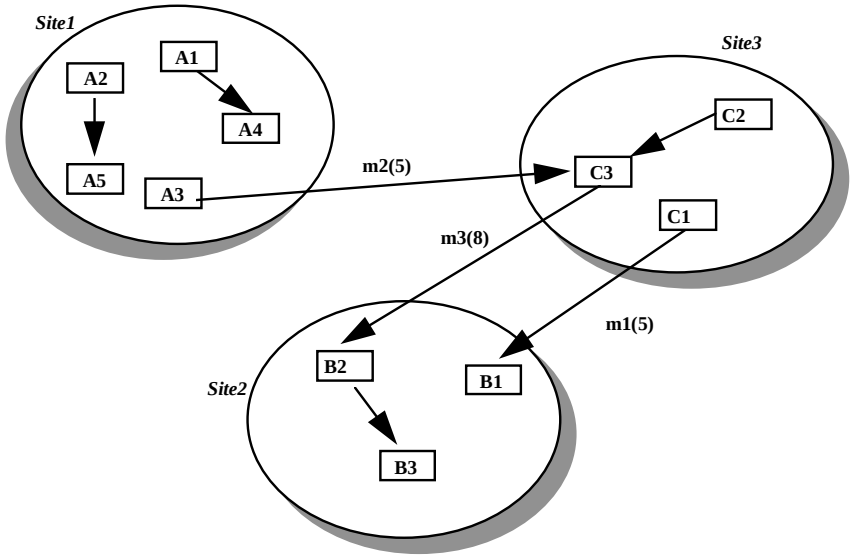


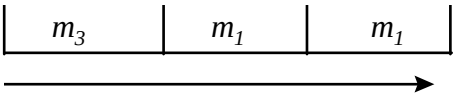
Figure 5.3: *Configuration d'une application répartie*

Tableau 5.1: *Caractéristiques temporelles des tâches de la figure 5.3 (r_i=0 pour toutes les tâches)*

Site	Tâche	C_i(ms)	D_i(ms)	P_i(ms)

Site1	A_1	10	50	60
	A_2	8	150	200
	A_3	10	100	100
	A_4	9	50	60
	A_5	5	200	200
Site2	B_1	5	50	50
	B_2	5	80	100
	B_3	2	100	100
Site3	C_1	6	50	50
	C_2	5	80	100
	C_3	5	80	100

Nous voulons construire la file de transmission pour le site 3 :
Cas 1: Si l'on s'intéresse au message m_3 émis par C_3 alors seul m_1 de C_1 peut précéder m_3. La file de transmission contient alors deux occurrences de m_1 pour une seule de m_3 à cause de leurs périodes:



Cas 2: si l'on s'intéresse à m_1, on s'aperçoit qu'une seule occurrence ($=\left\lceil \frac{P_{m1}}{P_{m3}} \right\rceil$) du message m_3 est émis avant l'émission de m_1.

Maintenant que nous avons montré comment constituer les *s* paquets d'une file de transmission précédant les *p* paquets d'un message *m* à insérer, on peut calculer le délai maximal de transmission du message *m*. Ce calcul varie d'un réseau de communication à un autre.

3.1.2.1. Réseau FIP

Remarquons que dans le cas précis de FIP, le contenu de la file de transmission ne nous intéresse pas car l'ordonnancement est centralisé par l'arbitre de bus et est connu, *a priori*, grâce à la table de scrutation construite une fois pour toute.
La table de scrutation contient l'ensemble des variables avec les différentes périodes ainsi que les temps de propagation. Le calcul de D_m nécessite le tri de toutes les variables suivant leur période (voir description de FIP, §5 chapitre 3).

$$D_m = \sum C_x$$

où x est une variable de période plus petite que celle de m ($P_x \leq P_m$).

Remarquons que dans ce cas, l'ordonnancement des variables se base sur une priorité inversement proportionnelle à la période comme pour l'ordonnancement RM. En effet, comme expliqué dans [Son 96], l'ordonnancement des variables périodiques se fait hors ligne selon un algorithme RM non préemptif.

Exemple

Construisons la table de scrutation de l'arbitre de bus (voir tableau 5.2) pour l'exemple d'application de la figure 5.3. Les variables sont m_1, m_2 et m_3 de périodes respectives, héritées des tâches émettrices, 50, 100 et 100.
Les temps de transfert des variables correspondent, dans notre cas, aux temps de propagation C_m. L'arbitre de bus provoquera l'émission des variables dans l'ordre dicté par la table de scrutation (ordre croissant des périodes) c'est-à-dire m_1 ensuite m_2 et enfin m_3. Le délai de transmission de m_1 sera alors confondu avec C_{m1} c'est-à-dire $D_{m1}=C_{m1}=166\mu s$, celui de m_2 sera égal à $D_{m2}=C_{m2}+C_{m1}=166+166=332\mu s$ et enfin celui de m_3 correspondra à $D_{m3}=C_{m2}+C_{m1}+C_{m3}=166+166+190=522\mu s$.

3.1.2.2. Réseau CSMA/DCR

Soit la file de transmission, de la figure 5.4, qui contient les p paquets du message m précédés par s paquets et construite comme décrit précédemment. Posons C_{pmax} le temps de propagation d'un paquet de taille maximale. Il est égal à la durée de transmission de 1518 octets dans CSMA/DCR, à celle de l'émission de 4490 octets dans FDDI ou à la durée de transfert de 8 octets dans TDMA.

Nous pouvons distinguer deux cas :

Cas favorable:

L'émission des p paquets de m nécessite l'émission des s paquets précédant m dans la file. Dans un cas favorable, l'émission des (p+s) paquets se fait à travers un bus libre et sans collision. Ce qui correspond à l'expression de D_m suivante lorsque C_{pi} est la durée de transmission d'un paquet i.

$$D_m = \sum_{i=1}^{s+p} C_{pi}$$

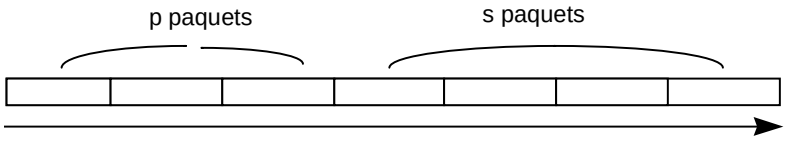


Figure 5.4: *Le contenu d'une file de transmission*

Tableau 5.2 : *La table de scrutation pour l'exemple de la figure 5.3*

Variable (taille en octets)	Périodicité (ms) croissante	Temps de transfert de la variable (µs)
m_1 (5)	50	166

m_2(5)	100	166
m_3 (8)	100	190

Cas défavorable:

Dans le cas défavorable, l'émission des (p+s) paquets se fait dans une situation de collision permanente, c'est-à-dire que chaque paquet émis est en conflit avec d'autres sur le réseau. Dans ce cas, $D_m = C_{p\max} + \sum_{i=1}^{s+p}(C_{pi} + \sum_{j=1}^{n-1} C_{pj} + (n-1)T_c)$ où n est le nombre total de sites et T_c la tranche canal. La formule de D_m signifie que, durant une époque, un paquet i émis par un site est en conflit avec les paquets j émis par les (n-1) autres sites. C_{pmax} représente le temps d'attente maximal de la libération du médium en vue d'émettre. Pour un débit de 10Mb/s, ce temps est égal à 1,2ms.

Notons que la situation la plus fréquente est celle qui suppose que x sites (x<n) au maximum veulent émettre. Dans ce cas, la durée de résolution des conflits (époque) se réduit à l'émission de x paquets. Si $\theta_x^n = \min\{n\text{-}1,\ x+[x\mathrm{Log}(n/x)]\}$ ([y] est la partie entière de y) [LR94], D_m s'exprime comme suit :

$$D_m = C_{p\max} + \sum_{i=1}^{s+p}(C_{pi} + \sum_{j=1}^{x-1} C_{pj} + \theta_x^n T_c)$$

Exemple

Nous pouvons calculer C_m et D_m des messages de l'application de la figure 5.3 en supposant un réseau CSMA/DCR de débit 10 Mb/s, une tranche canal de 50µs (voir tableau 5.3). Remarquons que seuls deux sites sur trois sont émetteurs ; ce qui signifie que le calcul de D_m se fera en utilisant la formule de [LR 94]. Sachant qu'un paquet peut contenir au maximum 1518 octets d'information utile, m_1, m_2 et m_3 correspondent chacun à un seul paquet.

m_2 est le seul message susceptible d'être émis par le site 1. Il peut rentrer en conflit pour l'utilisation du médium de communication avec m_1 ou m_3 du site 3.

Calculons d'abord les temps de propagation des messages :

C_{m1}=C_{m2}=50,4µs en rajoutant 40 octets de bourrage pour atteindre une tranche canal car le nombre d'octets de données à transférer est trop petit. De même, en rajoutant 37 octets de bourrage, C_{m3}=50,4µs.

Calculons les délais de transmission :

Sur le site 1, si m_2 rentre en conflit avec m_1 alors :

Tableau 5.3 : *Valeurs de C_m et D_m calculées dans le cas de CSMA/DCR*

Message	C_m=p*C_p	D_m
m_1	50,4µs	1,629 ms

m_2	50,4µs	1,428 ms
m_3	50,4µs	1,830 ms

$D_{m2}=1*(C_{m1}+ C_{m2}+ \theta_{2^3}T_c)+C_{pmax}= 1{,}428\mu s$ (s=0)

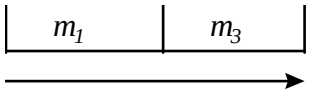
Si m_2 rentre en conflit avec m_3 alors :

$D_{m2}=1*(C_{m3}+ C_{m2}+ \theta_{2^3}T_c)+C_{pmax}= 1{,}428\mu s$

Si les délais calculés étaient différents, on aurait pris celui qui donne le plus grand temps.

Sur le site 2, m_1 peut être en concurrence avec m_2 pour l'obtention du médium, de plus il peut être précédé de m_3 dans la file de transmission. Par conséquent, m_1 et m_3 peuvent tous les deux rentrer en conflit avec m_2. D'où :

$D_{m1}=(C_{m3}+ C_{m2}+ \theta_{2^3}T_c)+ (C_{m1}+ C_{m2}+ \theta_{2^3}T_c)+C_{pmax}= 1{,}629\mu s$



- File de transmission du site3 -

m_3, par contre, ne sera émis qu'après avoir émis deux fois m_1. De plus, m_3 et m_1 peuvent être en conflit avec m_2. Ce qui donne:

$D_{m3}= 2(C_{m1}+ C_{m2}+ \theta_{2^3}T_c)+ (C_{m3}+ C_{m2}+ \theta_{2^3}T_c)+C_{pmax}= 1{,}830\mu s$

3.1.2.3. Réseau CAN

Comme les messages ont des priorités distinctes, le message le plus prioritaire du système est unique ; de plus, tout message n'attend que l'émission des messages plus prioritaires dans le réseau. A partir des files de transmission locales des sites, le protocole CAN en choisissant toujours le message le plus prioritaire dans le réseau pour l'émettre, construit implicitement une file de transmission globale contenant tous les messages triés selon leurs priorités. On peut donc raisonner par rapport à cette file globale.

Chaque message a une priorité distincte. Si un message m est le plus prioritaire du système alors il est envoyé en premier.

$$D_m=C_m+C_{pmax}$$

C_{pmax}, comptabilisé dans D_m, correspond à une occupation du réseau par un paquet émis juste avant que le paquet le plus prioritaire ne soit envoyé. Il correspond à un temps maximal.
Par contre, si le message est moins prioritaire, il doit attendre la transmission des messages x plus prioritaires dans le système: $D_m= C_m + C_{p\max} + \sum_x C_x$.

Comme les périodes des messages sont quelconques, un message prioritaire x de période P_x peut être envoyé plus d'une fois avant l'émission du message m de période P_m.

$$D_m = C_m + C_{p\max} + \sum_x \left(\left\lceil \frac{P_m}{P_x} \right\rceil \right) \times C_x$$

Exemple

Reprenons l'exemple de la figure 5.3 et supposons que m_1 est plus prioritaire que m_2 lui même plus prioritaire que m_3, que nous notons $m_1 >> m_2 >> m_3$.

Avec un débit de 1Mb/s, C_{pmax}=130µs. Les valeurs sont données dans le tableau 5.4.
Calculons d'abord les temps de propagation :
$C_{m1}=C_{m2}$ =101µs et C_{m3} =130µs.

Calculons les délais de transmission :
m_1 étant le plus prioritaire : $D_{m1}= C_{pmax}+C_{m1}$ =231µs
$D_{m2}= C_{pmax}+C_{m2}+2C_{m1}$ =433µs
$D_{m3}= C_{pmax}+C_{m3}+C_{m2}+2C_{m1}$=563µs

P_{m1}=50, P_{m2}=100 et P_{m3}=100

3.1.2.4. Réseau FDDI

Soit une file de transmission telle que celle de la figure 5.4.

Cas 1

Le délai de transmission varie selon que les (s+p) paquets peuvent être émis dans la bande synchrone du site émetteur au premier passage du jeton ou non. En effet, si $\sum_{i=1}^{s+p} C_{pi} \leq S.TTRT$ (*S.TTRT* est la bande passante synchrone qui désigne la portion de *TTRT* dont dispose un site pour émettre des messages synchrones) alors le délai maximal de transmission est :

$$D_m=(1-S).TTRT+S.TTRT+ (n-1)C_{pmax}= TTRT+(n-1)C_{pmax}$$

si C_{pmax} est le temps maximal de transmission d'un paquet et n le nombre total de sites.

Tableau 5.4 : *Valeurs de C_m et D_m calculées dans le cas de CAN*

Message	$C_m=p*C_p$	D_m
m_1	101µs	231µs
m_2	101µs	433 µs
m_3	130µs	563 µs

Cette formule exprime le fait que le délai de transmission correspond au temps d'attente du jeton, plus le temps de transmission durant la bande passante allouée ainsi que le temps nécessaire pour atteindre la destination la plus éloignée. Cette destination est celle qui nécessite le parcours de (n-1) sites à partir de la source. Remarquons que le

jeton prend (1-*S*).*TTRT*+*S.TTRT*+*TTRT*-*S.TTRT*=2*TTRT*-*S.TTRT*≤2*TTRT* pour visiter deux fois le même site [SJ 87].

Cas 2

Par ailleurs, si $\sum_{i=1}^{s+p} C_{pi} > S.TTRT$, le délai de transmission est :

$$D_m = k.TTRT+(n-1)C_{pmax} \quad \text{avec} \quad k = \left\lceil \frac{\sum_{i=1}^{s+p} C_{pi}}{S \times TTRT} \right\rceil$$

[ACZD 92] montrent que le jeton prend *k.TTRT-S.TTRT*≤*k.TTRT* pour faire *k* visites consécutives (*k*≥2) du même site.

Exemple

L'application de la figure 5.3 est répartie sur *n* sites, *n* égal à 3. Si le débit du réseau est de 100Mb/s et *TTRT* est égale à 3ms telle que la tranche de temps allouée à chaque site est *S***TTRT*=*TTRT*/3=1ms alors les délais de transmission des messages sont les suivants pour un temps maximal de transmission d'un paquet égal à 360µs (voir tableau 5.5).

Comme le nombre d'octets composant chaque message est inférieur à 4490 octets alors les messages correspondent directement aux paquets.
Calculons les temps de propagation des messages :
$C_{m1}=C_{m2}$ =2,08µs
C_{m3}=2,32µs

Calculons les délais de transmission :
Dans le site 1, la file de transmission ne contient que m_2 correspondant à un seul paquet. Comme *s*=0, *p*=1, *S***TTRT*=1000µs et C_{pi}=2,08µs alors :
$\sum_{i=1}^{s+p} C_{pi} < S.TTRT$(cas 1) donc $D_{m1}=TTRT+(n-1)C_{pmax}$ c'est-à-dire :
D_{m1}= 3000µs+360µs=3,360ms.

Dans le site 3, m_1 peut être précédé par une seule occurrence de m_3 dans la file. D'où $\sum_{i=1}^{s+p} C_{pi} = C_{m1}+C_{m3}$ =4,4µs < *S.TTRT* ; D_{m1} =3,36ms. Par contre, m_3 peut être précédé par deux occurrences de m_1, donc par deux paquets.

Dans ce cas, *s*=2 et *p*=1; $\sum_{i=1}^{s+p} C_{pi} = 2C_{m1}+C_{m3}$ =6,48µs qui est plus petit que la bande passante allouée (*S***TTRT*=1ms).
$D_{m1}=D_{m2}=D_{m3}$=3,36ms.

Tableau 5.5 : *Valeurs de C_m et D_m calculées dans le cas de FDDI*

Message	$C_m=p*C_p$	D_m
m_1	2,08µs	3,36ms

m_2	2,08µs	3,36ms
m_3	2,32µs	3,36ms

3.1.2.5. Réseau TDMA

Par rapport aux schémas d'allocation présentés pour le réseau TDMA au chapitre 3, nous supposons dans cette étude que seul le schéma qui affecte une ou plusieurs trames contiguës pour chaque site est considéré.
Une entité appelée *générateur de trames* génère de manière cyclique des trames. Chaque cycle de transmission contient plusieurs trames allouées aux différents sites. Les trames restent réservées à un site même lorsqu'il n'émet rien.
A chaque site i sont allouées q_i tranches de temps. Si un message composé de p paquets doit être émis, alors il sera transmis en un temps $C_m=p$ tranches de temps. Une tranche de temps complète est allouée à un paquet même s'il ne la consomme pas entièrement. Pour calculer le délai de transmission D_m du message m, nous distinguons deux cas comme pour FDDI.

Soit la file de transmission de la figure 5.4.
Cas 1

Si $(s+p) \leq q_i$ alors D_m= attente + transmission =(1 cycle -q_i)+q_i=1 cycle.

Cas 2

Si $(s+p)>q_i$ alors soit $k = \left\lceil \frac{s+p}{q_i} \right\rceil$ => $D_m=k$ cycles.

Exemple

Supposons que l'application de la figure 5.3 utilise un réseau TDMA de débit 1Mb/s, d'un cycle de 16 tranches de temps. Le site 1 dispose, par exemple, d'un nombre de tranches de temps q_1 égal à 2, le site 2 de q_2 égal à 3 tranches de temps et le site 3 de q_3 égal à 2 tranches de temps. (voir tableau 5.6).

Sur le site 1, seul m_2 est émis c'est-à-dire p=1 et s=0. Comme C_{m2} est égal à 68µs, il n'excède la durée maximale d'un paquet. D_{m2}=1 cycle=16 tranches de temps.
Sur le site 3, m_1 peut être précédé par une seule occurrence de m_3 ; nous avons : s=1 et p=1 ; $2 \leq q_3$; $D_{m1}= D_{m2}$=1 cycle=16 tranches de temps.
m_3, par contre, peut être précédé par deux occurrences de m_1, d'où :
s=2 et p=1 ; $3>q_3$; soit $k=\left\lceil \frac{s+p}{q_i} \right\rceil$=2, D_{m3}=2 cycles=32 tranches de temps.

Tableau 5.6 : *Valeurs de C_m et D_m calculées dans le cas de TDMA*

Message	$C_m=p*C_p$	D_m
m_1	68µs	16 tranches de temps

m_2	68µs	16 tranches de temps
m_3	92µs	32 tranches de temps

Une meilleure assignation des tranches de temps pourra réduire les délais de transmission.

3.1.2.6. Conclusion

Dans notre méthode d'analyse, nous avons supposé que les périodes des tâches émettrice et réceptrice sont identiques. Néanmoins, remarquons que la période de la tâche réceptrice peut être un multiple de celle de la tâche émettrice. Dans ce cas, les messages émis ne sont pas tous consommés. En effet, si $P_r=k*P_e$ alors une seule occurrence du message est consommée toutes les k émissions. Le choix de la valeur de la période pour la tâche réceptrice sera alors motivé par le type d'applications à traiter car selon la pertinence du message échangé, véhiculant une information, ceci peut s'avérer suffisant ou dangereux (perte d'information).
Au contraire si la période de la tâche réceptrice est plus petite que celle de la tâche émettrice, la tâche réceptrice risque de consommer le même message plusieurs fois avant qu'une nouvelle instance de ce dernier ne soit arrivée.

Au terme du calcul présenté pour déterminer les caractéristiques temporelles des messages, chaque message m est clairement défini du point de vue temporel, $m=(r_m, C_m, D_m, P_e)$. Une seule donnée importante reste à déterminer, c'est l'instant r_m de génération de m par la tâche émettrice. Cet instant équivaut à l'instant d'insertion de m dans la file de transmission lorsque m est sans contrainte de gigue.

3.1.3. Calcul des dates d'insertion des messages

Selon les hypothèses relatives aux tâches, les messages sont générés en fin d'exécution des tâches émettrices. Le problème clé est de trouver cet instant puisqu'il correspond à r_m. La fin d'exécution de la tâche émettrice $\mathcal{E}$ est $r_e+C_e+B_e=r_m$, B_e est un facteur qui comptabilise tous les retards dus à l'interaction de $\mathcal{E}$ avec son contexte local. Il inclut deux types de retard:

- H_e dû à la préemption de $\mathcal{E}$ par des tâches plus prioritaires,
- R_e dû à l'attente de ressources critiques locales détenues par d'autres tâches moins prioritaires.

$$r_m= r_e+C_e+B_e = r_e+C_e+H_e+R_e$$

Le facteur de blocage, en dépendant de la préemption des tâches, dépend du type d'algorithme d'ordonnancement utilisé car la priorité d'une tâche varie avec le type d'algorithme. De plus, en s'exprimant en fonction du temps R_e, il dépend du protocole d'allocation de ressources utilisé puisque le calcul de R_e varie selon le type de protocole. Dans le paragraphe suivant, nous examinerons tous les cas possibles.

3.1.3.1. Le facteur de blocage dans le cas d'un ordonnancement local à priorité fixe

Nous désirons calculer le facteur de blocage dû à la préemption et à l'attente de ressources dans le cas d'un ordonnanceur local à priorité fixe des tâches tel que RM ou DM. Comme $B_e=H_e+R_e$, nous commençons par le calcul du terme H_e.

Calcul de H_e

- Dans RM, les tâches les plus prioritaires que $\mathcal{E}$ sont celles de périodes plus petites c'est-à-dire toutes les tâches T_x tel que $P_x<P_e$ et toutes les tâches T_y de même période que $\mathcal{E}$ mais de plus grande priorité affectée.

  $H_e = \sum_x \left\lceil \frac{D_e}{P_x} \right\rceil \times C_x + \sum_y C_y$ $\forall\ T_x$: $P_x<P_e$ $\forall\ T_y$: $P_y=P_e$ et priorité(T_y)>priorité($\mathcal{E}$) [TH 95]

- Dans DM, les tâches plus prioritaires que $\mathcal{E}$ sont celles de délais critiques plus petits:

  $H_e = \sum_x \left\lceil \frac{D_e}{P_x} \right\rceil \times C_x$ $\forall\ T_x : D_x<D_e$.

Nous passons au calcul du terme R_e qui désigne le temps d'attente, par une tâche, d'une ou de plusieurs ressources critiques partagées avec d'autres tâches sur le même site.

Calcul de R_e

Si le protocole d'allocation de ressources utilisé au niveau du site est le protocole à héritage de priorités, la durée maximale de blocage R_e de la tâche $\mathcal{E}$ par des tâches moins prioritaires s'exprime comme suit:

$$R_e = \max(l,m) * \max_{j=1,n-l} \{\text{durée de la section critique de } \mathcal{T}_j\}$$

Comme expliqué, au chapitre 2, dans la partie descriptive du protocole, l est le nombre de tâches moins prioritaires que $\mathcal{E}$, m le nombre de sections critiques qu'elle peut demander et n est le nombre total de tâches dans le site.
Si le protocole d'allocation de ressources est le protocole à priorités plafond alors:

$$R_e = \max_{j=1,l} \{\text{durée de la section critique de } \mathcal{T}_j\}$$

si $\mathcal{T}_1$, ..., $\mathcal{T}_l$ sont des tâches moins prioritaires que $\mathcal{E}$.

