Impact de choix d'implantation sur les performances d'une application de Contrôle-Commande

Fabrice Jumel[*] — **Nicolas Navet**[**] — **Françoise Simonot-Lion**[**]

[*] *CITI - INSA*
20, Avenue Albert Einstein, F69621 Villeurbanne Cedex
fabrice.jumel@insa-lyon.fr
[**] *LORIA - INPL*
Campus Scientifique - BP 239, F54506 Vandœuvre-lès-Nancy
{nnavet,simonot}@loria.fr

RÉSUMÉ. *Nous étudions l'évaluation de l'impact des choix d'implantation d'un système informatisé de contrôle-commande sur la qualité du système contrôlé. Nous plaçons cette étude dans le cadre des systèmes échantillonnés. Nous présentons d'abord comment modéliser l'implantation d'une application de contrôle-commande sur une architecture informatique et comment construire un modèle du système contrôlé et de son système de commande. Les techniques d'évaluation ciblées reposent sur la simulation de ces modèles. Nous appliquons nos principes sur une étude de cas et identifions les résultats qui peuvent être obtenus et les renseignements fournis pour concevoir une implantation du système de contrôle-commande assurant des propriétés de stabilité. Nous montrons que dans le cas où plusieurs algorithmes de régulation partagent le même processeur, une politique à priorités fixes dégrade la qualité des régulations par rapport à une politique où les priorités seraient alternées.*

ABSTRACT. *We study how to evaluate the impact of the implementation choices of a computerized control systems on the quality of the controlled system. We focus on sampled systems. First, we show how to model the implementation of a control application and how to build a model of the controlled system with its controller. The evaluation is done by simulation. The methodology is applied on a case study; the results that can be obtained are identified as well as the guidelines they provide to conceive an implementation that ensures some stability properties. We show that when several control laws share the same processor, the Fixed Priority Pre-emptive policy performs poorly in comparison to a policy where priority levels would be alternated.*

MOTS-CLÉS : *systèmes échantillonnés, systèmes à retards, évaluation de performances, simulation, ordonnancement.*

KEYWORDS: *digital control, systems with delays, performance evaluation, simulation, scheduling.*

RSTI - TSI – 25/2006. Contrôle temporisé, pages 377 à 404

1. Introduction

Dans cet article nous proposons des techniques, reposant sur la simulation, d'évaluation quantitative de l'impact des choix d'implantation d'un système numérique de contrôle-commande sur la qualité d'un système contrôlé. Nous plaçons plus particulièrement cette étude dans le cadre des systèmes échantillonnés. Les performances et les services de l'architecture informatique support ont une influence qui va de la dégradation de la qualité d'une régulation à la perte de la contrôlabilité du système (c'est-à-dire à l'instabilité du système contrôlé). Si l'on s'attache particulièrement aux critères de stabilité, ceux-ci dépendent de l'évolution de la consigne et du niveau de perturbation que la loi de commande doit tolérer afin que l'évolution du système ne soit pas perturbée.

L'implantation d'un système numérique de contrôle-commande engendre des retards et des gigues sur des événements significatifs et peuvent également créer des écarts de valeurs par rapport au cas idéal sur les informations transmises (écrasement d'une donnée dans un tampon). La raison est, d'une part, le temps pris par l'exécution des algorithmes et la transmission des données sur les réseaux et d'autre part, les temps d'attente induits par les politiques d'ordonnancement sur les processeurs et les protocoles d'accès aux réseaux. Pour comprendre le phénomène de retard, il suffit d'observer, au niveau des actionneurs, une commande appliquée sur le système contrôlé. Cette commande est obtenue à partir d'une représentation de l'état du système. Cet état peut être une représentation exacte du système, dans le cas de commande par retour d'état, ou approchée, par exemple dans le cas des commandes PID (*Proportional Integrator Derivator*). L'existence de retards ou de gigues sur les événements de production/consommation des informations manipulées dans le système de contrôle-commande entraîne une représentation incohérente du système au niveau des algorithmes de commande. Dans le cas d'un retard, l'état considéré correspond à un état passé du système. Pendant ce laps de temps, que l'on qualifie de « temps mort » en automatique, le système a évolué et la commande n'est donc pas forcément adaptée à l'état actuel du système. De la même manière, l'existence d'une gigue sur les dates d'acquisition des grandeurs peut entraîner une incohérence similaire. En effet, la définition de la loi de commande repose sur des calculs qui peuvent être assimilés à des calculs approchés de dérivées ou d'intégrales. Il est donc important que les intervalles de temps qui séparent les acquisitions soient conformes aux hypothèses qui sont sous-jacentes aux approximations faites sans quoi les résultats des calculs sont incohérents. Par exemple, il est nécessaire que le temps entre deux acquisitions corresponde à celui qui sert dans la fonction d'approximation d'une dérivée.

Sous de fortes hypothèses, comme des retards multiples de la période d'échantillonnage, la mesure de l'impact d'une implantation sur la qualité d'un système par une analyse mathématique est possible (cf. section 2.2 et, pour plus de détails, (Jumel, 2003)). Lorsque ces hypothèses ne sont plus garanties, une solution est d'appliquer des techniques de simulation pour mesurer cet impact. Dans ce contexte, nous avons développé une méthode de modélisation et de simulation qui repose sur l'atelier Matlab/Simulink (Matlab, 2004). Matlab/Simulink est un outil très largement utilisé

dans la communauté de l'automatique ; il offre un éditeur graphique de modèles de systèmes continus et échantillonnés ainsi qu'une machine d'exécution permettant la simulation de ces modèles.

Après avoir introduit la problématique de l'étude en section 2, nous donnons les principes généraux d'une modélisation Matlab/Simulink intégrant les performances de l'architecture informatique support du système de contrôle-commande dans la section 3. La section 4 présente des modèles d'architecture classiquement utilisés en automatique tandis que dans la section 6, nous montrons quels types de résultats il est possible d'obtenir sur une étude de cas introduite en section 5.

2. Problématique

En automatique, un système se caractérise par des grandeurs qui peuvent être d'entrée, d'état ou de sortie. Sous l'effet des grandeurs d'entrée, le système se modifie : une partie de ses caractéristiques internes varient (grandeurs d'état) et ses valeurs externes surveillées à l'aide de capteurs changent (par exemple, la vitesse d'un véhicule). Parmi les grandeurs d'entrée, une partie est maîtrisée ou maîtrisable. En particulier, certaines sont volontairement modifiées afin d'obtenir un nouvel état du système : on parle alors de *consignes* appliquées au système. Les autres grandeurs d'entrée, ou *perturbations*, modifient le système par des effets indésirables qui ne sont pas connus de manière déterministe et sur lesquelles on ne peut pas agir. Les grandeurs de sortie sont les *réponses* du système aux entrées. Elles peuvent faire ou non l'objet d'une mesure, d'une surveillance ou d'un contrôle : on parle alors de *régulation* ou d'*asservissement*.

La *régulation* consiste à maintenir un certain état du système soumis à l'évolution de l'environnement extérieur et l'*asservissement* à fournir au système la capacité de suivre au mieux les variations de la consigne qui lui est appliquée. Les systèmes sont généralement à la fois régulés et asservis. Le système que l'on régule est alors appelé *système réglé* (ou procédé) et sa régulation est rendue possible par l'introduction d'un autre système, appelé *système réglant*. L'ensemble constitue le *système régulé*. Le but de l'automatique est de fixer les règles et les différentes méthodes pour construire le système régulant (aussi appelé *correcteur*) qui permet d'obtenir le comportement désiré du système *réglé*.

2.1. *Qualité d'un système*

La *qualité d'un système* se définit en caractérisant l'ensemble des comportements admissibles du système réglé : réaction du système face à un bruit, évolution des différentes grandeurs dans le cadre du changement de consigne. Ainsi, un système est dit en équilibre quand il n'y a plus de variation de ses grandeurs d'états (la sortie est alors constante). Un paramètre exprimant la qualité d'un système doit alors représenter sa capacité à atteindre l'équilibre. Plus généralement, dans le cas d'un changement de consigne, ou asservissement, il s'agit de quantifier la stabilité du système, à savoir sa

capacité à atteindre l'équilibre désiré (régime permanent) au bout d'un temps fini. Globalement, pour caractériser la qualité d'un système, deux paramètres sont importants : la précision et le temps de réponse. D'une part, il s'agit de prouver que le système est *stable*, c'est-à-dire de garantir qu'il atteint toujours un équilibre après une perturbation ou un changement de consigne et, donc, d'évaluer le temps mis pour atteindre l'équilibre. D'autre part, il s'agit de mesurer l'écart entre la valeur de sortie atteinte, à partir du point d'équilibre, et celle voulue, appelé *erreur statique* ; ceci permet de calculer la *précision* du système. Ces deux critères sont illustrés sur la figure 1. Dans la suite, nous ne considérons pas la précision qui est, le plus généralement, intrinséquement garantie par la loi de commande.

