Modélisation par réseau de Petri coloré des architectures de commande distribuées sur réseau de terrain Ethernet et TCP/IP

Gaëlle Poulard, Bruno Denis, Jean-Marc Faure

5e Conférence Francophone de MOdélisation et SIMulation
"Modélisation et simulation pour l'analyse et l'optimisation des systèmes industriels et logistiques"
MOSIM'04 – du 1er au 3 septembre 2004 - Nantes (France)

MODÉLISATION PAR RÉSEAU DE PETRI COLORÉ DES ARCHITECTURES DE COMMANDE DISTRIBUÉES SUR RÉSEAU DE TERRAIN ETHERNET ET TCP/IP

G. POULARD[1], B. DENIS[1] et J.-M. FAURE[1,2]

[1] LURPA - Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson
F-94235 Cachan CEDEX - France
{poulard, denis, faure}@lurpa.ens-cachan.fr

[2] Supméca
3, rue Fernand Hainaut
F-93407 St Ouen CEDEX - France

RÉSUMÉ : *Les architectures opérationnelles de commande réparties dans plusieurs constituants de commande comme les automates programmables ou les modules d'entrées-sorties déportés font traditionnellement appel à des réseaux de terrain au comportement déterministe (par exemple : PROFIBUS, FIP sur RS485). L'arrivée sur ce segment de marché des protocoles construit autour de TCP/IP sur le media Ethernet introduit une nouvelle difficulté pour les concepteurs de ces architectures de commande. En effet, les exigences de réactivité des architectures, souvent exprimées en terme de temps de propagation d'un événement, sont plus que jamais difficiles à estimer en phase de conception. A cette fin, nous proposons un modèle de simulation du comportement réactif de ces architectures à l'aide de réseaux de Petri colorés et temporisés. Notre modèle est entièrement paramétrable de sorte que l'étude de deux architectures différentes se traduise par le même modèle avec deux jeux de paramètres différents. Une confrontation entre des résultats de simulation sur CPNTools et de mesure sur une architecture réelle sera finalement présentée.*

MOTS-CLÉS : *réseau de Petri coloré temporisé, Ethernet commuté, simulateur à événements discrets, temps de réponse*

1. INTRODUCTION ET PROBLÉMATIQUE

Depuis quelques années, on constate l'arrivée en force d'une offre construite autour des technologies Ethernet et TCP/IP pour la commande des systèmes industriels. Ces technologies possèdent certains avantages tels que des coûts réduits grâce à une production à grande échelle, et des débits de communication importants, pouvant aller jusqu'à 1 Gb/s. Sans oublier la multitude de services qui est disponible sur ces réseaux et l'uniformisation du media de communication dans l'entreprise. L'intérêt du monde industriel est réel; on citera par exemple la journée «Technologie de l'information et de la communication en production» organisé par le CETIM et l'EXERA le 27 janvier 2004. Mais certaines caractéristiques d'Ethernet et TCP/IP comme le protocole d'accès non déterministe, le temps de propagation des informations dans le réseau non borné et le support physique plus complexe qu'un bus RS485 soulèvent des questions quand à la capacité de ce media à garantir des propriétés de déterminisme et de réactivité de la commande.

L'utilisation rationnelle d'Ethernet et TCP/IP comme bus de terrain nécessite de pouvoir répondre à ces questions. Une Action Spécifique du département STIC du CNRS a justement été lancée sur ce thème[1]. C'est également dans cet esprit que de nombreux travaux scientifiques ont déjà portés sur le media Ethernet [13] [7] [6], les protocoles [15] et les nouveaux services offert par ces technologies [5]. Mais peu de travaux proposent une évaluation des architectures de commande au travers d'Ethernet en utilisant les indicateurs de l'utilisateur final (par exemple, le temps de propagation de bout en bout d'un événement). C'est pourquoi nous avons construit un modèle de simulation permettant de connaître à priori les performances temporelles des architectures de commande via Ethernet et TCP/IP. Notre modélisation s'est faite à l'aide de réseaux de Petri temporisés qui sont bien adaptés à notre problème. Nous nous sommes contraints à une approche générique et modulaire qui requière

1. AS 198 (RTP 47) «impact des nouvelles technologies de l'information et de la communication en automatisation».

l'utilisation de coloration et la hiérarchie du modèle. L'analyse temporelle de ce réseau de Petri temporisé coloré hiérarchique est obtenue par simulation avec CPNTools, logiciel développé par l'université d'Aarhus au Danemark [12].

