Etude de l'ordonnancement conjoint de tâches et de messages : application à l'exécutif OSEK/VDX et au réseau CAN

Gérald Cabus

Etude de l'ordonnancement conjoint de tâches et de messages : application à l'exécutif OSEK/VDX et au réseau CAN

étude réalisé au sein de l'équipe TRIO
(Temps Réel et InterOpérabilité) du LORIA
(Laboratoire Lorrain de Recherche en Informatique et ses Applications)

Mémoire de DEA d'i nformatique

présenté et soutenu le 17 septembre 1999

par

Gérald CABUS

Composition du jury :

Jean-Paul Haton	Professeur de l'université Nancy I
Michaël Rusinowitch	Directeur de recherche INRIA
Didier Galmiche	Maître de conférence Nancy I
Antony Vignier	Maître de conférence Nancy I
Ye-Qiong Song	Maître de conférence Nancy I
Bruno Gaujal	Chargé de recherche INRIA

Remerciements

Je tiens tout d' abord à remercier très sincèrement mes tuteurs : Monsieur Ye-Qiong SONG et Madame Françoise Simonot-Lion pour leur encadrement durant ce stage.

Je remercie tout particulièrement Nicolas Navet pour sa perpétuelle disponibilité à répondre à mes questions les plus diverses.

Toute ma gratitude pour Fabrice Jumel en compagnie duquel j' ai passé ces cinq mois de stage.

J' associe enfin le reste de l' équipe pour leur accueil chaleureux et tous les bons moments que nous avons passés ensemble.

Et pardon à toutes les personnes involontairement oubliées

Résumé

Dans les systèmes temps réel, les tâches et les messages sont en principe ordonnancés grâce à leur priorité. Une fois ces priorités affectées et connaissant les caractéristiques des tâches, du réseau et des messages, on sait calculer analytiquement le pire temps de réponse de ces tâches et celui des messages. Cependant, contrairement au cas uniprocesseur, il n' existe pas d' algorithme optimal pour affecter ces propriétés dans le cas distribué. En effet, les calculs du pire temps de réponse des tâches et des messages sont interdépendants. On parle alors de transactions, lorsque des messages et des tâches sont liés : d' une part les messages sont toujours crées par les tâches et d' autre part certaines tâches sont activées par des messages. Le calcul des temps de réponse dans le pire cas de ces transactions nécessite alors la connaissance globale du système. L'u tilisation d' algorithmes génétiques couplée à celles d' heuristiques permet pour des systèmes complexes de trouver une ou plusieurs solutions vérifiant les contraintes temporelles du système.

Abstract

In the real time systems, tasks and messages are scheduled in theory thanks to their priority. Once these priorities are affected and if we know the characteristics of the tasks, of the network and of the messages, we can analytically calculate the worst response time of the tasks and of the messages. However, contrary to the uniprocessor case, there is no optimal algorithm to affect these priorities in the distributed case. Indeed, calculations of the worst response time of the tasks and the messages are interdependent. One speaks then about transactions if messages and tasks are dependent : on the one hand the messages are always create by the tasks and on the other hand some tasks are activated by messages. Then the calculation of the response times in the worst case of these transactions requires the total knowledge of the system. The use of genetic algorithms coupled to those of heuristic makes possible to find one or more solutions checking them for complex systems.

Sommaire

Introduction

Contexte du stage et problématique

I. Contexte du stage

J' ai effectué mon stage DEA au sein du LORIA dans l' équipe TRIO. Les travaux de cette équipe se situent dans le cadre de la conception d' applications temps réel distribuées. La problématique générale de cette équipe se décompose en trois axes complémentaires :

- les procédés de construction de modèles d' architecture, afin d' une part, de les exploiter pour la preuve de propriétés et d' autre part, de construire et d'engendrer le squelette de cette application,
- les méthodes de vérification de propriétés temporelles d' une architecture opérationnelle par exploitation de modèles de cette architecture,
- la spécification de mécanismes exécutifs tels qu' ils permettent aux applications supportées de respecter les contraintes de temps du cahier des charges.

Mon travail s' insère à la fois dans les axes de vérification de propriétés temps réel et de spécification de mécanismes exécutifs.

Cette étude s' inscrit dans le cadre de projets de l' équipe TRIO liés à l' automobile :

- le projet Carosse : évaluation de performances d' applications embarquées 1998-2000.
- le projet AEE (Architecture Electronique Embarquée) : outils et méthodes de développement d' applications embarquées modulaires et sûres 1998-2001.

Les réalisations pratiques de ce stage se sont inscrites dans le cadre d' un stage ingénieur ENSEM.

II. Problématique du stage

L'ob jectif de ce stage est de fournir un moyen de déterminer une configuration correcte et éventuellement optimale de tâches et de messages d' un système distribué, et plus précisément d' un système distribué embarqué dans l' automobile. Les paramètres à calculer sont les priorités des tâches et des messages.

La conception d' applications temps réel réparties passe inévitablement par une étape de 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 mais aussi des messages échangés entre ces tâches.

Cette validation peut être réalisée de différentes manières :

- soit par la construction d' un prototype et la prise de mesures sous différentes conditions de test.
- soit par la simulation.
- soit par l' approche analytique.

Les deux dernières approches nécessitent de construire un modèle (exécutable ou analysable) de l' application et se situent avant l' implantation réelle.

Dans le domaine automobile, on assiste souvent à la construction de prototypes pour valider un système. Cependant une telle approche est très onéreuse et ne devrait donc être utilisée que lorsque le système a déjà subi de nombreuses vérifications au préalable.

Ces vérifications proviennent le plus souvent de simulations, mais l' automobile s' ouvre de plus en plus vers d' autres solutions comme la validation analytique.

L' approche analytique repose sur le calcul du pire temps de réponse de chaque tâche et de chaque message. On peut alors vérifier si les échéances sont respectées ou non. Dans un système à priorité fixe, si les échéances ne sont pas respectées, il est alors nécessaire de changer la politique d' ordonnancement des tâches et des messages i.e. leur priorité. De plus lorsqu' une solution ordonnançable a été trouvée, il peut être intéressant de rechercher une autre solution plus performante selon certains critères (taux d' utilisation du processeur sur chaque site...).

Mon travail a consisté :

- sur le plan théorique : en une étude analytique du pire temps de réponse et en la proposition d' une méthode d' affectation des priorités des tâches et des messages.
- sur le plan pratique : d' une part, à la conception d' un outil de calcul du pire temps de réponse, et d' autre part, à la conception d' un algorithme génétique.

Ainsi dans la suite, dans un premier temps nous rappellerons les connaissances de base sur un système temps réel réparti, en particulier en rappelant les caractéristiques propres aux systèmes temps réel, certaines politiques d' ordonnancement et certaines propriétés des réseaux de communication. Dans un second temps nous réaliserons l' étude analytique des pires temps de réponse des tâches, puis des messages, et enfin des deux simultanément pour l' appliquer sur un exemple de système. Dans un troisième temps, nous proposerons une méthode basée sur les algorithmes génétiques pour affecter les priorités aux tâches et messages afin qu' ils respectent leurs échéances. Enfin nous conclurons sur ce stage.

Contexte et état de l' art

I. Présentation générale d' un système temps réel

I.1. Définition

Un système est en général composé de deux parties [Bayard 95] :

- le système contrôlé ou surveillé appelé aussi partie opérative. Celui-ci correspond à un processus physique dont l' état évolue au cours du temps
- le système contrôlant appelé aussi partie commande. Ce système informatique interagit avec le système physique en contrôlant l' état du processus physique par le biais de capteurs, puis en calculant à partir de ces dernières mesures la prochaine action, et enfin en appliquant cette consigne au système contrôlé par l' intermédiaire d' actionneurs.

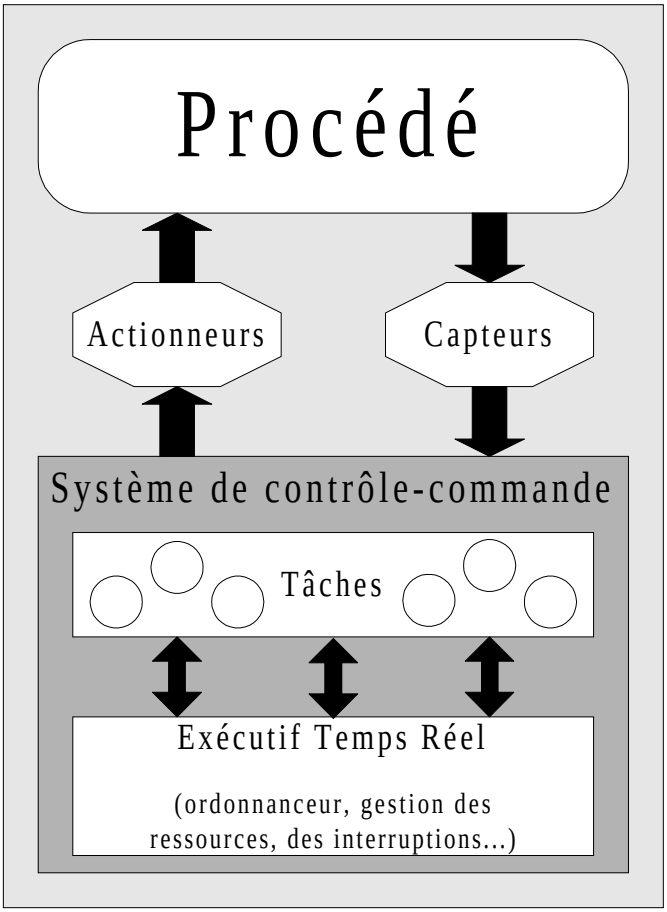


Figure 1 - Structure d' un système temps réel sur un site

Un système est qualifié de temps réel lorsque sa correction et sa validité dépendent non seulement de la justesse des résultats obtenus mais aussi des instants de production de ces résultats. En effet les différentes activités d' un tel système doivent nécessairement s' exécuter dans des laps de temps limité.

Le respect des contraintes temporelles est un aspect fondamental et spécifique aux systèmes temps réel, ce qui constitue leur caractéristique principale et qui les distingue des systèmes informatiques classiques (non temps réel).

Les systèmes temps réel se subdivisent en trois classes majeures [Stankovic 88] :

- Les systèmes temps réel "strict" i.e. à contraintes strictes (hard real time) qui sont des systèmes pour lesquels le non-respect d' une échéance peut engendrer une catastrophe (la destruction du système ou la mise en danger de l' intégrité physique de personnes).
- Les systèmes temps réel "souple" i.e. à contraintes relatives (soft real time) qui, par opposition, sont des systèmes qui tolèrent les dépassements d' échéances jusqu' à un certain seuil.
- Les systèmes temps réel "mixte" à contraintes mixtes qui sont des systèmes qui comprennent à la fois des sous-systèmes à contraintes strictes et des sous-systèmes à contraintes relatives.

Un système de contrôle-commande numérique consiste en une architecture matérielle supportant un ensemble de logiciel permettant d' interagir sur le système physique contrôlé par le biais de capteurs et d' actionneurs. Cette structure matérielle correspond donc à l' ensemble des ressources physiques (processeurs, cartes d' entrées-sorties...).

Cette architecture matérielle peut être :

- Monoprocesseur : tous les programmes s' exécutent sur un seul processeur en parallélisme apparent.
- Multiprocesseur : tous les programmes sont répartis sur plusieurs processeurs partageant une même mémoire commune.
- Répartie : tous les programmes sont répartis sur plusieurs processeurs dépourvus de mémoire commune et d' horloge commune. Ils sont reliés par un médium de communications par lequel ils peuvent communiquer par échanges 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 communications 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 répartis.

La partie logicielle du système de contrôle-commande comprend :

- d' une part, un système d' exploitation, appelé exécutif temps réel chargé de fournir un ensemble de services que les programmes de l' application peuvent utiliser durant leur exécution. Il implante une stratégie pour partager le temps processeur (appelé ordonnancement) dans un système multitâches entre les différentes activités en attente d' exécution en favorisant celles qui ont la plus forte priorité.
- d' autre part, des programmes sont implantés par l' utilisateur et traduisent les fonctions que doit réaliser le système de contrôle commande pour la prise en compte de l' état du procédé et sa commande. Ces programmes sont exécutés par un ensemble d' unités d' exécution appelées tâches.

I.2. Les tâches

Les tâches peuvent être indépendantes ou 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.

Comme nous l' avons vu précédemment, dans un système temps réel, les tâches doivent respecter des délais critiques imposés par les dynamiques de l' environnement à contrôler. Ainsi d' un point de vue temporel, on prend compte que les tâches périodiques et leur triplet d'attributs (C_i, D_i, P_i) [Cardeira 94] où :

- C_i correspond à la durée d' exécution de la tâche seule sur le processeur.
- D_i correspond à l' échéance, qui permet de fixer l' intervalle de temps durant lequel la tâche doit finir son exécution. En dehors de cet intervalle son exécution devient inutile.
- P_i correspond à la période, c' est l' intervalle de temps fixe qui sépare deux arrivées successives d' une tâche. Ce paramètre concerne les tâches activées par un signal périodique provenant d' une horloge temps réel interne. Ce type de tâches est appelé tâches périodiques.

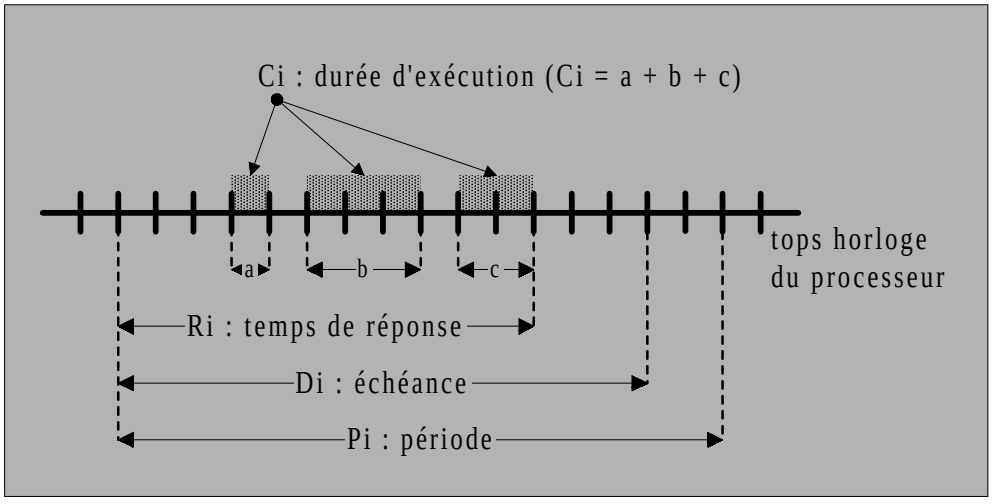


Figure 2 - Les caractéristiques temporelles d' une tâche périodique

Il existe d' autres types de tâches : les tâches apériodiques et les tâches sporadiques.

Les tâches apériodiques correspondent à un service dont l' exécution est déclenchée par un événement aléatoire qui provient soit du procédé, soit par une action interne du système mais sans aucune synchronisation avec ce dernier.

Les tâches sporadiques sont comme des tâches apériodiques mais on constate en plus l' existence d' une durée minimale entre deux événements apériodiques (issus de la même source). Pour considérer le pire cas, il suffit alors de les traiter comme des tâches périodiques de période égale à cette durée minimale entre deux activations.

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 périodiques. On commence par ordonnancer l' ensemble des tâches périodiques afin de déduire des temps creux où seront insérées les tâches apériodiques et sporadiques.
- une autre solution consiste à ordonnancer les tâches apériodiques sur le même plan que les tâches périodiques. Une tâche périodique appelée serveur est dédié à l' ordonnancement des tâches apériodiques. Ce serveur peut être à scrutation ou ajournable.

De plus une tâche peut être préemptible ou non. On dit qu' une tâche est non préemptible si lorsqu' elle a commencé son exécution, elle doit être achevée avant qu' une autre tâche obtienne le processeur. Inversement une tâche est préemptible lorsqu' elle peut être interrompue au profit d' une autre et être reprise ultérieurement.

I.3. Application au domaine automobile

Les systèmes embarqués dans l' automobile sont supportés par desarchitectures matérielles composées de sites distribués (capteurs, calculateurs, actionneurs) qui sont reliés par des réseaux. Dans le cas de PSA (Peugeot Société Automobile), deux réseaux aux objectifs différents sont utilisés :

- le réseau CAN (Controller Area Network) est le réseau de contrôle-commande lié au moteur, dont les contraintes temporelles sont strictes. [ISO 94, CAN]
- le réseau VAN (Vehicle Area Network) est le réseau de l' habitacle qui gère le confort des occupants et disposera bientôt des fonctionnalités multimédias (navigation par satellite...) Les contraintes temporelles associées y sont généralement moins critiques. [VAN]

Les fonctions (ou tâches) s' exécutent de façon parallèle sur des sites répartis, la synchronisation et la coordination se font par échanges de messages. Ce système doit respecter un certain nombre de contraintes strictes (temporelles, sûreté de fonctionnement,...) et la conséquence du non-respect peut être grave, voire catastrophique vis à vis du bon fonctionnement du véhicule et de la sécurité des passagers. Par exemple le changement de rapport de boîte de vitesse doit provoquer un changement au niveau du couple moteur et ceci doit être borné dans une fenêtre temporelle donnée.

La validation (vérification du respect des contraintes temporelles et de sécurité de fonctionnement) a priori d' un tel système est donc primordiale et la recherche de méthode/outils pour cette validation suscite beaucoup de travaux de recherche et de projets de grandes envergures. [SONG 96].

Enfin dans le domaine automobile les contraintes de coûts sont très sensibles du fait de la production de série, cela justifie que l' on cherche à rentabiliser au mieux les équipements intervenant dans la fabrication d'un e automobile. Ainsi, une dernière préoccupation des constructeurs est de bien choisir et de dimensionner les microcontrôleurs utilisés pour leurs véhicules, i.e. ceux dont le prix soit le plus bas et dont la puissance de calcul permette d' exécuter leurs applications. On est donc devant un domaine où l' optimisation des performances revêt une importance non négligeable [Courrier 98].