Exemple

Supposons que l'application de la figure 5.3 utilise des ressources critiques locales. Soient deux ressources critiques c_1 et c_2 disponibles sur le site 1 tels que A_1, A_2 et A_3 partagent la ressource c_1 et A_3, A_4 et A_5 la ressource c_2.
Sachant que seule la tâche A_3 est émettrice, calculer r_{m1}: $r_{m1}=r_{A3}+C_{A3}+B_{A3}$ revient à trouver d'abord B_{A3}.

$B_{A3}= H_{A3}+R_{A3.}$

Calculons H_{A3}:
Si l'algorithme d'ordonnancement utilisé est de type RM, alors A_1 et A_4 de période 60 sont plus prioritaires que A_3 de période 100.

$$H_{A3} = \sum_x \left\lceil \frac{D_{A3}}{P_x} \right\rceil \times C_x = \left\lceil \frac{D_{A3}}{P_{A1}} \right\rceil \times C_{A1} + \left\lceil \frac{D_{A3}}{P_{A4}} \right\rceil \times C_{A4} = 38$$

Le principe est le même si on utilise l'algorithme DM.

Calculons R_{A3}:
Si le protocole à héritage de priorités est utilisé alors R_{A3} est égal à 8 (voir exemple décrit dans le paragraphe 6.2.1 du chapitre 2).
Donc $R_{A3}= H_{A3} + B_{A3}= 38+8= 46$. Nous déduisons que $r_{m1}= r_{A3}+C_{A3}+B_{A3} =0+10+46 =56$.

Si au contraire on utilise le protocole à priorités plafond, R_{A3} est minimal et est égal à 4 (voir exemple décrit dans le paragraphe 6.2.2 du chapitre 2).
D'où, $B_{A3}=H_{A3}+R_{A3}=38+4=42$ et $r_{m1}=r_{A3}+C_{A3}+B_{A3}=0+10+42=52$.

3.1.3.2. Le facteur de blocage dans le cas d'un ordonnancement local à priorité dynamique

De manière similaire au paragraphe précédent, nous calculons le facteur de blocage $B_e=H_e+R_e$ pour une tâche émettrice lorsque l'ordonnanceur utilisé localement au niveau du site est dynamique tel que l'algorithme d'ordonnancement ED. Nous commençons par calculer le temps H_e dû à la préemption de la tâche émettrice par des tâches plus prioritaires.

Calcul de H_e

Dans ED, nous considérons toutes les tâches du site qui possèdent des échéances plus petites que celle de $\mathcal{E}$. Comme les échéances varient, il y a lieu de prendre toutes les tâches susceptibles d'interrompre $\mathcal{E}$ c'est-à-dire considérer le cas défavorable. Pour cela, nous classons toutes les échéances sur une fenêtre temporelle de largeur égale au PPCM des périodes P_i de toutes les tâches et nous déduisons la situation défavorable.

$$H_e = \sum_x \left\lceil \frac{P_e}{P_x} \right\rceil \times C_x \quad \forall T_x : d_x<d_e \text{ sur } [0, \text{PPCM}(P_i)]$$

Calculons maintenant le temps d'attente dû au blocage sur des ressources critiques.

Calcul de R_e

Si le protocole d'allocation de ressources est le protocole à pile, alors R_e est identique à l'expression calculée dans le cas du protocole à priorités plafond, c'est-à-dire:

$$R_e = \max_{j=1,l} \{\text{durée de la section critique de } \mathcal{T}_j\}$$

si $\mathcal{T}_1$, ..., $\mathcal{T}_l$ sont des tâches moins prioritaires que $\mathcal{E}$.

L'inversion de priorités se fait pour la durée d'une seule section critique. La priorité dans ce cas correspond au niveau de préemption défini par le protocole à pile puisque la priorité principale est dynamique.

Exemple

Reprenons les mêmes conditions de l'exemple précédent en considérant les mêmes ressources c_1 et c_2 dans le site 1 avec les mêmes durées des sections critiques pour les différentes tâches. En prenant ED comme algorithme d'ordonnancement local, il est nécessaire de déterminer d'abord les échéances initiales des tâches (du site 1) en prenant en compte la précédence locale selon des règles décrites dans le chapitre 2 et récapitulées dans le tableau 5.8. Dans cet exemple, nous nous contentons de donner les nouvelles valeurs des échéances (voir colonne d_i du tableau 5.7) sans montrer comment se fait le calcul car il a été expliqué dans le chapitre 2. Le but de cet exemple est de calculer le temps H_{A3} de préemption de A_3 lorsque l'algorithme d'ordonnancement local est ED.
La fenêtre temporelle est de largeur égale à l'intervalle de temps [0, PPCM(P_i)] où PPCM(P_i)=PPCM(50, 100, 150, 200)=3000. Dans cette fenêtre, nous classons les échéances des tâches en notant le nom de la tâche suivie de son échéance entre parenthèses comme suit :
A_1(41), A_4(50), A_3(100), A_1(101), A_4(110), A_2(150), A_1(161), A_4(170), A_3(200), A_5(200), A_1(221), A_4(230), A_1(281), A_4(290), A_3(300), A_2(350), A_5(400), ...
Remarquons que les tâches qui peuvent interrompre A_3 sont, dans le pire des cas, A_1, A_2 et A_4 car à un moment donné elles peuvent avoir des échéances plus petites que celle de A_3. D'où :

$$H_{A3} = \left\lceil \frac{P_{A3}}{P_{A1}} \right\rceil \times C_{A1} + \left\lceil \frac{P_{A3}}{P_{A2}} \right\rceil \times C_{A2} + \left\lceil \frac{P_{A3}}{P_{A4}} \right\rceil \times C_{A4} = 46$$

Comme l'algorithme d'ordonnancement est dynamique, il est intéressant d'utiliser le protocole à pile puisqu'il s'adapte bien à ce type d'algorithme. Supposons que le niveau de préemption est défini en fonction des délais critiques des tâches, autrement dit $\pi(T_i) > \pi(T_j)$ si $D(T_i) < D(T_j)$. Dans ce cas, les niveaux de préemption des tâches du site 1 sont ceux définis dans le tableau 5.7.

Tableau 5.7 : *Les caractéristiques des tâches du site 1 dans le cas de ED*

Tâche	r_i	C_i	*Di*	d_i	P_i	$\pi(Ti)$
A_1	0	10	50	41	60	5
A_2	0	8	150	150	200	2
A_3	0	10	100	100	100	3
A_4	10	9	50	50	60	4
A_5	8	5	200	200	200	1

L'algorithme de calcul de R_{A3} est identique à celui du protocole à priorités plafond, ce qui donne R_{A3}=4.
D'où $B_{A3}=H_{A3}+R_{A3}$=46+4=51 et $r_{m1}=r_{A3}+C_{A3}+B_{A3}$=0+51+10=61.
Dans une situation très favorable, A_3 peut être préemptée une seule fois uniquement par A_1 et A_4, ce qui donne :
$H_{A3}=C_{A1}+C_{A4}$=10+9=19 ; d'où : $B_{A3}= H_{A3}+ R_{A3}$ =19+ 4=23 et $r_{m1}=r_{A3}+C_{A3}+ B_{A3}$ = 0+23+10=33.

Selon l'algorithme d'ordonnancement local utilisé et le protocole d'allocation de ressources, nous venons de montrer comment calculer le facteur de blocage B_e afin de déduire la date de génération r_m d'un message m par la tâche $\mathcal{E}$: $r_m=r_e+C_e+B_e$

Au terme de ce calcul, m est clairement défini et est caractérisé par (r_m, C_m, D_m, P_m) avec :

- $r_m=r_e+C_e+B_e$,
- $C_m=p*C_p$ si m est constitué de p paquets,
- D_m calculé à partir des caractéristiques physiques du réseau,
- P_m égale à la période de la tâche émettrice.

3.2. Prise en compte de la précédence

Maintenant que les caractéristiques temporelles des tâches et des messages sont complètement définies, il y a lieu de prendre en compte la précédence globale en modifiant certains paramètres. On ne s'intéressera pas aux relations de précédence locale, au niveau des sites, qui sont supposées être déjà prises en compte à ce niveau. Suite à la constatation suivante : « les tâches réceptrices possèdent des dates d'arrivée qui ne correspondent pas nécessairement à l'instant de réception des messages attendus », la prise en compte de la communication consiste à modifier les dates d'arrivée en vue de réveiller les tâches correspondantes à des instants où l'on est sûr qu'un nouveau message est arrivé à destination et est prêt à être consommé.
Les relations de précédence locale et globale d'une application sont représentées à l'aide d'un graphe orienté appelé *graphe de précédence global* où les nœuds sont des tâches ou des messages de l'application et où les arcs sont les relations de précédence qui lient ces entités. La figure 5.5 montre un exemple de graphe de précédence global.

Le prochain paragraphe montre comment mettre à jour les dates d'arrivée des tâches réceptrices en vue de tenir compte de la précédence globale.

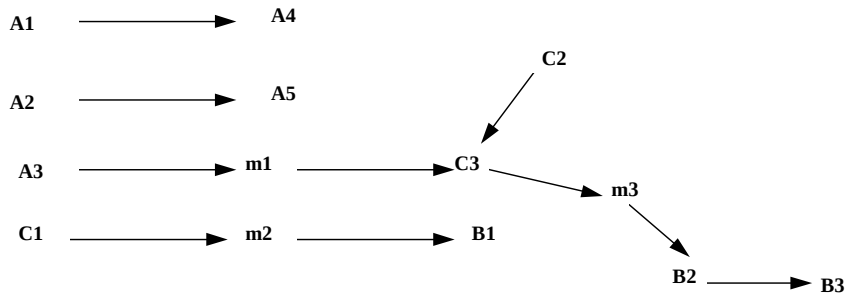


Figure 5.5: *Le graphe de précédence global de l'application de la figure 5.3*

3.2.1.Mise à jour des dates de réveil des tâches réceptrices

Dans toute application distribuée composée de tâches périodiques, nous avons toujours le triplet suivant:

tâche émettrice	→	message émis	→	tâche réceptrice
(r_e, C_e, D_e, P_e)		(r_m, C_m, D_m, P_e)		(r_r, C_r, D_r, P_e)

Il est nécessaire de modifier la valeur d'activation r_r de la tâche réceptrice si on veut réveiller cette dernière à un moment où l'on est sûr que *m* est reçu par le site destinataire. Si la tâche $\mathcal{R}$ attend aussi un message *m'* d'une autre tâche, $\mathcal{R}$ doit être réveillée de préférence à un moment où *m* et *m'* sont arrivés à destination. Il est donc clair que tout triplet (*tâche émettrice, message, tâche réceptrice*) ne suffit pas seul pour la mise à jour des dates de réveil des tâches réceptrices. Il est donc nécessaire de parcourir le graphe de précédence global afin de tenir compte de l'arrivée de tous les messages au niveau de chaque tâche réceptrice.

Réveiller la tâche réceptrice à un instant où l'on est sûr que le message *m* attendu est arrivé à destination signifie que la nouvelle date d'arrivée $r_r^*=\max\{r_r, r_m+D_m\}$ car dans le pire des cas, *m* est transmis au bout de r_m+D_m unités de temps. Si de plus, la tâche réceptrice attend un autre message *m'*, d'après le graphe de précédence global, alors: $r_r^*=\max\{r_r, r_m+D_m, r_{m'}+D_{m'}\}$. Dans le cas général, nous noterons :

$$r^*_r=\max\{(r_r, r_m+D_m)\}$$

avec r^*_r la nouvelle valeur de r_r.

3.2.2. Mise à jour des dates de réveil des tâches successeurs

Sur le graphe de précédence global, on peut avoir des tâches réceptrices qui précèdent d'autres tâches sur le même site, appelés *tâches successeurs* (dans la figure 5.3 par exemple, B_3 est la tâche successeur de la tâche B_2). Nous venons de voir que la prise en compte des messages nécessite de retarder les tâches réceptrices en modifiant leurs dates d'arrivée. Si de plus, elles précèdent d'autres tâches locales, il y a lieu de modifier les dates d'arrivée de ces tâches pour maintenir les relations de précédence. Cette modification ne peut s'opérer n'importe comment mais doit tenir compte de l'algorithme d'ordonnancement local. En effet, la prise en compte des relations de précédence, comme nous l'avons vu dans le chapitre 2, doit se faire selon les règles du tableau 5.8 si r_s est la date initiale de réveil de la tâche successeur et r_s* est sa nouvelle date de réveil.

Le tableau 5.8 récapitule les transformations à effectuer sur les paramètres temporels des tâches en vue de prendre en compte la précédence locale. Le principe est de retarder le réveil des tâches successeurs du graphe de précédence locale (condition 1), ce qui a pour conséquence de réduire les délais critiques des tâches (condition 2). Toutes les tâches liées par une relation de précédence possèdent la même période (condition 3) par hypothèse.

Tableau 5.8: *Règles de prise en compte des relations de précédence locale entre toute tâche $T_r=(r_r, C_r, D_r, P_r)$ précédant une tâche $T_s=(r_s, c_s, D_s, T_s)$*

Algorithme	condition 1	condition 2	condition 3
RM	r_s*=Max{(r_s, r_r*)}	affectation statique des priorités avec respect des précédences	$P_s=P_r$
DM	r_s*=Max{(r_s, r_r*)}	Min(D_i, D_s*)	$P_s=P_r$
ED	r_s*=Max{(r_s, r_r*+ C_r)}	d_r*=Min{(d_r, d_s*-C_s)}[CSB90]	$P_s=P_r$

(* signifie valeur mise à jour)

Les règles du tableau 5.8 doivent s'appliquer, de manière récursive, à chaque portion du graphe de précédence global relative à un site. En d'autres termes, ces règles concernent les successeurs directs des tâches réceptrices et les successeurs indirects (successeurs des successeurs). De plus, si une tâche émettrice est en même temps réceptrice ou successeur, le parcours du graphe global va provoquer la mise à jour de la date d'insertion du message émis notée r_m* déclenchant ainsi la mise à jour de la date de réveil de la tâche réceptrice impliquée, c'est-à-dire : $r_r^* = \max\{(r_r, r_m^* + D_m)\}$, ainsi de suite, ...

Le graphe de précédence global doit obligatoirement être acyclique afin que le calcul, basé sur un parcours récursif du graphe, puisse s'arrêter à un moment donné.

Exemple

Reprenons l'exemple de la figure 5.3 et considérons la séquence de tâches A_3, C_3, B_2 et B_3 qui s'exécutent dans l'ordre sur des sites différents en coopérant pour fournir un résultat (voir figure 5.6). Notons que dans le projet MARS, cette séquence est dite transaction.

L'échéance de bout en bout (l'échéance de toute la séquence) est respectée si chaque tâche de la séquence et chaque message échangé respecte son échéance.

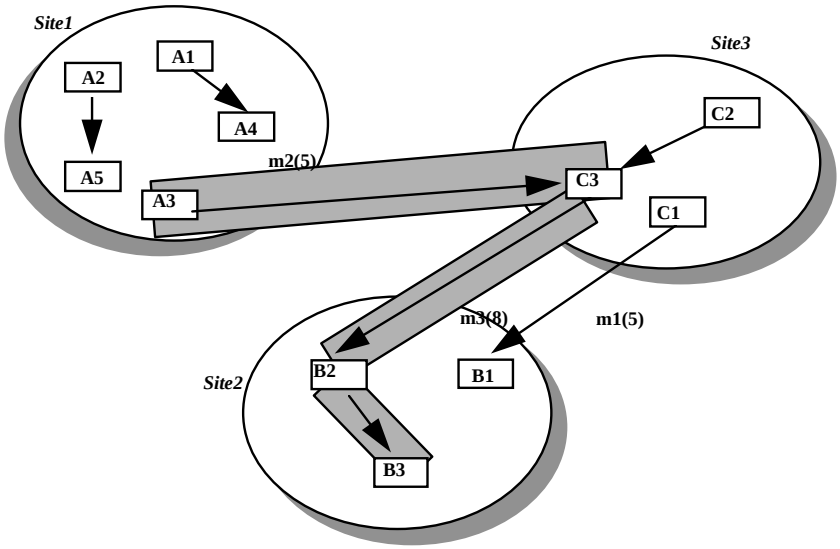


Figure 5.6 : *Exemple de séquence de tâches*

Nous allons donc appliquer la méthodologie en procédant par triplet (émetteur, message, récepteur) et en parcourant le graphe de précédence global. Pour le calcul des attributs temporels, nous considérons en premier RM ensuite ED tout en supposant un réseau CAN dans les deux situations.

1er cas L'algorithme d'ordonnancement utilisé est RM avec un protocole à priorité plafond

♦ Soit le triplet (A_3, m_2, C_3). Lorsque les valeurs s'expriment en ms, l'unité de mesure n'est pas spécifiée.

$\boldsymbol{A_3=(r_e=0, C_e=10, D_e=100, P_e=100)}$ et $C_3=(r_r=0, C_r=5, D_r=80, P_r=100)$.

C_3 a une date de réveil nulle identique à celle de C_2. La précédence est respectée avec l'algorithme RM car l'affectation de la priorité est faite de manière à ce que C_2 soit plus prioritaire.

Il faut déterminer les caractéristiques de m_2 :

- m_2 hérite de la période de l'émetteur A_3, d'où $P_{m2}=P_{A3}=100$
- les temps C_{m2} et D_{m2} sont calculés comme dans le paragraphe 3.1.2.3

  - $C_{m2}=101\mu s$

  - $D_{m2}=433\mu s$

- r_{m2} est déterminé en se basant sur le principe de l'algorithme d'ordonnancement local : $r_{m2}=r_{A3}+C_{A3}+B_{A3}$. $B_{A3}=H_{A3}+R_{A3}=52$ a été calculé précédemment dans l'exemple relatif au calcul du facteur de blocage (voir § 3.1.3.1). D'où $r_{m2}=0+10+42=52$ et $\boldsymbol{m_2=(r_m=52, C_m=101\mu s, D_m=433\mu s, P_m=100)}$

Comme C_3 est réceptrice de message, sa date de réveil est mise à jour :

$r^*_{C3}=\max(r_{C3}, r_{m2}+D_{m2})=52+0,43=52,43$ms ; d'où : $\boldsymbol{C_3=(r_r=52,43, C_r=5, D_r=80, P_r=100)}$.

♦ Considérons maintenant le triplet (C_3, m_3, B_2). Nous procédons de la même manière :

$C_3=(r_e=52,43, C_e=5, D_e=80, P_e=100)$ et $B_2=(r_r=0, C_r=5, D_r=80, P_r=100)$.

Caractéristiques de m_3 :

- m_3 hérite de la période de l'émetteur C_3, d'où $P_{m3}=P_{C3}=100$
- Avec le réseau CAN :

  - $C_{m3}=130\mu s$
  - $D_{m3}=563\mu s$

- $r_{m3}=r_{C3}+C_{C3}=B_{C3}$; comme $B_{C3}=H_{C3}$ car il n'y a pas de ressources sur le site 3. Dans RM, C_1 est plus prioritaire que C_2 (car sa période est plus petite) qui est plus prioritaire que C_3 (car C_2 précède C_3).

$$H_e=\left\lceil \frac{P_{C3}}{P_{C1}} \right\rceil \times C_{C1}+\left\lceil \frac{P_{C3}}{P_{C2}} \right\rceil \times C_{C2}=2C_{C1}+C_{C2}=12+5=17$$

D'où r_{m3}=52,43+5+17=74,43 et $\boldsymbol{m_3}$**=(**$\boldsymbol{r_m}$**=74,43,** $\boldsymbol{C_m}$**=130µs,** $\boldsymbol{D_m}$**=563µs,** $\boldsymbol{P_m}$**=100)**

Comme B_2 est réceptrice du message m_3, sa date de réveil est mise à jour :

$r_{B2} = r_{m3} + D_{m3}$ = 74,43 +0,56 =74,99ms ; d'où : $\boldsymbol{B_2}$**=(**$\boldsymbol{r_r}$**=74,99,** $\boldsymbol{C_r}$**=5,** $\boldsymbol{D_r}$**=80,** $\boldsymbol{P_r}$**=100)**.

♦ Au niveau du site 2, B_3=(0, 2, 100, 100) est la tâche successeur de B_2. En appliquant les règles de précédence, dans le cas de RM, r_{B3}*=max(r_{B3}, r_{B2})= max(0, 74,99)=74,99, d'où : $\boldsymbol{B_3}$**=(74,99, 2, 100, 100)**.

Les résultats sont récapitulés dans le tableau 5.9.

Tableau 5.9 : *Nouvelles caractéristiques des tâches et des messages pour l'application de la figure 5.3 dans le cas de RM*

Site / Réseau	Tâche / Message	r^*_i *(ms)*	C_i *(ms)*	D_i *(ms)*	P_i *(ms)*
Site1	A_1	0	10	50	60
	A_2	0	8	150	200
	A_3	0	10	100	100
	A_4	0	9	50	60
	A_5	0	5	200	200
Site2	B_1	6,23	5	50	50
	B_2	74,99	5	80	100
	B_3	74,99	2	100	100
Site3	C_1	0	6	50	50
	C_2	0	5	80	100
	C_3	52,43	5	80	100
réseau CAN	m_1	6	0,101	0,231	50
	m_2	52	0,101	0,433	100
	m_3	74.43	0,13	0,563	100

<u>2er cas L'algorithme d'ordonnancement utilisé est ED au niveau de chaque site</u>

♦ Prise en compte de la précédence locale au niveau du site 3 :

C_2 précède C_3. Selon les règles de transformation relatives à ED (voir tableau 5.8), les dates de réveil et les délais critiques des deux tâches doivent être modifiées en vue de tenir compte de la précédence locale. Initialement des caractéristiques des tâches C_2 et C_3 sont les suivantes :

C_2=(r=0, C=5, D=80, P=100) et C_3=(r_r=0, C_r=5, D_r=80, P_r=100)

Calculons les nouvelles dates de réveil :

r_{C2}*=r_{C2}=0
Comme C_3 est un successeur : r_{C3}*=max(r_{C3}, r_{C2}+C_{C2})=max(0, 0+5)=5

Calculons les échéances sachant que d_{C3}=D_{C3}+r_{C3}=80 et d_{C2}=D_{C2}+r_{C2}=80.

d_{C3}*=d_{C3}=80
d_{C2}*=min(d_{C2}, d_{C3}*-C_{C3})=min(80, 80-5)=75
D'où : D_{C2}*=d_{C2}*-r_{C2}*=75-0=75 et D_{C3}*=d_{C3}*-r_{C3}*=80-5=75

D'où : **C_2=(r=0, C=5, D=75, P=100) et C_3=(r_r=5, C_r=5, D_r=75, P_r=100)**

♦ Soit le triplet (A_3, m_2, C_3) :

A_3=(r_e=0, C_e=10, D_e=100, P_e=100) et C_3=(r_r=5, C_r=5, D_r=75, P_r=100).

Il faut déterminer les caractéristiques de m_2 :

- m_2 hérite de la période de l'émetteur A_3, d'où P_{m2}= P_{A3}=100
- Avec le réseau CAN :
  - C_{m2}=101µs
  - D_{m2}=433µs
- r_{m2} est déterminé en se basant sur le principe de l'algorithme d'ordonnancement local : r_{m1}=r_{A3}+C_{A3}+B_{A3}. B_{A3}=H_{A3}+R_{A3}=55 a été calculé précédemment dans l'exemple relatif au calcul du facteur de blocage (voir § 3.1.3.2). D'où r_{m2}=0+10+55=65 et **m_2=(r_m=65, C_m=101µs, D_m=433µs, P_m=100)**

Comme C_3 est réceptrice de message, sa date de réveil est mise à jour :

r_{C3}*=max(r_{C3}, r_{m2}+D_{m2})=max(5, 65+0,43)=65,43ms ; d'où : **C_3=(r_r=65,43, C_r=5, D_r=75, P_r=100)**.

♦ Considérons maintenant le triplet (C_3, m_3, B_2). Nous procédons de la même manière :

C_3=(r_e=65,43, C_e=5, D_e=80, P_e=100) et B_2=(r_r=0, C_r=5, D_r=75, P_r=100).