Ce papier décrit dans une première section les différents composants techniques étudiés. Ensuite, la section 3 est consacrée à l'explication du modèle. Enfin, une première utilisation du modèle sur un cas d'étude permettra de le valider par confrontation entre des résultats de simulation et des mesures physiques.

2. LES ARCHITECTURES DE COMMANDE VIA ETHERNET ET TCP/IP

2.1. Les Automates Programmables Industriels (API)

Les principales fonctions attendues d'un API sont l'acquisition de données, le traitement et l'émission d'ordres. L'acquisition et l'émission peuvent se faire localement ou au travers d'un réseau de terrain. Une structure modulaire par carte est couramment retenue pour remplir ces fonctions. Ainsi l'acquisition et l'émission directes se feront par une carte d'entrées-sorties, et au travers d'un réseau par un coupleur réseau. Le traitement sera assuré par la carte processeur, aussi appelée carte CPU. Il sera cyclique ou périodique [1]. Les caractéristiques importantes d'un API pour modéliser son comportement temporel sont le temps de cycle de traitement et les retards dûs au filtrage des entrées et sorties. Nous étudierons seulement ici le cas du traitement périodique.

2.2. Les modules d'entrées-sorties déportés sur réseau

Les modules d'entrées-sorties communiquent avec l'API par un réseau de terrain. Les fonctions attendues ici sont la gestion de l'accès au media unique ainsi que l'acquisition et l'émission de données. Ces fonctions sont assurées par un ou plusieurs protocoles de communication. L'accès au médium est souvent de type maître-esclave pour assurer qu'un seul composant envoie un message sur le réseau [14]. La régularité des acquisitions et emissions est souvent faite par une scrutation cyclique des modules déportés par l'API. Ce temps de cycle de scrutation sera la caractéristique importante pour modéliser les comportements temporels d'architectures distribuées. Même s'ils sont pour certains normalisés, les bus de terrain et les protocoles associés sont encore souvent spécifiques à une communauté limitée de constructeurs et d'utilisateurs.

2.3. Ethernet, TCP/IP et le modèle OSI

Etant donné les avantages que peuvent offrir un réseau et un protocole ouverts et très répandus, Ethernet et TCP/IP se positionnent aujourd'hui comme bus de terrain. On peut décrire ces protocoles en se basant sur le modèle OSI de l'ISO. Ce dernier se présente sous forme de couches, depuis le support physique jusqu'à l'application, précisant pour chacune la fonction attendue (tableau 1).

couche du modèle OSI	couche du modèle TCP/IP	exemple
Application	Application	Modbus et IO scanning
Présentation		
Session		
Transport	Transport	TCP
Réseau	Internet	IP
Liaison de données	Hôte-réseau	Ethernet
Physique		

Tableau 1: couches OSI et TCP/IP

L'utilisateur final va interagir uniquement avec la couche application, mais avant de détailler cette couche dans la prochaine sous-section, il est utile de donner quelques précisions sur les autres protocoles utilisés.

TCP permet une connexion sûre entre deux hôtes. Il découpe la masse de données à envoyer en segments TCP et les empile dans la pile IP [4] [10]. IP a, quand à lui, pour fonction d'envoyer les segments TCP de sa pile, encapsulés dans des paquets IP, à un hôte. C'est lui qui doit déterminer la route que le paquet doit parcourir dans le réseau [11]. Ethernet est un standard (IEEE802.3) de protocole de communication pour des connexions rapides dans les réseaux locaux (LAN). Il envoie sur le réseau des trames Ethernet qui encapsulent les paquets IP selon une méthode d'accès non déterministe (CSMA/CD) [8] [3].

Physiquement, un réseau Ethernet est différent des bus de terrain classiques car il comporte des composants actifs tel que concentrateur, commutateur et routeur qu'il faudra modéliser. Pour les architectures de commandes, les concentrateurs ne serons pas utilisés. Dans cette communication, nous nous intéresserons uniquement aux architectures Ethernet commuté sans routeur. Les commutateurs seront modélisés par un retard.

2.4. Protocole pour API sur TCP/IP

La couche application du modèle OSI a pour fonction l'acquisition et l'émission de données qui est assurée par un ou plusieurs protocoles de communication.

