7

Validation de la méthode par des études de cas

La validation de la démarche d'évaluation des performances et des règles de transcription du modèle de performance en VHDL repose sur deux exemples. Le premier exemple nous a été proposé par le CCETT de Rennes et concerne la conception d'un serveur pour séquences audio-visuelles. Le second exemple est un exemple interne à l'équipe de recherche MCSE et concerne un système de communication distribué basé sur l'interconnexion d'un ensemble de cartes identiques avec un bus série du type anneau à jeton.

Bien que dans un premier temps le code VHDL utilisé pour la simulation ait été écrit manuellement de manière à identifier les règles de transcription du modèle de performance en VHDL, ces deux exemples ont ensuite servi aux tests du générateur de code présenté dans le chapitre précédent.

Pour chaque exemple, après une présentation succincte de l'exemple et des objectifs de l'évaluation des performances, nous décrivons le modèle de performance du système et de son environnement. Puis nous commentons les résultats obtenus par simulation du modèle de performance.

7.1 SERVEUR VIDEO TEMPS REEL

Cet exemple fourni par le CCETT de Rennes a été choisi pour montrer l'adéquation de la méthodologie MCSE pour la spécification, la conception et l'évaluation des performances d'un système complexe. Ce travail qui a été effectué en collaboration avec le CCETT de Rennes (G. Babonneau) et l'équipe de J.M Bergé du CNET de Meylan est également présenté dans [CALVEZ-97a].

7.1.1 Présentation de l'exemple

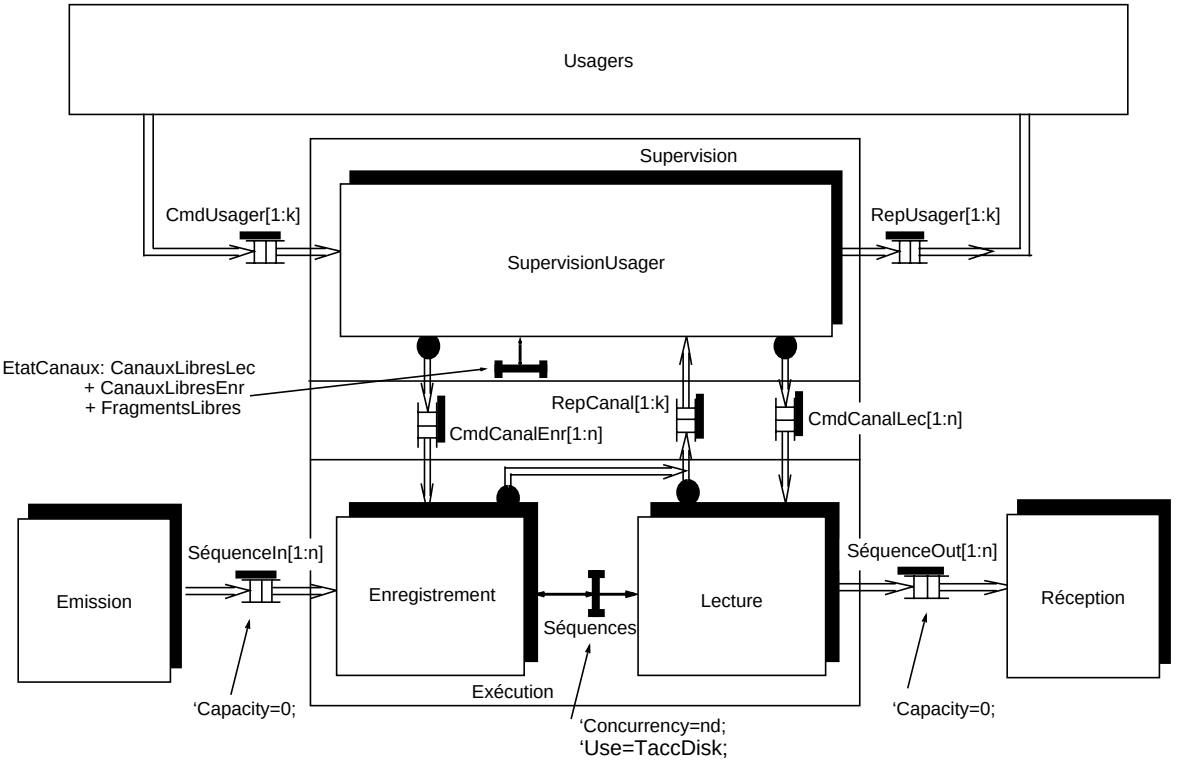
Le serveur audio-visuel se place dans un service de consultation à distance de séquences vidéo ou sonores. Le terme *séquence* signifie ici une unité visuelle ou audio de durée allant de quelques secondes à plusieurs heures. Vis-à-vis de l'usager, une séquence respecte un débit pouvant varier entre 0 et 15 Mbits/s et est décomposée temporellement en unités plus élémentaires appelées fragments. Un *fragment* correspond à une quantité fixe d'information définie par la technique de mémorisation du serveur. L'usager peut contrôler la visualisation ou l'enregistrement par fragment (par exemple pause entre 2 fragments mais pas durant un fragment). En terme de fonctionnalité pour un usager, le serveur joue un rôle analogue au magnétoscope et au magnétophone. L'usager peut fournir les commandes suivantes: lecture d'une séquence, enregistrement d'une séquence, pause, restart, stop (ce qui veut dire fin de la séquence courante), avance rapide, retour rapide, effacement séquence. Les réponses du système lui permettront de connaître l'état et les réactions du serveur.

Pour les interfaces physiques, le couplage de chaque usager se fait par un réseau de faible débit et un réseau de haut débit. Le réseau faible débit peut être: X25, NUMERIS, Ethernet, etc. Le réseau haut débit considéré est le réseau ATM.

Les fonctions du système sont les suivantes:

- la lecture de séquence(s) qui assure la sortie fragment par fragment de la séquence choisie au débit souhaité,
- l'enregistrement de séquence(s) qui reçoit en temps-réel une séquence et la mémorise,
- la gestion des requêtes de chaque utilisateur qui alloue si nécessaire un canal et transmet l'ordre pour la lecture ou pour l'enregistrement

La figure 7.1 représente la structure fonctionnelle de l'application complète.



-Figure 7.1- Structure fonctionnelle complète permettant la modélisation de performances.

La structure fonctionnelle est basée sur le modèle Supervision/Contrôle-Commande. Pour la partie Supervision, la fonction SupervisionUsager gère les ordres provenant des utilisateurs (port CmdUsager en entrée et CmdCanalEnr et CmdCanalLec en sortie) et s'occupe aussi du retour d'information vers l'usager concerné (port RepCanal en entrée et RepUsager en sortie). Elle attribue entre autre un canal haut débit à un utilisateur en fonction de l'état (libre ou occupé) de l'ensemble des canaux modélisé par la variable EtatCanaux. La partie Contrôle/Commande est composée des fonctions Lecture et Enregistrement. Ces deux fonctions sont reliées entre elles par la variable partagée Séquences qui représente l'ensemble des disques où sont stockés les fragments des séquences vidéo ou sonores disponibles sur le serveur.

Le comportement du système pour les fonctions de la structure fonctionnelle doit être explicité pour disposer d'une spécification exécutable (ce qui veut dire simulable pour la vérification). L'évaluation nécessite aussi qu'un système (et ceci est particulièrement vrai durant la conception car il n'existe pas) soit décrit par un modèle qui, soumis à une configuration de charge donnée, permet d'extraire des informations quantitatives. Pour éviter de définir des stimuli, on modélise également l'environnement du système (fonctions Usagers, Emission et Réception).

7.1.2 Objectif de l'évaluation des performances pour cet exemple

Cet exemple permet de montrer que le modèle de performance est approprié pour aider au dimensionnement et au partitionnement matériel/logiciel d'un système. Dans cet exemple, avant de commencer la conception architecturale, le concepteur doit en effet déterminer:

- le nombre de disques utilisés pour stocker les séquences vidéo ou sonores. Cette information est indispensable pour concevoir l'interface avec les disques.
- la taille des buffers d'anticipation nécessaires pour éviter toutes rupture de séquences. Cette information est utile pour déterminer le taille de la mémoire du système.

L'allocation des éléments fonctionnels sur les éléments exécutifs et le partitionnement matériel/logiciel ne pose pas trop de problèmes. En effet, le système est basé sur le modèle Supervision/Contrôle-Commande avec des contraintes faibles pour la partie Supervision et des contraintes fortes pour la partie Commande. On s'intéressera donc plus particulièrement aux performances du processeur en charge de la partie commande (ou partie dite opérative).

7.1.3 Modèle de performance du système

L'étude des performances peut commencer par une modélisation statique qui permet d'obtenir le nombre des disques représentés dans un premier temps par la variable partagée séquences. Ensuite pour cet exemple, il est indispensable d'effectuer une modélisation dynamique de manière à déterminer plus précisément les paramètres de l'architecture fonctionnelle du système que sont la taille des ports et des fifo internes aux fonctions Lecture et Ecriture et les performances des fonctions. Cette modélisation résumée ci-après s'obtient:

- en ajoutant des paramètres ou attributs de comportement aux constituants de la solution fonctionnelle de la figure 7.1, (‘Concurrency définit le nombre de disques, ‘Use définit le temps d'accès pour la variable Séquences et donc celui des disques, ‘Capacity est la taille en nombre de messages de chaque port),

- en exprimant le modèle comportemental de performance pour chaque fonction de la structure fonctionnelle (voir les paragraphes suivants),
- en modélisant la charge du serveur (workload) par simulation de l'environnement pour représenter le scénario imposé pour l'évaluation.

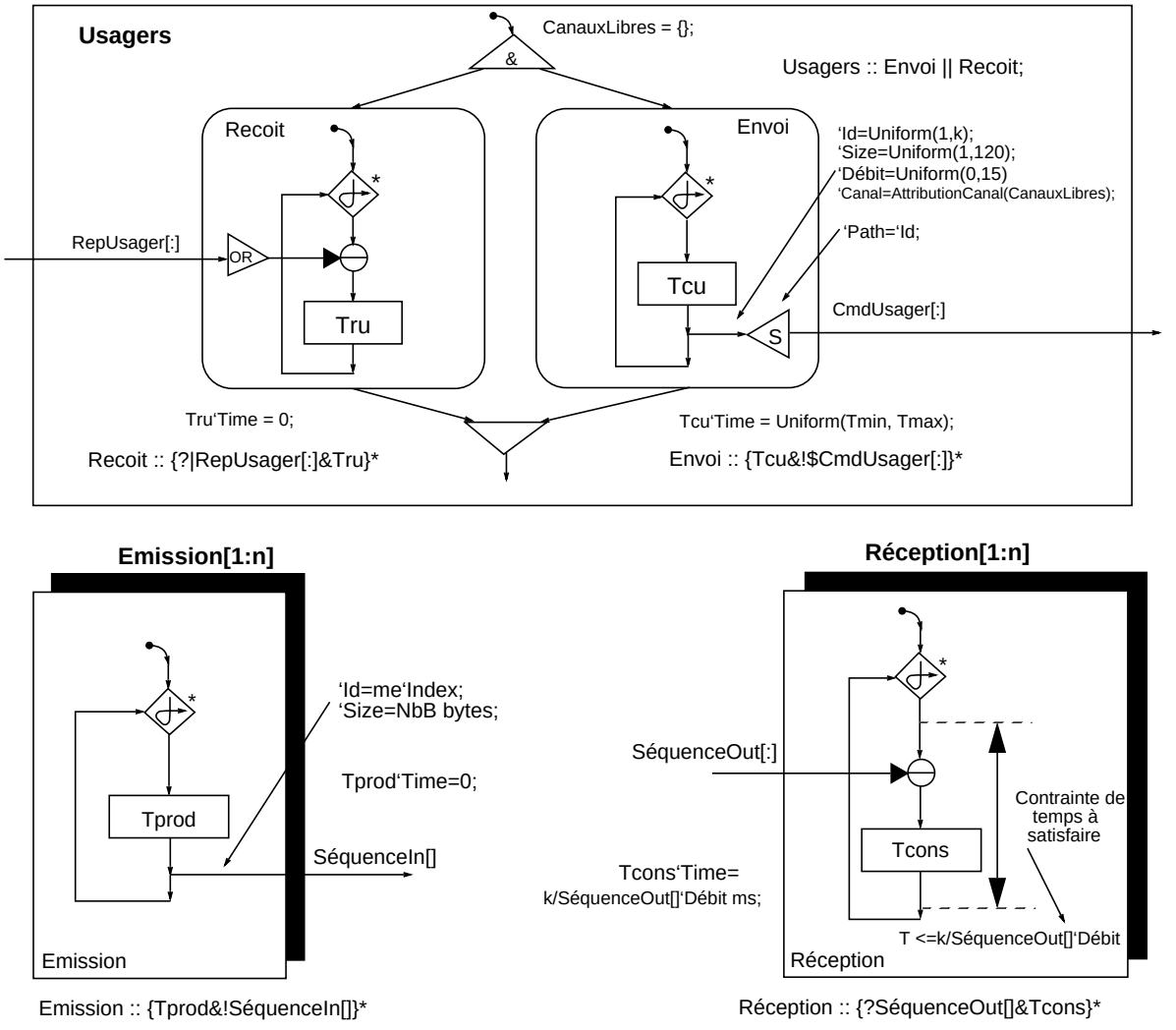
-A- Modélisation Statique

Pour avoir un système stable, il faut obligatoirement que le flux de sortie soit supérieur ou égal au flux d'entrée. Le nombre de disques nécessaires se calcule donc par:

$$NbDisques = (NbUtilisateurs \times DebitMoy \times TaccDiskMoy) / (TailleFragment)$$

-B- Modélisation des fonctions de l'environnement

La modélisation de l'environnement comprend la spécification du comportement souhaité pour placer le système dans le contexte imposé pour son évaluation. La figure 7.2 représente le résultat de cette modélisation.