II. Les différents algorithmes d' ordonnancement

Plusieurs tâches peuvent concourir pour l' obtention du processeur. L' attribution du processeur aux tâches i.e. 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'o rdonnanceur 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 le sujet reposent sur la modélisation des tâches et la définition d' algorithmes d' ordonnancement. [Cardeira 94]

Les algorithmes d' ordonnancement peuvent être qualifiés d' ordonnancement hors ligne ou d' ordonnancement en ligne.

- Le premier type d' ordonnancement nécessite la connaissance de toutes les tâches et de leurs caractéristiques temporelles 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 alors à une entité appelée répartiteur qui ne fait que lancer les tâches dans l' ordre dicté par la séquence. C' est la solution actuellement employée dans l' automobile. Evidemment une telle approche ne permet pas de prendre en compte les tâches dont la date d' activation n' est pas connue au départ.
- Le second type d' ordonnancement est l' ordonnancement en ligne qui construit au fur et à mesure du déroulement de l' application, la séquence d' exécution à partir des tâches prêtes à l' instant courant. Ceci permet entre autre la prise en compte des tâches apériodiques. De tels ordonnancements attribuent le processeur à une tâche en fonction d' un algorithme d' ordonnancement. Ce dernier est conduit par priorité s' il exécute d' abord la tâche la plus prioritaire. La priorité est soit affectée par le concepteur de l' application en fonction de la criticité des tâches, soit assignée de manière automatique en fonction d' un paramètre temporel caractérisant les tâches tel que la période ou l' échéance qui reflètent l' urgence et non l' importance de la tâche au sens applicatif.

Compte tenu de la complexité des applications et de l' émergence de l' exécutif OSEK/VDX [OSEK], cette approche doit être maintenant étudiée dans le domaine automobile. De plus cet exécutif temps réel permet d' implanter des tâches préemptibles ou non avec leur ordonnancement [George 96].

Les ordonnancements à priorité statique et les ordonnancements à priorité dynamique sont ceux que nous étudions ci-dessous.

II.1. Les algorithmes d' ordonnancement à priorité fixe

Un tel algorithme affecte à chaque tâche une priorité qu' elle conserve durant toute la vie de l' application.

II.1.a. L'algorithme Rate Monotonic [Liu 73]

C' est un des algorithmes de base dans la littérature sur l' ordonnancement dans les systèmes temps réel. L' algorithme s' applique à des tâches périodiques sur un seul processeur. 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 pour les algorithmes à priorité fixe et pour les tâches à échéance sur requête, i.e. qu' il permet d' ordonnancer toute application ordonnançable par un autre algorithme de même classe.

Une condition suffisante d' ordonnançabilité pour n tâches est [Layland 73] :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq n\left(2^{1/n} - 1\right)$$ avec $n(2^{1/n}-1)$ qui converge vers ln2 = 0,69 pour n grand

II.1.b. L'algorithme Deadline Monotonic

La plus grande priorité est associée à la tâche possédant la plus petite échéance. Pour les tâches de même échéance, l' affectation de priorité se fait de manière aléatoire. En échéance sur requête l' algorithme est équivalent à l' algorithme Rate Monotonic. L' algorithme est optimal pour des tâches indépendantes.

Une condition suffisante d' ordonnançabilité pour n tâches est [Layland 73] :

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$$

Ceci permet de savoir à tout instant si la configuration peut accepter une surcharge ou non. Un algorithme d' acceptation de tâches en ligne peut donc être implanté.

II.2. Les algorithmes d' ordonnancement à priorité dynamique

Un tel algorithme recalcule la priorité de chaque tâche durant leur exécution. Ainsi au cours de la vie de l' application la priorité de la tâche évolue.

II.2.a. L' algorithme Earliest Deadline First

Cet algorithme ordonne les tâches selon les échéances croissantes, i.e. la tâche en attente ayant l' échéance la plus proche obtient le processeur. Cet algorithme est optimal pour les systèmes sans partage de ressources.

Une condition suffisante d' ordonnançabilité pour n tâches est :

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1$$

De plus si les tâches sont périodiques, indépendantes à échéance sur requête, alors on obtient la condition nécessaire et suffisante suivante :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

Il a été démontré que l' algorithme qui engendre le moins de préemptions donc le moins de commutation de contexte, parmi les algorithmes précédents est EDF [Henn 75]. Il utilise donc le mieux le temps CPU.

II.2.b. L' algorithme Least Laxity First

Cet algorithme est aussi un algorithme dynamique. Si on note t l' instant courant, D l' échéance de la tâche et e son temps d' exécution alors la laxité de cette tâche est l, définie par l=D-t-e. La tâche avec la plus faible laxité obtient le processeur et si deux tâches ont la même laxité alors le choix est arbitraire (habituellement on choisit celle qui était en cours d' exécution ou on se base sur un autre critère de priorité).

Contrairement à EDF où les échéances ne varient pas, la laxité évolue au cours du temps selon certaines règles :

- l(t+1) = l(t) pour toute tâche en cours d' exécution
- l(t+1) = l(t) – 1 pour toute tâche en attente.

Ainsi l' ordonnanceur doit pas seulement être exécuté lors des temps d' activation de tâches ce qui conduit à plus de préemption qu' avec l' algorithme EDF.

Une condition nécessaire et suffisante d' ordonnançabilité est : toute configuration de n tâches périodiques indépendantes à échéance sur requête est fiablement ordonnancée par Least Laxity First si et seulement si :

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

II.3. Résolution de l' interblocage et de l' inversion de priorité

II.3.a. L' inversion de priorité

Un problème d' inversion de priorité peut intervenir dans un système utilisant des sémaphores. Cette situation correspond au cas ou une tâche de priorité supérieure est bloquée par une tâche de priorité inférieure pour un temps indéterminé (cette tâche de faible priorité peut être préemptée par une tâche de priorité moyenne ce qui retarde encore plus la tâche de forte priorité)

Une solution consiste à utiliser un algorithme appelé protocole d' héritage de priorité (Priority Inheritance Protocol proposé par Sha, Rajkumar et Lehoczky [Sha88]) qui est invoqué lorsqu' une tâche de forte priorité est bloquée par une tâche de plus faible priorité. Quand une telle tâche bloque l' exécution d' une tâche de plus forte priorité, la tâche bloquante hérite de la priorité de la tâche bloquée. Ainsi les tâches de priorité moyenne ne peuvent pas retarder l' exécution de la tâche bloquante. Aussitôt que cette dernière n' est plus bloquante elle reprend sa priorité initiale.

Ainsi une tâche ne peut être bloquée qu'une fois par chaque tâche de plus faible priorité et la durée de chaque blocage est limitée à la durée de la section critique. Cependant plusieurs blocages peuvent être enchaînés et de plus cet algorithme n' empêche pas l' interblocage.

II.3.b. L' interblocage

L' interblocage est un phénomène qui intervient lorsqu' une tâche de priorité faible prend un sémaphore S1, puis est préemptée par une tâche de priorité supérieure qui prend le sémaphore S2, puis qui désire prendre S1, alors elle se trouve bloquée et alors la tâche de priorité inférieure obtient le processeur et demande S2 donc elle se trouve alors aussi bloquée, les deux tâches se bloquent mutuellement.

Une solution consiste à utiliser un algorithme appelé protocole à priorité plafond (Priority Ceiling Protocol proposé par Sha, Rajkumar et Lehoczky [Sha90]) qui à chaque sémaphore associe la priorité de la tâche de plus forte priorité pouvant accéder à ce sémaphore. Une tâche peut prendre un sémaphore si sa priorité est supérieure au plafond de tous les sémaphores bloqués. Ainsi une tâche peut être bloquée au maximum par la section critique d' une seule autre tâche. L' idée sous-jacente est de créer un ordre total d' exécution avec prise en compte des sections critiques. De plus on règle l' inversion de priorité.

III. 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 communications sont dits réseaux temps réel.

En comparaison avec les réseaux classiques, le temps de réponse des messages échangés est borné, leur architecture de communication est souvent 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 multiples (FIP, CAN,...) 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 conflits 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 [Mammeri 97].

III.1. Architecture

L' architecture la plus générale d' un réseau de communication est une architecture à sept couches basées sur le modèle OSI [Zimmerman 80]. Néanmoins, dans le cadre des réseaux locaux temps réel [Mammeri 91], une architecture de communication limitée à trois couches suffit : la couche physique, la couche liaison de données elle-même décomposée en deux sous-couches LLC et MAC et enfin la couche application. Pour le réseau CAN, seules les deux premières couches sont normalisées.

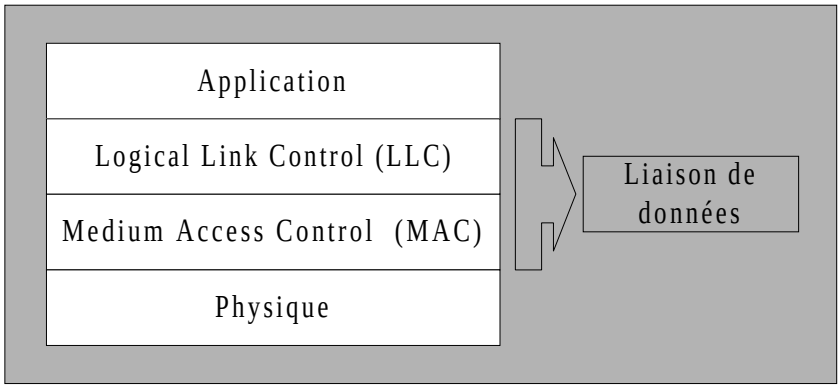


Figure 3 - Architecture d' un réseau de communication temps réel

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 liaisons 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, à ce titre c' est la couche la plus importante de l' architecture car on y décide la possibilité de transmettre un message.

La couche application fournit les services nécessaires à la gestion des tâches et du contrôle commande.

III.2. 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).

On distingue deux types de messages :

- les messages périodiques ou synchrones sont caractérisés par un intervalle d' interarrivée 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 apériodiques ou asynchrones sont souvent utilisés pour véhiculer une information de type alarme ou une communication entre tâches apériodiques. Contrairement aux messages périodiques, les messages apériodiques arrivent à des instants non fixés.

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

III.3. Ordonnancement de 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 borné. L' adaptation de la couche MAC en vue de prendre en compte les contraintes de temps reste encore au stade de la recherche.

L' accès au médium peut être réalisé de manière différente, chacune correspondant à une classe de protocole de communication :

- multiplexage temporel, chaque site dispose d' une tranche de temps dans un cycle répétitif (TDMA par exemple)
- Compétition, chaque site émet quand il le désire et doit coopérer avec les autres sites en vue de résoudre les situations de conflit comme les collisions (CSMA/CA pour CAN -cf. annexe 1-)
- Consultation, un site particulier autorise le site qui doit émettre (cas de FIP [Thomesse 93]), ou bien un message particulier appelé jeton parcourt les sites en autorisant à émettre celui qui le détient à un moment donné (cas de FDDI)

Analyse de l' ordonnancement conjoint de tâches et de messages

I. Analyse de l'ordonnancement sur un processeur seul

Grâce à une telle analyse, on cherche à vérifier a priori que chaque tâche du système étudié respectera les contraintes qui lui sont imposées. Ainsi cette analyse repose sur l'évaluation du temps de réponse de chaque tâche dans son pire cas d'exécution possible. Si dans ce pire cas les contraintes sont respectées alors elles le seront toujours quelque soient les conditions de fonctionnement.

Chaque tâche i est définie par différents attributs :

- sa période notée T_i (toute tâche sporadique peut être considérée comme périodique et dans le pire cas cette période est la durée minimale entre deux activations)
- sa priorité fixe
- son temps d'exécution propre notée C_i
- son échéance notée D_i.

A partir de la connaissance de toutes les tâches sur ce processeur on peut déterminer le facteur de blocage B_i qui est défini différemment selon que l'ordonnancement est préemptif ou non [George 96] :

- en mode non préemptif B_i correspond au plus grand temps d'exécution propre parmi les tâches de priorités inférieures :

$$B_i = \max_{j \in lp(i)} C_j$$ où lp(i) représente les tâches moins prioritaires que la tâche i.

- en mode préemptif B_i correspond au temps de blocage du à la prise de ressources critiques par des tâches de priorités inférieures.

Dans un protocole à héritage de priorité, la tâche entrant dans une section critique hérite de la priorité la plus élevé des tâches en attente de cette ressource, on évite l'inversion de priorité mais pas l' interblocage, alors

$$B_i = \max_{j \in lp(i)} \left(\text{durée section critique de Tj}\right) * \max\left(\text{nb sémaphore}, \text{cardinal}(lp(i))\right)$$

Dans un protocole à priorité plafond, la tâche entrant dans une section critique hérite de la priorité la plus élevée de toutes les tâches pouvant accéder à cette ressource, on évite alors aussi l'interblocage, alors :

$$B_i = \max_{j \in lp(i)} \left(\text{durée section critique de Tj}\right)$$

Remarque :

Dans Osek (cf. annexe 2), on peut gérer des tâches préemptibles et non préemptibles simultanément, ainsi le facteur de blocage correspond au maximum des temps de blocage du à la prise de ressources critiques, et des temps d'e xécution des tâches de plus faible priorité non préemptible.

I.1. Etude simpliste

Durant sa période T_i la tâche i ne s'exécute qu'une seule fois, mais des tâches de période différente peuvent s'exécuter plusieurs fois : ce nombre dépend du rapport des périodes. Ainsi une tache j s'exécute : $\left\lceil \frac{T_i}{T_j} \right\rceil$ fois au maximum durant T_i avec $\lceil X \rceil$ la partie entière supérieure de X. (i.e. le plus petit entier supérieur à X)

I.2. Etude approfondie avec l'échéance inférieure à la période

Cette fois pour connaître le nombre d'exécution d'une tâche j pendant une exécution de la tâche i, on raisonne non plus sur la période de la tâche i mais sur son temps d'exécution.

I.2.a. En mode non préemptif

Dans ce cas on évalue le temps d'attente du processeur par la tâche. Le pire temps de réponse R_i est alors donné par les équations :

$$R_i = W_i + C_i$$

$$W_i = \sum_{j \in hp(i)} \left\lceil \frac{W_i}{T_j} \right\rceil . C_j + B_i$$

Pour calculer W_i on peut opérer par récurrence en utilisant la relation suivante qui converge si l'utilisation du processeur est inférieure ou égale à 100% :

$$W_i^{n+1} = \sum_{j \in hp(i)} \left\lceil \frac{W_i^n}{T_j} \right\rceil . C_j + B_i \quad \text{avec } W_i^0 = 0 \text{ comme valeur initiale possible}$$

De plus lorsqu'une tâche passe à l'état prêt, elle n'obtient pas toujours le processeur immédiatement en raison soit d'un tick d'ordonnancement soit de l'attente de l'arrivée d'un message. Ce temps de latence est appelé release jitter (gigue d' activation) et est noté J_i Cette gigue correspond à la différence entre la plus rapide et la plus lente activation d' une tâche relativement à une invocation [Tindell-Hansson 97]. Les équations deviennent alors :

$$R_i = W_i + C_i + J_i$$

$$W_i = \sum_{j \in hp(i)} \left\lceil \frac{W_i + J_j}{T_j} \right\rceil . C_j + B_i$$

Le release jitter est un problème car le temps, dans le pire cas, entre deux débuts d' exécution successifs d' une tâche est plus court que le temps mis dans le pire cas entre deux arrivées d'une tâche.