Dans notre cas, deux protocoles seront nécessaires. Modbus [9] permet d'offrir un service de messagerie pour faire communiquer un client et un serveur sur un réseau Ethernet TCP/IP. Le «IO scanning», quand à lui, assure le caractère cyclique d'envoi des messages. Ce dernier est propre au coupleur réseau de l'API, ici un coupleur Ethernet. Le paramètre à prendre en compte pour modéliser ces protocoles sera le cycle de IO scanning.

3. MODÉLISATION

Le modèle va s'attacher à expliciter tous les délais significatifs qui peuvent se produire au cours du traitement et de la transmission de l'information dans l'architecture. Les attentes de disponibilité de ressources partagées seront également à prendre en compte. La difficulté inhérente à toute modélisation est la détermination du niveau de détail que le modèle doit atteindre. Ici nous considérerons qu'un délai est significatif dans deux cas de figures. D'une part, lorsque sa valeur est supérieure à 1% de la valeur des délais que l'on ambitionne d'analyser (dans notre étude 10ms), et d'autre part lorsque son existence peut entraîner des changements dans l'ordre des conséquences d'événements.

3.1. Structure du modèle

Parmi les contraintes que nous nous sommes fixées concernant le modèle il y a l'invariance structurelle vis-à-vis des différentes architectures de commande et vis-à-vis des différentes analyses sur une même architecture. Les modèles de deux architectures différentes ne doivent différer que par le marquage initial de leurs places et la pondération d'un nombre limité de leurs arcs. C'est l'idée du modèle paramétrable, ce qui permettra de réduire considérablement le temps d'adaptation et de mise au point du modèle de toute nouvelle architecture. De la même façon une nouvelle analyse sur un modèle d'architecture existant ne doit entraîner que le changement d'un marquage initial.

Les possibilités de modélisation de CPNtools vont ainsi être pleinement mises à contribution. En particulier la représentation hiérarchique d'un réseau de Petri va accueillir la nature modulaire des applications (assemblage de constituants, composés- d'autres constituants, eux-mêmes composés de composants logiciels). La structure de notre modèle est présentée dans la figure 1. Chaque noeud de l'arborescence est un sous-modèle qui correspond à une transition de substitution du modèle père.

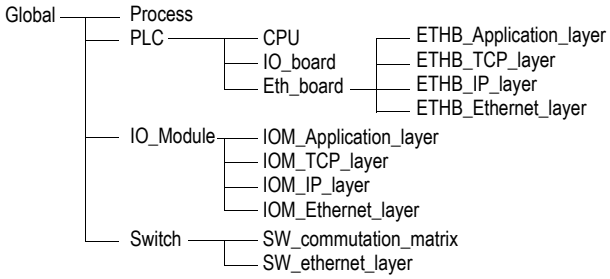

Figure 1. Structure arborescente du modèle

La figure 2 montre la racine de la représentation hiérarchique du modèle d'architecture. L'interprétation de ce niveau de modélisation se fait donc comme suit. Le processus dont l'activité est modélisée par la transition Process génère des événements à destination de la commande en produisant des jetons dans la place Process_event. Le processus est lui-même sollicité par des événements issus des automates programmables (transition PLC) et des modules d'entrées et de sorties déportés (transition IO_module). À l'intérieur de l'architecture de commande, les échanges d'information entre les composants de l'architecture Ethernet commutée sont modélisés par des jetons de couleur ETHFRAME dans la place Ethernet_frame.

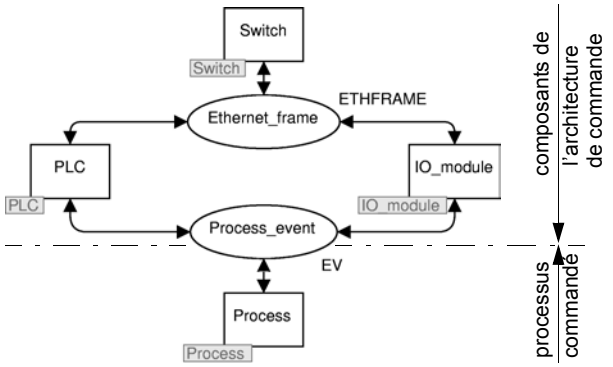

Figure 2. Niveau racine du modèle

Les places jouent le rôle d'entrées et de sorties pour les sous-modèles. Par exemple, le sous-modèle de la transition de substitution PLC est présenté sur la figure 3. On y retrouve les deux places Process_event et Ethernet_frame avec le logo I/O pour indiquer leur rôle particulier dans la hiérarchie de représentation.