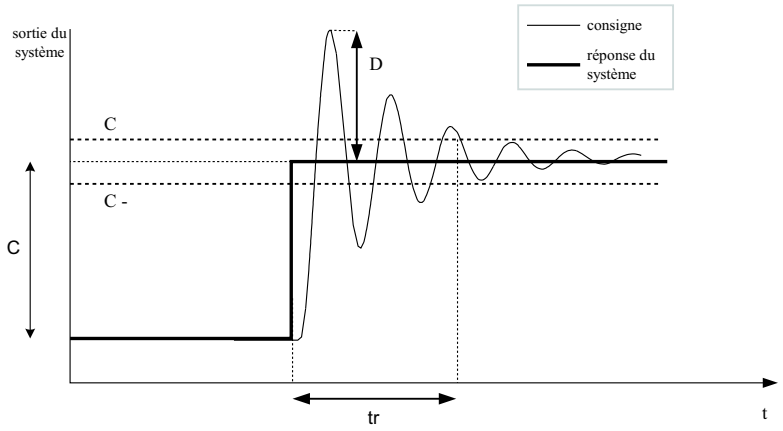

Figure 1. *Illustration des critères utilisés pour évaluer un système lors d'un changement de consigne. Le changement de consigne est noté Δ_c ; le temps de réponse tr_ϵ est le temps nécessaire entre le changement de consigne et la stabilisation à ϵ% du nouvel équilibre désiré et le dépassement D est le maximum des écarts à la consigne attendue*

Plus précisément, dans cet article, nous apprécierons la qualité d'un système régulé par l'évaluation de trois critères qui sont détaillés par la suite : le critère intégral IAE, le temps de réponse et le dépassement.

Rejet de perturbation. Dans le cas d'un système parfait, en absence de changement de consigne, un écart nul entre la sortie du système et la consigne soumise est attendu. Le critère intégral IAE (*Integral of Absolute Error*) permet d'évaluer la distance entre la sortie du système réel et celle qui serait fournie par un système parfait. Ainsi, il est utilisé pour décrire la capacité de régulation du système en absence de changement de consigne. Plus le fonctionnement sous une consigne constante est perturbé, plus l'IAE obtenu sera élevé.

$$IAE = \int_0^T |\varepsilon(t)|dt$$

où $\epsilon(t)$ est l'écart à la consigne attendu à l'instant t. Si l'on ramène l'IAE à la durée de simulation considérée, on obtient une mesure de l'écart moyen vis à vis du régime constant souhaité.

Changement de consigne. Le but de la régulation est que la sortie atteigne la consigne le plus rapidement possible et sans oscillations intempestives. Les critères de performances choisis sont le *temps de réponse* et le *dépassement*. Le temps de réponse est le temps nécessaire entre le changement de consigne et la stabilisation autour du nouvel équilibre désiré. Le dépassement est le plus grand écart constaté à l'équilibre désiré. Ces critères sont généralement contraints par une borne supérieure (pire dépassement et pire temps de réponse admissibles). En termes d'optimisation, la meilleure qualité de l'asservissement est, évidemment, obtenue pour un temps de réponse et un dépassement les plus bas possibles.

Le cas spécial d'asservissement consistant en une suite de consignes rapprochées (par opposition à une consigne par palier avec stabilisation à chaque palier) n'est pas traité dans cet article. Notons néanmoins que la qualité d'une implantation d'un tel asservissement s'évaluerait à l'aide de critères similaires.

2.2. *Les systèmes échantillonnés et leur représentation*

La réalisation des systèmes de contrôle-commande, systèmes *réglants*, est actuellement majoritairement numérique. Ceci signifie qu'ils sont implantés sous la forme d'algorithmes (programmes) et de flux de données distribuées sur une architecture de calculateurs munis de services exécutifs et de réseaux gérés par des protocoles. Les systèmes à contrôler sont, par nature, continus, alors qu'une régulation « informatisée » est discrète. La connaissance des variables de sortie du système réglé par le système réglant ne peut alors se faire qu'à des instants donnés. Il est donc nécessaire de « discrétiser » le système continu, c'est à dire de l'*échantillonner*. On parle, alors, de *systèmes échantillonnés*. L'échantillonnage d'un système consiste à procéder, en général, périodiquement à l'acquisition de l'état du système réglé et à la mise à jour correspondante de ses variables d'entrée.

Cette notion de système échantillonné conduit à plusieurs problèmes qu'il s'agit de maîtriser :

– l'échantillonnage des données a pour conséquence que le système n'est pas contrôlé entre deux instants d'échantillonnage,

– les lois de commande étant réalisées par des algorithmes s'exécutant sur des processeurs, l'architecture de ceux-ci a un impact sur la précision des calculs et, en

conséquence, sur la précision des informations manipulées (entrées, sorties, état) ; il est donc nécessaire d'identifier les erreurs induites sur ces informations,

– le (ou les) échantillons utilisés par une loi de commande ne sont, dans la réalité, pas contemporains de l'activation de la loi en raison de temps de traitement et transmission incontournables. C'est à ce dernier point que nous nous intéressons dans cet article.

L'influence d'une implantation apparaît donc sous forme de *fautes temporelles* et de *fautes de valeurs*. Nous faisons l'hypothèse, dans ce travail, qu'il n'y a pas fautes de valeurs dues à un calcul (les algorithmes sont corrects, la précision des calculs est suffisante). Une faute temporelle signifie que les commandes calculées sont appliquées au mauvais moment et/ou sur des données qui n'ont pas été produites au bon moment. Ces deux phénomènes bien connus des automaticiens s'expriment usuellement sous la forme générale de *retards* et de *gigues* sur l'acquisition.

Pour analyser un système numérique de contrôle-commande obéissant à la théorie des systèmes échantillonnés, il serait logique d'utiliser la représentation en z de ces systèmes et d'y injecter les retards. Cette approche est cependant rarement possible car la représentation en z suppose que les différents systèmes soient parfaitement périodiques. Cela signifie que l'ensemble des échanges de signaux est synchronisé, par exemple, sur le début de la période d'échantillonnage et que l'évolution des systèmes n'est considérée qu'à ces instants. Ainsi l'intégration de retards dans un modèle en z n'est possible que si les retards sont des multiples de la période d'échantillonnage, ce qu'on ne peut pas assurer dans le cas général.

Deux principales approches sont généralement proposées pour modéliser et analyser un système obéissant à la théorie des systèmes échantillonnés de manière à prendre en compte ces retards ou gigues :

– La première repose sur l'utilisation de la transformée en z du système (Ogata, 1987) ; néanmoins, cette technique nécessite que les échanges de signaux soient synchronisés, par exemple, sur le début de la période d'échantillonnage et que l'évolution du système ne soit considérée qu'à ces instants. Ainsi l'intégration de retard dans un modèle en z est alors possible à la condition que les retards soient des multiples de la période d'échantillonnage. Pour ce faire, il suffit d'introduire dans le modèle une fréquence d'observation multiple de la fréquence réelle d'échantillonnage et de faire apparaître les retards comme des multiples de la période d'observation. Il faut alors adapter les traitements de manière à ce qu'ils correspondent bien aux traitements réels définis sur le système et conformes à la fréquence d'échantillonnage spécifiée.

– La deuxième approche propose d'utiliser un modèle continu des systèmes physiques et de modéliser les lois de commandes sous forme de transformées en q (Ogata, 1987), c'est-à-dire sous forme d'un opérateur qui travaille sur les différents échantillons reçus successivement sans besoin de considérer une hypothèse de périodicité sur la production et l'utilisation des échantillons. La contrepartie de cette souplesse de modélisation est que des résultats analytiques sont beaucoup plus difficiles à obtenir, ce qui limite l'utilisation de la transformée en q à des études par simulation.

Une étude comparative de ces deux techniques a été réalisée par Nilsson (Nilsson, 1998). Les conclusions obtenues sont que la première technique permet plus facilement une étude analytique en considérant la notion de discrétisation, tandis que la seconde, faisant apparaître de manière bien distincte la partie physique du système et la partie de contrôle, permet une plus grande liberté de modélisation des retards mais, par contre ne s'accompagne pas, dans le cas général, de forme close permettant une analyse mathématique du modèle.

Pour les objectifs ciblés dans cet article, c'est donc la deuxième approche que nous mettrons en œuvre et, en conséquence, la *simulation* de modèles de systèmes. Cette proposition repose sur une approche modulaire intégrant des composants propres à l'automatique et d'autres propres à l'implantation. Les caractéristiques principales en sont :

– une modélisation des systèmes physiques sous forme de modèles continus,

– une définition des lois de commande sous forme récurrente (notation en q ou en k),

– une manipulation explicite des échantillons avec leurs dates d'occurrences de production et d'utilisation,

– la possibilité de modifier simplement les modèles de régulation pré-existants pour analyser l'influence des choix d'implantation.

L'évaluation que nous proposons se fait, ainsi que nous l'avons dit précédemment, par simulation. Il faut, donc, disposer d'un moteur de simulation (machine d'exécution, simulateur) et programmer les modèles dans le langage d'entrée de ce simulateur. Parmi les outils de simulation disponibles, on peut citer Ptolemy II, développé à l'Université de Berkeley (Ptolemy, 2004), Scilab/Scicos développé à l'INRIA (Scilab, 2004, Kocick *et al.*, 2000) et Matlab/Simulink (Matlab, 2004). Scilab/Scicos et Matlab/Simulink offrent des services équivalents (modélisation continue et discrète adaptées à l'automatique) alors que l'outil Ptolemy II fournit un ensemble plus complet de formalismes de modélisation comme les automates ou les files d'attente. Nous avons fait le choix de Matlab/Simulink comme moteur de simulation et outil support car il constitue un standard de fait dans la simulation des systèmes en automatique. Les modèles développés pourraient cependant être très simplement convertis pour les autres outils. En particulier, l'utilisation de l'outil Ptolemy II simplifierait certains développements car il possède un formalisme pour gérer l'accès aux ressources (Henzinger *et al.*, 2003).