I.2.b. En mode préemptif

Cette fois lorsque la tâche obtient le processeur, celui-ci peut lui être repris par une tâche de priorité supérieure donc il est nécessaire d'inclure le temps d'exécution propre dans la partie récursive des équations d'où :

$$R_i = W_i + J_i$$

$$W_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i + J_j}{T_j} \right\rceil . C_j + B_i$$

Pour éviter le problème qui correspond à ce que l'activation d'une tâche soit retardée par la non-terminaison de la précédente activation, il faut insister sur le fait que dans l'équation précédente le temps passé avant la fin de l'exécution doit être inférieur à la période de la tâche ($W_i \leq T_i$)

I.3. Etude approfondie avec l' échéance arbitraire

On considère maintenant qu'une tâche peut avoir une échéance supérieure à sa période. De ce fait une nouvelle activation d'une tâche peut intervenir alors que la précédente n'est pas encore finie. Plus précisément une invocation précédente peut interférer avec l' invocation courante et donc le pire temps d'exécution ne peut plus être évalué en considérant une seule activation de la tâche mais en comparant les temps de réponse de la tâche pour toutes ses activations durant sa période d'activation (ou d'interférence) de niveau i. Celle-ci est délimitée par l'obtention du processeur par une tâche de priorité inférieure à celle de la tâche étudiée (ici i). On obtient alors les équations : [Tindell et al]

$$R_i = \max_{q=0,1,2....} (W_i(q) - qT_i + J_i)$$

$$W_i(q) = (q+1).C_i + \sum_{j \in hp(i)} \left\lceil \frac{W_i(q) + J_j}{T_j} \right\rceil . C_j + B_i$$

avec q variant de 0 à n où n correspond à la dernière activation de la tâche durant sa période d'activation

Remarque :

- *ces équations sont valables en mode préemptif (en non préemptif on transforme le $(q+1).C_i$ de l'équation récursive en $q.C_i$ et dans le calcul du maximum de R_i on ajoute C_i).*
- *ces équations sont équivalentes aux équations précédentes dans le cas $W_i \leq T_i$ et convergent aussi dans le cas où le taux d'utilisation du processeur est inférieur à 100%*

Le calcul de l' intervalle de définition de q est réalisé grâce à la période d' occupation de niveau i (level i busy period L_i). Cette période est en fait la durée pendant laquelle seule les tâches de priorités supérieures ou égale à i obtiennent le processeur. Alors q est calculé de telle manière que l' inéquation suivante soit $qT_i < L_i < (q+1)T_i$ vérifiée :

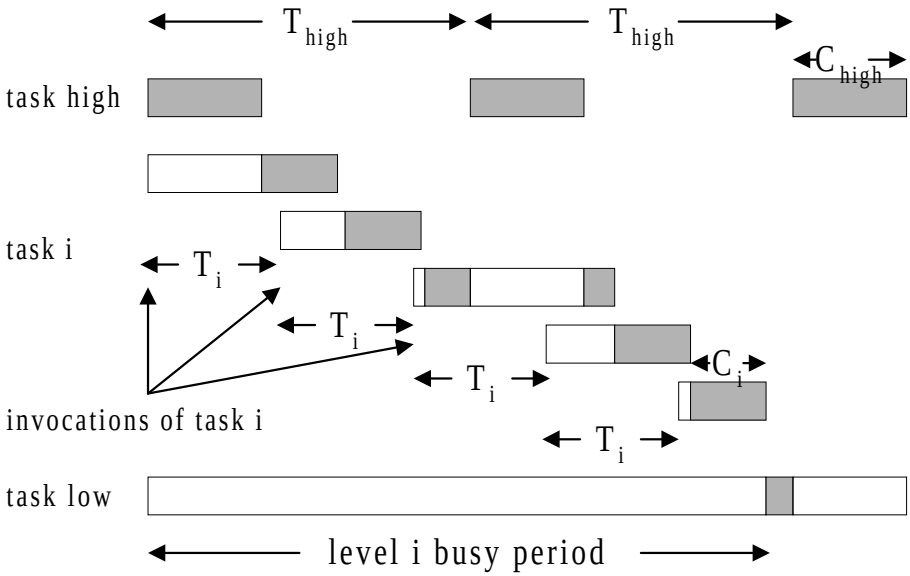


Figure 4 - exemple d' une configuration où différentes invocations de la même tâche sont en concurrence

II. Analyse de l'ordonnancement des communications dans le réseau CAN (Control Area Network)

Dans le réseau CAN, comme les trames ont des priorités distinctes, la trame la plus prioritaire du système à un instant donné est unique, toute trame n'attend que l'émission des trames plus prioritaires dans le réseau dans le pire cas. A partir de ces files de transmission locales aux nœuds du réseau, le protocole CAN en choisissant toujours la trame la plus prioritaire dans le réseau pour l'émettre construit implicitement une file de transmission globale contenant toutes les trames triées. On peut donc raisonner à partir de cette file globale. De plus lorsqu'une trame est sur le réseau elle ne peut être préemptée par la mise d'une trame plus prioritaire dans la file d'attente. En conséquence pour la transmission d'une trame, on se trouve dans le cas de l'étude du temps de réponse dans le pire cas d'une tâche sur un seul processeur sans préemption. On obtient alors le temps de réponse du message. [Tindell et al 95]. Une étude des communications temps réel a été réalisée par Tindell [Tindell 93] dans le cas général.

On supposera dans la suite que chaque message ne comporte qu'une trame.
Chaque message m est définie par différents attributs :

- sa période notée T_m (tout message sporadique peut être considéré comme périodique et dans le pire cas cette période est la durée minimale entre deux demandes d'émission)
- sa priorité fixe
- son temps de transmission propre notée C_m
- son échéance notée D_m.

A partir de la connaissance de tous les messages pouvant être émis sur le réseau, on peut déterminer le facteur de blocage B_m qui correspond au plus grand temps de transmission parmi les messages de priorités inférieures :

$$B_m = \max_{n \in lp(m)} C_n \leq 135\tau_{bit}$$

II.1. Etude approfondie avec l'échéance inférieure à la période

Le temps de transmission de bout en bout est alors solution des équations :

$$R_m = C_m + W_m + J_m - \tau_{bit}$$

$$W_m = \tau_{bit} + \sum_{n \in hp(m)} \left\lceil \frac{W_m + J_n}{T_n} \right\rceil . C_n + B_m$$

où hp(m) représente les messages plus prioritaires que le message m.

Le terme τ_{bit} correspond au fait que dans le réseau CAN entre l'émission de deux messages on attend durant ce temps avant de commencer le processus d'arbitrage. Ce terme correspond au temps d'aller-retour d'un bit sur le réseau. On ôte le τ_{bit} qui correspond au dernier bit émis. A l' aide d' un changement de variable on obtient les équations suivantes : [Tindell-Burns 94]

$$R_m = C_m + W_m + J_m$$

$$W_m = \sum_{n \in hp(m)} \left\lceil \frac{W_m + J_n + \tau_{bit}}{T_n} \right\rceil . C_n + B_m$$

Le terme τ_{bit} correspond alors au temps d' écoute avant l' accès au bus.

On peut donc définir C_m à partir de ce terme et du format de la trame :

$$C_m = \left(47 + 8d + \left\lfloor \frac{34 + 8d}{4} \right\rfloor\right).\tau_{bit}$$ avec d taille en bytes des données dans la trame.

II.2. Etude approfondie avec l' échéance arbitraire

On reprend l'étude faite pour les tâches mais en l'adaptant à un mode non préemptif : on calcul récursivement uniquement le temps d'attente dans la file de transmission. On en déduit donc les équations [Tindell et al 94]

$$R_m = \max_{q=0,1,2....} (W_m(q) - qT_m + J_m + (q+1).C_m)$$

$$W_m(q) = \sum_{n \in hp(m)} \left\lceil \frac{W_m(q) + J_n + \tau_{bit}}{T_n} \right\rceil .C_n + B_m$$

II.3. Etude approfondie avec prise en compte des erreurs de transmission

Pour prendre en compte les éventuelles erreurs de transmission, on peut rajouter le terme $E_m(n)$ dans l'équation récursive comme proposé dans [Navet 99] :

$$R_m(n) = C_m + W_m$$

$$W_m(n) = E_m(n) + B_m + \sum_{j \in hp(m)} \left\lceil \frac{W_m(n) + \tau_{bit}}{T_j} \right\rceil .C_j$$

$$E_m(n) = n.\left(23.\tau_{bit} + \max_{\forall o \in hp(m) \cup \{m\}} C_o\right).\tau_{bit}$$

Le terme $23.\tau_{bit}$ correspond à la trame d' erreur émise et le second terme à la retransmission de la trame corrompue qui dans le pire cas est la trame la plus grande.

On calcule n le nombre d'erreurs par incrémentation jusqu'à ce que l'inégalité suivante soit vérifiée :

$$\left(1 - \sum_{i=0}^{i=n} P\left[X\left(R_m(n)\right) = i\right]\right) \leq \alpha$$

avec X processus stochastique comptant le nombre d'erreurs de transmission sur le temps et α un paramètre représentant la qualité de service désirée. Si durant $R_m(n)$ se produisent moins de n+1 erreurs alors la trame respectera son échéance.

Remarque :

Si l'on suppose qu'un message peut comporter plusieurs trames, il est nécessaire de raisonner comme en préemptif puisque entre l'envoi de deux trames d'un même message d'autres trames peuvent être transmises si elles sont plus prioritaires. On obtient alors les équations suivantes :

$$R_{m,k} = \max_{q=0,1,2....} (W_{m,k}(q) - qT_m + J_m + \rho + \wp + \tau_{notif})$$

$$W_{m,k}(q) = qN_m\rho + (k-1)\rho + \sum_{n \in hp(m)} \left\lceil \frac{W_{m,k}(q) + J_n + \tau_{bit}}{T_n} \right\rceil . \rho + B_m$$

avec $R_{m,k}$ temps de réponse de la $k^{iéme}$ trame dans le pire cas, ρ temps de transmission d'une trame dans le pire cas et N_m le nombre de trames dans le message m.

Les termes suivants devraient aussi apparaître dans les analyses précédentes :

- $\wp$ est le délai de propagation électrique, que l'on peut négliger pour un réseau de petite taille (cela n'est pas valable pour un réseau faisant intervenir des satellites)
- τ_{notif} représente le pire temps pris par le contrôleur de communications pour détecter l'arrivée d'une trame et lever une interruption sur le processeur hôte ; là encore, ce temps peut-être pris en compte ou non selon son ordre de grandeur

Or $B_m=\rho$ d'où en remplaçant k par N_m on obtient le temps de réponse dans le pire cas du message en entier :

$$R_m = \max_{q=0,1,2....} (W_m(q) - qT_m + J_m + \rho + \wp + \tau_{notif})$$

$$W_m(q) = \left\{ (q+1)N_m + \sum_{n \in hp(m)} \left\lceil \frac{W_{m,k}(q) + J_n + \tau_{bit}}{T_n} \right\rceil \right\} . \rho$$

III. Analyse holistique de l'ordonnancement pour un système distribué

III.1 Etude théorique

On utilise les deux analyses précédentes afin de vérifier si les contraintes d'un tel système sont respectées dans le pire cas. Cette étude a été réalisé par Tindell et Clark [Tindell-Clark 93] en utilisant comme protocole de communication le protocole TDMA

On suppose que chaque tâche envoie un message et que certaines tâches attendent l'arrivée d'un certain message pour continuer leur exécution. Ces hypothèses impliquent donc des relations de précédence entre certaines tâches et certains messages. De plus les propriétés des messages et des tâches deviennent liées :

- un message émis par une tâche hérite de sa période (éventuellement d'un multiple de la période de la tâche s'il est émis plus d'une fois à chaque exécution).
- un message n'est placé dans la file d'attente de transmission que lorsque la tâche l'envoie, d'où l'apparition d'un nouveau délai.
- une tâche qui attend un message est donc tributaire du temps de réponse de ce message.

Ainsi, en plus de la période de la tâche un message hérite aussi d'un release jitter. Celui-ci peut être vu comme la différence entre la plus récente et la plus tardive activation de la tâche. Si on considère la plus récente, un message m peut potentiellement être mis dans la file d'attente dès que la tâche émettrice arrive. Au plus tard, le message peut être mis dans la file d'attente juste avant que la tâche émettrice ne s'achève. Cette différence est bornée par le temps de réponse R dans le pire cas de la tâche émettrice. Alors $J_m=R_e$ avec e indice de la tâche émettrice du message m.

De même la tâche destinatrice hérite de la période du message et aussi d'un release jitter :

$$J_d = R_e + A_m + R_{deliver} + T_{tick}$$

avec :

- d indice de la tâche destinatrice
- m indice du message
- e indice de la tâche émettrice
- deliver est le gestionnaire de trames du processeur destinataire du message m
- T_{tick} correspond au jitter du à l'ordonnanceur
- A_m étant la date d'arrivée du message (si on raisonne avec l'analyse à échéance arbitraire, le pire temps de réponse dans le cas contraire)

La propriété remarquable qui découle des équations d'ordonnancement est qu'elles sont mutuellement dépendantes : le release jitter de la tâche réceptrice dépend de l'instant d'arrivée du message, qui lui-même dépend de possibles interférences de messages de plus forte priorité, qui à leur tour dépendent du release jitter de leurs tâches émettrices. On comprend mieux pourquoi les équations d'ordonnancement holistique ne peuvent pas être résolues trivialement.

Une solution à ce problème peut-être calculée en réalisant que toutes les équations sont monotones dans une fenêtre temporelle. Plus simplement cela signifie qu'augmenter la valeur d'une des variables ne peut amener à la décroissance d'une autre.

Alors il est possible comme dans les analyses précédentes d'opérer par récurrence jusqu'à obtenir une solution. Les valeurs de release jitter sont initialement nulles, puis à l'étape n+1 on utilise les valeurs de release jitter obtenues à l'étape n.

Les restrictions principales de ce genre d'analyse sont donc :

- une tâche ne peut recevoir qu'un seul message en raison de la difficulté dans l'évaluation de savoir quel message déclenche la tâche.
- les tâches n'ont accès qu'à des ressources critiques locales (pour les ressources critiques distantes, il serait nécessaire d'utiliser des tâches annexes sur le modèle client-serveur)

Audsley définit l' instant critique comme l' instant qui correspond à la libération (release time) simultanée de toutes les tâches du système. Alors, garantir le respect des échéances pour un instant critique permet de les garantir pour la vie entière du système

III.2. Programmation de l' étude théorique

A l' aide d' un programme on peut donc calculer récursivement le pire temps de réponse de chaque élément du système.

Le calcul repose donc sur le calcul sur chaque site du pire temps de réponse de chaque tâche avec comme conditions initiales un jitter nul. Puis à chaque étape, on met à jour les jitters des messages et des tâches et on recalcule le pire temps de réponse jusqu' à ce que le système global converge. Afin d' assurer de pouvoir assurer la convergence de l' algorithme, on suppose d' une part la non existence de cycles (i.e. qu' une tâche ne peut être démarrée par un message qui dépend directement ou non du message qu' elle émet en fin d' exécution), d' autre part qu' une tâche ne peut être activée que par un message, et n' en émettre qu' un seul.

J' ai donc réalisé un programme en c (~800 lignes de code) qui à partir d' une configuration donnée, calcul le pire temps de réponse pour chaque tâche et chaque message tant que leur temps de réponse est inférieur à leur échéance, et sinon prévient l' utilisateur en indiquant la tâche ou le message qui ne respecte pas son échéance. Ce programme est basé sur les équations valables dans le cas de l' échéance inférieure à la période ce qui est le cas dans le domaine automobile, sans erreurs de transmission.

J' ai supposé que toutes les tâches sont périodiques, que chaque tâche peut émettre un message en fin d' exécution, que certaines tâches peuvent être activées par un message et que chaque message ne nécessite qu' une seule trame. Ainsi un message hérite de la période de sa tâche émettrice et la propage à la tâche qu' il active éventuellement. De plus puisque tout le système est activé au même instant (i.e. toutes les tâches et les messages sont dans l' état prêt à cet instant), on se trouve dans la pire configuration théoriquement possible (instant critique de Audsley) et donc on obtient directement les pires temps de réponse. [Grolleau 98]

III.3. Résultat du calcul pour un exemple de système

On raisonne donc sur un système automobile dénommé "réseau inter-système" chez PSA. Ce réseau comporte six calculateurs reliés par un réseau CAN.

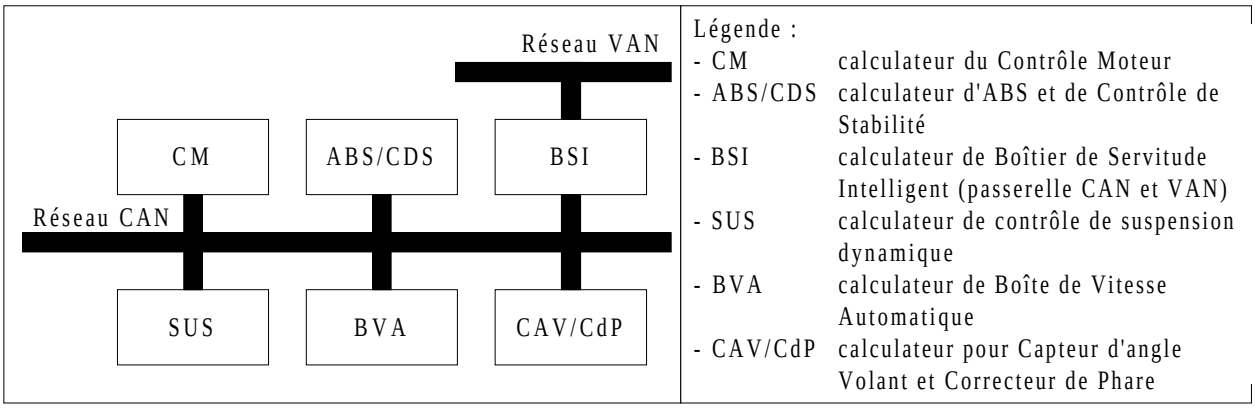


Figure 5 - Architecture matérielle

Les temps de réponse calculés sont des temps de réponse de bout en bout. Par exemple pour un message ce temps correspond au pire temps de réponse de la tâche émettrice plus le pire temps de transmission du message. De même pour une tâche démarrant sur un message ce temps correspond au temps de réponse du message (incluant déjà celui de la tâche émettrice) plus le pire temps de réponse de la tâche destinatrice.

Les messages qui transitent sur le réseau CAN sont donnés dans le tableau suivant :

Trame	Emetteur	d (octet)	Période Ti (ms)	Temps de réponse (ms)
M1	CM	8	10	3.01
M2	CAV/CdP	3	14	5.34
M3	CM	3	20	9.67
M4	BVA	2	15	5.96
M5	ABS/CDS	5	20	7.37
M6	ABS/CDS	5	40	9.78
M7	ABS/CDS	4	15	4.14
M8	BSI	5	50	13.55
M9	SUS	4	20	9.92
M10	CM	7	100	32.33
M11	BVA	5	50	12.47
M12	ABS/CDS	1	100	13.54

Les messages sont classés dans l' ordre décroissant de priorité, i.e. Mi plus prioritaire que Mi+1.

Sur les différents nœuds sont implantées des tâches dont la répartition et leurs caractéristiques sont données dans le tableau suivant :

Nœud	Nom de tâche	Message entrant	Durée (ms)	Période (ms)	Message généré	Charge	Temps de réponse(ms)
CM	T_CM1		2	10	M1	0.20	2
	T_CM2		2	20	M3	0.10	8
	T_CM3		2	100	M10	0.02	28
	T_CM4	M4	2	15		0.13	11.96
	T_CM5	M2	2	15		0.14	9.34
	T_CM6	M8	2	50		0.04	31.55
	T_CM7	M6	2	40		0.05	25.78
	Charge totale sur le nœud					0.69	
BVA	T_BVA1		2	15	M4	0.13	4
	T_BVA2		2	50	M11	0.04	8
	T_BVA3	M8	2	50		0.04	19.55
	T_BVA4	M2	2	14		0.14	7.34
	Charge totale sur le nœud					0.36	
ABS	T_ABS/CDS1		1	20	M5	0.05	5
	T_ABS/CDS2		2	40	M6	0.05	7
	T_ABS/CDS3		1	15	M7	0.07	1
	T_ABS/CDS4		2	100	M12	0.02	9
	T_ABS/CDS5	M3	1	20		0.05	11.67
	T_ABS/CDS6	M9	2	20		0.10	13.92
	Charge totale sur le nœud					0.34	
CAV	T_CAV/CdP1		4	14	M2	0.29	4
	T_CAV/CdP2	M9	4	20		0.20	17.92
	Charge totale sur le nœud					0.49	
SUS	T_SUS1		1	20	M9	0.05	6
	T_SUS2	M5	1	20		0.05	15.37
	T_SUS3	M1	1	10		0.10	4.01
	T_SUS4	M2	2	14		0.14	8.34
	T_SUS5	M7	2	15		0.13	9.15
	Charge totale sur le nœud					0.48	
BSI	T_BSI1		2	50	M8	0.04	10
	T_BSI2	M11	2	50		0.04	26.47
	T_BSI3	M1	2	10		0.20	5.01
	T_BSI4	M10	2	100		0.02	52.33
	T_BSI5	M6	2	40		0.05	15.78
	T_BSI6	M9	2	20		0.10	13.92
	T_BSI7	M12	2	100		0.02	29.47
	Charge totale sur le nœud					0.47	

Les tâches grisées correspondent aux tâches qui sont activées par des messages.