-Figure 7.2- Spécification du comportement de l'environnement pour les performances.

La fonction Usagers est modélisée comme 2 activités simultanées: Envoi et Reçoit. Reçoit sert simplement à consommer les messages en provenance du serveur. Envoi génère selon un intervalle aléatoire défini par le temps Tcu, un message destiné à un usager tiré d'une manière

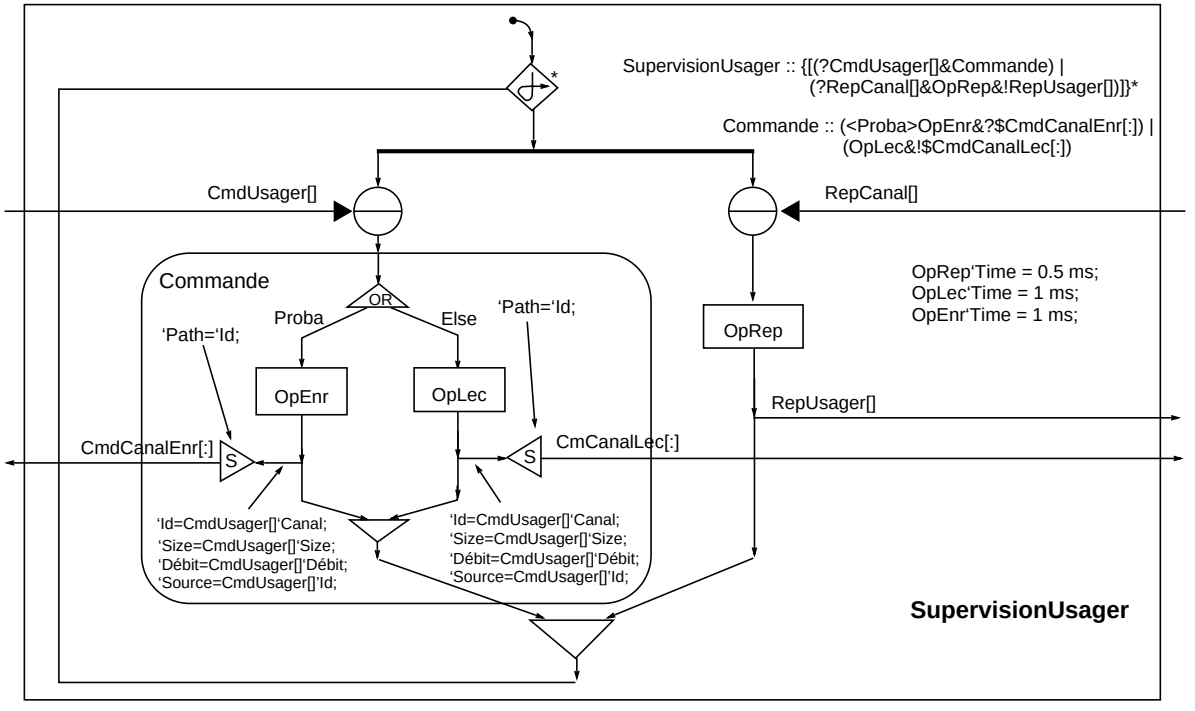
aléatoire et possédant une taille en nombre de fragments elle aussi aléatoire qui servira à définir la durée de la séquence. Un attribut particulier ‘Débit a été ajouté pour définir le débit de transmission pour la séquence correspondante, ainsi qu’un attribut ‘Canal pour définir le canal à utiliser.

Chaque fonction Emission produit en permanence des fragments. La vitesse de consommation de ces messages sera définie par le serveur. Chaque fonction Réception se charge de consommer les fragments à la vitesse définie par le débit de la séquence. La vérification de l’absence de rupture de séquence se fait par chaque fonction Réception par vérification de la contrainte de temps. Pour cette vérification, le message reçu par SéquenceOut[] doit contenir le débit ‘Débit comme attribut.

-C- Modèle pour SupervisionUsager

Chaque message CmdUsager[] est interprété puis transmis vers la lecture ou l’enregistrement d’un canal tiré d’une manière aléatoire (Proba sur la branche). Chaque message RepCanal[] possède la signification de la fin de la séquence, ce qui engendre un message RepUsager[]. Le comportement de la fonction est voulu séquentiel pour une modélisation la plus réaliste possible, ce qui justifie l’attente alternative. La taille de la séquence, le débit et l’usager source sont transmis vers le canal correspondant désigné par l’attribut ‘Canal.

La figure 7.3 donne le modèle de cette fonction.



-Figure 7.3- Modèle de comportement pour chaque fonction SupervisionUsager.

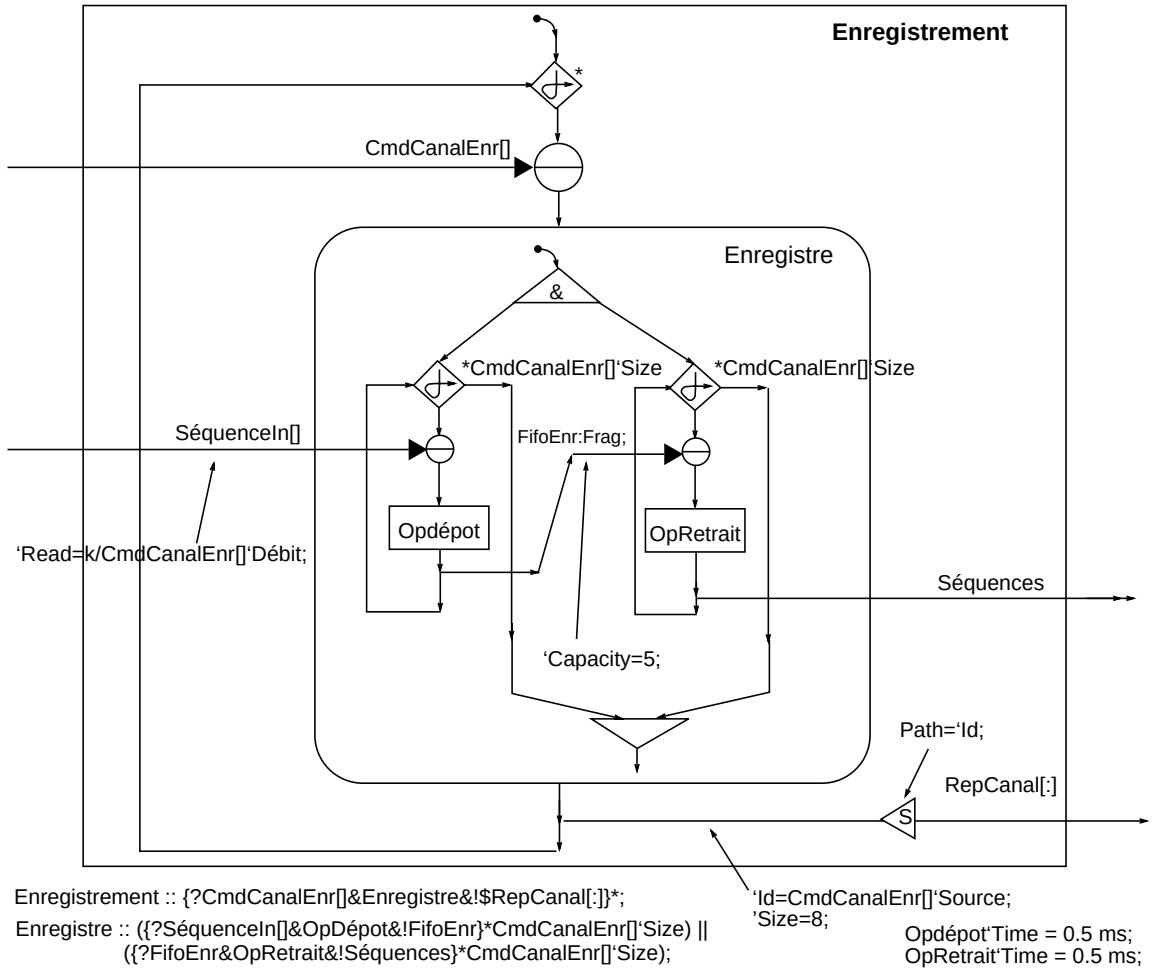
-D- Modèle pour Enregistrement

Une analyse statique de l’application montre que la fréquence d’arrivée de fragments peut être supérieure à la fréquence maximale d’écriture sur un disque. L’emploi de disques multiples permet de résoudre ce problème. Pour les variations ponctuelles de vitesse d’écriture sur les disques dues à la concurrence d’accès et à la variation du temps d’accès, il est nécessaire de considérer une fifo entre les messages de SéquencesIn[] et les écritures dans Séquences.

La figure 7.4 donne le modèle de cette fonction. Pour simuler une séquence, elle reçoit un nombre de fragments égal à l'attribut 'Size défini dans le message de commande en provenance de CmdCanalEnr[] et qui provient de la demande de l'usager. Lorsque tous les fragments ont été enregistrés, la fin de séquence est représentée par un message transmis à RepCanal[Source].

Les temps d'écriture et de lecture dans FifoEnr sont considérés nuls car ces temps sont associés aux opérations OpDépot et OpRetrait. Un attribut structurel 'Capacity pour FifoEnr définit la taille de la fifo en nombre de fragments. On suppose aussi un accès simultané possible.

Pour simuler correctement l'arrivée de fragments à la vitesse définie par le débit de l'enregistrement, on utilise l'attribut 'Read pour la lecture d'un fragment dans le port SéquenceIn[]. La capacité de 0 pour ce port (port du type rendez-vous) va engendrer un asservissement de la fonction Emission à la fonction Enregistrement.