Des travaux, similaires à ceux proposés ici, ont conduit à la proposition de plusieurs simulateurs. Parmi ceux-ci, nous pouvons citer les simulateurs Jitterbug & TrueTime (Cervin *et al.*, 2002, Henriksson *et al.*, 2002) et RTkernel (Nilsson, 1996, Nilsson, 1998), (Elkhoury *et al.*, 2001). RTkernel ne permet que la prise en compte des retards liés à l'existence d'un ordonnanceur ; en particulier, il ne permet pas de manipuler explicitement la notion d'échantillon. Jitterbug & TrueTime offre lui plus de possibilités (mise en série d'ordonnanceurs) mais cache la loi de commande à l'intérieur des modèles d'ordonnancement, ce qui ne donne pas un modèle très lisible,

ni très flexible. C'est pourquoi, nous avons développé une technique de modélisation qui permet de faire apparaître explicitement le modèle usuel de l'automaticien (c'est-à-dire système physique et loi de contrôle) mais aussi de rendre compte de manière naturelle des retards engendrés par l'implantation.

3. Principes de base de la modélisation

Le logiciel Matlab/Simulink, utilisé comme support de simulation, offre des fonctionnalités intéressantes pour la technique que nous avons développée :

– une approche reposant sur les signaux,

– un grand nombre de modèles prédéfinis, sous forme de composants intégrés dans des boîtes à outils,

– un langage de programmation spécifique qui permet de définir de nouveaux composants,

– enfin, la possibilité d'encapsuler des modèles les uns dans les autres (hiérarchie de composants).

La machine d'exécution Simulink permet de simuler l'évolution des systèmes continus représentés par des équations différentielles ainsi que celle des systèmes échantillonnés. Dans cet article nous montrons comment utiliser ces moyens pour intégrer dans un même modèle le point de vue du système et de sa commande, d'une part et celui de l'architecture d'implantation du système de contrôle-commande, d'autre part. Si la définition des perturbations dans un modèle Matlab/Simulink ne pose pas de problème, il n'en est pas de même de la modélisation des retards. En utilisant les blocs prédéfinis de Matlab/Simulink, on peut modéliser partiellement l'influence de l'implantation. Il s'agit notamment de certaines formes de retards et des gigues (voir section précédente). Cependant, pour prendre en compte de manière générale tout modèle de retard tout en assurant la plus grande indépendance entre blocs propres à l'automatique et blocs relevant de l'implantation, nous proposons des *blocs événementiels* qui modélisent à la fois les traitements et l'activation de ces traitements sur événement. Nous distinguons, dans la suite, les *blocs continu*, les *blocs événementiels élémentaires* et les *blocs événementiels de gestion d'accès aux ressources* et présentons par quels moyens sont modélisées les interactions entre ces blocs.

3.1. *Bloc continu*

Un bloc continu modélise un système physique à contrôler (voir figure 2). Il est représenté par :

– un système d'équations différentielles,

– des données d'entrée I,

– des données de sortie O.

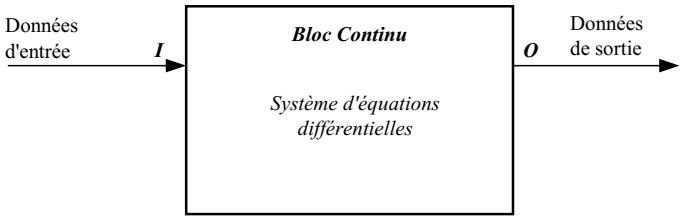

Figure 2. *Représentation d'un système continu*

3.2. *Bloc événementiel élémentaire*

Un *bloc événementiel élémentaire* possède une entrée pour les données (notées I) et une pour l'événement (noté E_I) activant le traitement associé à la consommation de ces données dans le bloc. De la même manière, il peut produire des données en sortie (notées O). La production de données en sortie se fait sur l'occurrence d'un événement (noté E_O). Les caractéristiques d'un bloc élémentaire sont :

– I représentant les données d'entrée (dont la consommation est supposée ici atomique),

– O représentant les données de sortie (dont la production est supposée atomique),

– $f_{traitement}$ est la fonction de transformation des données d'entrée I en données de sortie O,

– E_I est l'événement dont l'occurrence active la fonction $f_{traitement}$,

– E_O est l'événement associé à la la production des données en sortie O,

– $f_{controle}$ est la fonction qui, à une date d'occurrence de l'événement d'entrée E_I fait correspondre la date d'occurrence de l'événement de sortie E_O.

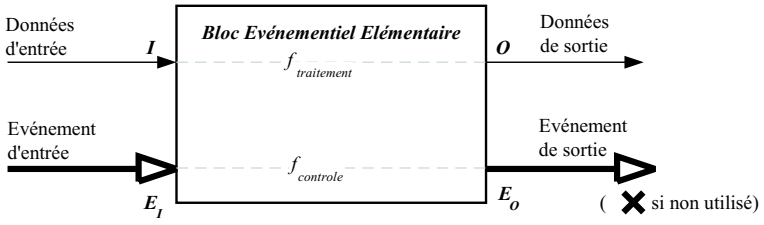

Figure 3. *Modèle générique d'un bloc événementiel élémentaire*

La forme générique d'un bloc événementiel élémentaire est donnée à la figure 3. Ci-dessous, on note I^j et O^j, les valeurs de données d'entrée consommées et de données de sortie produites lors de la $j^{\text{ème}}$ occurrence de l'événement E_I , $val\,(d)$, la

valeur d'une donnée d et $date(E, j)$ la date de la $j^{\text{ème}}$ occurrence de l'événement E. Les différents blocs événementiels élémentaires développés sont :

– le bloc *Bloqueur* (ou échantillonneur) : c'est le bloc fondamental qui discrétise un signal continu. Notons, que dans le cas des systèmes échantillonnés, les données produites par le bloc modélisant le système à contrôler (continu) sont toujours consommées par un bloc de type *bloqueur*. Sur l'occurrence d'un événement d'entrée, la valeur du flux d'entrée est collectée. Elle est retransmise en sortie, sans modification, et associée à un événement qui pourra servir à activer d'autres blocs. L'événement de sortie est synchrone avec celui d'entrée. Plus précisément, les fonctions $f_{traitement}$ et $f_{controle}$ sont telles que :

$$\forall j, val(O^j) = val(I^j)$$

$$\forall j, date(E_O, j) = date(E_I, j)$$

– le bloc *Loi de Commande* : il modélise un algorithme de traitement défini sur les échantillons. Il consiste en une fonction de transfert en q (notée ici $transfert(x)$). La notation en q est utilisée ici pour rappeler que la propriété de périodicité du traitement n'est pas systématiquement assurée et que la fonction de transfert est donc définie sur les échantillons quelles que soient leurs dates de production (Ogata, 1987). Dans ce cas, également, l'événement de sortie est synchrone avec celui d'entrée. Il faut noter, ici, que le retard induit par l'exécution de la loi de commande est modélisé par un autre bloc ainsi que nous le verrons dans les exemples de la section suivante. Plus précisément, les fonctions $f_{traitement}$ et $f_{controle}$ sont telles que :

$$\forall j, val(O^j) = transfert(val(I^j))$$

$$\forall j, date(E_O, j) = date(E_I, j)$$

– les blocs *Retard Constant* et *Retard Variable* : comme leur nom l'indique, ils modélisent des retards spécifiant que la production de l'événement E_O est décalée d'un certain retard (noté $retard(t)$) de la date d'occurrence de l'événement E_I :

$$\forall j, val(O^j) = val(I^j)$$

$$\forall j, date(E_O, j) = date(E_I, j) + retard(date(E_I, j))$$

– le bloc *Perte* : il représente l'occurrence de pertes de données entre deux blocs suivant un certain motif qui se reproduit périodiquement ou qui est généré aléatoirement en respectant une contrainte sur la moyenne des occurrences. Ce bloc est utile pour comprendre l'influence des pertes sur l'évolution du système. On note $perte(j)$ la fonction à valeur booléenne qui retourne $vrai$ si, la $j^{\text{ème}}$ occurrence de l'événement E_I ne conduit pas à la production d'un événement E_O et $faux$, sinon.

$$\forall j, val(O^j) = val(I^j)$$

$$\forall j, date(E_O, j) = \begin{cases} \infty, \text{si } perte(j) = vrai \\ date(E_I, j), \text{sinon} \end{cases}$$

3.3. *Bloc événementiel de gestion d'accès aux ressources*

Un *bloc événementiel de gestion d'accès aux ressources* est utilisé lorsque la politique d'accès aux ressources induit des retards dont la durée ne peut pas être calculée par une fonction définie a priori (par exemple, lorsqu'il est impossible de prévoir les dates d'occurrences de certains événements en raison d'une loi de probabilités sur les pertes). La représentation complète d'un tel bloc est donnée à la figure 4.