Caractéristiques de m_3 :

- m_3 hérite de la période de l'émetteur C_3, d'où P_{m3}=P_{C3}=100
- Avec le réseau CAN : C_{m3}=130µs, D_{m3}=563µs.
- r_{m3}=r_{C3}+C_{C3}=B_{C3}. B_e=H_e car il n'y a pas de ressources sur le site 3. H_e= H_{C3}=$\left\lceil \frac{P_{C3}}{P_{C1}} \right\rceil \times C_{C1}$=2$C_{C1}$ =12 car seul C_1 peut préempter C_3 étant donné

que C_3 a été déjà retardée pour succéder à C_2. D'où r_{m3}=65,43+5+12=82,43 et $\boldsymbol{m_3=(r_m=82,43,\ C_m=130\mu s,\ D_m=563\mu s,\ P_m=100)}$

Comme B_2 est réceptrice du message m_3, sa date de réveil est mise à jour : r^*_{B2}=max(r_{B2}, r_{m3}+D_{m3})=82,43+0,56=82,99ms ; d'où : $\boldsymbol{B_2=(r_r=82,99,\ C_r=5,\ D_r=80,\ P_r=100)}$.

- Au niveau du site 2, B_3=(0,2,100,100) est la tâche successeur de B_2. En appliquant les règles de précédence, dans le cas de ED, r_{B3}*=max(r_{B3}, r_{B2}*+C_{B2})=max(0, 82,99+5)=87,99, d'où : $\boldsymbol{B_3=(87,99,\ 2,\ 100,\ 100)}$.

Les résultats sont récapitulés dans le tableau 5.10.

Tableau 5.10 : *Nouvelles caractéristiques des tâches et des messages pour l'application de la figure 5.3 dans le cas de ED*

Site / Réseau	Tâche / Message	r^*_i (ms)	C_i (ms)	D_i (ms)	P_i (ms)
Site1	A_1	0	10	50	60
	A_2	0	8	150	200
	A_3	0	10	100	100
	A_4	0	9	50	60
	A_5	0	5	200	200
Site2	B_1	16,23	5	50	50
	B_2	82,99	5	80	100
	B_3	82,99	2	100	100
Site3	C_1	0	6	50	50
	C_2	0	5	75	100
	C_3	65,43	5	75	100
réseau CAN	m_1	16	0,101	0,231	50
	m_2	65	0,101	0,433	100
	m_3	82,43	0,13	0,563	100

3.3. Ordonnancement des tâches et des messages

Une fois que toutes les caractéristiques temporelles sont modifiées pour la prise en compte de la précédence globale, l'ordonnancement des tâches est fait en utilisant un algorithme, parmi ceux que nous avons décrit (RM, DM ou ED), identique pour tous les sites et l'ordonnancement des messages est fait en fonction du protocole de communication (CSMA/DCR, CAN, FDDI, TDMA ou FIP). Un diagramme de Gantt (voir figure 5.7) est tracé pour chaque site et pour le réseau.

Le diagramme de Gantt montre l'utilisation du processeur et l'activité du réseau sur un cycle de temps qui commence à l'instant min(r_i) et finit à l'instant max(r_i)+PPCM(P_i) lorsque P_i sont les périodes des différentes tâches de l'application et r_i leurs dates d'activation. Pour des raisons d'espace, le diagramme de la figure 5.7 ne représente pas toute la durée de simulation qui est égale à 75690+PPCM(60, 200, 100, 50)=675690µs. A l'aide du diagramme de Gantt, nous vérifions le respect des échéances pour les tâches et pour les messages afin de déduire la validité de l'application. Dans une application temps réel à contraintes strictes, si une tâche ou un message dépasse son échéance, l'application n'est pas ordonnançable.

4. Exemples d'applications temps réel réparties

Nous traitons deux exemples d'application, le premier est basé sur le modèle producteur/consommateur, le second, d'ordre plus pratique, concerne l'exemple de l'ascenseur.

4.1. Exemple du producteur/consommateur

Nous avons choisi un exemple simple du modèle producteur/consommateur qui montre qu'avec la même période du producteur et du consommateur, tout message produit est consommé. Un tampon à une seule case suffit pour mémoriser chaque message si nous supposons que les flux de production et de consommation sont les mêmes. A travers cet exemple, nous montrons aussi ce qu'est la préemption des messages à l'échelle des paquets car la plus petite entité indivisible transmise sur le réseau est le paquet. Soit alors l'exemple de la figure 5.8 dans lequel les tâches *A* et *C* placées sur sites distincts sont productrices respectivement des messages m_1 et m_2 consommés respectivement par les tâches *B* et *D* placées sur des sites distincts.

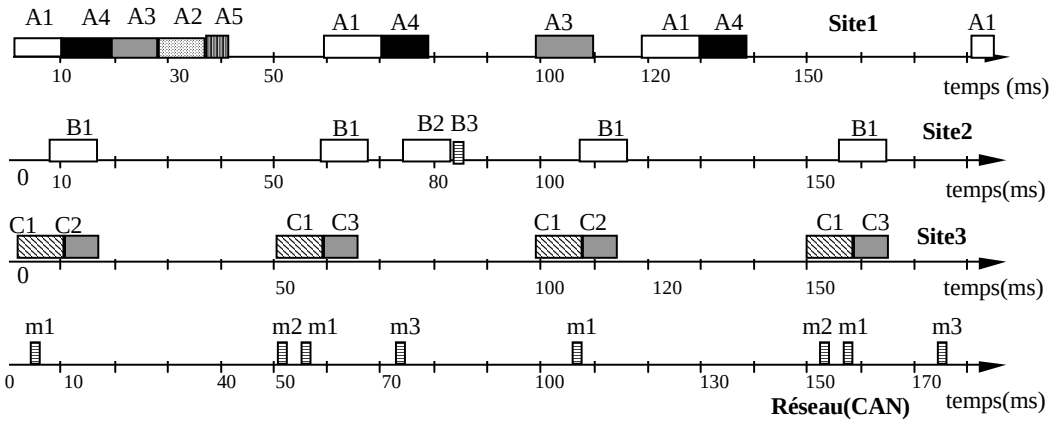


Figure 5.7: *Le diagramme de Gantt de l'application de la figure 5.3*

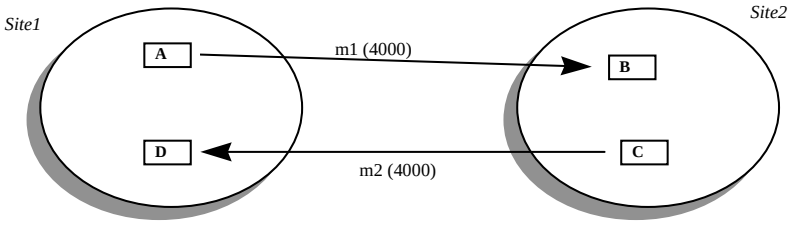


Figure 5.8 : *Exemple de producteurs/consommateurs*

Les caractéristiques temporelles des tâches sont données dans le tableau 5.11.

Utilisons le réseau CSMA/DCR avec les caractéristiques suivantes : débit=10Mb/s et T_c=50µs. En cas de collision, le protocole de communication doit résoudre les conflits en se basant sur les priorités des sites en considérant le site 1 plus prioritaire que le site 2.

4.1.1.Calcul des paramètres temporels des messages

♦ Calculons les temps de propagation C_m des messages :
L'information utile d'un paquet CSMA/DCR est de 1518octets, par conséquent m_1 et m_2 sont composés chacun de 3 ($=\left\lceil \frac{4000}{1518} \right\rceil$) paquets dont deux comportent 1518 octets de données et un contient 964 octets (4000-2*1518).
$C_{m1}=C_{m2}=\sum C_{pi}$=2013µs.

♦ Calculons maintenant le délai de transmission D_m pour chaque message :
Au niveau de chaque site émetteur, un seul message *m* peut être émis ; autrement dit il n'y a pas de message dans la file de transmission pouvant précéder *m* (*s*=0).

Au niveau du site 1 :

m_1 peut entrer en conflit avec m_2 puisque ce dernier est émis par un autre site.

$$D_{m1}= C_p + \sum_{i=1}^{s+p}(C_{pi} + \sum_{j=1}^{n-1} C_{pj} + (n-1)T_c) \text{ avec } n=2,\ p=3 \text{ et } s=0$$

D_{m1}=8135µs

Au niveau du site 2 :

m_2 se comportant de manière symétrique à m_1 : $D_{m2}= D_{m1}$=8135µs.

♦ Déduisons la période P_m pour chaque message :

La période du message est celle de la tâche productrice, c'est-à-dire $P_{m1}=P_A$=10000µs et $P_{m2}=P_C$=10000µs

Tableau 5.11: *Caractéristiques temporelles des tâches de la figure 5.8 (exprimées en µs, r_i=0 pour toutes les tâches)*

Site	Tâche	C_i	D_i	P_i
Site1	*A*	500	10000	10000
	D	400	10000	10000

Site2	*C*	400	10000	10000
	B	500	10000	10000

♦ Calculons la date d'insertion r_m pour chaque message :

Supposons que l'algorithme d'ordonnancement des tâches au niveau de chaque site est RM avec *A* plus prioritaire que *D* dans le site 1 et *C* plus prioritaire que *B* dans le site 2.
$R_{m1}=r_A+C_A+B_A=r_A+C_A+H_A+R_A$. H_A est nul car *A* est la tâche la plus prioritaire du site par conséquent elle ne peut être préemptée. R_A est aussi nul car il n'existe pas de ressources sur le site. D'où : $r_{m1}=r_A+C_A$=500µs.
De même, $r_{m2}=r_C+C_C+B_C=r_C+C_C$=400µs ($B_C$=0 pour les mêmes raisons que précédemment).

Le tableau 5.12 récapitule toutes les caractéristiques des messages.

4.1.2. Prise en compte de la précédence globale

Au niveau local des sites, il n'y a pas de précédence locale, seule la communication est à prendre en compte. *B* et *D* sont des tâches consommatrices respectivement sur le site 2 et le site 1 :

$r^*_B=\max(r_B, r_{m1}+D_{m1})=\max(0, 500+8135)=8635$

$r^*_D=\max(r_D, r_{m2}+D_{m2})=\max(0, 400+8135)=8535$

4.1.3. Ordonnancement des tâches et des messages

L'ordonnancement est fait sur l'intervalle [*deb*, *fin*] tels que *deb*=min(r_i)=0 et *fin*= max(r_i)+ PPCM(P_i) =8635+ 10000 =18635, pour toute tâche du système $T_i=(r_i, C_i, D_i, P_i)$.
Pour des raisons d'espace, la séquence d'ordonnancement des tâches et des messages est représentée, par la figure 5.9, sur une durée inférieure à l'intervalle [*deb*, *fin*]=[0, 18635].

Tableau 5.12 : *Caractéristiques des messages pour l'application de la figure 5.8*

Message	r_i (µs)	C_i (µs)	D_i (µs)	P_i (µs)
m_1	500	2013	8135	10000
m_2	400	2013	8135	10000

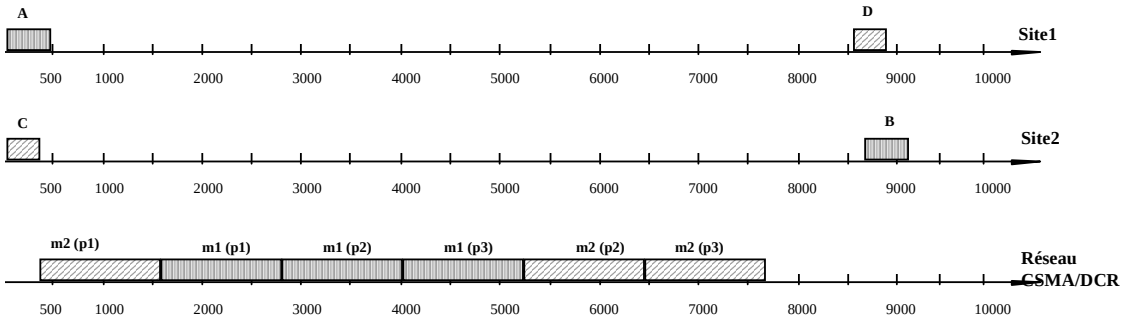


Figure 5.9 : *Le diagramme de Gantt pour l'exemple de la figure 5.8*

La figure 5.9 montre un cas de préemption de messages. En effet, les tâches *A* et *C* qui sont émettrices de messages s'exécutent simultanément sur des sites différents. Comme *C* se termine la première, elle envoie m_2 qui commence à être transmis sur le réseau. Lorsque *A* se termine, m_1 est inséré dans la file de transmission locale en attente de la libération du réseau. A la fin de la transmission du premier paquet de m_2, les paquets de m_1 et les paquets restants de m_2 sont émis par leurs sites. Ils rentrent en collision. La résolution de la collision se fait en autorisant le site le plus prioritaire à émettre c'est-à-dire le site 1. Les paquets de m_1 sont, par conséquent, émis avant les deux paquets restants de m_2 : nous assistons à une préemption.

Si nous augmentons la taille du message émis, par exemple, par *A*, nous constatons qu'au delà d'une taille équivalente à trois paquets c'est-à-dire 3*1518=4554 octets, le système n'est pas ordonnançable. En effet, si par exemple, m_1 a une taille de 4700 octets, ce qui équivaut à 4 paquets CSMA, alors le délai de transmission correspondra à D_{m1} égal à 10155µs qui est supérieur à la période de la tâche consommatrice. Dans ce cas, selon la méthodologie appliquée, cette tâche doit être réveillée après ce délai pour consommer le message m_1. Or, par hypothèse, la période de la tâche consommatrice est de 10 000µs c'est-à-dire inférieure à ce délai. Nous concluons que ce système reste ordonnançable jusqu'à une taille des messages n'excédant pas 4554 octets en garantissant que tout message produit est consommé.

4.2. Exemple de l'ascenseur

Dans le cadre du groupe de travail « Réseaux et Temps Réel » du GDR-PRC PRS, il a été choisi une application de référence afin de pouvoir exercer et comparer les différents modèles et méthodes utilisés par les équipes de recherche participant à ce groupe de travail. Cette application possède les deux caractéristiques essentielles à cette étude, c'est-à-dire application temps réel à contraintes strictes et application répartie. L'application choisie concerne *la gestion d'un ensemble d'ascenseurs* et est issue d'un exemple donné dans [Gom 93]. Notre contribution a été d'appliquer notre méthodologie à cette application.
Dans ce paragraphe, nous commençons d'abord par présenter le cahier des charges de cette application et ensuite par spécifier les différentes tâches afin de procéder à son analyse temporelle.

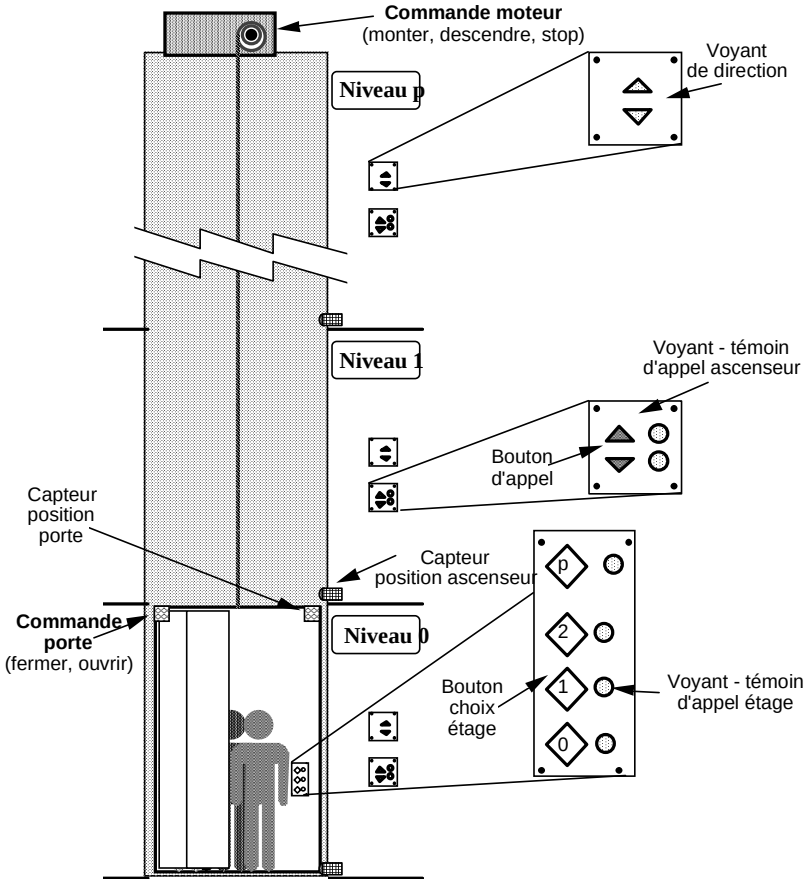


Figure 5.10 : *Exemple d'une colonne ascenseur*

4.2.1. Cahier des charges

L'application concerne la gestion d'un nombre *n* d'ascenseurs devant desservir *p* étages.

4.2.1.1. Description des ascenseurs

Chaque ascenseur est muni de *p* boutons poussoirs (un bouton par étage) permettant de choisir le numéro de l'étage auquel on veut se rendre. A chaque bouton est associé un voyant (voir figure 5.10). Ces voyants sont allumés automatiquement. Le système ne devra donc commander que leur extinction.
La position des ascenseurs est repérée par l'intermédiaire de capteurs de positions (un capteur par étage et par ascenseur). Ces capteurs délivrent un signal d'interruption au passage de l'ascenseur.
Le contrôle du mouvement des ascenseurs s'effectue en agissant directement sur le moteur de l'ascenseur. Le moteur répond aux trois commandes suivantes :

- monter
- descendre
- stop

On peut également agir sur la porte de chaque ascenseur par l'intermédiaire des commandes :

- ouvrir porte
- fermer porte.

4.2.1.2. Description des étages

Comme dans la figure 5.10, chaque étage est pourvu de deux boutons d'appel : l'un pour monter, l'autre pour descendre. Les étages inférieurs et supérieurs n'ont qu'un seul bouton d'appel.
Un voyant est associé à chacun des boutons. Ces voyants sont allumés automatiquement. Le système ne devra donc commander que leur extinction.
Chaque étage est également pourvu de deux voyants de direction par ascenseur. Les étages inférieurs et supérieurs n'ont qu'un seul voyant de direction par ascenseur. Ces voyants indiquent si l'ascenseur qui se dirige vers l'étage est entrain de monter ou de descendre. Le système devra prendre en charge l'allumage et l'extinction de ces voyants.

4.2.1.3. Contraintes temporelles

Contrainte critique

Les signaux provenant de capteurs de position doivent être pris en compte par le système dans un délai maximum de 50ms.

Contraintes relatives

- L'intervalle minimum entre deux pressions sur un bouton de sélection d'étage à l'intérieur de l'ascenseur est de 250ms.
- Les appels provenant des boutons d'appel des étages sont au minimum espacés de 200ms.

4.2.2.Étude de l'application

4.2.2.1. Spécification des sous-systèmes

Les entités extérieures avec lesquelles le système est en relation sont (voir figure 5.11):

* les capteurs de position
* le moteur des ascenseurs
* la porte des ascenseurs
* les boutons de sélection d'étages des ascenseurs
* les voyants associés aux boutons de sélection d'étage
* les boutons d'appel
* les voyants associés aux boutons d'appel
* les voyants de direction

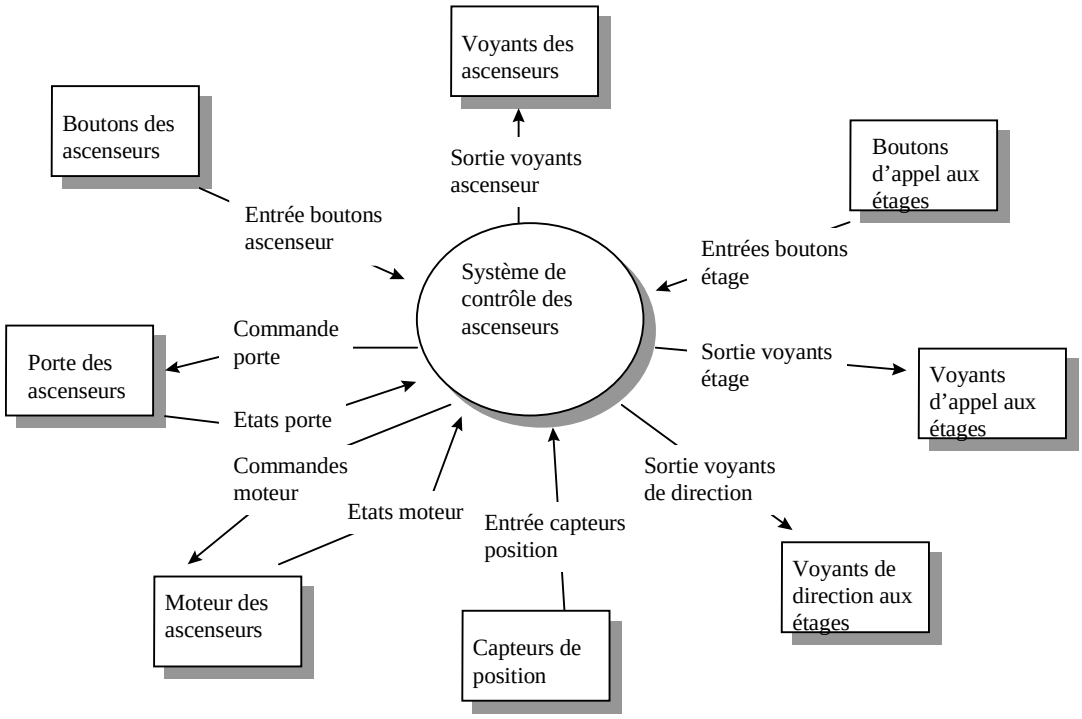


Figure 5.11 : *Architecture globale de l'application*

Le système se décompose en trois sous-systèmes (voir figure 5.12):

- le sous-système « ascenseur »
- le sous-système « étage »
- le sous-système « contrôleur application ».

Le sous-système « ascenseur » prend en charge tous les services relatifs à l'ascenseur : il reçoit les données en provenance des capteurs de position et des boutons ascenseurs, il s'occupe de la commande de la porte et du moteur, il gère les voyants associés aux boutons ascenseur et informe les différents sous-systèmes de l'état de l'ascenseur. Le sous-système « étage » s'occupe de la gestion des voyants étages et des voyants de direction, ainsi que de la réception des données provenant des boutons étages, dont il informe le sous-système « contrôleur d'application ». Le sous-système « contrôleur application » est chargé de choisir l'ascenseur le plus apte à répondre à une demande donnée.