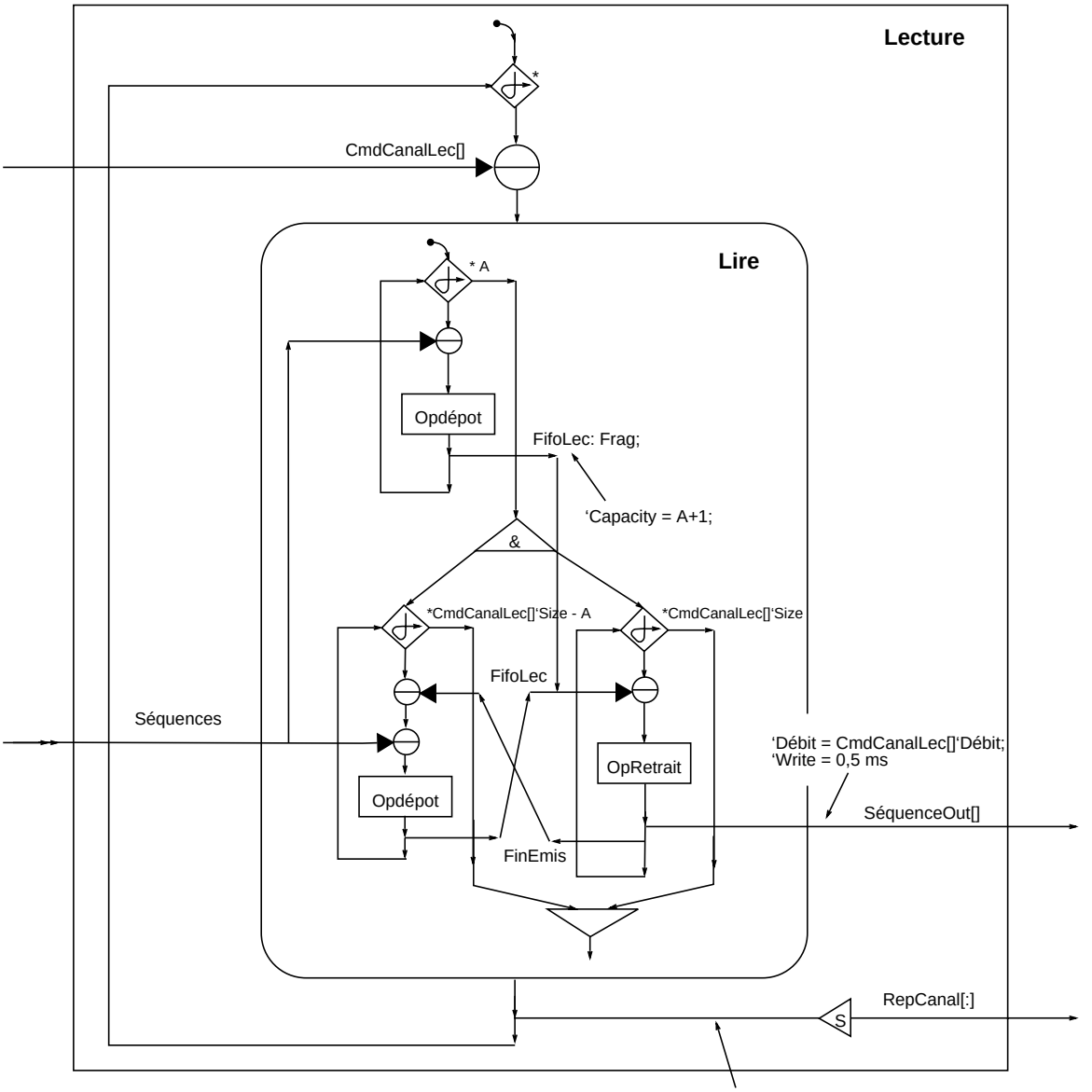


-Figure 7.4- Modèle de comportement pour la fonction Enregistrement.

-E- - Modèle pour Lecture

La modélisation pour la fonction Lecture est assez similaire à celle adoptée pour Enregistrement à la différence qu'il faut disposer d'un certain nombre de fragments en anticipation avant d'assurer l'émission en temps-réel sans rupture de séquence. Une Fifo appelée FifoLec est utilisée pour cela. Une première phase consiste à rechercher A fragments avant d'initialiser la transmission. FifoLec est caractérisée par son attribut structurel 'Capacity dont la valeur est supérieure à A.

Pour simuler correctement la vitesse de transfert correspondant au débit, on utilise d'une part l'attribut 'Write pour l'accès au port SéquenceOut[], d'autre part l'attribut 'Débit dans le message SéquenceOut[] et qui définit la vitesse de lecture du port SéquenceOut[] par la fonction Réception[] chargée de simuler le récepteur.

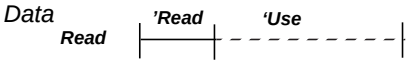


-Figure 7.5- Modèle de comportement pour la fonction Lecture.

7.1.4 Simulation du serveur en lecture

Les disques représentés dans le modèle par la variable partagée Séquences sont modélisés sous la forme d'une ressource commune. Cette ressource (voir figure 7.1) est caractérisée par son degré de partage 'Concurrency et sa capacité en nombre de bits ou de mots 'Capacity (grandeur statique ou dynamique). Pour le comportement, 3 temps sont à considérer pour son utilisation: temps d'allocation ('Write), temps d'utilisation ('Use) et temps de libération ('Read). Le degré de partage concerne les accès simultanés pour ces 3 temps.

L'attente sur la variable Séquence se traduit par une primitive ReadSharVar.



De sorte que si un disque est libre, pour la fonction l'accès au disque ne dure que la valeur de 'Read, mais le disque n'est libéré qu'après 'Read+'Use.

Le temps d'accès aux disques (attribut 'Use de la variable partagée) a d'abord été fixé à 30 ms. Puis pour avoir une modélisation plus réaliste des disques, nous avons considéré que TaccDisk = Tseek + Tlatency + Tread avec les lois aléatoires suivantes pour Tseek et Tlatency:

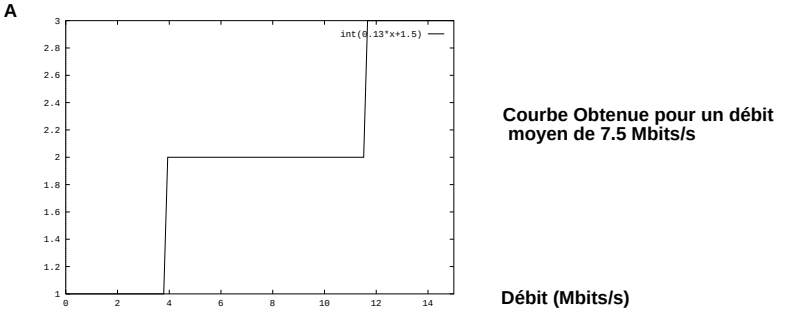
- Tseek = U(2,24) ms (U loi aléatoire uniforme),
- Tlatency = U (0,11) ms
- Tread = 12,8 ms (temps de lecture nécessaire à 5300 trs/mn pour lire 32 Ko).

Le débit pour une séquence et un utilisateur est tiré aléatoirement selon une loi uniforme entre 0 et 15 Mbits/s. De plus, pour privilégier la fonction lecture par rapport à la fonction enregistrement, dans le modèle de comportement de la fonction SupervisionUsager la valeur de la probabilité Proba associée à l'alternance est très faible (0,0005).

Enfin et surtout la valeur de **A** (taille du buffer d'anticipation) n'est pas fixe et commune à tous les canaux haut débit, mais elle **dépend de la valeur courante du débit de la séquence lue**. Pour notre simulation, A est calculé par la loi empirique suivante:

$$A = E(NbUtilisateurs/Nbdisques \times DebitSequence \times TaccDiskMoy/TailleFragment + 1,5)$$

Soit encore $A = E(DebitSequence/DebitMoy + 1,5)$ avec E partie entière



-Figure 7.6- Valeur de A en fonction du débit.

Un autre aspect important concerne la gestion des accès simultanés sur notre variable partagée Séquences. L'entité gérant cet élément de relation est capable d'ordonnancer les accès multiples soit en fonction de leur priorité ('Priority) soit en fonction de leur estampille ('Date).

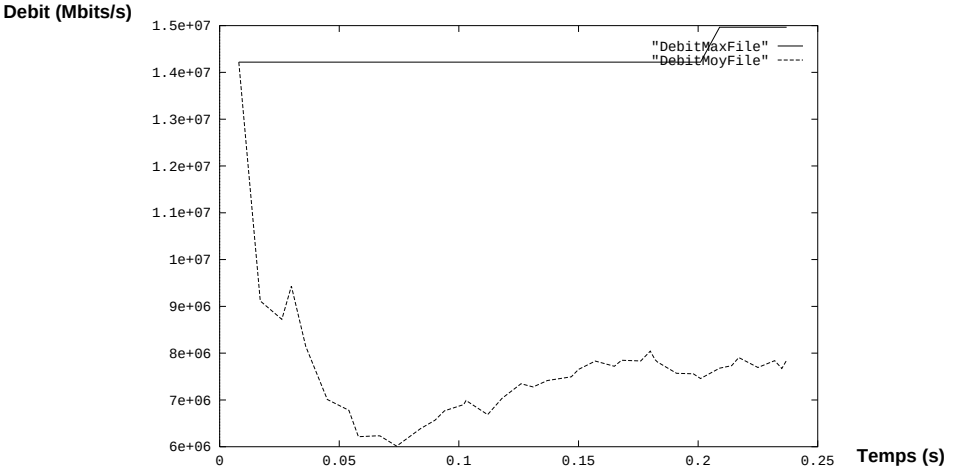
Dans le cas d'un ordonnancement utilisant les estampilles des demandes d'accès (l'ordonnanceur ne prenant en compte que les demandes dont la date est inférieure ou égale à la valeur courante du temps de simulation now), deux politiques sont applicables:

- une politique d'ordonnancement au plus tôt,
- une politique d'ordonnancement au plus tard: cette solution a l'avantage de privilégier les accès aux débits élevés par rapport à ceux correspondant à des débits faibles. Dans ce cas, l'estampille d'une demande d'accès se calcule par:

'date = now + ConversionDebitTemps(debit) - k * TrwFrag,

où now représente la valeur courante de l'instant de simulation et TrwFrag représente le temps d'accès moyen aux disques. K est un coefficient de sécurité (k=2) pour éviter les coïncidences de cas défavorables (ce coefficient n'a pas d'incidence sur A).

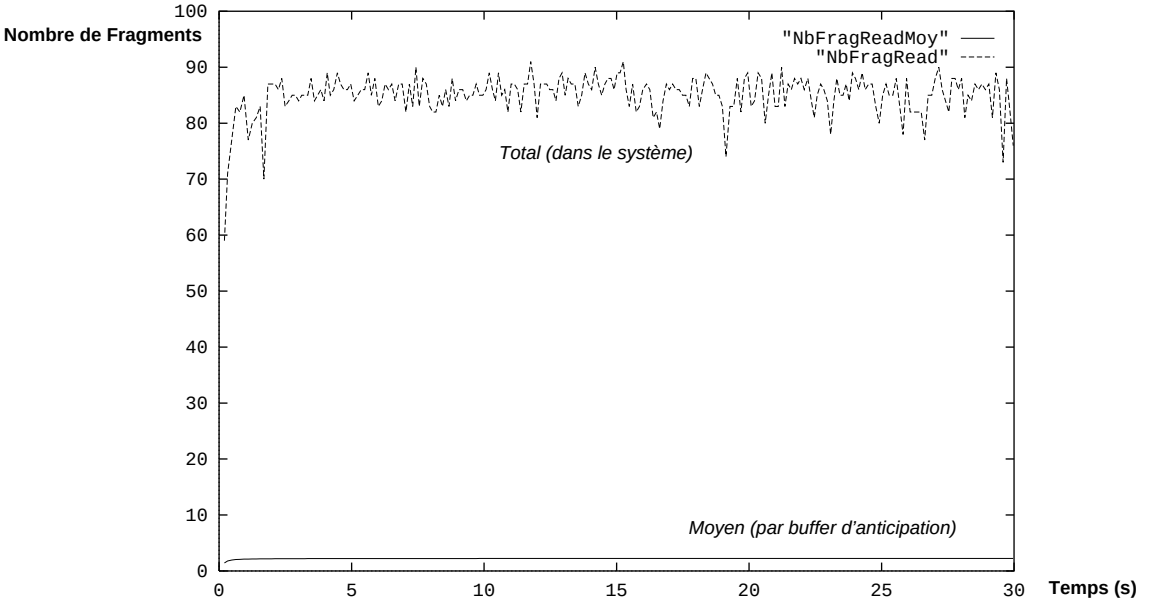
Nous avons choisi les conditions suivantes: le nombre d'utilisateurs et de canaux haut débit du serveur est 40 et le nombre de disques est de 35. La charge imposée au système est représentée par la figure suivante. Les usagers sont activés progressivement de manière aléatoire.



-Figure 7.7- Charge imposée au système pour la simulation en phase de démarrage.

La courbe montre que le premier usager possède un débit de 14 Mb/s. Les figures montrent la valeur moyenne et la valeur maximale.

La variation du nombre de fragments présents dans les fifos internes des fonctions Lecture au cours de la simulation est la suivante.



-Figure 7.8- Nombre de Fragments dans les fifos internes du système pour 40 utilisateurs.

Cette simulation nous a permis d'obtenir un **encadrement de la valeur de A** qui doit se situer entre **2 et 3**. Pour obtenir une valeur plus précise du paramètre A, il faut modéliser les disques d'une manière plus réaliste ce qui est l'objectif du paragraphe suivant.