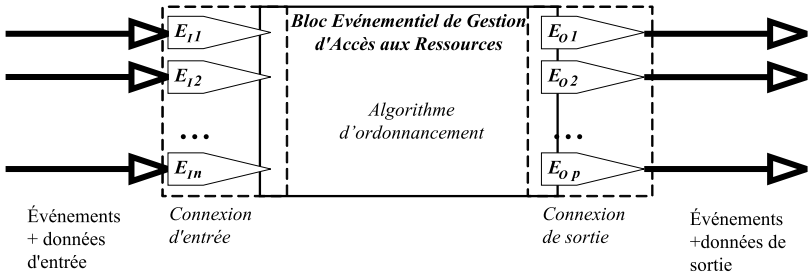

Figure 4. *Modèle générique d'un bloc événementiel de gestion d'accès aux ressources ; les données en entrée ne sont pas modifiées par le traitement du bloc*

Deux types de bloc événementiel de gestion d'accès aux ressources ont été étudiés : le premier modélise la gestion d'accès à la ressource processeur (bloc *Ordonnanceur CPU*) tandis que le deuxième représente la gestion d'accès à un médium de communication (Bloc *Ordonnanceur Réseau*).

De manière générique, un tel bloc se compose de trois éléments principaux :

– les *connexions d'entrées*, notées IC, par lesquelles les événements d'entrée se produisent. Les seuls types d'événements d'entrée sont, pour un bloc *Ordonnanceur CPU*, l'occurrence de l'activation d'un traitement (par exemple, l'événement de sortie d'un bloc *Loi de Commande*) et, pour un bloc *Ordonnanceur Réseau*, l'occurrence d'un événement correspondant à la production d'une donnée le long d'un flux de données supporté par un réseau. Les événements d'entrée des blocs de gestion d'accès aux ressources sont valués. Dans le cas d'un bloc de type *Ordonnanceur CPU*, chaque événement d'entrée est valué par le temps d'exécution sur le processeur (par exemple, le pire temps d'exécution – WCET –) du traitement associé au bloc producteur de cet événement ainsi que d'autres caractéristiques propres à la stratégie d'ordonnancement modélisée (priorité de ce même traitement pour les ordonnanceurs reposant sur une priorité fixe, échéance pour des ordonnanceurs de type « *Earliest Deadline First* »...). Pour un bloc de type *Ordonnanceur Réseau*, à tout événement sont associées les valeurs suivantes : une information permettant de calculer le temps de transmission (par exemple, la taille utile des données dont la production se fait dans le bloc producteur de cet événement) et, suivant le protocole modélisé, certaines caractéristiques du message constitué par ces données (par exemple, sa priorité pour un accès priorisé

au médium de communication ou la place qui lui est réservée dans un réseau de type TDMA) ;

– les *connexions de sortie*, notées OC, par lesquelles sont produits les événements de sortie. Les seuls événements de sortie sont, pour un bloc *Ordonnanceur CPU*, l'occurrence de la fin d'exécution d'un traitement (correspondant, par exemple, au temps nécessaire à l'exécution de la fonction de transfert d'une loi de commande) et, pour un bloc *Ordonnanceur Réseau*, l'occurrence d'un événement correspondant à la fin de transmission d'une donnée sur un réseau ;

– l'*algorithme* qui produit les occurrences des événements de sortie (il s'agit de l'automate implantant la stratégie d'ordonnancement ou le protocole d'accès au réseau).

Les algorithmes disponibles sont, d'une part, pour les blocs *Ordonnanceur CPU*, les politiques EDF (*Earliest Deadline First*), FPP (*Fixed Preemptive Priority*), *Round Robin* et FIFO (*First In First Out*) et, d'autre part, pour les blocs *Ordonnanceur Réseau*, un protocole de communication « à priorités ».

3.4. *Connexion entre blocs*

Deux types de liens sont disponibles pour construire un modèle complet de système régulé à partir des blocs élémentaires décrits précédemment :

– les *liens de données* : ils sont utilisés pour relier les données de sorties d'un bloc *continu* ou *événementiel* aux données d'entrée d'un autre bloc *continu* ou *événementiel*. La valeur d'une donnée d'entrée d'un bloc événementiel est consommée à l'occurrence de l'événement d'entrée de ce bloc. Dans le cas où le bloc d'origine de cette donnée est *continu*, la donnée est égale à la dernière valeur obtenue par résolution pas à pas du système d'équation différentielles du bloc amont *continu*. Dans le cas où le bloc d'origine de cette donnée est de type *événementiel*, la donnée est la dernière valeur produite par ce bloc amont ;

– les *liens de* contrôle : ceux-ci sont utilisés pour relier l'événement de sortie E_O d'un bloc à l'événement d'entrée E_I d'un autre bloc. Cette relation s'établit directement ou via des *connexions d'entrée* ou des *connexions de sortie*. Un exemple de l'utilisation des connecteurs est donné sur la figure 10.

La méthode de modélisation proposée permet donc de représenter l'influence de l'architecture informatique support du système de contrôle-commande à plusieurs niveaux d'abstraction :

– *simple retard* dont la fonction d'évaluation est définie *a priori* (par exemple, un retard constant),

– *retard calculé en ligne en cours de simulation* (par exemple, un retard lié à l'accès à une ressource partagée).

Notons que l'outil TrueTime (Cervin *et al.*, 2002, Henriksson *et al.*, 2002) propose ces deux mêmes possibilités. Par comparaison à ce produit, notre proposition offre l'avantage de séparer clairement le problème d'automatique du problème de l'implantation du système en ce sens que les protocoles d'accès aux ressources et les lois de commande ne sont pas mélangés.

4. Exemples de modèles de systèmes

4.1. *Modélisation de systèmes de commande élémentaires*

La figure 5 illustre un exemple élémentaire de modèle d'implantation d'un système échantillonné en faisant abstraction des retards liés à l'implantation ; le système physique est modélisé en utilisant les blocs habituels de Matlab/Simulink, par contre le système de contrôle est modélisé à partir des blocs événementiels proposés. On voit que le bloc de calcul de la commande à appliquer au système est activé sur réception d'un événement produit par le bloqueur, lui-même étant activé périodiquement (échantillonnage périodique du système physique).

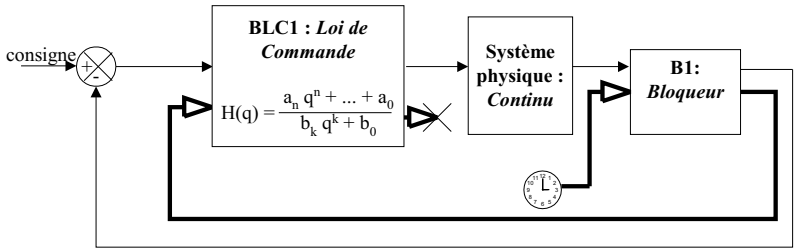

Figure 5. *Exemple élémentaire d'une loi de commande*

4.2. *Modélisation à l'aide de retards à fonction de calcul définie a priori*

4.2.1. *Système à retard variable*

La figure 6 montre comment modéliser un retard variable au niveau de la chaîne de contrôle. Dans le cas où le traitement n'est effectué que dans un seul bloc (c'est-à-dire, qu'implicitement, on ne considère pas de distribution du traitement sous la forme de blocs activés indépendamment les uns des autres), on peut considérer l'ensemble des retards au niveau de l'acquisition des données.

4.2.2. *Gigue sur l'acquisition des données d'entrée*

Le schéma 7 modélise un phénomène de gigue au niveau de l'acquisition. Cette gigue est représentée en utilisant un retard variable appliqué à tout événement pério-

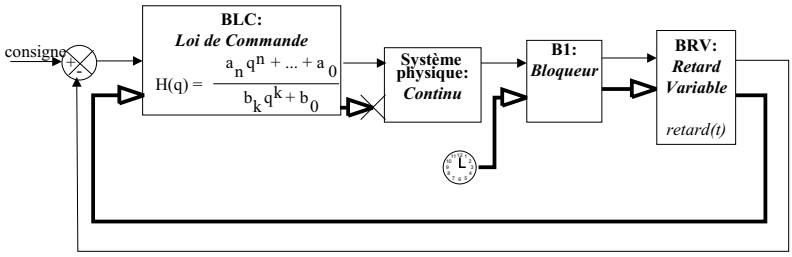

Figure 6. *Modélisation à l'aide de retards à fonction de calcul définie a priori*

diquement généré par l'horloge. L'événement de sortie de ce bloc « retard variable » active le bloqueur.

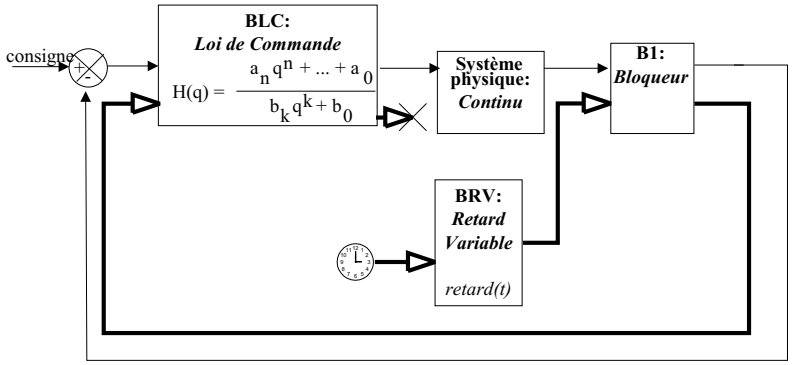

Figure 7. *Modélisation d'une gigue sur l'acquisition des données*

4.3. *Modélisation d'un système dont les traitements suivent différentes lois d'activation.*

La figure 8 présente un exemple où les activations de l'acquisition et du traitement ne sont pas synchronisées sur la même horloge. En effet, il est courant de procéder à une acquisition des données à une fréquence plus élevée que celle d'activation de la loi de commande et de calculer la commande avec la donnée la plus récente.