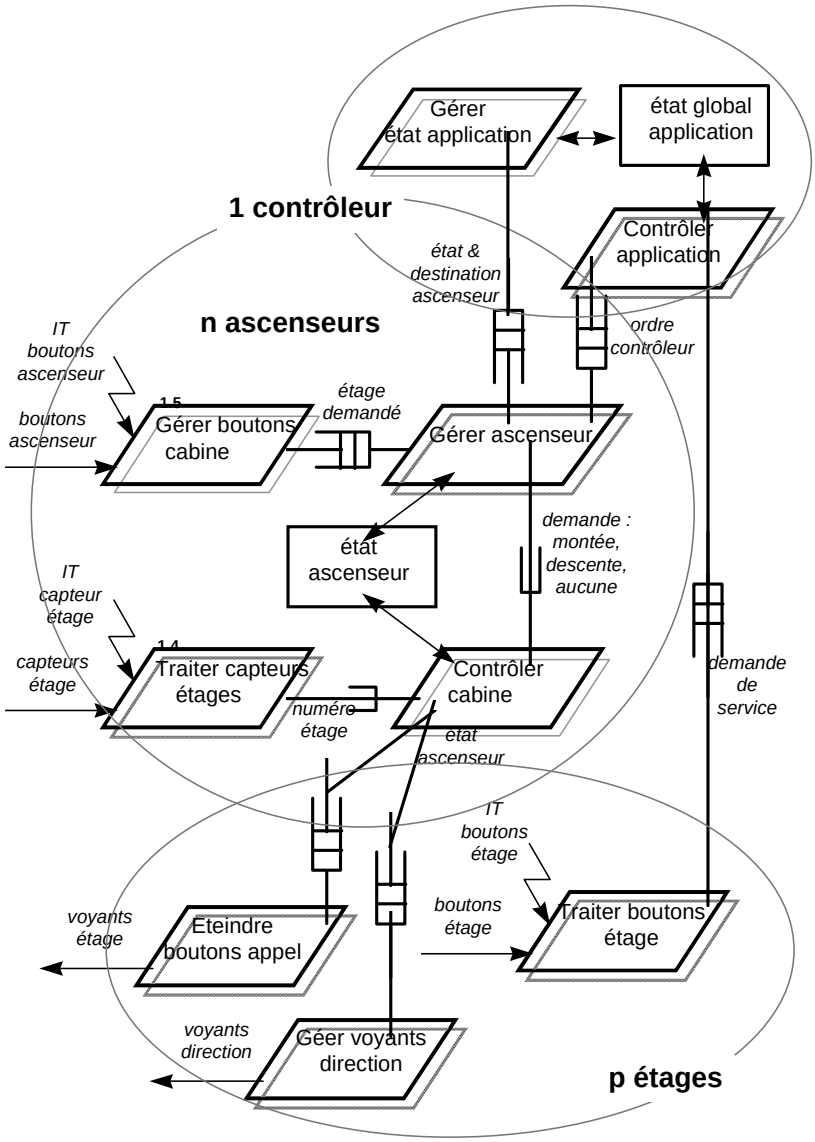


Figure 5.12 : *Architecture en tâches de l'exemple de l'ascenseur*

4.2.2.2. Spécification des tâches

Une étude approfondie des sous-systèmes nous permet d'obtenir les tâches, de la figure 5.12, pour lesquelles nous reprenons les caractéristiques temporelles définies dans la spécification de Gomaa [Gom 93] (voir tableau 5.13). La tâche *GVE* du sous-système étage, donnée dans le tableau 5.13, regroupe les tâches *Eteindre boutons appel* et *Gérer voyants direction* présentées dans le schéma de la figure 5.12. *Etat ascenseur* est une donnée partagée par les tâches *Gérer ascenseur* et *Contrôler cabine* du sous-système ascenseur, comme son l'indique, elle désigne l'état de l'ascenseur. De même, *état global application* est une donnée, partagée par les tâches du contrôleur, qui mémorise l'état de tous les ascenseurs qu'il gère.

Les tâches sont synchrones c'est-à-dire activées à l'instant 0. Les priorités des tâches (voir P_r du tableau 5.13) sont données selon un algorithme à priorité fixe tout en respectant la précédence locale. Nous supposons une architecture réseau multi-site dans

laquelle chaque étage est sur un site, chaque ascenseur est sur un site et le contrôleur sur un autre site. Chaque message est composé de 3 octets de données.

4.2.3. Application de la méthodologie d'analyse

Notre analyse temporelle a été faite sur un exemple de deux ascenseurs à deux étages chacun (voir figure 5.13). Chaque sous-système *ascenseur*$_i$ communique avec ses sous-systèmes étages *Etage*$_{1i}$ et *Etage*$_{2i}$, l'ensemble coopérant avec le système *contrôleur*.
Les tâches émettrices et réceptrices dont les messages sont présentés dans la figure 5.13 sont données au tableau 5.14.

Tableau 5.13: *Caractéristiques temporelles des tâches des différents sous-systèmes*

Tâches synchrones (r_i=0)	C_i(ms)	D_i=P_i (ms)	Pr_i
Sous-système ascenseur:			
- Gérer Boutons Cabine : *GBC*	3	100	4
- Gérer Ascenseur: *GA*	6	100	2
- Traiter Capteurs Étages: *TCE*	2	50	3
- Contrôler Cabine: *CC*	5	50	1
Sous-système étage:			
- Traiter Boutons Étage: *TBE*	4	200	2
- Gérer Voyants Étages : *GVE*	10(5+5)	200	1
Sous-système contrôleur:			
- Gérer État Application: *GEA*	2	50	2
- Contrôler Application: *CA*	20	200	1

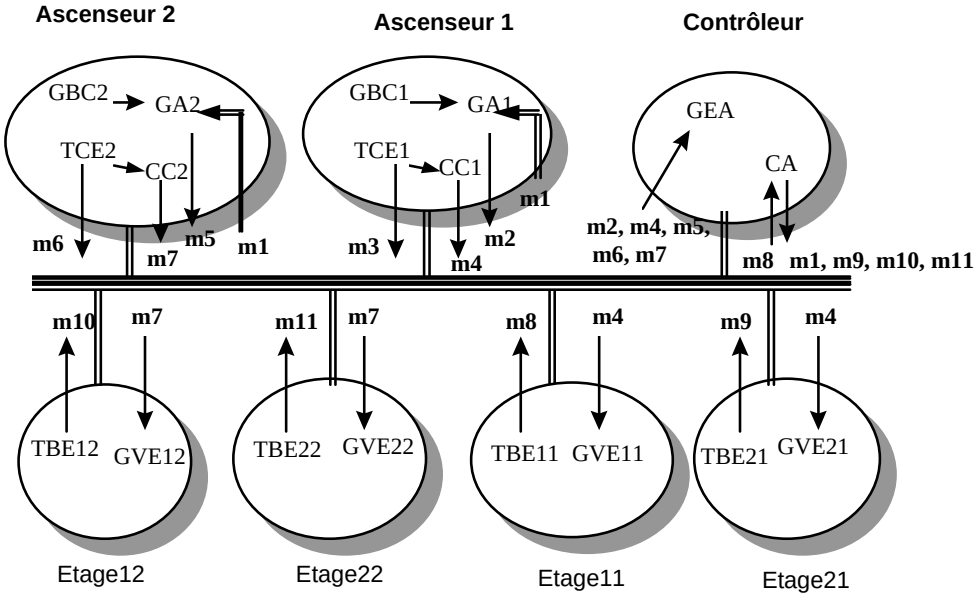


Figure 5.13: *Cas d'une application à deux ascenseurs*

Tableau 5.14 : *Les messages échangés*

Message	tâche émettrice	site émetteur	tâche réceptrice	site récepteur
m1	*CA*	Contrôleur	*GA1* *GA2*	Ascenseur1 Ascenseur2
m2	*GA1*	Ascenseur1	*GEA*	Contrôleur
m3	*TCE1*	Ascenseur1	*GEA*	Contrôleur
m4	*CC1*	Ascenseur1	*GEA* *GVE11* *GVE21*	Contrôleur Étage11 Étage21
m5	*GA2*	Ascenseur2	*GEA*	Contrôleur
m6	*TCE2*	Ascenseur2	*GEA*	Contrôleur
m7	*CC2*	Ascenseur2	*GEA* *GVE12* *GVE22*	Contrôleur Étage12 Étage22
m8	*TBE11*	Étage11	*CA*	Contrôleur
m9	*TBE21*	Étage21	*CA*	Contrôleur
m10	*TBE12*	Étage12	*CA*	Contrôleur
m11	*TBE22*	Étage22	*CA*	Contrôleur

Tableau 5.15 : *Caractéristiques temporelles des messages (en µs) si CAN*

Message	C_m	D_m	P_m	P_{rm}
m1	82	1032	200 000	1
m2	82	212	100 000	11
m3	82	376	50 000	9
m4	82	540	50 000	7
m5	82	294	50 000	10
m6	82	458	50 000	8
m7	82	622	50 000	6
m8	82	704	200 000	5
m9	82	786	200 000	4
m10	82	868	200 000	3
m11	82	950	200 000	2

En appliquant la méthodologie, nous déterminons d'abord les caractéristiques des messages. Dans le cas d'un réseau CAN avec un débit de 1Mb/s, les caractéristiques temporelles présentées au tableau 5.15 sont obtenues pour des priorités de messages spécifiées dans la colonne Pr_m.
Dans le cas d'un réseau FDDI, avec *TTRT*=5000µs et un débit de 100Mb/s, les caractéristiques des messages sont différentes (voir tableau 5.16). Le calcul est fait en affectant, aux sites ascenseurs et contrôleur, des bandes passantes plus importantes que celles des sites étages car ils émettent plus de messages.
S_i.*TTRT* pour les sites contrôleur et ascenseur: 1089µs
S_i.*TTRT* pour les sites étages: 360µs.

En parcourant le graphe de précédence global (voir figure 5.14), les dates d'insertion des messages sont calculées (voir tableau 5.17) et une mise à jour des dates de réveil des tâches réceptrices est faite (voir tableau 5.18).
Une dernière étape consiste à ordonnancer les tâches sur les sites propriétaires et les messages sur le médium. Pour des raisons d'espace, nous ne pouvons tracer le diagramme de Gantt. Néanmoins, nous montrons, grâce à l'outil qui implémente cette méthodologie, que cette application est valide, c'est-à-dire les contraintes de temps spécifiées dans le cahier des charges sont bien respectées.
Le diagramme de Gantt a été tracé pour une durée de simulation égale à [0, 244 948] dans le cas du réseau CAN et égale à [0, 258 000] dans le cas du réseau FDDI. Les contraintes de temps spécifiées dans le cahier des charges sont respectées puisque toutes les tâches respectent leurs échéances.

Tableau 5.16 : *Caractéristiques temporelles des messages si FDDI en µs*

Message	C_m	D_m	P_m
m1	192	5000	200 000
m2	192	5000	100 000
m3	192	5000	50 000
m4	192	5000	50 000
m5	192	5000	50 000
m6	192	5000	50 000
m7	192	5000	50 000
m8	192	5000	200 000
m9	192	5000	200 000
m10	192	5000	200 000
m11	192	5000	200 000

Tableau 5. 17 : *Dates d'insertion des messages en µs*

Message	r_m(CAN)	r_m(FDDI)
m1	32704	37000
m2	44736	53000
m3	5000	5000
m4	16000	16000
m5	44736	53000
m6	5000	5000
m7	16000	16000
m8	4000	4000
m9	4000	4000
m10	4000	4000
m11	4000	4000

Tableau 5.18 : *Nouvelles dates de réveil pour les tâches réceptrices en µs*

Tâches réceptrices ou successeurs	r_i (CAN)	r_i (FDDI)
Site contrôleur		
- *GEA*	44948	58000
- *CA*	4704	9000
Site ascenseur1		
- *GA1*	33736	42000
Site ascenseur2		
- *GA2*	33736	42000
Site étage		
- *GVE11*	16540	21768
Site étage		
- *GVE21*	16540	21768
Site étage		
- *GVE12*	16540	21768
Site étage		
- *GVE22*	16540	21768

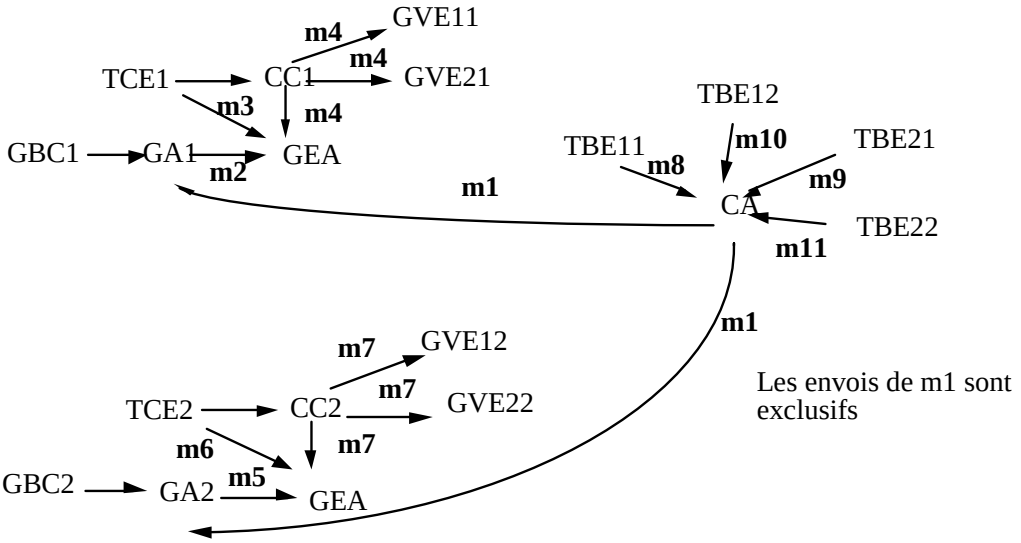


Figure 5.14: *Le graphe de précédence de l'application ascenseur*

5. Conclusion

A travers ce chapitre, nous avons développé une méthodologie pour analyser et valider les applications temps réel distribuées à contraintes strictes. Nous avons défini des modèles temporels analogues pour les tâches et les messages. Nous avons supposé que les attributs temporels des tâches étaient définis par le concepteur de l'application, il restait donc à calculer ceux des messages. Deux paramètres particuliers (le temps de propagation et le délai de transmission) dépendent étroitement du réseau choisi. A travers cinq exemples de protocoles de communication, nous avons montré comment calculer ces paramètres. Les relations de précédence sont ensuite dérivées par modification des attributs des tâches et des messages conduisant ainsi à une configuration de tâches et de messages indépendants. La figure 5.15 illustre, dans le temps, les instants pertinents calculés tout au long de ce chapitre, pour un triplet (tâche émettrice, message échangé, tâche réceptrice).

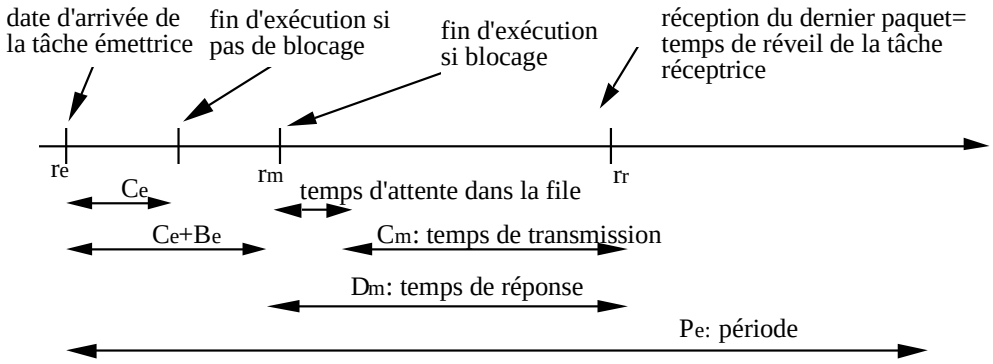


Figure 5.15: *La communication de bout en bout*

Il s'agit alors d'ordonnancer :

- les tâches sur chaque site en appliquant l'algorithme d'ordonnancement choisi avec le protocole d'allocation de ressources approprié, et

- les messages sur le réseau selon la stratégie du protocole de communication choisi.

L'objectif est de vérifier si les tâches et les messages ne dépassent pas leurs échéances dans le cas d'une application temps réel répartie à contraintes strictes.
Cette phase de validation est très importante dans le sens où elle peut donner au concepteur une garantie ou non quant à l'ordonnançabilité de son application avant qu'il ne procède à son implémentation et éventuellement ajuster les paramètres d'entrée tels que ceux des tâches ou du réseau.
A travers cette méthodologie, nous venons de montrer que l'ordonnançabilité d'une application temps réel répartie à contraintes strictes dépend non seulement de l'ordonnançabilité des tâches mais aussi de celle des messages. L'ordonnançabilité des tâches dépend de l'algorithme d'ordonnancement utilisé localement sur le site et du protocole d'allocation de ressources alors que l'ordonnançabilité des messages dépend des paramètres physiques du réseau et du protocole de communication gérant l'accès au réseau.

Le prochain chapitre définit un certain nombre de critères pour analyser les performances du système en se basant sur les séquences d'exécution relatives aux sites et au réseau obtenues à l'aide de l'outil MOSARTS qui implante cette méthodologie.

CHAPITRE 6

ANALYSE DES PERFORMANCES

1. Introduction

Ce chapitre présente l'outil MOSARTS que nous avons développé et qui implante la méthodologie de validation décrite dans le chapitre précédent.

Après une brève présentation de MOSARTS, les différents critères de performance pour les tâches et les messages sont exposés. Ensuite les résultats de simulation qui montrent les temps de réponse obtenus en fonction des différents algorithmes d'ordonnancement et des protocoles de communication concluent ce chapitre. A travers ces résultats, nous montrons que l'ordonnançabilité de toute application temps réel répartie dépend étroitement de l'algorithme d'ordonnancement et du réseau utilisés.

2. MOSARTS : un outil de validation d'applications temps réel réparties

MOSART est un outil d'aide à la validation d'applications temps réel réparties basé sur la méthodologie que nous avons proposée. J'ai développé, en langage C, tous les programmes qui implémentent cette méthodologie en incluant les trois types d'ordonnancement (RM, DM, ED) pour les sites ainsi que les stratégies de communication (FIP, CSMA/DCR, CAN, FDDI, TDMA) pour le réseau. D'autres personnes ont contribué au développement de MOSARTS : deux étudiants de l'université de Clausthal (Allemagne) ont réalisé l'interface en Tcl/Tk (voir annexe1) et un étudiant de maîtrise informatique de l'université de Poitiers a contribué au calcul des critères de performance des messages.

L'interface graphique permet de saisir les différents paramètres de l'application (voir figure 6.1), les caractéristiques physiques du réseau ainsi que les paramètres liés au protocole de communication choisi. Les principaux menus de l'interface graphique

servant à la saisie des données et à la visualisation des résultats figurent dans l'annexe 2.

2.1. Description de l'application

La description de l'application consiste à définir chaque site et le réseau sur lesquels est implantée l'application.

2.1.1. Description d'un site

Un site est défini par :

- les tâches de l'application qui lui sont allouées,
- les ressources dont il dispose,
- l'ordonnanceur local de son exécutif.

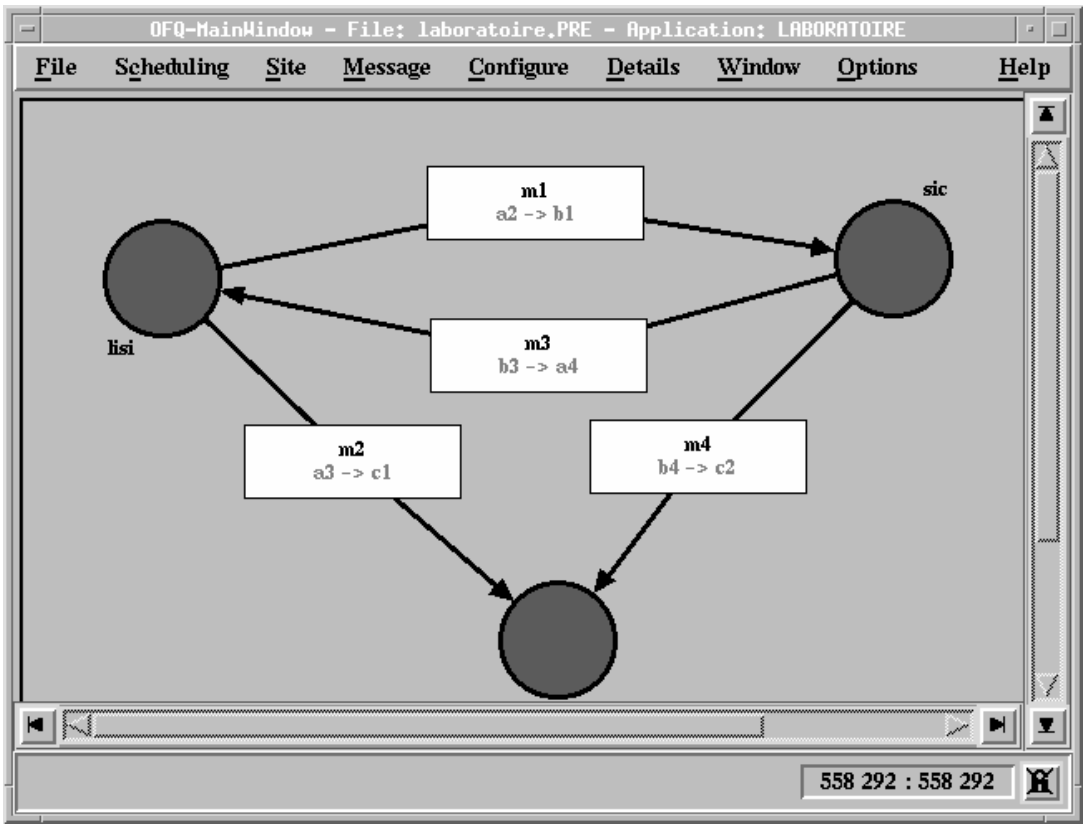


Figure 6.1: *Description d'un exemple d'application*

2.1.1.1. Les Tâches

L'utilisateur saisit les différentes caractéristiques temporelles numériques (r_i, C_i, D_i, P_i) des tâches pour chaque site. Si un site est doté de ressources, on définit chacune d'elles par son nom et on associe, à chaque tâche, éventuellement une durée d'utilisation de cette ressource. Les relations de précédence locale sont représentées à l'aide d'arcs allant des tâches vers leurs successeurs dans le site. Le graphe de précédence doit être obligatoirement acyclique.

2.1.1.2. L'ordonnancement local

L'étude de l'ordonnançabilité d'une application temps réel distribuée nécessite de choisir un algorithme d'ordonnancement particulier qui sera le même au niveau de tous les sites. Il sera associé à un protocole d'allocation de ressources.

MOSARTS implémente les cas suivants :

- RM avec PHP[4]
- RM avec PPP[5]
- DM avec PHP
- DM avec PPP
- ED avec PAP[6]

Les priorités des tâches sont affectées automatiquement par l'algorithme d'ordonnancement choisi.

2.1.2. Description du réseau

Le réseau est décrit par

- les messages échangés,
- les caractéristiques physiques du réseau : le débit, la taille du paquet, la bande passante allouée (FDDI), etc.
- le protocole de communication.

2.1.2.1. Les messages

L'utilisateur définit chaque message en lui associant une tâche émettrice sur un site et une tâche réceptrice sur un autre site. Il ne fournit que la taille du message en octets car les caractéristiques temporelles seront calculées automatiquement par MOSARTS.

[4] Protocole à héritage de priorités
[5] Protocole à priorités plafond
[6] Protocole à pile

Les priorités des messages peuvent être définies selon le protocole de communication utilisé. En effet, elles sont données explicitement par l'utilisateur dans CAN, elles sont confondues avec celles des sites dans CSMA/DCR, elles dépendent des bandes passantes allouées aux sites dans FDDI, ainsi que de la disposition des sites dans l'anneau et dépendent du nombre de tranches de temps allouées à chaque site et de la position de chaque site par rapport au générateur de trames dans TDMA. Par contre, dans FIP, l'utilisateur ne fournit aucune information relative à la priorité car les messages sont triés, dans la table de scrutation, dans l'ordre croissant de leurs périodes et sont transmis dans cet ordre.

2.1.2.2. Le protocole de communication

MOSARTS possède des caractéristiques physiques, par défaut, pour chaque réseau mais l'utilisateur peut les modifier. Les protocoles MAC implantés sont : CAN, FDDI, CSMA/DCR, FIP et TDMA.

2.2. Le noyau fonctionnel de MOSARTS

Une fois l'application complètement décrite, une séquence de programmes implantant la méthodologie est lancée. Elle correspond à l'ensemble des modules schématisés dans la figue 6.2.

Le noyau implante toutes les étapes présentées dans le chapitre 5 c'est-à-dire :

- *La modélisation de l'application* : calcul des attributs temporels des messages et mise à jour de ceux des tâches en prenant en compte les relations de précédence locale.

- *La prise en compte de la précédence globale* : parcours du graphe de précédence globale en vue de prendre en compte la communication en calculant les dates d'insertion des messages et en mettant à jour les dates de réveil des tâches réceptrices et successeurs.

- *L'ordonnancement* : ordonnancer les tâches de tous les sites en appliquant le même algorithme d'ordonnancement et ordonnancer les messages sur le réseau en appliquant la technique MAC choisie.

- *L'analyse des performances* : des critères de performance sont calculés pour les tâches et d'autres pour les messages en déduisant les taux d'utilisation des sites et du réseau ainsi que la validité de l'application dans sa globalité. En fonction des résultats de performance obtenus à ce niveau, l'utilisateur pourra modifier les paramètres d'entrée et relancer le noyau pour analyser les nouveaux résultats. En effet, il peut changer les caractéristiques temporelles des tâches, les tailles des messages, le protocole de communication et les caractéristiques physiques du réseau.