7.1.5 Modélisation plus réaliste des disques

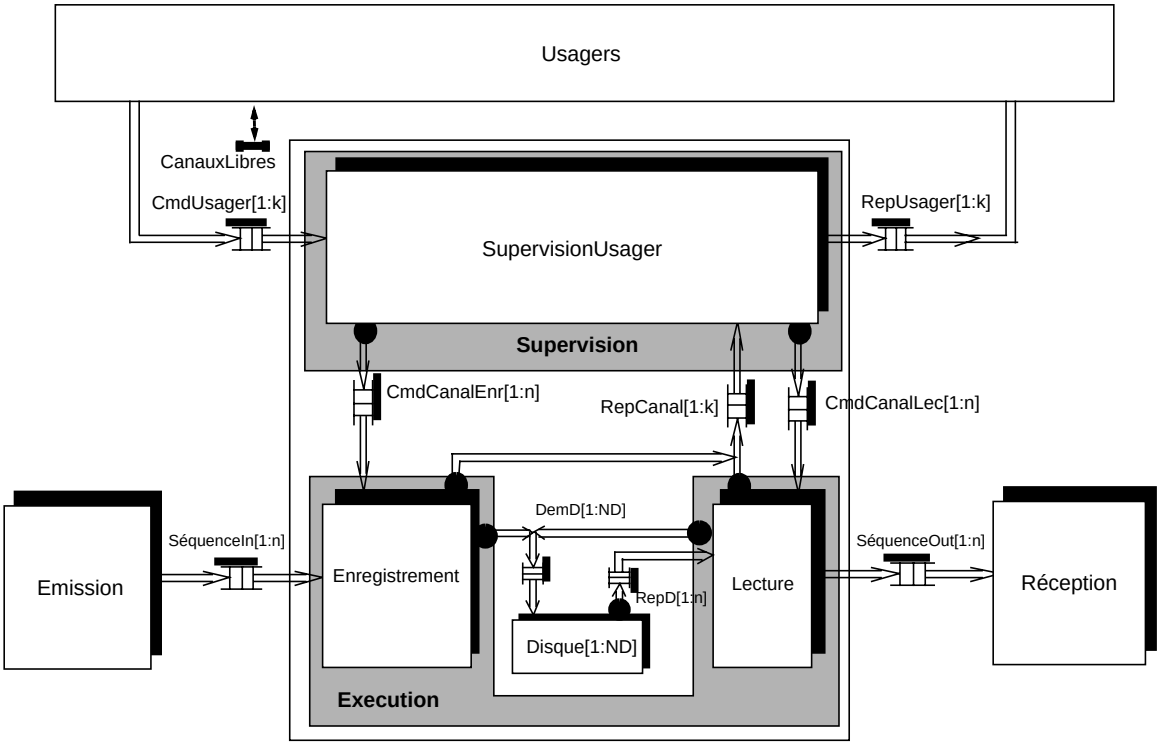
De manière à avoir une évaluation plus fine du serveur, il est nécessaire de considérer un modèle plus réel des disques. Ainsi dans ce paragraphe la variable partagée Disques est remplacée par des fonctions, chacune représentant le comportement réaliste d'un disque.

-A- Structure fonctionnelle et modèles de comportement utilisés

La structure fonctionnelle simulée est représentée par la figure suivante. La variable partagée Séquences est maintenant remplacée par un ensemble de ND fonctions, chacune modélisant un disque.

La nouvelle fonction Disque[i] est décrite ci-après.

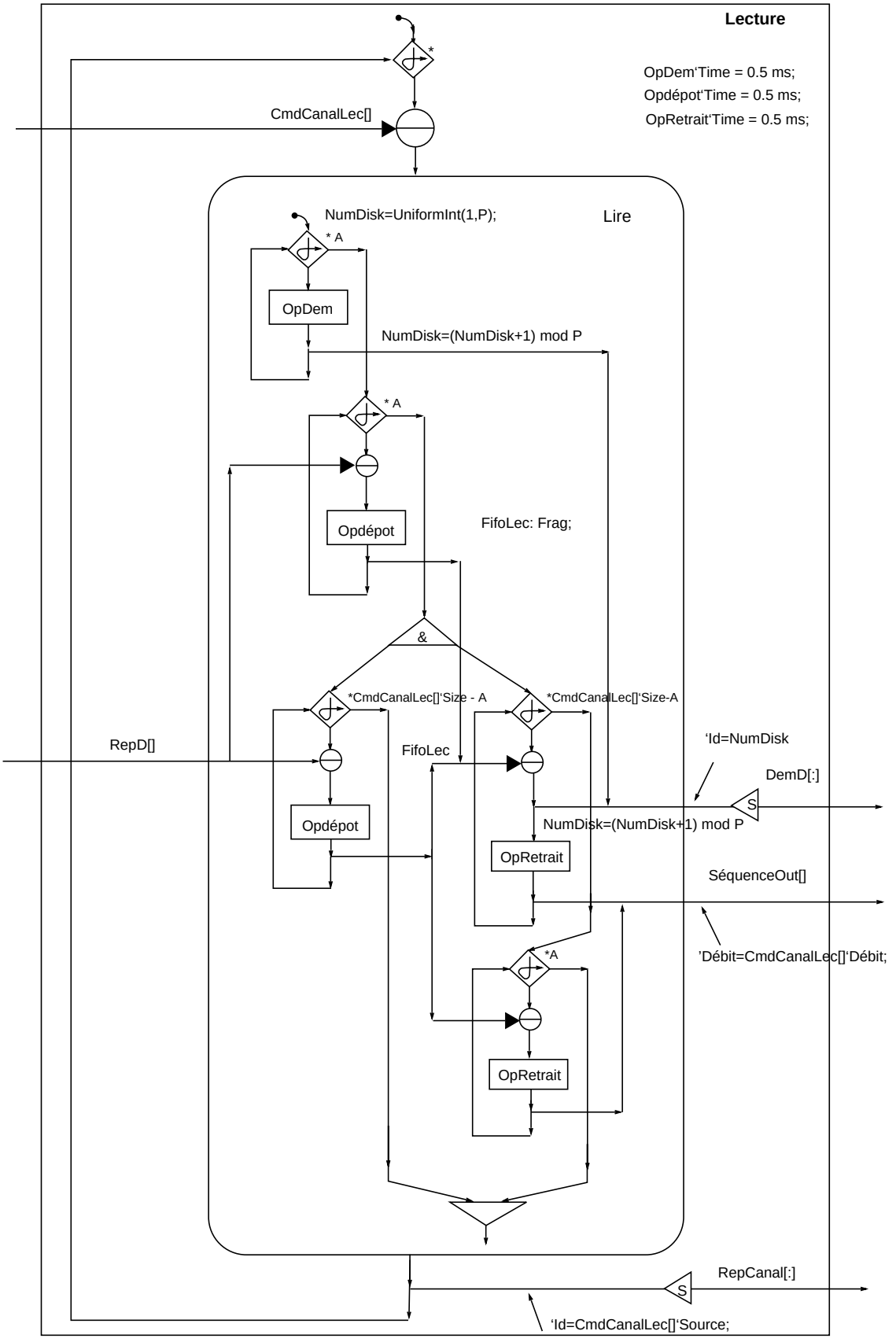
```
Action Disque[] sur mess DemD[]: DefDemD avec
            sortie mess RepD[:]: DefRepD;);
Begin
Cycle DemD[]:
           Attente(TaccDisk);
           IF DemD[]=Lecture THEN
             Send(Disque[DemD[].NoBloc], RepD[DemD[].Dest]);
           EndIF;
      EndCycle;
End Disque[];
```



-Figure 7.9- Structure fonctionnelle avec modélisation réaliste des disques.

Le temps d'accès aux disques est la somme de trois composantes (TaccDisk = Tseek + Tlatency + Tread) avec des lois aléatoires pour Tseek et Tlatency: Tseek = U(2,24) ms (U loi aléatoire uniforme), Tlatency = U (0,11) ms et Tread = 12,8 ms (temps de lecture nécessaire à 5300 trs/mn pour lire 32 Ko).

Avec la nouvelle représentation des disques, le comportement considéré pour la fonction Lecture est représentée par la figure suivante.



-Figure 7.10- Modèle de comportement pour la fonction Lecture.

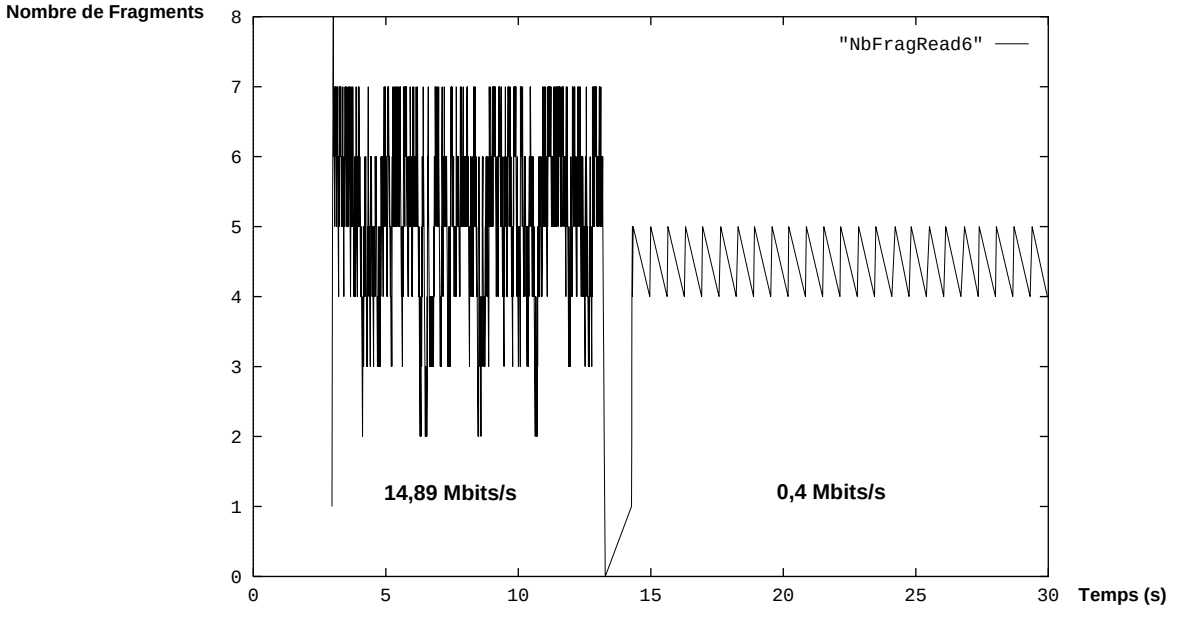
La fonction commence par demander les A premiers fragments puis attend la réception de ceux-ci. Ensuite, elle envoie les fragments les uns après les autres en demandant un nouveau fragment après chaque envoi.

Pour une séquence donnée, l'accès des fragments successifs sur l'ensemble des disques se fait maintenant selon une technique de Round Robin et le numéro de disque du premier fragment est tiré aléatoirement.

Pour cette simulation, le nombre d'utilisateurs et de canaux haut débit du serveur est 10 (limitation liée à la taille de la mémoire dynamique et virtuelle de la machine utilisée pour la simulation). Le débit d'un utilisateur est tiré aléatoirement selon une loi uniforme U(0,15) (débit exprimé en Mbits/s).

-B- Détermination d'une loi pour la valeur de A

La loi empirique de la valeur de A en fonction du débit d'une séquence utilisée lors de la modélisation avec une variable partagée ne s'est plus avérée valable (valeur de A trop faible). Pour déterminer une nouvelle loi, nous avons choisi une valeur de A suffisamment grande pour ne pas avoir de rupture de séquence et nous avons observé l'évolution du nombre de fragments dans le buffer d'anticipation de chaque canal pour des débits différents.