On s' aperçoit que le système est bien ordonnançable parceque toutes les échéances sont respectées pour chaque tâche et chaque message dans leur pire cas.

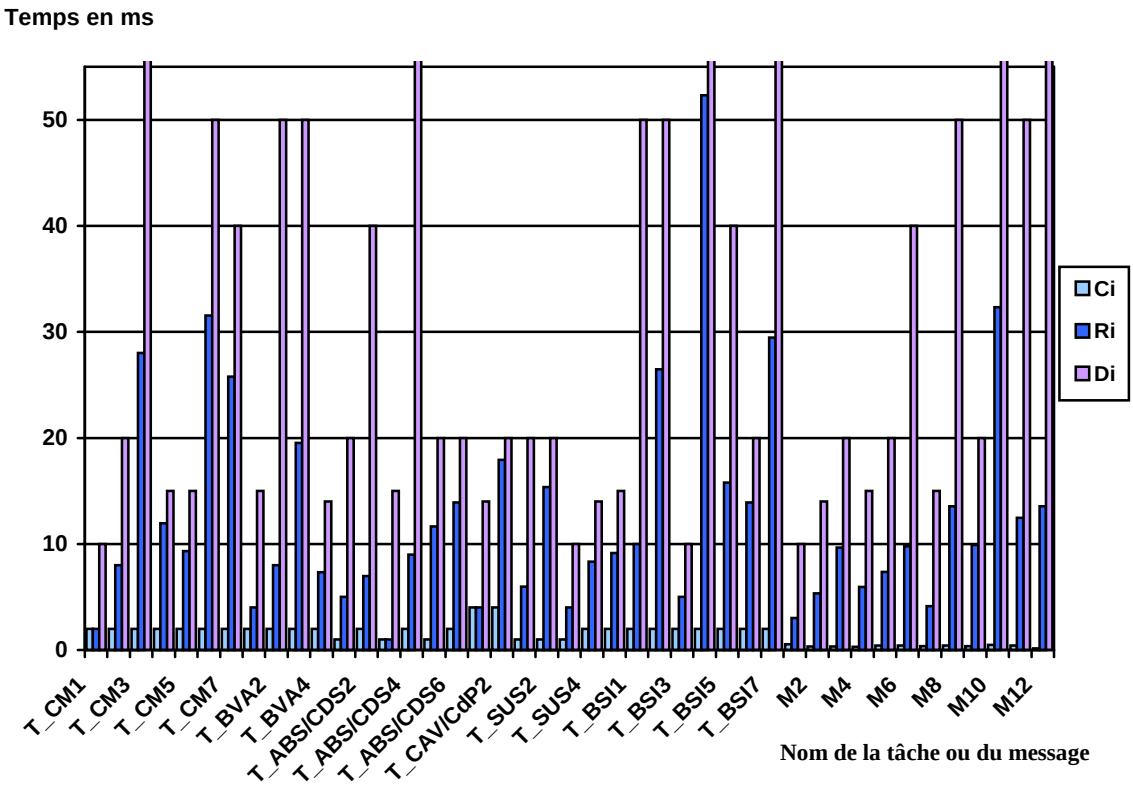


figure 6 - Comparaison des temps de calcul (transmission), échéances et temps de réponse de la configuration précédente

Les messages qui transitent sur le réseau CAN sont donnés dans le tableau suivant :

Trame	Emetteur	d (octet)	Période Ti (ms)	Temps de réponse (ms)
M1	CM	8	10	3.01
M2	CAV/CdP	3	14	5.34
M3	CM	3	20	9.67
M4	BVA	2	15	5.96
M5	ABS/CDS	5	20	12.37
M6	ABS/CDS	5	40	16.78
M7	ABS/CDS	4	15	5.14
M8	BSI	5	50	15.55
M9	SUS	4	20	17.92
M10	CM	7	100	36.33
M11	BVA	5	50	12.47
M12	ABS/CDS	1	100	22.47

Dans l' exemple suivant, on a modifié seulement quelque temps de calcul (ceux valant 1 valent maintenant 2). Le tableau suivant retrace les données et les résultats obtenus :

Nœud	Nom de tâche	Message entrant	Durée (ms)	Période (ms)	Message généré	Charge	Temps de réponse(ms)
CM	T_CM1		2	10	M1	0.20	2
	T_CM2		2	20	M3	0.10	8
	T_CM3		2	100	M10	0.02	32
	T_CM4	M4	2	15		0.13	11.96
	T_CM5	M2	2	15		0.14	9.34
	T_CM6	M8	2	50		0.04	33.55
	T_CM7	M6	2	40		0.05	32.77
	Charge totale sur le nœud					0.69	
BVA	T_BVA1		2	15	M4	0.13	4
	T_BVA2		2	50	M11	0.04	8
	T_BVA3	M8	2	50		0.04	21.55
	T_BVA4	M2	2	14		0.14	7.34
	Charge totale sur le nœud					0.36	
ABS	T_ABS/CDS1		2	20	M5	0.10	10
	T_ABS/CDS2		2	40	M6	0.05	14
	T_ABS/CDS3		2	15	M7	0.13	2
	T_ABS/CDS4		2	100	M12	0.02	18
	T_ABS/CDS5	M3	2	20		0.10	13.67
	T_ABS/CDS6	M9	2	20		0.10	23.92
	Charge totale sur le nœud					0.50	
CAV	T_CAV/CdP1		4	14	M2	0.29	4
	T_CAV/CdP2	M9	4	20		0.20	25.92
	Charge totale sur le nœud					0.49	
SUS	T_SUS1		2	20	M9	0.10	6
	T_SUS2	M5	2	20		0.10	28.37
	T_SUS3	M1	2	10		0.20	5.01
	T_SUS4	M2	2	14		0.14	9.34
	T_SUS5	M7	2	15		0.13	11.14
	Charge totale sur le nœud					0.68	
BSI	T_BSI1		2	50	M8	0.04	12
	T_BSI2	M11	2	50		0.04	26.47
	T_BSI3	M1	2	10		0.20	5.01
	T_BSI4	M10	2	100		0.02	56.33
	T_BSI5	M6	2	40		0.05	26.78
	T_BSI6	M9	2	20		0.10	21.92
	T_BSI7	M12	2	100		0.02	38.47
	Charge totale sur le nœud					0.47	

Les tâches grisées correspondent toujours aux tâches qui sont activées par des messages.

On s' aperçoit que le système ne respecte pas ses échéances pour quatre tâches dont les temps de réponse sont grisés. On peut noter cependant que tous les messages respectent leur échéance.

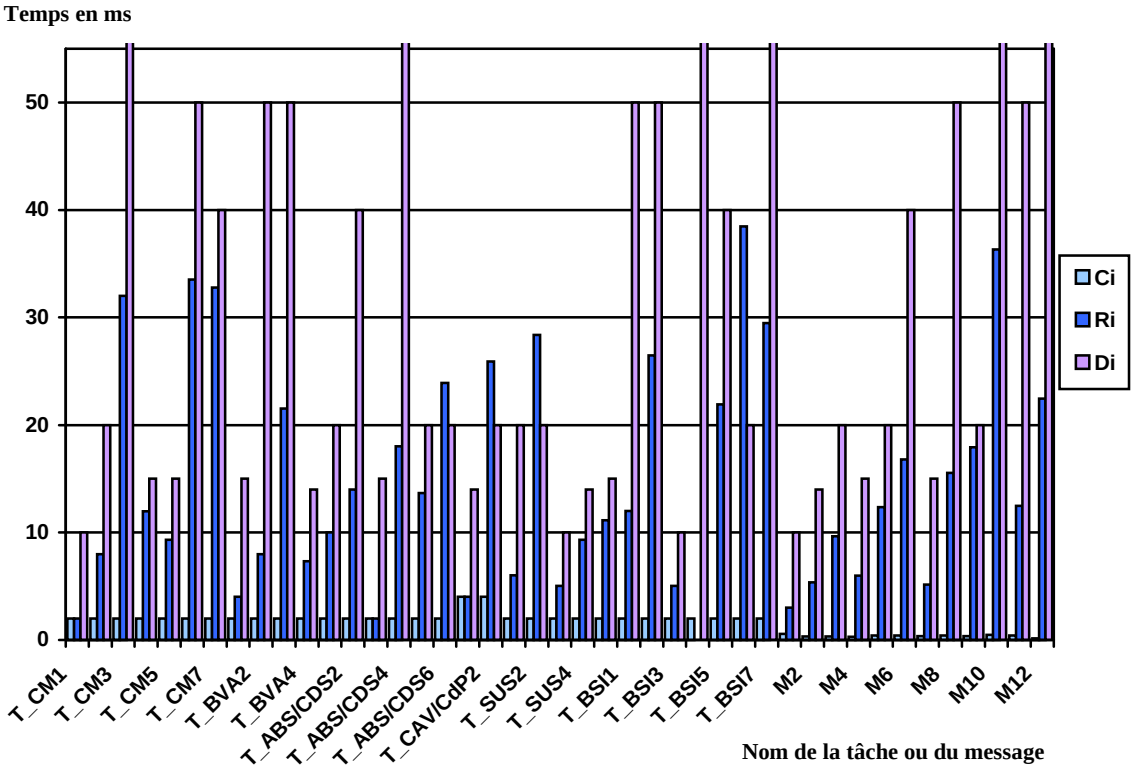


figure 7 - Comparaison des temps de calcul (transmission), échéances et temps de réponse de la configuration précédente

Cependant on s' aperçoit alors que rien ne garantit que la configuration la pire que l' on utilise pour calculer les temps de réponse peut physiquement intervenir dans la vie du système. On calcule donc peut-être une borne sur le pire temps de réponse et non le pire temps de réponse.

On peut noter que ces calculs fournissent soit le pire temps de réponse, soit une borne supérieure sur ce pire temps de réponse. Dans le deuxième cas, si un message ou une tâche ne respecte pas son échéance on ne peut conclure sur l' ordonnançabilité. La différence entre ces deux cas réside dans le fait que le calcul théorique se base sur des faits théoriquement possibles mais qui physiquement sont incompatibles. En fait ce problème de borne intervient aussitôt que deux tâches sont théoriquement en concurrence sur un site mais qui ne peuvent l' être physiquement pour des raisons de précédence (messages réseaux, messages internes...).

Avec Bruno Gaujal (Chercheur INRIA), on a essayé de vérifier, si on pouvait assurer physiquement de telles concurrence avec des décalages physiques entre les activations des tâches à la mise en route du système. Avec comme hypothèse des temps constants de calcul et de communication, on s' est aperçu qu' il existe toujours une configuration possible permettant d' éviter cette simultanéité. Donc il n' existe pas d' algorithme général et seul une étude directe sur l' implantation physique peut permettre de vérifier si on est dans le pire cas ou non.

Détermination des priorités de tâches et de messages à l' aide d' algorithmes génétiques

On a vu précédemment que lorsque l' on dispose des priorités des tâches, on peut a priori calculer le pire temps de réponse de chaque tâche du système. Cependant, d' une part le placement des tâches et d' autre part l' affectation de leurs priorités, ne peuvent pas être réalisés de manière aléatoire si l' on désire que le système respecte ses contraintes. Ce problème de placement et d' ordonnancement est réputé comme étant NP-complet. Pour ce faire j' ai décidé d' utiliser une méthode basée sur les algorithmes génétique [Caux 94, Whitley 94, Cartwright 95, Portmann 96].

I. Les algorithmes génétiques

I.1. Introduction

Les algorithmes génétiques ont été développés afin de pouvoir offrir de manière systématique une solution à des problèmes très variés. En effet les méthodes classiques dépendent excessivement des conditions initiales : les extremis qu' elles atteignent sont optimaux dans un voisinage du point de départ. Une fois un sommet atteint la seule possibilité d' amélioration est une réinitialisation aléatoire. Il est bien entendu toujours possible d' utiliser des méthodes dites "énumératives" qui permettent dans un espace de recherche fini ou un espace de recherche infini mais discrétisé de parcourir tous les points un par un. Mais ces méthodes sont peu efficaces car bien souvent les espaces de recherche sont beaucoup trop grands pour que l' on puisse explorer toutes les configurations possibles.

Les algorithmes génétiques ont été mis au point par John Holland [Holland 75] et ses étudiants de l' université du Michigan, et développés par David Golberg [Golberg 89]. Ils peuvent être considérés comme des algorithmes d' exploration ayant pour règles de base les lois de Darwin sur la survie et la reproduction des espèces.

Dans la nature, les individus d' une population donnée sont souvent en compétition, que ce soit pour la survie ou pour le choix d' un compagnon. Seuls les mieux adaptés d' entre eux survivent et se reproduisent. En d' autres termes, les gènes des individus performants vont se retrouver dans de plus en plus d' individus au fil des générations. La combinaison des bons caractères des différents ancêtres peut conduire à des individus très adaptés, dont l' adaptation est encore plus grande que celle de chacun des parents. C' est une description simplifiée de la façon dont les espèces évoluent de manière à être de plus en plus adaptées à leur environnement.

Les algorithmes génétiques utilisent une analogie directe de ce comportement naturel pour résoudre des problèmes d' optimisation. Cette analogie explique la spécificité du vocabulaire utilisé pour les algorithmes génétiques.

Le but de ces algorithmes est d' établir une solution, parfois seulement une solution approchée à un problème. Cette solution est crée en parcourant l' espace de recherche. Cette exploration commence à partir d' une population initiale qui évolue. Comme dans la nature, seules les individus les plus forts subsisteront, c' est à dire les meilleures solutions pour notre problème. Les mécanismes génétiques et de sélection naturelle reposent sur trois idées essentielles :

- l' évolution est le résultat d' une altération progressive et continue des êtres vivants au cours des générations.
- la reproduction implique une mémoire : l' hérédité, sous la forme de gènes, qui peuvent prendre différentes valeurs que l' on appelle allèles ; ce matériel héréditaire, appelé génotype subit, au niveau moléculaire, des modifications constantes par mutations et recombinaisons, aboutissant à une grande diversité.
- le mécanisme central est la sélection naturelle, qui opère au niveau des populations en sélectionnant les individus les mieux adaptés. Cette sélection s' effectue sur le phénotype (valeur réelle d' une variable), c' est-à-dire la structure qui émerge du développement du génotype.

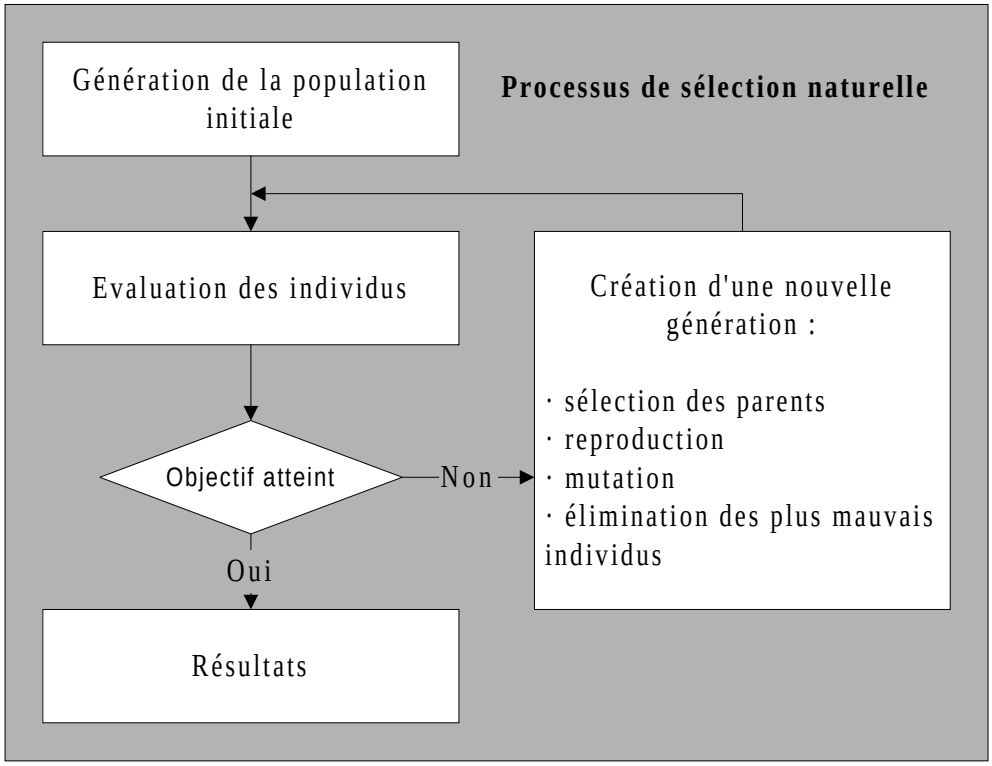


Figure 8 - Schématisation d' un algorithme génétique

Les algorithmes génétiques peuvent donc être considérés comme des algorithmes pseudo-aléatoires d' exploration, c' est à dire qu' ils utilisent des tirages aléatoires mais sont aussi guidés par des informations obtenues durant la phase de recherche. Ils se distinguent des algorithmes classiques par les quatre caractéristiques suivantes, indépendamment du problème étudié :

- la manipulation directe d' un codage des paramètres, et non des paramètres eux-mêmes,
- l' exploration au moyen d' une population de points et non d' un point unique,
- l' exploration aveugle à partir uniquement des données du problème,
- l' utilisation pour l' optimisation d' opérateurs stochastiques (faisant appel au hasard) et non de règles déterministes.