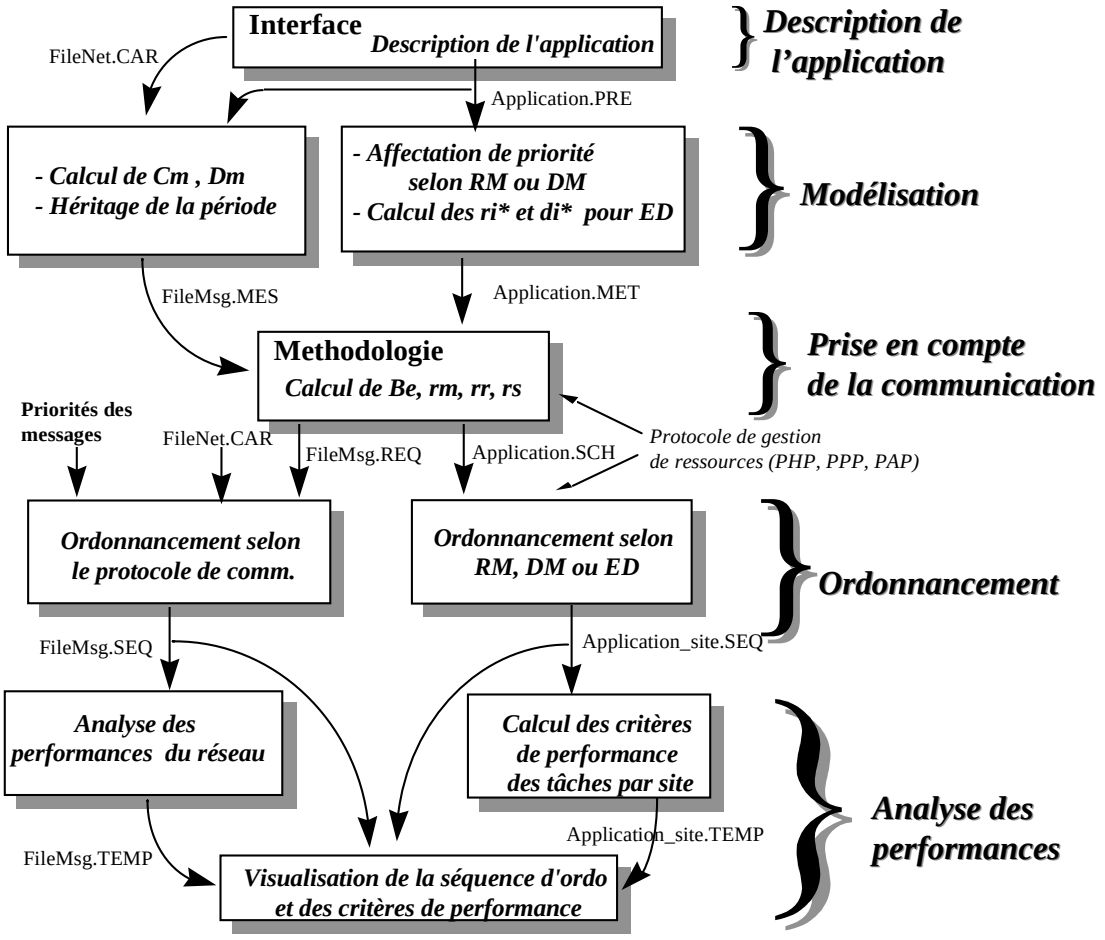


Figure 6.2: *Le noyau de MOSARTS*

Les trois premières étapes ont été détaillées dans le chapitre précédent. Il reste à s'intéresser aux critères de performance définis pour les sites et le réseau. En effet, l'étape concernant l'ordonnancement nous trace le diagramme de Gantt relatif à chaque site et au réseau en montrant les instants occupés et les instants libres du processeur ou du réseau comme dans la figure 6.3. Ce diagramme est un résultat brut qu'il convient d'analyser pour connaître la pertinence des résultats, selon des critères de performance que nous avons développés.

3. Les critères de performance mesurés

3.1. Pour les tâches

Le diagramme de Gantt est construit pour toute la durée de simulation qui est délimitée par les instants initial et final où :

* instant initial = Min (r_i), r_i date d'activation de toute tâche du système

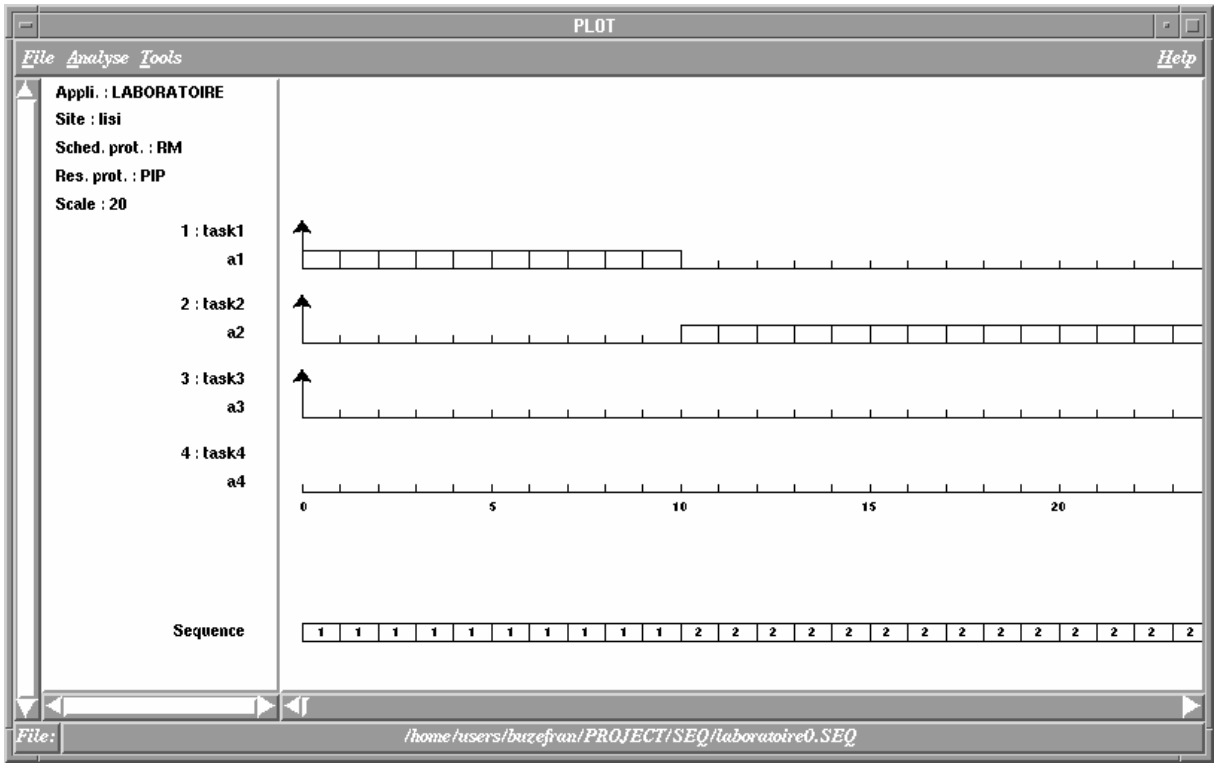


Figure 6.3: *La séquence d'exécution d'un site*

* instant final = Max (r_i) + PPCM(P_i), P_i la période de toute tâche du système.

Au delà de l'instant final, le système se retrouve globalement dans le même état qu'au début.

Les différents critères de performance que fournit l'outil MOSARTS sont présentés dans la figure 6.4. Ils sont calculés pour les tâches, pour le réseau et pour l'application globale. Les critères de performance relatifs aux tâches ont été initialement calculés pour des applications temps réel monoprocesseur [Bab 96] et ont été repris et intégrés dans MOSARTS.

a. ***Temps de réponse***

On définit le temps de réponse d'une tâche par l'intervalle de temps qui sépare son instant de réveil de la fin de son exécution (figure 6.5). Sur une séquence d'exécution relative à un site, on évalue classiquement les temps de réponse minimal (TR_{min}), moyen (TR_{my}) et maximal (TR_{max}) d'une tâche. Dans une séquence valide, on doit toujours avoir la relation [KRP 94 *et al.*] suivante pour une tâche périodique *i*:

$$C_i \leq TR_i \leq D_i \text{ soit } TR_{max} \leq D_i$$

Le calcul de ce paramètre a une grande utilité surtout lorsque les tâches périodiques correspondent à des serveurs qui traitent des tâches apériodiques car, dans ce cas, le temps de réponse nous informe sur le temps moyen nécessaire pour servir une tâche apériodique.

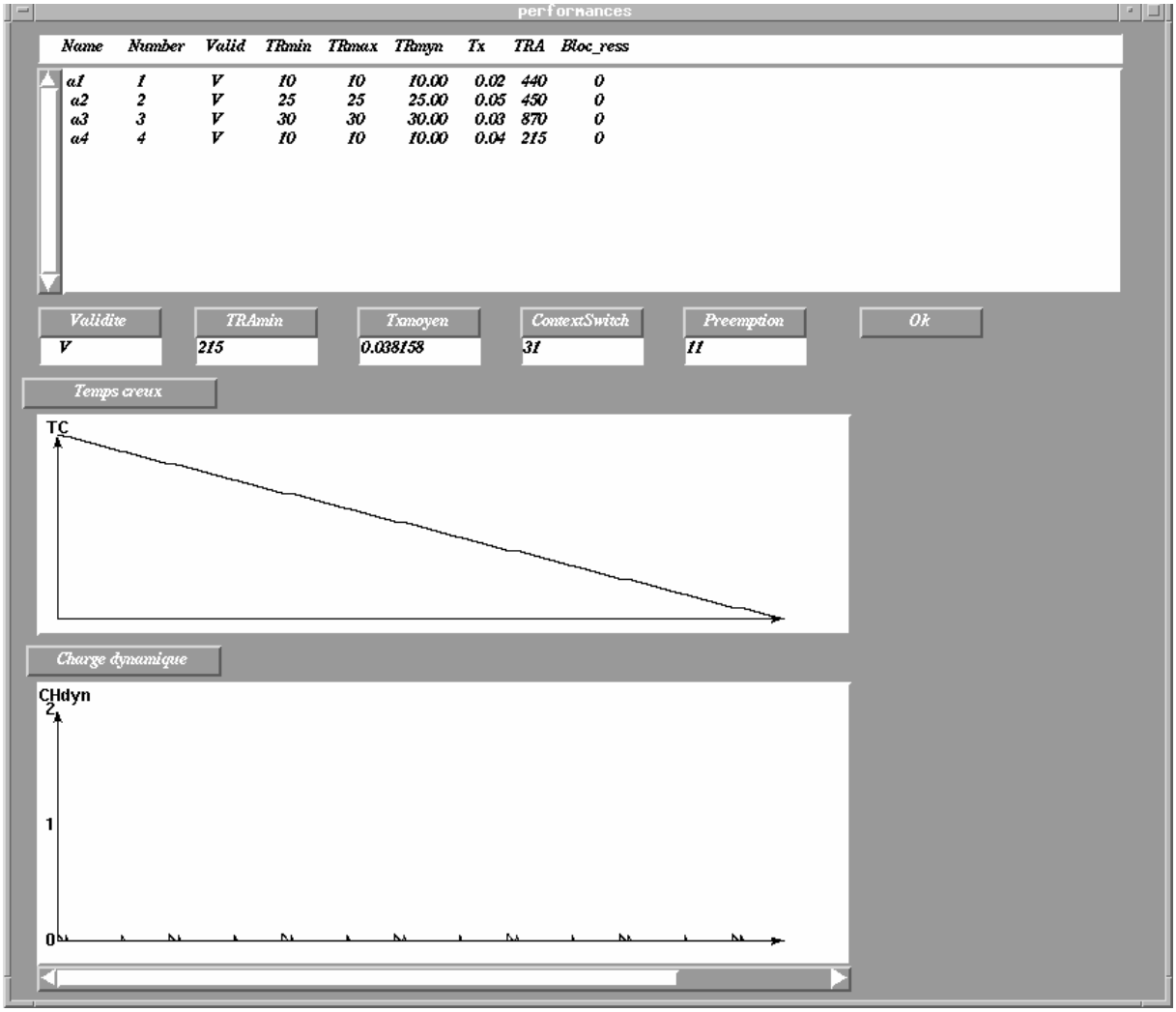


Figure 6.4: *Les critères de performance des tâches dans MOSARTS*
Valid : validité d'une tâche, TRmin, TRmy, TRmax : temps de réponse minimal, moyen et maximal, Tx : taux de réaction pour une tâche, TRA : temps de retard minimal admissible pour une tâche, Bloc_ress : temps de blocage dû aux ressources, TRAmin : le temps de retard admissible pour la configuration de tâches, Txmoyen : taux de réaction moyen pour la configuration de tâches, ContextSwich : nombre de changement de contexte sur la durée de simulation, Preemption : nombre de préemptions sur la durée de simulation, TC : Temps creux, Chdyn : charge dynamique

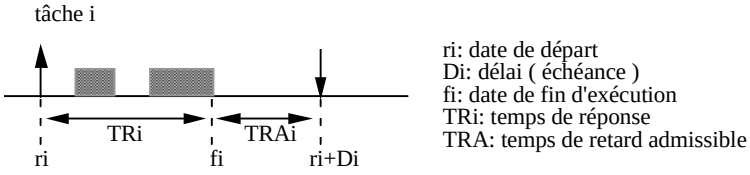
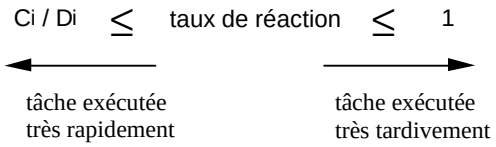


Figure 6.5: *Evaluation du temps de réponse d'une tâche* [Bab 96]

b. *Taux de réaction*

On définit le taux de réaction par le temps de réponse d'une tâche, rapporté à son délai critique. On s'intéresse aux taux de réaction moyen et maximal.

Le taux de réaction étant relatif à une tâche, on peut alors déterminer celui des différentes tâches afin d'en déduire un taux moyen pour le site. Ce paramètre nous indique si les tâches s'exécutent plus ou moins rapidement ou pas au niveau d'un site.



taux de réaction = $\frac{f_i - r_i}{D_i} = \frac{TR_i}{D_i}$

c. ***Temps de retard admissible***

Il s'agit de la durée qui sépare la fin de l'exécution de la tâche de son échéance (voir figure 6.5). Cette durée est importante pour mesurer la capacité de la configuration d'un site à accepter une surcharge momentanée. Elle est définie comme suit : pour une tâche périodique *i* d'un site:

$$TRA_i = D_i - TR_{imax}$$

De même, on définit un temps de retard minimal au niveau d'un site :

$$TRA_{min} = \text{Min}(TRA_i)$$

Si le temps de retard minimal est assez grand pour accepter une surcharge au niveau d'un site, on ne peut pas déduire que cela n'altérera pas l'ordonnancement de l'application dans sa globalité car rajouter de l'activité au niveau d'un site peut retarder l'exécution des tâches émettrices, par conséquent l'émission des messages et leur réception sur un autre site.

d. ***Charge dynamique***

On représente la charge du processeur par ce paramètre, et ce, à chaque instant de la durée de la simulation. Elle est définie comme étant la somme des durées restantes des tâches non terminées rapportées à leurs délais critiques dynamiques:

$$\text{charge} = \sum_{\text{les tâches non terminées}} \frac{C_{i\,\text{restant}}}{D_i(t)}$$

charge=0 $\Rightarrow$ processeur (site) libre
charge=1 $\Rightarrow$ processeur (site) totalement chargé

Le délai critique dynamique $D_i(t)$ d'une tache T_i est le temps séparant l'instant courant t de la prochaine échéance de la tâche non terminée. Elle est égale à l'expression suivante :

$$D_i(t) = D_i + r_i - t = d_i - t$$

La connaissance de cette charge dynamique sur un site nous permet, à nouveau mais sous un nouveau point de vue, de mesurer l'aptitude d'une séquence à supporter l'ajout d'une nouvelle tâche. Tout comme le cas du temps de retard admissible, si la charge nous permet de dire que la possibilité de l'ajout d'une tâche n'affecterait en rien la validité du site, il n'en est pas de même au niveau de l'application globale.

e. ***Temps creux dynamique***

Ce paramètre est calculé pour chaque site et à chaque instant. C'est le nombre d'instants restants dans la séquence où le processeur n'exécute aucune tâche.

$$TC = durée\_totale - t_{courant} - \sum_{pour\ toute\ tache\ i\ du\ site} C_{i\,\text{restant}}$$

$t_{courant}$ est calculé à partir du début de la séquence.

f. ***Nombre de changement de contexte***

Nous avons jusqu'ici négligé le temps qu'impliquent les changements de contexte au niveau du processeur. Ce qui n'est pas une erreur si l'on inclut ce dernier dans la durée d'exécution des tâches. Le nombre de changements de contexte est donc le nombre de fois que le processeur a changé de tâche à exécuter. Ce paramètre est essentiel à mesurer, car s'il est élevé, il peut générer une perte de temps non négligeable du fait de la sauvegarde du contexte de la tâche interrompue et le chargement du contexte de la nouvelle tâche à exécuter, qui dépasserait le temps inclus dans la durée d'exécution des tâches. A ce propos, nous pouvons noter que nous avons négligé ce temps de changement de contexte dans cette étude. Cette hypothèse, vérifiée dans la plupart des cas, doit être remise en cause si le nombre de changement de contexte est important.

g. ***Nombre de préemptions***

Il s'agit du nombre de fois qu'une tâche plus prioritaire a pu préempter une autre de moindre priorité. Cela entraîne un retard dans l'exécution de cette dernière.
Il peut être intéressant de connaître le nombre de préemptions pour pouvoir limiter celles-ci par une redéfinition des priorités, afin de minimiser le temps de réponse. Ceci s'avérant intéressant pour l'ajout de tâches supplémentaires. Mais ce changement de priorité devrait se faire par le changement des périodes ou des délais critiques dans le cas d'algorithme comme RM ou DM, ce qui implique le changement du comportement de la tâche.

h. ***Nombre de dépassement et validité***

Ce nombre caractérise une séquence non valide. Il comptabilise le nombre de dépassements d'échéances des tâches. Le dépassement d'échéance pour une tâche est détectée lorsque l'échéance est atteinte sans toutefois que la tâche ait fini de s'exécuter. Pour éviter des fautes en cascade et corrélées avec la première faute, on impose à ce que l'arrivée d'une nouvelle requête de la tâche annule la requête précédente même si l'exécution de cette dernière n'est pas achevée. Suivant l'importance des tâches, les dépassements d'échéance peuvent ou non mettre en péril la sûreté de fonctionnement du système. Le moindre dépassement d'échéance implique la non validité de toute application à contraintes strictes.

i. Temps de retard sur blocage dû aux ressources

Même si une tâche interrompt d'autres tâches et occupe le processeur, car elle est plus prioritaire, elle peut se voir bloquée par une autre tâche lorsqu'elle demande une ressource déjà occupée par l'une des tâches préemptée en attente du processeur. La tâche prioritaire sera désormais bloquée en attente de la libération de la ressource désirée pendant un temps dit "temps de blocage" qui est minimisé grâce à l'utilisation de protocoles d'allocation de ressources (voir figure 6.6).

3.2. Pour les messages

De manière similaire aux tâches, nous définissons les critères suivants récapitulés dans MOSARTS comme dans la figure 6.7.

a. ***Le temps de réponse***
Trois temps de réponse sont calculés pour un message : le temps de réponse minimal (TR_{min}), maximal (TR_{max}) et moyen (TR_{my}). Comme pour les tâches, le temps de réponse d'un message est le temps qui sépare l'instant de son insertion dans la file de transmission de la fin de sa transmission sur le réseau. Ce temps inclut le temps dû à son attente dans la file et celui qui est dû à sa préemption par d'autres messages comme le montre la figure 6.8.

b. ***Le nombre de préemptions***
Le nombre de préemptions est le nombre de fois qu'un message prioritaire a préempté un message moins prioritaire. Rappelons que la préemption d'un message se fait à l'échelle du paquet. La priorité est associée directement au message dans CAN et chaque paquet du message hérite de cette priorité. Elle est confondue avec celle du site émetteur dans CSMA/DCR puisque le conflit se produit entre les messages de sites distincts et les messages du même site sont émis dans l'ordre de leur insertion. La notion de priorité n'a pas de sens dans TDMA ni FDDI car les protocoles associés n'utilisent pas cette notion même s'il arrive qu'un message d'un site *i* entame sa transmission avant même qu'un message d'un site *j* qui a occupé le réseau n'ait fini sa transmission (seuls les premiers paquets sont émis) parce que, tout simplement, le site *j* a consommé toute la bande passante qui lui est allouée (FDDI) ou a épuisé le nombre de tranches de temps assigné (TDMA). Les paquets restants non encore émis pour le message du site *j* seront transmis au prochain passage du jeton (FDDI) ou après un cycle (TDMA).

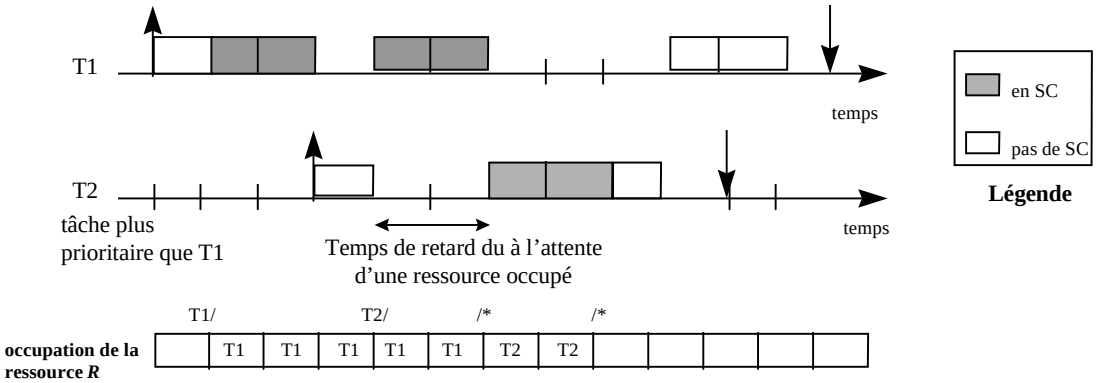


Figure 6.6: *Évaluation du temps de blocage dans une situation d'attente d'une ressource critique*

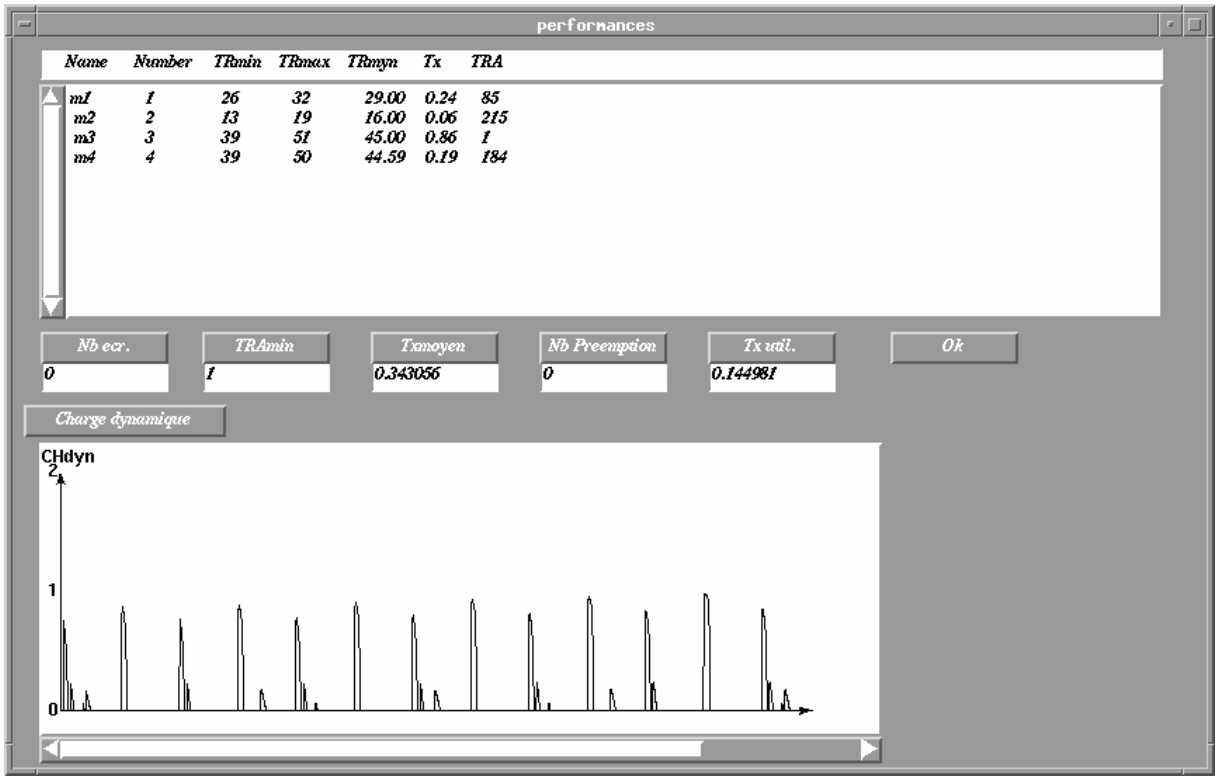


Figure 6.7: *Critères de Performance mesurés pour les messages*

Dans ce cas, nous assistons à une sorte de préemption puisque le médium est alloué à un message avant que le message qui l'occupait n'ait fini sa transmission. MOSARTS calcule le nombre de préemptions dans tous les réseaux y compris dans le cas de FDDI et TDMA même si cette notion est interprétée différemment selon le protocole de communication.