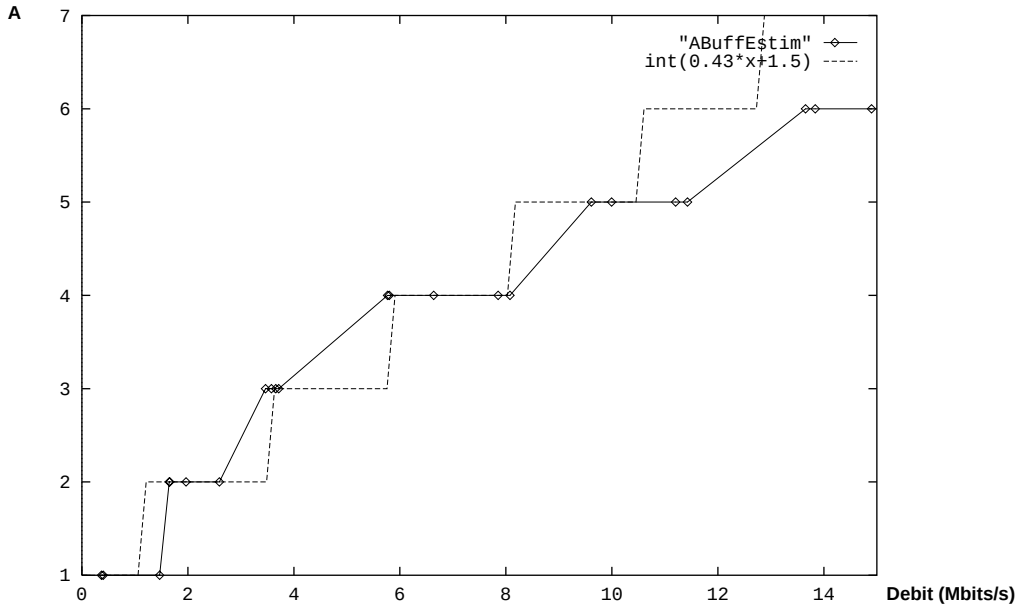
-Figure 7.11- Evolution du nombre de fragments dans le buffer d'anticipation d'un canal.

L'ensemble des courbes obtenues nous a permis en regardant l'amplitude de variation du nombre de fragments dans le buffer d'obtenir un ensemble de points donnant une valeur expérimentale de la taille du buffer pour un débit donné.

Nous avons alors essayé d'extrapoler au mieux les points par une loi qui ne provoquait pas de rupture de séquence en simulation comme le montre la figure 7.12.

A est maintenant calculé par la loi empirique suivante:

$$A = E(\mathbf{3} \times NbUtilisateurs / Nbdisques \times DebitSequence \times TaccDiskMoy / TailleFragment + 1,5)$$

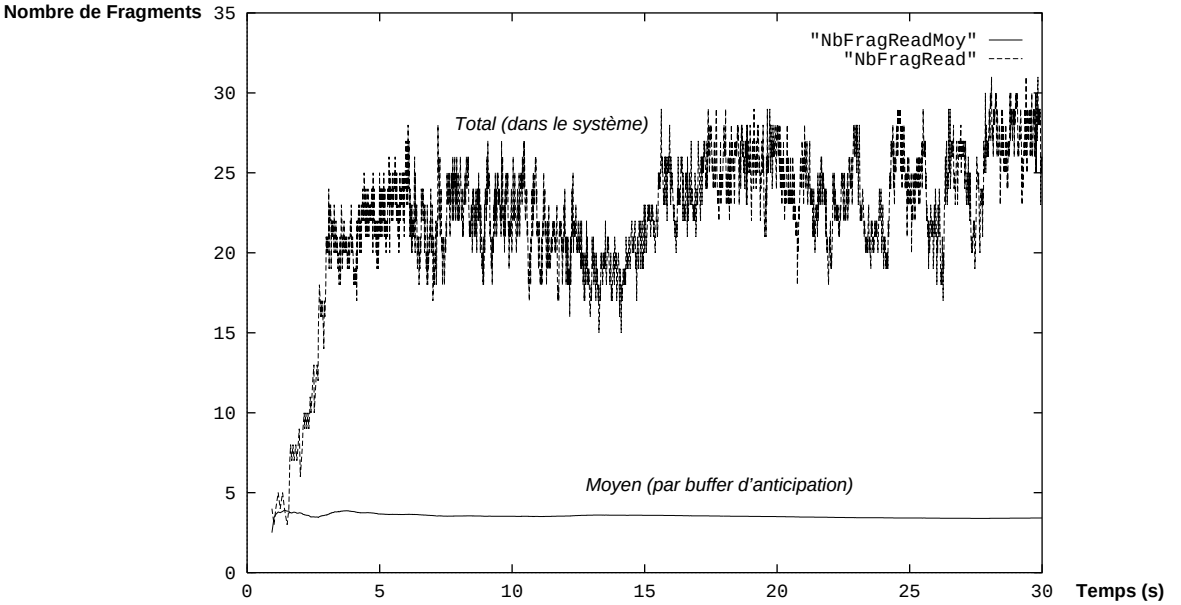


-Figure 7.12- Détermination d'une loi pour le calcul de A en fonction du débit.

-C- Résultats obtenus par simulation

Le nombre de disques est calculé de façon à avoir un flux de sortie supérieur ou égal au flux d'entrée: $NbDisques = (NbUtilisateurs \times DebitMoy \times TaccDiskMoy)/(TailleFragment) = 9$ avec NbUtilisateurs=10, DebitMoy=7.5 Mbits/s, TaccDiskMoy=30 ms et TailleFragment=32767x8.

La variation du nombre de fragments présents dans les fifos internes des fonctions Lecture au cours de la simulation est la suivante.



-Figure 7.13- Nombres de fragments moyen et total dans l'ensemble des buffers.

La valeur moyenne du nombre de fragments dans les fifos internes pour les fonctions Lecture est de l'ordre de 3,5. La valeur plus élevée se justifie par le coté plus réaliste d'implantation des blocs ou des disques consécutifs (Round-Robin). La simulation a bien montré l'effet de coïncidences défavorables (accès aux mêmes disques ou temps d'accès consécutifs à des disques plus élevés que la valeur moyenne) entraînant des ruptures de séquences.

La taille moyenne du buffer d'anticipation nécessaire à la fonction Lecture pour éviter toute rupture de séquences est donc 4. Il s'agit d'une valeur moyenne obtenue en considérant un débit moyen de 7,5 Mbits/s par utilisateur, et il ne faut pas oublier que pour des séquences de haut débit la valeur de A est supérieur à 4 (5 à 7). Comme la taille des buffers d'anticipation doit être dynamique, l'ensemble des buffers d'anticipation doit être implanté non pas sous la forme d'un ensemble disjoint de fifos mais dans une zone mémoire commune dont la dimension sera calculée avec une valeur moyenne de A de 4. Ce résultat est important pour la phase de conception architecturale qui est présentée dans [CALVEZ-97a].

7.1.6 Modélisation des performances du serveur avec des processeurs

Une simulation qui consiste à faire intervenir la notion de structure d'exécution pour le dimensionnement de la solution matérielle est bien moins évidente. Le but d'une telle simulation est d'étudier l'influence du degré de parallélisme (nombre de processeurs) et de la puissance de traitement de chaque processeur Exécution comme support d'implantation de toutes les fonctions Enregistrement et Lecture. A ce stade, on peut parler de co-simulation parce que le logiciel est simulé en même temps que l'architecture matérielle.

-A- Modèle et résultats de simulation

La structure fonctionnelle simulée est représentée par la figure 7.9.

Le processeur Supervision supporte toutes les fonctions de dialogue. Le processeur Exécution supporte toutes les fonctions critiques Lecture et Enregistrement. Les disques sont des fonctions avec leur propre ressource d'exécution.

Les attributs caractérisant chaque processeur sont au départ:

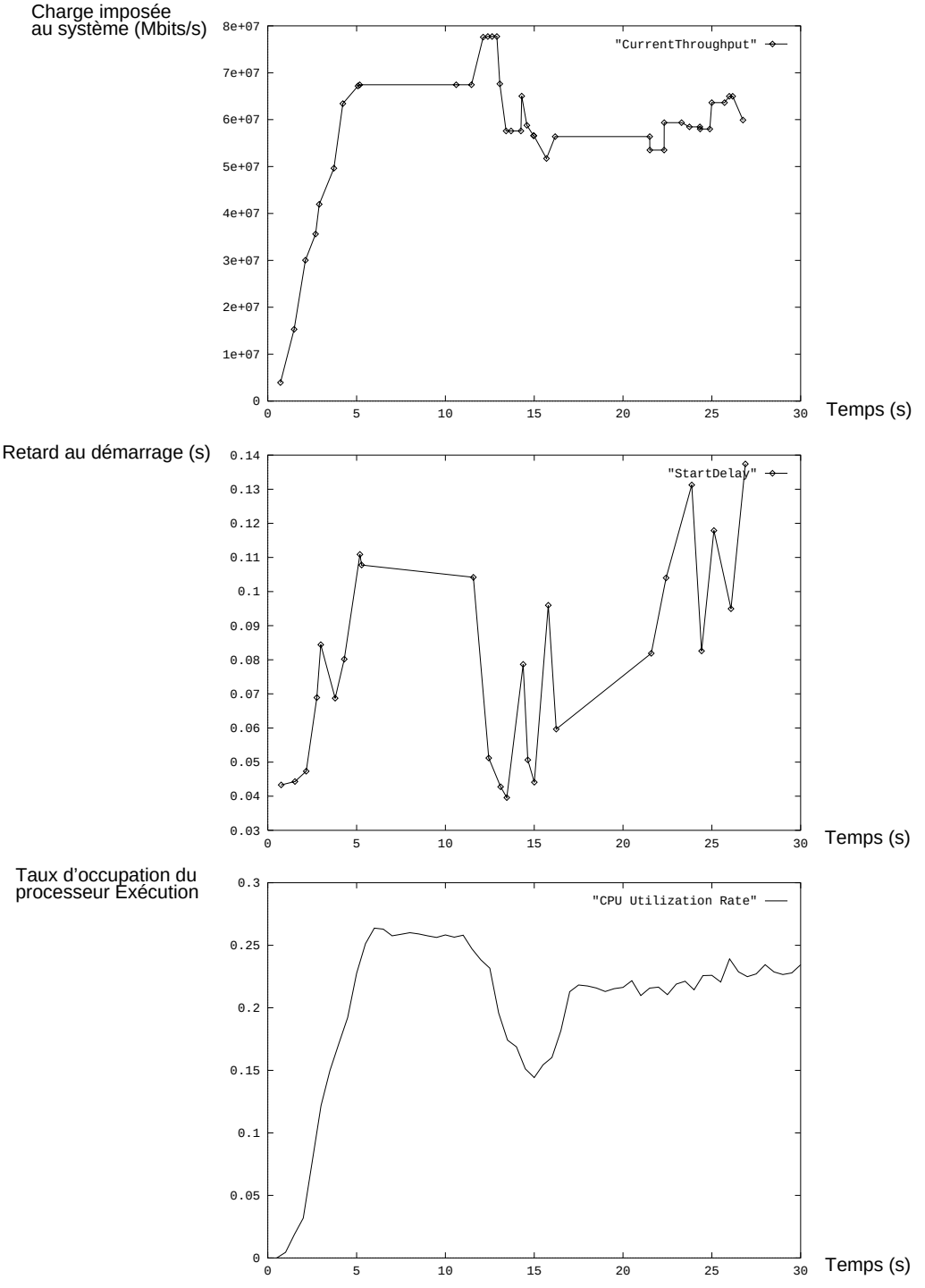
- pour Supervision: 'Power=1, 'Concurrency=1 (à faire varier), 'Policy=PSP,
- pour Exécution: 'Power=1, 'Concurrency=m (à faire varier), 'Policy=PSP.

PSP (Priority Scheduling Policy) veut dire le choix d'un ordonnancement des fonctions selon leur priorité. 'Power représente la puissance relative du processeur, 'Concurrency représente le nombre de fonctions exécutables simultanément ce qui simule en fait le nombre de processeurs physiques.

La simulation du modèle a permis par exemple d'estimer avec 10 utilisateurs et pour un scénario de charge donné:

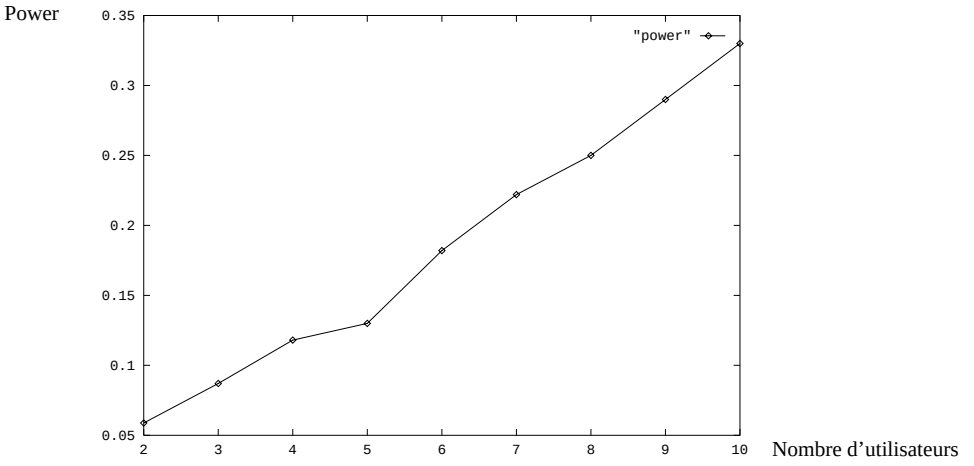
- le retard de démarrage d'une séquence (temps entre la demande faite par un utilisateur et le premier fragment émis),
- le taux d'occupation du processeur d'exécution.