Cette approche stochastique a pour principale conséquence d' éviter la convergence vers un optimum local.

I.2. Les mécanismes de base

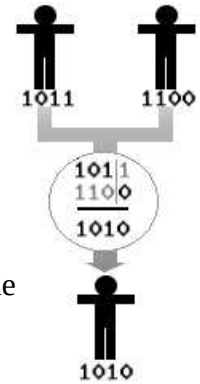
I.2.a. La sélection

La sélection permet de choisir quel(s) individu(s) vont se reproduire en fonction des valeurs de la fonction d' adaptation (voir ci-dessous). En effet si l' on se contentait de tirer des individus au hasard avec équiprobabilité, il n' y aurait pas cette notion d' évolution vers la performance globale de la population.

Une des idées directrices des algorithmes génétiques est donc d' affecter une probabilité de sélection supérieure aux meilleurs individus et, inversement, d' éliminer au fur et à mesure les plus mauvais (c' est peut être cruel mais c' est aussi la loi de la nature !)

Il existe diverses manières de réaliser cette sélection. La plus simple pour choisir un individu est de faire un tirage aléatoire avec une roue de loterie où chaque individu se voit attribuer une place plus ou moins grande selon son adaptation.

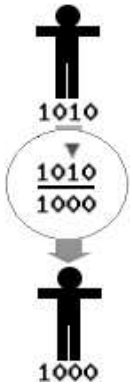
I.2.b. Le croisement



Après la sélection, le croisement a lieu. Dans ce cas, deux individus de la nouvelle génération échangent aléatoirement plusieurs parties de leur génotype pour former un individu différent de ceux d' origine.

Cet opérateur est essentiel, car il qui permet d' obtenir de nouveaux individus, distincts de ceux déjà existants, et donc d' explorer tout l' espace de recherche. L.Djerid a analysé différents opérateurs de croisement [Djerid 95]

I.2.c. La mutation



La mutation est la modification de la valeur d' un élément du génotype d' un individu. Sa mise en œuvre est également décidée de façon aléatoire sur le résultat d' un croisement. Elle constitue une exploration totalement au hasard de l' espace de recherche et permet également d' éviter la perte du matériel génétique potentiellement utile. Mais, par rapport au croisement, son rôle reste secondaire, et on lui attribue une probabilité plus faible.

I.2.d. La fonction d' adaptation

Cette fonction correspond à une mesure de profit, d' utilité ou de qualité que l' on souhaite maximiser, ou à un coût que l' on souhaite minimiser. L' opérateur de sélection, qui est une version artificielle de la sélection naturelle définie par Darwin, consiste à donner aux individus ayant une adaptation plus grande de plus grandes chances d' engendrer un ou plusieurs descendants.

I.3. Optimisation de l' algorithme

I.3.a. La représentation réelle des chromosomes

Les algorithmes génétiques sont à l' origine appliqués à des données codées en suites de 0 ou de 1, ce qui les rend indépendants du problème traité. Cependant les nombres binaires sont pour nous moins évocateurs que les nombres réels. En outre, des difficultés surviennent pour exprimer la fonction d' évaluation, et traiter les problèmes à plusieurs variables :

- Les fonctions d' évaluation conservent souvent leur forme réelle, c' est-à-dire s' applique sur les données brutes et non celles converties en suites de 0 et de 1. Les chromosomes sont alors convertis à chaque évaluation.
- Les problèmes multi-variables sont ramenés à un problème mono-variable par concaténation des inconnues en un seul chromosome. A chaque évaluation, la chaîne de bits résultante est alors découpée en autant de sous-chaînes qu' il y a d' inconnues; ces sous-chaînes sont ensuite converties en nombres réels pour être appliquées à la fonction d' évaluation.

Ces opérations de conversion sont coûteuses en temps-machine, et sont répétées un grand nombre de fois à chaque génération. Nous discernons ici les limites de la représentation en suites de 0 ou de 1.

La représentation réelle est une représentation où les données ne sont plus codées en suites de 0 ou de 1, mais les informations y sont structurées afin d' en avoir une lisibilité plus confortable. Elle propose un compromis intéressant : elle élimine toutes les opérations de conversion, mais en contrepartie, elle rend l' algorithme génétique avancé plus dépendant du problème traité. Dans ce type de représentation, un chromosome est un n-uplet de nombres réels, chacune des composantes correspondant alors à une inconnue du problème. La représentation réelle permet en outre la génération d' individus qui physiquement correspondent à un individu de l' espace des solutions, on élimine ainsi les individus que l' algorithme binaire pourrait créer mais qui réellement ne correspondrait à aucune configuration réelle.

I.3.b. La gestion de la population

La création de la population initiale se fait habituellement de manière aléatoire. Ce hasard permet d' assurer l' exploration la plus vaste possible de l' espace des solutions. En effet, le fait de créer des individus sans réflexion approfondie permet de ne pas focaliser la recherche uniquement sur un type de solutions. Cependant l' ajout dans la population initiale d' individus créés selon certaines heuristiques peut permettre à l' algorithme de converger plus rapidement. En effet, fournir à l' algorithme des individus aux caractéristiques proches de la solution permet à l' algorithme de posséder dans sa population des types d' individus qui ne seraient apparus sans cela qu' au bout de plusieurs générations. A partir de ces individus, il peut alors les combiner avec les individus de la population initiale qu' il a créés aléatoirement. Ainsi l' algorithme explore différentes voies mais avec déjà des individus aux caractéristiques remarquables.

De plus jusqu' à présent, tous les chromosomes générés par les opérateurs génétiques étaient insérés directement dans la population de la génération suivante. C' est en effet la

stratégie de l' algorithme génétique de base. Il existe cependant d' autres alternatives qui contribuent à réduire le temps de convergence de l' algorithme génétique avancé :

- La stratégie dite de "l' élitisme" qui consiste à recopier le meilleur chromosome de la population actuelle dans la population de la génération suivante, et de la compléter par d' autres chromosomes générés de manière traditionnelle jusqu' à obtenir le nombre d' individus nécessaires. Il devient alors impossible pour les générations suivantes que leur meilleur chromosome soit inférieur à celui des générations précédentes. Les performances de l' algorithme génétique en sont grandement améliorées.
- La stratégie dite de "la population sans doubles" qui consiste à n' insérer le chromosome généré dans la population de la génération suivante que s' il est différent de tous les chromosomes présents dans celle-ci. De nouveaux chromosomes sont générés jusqu' à obtenir le nombre d' individus nécessaire dans la population suivante. Il devient alors impossible d' appliquer l' opérateur de croisement génétique à une paire de chromosomes identiques (cas défavorable n' introduisant aucune amélioration dans la population suivante). Cette optimisation décuple l' efficacité des opérateurs.
- La stratégie dite "stratégie à un pas" qui consiste à n' insérer les individus qu' un par un. A chaque fois que l' on crée un nouvel individu, on remplace dans la population l' individu le plus faible par ce nouvel individu.

Toutes ces méthodes accroissent la complexité des algorithmes du fait, soit qu' elles introduisent de nouvelles opérations, soit parce qu' elles font évoluer la population plus lentement. Cependant les gains ne sont pas négligeables puisque d' une part en conservant les meilleurs individus, on assure une continuité dans la pertinence des solutions et d' autre part en empêchant la création de doubles, l' algorithme évite la dégénérescence de sa population.

I.3.c. Les opérateurs génétiques réels

A la différence de l' algorithme génétique de base, l' algorithme avancé propose plusieurs stratégies concernant l' emploi de ces opérateurs.

Une possibilité est d' appliquer de façon systématique le croisement et la mutation génétiques, mais avec des opérateurs variant en fonction de l' avancée dans les générations (comme la mutation non uniforme). Une autre alternative consiste à ne pas appliquer ces opérateurs de façon systématique : l' algorithme en choisit un au hasard pour chaque paire d' individus. Les opérateurs entrent alors en compétition, tout comme le font les chromosomes lors de la sélection par la roulette de casino.

Le choix de l' opérateur dépend de son poids, équivalent à l' indice de qualité des individus et reflète sa probabilité d' être sélectionné. Plusieurs options s' offrent à alors :

- les opérateurs sont considérés "équiprobables" (i.e. de poids identiques).
- les opérateurs ont des poids différents mais constants au cours des générations.
- les opérateurs ont des poids différents et variant au cours des générations.

Dès lors, croisement et mutation deviennent des opérateurs distincts, tout comme leurs variantes.

Le croisement réel ne se différencie du croisement binaire que par la nature des éléments qu' il altère : ce ne sont plus des bits qui sont échangés à droite du point de croisement, mais des variables réelles.

Le croisement arithmétique est propre à la représentation réelle. Il s' applique à une paire de chromosomes et se résume à une moyenne pondérée des variables des deux parents.

La mutation réelle ne se différencie de la mutation binaire que par la nature de l' élément qu' elle altère : ce n' est plus un bit qui est inversé, mais une variable réelle qui est de nouveau tirée au hasard sur son intervalle de définition.

La mutation non uniforme possède la particularité de retirer les éléments qu' elle altère dans un intervalle de définition variable et de plus en plus petit. Plus nous avançons dans les générations, moins la mutation n' écarte les éléments de la zone de convergence. Cette mutation adaptative offre un bon équilibre entre l' exploration du domaine de recherche et un affinement des individus. Le coefficient d' atténuation de l' intervalle est un paramètre de cet opérateur.

II. Analyse des heuristiques utilisables pour que le système respecte ses contraintes

Dans un premier temps, on liste toutes les tâches avec leurs périodes, leurs échéances, leurs messages à émettre ou à recevoir... Ensuite, à partir de ces données, on cherche des heuristiques afin de placer les tâches sur les différents sites, puis des heuristiques pour affecter des priorités aux tâches et messages.

II.1. Les heuristiques utilisables pour placer les tâches sur les sites

Pour placer les différentes tâches sur les différents sites, plusieurs stratégies sont possibles. Une telle projection matérielle doit suivre certains critères :

- les critères matériels qui imposent à certaines tâches d' être disposées sur certains processeurs en raison de leur fonction applicative (par exemple le contrôle moteur doit être situé sur le site où sont câblées les entrées-sorties).
- les critères fonctionnels qui sont imposés de manière à garantir le fonctionnement de tout le système. [Audsley 95]

Ainsi on dispose en premier lieu les tâches qui ont des critères matériels à respecter. Ensuite on dispose le reste des tâches en cherchant à respecter certaines contraintes :

- la charge du processeur de chaque site
- la mémoire allouable sur chaque site
- l' ordonnançabilité des tâches sur chaque site
- l'accès aux ressources critiques sur chaque site
- la diminution du nombre de messages (en effet si deux tâches communiquantes sont situées sur un même site alors leur communication ne passe pas par le bus CAN)

La première partie de la thèse de J.P. Beauvais [Beauvais 99] traite plus spécifiquement des méthodes utilisables pour placer des tâches temps réel sur un système réparti.

Suivant que l' on privilégie un ou plusieurs critères, on crée une heuristique qui dans l' algorithme génétique va créer des configurations de base qui physiquement sont acceptables et dont la probabilité que le système ainsi créé respecte toutes ses échéances dans les pires cas.

II.2. Les heuristiques utilisables pour affecter les priorités

Une fois que les tâches ont été placées sur des sites, il s' agit d' affecter des priorités aux tâches et aux éventuels messages nécessaires. Là encore différentes stratégies sont possibles :

- l' utilisation de l' algorithme rate monotonic car celui-ci est optimal pour des tâches à échéance sur requête.
- l' utilisation de l' algorithme deadline monotonic [Audsley 90] car celui-ci est optimal pour des tâches indépendantes
- l' utilisation de l' algorithme d' Audsley [Audsley91]

De plus, si une tâche A doit s' exécuter avant une autre tâche B sur un site, il est alors nécessaire de lui donner une priorité plus forte que la priorité de la tâche B.

Il reste alors à affecter les priorités des messages. Pour ce faire, on liste en fonction de la projection réalisée les messages effectivement transmis sur le bus. Puis on peut procéder soit comme pour les tâches en utilisant un algorithme rate monotonic ou deadline monotonic, soit en utilisant d' autres critères tels que :

- de garder le même ordre que les tâches émettrices.
- rendre plus prioritaire les messages qui démarrent d' autres tâches.

II.3. L' algorithme d' Audsley adapté aux systèmes distribués

J' ai choisi d' adapter l' algorithme d' Audsley pour une configuration distribuée avec échange de messages. Je rappelle la constatation suivante : si une tâche respecte ses échéances avec une certaine priorité alors elle les respectera avec une priorité plus forte. L' algorithme d' Audsley repose sur le fait que s' il existe au moins une configuration ordonnançable, alors si dans une certaine configuration la tâche de plus faible priorité respecte son échéance alors il existe une configuration ordonnançable dans laquelle on retrouve cette tâche avec sa même priorité. Ainsi on affecte récursivement les priorités aux tâches (des plus faibles aux plus fortes)

Le raisonnement est basé sur le principe de la récurrence sur la priorité et est le même pour chaque site :

- choix d' une tâche à laquelle on affecte la plus faible priorité disponible
- vérification qu' elle respecte son échéance. Si elle ne le respecte pas, on recommence avec une autre tâche pour la même priorité, sinon on recommence pour la priorité supérieure avec une autre tâche.

On raisonne de même avec les messages. La difficulté réside dans la prise en compte de la gigue (gigue due par exemple à l' arrivée d' un message par exemple). En raison de l' existence de cette gigue, il est difficile d' affirmer qu' une tâche qui respecte son échéance à un instant donné de l' algorithme, le respectera toujours à la fin de celui-ci. De ce fait, il est nécessaire d' affecter en premier lieu des priorités aux tâches qui ne sont pas activées par un message. Ainsi sur les différents sites, ces tâches reçoivent les priorités les plus faibles (toujours à condition qu' elle respecte leur échéance). Au fur et à mesure que des tâches ont une priorité, les messages émis par ces tâches ont alors des gigues minimales connues, il est alors possible d' affecter des priorités à ces messages. Une fois qu' un message a sa propre priorité et qu' il respecte son échéance, on connaît les gigues minimales des éventuelles tâches réceptrices. On peut alors essayer d' affecter une priorité à ces tâches. Ainsi si toutes les tâches et les messages obtiennent une priorité de cette manière, on obtient une configuration qui a une forte probabilité d' être solution de notre problème. Cependant, il est possible qu' une tâche ne respecte plus à présent son échéance, ou que l' algorithme ne réussisse pas à créer une configuration "valable", alors on conserve tout de même la configuration finale afin de fournir à l' algorithme génétique une voie de recherche à explorer.

L' algorithme d'implantation est alors le suivant :

```
ensemble des messages = vide
pour chaque site faire
    priorité en cours sur ce site = plus faible priorité du site disponible
    faire tant que l' ensemble des tâches sans priorité sur ce site et non activées par un
    message et n' ayant pas encore été choisies pour la priorité courante est non vide
        choisir une tâche parmi cet ensemble
        si la tache respecte ses échéances avec priorité en cours sur ce site alors
            affectation de la priorité en cours sur ce site à la tâche
            mise à jour de la gigue sur l' éventuel message émis
            ajout de l' éventuel message dans ensemble des messages
            priorité en cours sur ce site = priorité immédiatement supérieure
        fin si
    fin faire
fin faire
faire tant que l' ensemble des messages non vide ou ensemble des tâches sans priorité non vide
    affectation fausse
    priorité message en cours = plus faible priorité pour un message disponible
    si ensemble des messages non vide alors
        faire tant affectation fausse et il reste des messages non choisis pour la priorité
        message en cours
            choisir un message parmi l' ensemble de messages
            si le message respecte son échéance avec priorité message en cours
            alors
                affectation de la priorité message en cours au message
                mise à jour de la gigue sur les éventuelles tâches réceptrices
                ajout des éventuelles tâches réceptrices dans l' ensemble des
                tâches
                priorité message en cours = priorité immédiatement supérieure
                affectation vraie
            fin si
        fin faire
    fin si
    affectation fausse
    si ensemble des tâches sans priorité non vide alors
        faire tant affectation fausse et il reste des tâches non choisies pour la priorité
        tâche en cours sur ce site
            choisir une tâche parmi l' ensemble des tâches sans priorité
            si la tâche respecte ses échéances avec priorité en cours sur ce site alors
                affectation de la priorité en cours à la tâche
                mise à jour de la gigue sur l' éventuel message émis
                ajout de l' éventuel message dans ensemble des messages
                priorité en cours sur ce site = priorité immédiatement supérieure
                affectation vraie
            fin si
        fin faire
    fin si
    si affectation fausse alors
        pas de configuration ordonnançable avec cette projection matérielle
        affectation aléatoire des priorités disponibles et sortie de l' algorithme
    fin si
fin faire
```

III. Réalisation

III.1. Codage du problème

III.1.a. Structure des données

Pour représenter l' état de configuration du système, on choisit d' utiliser une représentation matricielle. En effet avec une telle représentation les opérations de mutation, reproduction sont plus faciles à mettre en œuvre. De plus une telle représentation permet visuellement de vérifier simplement certains invariants du système :

- une tâche ne se trouve que sur un site,
- la nécessité de l' existence physique sur le bus CAN d' un message...

J' ai décider que chaque ligne correspondrait à une unique tâche et que chaque colonne à un site. De plus j' ai rajouté une colonne, la dernière, pour les messages. En fait on peut diviser ma matrice en deux sous matrices :

- la première avec comme colonnes les sites : celle ci retrace l' implantation physique des tâches sur les différents sites. Chaque terme de la matrice est soit vide soit comporte un élément composé de plusieurs champs :
  - δ_{pol} qui indique si la tâche est préemptible ou non
  - la priorité de la tâche sur ce site
- la seconde comporte en fait la priorité du message envoyé par la tâche correspondant à la ligne.