Au contraire, dans FIP, ce paramètre est inopérant puisque le protocole de communication se comporte comme un séquenceur en se contentant d'inviter le producteur à émettre et en ne lançant de nouvelle requête que lorsque la variable précédente est bien consommée.

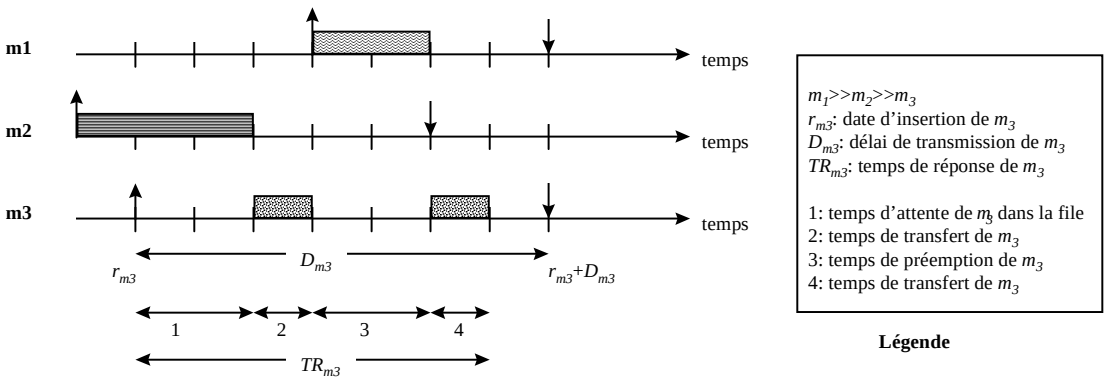


Figure 6.8 : *Le temps de réponse d'un message*

c. ***Le nombre d'écrasements***
Le nombre d'écrasements est le nombre de messages dont le temps de réponse dépasse la période sachant que le délai critique d'un message n'est rien d'autre que sa période. Si un message dépasse son échéance, il sera qualifié de non valide et entraîne la non validité de l'application si cette dernière est à contraintes strictes.

d. ***Le temps de retard admissible***
Le temps de retard admissible d'un message s'exprime en fonction du délai de transmission et du temps de réponse maximal. Il est égal au temps qui sépare la fin de transfert du délai de transmission :

$$TRA_m = D_m - TR_{max}$$

Nous définissons également un temps de retard minimal pour tous les messages émis sur le réseau : $TRA_{min} = \text{Min}(TRA_m)\ \forall m$

e. ***Le taux de réaction***
Le taux de réaction est égal au rapport du temps de réponse sur le délai de transmission : ($C_m/D_m \leq TR_{max}/D_m \leq 1$). Si ce taux est proche de 1, cela signifie que le message est émis très tardivement.

f. ***La charge dynamique***
La charge dynamique nous donne une idée de la charge du médium à chaque instant. Elle est définie comme étant la somme des durées nécessaires à la transmission des messages en attente dans la file, rapportées aux délais de transmission dynamiques. Le délai de transmission dynamique $D_m(t)$ d'un message est le temps séparant l'instant courant t de la date de son délai de transmission.

Charge = $\sum \frac{C_{mres\tan t}}{D_m(t)}$ pour tout message m en cours ou en attente de transmission.

charge=0 $\Rightarrow$ médium libre
charge=1 $\Rightarrow$ médium totalement chargé

g. ***Le taux d'utilisation du médium***
C'est le rapport du temps d'occupation du réseau sur la durée de simulation. Plus il sera proche de 1 plus le réseau est chargé : 0≤ taux d'utilisation≤1.

Parmi les paramètres cités ci-dessus, les quatre derniers nous informent de la charge du réseau, par conséquent de la possibilité ou non de rajouter du flux de messages.

3.3. Pour l'application globale

Pour l'application globale, les paramètres suivants sont déduits:

- le *taux d'occupation* (ou d'utilisation) du processeur pour chaque site,
- le *taux d'occupation* (ou d'utilisation) du médium,

- la *validité* de l'application obtenue à partir de l'état de validité des tâches et des messages. Dans une application temps réel répartie à contraintes strictes, si une tâche ou un message ne respecte pas son échéance, l'application n'est pas valide.

4. Résultats de simulation

MOSARTS construit la situation la plus pessimiste du point de vue temporel. Des mesures ont été faites, dans cette situation, certaines sont relatives aux sites d'autres au réseau.
MOSARTS peut être utilisé, d'une part, pour faire l'étude des performances temporelles d'une application donnée et, d'autre part, pour tester l'influence de certains paramètres, aussi bien associés aux tâches qu'aux messages (réseau), sur les performances temporelles de l'application ou sur l'ordonnançabilité de celle-ci.

Dans la suite de ce paragraphe, nous étudierons deux applications distinctes. La première application servira principalement à étudier l'influence de certains paramètres sur son ordonnançabilité dans le cas des réseaux CAN et FDDI. Sur la deuxième, nous faisons une analyse temporelle, à travers l'étude de certains critères de performance, afin de trouver l'algorithme d'ordonnancement et le protocole de communication qui, *a priori*, seraient performants pour cette application.

4.1. Premier exemple

Afin d'étudier l'influence des paramètres: taille des messages et largeur des bandes passantes des sites sur l'ordonnançabilité d'une application, dans le cas des réseaux CAN et FDDI, nous avons construit un exemple d'application tel qu'il est présenté dans la figure 6.9 avec les caractéristiques temporelles des tâches définies dans le tableau 6.1. Comme on peut le constater sur la figure 6.9, la taille des messages n'est pas fixe car elle varie d'un protocole à un autre. Cette définition est faite dans le but de comparer les réseaux CAN et FDDI pour cette application car en augmentant la taille d'un message dans CAN et en prenant la même taille dans FDDI, celle-ci correspondra toujours à un seul paquet FDDI. Nous avons alors fait varier les tailles des messages dans ces réseaux de manière indépendante.

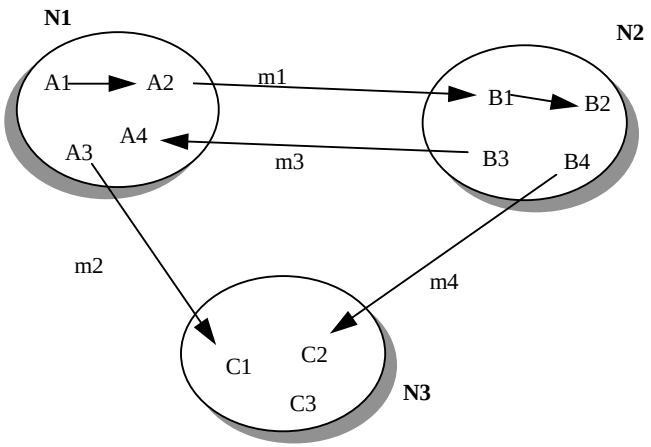


Figure 6.9: *Configuration d'une application distribuée*

Tableau 6.1: Caractéristiques des tâches de l'application de la figure 6.9

Tâche	Site	C (µs)	P=D (µs)	Pr[7]
A_1	N_1	200	10000	3
A_2	N_1	300	10000	2
A_3	N_1	100	20000	1
A_4	N_1	200	5000	4
B_1	N_2	300	10000	3
B_2	N_2	400	10000	2
B_3	N_2	200	5000	4
B_4	N_2	100	15000	1
C_1	N_3	300	20000	1
C_2	N_3	150	15000	2
C_3	N_3	200	10000	3

Les caractéristiques physiques des deux réseaux sont les suivantes:

1.CAN : Débit=1Mb/s ; les priorités des messages sont : $m_1>>m_2>>m_3>>m_4$.

2. FDDI : Débit=100Mb/s, *TTRT*=5000µs.

L'anneau est composé, dans l'ordre, de *N2*, *N3* et *N1* qui détient le jeton initialement. Les bandes passantes respectives des sites *N1*, *N2* et *N3* sont H_1=1815µs, H_2=2822µs et H_3=363µs.

Nous avons créé plusieurs situations que nous avons analysées en mesurant les temps de réponse moyens des messages notés R_m dans la suite.

1er Cas

Nous affectons des tailles identiques à tous les messages en commençant par un paquet par message dans FDDI et un octet dans CAN et nous essayons de déduire la taille maximale atteinte sans violation des échéances des tâches et des messages.

Vu le débit important de FDDI, nous constatons que les tâches de l'application peuvent échanger jusqu'à 6 paquets (6*4490 octets) par message sans violation des échéances comme le montre la figure 6.10. Par contre, CAN est dédié aux petits messages puisqu'un message ne dépasse pas 8 octets d'information utile (voir figure 6.11). Ces résultats confirment bien ce à quoi ces protocoles sont dédiés. En effet, CAN convient bien aux messages courts alors que FDDI a de meilleures performances lorsque les messages ont des tailles plus importantes.

[7] Priorité selon RM

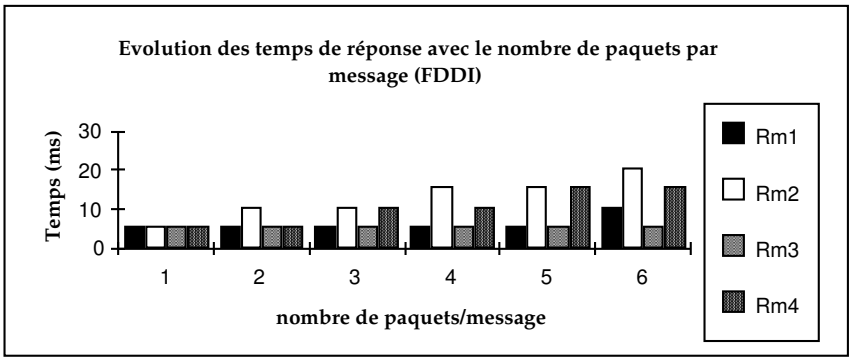


Figure 6.10: *Les temps de réponse variant en fonction du nombre de paquets par message dans FDDI*

2ème Cas:

Dans les mêmes conditions que dans le premier cas c'est-à-dire pour les mêmes caractéristiques physiques des deux réseaux et pour des tailles des messages initialement égales à un paquet pour FDDI et un octet pour CAN que l'on augmente respectivement d'un paquet et d'un octet. Nous comparons alors les temps de réponse R_m mesurés sur le diagramme de Gantt avec les délais de transmission D_m calculés par la méthodologie. Nous remarquons que ces temps sont plus proches dans le protocole CAN que dans le protocole FDDI (voir figures 6.12, 6.13). Cette différence entre les temps R_m et D_m, dans le cas de FDDI, est due au fait que dans le calcul de D_m nous supposons que les bandes passantes allouées sont complètement consommées par chaque site à chaque tour du jeton. Mais la simulation se rapproche de la réalité et donne des temps de réponse plus petits car les sites n'émettent pas à chaque passage du jeton, c'est-à-dire que chaque site qui n'a pas de message à émettre libère le jeton et le prochain site émetteur commencera sa transmission plus tôt que prévu. Par exemple, si un paquet d'un message *m* est transmis et que la bande passante est consommée parce qu'elle a été utilisée pour la transmission d'autres paquets et si, de plus, aucun autre site dans le réseau ne veut émettre alors les paquets de *m* restants n'attendront que le temps que mettra le jeton pour parcourir le réseau sans transmission.

Dans le réseau CAN par contre, lorsque la charge du réseau commence à devenir importante, les temps de réponse (voir figure 6.13) approchent les délais de transmission car dans le cas de surcharge du réseau, les temps de réponse correspondent à une situation défavorable exprimée dans le délai D_m.

3ème Cas

Nous considérons, dans ce cas, uniquement le réseau FDDI avec les caractéristiques physiques suivantes:

Débit=100Mb/s, *TTRT*=5000µs.

m_1 comporte 8980 octets de données correpondant à deux paquets, m_2 contient 4490 octets c'est-à-dire un paquet, m_3 et m_4 contiennent chacun 13470 octets c'est-à-dire 3 paquets.

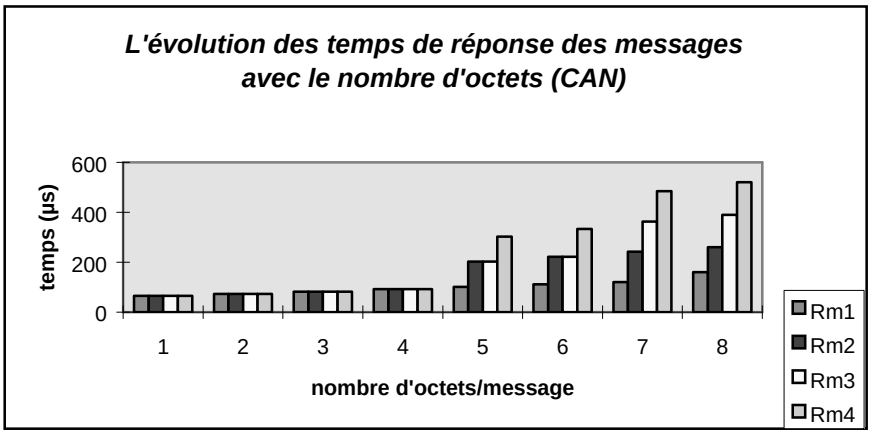


Figure **6.11***: Les temps de réponse variant avec le nombre de paquets par message dans CAN*

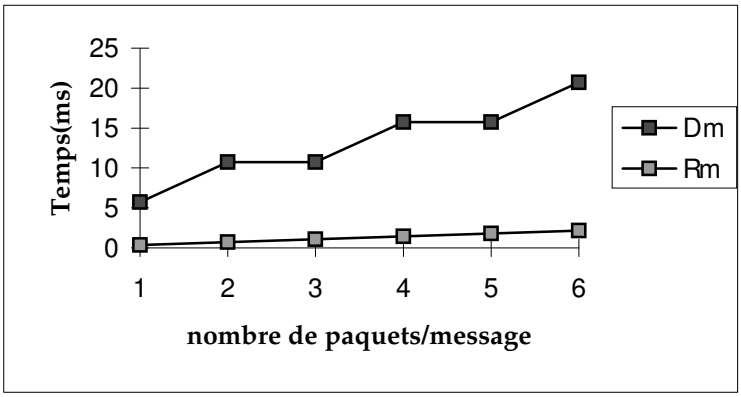


Figure 6.12: *Comparaison des temps de réponse avec les délais de transmission dans FDDI*

Pour voir si les bandes passantes allouées influent sur l'ordonnançabilité de l'application, nous faisons varier les bandes passantes H_i des différents sites.

Dans un premier temps, nous affectons 0,363 ms correspondant à la durée d'émission d'un seul paquet à H_1 et nous l'incrémentons d'un pas égal à 0,363ms, à chaque fois, en déduisant les bandes passantes des sites *N2* et *N3* comme suit: $H_2=H_3=$ ($TTRT-H_1$)/2. Les temps de réponse obtenus, dans ce cas, et présentés en figure 6.14 montrent que l'application est ordonnançable de H_1=0,363ms à H_1=2,54 ms ±0,36 à partir de quoi au moins une tâche rate son échéance.

Dans un deuxième temps, nous affectons les bandes passantes de manière à ce qu'elles soient proportionnelles à l'activité de communication des sites. Nous considérons alors une situation dans laquelle un site qui a des messages à émettre verra sa bande passante croître. Comme le site *N3* n'émet pas, nous lui affectons une bande passante H_3 de 0,363 ms et nous partageons le reste de la bande passante c'est-à-dire $TTRT-H_3$ entre *N1* et *N2*. De manière similaire, nous augmentons H_1 d'un pas de 0,363ms et nous déduisons H_2 qui est égal à ($TTRT-H_3$)-H_1.
Notre constatation est que les temps de réponse dans la figure 6.15 sont plus petits que ceux de la figure 6.14 et que l'ordonnançabilité est vérifiée jusqu'à une valeur de la bande passante égale à H_1=3,26ms. Dans la figure 6.15, les temps de réponse sont meilleurs dans l'intervalle [1,09±0,36, 2,54±0,36] de H_1 correspondant à l'intervalle

[2,09±0,36, 3,54±0,36] de H_2. Ils sont optimaux car dans cet intervalle, *N1* et *N2* peuvent envoyer tous leurs messages dans les bandes passantes allouées.

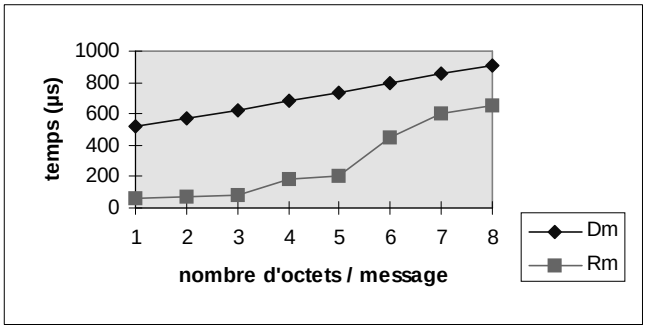


Figure 6.13: *Comparaison des temps de réponse avec les délais de transmission dans CAN*

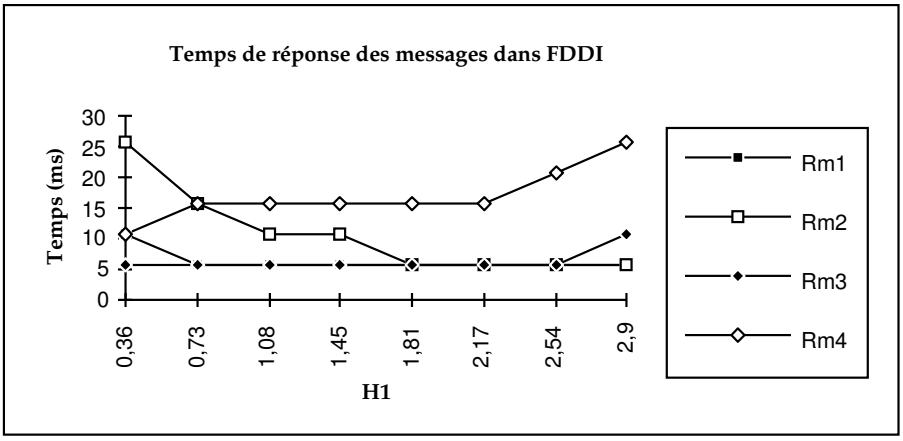


Figure 6.14: *Les temps de réponse des messages lors de l'évolution de H_1 dans FDDI*

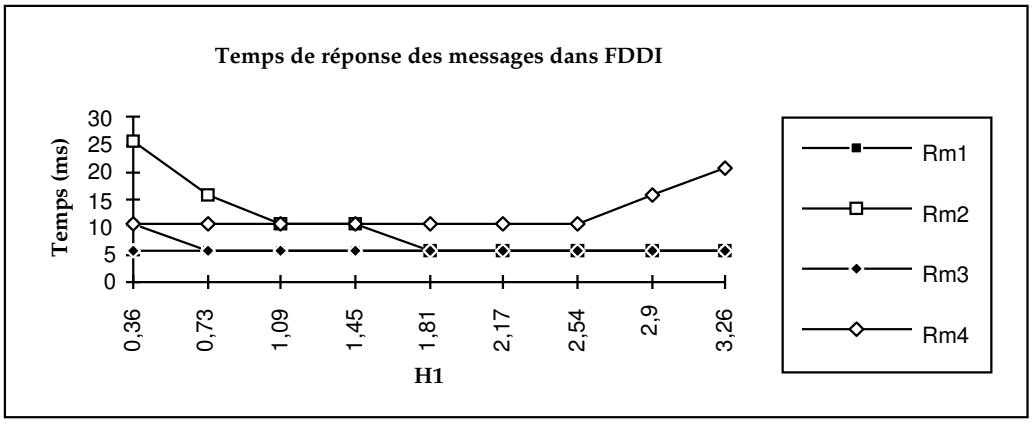


Figure 6.15: *Les temps de réponse des messages lorsque H_1 varie dans FDDI*

4.2. Deuxième exemple

Considérons l'application répartie schématisée par la figure 6.16 et implantée sur 4 sites. Les caractéristiques temporelles des tâches sont données dans le tableau 6.2 et s'expriment en µs. Les tailles des messages sont toutes identiques et égales à 8 octets (voir figure 6.16). Le site 3 et le site 1 disposent respectivement de deux ressources s_1 et s_2 et d'une ressource s_3. Les durées d'utilisation de ces ressources sont données dans les tableaux 6.3 et 6.4.

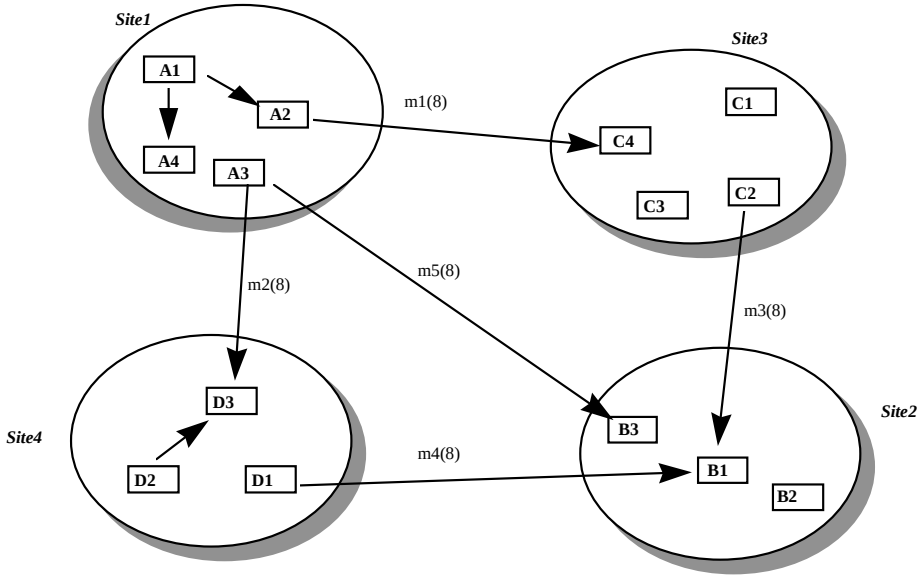


Figure 6.16 : *Un exemple d'application temps réel répartie*

4.2.1. Analyse temporelle des sites

Nous voulons étudier, pour un réseau donné, l'influence de l'algorithme d'ordonnancement sur le comportement temporel de l'application. Le paramètre qui nous semble le plus pertinent est le temps de réponse moyen lorsque l'objectif principal qui est de vérifier le respect des échéances pour les tâches et les messages est atteint.