Ces résultats sont représentés par les courbes de la figure 7.14.



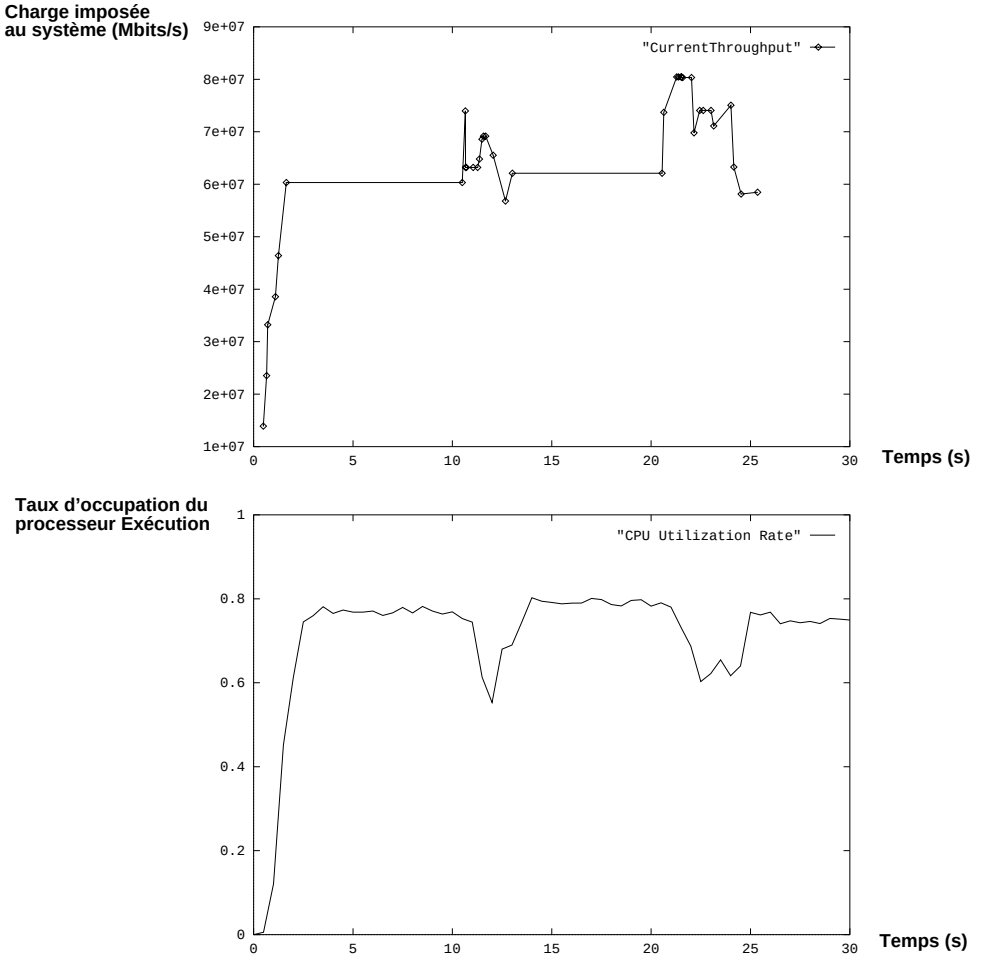
-Figure 7.14- Scénario de charge, retard au démarrage et taux d'occupation du processeur.

Comme le taux d'occupation du processeur d'exécution est faible (25%), nous nous sommes intéressés à l'influence du nombre d'utilisateurs sur l'attribut 'Power du processeur d'exécution. Pour qu'il n'y ait pas de rupture de séquences en fonction du nombre d'utilisateurs, la valeur minimale de l'attribut 'Power ('Concurrency=1) est donnée par la courbe suivante.



-Figure 7.15- Influence du nombre d'utilisateurs sur l'attribut 'Power.

Avec comme caractéristiques du processeur d'exécution 'Concurrency=1 et 'Power=0.33, le taux d'occupation du processeur pour une charge donnée est de l'ordre de 80%.



-Figure 7.16- Charge imposée au système et taux d'occupation du cpu.

Avec les temps choisis pour les opérations élémentaires du modèle de performance de la fonction Lecture, on peut seulement conclure qu'un seul processeur d'exécution peut gérer 10

séquences simultanées ayant un débit moyen de 7,5 Mbits/s, soit un débit de 75 Mbits/s. Il est dommage que les limitations mémoire de l'ordinateur utilisé pour la simulation et les temps de simulation qui sont relativement longs (environ 3 heures de simulation) ne nous ont pas permis d'étudier plus précisément l'influence du nombre d'utilisateurs sur les attributs 'Concurrency et 'Power du processeur d'exécution.

7.1.7 Bilan sur l'évaluation des performances du serveur vidéo

Sur la base de cet exemple, nous pouvons affirmer que l'évaluation des performances dynamiques d'un système par co-simulation n'est pas limitée par la complexité du système et permet d'extraire un ensemble de résultats de performances plus riche que les approches analytiques tels que le débit sur un bus, le taux d'occupation d'une ressource (processeurs logiciels, disques, bus,...), le temps de latence d'un message, la détection du non respect d'une contrainte temporelle, un temps de retard au démarrage, etc. Cependant, la co-simulation souffre d'un temps de simulation trop long. Pour réduire les temps de simulation, il faut augmenter le niveau d'abstraction des modèles d'où l'intérêt d'un modèle de performance mascroscopique et non interprété: les entrées et les données internes influencent le comportement du système uniquement par l'intermédiaire d'attributs. Par exemple, les attributs Size (taille) et Id (destinataire) d'un message remplace le contenu du message.

L'analyse des performances qui a été effectuée lors de l'étape de conception fonctionnelle ou conception préliminaire aide au dimensionnement des éléments internes du système. Nous avons pu montrer l'efficacité du modèle en considérant particulièrement les 2 niveaux de modélisation considérés pour les disques: une seule ressource avec un degré de concurrence, autant de disques avec chacun un modèle aléatoire représentatif de son comportement. La modélisation des performances faite pour les 2 niveaux d'abstraction du serveur (fonctionnel et exécutif) a permis de déduire et confirmer des paramètres importants que nous ne pouvions pas déterminer par la modélisation statique. Les résultats apparaissent similaires avec les 2 modélisations. Nous avons la confirmation que la valeur de A moyen est de 4 (donc 4 places par séquence), ce qui permet de calculer la taille de la mémoire interne nécessaire. Au niveau exécutif, il est possible d'observer l'influence de la politique d'ordonnancement (attribut 'Policy) et de la puissance du processeur (attribut 'Power). La politique d'ordonnancement n'apparaît pas significative. Peut-être ceci est lié à la limitation à 10 utilisateurs.

Dans [CALVEZ-97a], cet exemple a également servi à montrer l'importance de l'approche système et d'un partitionnement en deux temps (système et matériel/logiciel) pour déterminer les parties du système relevant de l'activité de co-design. Ces parties concernent ici l'interfaçage de la mémoire centrale implantant les buffers d'anticipation de taille dynamique avec les disques et les cartes contrôleur ATM.

7.2 SYSTEME DE COMMUNICATION

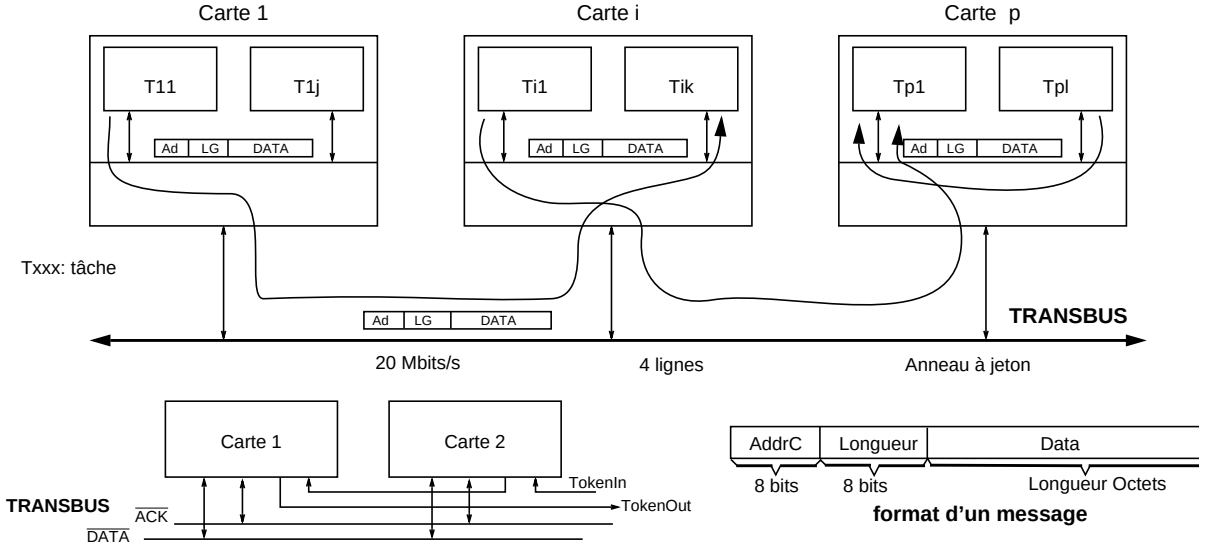
Le second exemple présenté ici a servi d'exemple d'illustration de la démarche de co-design associée à la méthodologie MCSE. Il a été détaillé dans différentes publications [CALVEZ-94] [CALVEZ-96c]. Il a également servi à la présentation du modèle de performance et du principe d'évaluation des performances dans [CALVEZ-96b].

7.2.1 Présentation de l'exemple

Il s'agit d'un système de communication basé sur un ensemble de cartes identiques inter-connectées par un bus série du type anneau à jeton appelé Transbus et détaillé dans [CALVEZ-96c]. Sur chaque carte, les producteurs doivent envoyer de courts messages (256 octets maximum) à des consommateurs situés sur la même carte ou sur une autre carte. Les producteurs et consommateurs sont implantés par des tâches logicielles. Les spécifications du système et du bus sont représentées par la figure 7.17. Le bus au débit maximum de 20 Mbits/s se compose de 4 lignes:

- une ligne (Data) de transport des données dont le protocole de communication au niveau bit est similaire à celui utilisé par les liens séries d'un transputer (T800) excepté pour l'acquittement,
- une ligne (Ack) utilisée pour l'acquittement de chaque octet transmis,
- deux lignes (TokenIn et TokenOut) pour gérer l'accès au bus selon le principe des bus du type anneau à jeton.

Chaque message transmis contient l'adresse du destinataire, la longueur de l'information transmise et l'information. A tout moment, seule la carte qui possède le jeton peut envoyer un message et/ou passer le jeton à sa voisine.



-Figure 7.17- Spécification du système de communication.