On peut remarquer que dans les exemples illustrés aux figures 5, 6, 7, le mode de coopération implicite entre le bloc *Bloqueur* et le bloc *Loi de Commande*, ou entre le bloc *Bloqueur* et le bloc *Retard*, ou entre le bloc *Retard* et le bloc *Loi de Commande* est de type « push entre blocs ». C'est-à-dire que le bloc producteur d'une donnée (par

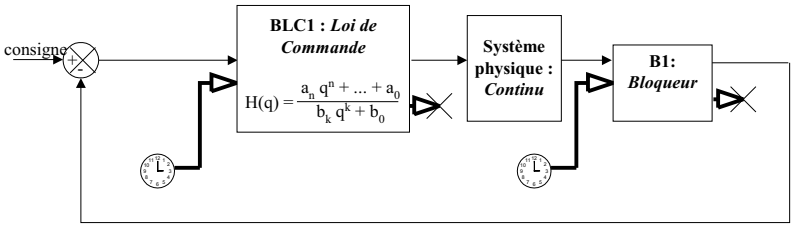

Figure 8. *Modélisation d'activations indépendantes du calcul de la loi de commande et de l'acquisition des données*

exemple, bloc *Bloqueur*) active la consommation de la donnée dans le bloc consommateur de celle-ci (par exemple, bloc *Loi de Commande*). Pour ne pas alourdir la représentation graphique Matlab/Simulink, nous les représentons uniquement par des liens entre événement de sortie d'un bloc et événement d'entrée d'un autre bloc. Il n'en est pas de même dans le cas de l'exemple de la figure 8 ; l'algorithme associé au bloc *Loi de Commande* a sa propre loi d'activation. Cet algorithme consomme, à l'occurrence de son événement d'entrée (périodique, ici) la dernière valeur produite par le bloc *Bloqueur*. Il existe également un mode « pull » permettant, par exemple, de définir une relation de type client-serveur ; pour une définition formelle des modes « push » ou « pull » et de leurs variantes, le lecteur peut se référer à (Jumel, 2003).

4.4. *Modèles explicites de la gestion d'accès aux ressources*

Dans le cas où l'on désire analyser de façon plus précise l'impact d'une politique d'accès aux ressources (ordonnancement, protocole d'accès au médium de communication) ainsi que la configuration de tâches et/ou de messages au niveau de l'implantation (priorités, placement...), il faut utiliser les *blocs événementiels de gestion d'accès aux ressources*.

L'exploitation de ces blocs est illustrée sur l'exemple de la figure 9. Fonctionnellement, trois fonctions du système de contrôle-commande sont identifiées : l'*acquisition* d'un échantillon sur le procédé physique à contrôler, le *calcul* de la commande à appliquer à l'actionneur en fonction de la valeur de l'échantillon et de la consigne demandée et l'*application de la commande* calculée, par l'actionneur. La fonction d'*acquisition* est périodiquement activée et les modes de coopération sous-jacents entre fonction *acquisition* et fonction *calcul* ainsi qu'entre fonction *calcul* et fonction *application de la commande* sont fonctionnellement de type « push entre blocs ». Ceci signifie que, fonctionnellement, les activations de la fonction *calcul* (respectivement *application de la commande*) dépendent des instants de production des informations de sortie de la fonction *acquisition* (respectivement *calcul*). Nous proposons une implantation de cette « architecture fonctionnelle » sur trois calculateurs (nommés sur la figure 9 res-

pectivement, Capteur, Contrôleur et Actionneur) communiquant par l'intermédiaire d'un réseau.

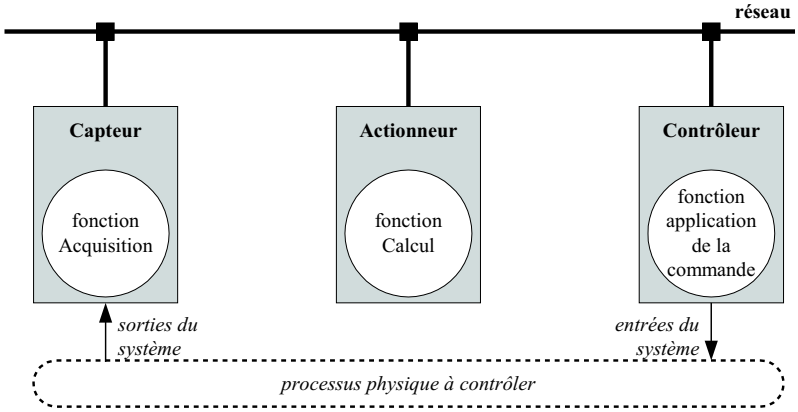

Figure 9. *Exemple d'architecture opérationnelle de système de contrôle-commande distribué*

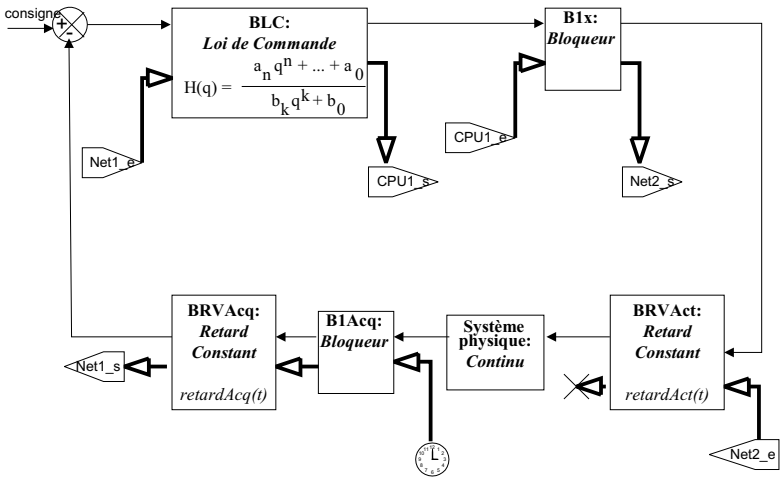

Figure 10. *Modélisation de l'architecture de commande ; l'accès aux ressources CPU et réseaux est représentée sur la figure 11*

La modélisation de cette architecture en utilisant les blocs proposés, est donnée dans les figures 10 et 11. Nous supposons dans cet exemple, que sur le calculateur Capteur (respectivement Actionneur), seule une tâche implante la fonction acquisition

(respectivement application de la commande). Il n'est donc pas utile d'introduire de blocs pour gérer la ressource processeur sur ces calculateurs ; par contre, si on fait l'hypothèse que ces fonctions utilisent un temps constant pour s'exécuter, les blocs BRVAcq (respectivement BRVAct) doivent intégrer un retard constant correspondant à ces temps de traitement. La ressource processeur du calculateur Commande est supposée, sur cet exemple, partagée entre la tâche implantant la fonction calcul et une autre tâche non représentée ici (correspondant, par exemple, à une autre régulation).

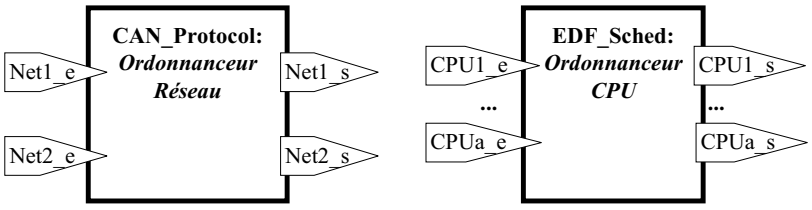

Figure 11. *Modélisation des gestionnaires d'accès aux ressources*

Pour plus de clarté, nous présentons le modèle en deux parties qui formeraient sous Matlab/Simulink un modèle unique. La première partie (figure 10) introduit, de manière simplifiée, les blocs traditionnels du modèle de commande tandis que la deuxième partie (figure 11) montre l'intégration des blocs de gestion des ressources réseau et processeur du calculateur Contrôleur. Les liens sont indiqués par les motifs Neti_e, Neti_s, CPUi_e et CPUi_s. Ceux-ci contiennent l'événement et les données associées ainsi que décrit précédemment.

5. Présentation de l'étude de cas

L'exemple étudié est tiré d'un des ouvrages de référence des systèmes échantillonnés écrit par K. Ogata (Ogata, 1987). Il concerne la régulation en position d'un moteur à courant continu, pouvant par exemple contrôler le déplacement d'un mobile sur une crémaillère. Il est constitué :

– du modèle d'un moteur,

– des algorithmes de commande.

5.1. *Modèle du moteur*

Le comportement du moteur est donné par sa fonction de transfert G qui lie la tension appliquée U à la position du moteur m :

$$G(p) = \frac{m(p)}{U(p)} = \frac{1}{p(p+2)}$$

Ceci correspond à une relation différentielle entre la tension et la position du type $m'' + 2m' = U$. Un certain nombre de perturbations écartent le moteur de son fonctionnement idéal, nous les avons modélisées sous forme de bruits blancs (générateur de bruit blanc standard sous Matlab de paramètre 0.05).

5.2. *Commande du système*

La commande est un correcteur en boucle directe (cf. figure 12). La période choisie pour le contrôle est de 0.2 s. Ce correcteur a été calculé par placement de pôles sur le système continu équivalent puis a été discrétisé. Le contrôleur est défini par sa fonction de transfert en q (cf. (Ogata, 1987)) :

$$H(q) = \frac{13.57(q - 0.6703)}{(q - 0.2644)}$$

ce qui correspond à un algorithme de traitement de la forme :

$$c(n) = c(n-1)*0.2644+13.57*(cn(n)-p(n))-9.096*(cn(n-1)-p(n-1))$$

où $c(k)$ est la commande, $cn(k)$ est la consigne et $p(k)$ est la position pour le cycle k.