4.2.1.1. Étude des temps de réponse

Étant donné le réseau CAN, nous voulons étudier les temps de réponse des tâches et des messages dans le cas de l'algorithme d'ordonnancement RM avec le protocole à priorité plafond (PPP) ensuite de l'algorithme ED avec le protocole à pile (PAP).

4.2.1.2. Les caractéristiques physiques du réseau sont:

- le débit est égal à 1Mb/s,
- les priorités des messages sont : $m_1 >> m_2 >> m_3 >> m_4$.

Tableau 6.2 : *Les caractéristiques temporelles des tâches de l'application de la figure 6.16*

Site	Tâche	r_i	C_i	D_i	P_i
Site1	A_1	0	1000	4000	7000
	A_2	0	800	3500	7000
	A_3	0	1000	4000	7000
	A_4	0	900	3000	7000
Site2	B_1	0	500	2000	5000
	B_2	0	500	1500	3000
	B_3	0	1000	4000	7000
Site3	C_1	20	300	2000	5000
	C_2	90	800	2500	5000
	C_3	30	1000	4500	7000
	C_4	0	800	3400	7000
Site4	D_1	0	500	2000	5000
	D_2	0	1000	4500	7000
	D_3	0	700	3500	7000

Tableau 6.3 : *Les durées d'utilisation des ressources par les tâches du site 3*

Tâche	$C^{\alpha}(s_1)$	$C^{\beta}(s_1)$	$C^{\gamma}(s_1)$	$C^{\alpha}(s_2)$	$C^{\beta}(s_2)$	$C^{\gamma}(s_2)$
C_1	100	100	100	300	0	0
C_2	800	0	0	300	200	300
C_3	400	400	200	900	0	0
C_4	800	0	0	300	300	200

Même si la figure 6.16 distingue cinq messages car la communication est représentée, dans notre modèle, à l'aide d'un échange de messages entre deux tâches distantes, sur le médium de communication il n'existe que quatre messages car les messages m_2 et m_5 correspondent physiquement à un seul message émis sur le réseau puisqu'ils sont émis par la même tâche A_3 même s'ils sont reçus par deux tâches distinctes B_3 et D_3.

Tableau 6.4 : *Les durées d'utilisation des ressources par les tâches du site 1*

Tâche	$C^{\alpha}(s_3)$	$C^{\beta}(s_3)$	$C^{\gamma}(s_3)$
A_1	1000	0	0
A_2	800	0	0
A_3	300	700	0
A_4	400	400	100

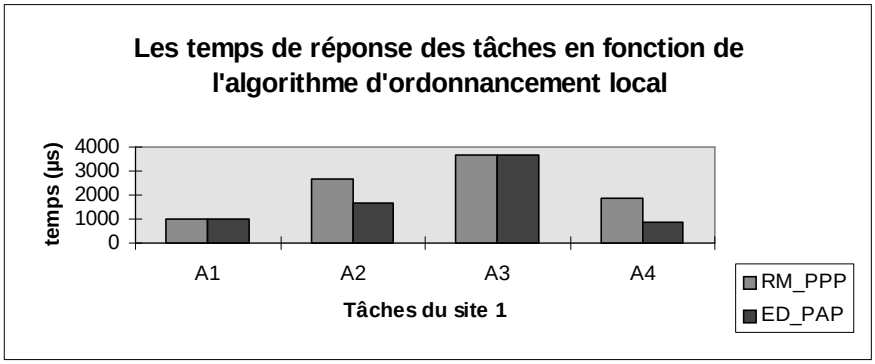


Figure 6.17 :

Variation des temps de réponse des tâches du site 1 en fonction de l'algorithme d'ordonnancement local dans le cas d'un réseau CAN

Les figures 6.17, 6.18, 6.19 et 6.20 représentent les temps de réponse des tâches respectivement sur les sites 1, 2, 3 et 4. La figure 6.14 donne les temps de réponse des messages en fonction de l'algorithme d'ordonnancement local.

On pourrait penser que la transmission des messages sur le médium n'a aucune relation avec l'algorithme d'ordonnancement des sites. Or, comme nous l'avons vu au chapitre 5, les dates d'insertion des messages sont étroitement liées au contexte des sites émetteurs. En effet, si avec un algorithme d'ordonnancement les dates d'insertion des messages sont telles que les messages sont émis séparément sur le réseau alors avec un autre algorithme les dates peuvent correspondre à d'autres instants provoquant ainsi des préemptions entre messages et modifiant par conséquent les temps de réponse obtenus initialement. Dans la figure 6.21, les temps de réponse semblent être meilleurs avec l'algorithme RM.

Nous constatons que l'algorithme d'ordonnancement qui donne les meilleurs temps de réponse n'est pas le même pour tous les sites. En effet, RM est meilleur dans le site 1 (figure 6.17) et ED est meilleur dans le site 3 (figure 6.19). Le concepteur pourra opter pour l'un ou l'autre des algorithmes selon qu'il s'intéresse aux performances d'un site particulier, ou à celles d'un réseau ou encore à celles de l'ensemble des sites et du réseau.

Si l'on s'intéresse au temps de réponse pour la globalité de l'application, on peut remarquer que sur 18 entités (tâches et messages), 7 cas présentent des temps de réponse identiques quel que soit l'algorithme d'ordonnancement, 7 cas donnent les meilleurs temps avec RM et 4 avec ED. On peut donc retenir RM, pour cette application, comme algorithme d'ordonnancement qui donne les meilleures

performances lorsque le réseau utilisé est CAN avec les caractéristiques physiques et les priorités des messages telles qu'elles sont spécifiées.

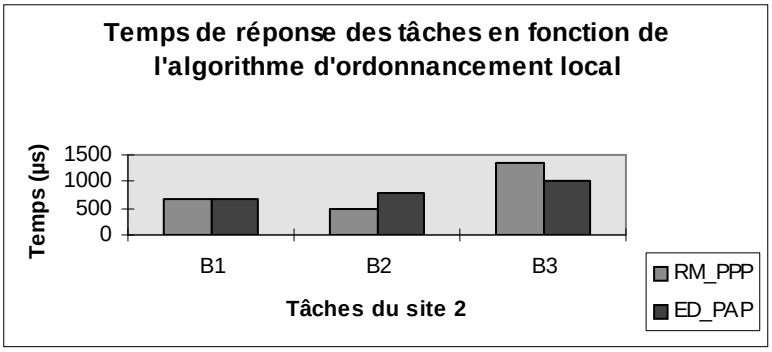


Figure 6.18 :

Variation des temps de réponse des tâches du site 2 en fonction de l'algorithme d'ordonnancement local dans le cas d'un réseau CAN

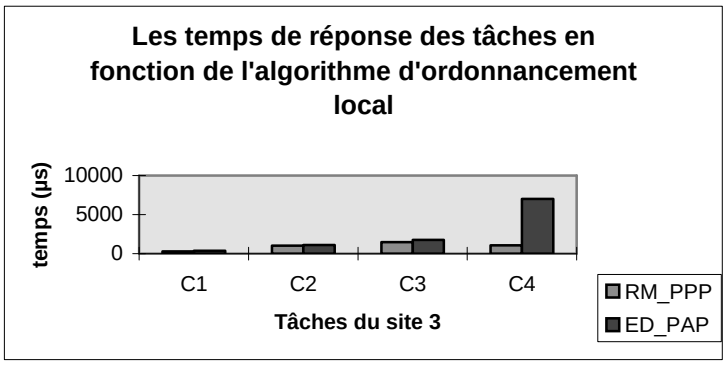


Figure 6.19 :

Variation des temps de réponse des tâches du site 3 en fonction de l'algorithme d'ordonnancement local dans le cas d'un réseau CAN

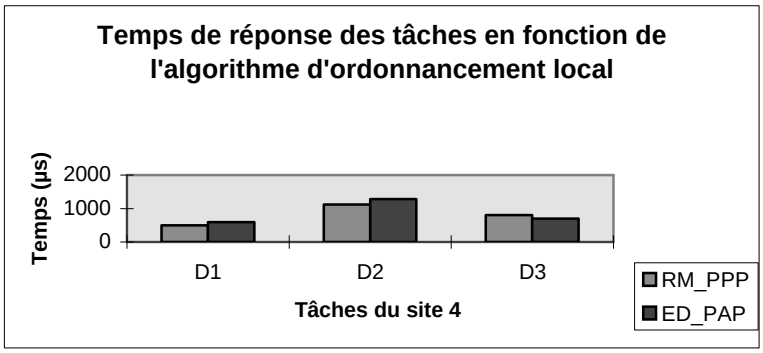


Figure 6.20 :

Variation des temps de réponse des tâches du site 4 en fonction de l'algorithme d'ordonnancement local dans le cas d'un réseau CAN

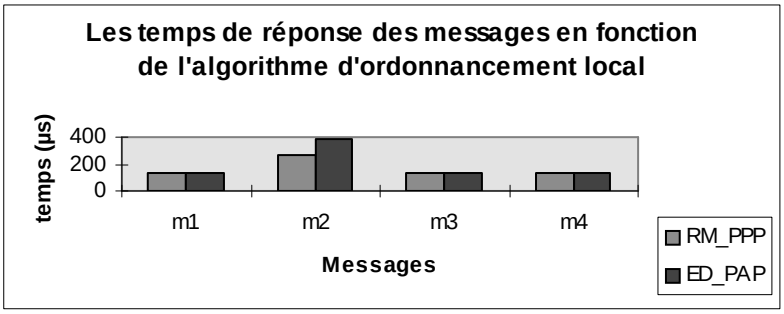


Figure 6.21 :

Variation des temps de réponse des messages en fonction de l'algorithme d'ordonnancement local des sites dans le cas d'un réseau CAN

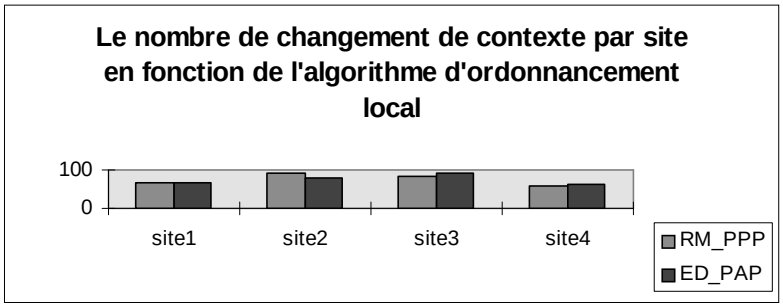


Figure 6.22 :

Variation du nombre de changement de contexte en fonction de l'algorithme d'ordonnancement local dans le cas d'un réseau CAN

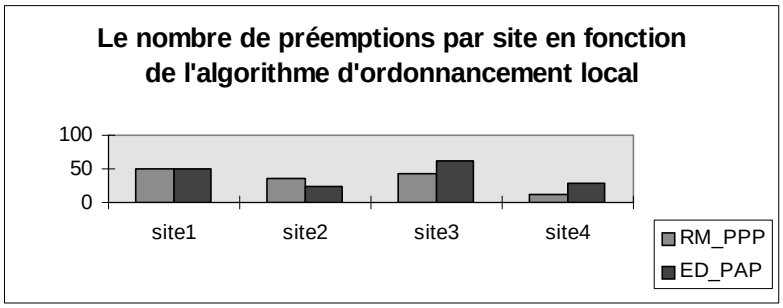


Figure 6.23 :

Variation du nombre de préemptions de tâches pour chaque site en fonction de l'algorithme d'ordonnancement local dans le cas d'un réseau CAN

4.2.1.3. Etude de deux autres paramètres

Le concepteur peut décider de choisir la solution qui génère le moins de préemptions et de changement de contexte pour réduire l'overhead du processeur. Dans ce cas là, l'étude de ces deux paramètres peut être intéressante. La figure 6.22 qui donne le nombre de changement de contexte par site montre des valeurs identiques pour les sites 1 et 4 quel que soit l'algorithme d'ordonnancement local. Au contraire, avec l'algorithme ED, ce nombre est plus faible dans le site 2 et est plus important dans le site 3. Globalement, les deux algorithmes peuvent être utilisés de manière indifférente car les résultats sont équivalents dans les deux cas.

La figure 6.23 donne le nombre de préemptions par site, qui est faible avec RM dans trois sites sur quatre.

4.2.1.4. Conclusion

A travers l'analyse temporelle des sites, nous pouvons conclure que cette application donne les meilleures performances, au sens des temps de réponse, avec l'algorithme d'ordonnancement RM lorsque le réseau utilisé est CAN.

4.2.2. Analyse temporelle du réseau

Etant donné un réseau de type CAN, il est possible, dans une deuxième étape, d'étudier l'influence des caractéristiques physiques du réseau, par exemple faire varier le débit, ou bien changer les priorités des messages toujours dans le souci de trouver les paramètres qui conviennent le mieux d'un point de vue temporel. Cela dépend des spécifications faites dans le cahier des charges.

L'objectif peut être aussi de tester l'application sur plusieurs réseaux et d'opter pour un type de réseau en fonction des résultats d'analyse obtenus.

Pour atteindre cette dernière possibilité, nous supposons utiliser l'algorithme RM au niveau de chaque site et nous analysons les performances de l'application pour chaque type de réseau. Les caractéristiques physiques de chaque réseau ainsi que les priorités des messages lorsque celles-ci existent sont les suivantes :

1. Réseau CAN

   Débit = 1Mb/s ; $m_1>>m_2>>m_3>>m_4$ (si $a>>b$ signifie a plus prioritaire que b).

2. Réseau FDDI

   Débit=100Mb/s, $TTRT$=5000µs ;

   L'anneau est composé, dans l'ordre, de : site 1, site 2, site 3 et site 4. Initialement le site 1 détient le jeton. Si H_i est la bande passante allouée au site i alors H_1=360µs, H_2=1080µs, H_3=720µs et H_4=1080µs.

3. Réseau CSMA/DCR

   Débit=10Mb/s, T_c=50µs ;

   Site1>>Site2>>Site3>>Site4.

4. Réseau TDMA

   Débit=1Mb/s, un cycle=16 tranches de temps ;

   Chaque site dispose de 4 tranches de temps pour émettre.

Nous avons choisi, volontairement, d'affecter à chaque message de l'application, une taille qui ne dépasse pas 8 octets de données ; ceci en vue de comparer les réseaux, à peu près, dans les mêmes conditions.

Pour chaque type de réseau, défini ci-dessus, nous avons mesuré les temps de réponse moyens des tâches et des messages.

4.2.2.1. Étude des temps de réponse

La figure 6.24 qui représente les temps de réponse des messages montre que FDDI est le plus performant, cela s'explique par le fait que son débit est important. Les figures 6.25, 6.26, 6.27 et 6.28 montrent les temps de réponse des tâches. On peut penser que les seules entités connues du réseau sont les messages et qu'il n'est pas nécessaire de s'intéresser au comportement des tâches vu que localement l'algorithme d'ordonnancement est toujours le même. Ce raisonnement est correct lorsqu'un site ne reçoit pas de messages. C'est le cas du site1 dans lequel les temps de réponse des tâches sont les mêmes quel que soit le réseau considéré, pourvu que l'algorithme d'ordonnancement ne change pas. En effet, comme nous l'avons vu dans le chapitre 5, la méthodologie n'altère que les tâches réceptrices et successeurs. Or dans le site 1, les tâches peuvent émettre mais pas recevoir.

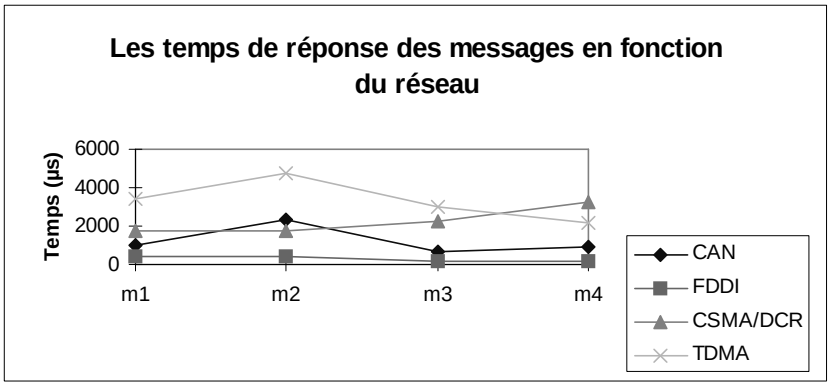


Figure 6.24 :

Variation des temps de réponse des messages en fonction du réseau local dans le cas de l'algorithme RM

Dans les figures relatives aux sites 3, 4 et 5 (figures 6.26, 6.27, 6.28) les temps de réponse ne sont pas identiques pour tous les réseaux. Ceci s'explique par le fait suivant : comme les messages ont des délais de transmission différents selon le réseau considéré les instants de réception des messages, par conséquent les dates de réveil des dates réceptrices sont retardées ou avancées. Ainsi l'ordonnancement des tâches au niveau d'un site récepteur change car, si dans un cas, deux tâches sont réveillées séparément et exécutées séparément, la modification d'une date de réveil d'une tâche (parce qu'elle est réceptrice de message) peut provoquer une préemption ou un blocage pour une ressource parce que la tâche qui occupe le processeur est moins prioritaire ou bien parce qu'elle détient la ressource demandée. Ces interactions peuvent diminuer ou accroître les temps de réponse des tâches et changer la séquence d'exécution du site qui peut alors générer plus ou moins de changements de contexte ou de préemptions comme dans la figure 6.29.

4.2.2.2. Conclusion

Comme les variations des temps de réponse pour chaque tâche ne sont pas très importantes en changeant de réseau, le choix de ce dernier se basera principalement, pour cette application, sur les temps de réponse des messages qui nous permettent de classer les protocoles de communication dans l'ordre suivant: FDDI, CAN, CSMA/DCR et TDMA en supposant un ordonnanceur de tâches de type RM au niveau des sites.

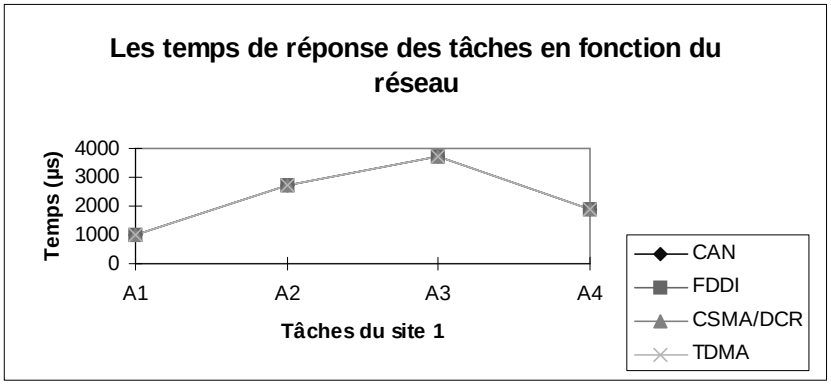


Figure 6.25 :

Variation des temps de réponse des tâches du site 1 en fonction du réseau local dans le cas de l'algorithme RM

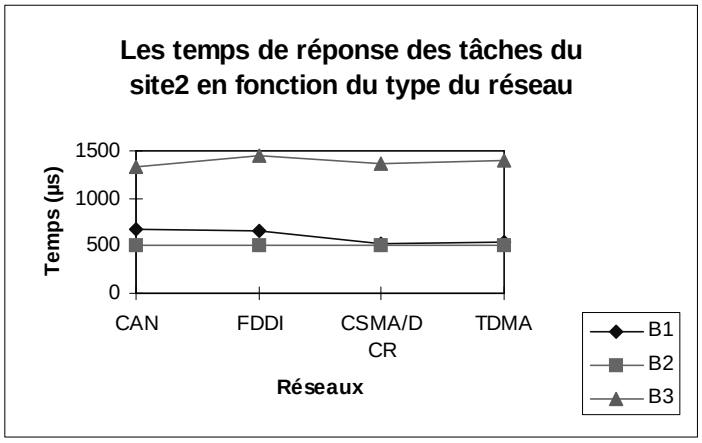


Figure 6.26 :

Variation des temps de réponse des tâches du site 2 en fonction du réseau local dans le cas de l'algorithme RM

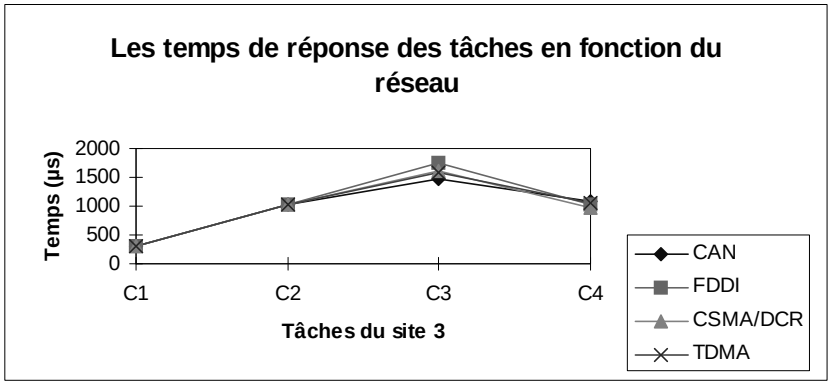


Figure 6.27 :

Variation des temps de réponse des tâches du site 3 en fonction du réseau local dans le cas de l'algorithme RM

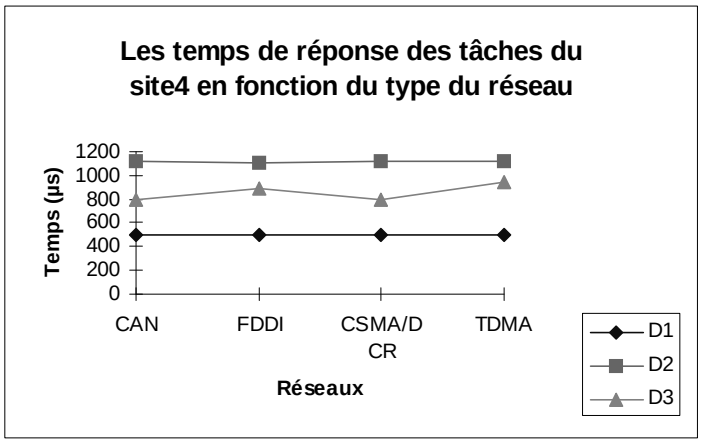


Figure 6.28 :

Variation des temps de réponse des tâches du site 4 en fonction du réseau local dans le cas de l'algorithme RM

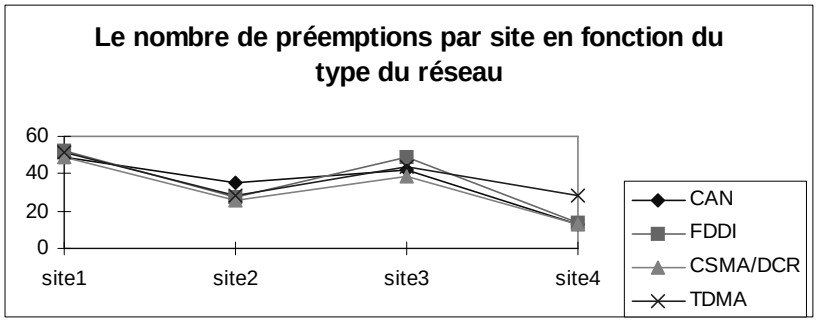


Figure 6.29 :

Variation du nombre de préemptions de tâches pour chaque site en fonction du réseau local dans le cas de l'algorithme RM

4.2.3.Analyse temporelle de bout en bout

L'analyse temporelle de bout en bout consiste à étudier d'un point de vue temporel un ensemble de tâches liées par des relations de précédence locale et globale. Dans l'exemple de la figure 6.9, si A_1 est une tâche qui lit un capteur de température, périodiquement, qui transmet ses données à la tâche A_2 qui les traite et les transmet à C_4 sur le site 3 pour afficher les résultats, il serait intéressant d'étudier le temps que mettrait la séquence de tâches A_1, A_2 et C_4 à partir de la lecture du capteur jusqu'à l'affichage. Ce temps peut être pertinent pour le concepteur. D'après l'analyse précédente faite pour les sites et le réseau, l'affichage des résultats sur le site3 obtenus à partir de données lues sur le site1 se fait dans le meilleur des cas au bout de 4668,7µs et ceci en utilisant l'algorithme RM avec le réseau CSMA/DCR comme le montre le tableau 6.5.