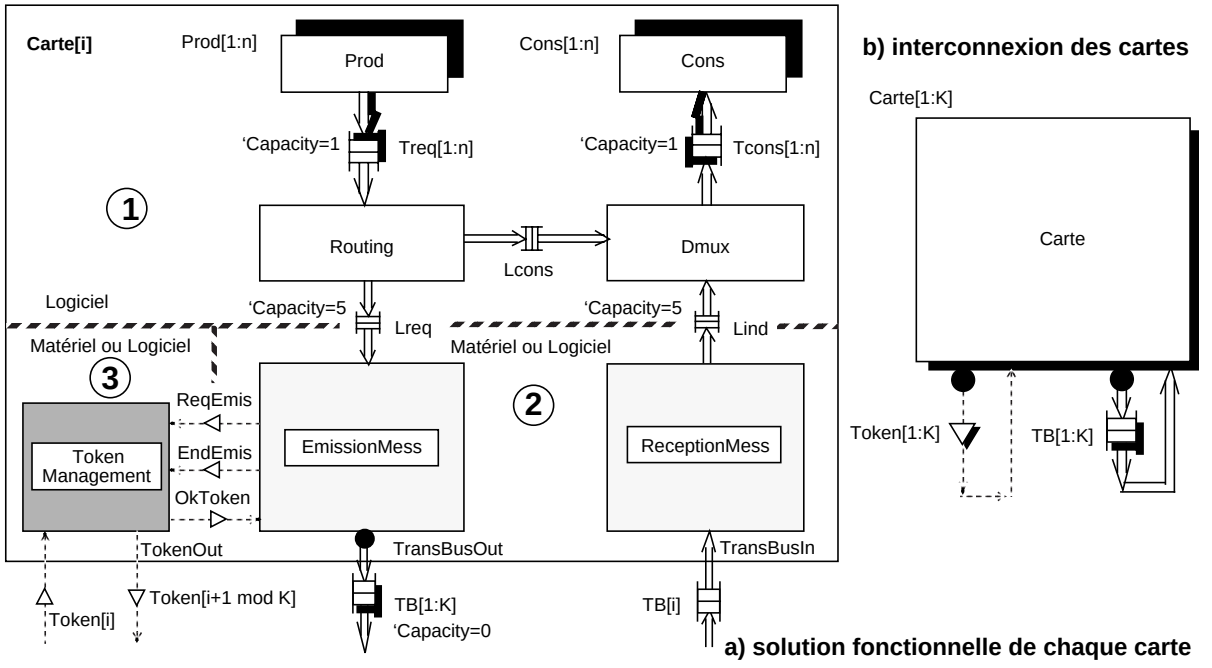
7.2.2 Objectif de l'évaluation des performances pour cet exemple

L'objectif du concepteur du système est de concevoir et implanter correctement une carte en tenant compte des contraintes de performances. Ce système peut être implanté selon une architecture générique basée sur un microprocesseur, un FPGA et une mémoire commune. Il est facile d'extraire dans cet exemple la partie minimale à implanter en matériel pour satisfaire le débit du bus et le protocole de communication au niveau bit imposé par Transbus [CALVEZ-96c]. Dans la suite, nous supposerons l'existence de cette partie matérielle qui implante l'interface bas niveau du bus. Elle comporte une fonction de conversion parallèle/série pour la transmission de chaque octet et la fonction inverse pour la réception bit à bit d'un octet.

Nous nous intéressons donc au système en considérant uniquement une transmission au niveau octet et non bit. Après avoir défini la structure fonctionnelle de la solution et le comportement de chaque fonction, nous étudions l'influence de plusieurs solutions de partitionnement matériel/logiciel sur les performances du système. Les critères de performances retenus ici sont à la fois internes (taux d'occupation du processeur) et externes (débit sur le bus, temps de latence des messages). Ces résultats sont obtenus par une co-simulation du matériel et du logiciel à un niveau macroscopique et non-interprété mais suffisamment détaillé pour observer rapidement des résultats intéressants.

7.2.3 Le modèle fonctionnel

Pour concevoir le système de communication, il est nécessaire de prendre en compte la décomposition du système en un ensemble de cartes et le bus d'interconnexion (répartition géographique de l'application). Ce travail correspond aux deux premières étapes de la méthodologie MCSE: l'étape de spécification et l'étape de conception fonctionnelle. La structure fonctionnelle obtenue est représentée sur la figure 7.18. Le bus transbus y est modélisé sous la forme d'un vecteur d'événements Token[1:k] représentant la circulation du jeton et un vecteur de ports de communication TB[1:k] pour le transfert de messages entre cartes.



-Figure 7.18- Structure fonctionnelle détaillée d'une carte du système de communication.

Chaque message produit est envoyé par un producteur Prod[i] à la fonction Routing via le port Treq[i]. Le champ d'adresse du message est utilisé par la fonction Routing pour déterminer si le destinataire du message est local (situé sur la même carte) ou distant.

Pour une communication à distance, la fonction EmissionMess envoie chaque message reçu de Lreq à la carte concernée via le port TB. Comme seulement une et une seule carte peut accéder au bus à un moment donné, EmissionMess commence par faire une requête du jeton (événement ReqEmis), attend le jeton (événement OkToken) et envoie le message octet par octet dans TB[]. Lorsque l'émission du message est terminée, EmissionMess relache le jeton

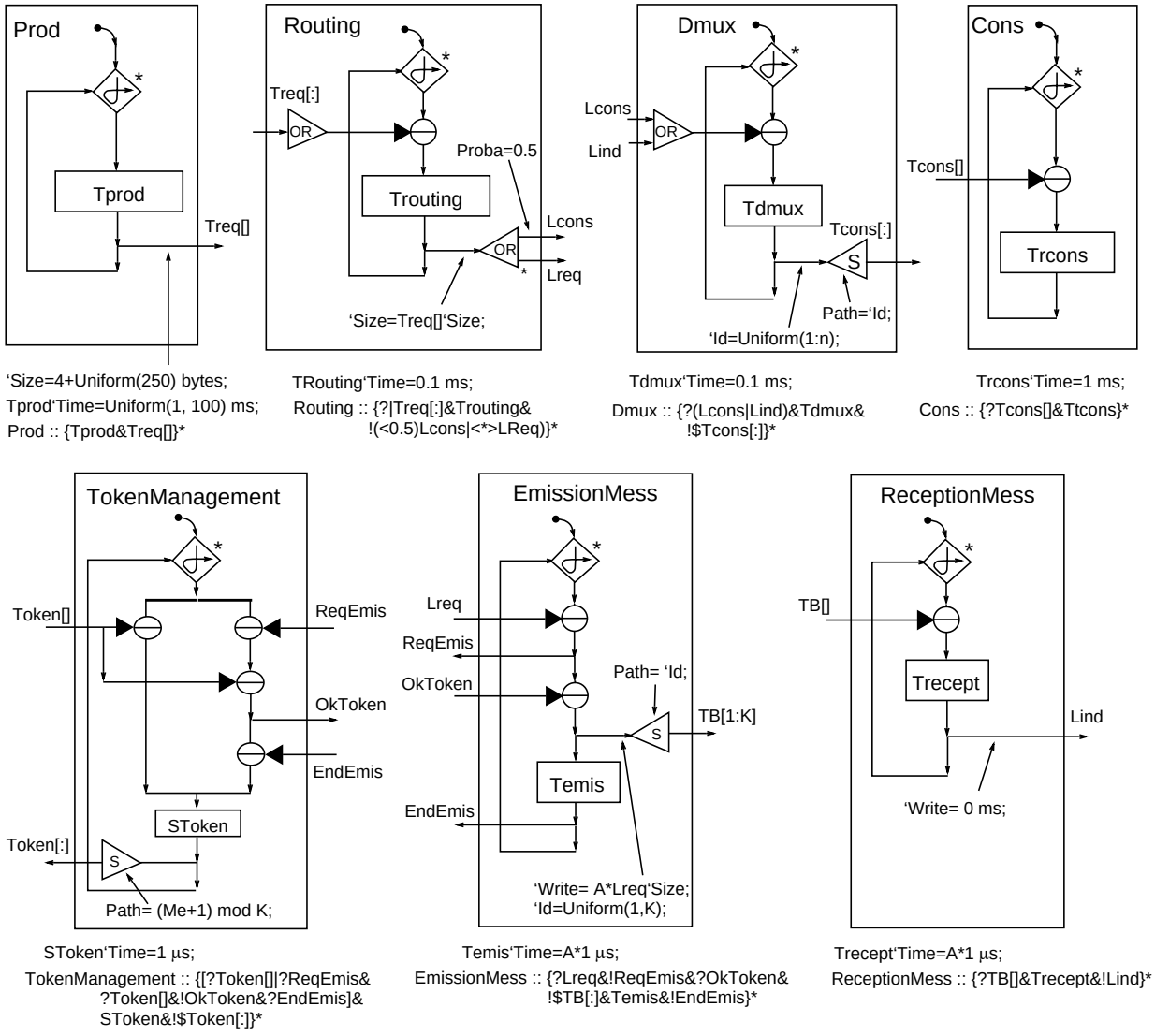
(événement EndEmis) qui est envoyé par la fonction TokenManagement à la carte voisine (événement Token[i+1MOD k])

La fonction ReceptionMess reçoit chaque message addressé à la carte et l'envoie à la fonction Dmux via le port Lind. Dmux envoie le message au consommateur concerné.

Sur la figure 7.18-b, le rond noir sert à spécifier que chaque carte est capable d'envoyer un message à tous les éléments du vecteur TB et de générer un événement pour chaque élément du vecteur Token.

7.2.4 Le modèle de comportement

Le modèle de comportement de chaque fonction est donné par la figure 7.19. Pour spécifier complètement le comportement, des attributs sont ajoutés au modèle graphique. Pour comprendre les notations utilisées, il est utile de préciser que Treq[] signifie que l'indice du port est le même que celui du producteur et que Treq[:] signifie que l'on considère le vecteur complet.



-Figure 7.19- Modèle de comportement non interprété de chaque fonction.

Les producteurs et consommateurs sont des process cycliques très simples. La taille des messages produits est tirée aléatoirement entre 4 et 255. L'intervalle de temps entre l'émission successive de deux messages est défini par le temps d'exécution de l'opération élémentaire Tprod. C'est une façon très simple de spécifier la charge imposée au système pour la simulation.

La fonction Routing reçoit les messages de tous les ports Treq[1:n] et simule le routage d'une partie de ces messages vers le port Lcons (messages locaux). l'attribut 'Proba est utilisé pour spécifier la sortie sélectionnée. Chaque message généré comporte l'attribut 'Size du message d'entrée. Dmux reçoit des messages des ports Lcons et Lind et les envoie vers le consommateur désigné par l'attribut 'Id tiré aléatoirement entre 1 et n.

La fonction TokenManagement a la charge d'allouer le bus à une seule carte. Une attente conditionnelle est utilisée pour attendre l'occurrence du jeton (Token) provenant de la carte voisine ou bien la requête du jeton par la fonction EmissionMess (ReqEmis). Si il n'y a pas de demande d'émission, le jeton est transmis directement à la carte voisine.

La fonction EmissionMess reçoit un message de Lreq et demande le jeton. La transmission de chaque message sur le bus transbus est modélisée par l'envoi du message sur un des éléments du port TB dont l'index est défini par l'attribut 'Id (valeur aléatoire entre 1 et K). Le temps de transmission est spécifié par l'attribut 'Write et dépend de la longueur du message et d'un paramètre A qui représente le temps de transmission de chaque octet. La fonction ReceptionMess est un process cyclique qui attend un message du port TB et l'envoie vers le port Lind.

7.2.5 Co-simulation et résultats

L'objectif de cette co-simulation est de montrer que le modèle de performance permet d'évaluer les performances globales de différentes implantations de la solution fonctionnelle d'un système. Pour notre exemple, nous avons considéré 3 implantations différentes. Sur la figure 7.18-a, la description fonctionnelle est décomposée en 3 zones: la zone 1 comporte les fonctions qui sont obligatoirement implantées en logiciel, la zone 2 inclut les fonctions de transmission et réception sur le bus Transbus, et la zone 3 est constituée uniquement de la fonction de gestion du jeton.

La co-simulation a pour but d'étudier la modification des performances si les zones 2 et 3 sont implantées en matériel ou en logiciel.

A ce stade, il est également important de bien définir la charge imposée au système et les critères de performances retenus pour décider laquelle des implantations est la meilleure.

En ce qui concerne la charge imposée au système, les producteurs envoient en permanence (Tprod'Time=0) des messages de taille aléatoire sur des cartes distantes (Proba=0) d'adresse aléatoire. Un consommateur est également supposé dépenser au moins 1 ms pour exploiter chaque message reçu. L'asservissement des producteurs aux consommateurs dépend de la capacité de chaque port. Nous avons choisi une capacité de 1 pour les ports Treq et Tcons et de 5 pour les ports Lreq et Lind. Enfin, le comportement correct du bus modélisé avec le port TB est obtenu avec un mécanisme de rendez-vous ('Capacity=0) entre l'émetteur EmissionMess et le récepteur ReceptionMess.