	Site 1	Site 2	Site 3		Site j		Messages
Tâche 1	+ 1	-	-		-		+ 3
Tâche 2	-	+ 1	-		-		-
Tâche 3	+ 2	-	-		-		+ 1
Tâche i	-	-	-		+ k		+ 2

Figure 9 - représentation de la matrice avec uniquement la priorité

De plus pour stocker les paramètres fixes des tâches et des messages, j' ai choisi d' utiliser deux tableaux : un pour les tâches et un pour les messages. On retrouve dans le premier tableau les paramètres suivants :

- le nom de la tâche.
- le temps de calcul de la tâche.
- la période de la tâche.
- l' échéance de la tâche.
- le temps de blocage de la tâche (calculé par le programme).
- le jitter de la tâche.
- le pire temps de réponse de la tâche.
- le nom du message émis.
- le nom du message reçu.
- δ_{prio} qui indique si la priorité de la tâche est fixe ou non (c' est le cas pour les tâches de l' exécutif par exemple) et alors cette priorité.
- δ_{loc} qui indique si la tâche est fixée ou non et alors le site.
- δ_{pol} qui indique si la tâche est préemptible ou non.
- un tableau indicé par les tâches et indiquant avec quelles tâches la tâche courante communique (un message est alors nécessaire si les tâches ne sont pas sur le même site). On indique par +1 si la tache courante envoie un message à la tache du tableau et par –1 si la tâche courante reçoit un message de la tâche du tableau.

On retrouve une structure équivalente dans le tableau des messages (celui-ci est indicé par le numéro de la tâche émettrice) :

- le nom du message.
- le temps de transmission du message.
- la période du message.
- l' échéance du message.
- le temps de blocage du message (calculé par le programme).
- le jitter du message.
- le pire temps de réponse du message.
- le nom de la tâche émettrice.

III.1.b. Les mécanismes de base

La sélection ne s' effectue pas selon le système de la roulette, mais selon un système un peu différent. Les individus sont rangés dans des classes d' adaptation, et les cinq meilleures classes obtiennent une probabilité d' avoir un de leurs individus sélectionné. Ainsi comme à la roulette on tire un nombre, on regarde s' il appartient à l' intervalle attribué à la meilleure classe et le cas échéant, on choisit aléatoirement un individu de cette classe, sinon on regarde s' il appartient à l' intervalle de la classe suivante et ainsi de suite. On peut noter que si une classe est vide, on choisit un individu de la classe suivante. Et pour le cas ou la cinquième classe est vide, l' individu est alors choisi dans les individus encore plus faibles.

Le croisement s' effectue en choisissant de manière aléatoire pour le fils et pour chaque tâche soit la configuration du père soit celle de la mère. Ensuite on effectue une remise en forme du fils.

Celle-ci réorganise sur chaque site les priorités en conservant l' ordre mais en les rendant contiguës. De plus, puisque sur chaque site des parents chaque priorité n' intervient qu' une fois, dans le pire cas le fils peut alors avoir sur un site deux tâches ayant la même priorité, alors on choisit aléatoirement la tâche la plus prioritaire.

La remise en forme du fils désactive d' une part les messages qui ne sont plus nécessaires lorsque la tâche émettrice et les taches destinataires sont sur le même site et d' autre part réactive ceux de nouveau utiles. Comme pour les tâches sur les sites, un compactage des priorités des messages avec choix aléatoire du plus prioritaire est effectué dans le cas ou deux messages ont la même priorité.

La mutation consiste à changer la valeur de certains paramètres susceptibles d' évoluer avec une certaine probabilité. Ces probabilités sont fixées à priori :

- $P_{priotache}$ qui indique la probabilité de changer la priorité pour une tâche
- $P_{priomessage}$ qui indique la probabilité de changer la priorité pour un message
- P_{loc} qui indique la probabilité de changer la localisation d'une tâche

On tire comme à la roulette un nombre et l' on regarde suivant l' intervalle dans lequel il se trouve la mutation que l' on va éventuellement effectuer. En effet si l' on désire muter un paramètre qui est fixe la mutation n' a pas lieu.

La mutation de priorité consiste à permuter les priorités de deux tâches ou de deux messages. La mutation de localisation consiste à déplacer la tâche sur un nouveau site. Il s' agit alors de répercuter cette mutation : il s' agit de vérifier si de nouveaux messages doivent être créer et si des anciens n' ont plus de raisons d' exister. Comme précédemment, un compactage des priorités est effectué.

La fonction d' adaptation calcule en fait le nombre d' échéance non respectée par la configuration testée pour chaque temps de réponse calculé dans son pire cas. Ainsi des individus différents peuvent appartenir à la même classe.

III.1.c.Les heuristiques

On suppose que sur chaque site on dispose de suffisamment de priorités afin que chaque tâche ait sa propre priorité. Cette hypothèse provient du fait que d' une part dans Osek chaque tâche à des priorités différentes et d' autre part dans l' étude des temps de réponse réalisée précédemment cette hypothèse est aussi faite.

En plus dans la génération aléatoire de configurations, j' ai utilisé des heuristiques afin de permettre une convergence plus rapide de l' algorithme génétique.

J' ai utilisé comme heuristiques pour placer les tâches sur les sites, une heuristique qui choisi aléatoirement une tâche et la place sur le site où la charge du processeur est la plus faible à condition que la charge résultante reste inférieure à 1.

Pour l' affectation des priorités des tâches et des messages, je me suis servi d' heuristiques reposant chacune soit sur l' algorithme rate monotonic, soit sur l' algorithme deadline monotonic. Cependant puisque sur chaque site les tâches doivent posséder une priorité propre, il faut alors parfois utiliser un autre critère pour classer les tâches par priorité. Puisque dans nos exemples, je ne disposais pas d' autres données, j' ai choisi de classer les tâches placées sur un même site et qui avaient la même période ou la même échéance aléatoirement. J' ai utilisé le même critère pour les messages

III.2. Résultats

Le programme ainsi réalisé (~2500 lignes de code) m' a permis de le tester. Pour ce faire, je me suis servi dans un premier temps de la même configuration que celle utilisée pour le calcul des temps de réponse. L' algorithme génétique m' a permis de trouver d' autres configurations solutions de mon problème.

Dans le tableau suivant (figure 11), trois configurations sont présentées correspondant au même énoncé que celui de l' exemple dans le calcul des pires temps de réponse (problème 1). Une des trois configurations est la configuration testée précédemment et les deux autres sont deux nouvelles configurations répondant à l' énoncé du problème et générées par l' algorithme génétique (ce dernier en a trouvé beaucoup d' autres, c' est à dire au moins 4000 pour une population de 5000 individus, ce nombre pouvant encore augmenter)

Puis à partir de cet énoncé de problème, j' ai décidé d' augmenter les temps de traitement de toutes les fonctions et de ne plus les affecter sur des sites prédéterminés, mais de laisser l' algorithme disposé les tâches sur les différents nœuds (problème 2). Si l' on conserve la disposition initiale des tâches, le système n' est pas ordonnançable : la charge processeur pour le nœud CM est de 1,11 (0,61 pour BVA, 0,94 pour ABS, 0,41 pour CdP, 1,01 pour SUS, 0,90 pour BSI). En fait, ceci revient à disposer une charge processeur de 5,00 sur six sites. J' ai conservé les mêmes noms pour que le lecteur puisse effectivement voir les éventuelles nouvelles dispositions (figure 12).

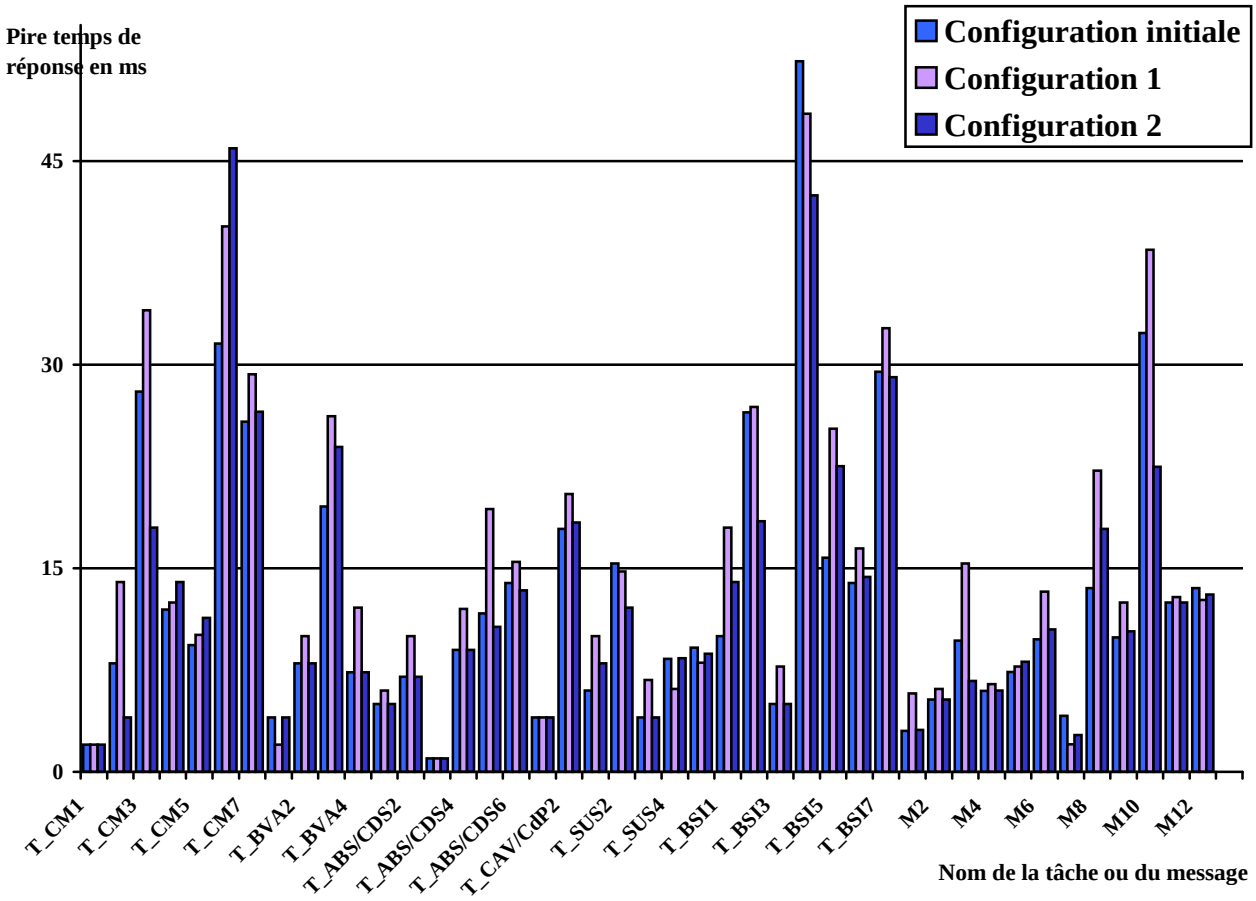


Figure 10 - Comparaison des temps de réponse des configurations du tableau suivant

Nœud	Nom de tâche	Configuration initiale		Autre configuration 1		Autre configuration 2	
		Priorité	Temps de réponse (ms)	Priorité	Temps de réponse (ms)	Priorité	Temps de réponse (ms)
CM	T_CM1	1	2	1	2	1	2
	T_CM2	4	8	4	14	2	4
	T_CM3	7	28	7	34	6	18
	T_CM4	3	11.96	3	12.47	4	13.99
	T_CM5	2	9.34	2	10.11	3	11.34
	T_CM6	6	31.55	6	40.18	7	45.92
	T_CM7	5	25.78	5	29.29	5	26.51
BVA	T_BVA1	2	4	1	2	2	4
	T_BVA2	4	8	4	10	4	8
	T_BVA3	3	19.55	2	26.18	3	23.92
	T_BVA4	1	7.34	3	12.11	1	7.34
ABS	T_ABS/CDS1	4	5	4	6	4	5
	T_ABS/CDS2	5	7	5	10	5	7
	T_ABS/CDS3	1	1	1	1	1	1
	T_ABS/CDS4	6	9	6	12	6	9
	T_ABS/CDS5	2	11.67	3	19.37	3	10.69
	T_ABS/CDS6	3	13.92	2	15.47	2	13.37
CAV	T_CAV/CdP1	1	4	1	4	1	4
	T_CAV/CdP2	2	17.92	2	20.47	2	18.37
SUS	T_SUS1	4	6	5	10	5	8
	T_SUS2	5	15.37	4	14.77	3	12.11
	T_SUS3	1	4.01	1	6.77	1	4.01
	T_SUS4	2	8.34	2	6.11	2	8.37
	T_SUS5	3	9.15	3	8.04	4	8.71
BSI	T_BSI1	4	10	6	18	5	14
	T_BSI2	5	26.47	5	26.88	3	18.47
	T_BSI3	1	5.01	1	7.77	1	5.01
	T_BSI4	7	52.33	3	48.47	7	42.47
	T_BSI5	3	15.78	4	25.28	4	22.51
	T_BSI6	2	13.92	2	16.47	2	14.37
	T_BSI7	6	29.47	7	32.67	6	29.06
Messages	M1	1	3.01	9	5.77	1	3.01
	M2	2	5.34	5	6.10	2	5.34
	M3	3	9.67	3	15.37	6	6.69
	M4	4	5.96	12	6.47	4	5.99
	M5	5	7.37	4	7.77	7	8.11
	M6	6	9.78	8	13.29	8	10.51
	M7	7	4.14	2	2.04	3	2.71
	M8	8	13.55	10	22.18	9	17.92
	M9	9	9.92	6	12.47	5	10.37
	M10	10	32.33	11	38.47	12	22.47
	M11	11	12.47	7	12.88	11	12.47
	M12	12	13.54	1	12.67	10	13.06

Figure 11 - Présentation de différentes affectations de priorité pour un même problème (n°1)

Nom de tâche	Temps de calcul	Configuration 1			Configuration 2		
		Nœud	Priorité	Temps de réponse (ms)	Nœud	Priorité	Temps de réponse (ms)
T_CM1	2	2	1	2	2	1	2
T_CM2	4	4	3	12	4	3	12
T_CM3	10	4	5	54	5	6	62
T_CM4	2	1	4	10	1	4	10
T_CM5	2	3	2	9.71	3	2	9.26
T_CM6	5	3	6	43	3	6	43
T_CM7	4	5	4	14	5	3	10
T_BVA1	2	1	3	6	1	3	6
T_BVA2	5	1	6	35	1	5	23
T_BVA3	5	3	5	34	3	5	34
T_BVA4	2	3	1	7.71	3	1	7.26
T_ABS/CDS1	4	2	3	8	2	3	8
T_ABS/CDS2	4	5	3	10	5	4	14
T_ABS/CDS3	2	5	1	2	5	1	2
T_ABS/CDS4	4	1	5	22	3	7	47
T_ABS/CDS5	4	4	2	8	4	2	8
T_ABS/CDS6	4	5	2	11.38	5	2	11.29
T_CAV/CdP1	2	2	2	4	2	2	4
T_CAV/CdP2	4	0	3	17.37	0	3	17.29
T_SUS1	4	4	1	4	4	1	4
T_SUS2	4	3	3	18.48	3	3	18.04
T_SUS3	2	1	1	5.01	1	1	4.93
T_SUS4	2	0	1	7.71	0	2	9.26
T_SUS5	2	1	2	8.07	1	2	7.63
T_BSI1	5	3	4	21	3	4	21
T_BSI2	5	1	7	44	1	6	36
T_BSI3	2	0	2	7.01	0	1	4.93
T_BSI4	10	5	5	89.03	5	5	32
T_BSI5	4	4	4	29.03	4	4	32.18
T_BSI6	4	0	4	27.38	0	4	27.29
T_BSI7	10	2	4	56.63	2	4	81.18
M1	0.5224		1	3.01		1	2.93
M2	0.3304		3	5.71		2	5.26
M3	0.3304	inutile			inutile		
M4	0.292	inutile			inutile		
M5	0.4072		5	10.48		4	10.04
M6	0.4072		7	13.03		5	16.18
M7	0.3688		4	4.07		3	3.63
M8	0.4072	inutile			inutile		
M9	0.3688		2	5.38	inutile		
M10	0.484		8	57.03	inutile		
M11	0.4072	inutile			inutile		
M12	0.146		6	24.62		6	49.18

Figure 12 - Présentation de différentes affectations de priorité pour un même problème (n°2)