5. Conclusion

Dans ce chapitre consacré à l'évaluation des performances temps réel d'une application répartie, nous nous sommes intéressés à deux exemples d'applications temps réel réparties.

Le premier exemple a été traité dans le but d'étudier les réseaux CAN et FDDI. Un des paramètres pertinents à mesurer concerne les temps de réponse des messages. D'abord, les résultats vérifient bien la propriété classique: CAN est plus approprié lorsque les messages sont petits alors que FDDI a de meilleures performances avec des messages de taille importante. Ensuite, la comparaison des délais de transmission et les temps de réponse observés en simulation montrent bien l'efficacité des différents protocoles dans le contexte d'une application spécifique.

Finalement, une étude numérique a été présentée concernant l'effet de l'allocation de la bande passante pour un réseau FDDI, qui montre qu'une application est ordonnançable sur un grand intervalle de temps lorsque la bande passante est proportionnelle au nombre d'émissions pour chaque site. Cette approche est particulièrement intéressante dans les systèmes distribués temps réel à contraintes strictes où les contraintes d'ordonnançabilité sont clairement définies et où les caractéristiques temporelles sont d'une importance primordiale.

Le deuxième exemple a été utilisé pour montrer le type d'analyse que l'on peut faire sur une application à partir de quelques paramètres mesurés sur le diagramme de Gantt. Cette analyse distingue plusieurs cas:

- l'étude temporelle des sites à travers les temps de réponse des tâches,
- l'étude temporelle du réseau à travers les temps de réponse des messages,
- l'étude temporelle de bout en bout d'une séquence de tâches.

Tableau 6.5 : *Les temps de réponse pour la séquence (A_1, A_2, C_4) dans différents cas*

Tâche/séquence	RM avec CAN	ED avec CAN	RM avec FDDI	RM avec CSMA	RM avec TDMA
A_1	1000	1000	1000	1000	1000
A_2	2700	2700	2700	2700	2700
C_4	1071,8	1091,7	1036	968,7	1061,2
(A_1, A_2, C_4)	4771,8	4791 ,8	4736	4668,2	4761,2

L'exemple illustre bien le fait que les performances d'une application dépend de l'algorithme d'ordonnancement des tâches, des caractéristiques physiques du réseau ainsi que du protocole de communication utilisé.

CONCLUSION GENERALE

Ce travail de thèse constitue une contribution à la validation d'applications temps réel réparties à contraintes strictes. En nous appuyant sur une méthodologie basée sur l'analyse d'ordonnançabilité, nous nous sommes intéressé aux problèmes d'ordonnancement des tâches et des messages que soulève ce type d'applications.

Après avoir défini une application temps réel répartie comme étant un ensemble de tâches réparties sur différents sites pour lesquels le seul moyen de communication est l'échange de messages reposant sur un réseau local doté d'un protocole de communication temps réel, nous avons développé notre étude selon trois axes principaux:

- classification des algorithmes d'ordonnancement et des protocoles d'allocation de ressources,
- classification des réseaux locaux temps réel et description de quelques exemples de protocoles de communication,
- proposition d'une méthodologie de validation temporelle d'applications temps réel réparties à contraintes strictes et évaluation des critères de performance.

<u>En ce qui concerne l'ordonnancement des tâches:</u>

Un modèle temporel de tâche a été proposé et une classification des algorithmes d'ordonnancement basée sur la nature de la priorité, fixe ou dynamique, est dressée. Une adaptation du modèle de tâches est présentée afin de tenir compte des relations de précédence et d'appliquer les mêmes algorithmes pour l'ordonnancement des tâches. Nous avons montré que la compétition pour l'accès à une ressource critique peut engendrer le phénomène d'inversion de priorités. Des solutions à ce problème qui tentent de borner le temps de blocage sont exposées. Elles correspondent aux différents protocoles d'allocation de ressources intégrés aux algorithmes d'ordonnancement.

<u>En ce qui concerne l'ordonnancement des messages:</u>

Les messages sont les entités d'information échangées entre les tâches application. Ils sont décomposés en plusieurs entités d'information de taille fixe appelées paquets ou trames. Les trames sont les unités d'information connues du réseau et sont ordonnancées sur le médium de communication à l'aide d'un protocole de communication MAC. Une classification de ces protocoles est présentée et illustrée par un exemple réel de protocole pour chaque classe.

En ce qui concerne la méthodologie de validation:

A travers les hypothèses qui sont faites, nous avons défini clairement le type d'applications temps réel réparties pour lequel nous pouvons appliquer notre analyse, à savoir les applications constituées de tâches périodiques à contraintes strictes implantées sur un réseau fiable, muni d'une horloge globale et fournissant un délai de communication borné.
Nous avons défini un modèle de message analogue à celui des tâches et, comme les tâches que nous traitons sont périodiques, nous avons restreint notre étude aux messages synchrones.
Les méthodes et outils qui ont traité le problème de l'ordonnancement des tâches et des messages ont toujours favorisé un aspect particulier. Dans PERTS [LRD *et al.*], par exemple, l'analyse d'ordonnançabilité des tâches est plus approfondie tandis que dans [TBW 95] et [SSB 97] c'est l'ordonnancement des messages qui est prépondérant. [KLR 94] tente bien une étude de l'ordonnançabilité de l'application répartie dans sa globalité (ordonnançabilité des tâches et des messages) en appliquant la même technique d'ordonnancement pour les tâches et pour les messages. Néanmoins, cette méthode est limitée à un réseau FDDI et ne tient pas compte de la politique d'ordonnancement des messages. Nous avons donc apporté une solution à ces carences. Notre méthodologie prend en compte aussi bien l'ordonnancement des tâches au niveau des sites, en considérant les relations de précédence entre tâches et le partage de ressources, que l'ordonnancement des messages en tenant compte des caractéristiques physiques du réseau et du protocole de communication associé. Nous ne faisons pas d'hypothèse restrictive sur l'algorithme d'ordonnancement des sites ni sur le protocole de communication du réseau. Notre méthodologie est donc générale et ne se borne pas à un algorithme d'ordonnancement local ni à un réseau de communication particuliers (voir tableau 6.6).
La stratégie de la méthodologie consiste à déterminer les paramètres temporels des messages en altérant ceux des tâches impliquées dans la communication, les caractéristiques temporelles des tâches étant définies par le concepteur et donc connues a priori. Une fois les tâches et les messages complètement définis du point de vue temporel, on opère un ordonnancement des tâches locales selon le même algorithme d'ordonnancement sur tous les sites et un ordonnancement des messages sur le réseau selon un protocole de communication.
Conjointement, nous avons développé un outil, appelé MOSARTS, basé sur la méthodologie proposée, qui offre la possibilité de choisir un algorithme d'ordonnancement associé à un protocole d'allocation de ressources pour tous les sites et un protocole de communication pour le réseau caractérisé par des paramètres physiques. En outre, il intègre un module d'évaluation des critères de performance, calculés pour les tâches et pour les messages. Ce calcul nous donne une appréciation, sous les conditions les plus pessimistes, sur les temps de réponse des tâches et des messages, les taux d'utilisation des processeurs des sites et du médium de communication, etc. L'outil d'analyse temporelle peut être utilisé, d'une part pour faire l'étude des performances temporelles d'une application donnée et, d'autre part, pour tester l'influence de certains paramètres, aussi bien associés aux tâches qu'aux messages, sur les performances temporelles de l'application ou sur l'ordonnançabilité de celle-ci.

Tableau 6.6 : *Tableau Comparatif*

	Algorithme d'ordonnance-ment	***Protocole d'allocation de ressources***	***Précédence/ Partage de ressources***	***Réseau***	***Calcul des performances***
PERTS [LRD 93 et *al.*]	RM DM	PPP	- précédence - ressources	modèle simplifié	des tâches
[TBW 95]	-	-	-	CAN	-
[KLR 94]	RM	PPP	-précédence -ressources	FDDI	-
MOSARTS	RM DM ED	PHP PPP PAP	-précédence -ressources	CAN CSMA/DCR FIP FDDI TDMA	- des tâches - des messages

Perspectives:

Au terme de cette étude se révèlent plusieurs perspectives d'investigation. Les plus pertinentes sont les suivantes:

- Cette méthodologie analyse les tâches périodiques qui échangent des messages synchrones. Les tâches apériodiques peuvent être intégrées si elles sont gérées par des serveurs périodiques. En d'autres termes, cette méthodologie telle qu'elle est présentée ne sait pas tenir compte des événements aléatoires qu'il s'agisse des tâches ou des messages. Il serait donc intéressant d'étendre ce travail à l'analyse des tâches apériodiques qui échangent des messages asynchrones en définissant, par exemple, des lois probabilistes d'arrivée des tâches apériodiques qui produisent par conséquent des messages asynchrones suivant les mêmes lois.

- Nous avons fait l'hypothèse, tout au long de notre étude, d'un réseau fiable c'est-à-dire présentant un taux d'erreurs négligeable. Une amélioration de notre travail serait de prendre en compte différents types d'erreurs de transmission, d'évaluer le surcoût en temps de calcul induit par ces erreurs et concevoir son intégration dans cette méthodologie. A première vue, les erreurs que nous pouvons borner dans le temps peuvent être prises en compte facilement en incluant le temps nécessaire à leur détection et leur traitement dans les délais de transmission des messages.

- Cette méthodologie suppose un placement statique des tâches et aucun moyen n'est offert pour placer les tâches selon des critères définis par l'utilisateur. Au lieu que celui-ci fixe le placement des tâches, une extension qui est en cours de réalisation est d'affecter des tâches à des processeurs en optimisant certains critères (équilibrage de la charge du processeur, coût des communications, etc.). Dans ce cas, le travail entamé consiste à modifier l'interface graphique de MOSARTS afin de définir, d'un côté, les tâches avec les messages échangés et, de l'autre, les sites munis éventuellement de ressources critiques locales et d'utiliser un algorithme qui donnerait le meilleur placement en respectant une liste de critères. Le résultat est d'obtenir une configuration pour l'application répartie qu'il faudra valider temporellement.

- L’ordonnancement des messages a consisté en l’utilisation simplement d’une technique MAC qui est plus destinée à la gestion d’accès au médium qu’à la prise en compte des contraintes de temps des messages. L’idée serait d’enrichir cette technique à l’aide d’un algorithme d’ordonnancement pour les messages (RM, EDF, etc.) et d’intégrer, dans cette méthodologie par conséquent, un vrai algorithme d’ordonnancement. La modularité de MOSARTS et la communication des modules à l’aide de fichiers donnent une certaine souplesse quant à la modification d’un module. En effet, dans ce cas précis, il suffit de remplacer le programme actuel qui implante une stratégie d’accès au médium en un programme qui implante un algorithme d’ordonnancement de messages.

- Cette méthodologie semble s’appliquer uniquement aux protocoles de communication à délai d’accès borné ; son extension à d’autres réseaux notamment ATM serait un apport intéressant. En effet, la technologie ATM (Asynchronous Transfer Mode) est la seule qui permet le transport simultané de la voix, des données informatiques et de la vidéo dans des conditions semblables à l’utilisation de réseaux séparés grâce à sa large gamme de débits (depuis quelques mégabits jusqu’à plusieurs gigabits) et à la qualité de service négociée au départ. La technologie ATM supporte entre autres, des applications multimédias qui nécessitent des qualités de service en termes de fiabilité, d’ordonnancement et de contraintes temporelles.

BIBLIOGRAPHIE

[ACZD 92]: G. Agrawal, B. Chen, W. Zhao, S. Davari, " *Guaranteeing synchronous message deadlines in high speed ring networks with the timed token protocol*", Proc. of the 12th inter. Conf. on Distributed Computing Systems, pp. 468-475, Japan, 1992.

[ARS91]: K. Arvind, K. Ramamritham, J.A. Stankovic, " *A local area network architecture for communication in distributed real-time systems*", J. of Real-Time Systems, 3(2), pp.115-147, 1991.

[Bab 96]: J. P. Babau, "*Etude du comportement temporel des applications temps réel à contraintes strictes basée sur une analyse d'ordonnançabilité*", thèse de doctorat, ENSMA, 1996.

[Bak 91]: T. P. Baker, "*Stack-Based Scheduling of Realtime Processes*", J. Real-Time Systems, Vol. 3, N° 1, pp. 67-99, March 1991.

[Bla 76]: J. Blazewicz, *« Scheduling Dependant Task with Different Arrival Time to meet Deadlines* », Modeling and Performance Evaluation Computer Systems, E. Geslenbe ED., pp. 57-65, 1976.

[Car 96]: F. Vascques de Carvalho, « *Sur l'intégration de mécanismes d'ordonnancement et de ommunication dans la sous-couche MAC de réseaux locaux temps réel* », rapport de Thèse de Doctorat, Université de Toulouse, 1996.

[Cho 92]: Chorus Systèmes, « *CHORUS/Kernel V3 r4.0 Specification and Interface* », Technical report, CHORUS Systèmes, Saint-Quentin-en-Yvelines,78, France, 1992.

[CM 94] : C. Cardeira, Z. Mammeri, « *Ordonnancement des tâches dans les systèmes temps réel et répartis - Algorithmes et critères de classification* », revue APII, Vol. 28, n°4, pp. 353-384, 1994.

[CSB 90]: M. Chetto, M. Silly, T. Bouchentouf, "*Dynamic scheduling of real-time tasks under precedence constraints*", J. of Real-Time Systems, 2, pp. 181-194, 1990.

[Del 94] J. Delacroix, « *Stabilité et Régisseur d'ordonnancement* », Technique et Science Informatique, 13(2):223-250, 1994.

[DRSK 89]: A. Damm, J. Reisinger, W. Schwabl & H. Kopetz, « *The Real-Time Operating System of MARS* », ACM Operating Systems Review, 23(3), july 1989, pp. 141-157.

[Ell91]: J.P. Elloy, "*Les contraintes de temps réel dans les systèmes industriels répartis*", RGE 2, pp. 26-30, 1991.

[EVC 96]: J. Echague, J. Vila, A. Crespo, "*Providing Generalized Rate Monotonic Scheduling Theory to I/O Abstractions over Timed Token Protocol MAC Networks*", Proc. of IEEE Real-Time Syst. Symp., pp.107-110, Montréal, 1996.

[FM 96] : P. Fonseca, Z. Mammeri, *« A Framework for the Analysis of non-deterministic Clock Synchronisation Algorithms* » Distributed Algorithms, LNCS n°1151, pp. 159-174, 1996.

[Gom 93] : H. Gomaa, « *Software Design Methods for Concurrent and Real-Time Systems* », Ed. Addison Wesley, 1993.

[Hen 75]: R. Henn, *« Deterministic Modelle fur die Prozessorzuteilung in einer harten Realzeit-Umgebung* », Phd Thesis, Technical Univ. Munich, 1975.

[HMT 94] : J. He, Z. Mammeri, J. P. Thomesse, *« Modélisation des Techniques de Synchronisation d'Horloges* », Revue RAIRO Technique et Science Informatiques, vol. 13, n° 2, pp. 185-221, 1994.

[HR 94]: N. Homayoun, P. Ramaritham, " *Dynamic Priority Scheduling of Periodic and Aperiodic Tasks in Hard Real-Time Systems*", The Journ. of Real-Time Systems, 6, pp. 207-232, 1994.

[IEEE 85]: *Carrier Sense Multiple Access with Collision Detection* (CSMA/CD), IEEE std 802.3, 1985.

[IEEE 802.5]: *Token Ring Access Method and Physical Layer Specification*, IEEE Std 802.5, 1985.

[ISO 84] : ISO 7498, *Information Processing Systems, Open System Connection*, Basic Reference Model, 1984.

[ISO 94]: International Standard Organization, "*Road Vehicles - Low Speed Serial Data Communication-Part2:Low-Speed Controller Area Network (CAN)*", ISO 11519-2, 1994.

[Kai 82]: C. Kaiser, « *Exclusion mutuelle et Ordonnancement par priorités* », TSI 1 (1), pp. 59-68, 1982.

[KG 94]: H. Kopetz, G. Grünsteidel, "*TTP Protocol for Fault-Tolerant Real-Time Systems*", IEEE Computer, pp. 14-23, jan. 1994.

[KLR 94]: H. Klein, J. Lehoczhy, R. Rajkumar, "*Rate monotonic analysis for real-time industrial computing*", Computer 1, pp. 24-33, 1994.

[Kop 86]: H. Kopetz, « *Scheduling in Distributed Real-Time Systems* », Proc. of Advanced Seminar on Real-Time Local Area Networks, Bandol, France, Apr. 1986, pp. 105-126.

[KRP 94 *et al.*]: M. H. Klein, T. Ralya, B. Pollok, R. Oberza, M. G. Harbour, "*A Practioner's Handbook for Real-Time Analysis*" Kluwer Academic Publishers 1994.

[KS 98] C. Kaiser, C. Santellani, « *Pétrarque. Plate forme d'expérimentation pour l'ordonnancement adaptatif temps réel strict d'applications réparties* », Revue Technique et Science Informatique, 1998, 25 p., (à paraître).

[KSY 84] : J. Kurose, M. Schwartz & Y. Yemini, *« Multiple-access protocols and time-constrained communication* », Computing Surveys, 16(1), pp. 43-70, march 1984.

[Lam 78] : L. Lamport, « *Time, Clocks and Ordering of Events in Distributed Systems* », Comm. of ACM, vol. 21, n°7, pp. 558-565, 1978.

[Let 92]: P. Leterrier, "*Le protocole FIP*", Centre de compétence de FIP, Nancy, 1992.

[LL 73]: C. L. Liu, J. W. Layland - *Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment*- Journal of the ACM, Vol. 20, n°1, pp. 46-61, 1973.

[LM 80] : J. Y. T. Leung, M. L. Merrill, « *A note on Preemptive Scheduling of Periodic Real-Time Tasks* », Information Processing Letters, vol. 11 n°3, pp. 115-118, 1980.

[LR 94]: G. Le Lann & N. Rivierre, " *Real-time communications over broadcast networks: the CSMA-DCR and the DOD-CSMA-CD protocols*", RTS'94, pp.67-84, 11-14 jan. 1994, Paris (France).

[LRD 93 *et al*.]: J. W. S. Liu, J. L. Redondo, Z. Deng, T. S. Tia, R. Bettati, A. Silberman, M.Storch, R. Ha & W. K. Shih, « *PERTS: a Prototyping Environment for Real Time Systems* », Proc. of the14th IEEE Real Time Systems Symposium, North Carolina, pp. 184-188, 1993.

[LW 82]: J. Y. K. Leung, J. Whitehead - *On the complexity of Fixed-Priority Scheduling of Periodic Tasks* - Performance Evaluation, Vol. 2, pp. 273-250, 1982.

[Mam 97] : Z. Mammeri, « *Réseaux et Temps Réel - Ordonnancement de Messages* », Actes de l'Ecole d'Eté Temps réel, pp. 62-79, 22-26 sept. 1997, Poitiers (France).

[Mar 94]: P. Martineau, "*Ordonnancement en ligne dans les systèmes informatiques temps réel*", thèse de doctorat, LAN, Ecole centrale de Nantes, 1994.

[MZ 95]: N. Malcolm, W. Zhao, "*Hard real-time communication in multiple-access networks*", Real-Time Systems, Kluwer Academic Publisher, 8(1), 35-77, jan. 1995.

[NFa 90]: *FIP bus for exchange of information between transmitters, actuators and programmable controllers*, data link layer, NF C46-603, june 1990.

[Raj 91]: R. Rajkumar, "*Synchronisation in Real-Time Systems: A priority Inheritance Approach*", Kluwer Academic Publishers, 1991.

[Ric 83]: E. Rich, « *Artificial Intelligence* », MacGraw Hill, 1983.

[Roz 91]: M. Rozier, « *Construction des systèmes d'exploitation répartis* », INRIA Collection didactique, chapitre 10, Rocquencourt, France, 1991.

[SC 95]: S. Saad, F. Cottet, " *Synthèse bibliographique sur le réseau de terrain FIP, Comparaison avec les autres réseaux à protocole déterministe*", RR. 95003, LISI, ENSMA, 44p., 1995.

[SJ 87]: K.C. Sevcik, M.J. Johnson, "*Cycle time properties of the FDDI token ring protocol*", IEEE Trans. on Soft. Eng., 13(3), 376-385, 1987.

[Son 96]: Y. Q. Song, "*Performance Analysis of Periodic and Aperiodic Real-Time Message Transmission in FIP Networks*", Proc. of the 9th Inter. Conf. on Parallel and Distributed Computing Systems, pp. 499-506, sept. 1996, Dijon(France).

[SR 91]: J. A. Stankovic & K. Ramamritham, *« The Spring Kernel: A New Paradigm for Real-Time Systems* », IEEE Software, May 1991, pp. 62-72.

[SRL 90]: L. Sha, R. Rajkumar, J. Lehoczky, *« Priority inheritance protocols: an approach to real-time synchronisation* », IEEE Transaction Computers, Vol. 39 n°9, pp. 1175-1185, 1990.

[SRS 94]: L. Sha, R. Rajkumar, S. S. Sathaye, « *Generalized Rate-Monotonic Scheduling Theory : A Framework for developing Real-Time Systems* », Proc. of the IEEE, Vol. 82, n°1, jan. 1994.

[SS 96a]: F. Simonot, Y. Q. Song, "*Real-time communicating using TDMA-based multi-access protocol*", Computer Communications (à paraître).

[SS 96b]: Y. Song, F. Simonot, "*Messages scheduling in FDDI for real-time communication"*, RTS'96 actes de conférences, 287-301, Paris, Jan. 1996.

[SSB 97]: Y. Q. Song, F. Simonot-Lion, P. Belissent, "*Validation des applications temps réel distribuées autour du réseau CAN à l'aide de l'évaluation de performances*", RTS'97, pp. 243-260, Paris, jan. 1997.

[SSL 89]: B. Sprunt, L. Sha, J. Lehoczky, *« Aperiodic Task Scheduling for Hard Real Time Systems* », Real Time Systems, Vol. 1, pp. 27-60, 1989.

[Sta 89] : J. A. Stankovic, « *Decentralised Decision Making for Task Reallocation in Hard Real-Time Systems* » , IEEE Trans. On Comp., Vol. C-38, n°3, pp. 341-355, 1989.

[TB 94]: K. Tindell, A. Burns, "*Guaranteeing message latencies on Control Area Network (CAN)*", 1st International CAN Conference, Maintz (Germany), sept. 1994.

[TBW 95]: K. Tindell, A. Burns, J. Wellings, " *Analysis of real-time communications*", J. of Real Time Systems 9, pp. 147-171, 1995.

[TH 95]: K. Tindell, H. Hansson, "*Real-Time Systems and Fixed Priority Scheduling*", Report of Uppsala Univ., oct. 1995.

[Tho 93] : J. P. Thomesse, « *Le réseau de terrain FIP* », Réseaux et Informatique Répartie, Vol. n°3, 1993, pp. 287-321.

[ZB 95]: S. Zhang, A. Burns, "*Guaranteeing synchronous message sets in FDDI networks*", DCCS'95, 107-112, Toulouse(France), sept. 1995.

[Zim 81 *et al*.]: H. Zimmerman, J. S. Banino, A. Caristan, M. Guillemont & G. Morisset, «*Basic concepts of the support of distributed systems: the Chorus approach* », Proc. 2nd Conf. Distributed Computing Systems, apr. 1981, pp. 60-66.

[ZR 87]: W. Zhao & K. Ramamritham, « *Virtual Time CSMA Protocols for Hard Real-Time Communication* », IEEE Trans. on Soft. Eng., vol. 13, n° 8, pp.938-952, Aug. 1987.

[ZS 95]: Q. Zheng & K. G. Shin, " *Synchronous Bandwidth Allocation in FDDI Networks*", IEEE Trans. on Parallel and Distributed Systems, 6(12), 1332-1338, dec. 1995.

[ZSR 90]: W. Zhao, J.A Stankovic & K. Ramamritham, - *A Window Protocol for Transmission of timed-Constrained Messages* -IEEE Trans. on Comp., Vol. 39 n° 9, sept. 1990, pp.1186-1203.

[Ray 92] : M. Raynal, « *Synchronisation et Etat global dans les Systèmes Répartis* », Collection des Etudes et Recherches d'Electricité de France, Ed. Eyrolles, 1992.

INDEX