Les critères de performances retenus sont le taux d'occupation du processeur exécutant le logiciel de chaque carte, le débit sur le bus et le temps de latence des messages. A cause du caractère aléatoire du modèle, ces trois résultats sont évalués par une moyenne des résultats de

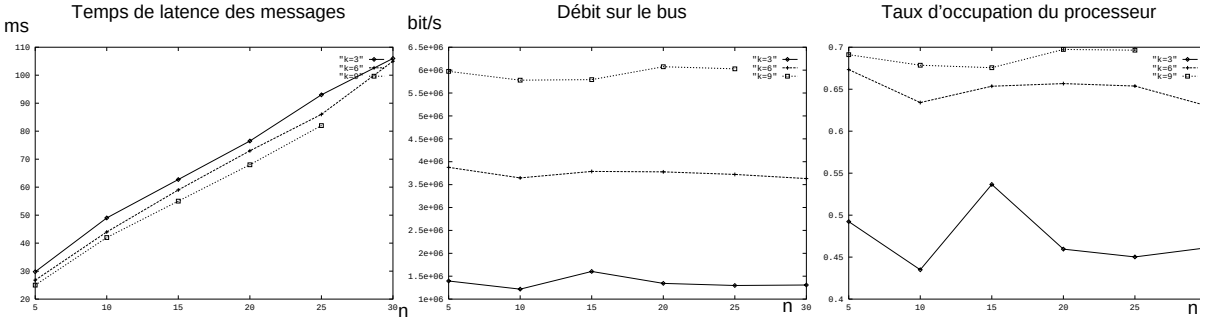
toutes les cartes et lorsque le système a atteint un état stationnaire (la durée de l'état transitoire observée par simulation est d'environ 0,1 s). Nous avons également fait varier le nombre de cartes (paramètre générique k) et le nombre de producteurs et consommateurs par carte (paramètre générique n).

-A- Maximum en matériel (zone 2 et 3 en matériel)

Pour obtenir des résultats appropriés, il est nécessaire de connaître le temps de transfert d'un octet sur le bus quand les fonctions EmissionMess et ReceptionMess sont implantées en matériel. Cette valeur provient de l'étude présentée dans [CALVEZ-94]. Pour obtenir cette valeur, on peut aussi considérer le protocole de communication de transbus au niveau bit:
11 bits x 50 ns = 0.7 µs (8 bits de données et 2 bits de start et 1 bit de stop émis à 20 Mbits/s). Nous avons donc choisi comme valeur du paramètre A=0.7 µs. Le temps de passage du jeton SToken'Time de la fonction TokenManagement est fixé à 1 µs.

Toutes les fonctions logicielles de la carte sont implantées sur le même processeur. On rajoute pour cela une fonction nommée Processeur et qui englobe les fonctions logicielles de la carte. Cette fonction simule une ressource ayant un degré de concurrence de 1, ce qui signifie qu'une seule fonction peut être active à un moment donné. Des attributs caractérisent cette fonction: son degré de concurrence (attribut 'Concurrency) et sa puissance (attribut 'Power). L'attribut 'Power est très utile pour modifier la vitesse d'exécution de toutes les tâches logicielles et d'étudier son influence sur les performances globales du système. Les attributs 'Policy et 'Priority permettent également de définir la politique d'ordonnancement des tâches. Ici, la priorité la plus élevée est pour Dmux, puis Routing, Cons et enfin Prod.

Les résultats de la co-simulation sont données par la figure 7.20.



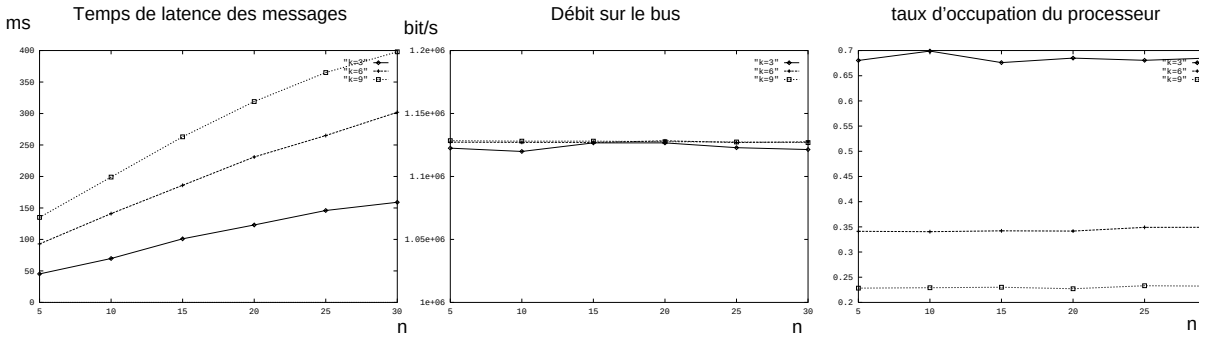
-Figure 7.20- Résultats pour le maximum de matériel.

Le temps de latence des messages croit avec le nombre n de producteurs et consommateurs car ils se partagent tous le même processeur. Le débit sur le bus dépend uniquement du paramètre k qui représente le nombre de cartes. La constance du débit par rapport au paramètre n est due au fait que le temps de transmission d'un message qui est en moyenne de 90 µs (129x0.7) est négligeable par rapport à son temps de consommation (1 ms). Par conséquent; le port Lind est rapidement saturé et après avoir obtenu le jeton, la fonction EmissionMess doit attendre jusqu'à ce que ReceptionMess soit prête à recevoir un message. Le taux d'occupation du processeur est relativement faible pour K=3 car comme nous le verrons plus loin le bus est mal utilisé.

-B- Transmission en logiciel (zone 1 et 2 en logiciel)

Pour une implantation logicielle de EmissionMess et ReceptionMess la valeur du paramètre de vitesse de transmission sur le bus est fixée à A=7 μs. Les deux nouvelles fonctions logicielles sont rajoutées à la fonction Processeur avec la priorité la plus élevée pour ReceptionMess puis pour EmissionMess.

Les résultats obtenus sont donnés par la figure 7.21.



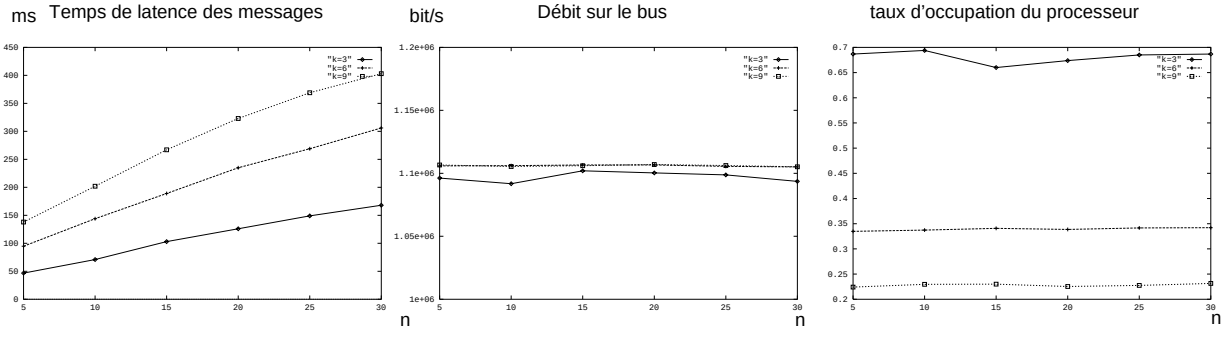
-Figure 7.21- Résultats pour la transmission implantée en logiciel.

Le temps de latence des messages croit en fonction du nombre n de producteurs/consommateurs et du nombre k de cartes. Le débit sur le bus est indépendant des paramètres n et k. Cette valeur constante est due au temps de transmission de chaque message: 129*8/(129x7 μs) = 1.15 Mbits/s. Le bus est donc le goulot d'étranglement du système. Le taux d'occupation du processeur est élevé pour k=3 et relativement faible pour k=6 et 9 car le bus limite le nombre de messages transmis.

-C- Tout en logiciel

La fonction TokenManagement est rajoutée à la fonction Processeur avec la priorité la plus élevée. Le temps d'exécution choisi pour l'opération élémentaire SToken est de 20 μs ce qui correspond au temps nécessaire au processeur pour recevoir et traiter une interruption.

Les résultats sont donnés par la figure 7.22.



-Figure 7.22- Résultats pour une implantation logicielle de toutes les fonctions du système.

Le débit sur le bus est encore constant et un peu plus faible que précédemment car 20 μs ont été rajoutées entre deux émissions successives de messages. Le temps de latence des messages et le taux d'occupation du processeur restent similaires.

7.2.6 Bilan sur l'évaluation des performances du système de communication

Dans le cas A (zone 2 et 3 en matériel), le débit maximum du bus égal à 11.4 Mbits/s (8/0.7μs) n'est pas atteint. Ceci est dû au fait que le protocole de communication au niveau message n'est pas correct. En effet, la fonction EmissionMess est capable d'obtenir le bus pour envoyer un message à une carte qui n'a plus de place libre dans le port Lind. Pour utiliser le bus à son débit maximum, le protocole de communication doit être modifié pour envoyer un message si et seulement si il existe une place libre pour recevoir le message. Cette modification du protocole est présenté dans [CALVEZ-96c].

Pour un système composé 2 ou 3 cartes, le temps de latence des messages est sensiblement le même pour les trois implantations. Dans ce cas, une implantation matérielle au coût économique plus élevé ne se justifie pas.

Dans cet exemple, tous les résultats sont très dépendants du temps d'exécution de chaque consommateur. La capacité de chaque port et la politique d'ordonnancement des tâches jouent également un rôle important.

Avec cet exemple, nous avons montré que notre modèle autorise la co-simulation macroscopique et non-interprétée des fonctions logicielles et des composants matériels. Il permet ainsi au concepteur de trouver par une démarche itérative le partitionnement et l'allocation optimale vis à vis des contraintes de performances dynamiques qui lui sont imposées. L'utilisation de paramètres génériques (paramètres n et K par exemple) associés aux attributs des éléments du modèle de performance permet aussi de parcourir un espace assez vaste des solutions possibles d'un partitionnement sans nécessiter de mise à jour du modèle de performance et de nouvelle génération du code VHDL équivalent.

7.3 CONCLUSION

Ces deux exemples ont permis de montrer que le modèle de performance est un outil approprié pour aider au dimensionnement et au partitionnement des systèmes matériels et logiciels. En particulier, il faut constater l'aptitude du modèle à représenter des systèmes ayant une structure générique (vecteurs de ports, de fonctions, d'activités). Le concept d'attributs associés aux objets permet aisément de définir un comportement. Des lois déterministes et aléatoires servent alors à produire des valeurs pour ces attributs pour placer le modèle dans une variété de situations réalistes. De plus, le concept de messages avec des attributs associés permet de propager dynamiquement un comportement dans la structure.

Ces deux exemples qui regroupent la plupart des constructions du modèle de performance nous ont également permis de valider les règles de transcription du modèle de performance en VHDL et l'implantation de ces règles dans un générateur de code automatique.

L'exemple du système de communication a également permis de tester la transcription du modèle de performance en C++/Kernel Windows NT. Il a permis de constater que l'exécution du programme C++ obtenu est environ 4 fois plus rapide que la simulation VHDL équivalente. Evidemment, le programme C++ seul ne permet pas de faire une analyse détaillée du comportement du système (chronogrammes du simulateur VHDL). Mais il peut être couplé directement à l'outil d'analyse de trace qui permet alors de faire du suivi du contenu de variable, la visualisation d'un graphe de déroulement temporel (time line) et la définition des

nombreux indices de performances. Le simulateur VHDL a l'avantage d'être disponible sous diverses plate-formes (Unix, Pc). Cependant, le meilleur compromis entre portabilité et temps de simulation est probablement de choisir comme langage cible JAVA au lieu de VHDL et C++.

Ces exemples ont également permis de mettre en évidence l'importance de l'extraction et de l'interprétation des résultats. Il semble judicieux de laisser le choix à l'utilisateur entre une génération automatique de trace et l'utilisation d'une librairie de composants de monitoring mise à la disposition du concepteur.

Pour une évaluation des performances, la procédure itérative à suivre par les concepteurs se décompose donc en 3 phases:

- modélisation incrémentale du système ou de l'application,
- évaluation des performances souhaitées,
- visualisation des résultats et interprétation.

La troisième phase conduit généralement à un retour sur la première phase pour modifier, améliorer, poursuivre la décomposition du modèle, ou à un retour sur la deuxième phase pour modifier la sélection des résultats demandés ou des paramètres de l'évaluation.