La figure 12 donne une représentation d'un tel système où l'on fait abstraction de ses caractéristiques d'implantation.

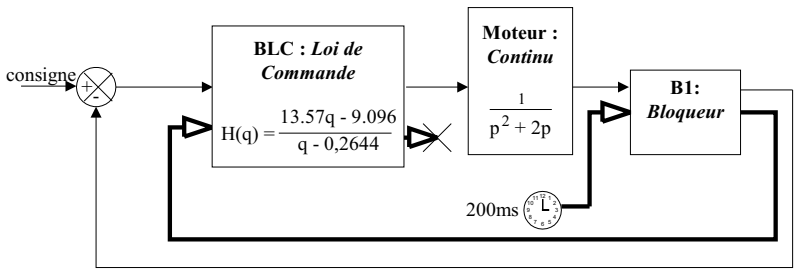

Figure 12. *Représentation schématique de la régulation du moteur*

6. Influence de l'implantation

Ainsi que nous l'avons montré précédemment, la prise en compte de l'implantation peut relever de deux niveaux d'abstraction différents :

– sans analyse fine de l'impact de la politique d'accès aux ressources de l'architecture informatique ; cette abstraction est suffisante dans le cas où l'application réalisant le système de contrôle-commande est implantée à l'aide d'une seule tâche s'exécutant seule sur un seul processeur. Dans ce cas, on utilise uniquement des blocs *retards* ;

– avec une prise en compte des stratégies d'ordonnancement locales et, le cas échéant, de protocoles de communication ; une telle technique est nécessaire dès que l'application de contrôle-commande est distribuée et/ou dès que l'un des calculateurs utilisés supporte l'exécution de plusieurs tâches. On est alors obligé d'intégrer les blocs *de gestion d'accés aux ressources*.

Nous montrons dans les sections suivantes l'utilisation de ce simulateur aux deux niveaux d'abstraction et les résultats obtenus.

6.1. *Implantation sous forme d'une seule tâche avec retard constant*

Dans ce cas, nous nous sommes attachés à évaluer l'impact de la durée d'exécution de la tâche implantant l'application de contrôle-commande sur les critères de qualité du système. Nous considérons ici une durée d'exécution constante. Cette durée se modélise alors sous forme *d'un retard constant*. Dans la suite, nous appelerons *temps de traitement* la différence entre la fin d'exécution de la tâche qui applique la consigne et sa date d'activation ; dans le cas d'une seule tâche, le temps de traitement est égal au temps d'exécution.

La figure 14 représente l'évolution des différents critères de qualité (cf. section 2.1) en fonction du temps de traitement. La première courbe à gauche représente l'évolution du dépassement après un changement de consigne pour un retard constant, on remarque une augmentation linéaire du dépassement jusqu'à un certain retard puis un pic. Ce pic correspond à la perte de stabilité du système (sous forme de sinusoïde d'amplitude augmentant exponentiellement et conduisant à la saturation ou la destruction du système) et correspond au retard critique du système. Le calcul explicite de ce retard critique est relativement simple à obtenir (cf. (Ogata, 1987)) et constitue une caractéristique importante du système à contrôler.

La courbe du milieu montre l'évolution, en fonction du temps de traitement des tâches, du temps de réponse du système pour une erreur statique limitée à 5 % (*i.e.* temps de réponse à 5 %) après un changement de consigne. Comme le critère de dépassement, le temps de réponse se dégrade avec l'augmentation du temps de traitement. Cette dégradation ne se fait plus de manière linéaire mais sous forme d'escaliers. Ce résultat s'explique par la nature oscillante du système, le temps de réponse est lié à la dernière oscillation hors gabarit (cf. figure 13). Il y a donc un changement important du temps de réponse dès que la stabilisation dans le gabarit nécessite une oscillation de plus. La connaissance de ce type de caractéristique peut se révéler très pertinente pour le dimensionnement et le choix de processeurs.

La dernière courbe sur la figure 14 représente l'évolution de l'IAE en absence de changement de consigne et donc la capacité du système régulé à rejeter les pertur-

bations. On remarque ici que la difficulté à rejeter les perturbations s'accentue très fortement avec l'augmentation du temps de traitement.

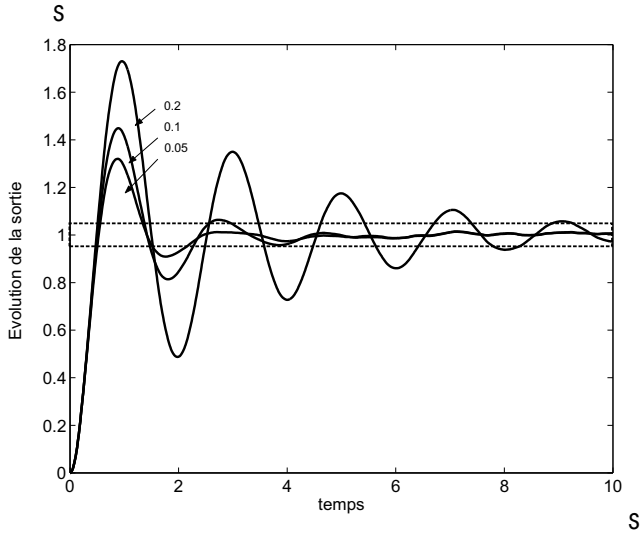

Figure 13. *Evolution de la sortie du système régulé pour différentes durées d'exécution de la tâche implantant l'application de commande*

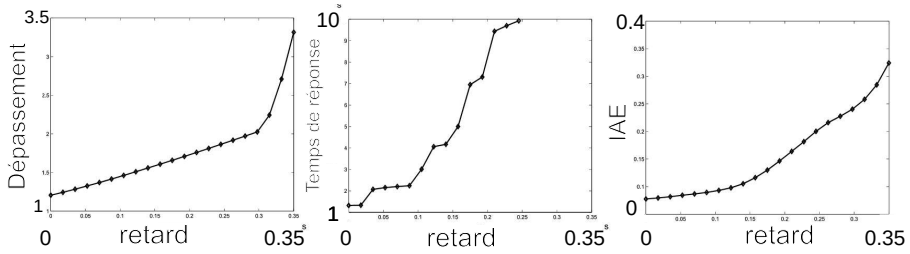

Figure 14. *Evolution du temps de réponse, du dépassement et de l'IAE en fonction d'un retard constant (retard en secondes)*

En pratique, il est possible d'adapter la loi de commande à la connaissance *a priori* de ces temps de traitement d'une façon qui permet d'améliorer considérablement les performances du système ; ce peut être réalisé par exemple avec un prédicteur de

Smith, cf. (Jumel, 2003). De telles techniques nécessitent donc que les temps d'exécution soient constants et aussi qu'ils soient *a priori* prévisibles. Le lecteur pourra consulter (Jumel *et al.*, 2002) où sont présentées des politiques de réservation de ressource en avance qui assurent une bonne prévisibilité du système.

6.2. *Implantation sous forme d'une seule tâche avec retard variable*

L'étude précédente repose sur l'hypothèse d'un temps de traitement constant pour l'application de contrôle-commande. Généralement, et cela même dans le cas d'implantations élémentaires, cette hypothèse n'est pas vérifiée. Les techniques de modélisation et de simulation présentées précédemment sont alors nécessaires. La variation du temps de traitement peut être liée à la durée d'exécution du code sur le CPU (qui est fonction du chemin d'exécution à l'intérieur de l'application), mais aussi, dans le cas de systèmes multitâches, à l'influence de l'ordonnancement ou des mécanismes de synchronisation inter-tâches. Le temps de traitement de la tâche considérée est donc dépendant de la politique d'ordonnancement et des caractéristiques des autres tâches du système. La prise en compte de traitement variables, des retards variables au sens de l'automatique, pose de réels problèmes en pratique ; les outils de modélisation et de simulation présentés précédemment offrent une possibilité pour l'étude de cette influence.

Nous présentons ci-dessous les résultats de simulation obtenus sur l'exemple du moteur en présence d'un temps de traitement variable. La figure 15 montre la distribution de la qualité de la régulation du moteur (temps de réponse et dépassement) sur plusieurs centaines de scénarios de fonctionnement avec des temps de traitement variables. Le temps de traitement est tiré aléatoirement suivant une loi uniforme dans un intervalle de taille 0.05 s centré autour de 0.1 s. Les traits en pointillés représentent la qualité obtenue pour un temps de traitement constant maximal (borne supérieure de l'intervalle), minimal et dans le cas d'un traitement constant avec la moyenne sur l'intervalle.

Sur la figure 15, on observe que les temps de réponse pour un temps de traitement constant respectivement minimal et maximal encadrent les résultats obtenus. On remarque également que les temps de réponse forment des amas autour de valeurs particulières : ces valeurs correspondent aux nombres d'oscillations avant stabilisation. Le cas de fonctionnement moyen est proche du cas donné par un temps de traitement constant correspondant à la moyenne des temps de traitement. La moyenne des temps de traitement semble donc être une caractéristique de fonctionnement plus fondamentale que la connaissance d'une borne pour appréhender les performances d'une régulation.