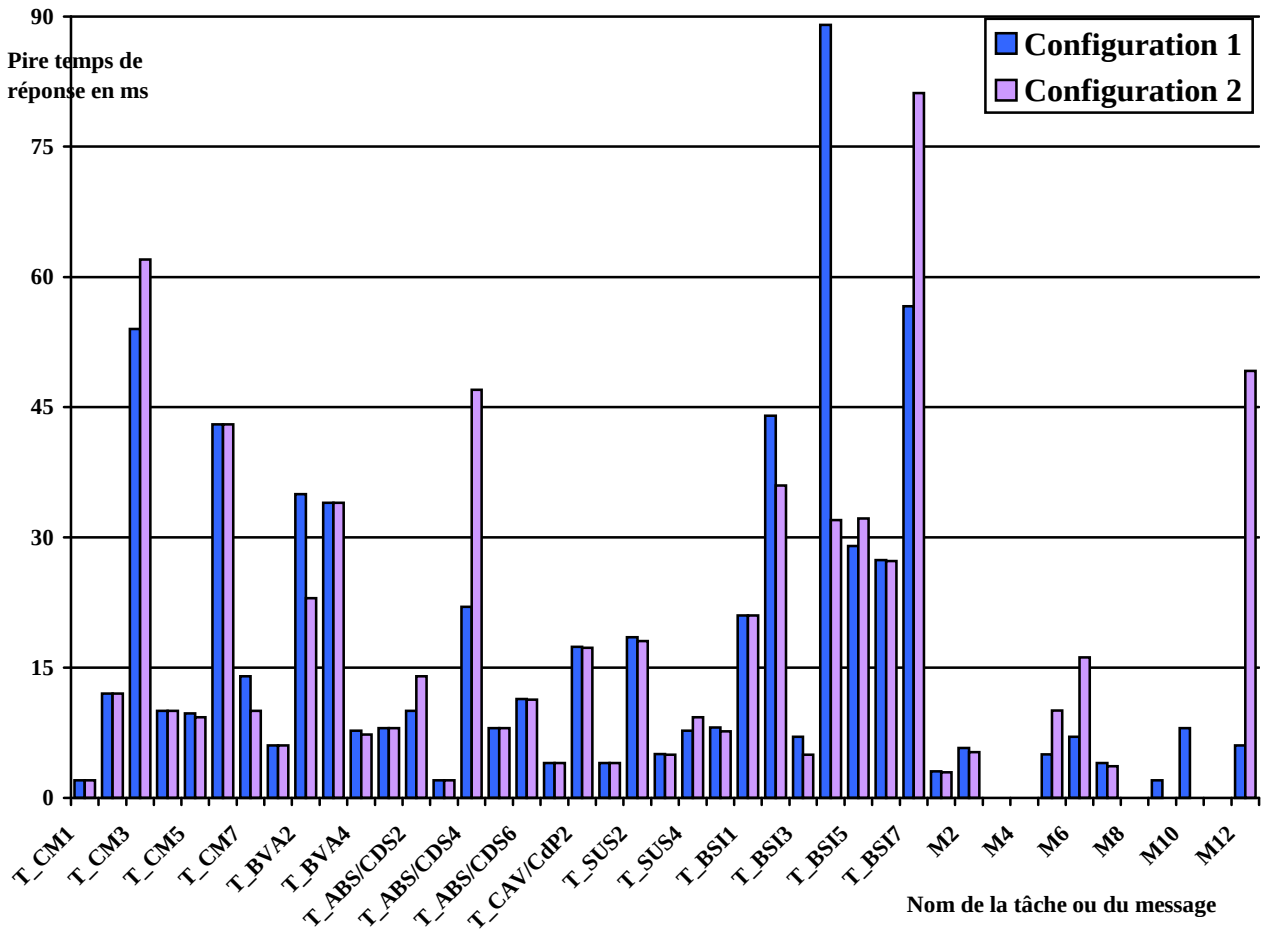


Figure 13 - Comparaison des temps de réponse des deux configurations de la figure 12

III.3. Conclusion

Les algorithmes génétiques fournissent donc un outil très intéressant pour l' affectation de priorités aux tâches et aux messages d' un système distribué. Afin d' optimiser les algorithmes, la génération de la population initiale doit être faite à l' aide d' heuristiques. Celles qui ont été utilisés, particulièrement au niveau du placement des tâches sur les différents sites ne sont pas suffisantes. En effet l' heuristique de placement sur site ne réalise pas une optimisation de celui-ci mais vérifie uniquement la contrainte sur la charge du processeur. Cependant, créer et implanter de nouvelles heuristiques sont des opérations coûteuses en temps. Ainsi cette version de l' algorithme génétique fournit déjà des résultats satisfaisants. Il a montré ses limites lors de la recherche d' une solution lorsque, dans l' énoncé initial on double les temps de calcul des tâches, alors l' algorithme génétique fournit des configurations avec un minimum de 9 erreurs (population de 1000 individus). Une erreur correspond au fait qu' une tâche ou un message ne respecte pas son échéance.

Conclusions et perspectives

I. Conclusions

La correction d' un système temps réel dépend non seulement de la justesse des résultats mais aussi de leur instant de production. Pour vérifier le respect de ces contraintes de temps il est nécessaire de pouvoir connaître le temps mis pour fabriquer ces résultats. Dans le cas où le système est distribué, l' élaboration des résultats implique des tâches et des messages. On cherche à évaluer leur borne maximale (pire temps de réponse).

D' une part le calcul du pire temps de réponse d' une tâche dépend de ces caractéristiques propres et de celles des autres tâches en concurrence avec elle, mais aussi de la politique d' ordonnancement choisie et des caractéristiques matérielles du site sur lequel elle s' exécute. De plus elle dépend, le cas échéant, du temps de réponse du message qui la rend active.

D' autre part, le calcul du pire temps de réponse d' un message dépend des caractéristiques du message et des messages en concurrence avec lui, mais aussi du protocole de communication et des performances du réseau. De plus puisqu' un message est émis par une tâche, son temps de réponse dépend aussi de celui de la tâche émettrice.

Il n' est donc pas possible de réaliser indépendamment une étude du pire temps de réponse sur chaque site et sur le réseau car on ne prendrait, alors, plus en compte les contraintes de précédence.

En raison de ces interdépendances, le calcul de ce pire temps de réponse est complexe et nécessite donc de mener une étude conjointe de l' ordonnancement des tâches et des messages. Le principe du calcul repose sur la prise en compte de ces temps dus à la précédence. On utilise ainsi la gigue pour intégrer ces temps au calcul indépendant sur chaque site et sur le réseau pour les pires temps de réponse respectifs des tâches et des messages. Cette gigue est supposée nulle au début et au fur et à mesure elle est mise à jour pour chaque élément jusqu' à ce que le calcul global des temps de réponse du système converge.

Cette étude m' a permis de créer un outil de calcul du pire temps de réponse des tâches et des messages. Cet outil nécessite la connaissance des priorités des tâches et des messages.

Puisque l' ordonnancement de taches et messages est un problème NP-complet, j' ai décidé d' utiliser comme méthode d' affectation de priorités un algorithme génétique avec comme fonction d' évaluation mon algorithme de pire temps de réponse. Les heuristiques d' affectation de priorités reposent sur les algorithmes Rate Monotonic et Deadline Monotonic, et l' heuristique de placement des tâches sur le site vérifie la charge processeur.

Cet algorithme génétique a fourni des solutions ordonnançables pour des problèmes divers. Dans certains cas, les solutions créées peuvent avoir des temps de réponse différents.

II. Perspectives

Cet algorithme est encore optimisable au niveau même de la programmation mais aussi au niveau des heuristiques de placement sur sites et d' affectation de priorités.

De plus cet algorithme génétique crée souvent de nombreuses configurations solution du problème traité, et alors une question de choix se pose : laquelle faut-il utiliser ? En fait, on pourrait envisager d' utiliser d' autres fonctions d' évaluation afin de trier les solutions selon certains critères. Là encore les critères peuvent être multiples, on peut citer par exemple :

- la plus faible somme de tous les temps de réponse
- la plus faible charge processeur sur chaque site
- le plus faible temps de réponse pour les chaînes critiques

Il serait aussi possible de relancer l' algorithme génétique avec comme population initiale la population solution et d' utiliser les nouvelles fonctions d' évaluation pour en plus de trier les solutions, en créer de nouvelles plus fortes selon le critère voulu.

Le but principal de ces critères est de pouvoir diminuer le cas échéant la puissance des processeurs afin de diminuer les coûts.

En fait, il est fort probable que la puissance des processeurs restera stable, que leurs coûts baisseront grâce aux économies d' échelles, que leur fiabilité s' accroîtra et que les gains obtenus seront utilisés pour implanter de nouvelles fonctions dans les véhicules. Le multimédia sera donc l' application qui en profitera majoritairement. De plus sur une telle application le respect des contraintes temporelles ne correspond plus à une sécurité de fonctionnement mais à une qualité de service pour laquelle on peut accepter des dégradations si nécessaire.

Bibliographie

[Audsley 91] N.C.Audsley : *"Optimal Priority Assignement and Feasability of Static Priority Tasks With Arbitrary Start Time", Technical Report number YCS164, Dept Computer Science, University of York 1991*

[Audsley 95] N.C.Audsley, I.J. Bate et A.Burns : *"Putting Fixed Priority Scheduling Theory into Enginneering Practice for Safety Critical Applications"*

[Audsley 90] N.C.Audsley,: *"Deadline Monotonic Scheduling", Dept of Computer Science, University of York, Y01-5DD, 1990*

[Audsley] N.C.Audsley, A.Burns, M.F. Richardson et A.J. Wellings : *"Incorporating Unbounded Algoritms into Predictable Real Time Systems"*

[Bate 98] I. Bate :*"Scheduling and Timing Analysis for Safety Critical Real-Time Systems", Dept Computer Science, University of York 1998*

[Bate-Burns] I. Bate, A. Burns : *"Schedulability Analysis of Fixed Priority Real-Time Systems with Offsets"*

[Bayard 95] M. Bayard et F. Simonot-Lion :*"Impact de l' émergence des réseaux de terrain et de l' instrumentation intelligente dans la conception des architectures des systèmes d' automatisation de processus", Rappor final, Convention 92-p-0239, 1995.*

[Beauvais 99] J.P. Beauvais :*"Placement de tâches temps réel dans un système réparti"*

[CAN] D. Paret : *"Le bus CAN", Réseaux locaux, Dunod.*
D. Paret : *"Le bus CAN, Applications CAL, CANopen, DeviceNet, OSEK,..", Dunod*
W. Lawrenz :*CAN System Engineering, From Theory to Practical Applications", Springer.*

[Cardeira 94] C. Cardeira : *"Ordonnancement temps réel par réseaux de neurones"*

[Cartwright 95] H. M. Cartwright : *"The Genetic Algorithm in Science", Physical and Theoretical Chemistry Laboratory, Oxford University OX1-3QZ, 1995*

[Caux 94] C. Caux, H. Pirreval et M.-C. Portmann : *Les algorithmes génétiques et leur application aux problèmes d' ordonnancement" Journées Ordonnancement et Entreprise, actes pages 5-45, juin 1994.*

[Courrier 98] M. Courrier, S. Wolf, F. Simonot-Lion, Y.-Q. Song : *"Modélisation de composants matériels et exécutifs en vue de la validation d' architecture opérationnelle par évaluation de performances Rapport intermédiaire du contrat PSA 033", 1998, Loria 98-R-056*

[Djerid 95] L. Djerid, M.-C. Portmann, et P. Villon : *"Performance Analysis of Previous and New Proposed Cross-Over Genetic Operators Designed for Permutation Scheduling Problems". In proceedings International Conference on Industrial Engineering and Production Management, pages 487-497, Marrakech(Maroc), Avril 1995.*

[George 96] L. George, N. Rivierre et M. Spuri :*"preemptive and Non-Preeptive Rel Time Uni-Processor Scheduling", Rapport de Recherche INRIA 2966, 1996.*

[Golberg 89] D.E. Goldberg :*"Genetic Algorithms in search, optimization, and machine learning", Addison-Wesley, Reading, MA, 1989*

[Grolleau 98] E. Grolleau, A. Choquet-Geniet et F. Cottet :*"Cyclicité des Ordonnancements au Plus Tôt des Systèmes de Tâches Temps Réel", RenPar' 10, Strasbourg, 1998*

[Hansson] H. Hansson : *"Distributed Real-Time Systems : A Survey", Technical report DoCS94/48*

[Henn 75] R. Henn :*"Deterministic Modelle fur die Prozessorzuteilung in einer harten Realzeit-Umgebung", Phd Thesis, Technical Univ. Munich, 1975.*

[Holland 75] J.A. Holland : *"Adaptation in Natural and Artificial Systems", Mit Press, Cambridge, Mass.,1975.*

[ISO 94] International Standard Organization :*"Road Vehicles – Low Speed Data Communication- Part2 : Low Speed Controller Area Network (CAN)", ISO 11519-2, 1994*

[Layland 73] J.W. Layland et C.L. Liu :*"Scheduling Algorithms for multiprogramming in a Hard Real Time Environment", Journal of the ACM, Vol. 20, pp 46-61, 1973.*

[Liu 73] C.L. Liu : *"Scheduling Algorithms for Multiprogramming in Hard Real-Time Environment", Journal of the ACM, 20, January 1973, pp46-61*

[Mammeri 91] Z. Mammeri et J.P. Thomesse :*"Réseaux locaux : couche physique et liaison de données", Editions Teknea, 1991*

[Mammeri 97] Z. Mammeri : *"Réseaux et Temps Réel – Ordonnancement de Messages", Actes de l' Ecole Temps Réel, pp 62-79, Poitiers 1997.*

[Navet 99] N. Navet et Y.-Q. Song : *"Une politique à changement de priorité pour l' ordonnancement de messages dans des environnements bruités", CFIPP99.*

[OSEK] Communication version 2.1r1
Operating System version 2.0r1
Network Management version 2.5
OIL (OSEK Implementation Language) version 2.1
(http://www-iiit.etec.uni-karlsruhe.de/~osek/)

[Portmann 96] M.-C. Portmann : *"Scheduling Methodology : Optimization and Compu-Search Approaches I. In A. Artiba and S.E. Elmahgraby, editors, Production and Scheduling of Manufactoring System, pages 271-300. Chapmann&Hall. 1997,1996.*

[Saad 98] S. Saad-Bouzefrane : *"Etude Temporelle des Applications Temps réel Distribuées à Contraintes Strictes basée sur une Analyse d' ordonnançabilité", doctorat de l' Université de Poitiers, janvi 1998.*

[Sha 88] L. Sha, R Rajkumar et J.P. Lehoczky : *"Real-time synchronisation protocols for multiprocessors", IEEE Real-Time Systems Symposium, pp. 259-269, 1988.*

[Sha 90] L. Sha, R Rajkumar et J.P. Lehoczky : *"Priority inheritance protocols: An approach to Real-Time Synchronisation", IEEE Trans. on Computers, -39(9):1175-1185,1990*

[Song 96] Y.-Q. Song, F. Simonot-Lion, N.Navet :*"Validation of Distributed Real Time Systems thanks to Performance Evaluation of their Physical Architecture", In Symposium on Discrete Events and Manufacturing Systems / Computational Engineering in Systems Applications, CESA 96 IMACS Multiconference, Lille, France, p.507-512 juillet 96, Loria 96-R-126*

[Stankovic 88] J; Stankovic: "*Misconception about Real Time Computing : a serious problem for the next generation systems", IEEE Computer Magazine, 21 (10), pp 0-19, 1988*

[Thomesse 93] J.P. Thomesse : *"Le réseau de terrain FIP", Réseaux et Informatique Répartie, Vol 3, pp 287-321, 1993*

[Tindell et al 94] K. Tindell, A. Burns et A. Wellings : *"An Extensible Approach for Analysing Fixed Priority Hard Real-Time Tasks", Real Time Systems, Vol 6, pp 133-151, 1994*

[Tindell 93] K. Tindell : *"Analysis of Hard Real Time Communications" Technical Report number YCS 222, Real-Time Research Group, university of York, 1993.*

[Tindell et al 95] K. Tindell, A. Burns et A. Wellings : *"Calculating Controller Area Network Message Response Time", Uni of York, Y-5DD*

[Tindell-Burns 94] K. Tindell, A. Burns : *"Guaranteeing Message Latencies on Control Area Network (CAN)", First International CAN Conference, Maintz (Germany), 1994*

[Tindell-Burns 93] K. Tindell, A. Burns : *"Scheduling Hard Real-Time Multi-Media Disk Traffic", YCS204, Dept Computer Science, University of York 1993*

[Tindell-Clark 93] K. Tindell, J. Clark : *"Holistic Schedulability Analysis for Distributed Hard Real-Time Systems", YCS197, Dept Computer Science , University of York 1993*

[Tindell-Hansson 97] K. Tindell, et H. Hansson : *"Real Time Systems by Fixed Priority Scheduling", Dept of Computer Systems, Uppsala University, 1994*

[VAN] B. Abou et J. Malville : *"Le bus VAN (Vehicle Area Network) Le fondement du protocole", Dunod*

[Whitley 94] D. Whitley : *"A Genetic Algorithm Tutorial", Computer Science Department, Colorado State University Fort Collins, CO 80523, 1994*

[Zimmerman 80] H. Zimmerman :*"Reference Model – the OSI Model of Architecture for Open Systems Interconnexion", IEEE Trans. on Comm., Vol C28, pp. 425-432, 1980.*

ANNEXE 1 : Le réseau CAN

(Controller Area Network)

L' introduction de l' électronique dans les véhicules au début des années 1980 a fait augmenter les besoins de communication entre les différents organes d' un véhicule. C' est pour répondre à ce besoin nouveau, qu' a été développé le protocole de communication CAN à partir de 1983 par l' équipementier allemand Bosch. Aujourd' hui, il est aussi largement utilisé dans les systèmes de contrôle-commande des procédés industriels de tous type en tant que réseau de terrain. Les contraintes de temps réel, de fiabilité, de résistance aux perturbations électromagnétiques et d' utilisation ont conduit à la définition de deux normes.

En effet, après qu' une première normalisation de CAN fut conduite à terme, il est apparu que le protocole pouvait être insuffisant pour certaines applications bien particulières. Une seconde normalisation a donc été menée à bien. La première norme a été alors renommée en CAN 2.0A (appelé aussi CAN Standard) et la deuxième en CAN 2.0B (CAN Etendu)

I. Architecture

Le réseau CAN est destiné à être embarqué à bord de véhicules. Il est prévu pour permettre l' échange de messages entre composants électroniques. La topologie est un bus qui autorise la diffusion de messages avec accès au médium priorisé et résolution des collisions non destructive (protocole CSMA/CA pour Carrier Sense Multiple Acces/Collison Avoidance).

Seules les couches physiques et liaison de données du modèle OSI ont fait l' objet de la normalisation de CAN. Au niveau applicatif, rien n' a été normalisé. Les applications qui utilisent le bus CAN peuvent être construites directement au-dessus de la couche liaison de données, comme le montre la figure suivante

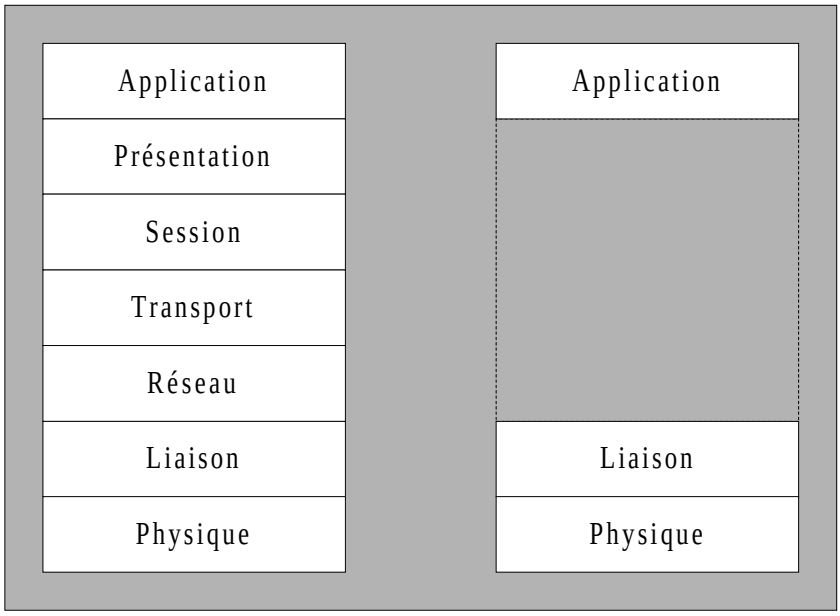


Figure 14 - Couches du réseau CAN

Il est à noter que plusieurs propositions de couches applications ont vu le jour, chacune vise un secteur d'u tilisation particulier.

Voici quelques exemples parmi les plus connus : CAL(CAN Application Layer) du CiA, DeviceNet de la société Allen Bradley et SDS (Smart Distributed System) de Honeywell.

Le rôle assumé par chaque couche est bien défini. Comme on le voit sur la figure suivante, la sous couche d' accès au média (MAC) est celle qui a le plus de responsabilités

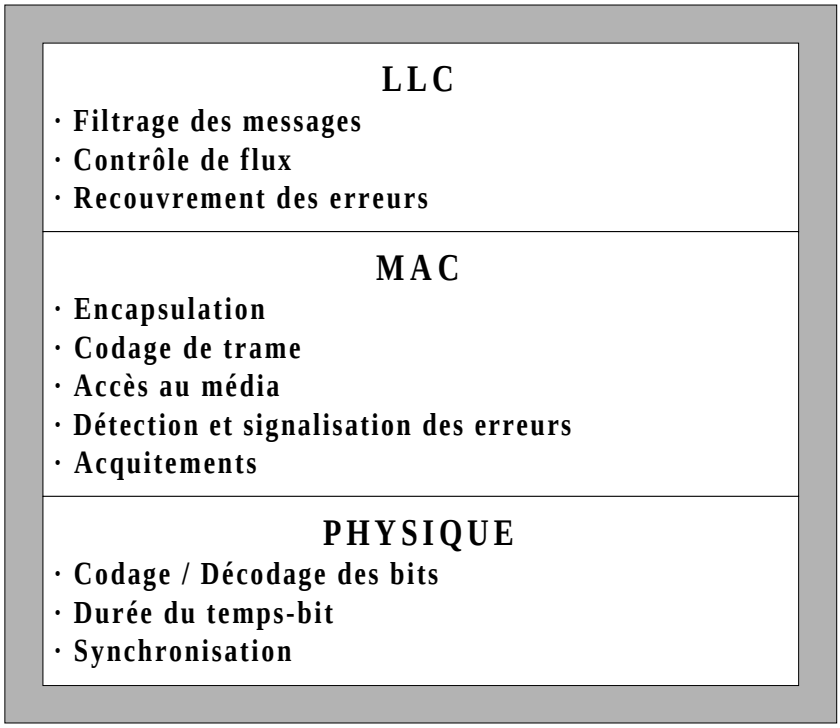


Figure 15 - Rôle des couches CAN

Une des particularités de CAN est qu' il a été conçu en fonction d' une utilisation bien précise. C' est ce qui explique que les couches inférieures sont adaptées aux applications et non l' inverse.

II. Le protocole CAN

Le réseau CAN a été introduit initialement pour gérer la communication dans les véhicules. Le protocole CAN s' appuie sur 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, sachant que plus le bus est étendu, plus le débit est faible.

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

Figure 16 - Débit et distance maximale du bus CAN

Chaque trame peut être transmise périodiquement, sporadiquement ou à la demande. Une trame contient au plus 8 octets d' information utile et des champs de contrôle, dont notamment un identificateur de 11 bits

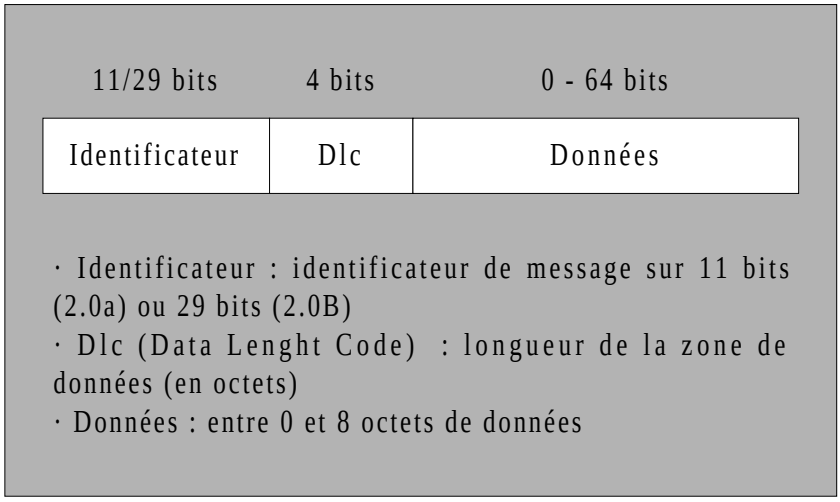


Figure 17 - Format d' une trame de données au niveau liaison

L' identificateur joue un double rôle car il est utilisé d' une part pour filtrer les trames à la réception et d' autre part pour assigner une priorité à la trame qui sert à contrôler l' accès au bus lorsque plus d' un site est émetteur.

En effet, CAN utilise le principe du bit dominant : 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 comporte comme un ET logique, seul la trame de plus grande priorité (ayant le plus petit identificateur) sera envoyée 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 résolution de collisions non destructive.

Le réseau CAN utilise un mécanisme de bitstuffing pour signaler les erreurs aux différents sites. Ce mécanisme 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.

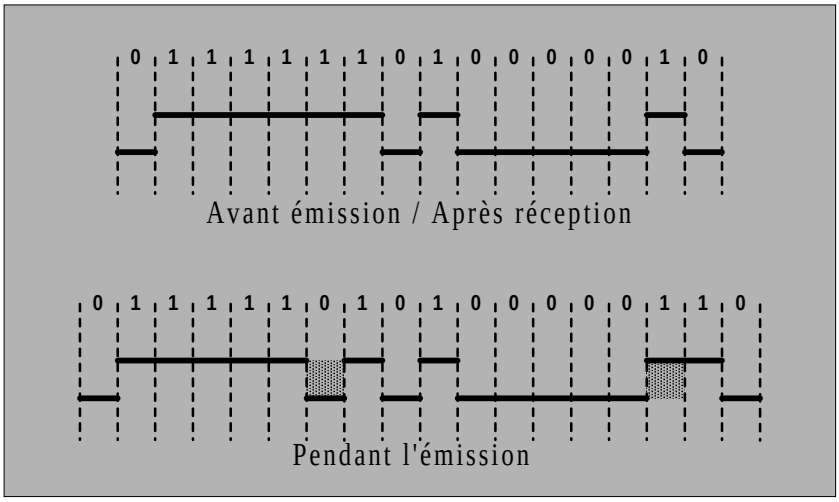


Figure 18 - Méthode de remplissage de bits

Parmi les 47 bits composants les champs de contrôle, 34 bits peuvent contenir des bits de bourrage en plus des 8 octets de données. Le nombre total de bits de bourrage est alors donné par la formule suivante :

(34+8n)/4 bits de bourrages avec n égal 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 130 mus, elle correspond à la taille maximale de la trame (130 bits) c' est à dire au nombre maximal de bits de bourrage (19).

Exemple de compétition entre trois sites pour l' obtention du médium de communication [Saad 98] :

Soient trois sites S1, S2 et S3 qui ont des messages à émettre de priorités respectives 111, 010 et 011.

Durant la première tranche de l' intervalle de temps réservé à l' arbitrage de priorités (figure 19), chaque site diffuse le bit le plus significatif (1 pour S1, 0 pour S2 et 0 pour S3). S1 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 seconde tranche de temps (figure 20), S2 et S3 envoient le bit 1. Le bit lu sur le bus étant identique à celui émis, les deux sites continus le processus d' arbitrage avec les bits suivants.

Dans la troisième tranche de temps (figure 21), S3 est éliminé et S2 gagne l' accès au bus.

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

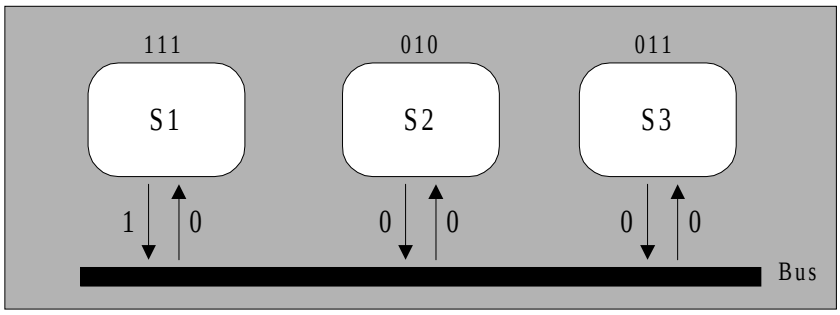


Figure 19 - Tous les sites envoie leur bit MSB

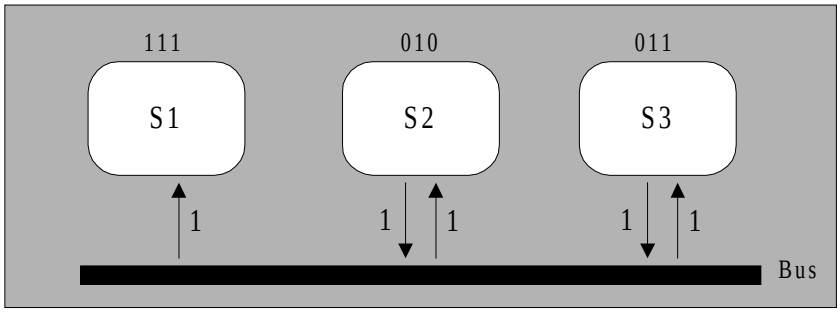


Figure 20 - S1 se retire de la compétition, s2 et s3 continuent le processus d' arbitrage

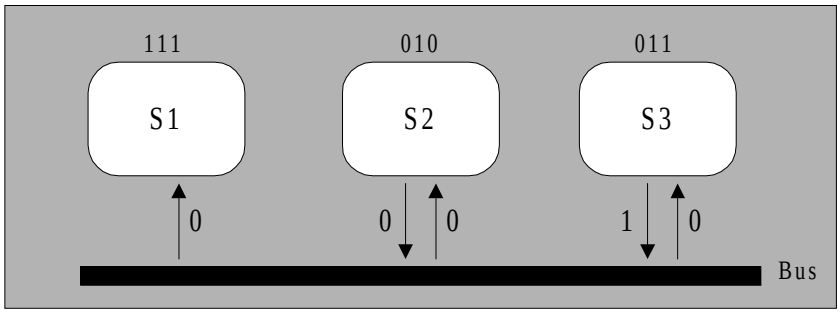


Figure 21 - S3 se retire de la compétition, S2 est donc le vainqueur

Remarques :

Tous les messages arrivés durant la phase d' arbitrage de propriétés ne seront considérés que durant la phase suivante pour l' accès au médium.

Si l' on choisit les identificateurs des trames inversement proportionnels à leurs périodes an associant les identificateurs différents aux trames de même période, alors l' ordonnancement des messages à l' aide de CAN correspondra directement à l' algorithme d' ordonnancement des tâches RM (Rate Monotonic) car les identificateurs sont directement utilisés comme priorité des messages.

ANNEXE 2 : L' exécutif OSEK/VDX

OSEK/VDX est un projet commun de l'industrie automobile. Son but est de fournir un standard d'architecture ouverte pour les unités de contrôle-commande distribuées dans les véhicules. Le système d'exploitation temps réel, l'interface logicielle, les fonctions de communication et les tâches de gestion du réseau sont ainsi spécifiées. Le terme OSEK signifie Offene System unt deren Schnittstellen für die Elektronik im Kraft – fahrzeug (Open systems and the corresponding interfaces for automotive electronics). Le terme VDX signifie Vehicule Distributed eXecutive. Les fonctionnalités du système d'exploitation OSEK ont été harmonisées avec VDX.

Les motivations d'un tel projet sont doubles :

- d'une part les grosses dépenses régulièrement mise en jeu par le développement de différentes gestions des unités de contrôle logicielles.
- d'autre part l'incompatibilité des unités de contrôle-commande fabriquées par différents équipementiers, due à des interfaces et des protocoles différents.

Les buts de ce projet sont le support de la portabilité et de la réutilisabilité des logiciels applicatifs par :

- la spécification des interfaces, aussi indépendantes que possible des applications, pour les domaines suivants : le système d'exploitation temps réel, la communication et la gestion du réseau
- la spécification d'une interface utilisateur indépendante du matériel et du réseau.
- une architecture efficace : la fonctionnalité doit être configurable pour permettre un ajustement optimal de l'architecture à l'application en question.

Les avantages obtenus grâce à ce projet sont multiples :

- la diminution des coûts et du temps de développement
- l'amélioration de la qualité des logiciels des unités de contrôle-commande des différentes compagnies
- la standardisation des objets de l'interface pour les unités de contrôle-commande ayant différents dessins d'architectures
- le séquencement de l'utilisation de l'intelligence (ressources existantes) distribuée dans le véhicule pour améliorer la performance globale du système, sans nécessité de matériel additionnel.
- permettre l'indépendance vis à vis des implémentations individuelles, puisque la spécification donnée n'impose pas les aspects précis de l'implémentation.

Les spécifications d' OSEK sont disponibles sur internet [OSEK-URL] et comportent quatre modules :

- Le premier, **OSEK-OS**, définit les services élémentaires d' un exécutif temps réel, permettant de faire fonctionner sur un seul processeur une application multitâches en fournissant des services de synchronisation, de gestion du temps, de préemption [OSEK-OS].

- Le second, **OSEK-COM**, définit des services de communication des tâches entre elles, en faisant abstraction de la distribution de l' application. Ces services se chargeront donc de la transmission des messages lorsque les taches communicantes se situent sur deux nœuds différents du réseau. [OSEK-COM]
- Le troisième, **OSEK Network Management**, définit des services de connexion et de gestion du réseau qui se basent sur les services offerts par OSEK-COM [OSEK-NET].
- Enfin, les spécifications de **OSEK-OIL** (OSEK Implementation Language) définissent un langage permettant de décrire une application écrite pour OSEK [OSEK-OIL]

OSEK/VDX fournit des services permettant de programmer une application temps réel distribuée :

- activation de tâches,
- signaux,
- timers et alarmes,
- communication par le réseau.

OSEK/VDX définit plusieurs types d'ordonnancement des tâches : préemptif, non-préemptif ou mixte :

- Le premier manipule des tâches qui peuvent être interrompues par d'autres tâches plus prioritaires.
- Le second manipule des tâches qui ne peuvent pas être interrompues. Il est idéal pour des applications faites de tâches très rapides et de durée bornée, pour lesquelles on perd plus de temps à commuter le contexte qu'à attendre la fin de l'exécution de la tâche.
- Enfin, pour le dernier, chaque tâche a un attribut qui indique si elle peut ou ne peut pas être interrompue par d'autres tâches plus prioritaires. On cumule donc les avantages des deux ordonnancements précédents, au prix d'une consommation de mémoire légèrement supérieure. OSEK est en effet un exécutif temps réel destiné aux applications embarquées, pour lesquelles la mémoire est une ressource critique.

OSEK/VDX gère deux types de tâches : les tâches basiques et les tâches étendues. Les premières peuvent se trouver dans trois états, PRETE, SUSPENDUE, EXECUTEE, selon qu'elles sont prêtes mais en attente de gagner l'ordonnancement, suspendue en attente d'une activation, ou en cours d'exécution. Les secondes peuvent accéder à un état supplémentaire, l'état EN ATTENTE, lorsqu'elles se sont mises en attente d'un événement (figure 22).

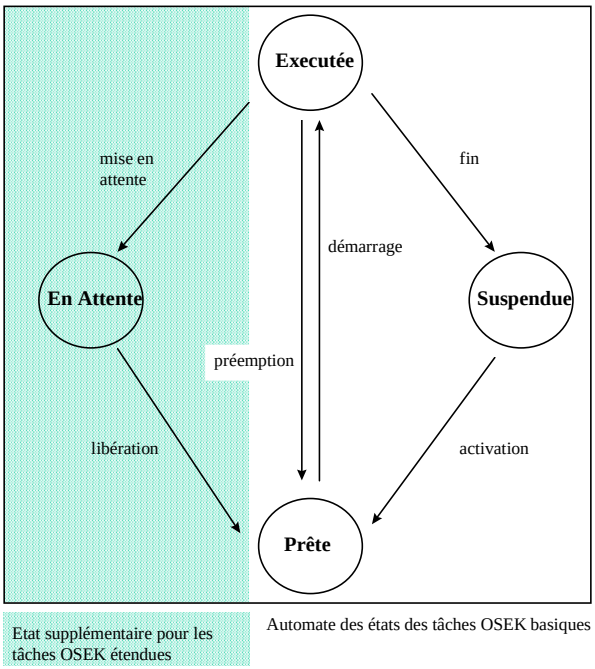


Figure 22 - Automate des états des tâches OSEK (ci contre)