L'évolution du dépassement fait apparaître que la qualité dans le cas variable peut être meilleure ou moins bonne que dans les cas constants, maximal et minimal, que l'on pense intuitivement être le pire cas et le meilleur cas (représentés en pointillés). Cet exemple illustre le fait que la validation d'une architecture en considérant unique-

ment ce que l'on pense être son pire cas de fonctionnement n'est pas suffisante dans le cadre du contrôle-commande. En effet, la variabilité peut amener à une dégradation plus forte de la qualité de fonctionnement.

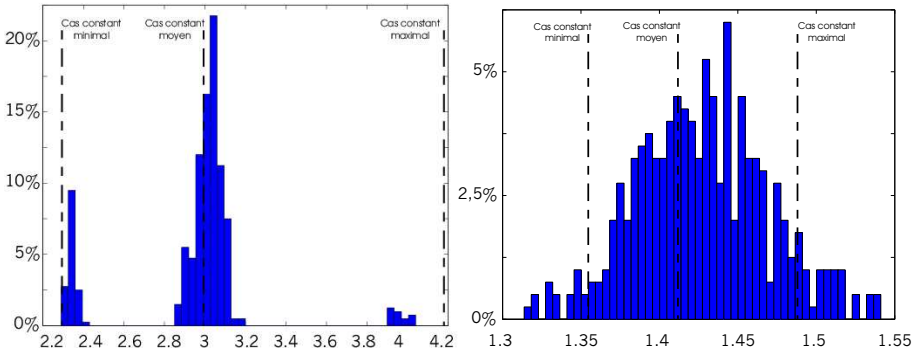

Figure 15. *Distribution du temps de réponse à 5 % (en secondes) et du dépassement (en mètres) en présence de temps de traitement variables*

Les modèles probabilistes basés sur des lois uniformes, comme ceux utilisés à la section précédente, ne sont, en règle générale, pas suffisants pour modéliser de façon satisfaisante le fonctionnement du système ; des blocs spécifiques (de gestion d'accès aux ressources) permettent de prendre en compte ou de générer des scénarios correspondant précisément à la politique d'ordonnancement utilisée et aux tâches du système. Dans le cas d'utilisation de ressources réseaux ou processeurs partagées entre plusieurs activités, le choix des politiques d'ordonnancement peut considérablement influencer la qualité des différents systèmes. Pour illustrer le phénomène dans le cas d'une ressource processeur, nous considérons 3 applications de contrôle-commande, implantées sur le même système multitâches et réalisant, chacune, la régulation d'un moteur. Les 3 moteurs sont identiques. Chaque régulation se fait par une seule tâche périodique T_i ($i \in \{1, 2, 3\}$) enchaînant, comme dans les exemples précédents, l'acquisition, le calcul et l'application de la commande. Afin de bien évaluer l'influence des solutions d'ordonnancement envisagées, nous faisons l'hypothèse que les 3 tâches ont les mêmes caractéristiques :

– la date d'activation initiale : 0,

– la période entre deux activations : 0.2 s,

– le temps d'exécution (WCET) : 0.05 s.

Les acquisitions sont réalisées périodiquement aux instants $t_i = 0.2 \cdot i$ et le seul événement observé correspond à l'application de la commande à la fin du traitement de la tâche.

6.2.1. *Choix d'ordonnancement*

Nous envisageons quatre solutions d'ordonnancement :

– *Une politique à priorité fixe* (FPP) : la tâche T_1 est plus prioritaire que T_2 qui elle-même est plus prioritaire que T_3. En raison de la simultanéité des débuts d'exécution, nous obtenons un motif périodique élémentaire pour les exécutions (cf. figure 16). Les dates de fin de traitement sont respectivement ($t_i + 0.05$, $t_i + 0.10$, $t_i + 0.15$) pour les tâches T_1, T_2, T_3.

– *Une politique Round Robin* : c'est une politique de « tourniquet » temporel avec un quantum de temps fixé ici à 10ms. Sous cette politique, à chaque cycle, une tâche se termine 0.13 s après son activation, une autre 0.14 s et une dernière à 0.15 s comme représenté sur la figure 16. Nous faisons l'hypothèse qu'il n'y a pas de règles *a priori* pour déterminer quelle sera la tâche à s'exécuter en premier et cette décision est donc prise aléatoirement au début de la simulation (un choix autre que aléatoire ne modifie pas sensiblement la forme des résultats).

– *Une politique à tourniquet sur les priorités ;* les priorités sont données aux différentes activités suivant un tourniquet ; la tâche la plus prioritaire lors d'un cycle, reçoit la priorité intermédiaire au cycle suivant et la priorité la plus faible lors du cycle qui suit. Comme représenté sur la figure 16, les dates de fin de traitement suivent pour chaque tâche un modèle cyclique du type $t_i + 0.05$, $t_{i+1} + 0.10$, $t_{i+2} + 0.15$.

– *Une politique à priorités aléatoires ;* les priorités sont données de façon équiprobable à chacune des tâches. Chaque activité a ainsi la même probabilité de terminer à $t_i + 0.05$, $t_i + 0.10$ ou $t_i + 0.15$.

6.2.2. *Qualité de la régulation*

La figure 17 montre, sous forme d'histogrammes, la répartition des temps de réponses du système pour les différentes politiques. Les résultats présentés correspondent à 500 scénarios de fonctionnement. Ce nombre relativement important de cas permet de faire apparaître les différentes évolutions possibles du système qui peuvent être dues à un certain indéterminisme du système physique (perturbation) ou des temps de réponse (pour la politique FIFO en particulier). Le temps de réponse de la régulation, obtenu pour chaque tâche, varie en fonction des politiques d'ordonnancement mais aussi de l'évolution du bruit sur le système physique au cours de la simulation. Ainsi, en raison du bruit, le temps de réponse n'est pas constant dans le cas FPP et on peut même constater quelques valeurs hors normes comme le troisième amas sous FPP qui correspond à des scénarios fortement bruités pour la priorité 2.

Logiquement, la loi de commande n'étant pas retardée dans son exécution, la meilleure qualité de régulation est obtenue pour la tâche la plus prioritaire avec un ordonnancement à priorité fixe. En revanche, sous cette politique, la qualité se dégrade fortement pour les autres tâches de plus faibles priorités. En comparaison, dans le cas Round Robin, on a une parfaite équité entre les différentes activités mais on obtient une qualité mauvaise pour chacune des activités ce qui s'explique par des temps de traitement de bout en bout très élevés pour toutes les activités.

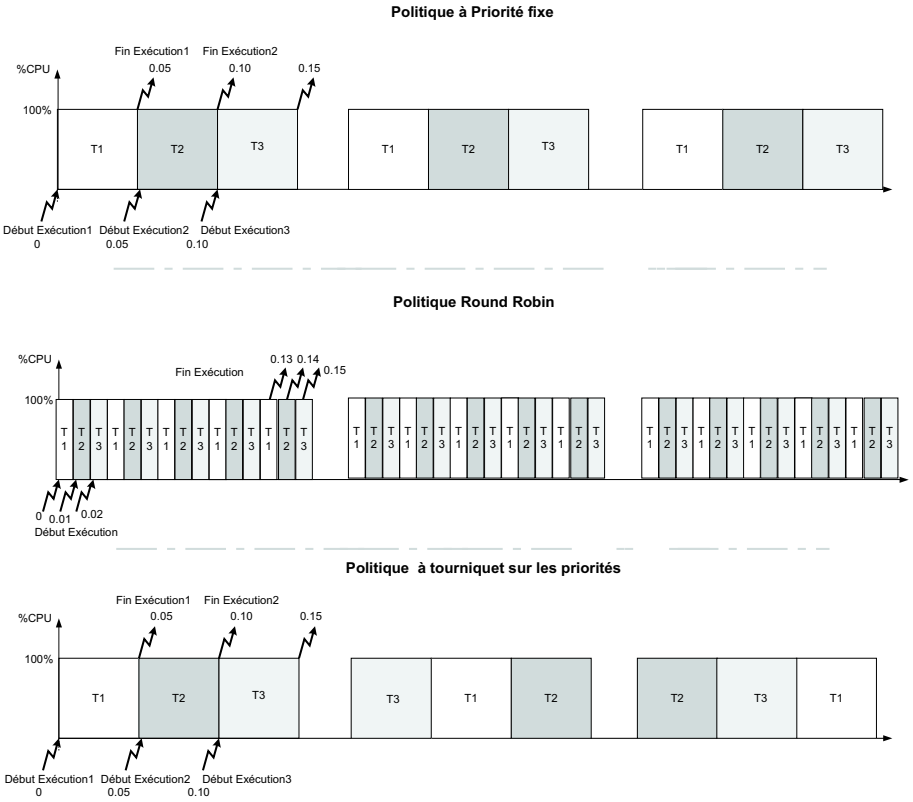

Figure 16. *Dates de début et de fin d'exécution des trois tâches implantant les lois de commande pour différentes politiques d'ordonnancement*

Dans le cas des deux politiques à priorités variables, toutes les tâches ont la même importance relative et il y a une parfaite équité sur l'accès aux ressources. Concernant la politique à priorités aléatoires, les temps de réponses ne sont pas distribués uniformément dans le temps, on retrouve trois amas associés à un temps de réponse de 2.4 s, 3.2 s et 4.1 s. Comme dans les résultats de la section 6.1, l'explication tient en la nature oscillante du système. La qualité obtenue est en moyenne bien plus intéressante que sous FPP car la proportion d'activités dont le temps de réponse de la régulation dépasse 4 s est très faible. La dernière politique (tourniquet sur les priorités) présente des résultats encore plus intéressants, toutes les régulations ont un temps de réponse de l'ordre de 3 s (équivalent à l'activité de priorité moyenne dans le cas FPP). Un aussi bon résultat s'explique par le fait que, dans l'étude de cas, un temps de traitement très long (0.15 s) est toujours suivi d'un temps de traitement très court (0.05 s), cette alternance a en fait pour effet de compenser immédiatement l'influence négative sur l'ensemble des 3 applications de commande, d'un temps de traitement long.

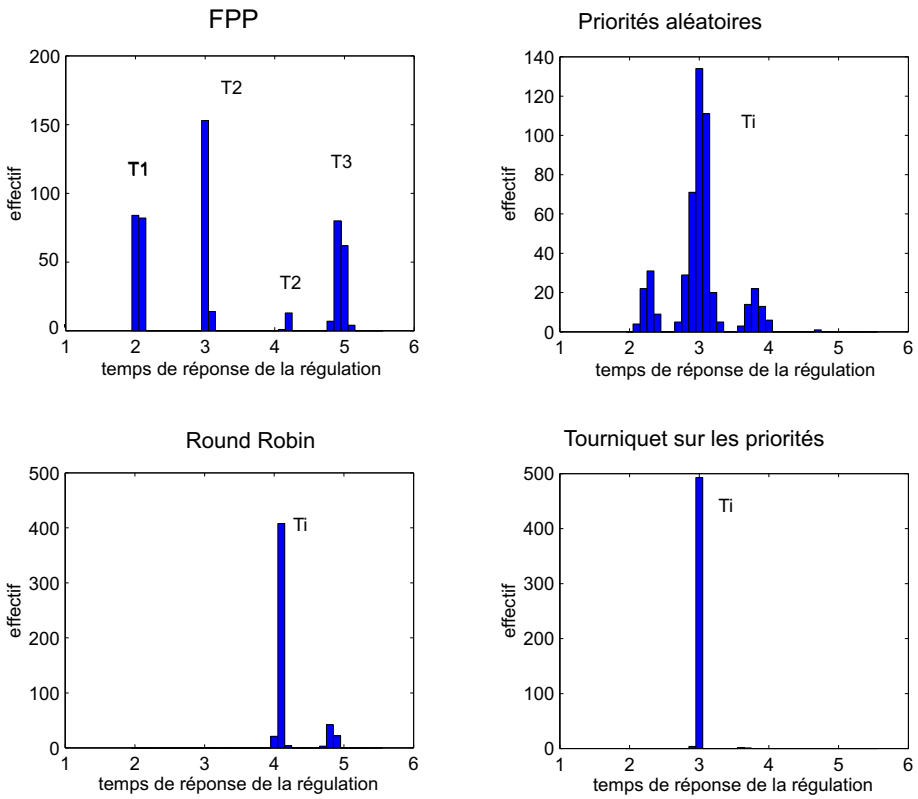

Figure 17. *Histogramme de la répartition des temps de réponse de la régulation (en secondes) avec les différentes politiques étudiées sur 500 simulations*

L'intérêt premier de ces simulations est de montrer que l'on peut évaluer l'influence des choix d'ordonnancement et que cette influence peut être significative sur la qualité de la régulation. Les politiques d'ordonnancement usuelles comme FPP et Round Robin, disponibles sur les systèmes d'exploitation conformes au standard Posix1003.1b (IEEE, 1996), ne conduisent pas à la meilleure qualité pour chaque application considérée. D'autres politiques, très simples conceptuellement, comme un tourniquet sur les priorités, peuvent donner de meilleurs résultats. Remarquons que cette politique peut être implantée sans difficulté particulière sur un système d'exploitation conforme à Posix1003.1b.

7. Conclusion

Nous avons présenté dans cet article comment il était possible d'évaluer la qualité d'une régulation. La mise en place d'outils de simulation est rendue nécessaire par l'absence, dans de nombreux cas, de résultats analytiques. En particulier, celui présenté au cours de cette étude nous a permis de mettre en évidence les relations qui lient les caractéristiques de l'architecture opérationnelle (en particulier temporelles) et l'évolution d'un système régulé, relations qui sont complexes et peu intuitives. Ces outils constituent une aide pour mieux comprendre quels sont les vrais besoins d'une régulation vis-à-vis des caractéristiques des supports d'exécution des algorithmes de commande.

Deux résultats intéressants sont apparus lors notre étude de la régulation du moteur. Tout d'abord, l'évolution de la qualité, en particulier le temps de réponse du système, ne suit pas un profil linéaire. Il est donc important de connaître ce profil si l'on veut améliorer la qualité d'une régulation d'autant plus qu'il existe des fonctions de la qualité de la régulation non monotones vis-à-vis de la durée du temps de traitement. C'est le cas lorsque la loi de commande essaie de compenser les temps de traitement à l'aide d'un prédicteur de Smith.

Le second résultat montre que la variabilité des temps de traitement peut avoir des conséquences très diverses comme par exemple :

– une dégradation de la qualité plus importante que ce que l'on pouvait intuitivement considérer comme le pire cas (cf. figure 15, critère dépassement),

– l'influence relativement faible des bornes sur les temps de traitement vis-à-vis de la moyenne.

Les choix faits lors de la phase de définition des contraintes temporelles liées aux algorithmes de traitement doivent donc être validés par exemple à l'aide de l'outil proposé dans cette étude.

Enfin, l'étude de cas nous a permis de montrer que des politiques d'ordonnancement visant à une certaine équité, comme la politique à priorités alternées, avaient un intérêt dans le domaine des applications temps réel de contrôle-commande.

Le but de nos travaux est de mieux comprendre comment, à partir de besoins en qualité de fonctionnement de haut niveau portant sur une régulation, dériver les propriétés importantes à exiger de l'implantation. Notre principale perspective est d'identifier des classes de systèmes ayant des besoins similaires en termes de qualité de service fournie par le système informatique support, et, à terme, de fournir des « guides » de conception pour chacune de ces classes.

8. Bibliographie

Cervin A., Henriksson D., Lincoln B., Eker J., Årzén K., « Jitterbug and TrueTime : Analysis Tools for Real-Time Control Systems », *Proceedings of the 2^{nd} Workshop on Real-Time Tools*, Juin, 2002.

Elkhoury J., Törngren M., « Towards a Toolset for Architectural Design of Distributed Real-Time Control Systems », *Proceedings of the 22^{nd} IEEE Real-Time Systems Symposium*, 2001.

Henriksson D., Cervin A., Årzén K., « TrueTime : Simulation of Control Loops Under Shared Computer Resources », *Proceedings of the 15^{th} IFAC World Congress on Automatic Control*, Mai, 2002.

Henzinger T., Kirsch C., Sanvido M., W.Pree, « From Control Models to Real-Time Code using Giotto », *IEEE Control Systems Magazine*, vol. 23, n° 1, p. 50-64, 2003.

IEEE, *1996 (ISO/IEC) [IEEE/ANSI Std 1003.1, 1996 Edition] Information Technology — Portable Operating System Interface (POSIX®) — Part 1 : System Application : Program Interface (API) [C Language]*, IEEE Standards Press, 1996.

Jumel F., Défi nition et gestion d'une qualité de service pour les applications temps réel, PhD thesis, Institut National Polytechnique de Lorraine, Novembre, 2003.

Jumel F., Navet N., Simonot-Lion F., « Nouvelles politiques pour la réservation explicite de ressources en avance », *Technique et Science Informatiques, Numéro spécial sur le temps réel*, vol. 22, n° 5, p. 599-617, Août, 2002.

Kocick R., Sorel Y., « De la modélisation à la réalisation : réduction du cycle de développement des applications temps réel distribuées », *Proceedings of the 8^{th} conference on Real-time and embedded systems*, Mars, 2000.

Matlab, « `http://www.mathworks.fr/products/controldesign/` », 2004.

Nilsson J., *« Analysis and Design of Real-Time Systems with Random Delays »*, Master's thesis, IT Lünd, Mai, 1996.

Nilsson J., Real-Time Control Systems with Delays, PhD thesis, IT Lünd, Février, 1998.

Ogata K., *Discrete-time control systems*, Prentice Hall, 1987.

Ptolemy, « `http://ptolemy.eecs.berkeley.edu/ptolemyII/` », 2004.

Scilab, « `http://scilabsoft.inria.fr/` », 2004.

Article reçu le 23 juin 2004

Version révisé le 17 main 2005

Fabrice Jumel *est enseignant-chercheur à l' École Supérieure de Chimie, Physique et Électronique et au laboratoire CITI à Lyon. Ses travaux de recherche portent sur la spécification de services et techniques permettant d'assurer les propriétés de sûreté de fonctionnement d'un système automatisé.*

Nicolas Navet *est chercheur à l'INRIA au sein du projet TRIO. Ses travaux se situent dans le contexte de la conception des applications temps réel et portent en particulier sur l'ordonnan-*

cement de tâches et de messages dans des environnements non déterministes. Plus de détails à l'url `http://www.loria.fr/~nnavet` .

Françoise Simonot-Lion *est professeur à l'École des Mines de Nancy. Elle est responsable scientifique du projet TRIO (Temps Réel et InterOpérabilité), projet commun au CNRS, à l'INRIA, aux Universités de Nancy au sein du LORIA (UMR CNRS 7503). Ses centres d'intérêt sont la modélisation d'architectures distribuées temps réel et les techniques de vérification de propriétés temporelles.*