3.2. Coloration des jetons

Les couleurs retenues pour les jetons qui représentent des événements et des trames Ethernet sont toutes les deux construites à partir d'une couleur fondamentale pour notre modèle qui représente une transmission d'information élémentaire. Nous l'avons nommée TICKET en référence au trajet que parcourt cette information élémentaire dans l'architecture.

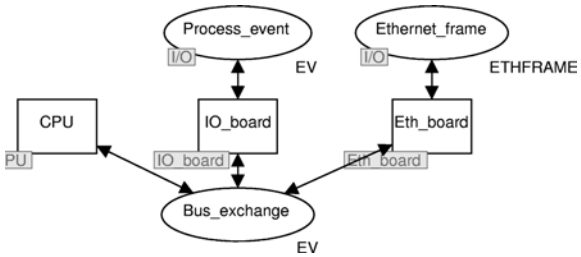

Figure 3. Sous-modèle PLC

```
color TICKET = product
   ID * SRC * DEST * ROUTE * TYPE * DATA timed;
```

Les jetons de la couleur TICKET sont des sextuplets enrichis d'un marqueur temporel avec principalement un identifiant ID, un équipement source SRC, un équipement destination DEST, et un chemin (ROUTE) qui est une liste d'équipements allant de proche en proche de la source à la destination. Le type et le contenu de l'information (TYPE et DATA) sont réservés à un usage ultérieur non développé dans ce papier. La couleur EV pour les jetons modélisant les événements

entre le processus et l'architecture de commande hérite directement de la couleur TICKET :

```
color EV : TICKET;
```

Chaque emplacement dans l'architecture est identifié par un numéro qui permet aux jetons d'être porteur de leur propre destination et de la route pour y parvenir. La figure 4 montre un exemple d'architecture construite autour d'un automate (PLC) en relation avec 9 modules d'entrées-sorties (IO_module) grâce à un réseau Ethernet commuté formé de 2 commutateurs (SWICTH).

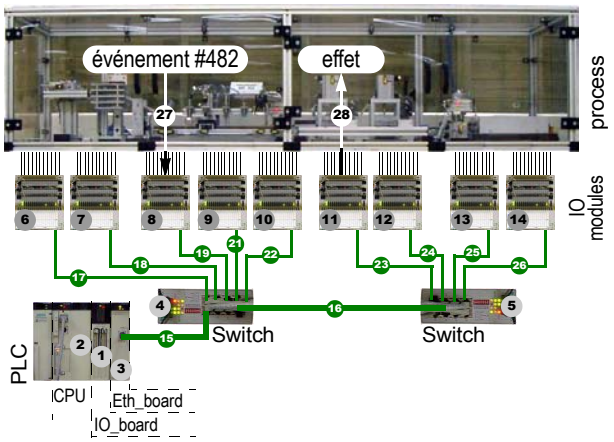

Figure 4. Architecture exemple

Dans cet exemple, lorsque se produit la 482e occurrence de l'événement issu de l'élément du processus identifié par le numéro 27, la logique de traitement de l'architecture est la suivante : le IO_module 8 détecte l'occurrence, la carte ethernet 3 (Eth_board) de l'automate en est informé, par scrutation la carte CPU de l'automate en a connaissance, puis, suite à un traitement, il demande à l'Eth_bord 3 de transmettre au IO_module 11 l'ordre de générer l'effet attendu auprès de l'élément de processus identifié par le numéro 28. Cette logique de traitement spécifique à l'exemple est modélisée pour une bonne part dans le jeton : (482, 8, 28, [8,3,2,3,11,28], 0, «»). La figure 5 présente la séquence de tir qui illustre la propagation du jeton dans le modèle, et en particulier comment la ROUTE du jeton se met à jour au fur et à mesure de sa progression.

On remarquera que les équipements du réseau Ethernet commuté comme les SWITCHs n'interviennent pas dans la ROUTE du jeton. En fait le routage dans le réseau est pris en compte dans la couleur ETHFRAME qui assure deux fonctions : le routage proprement dit, et le transport d'informations par encapsulation.

```
color EVL = list EV timed;
color ETHFRAME = product TICKET * EVL timed;
```

Les jetons de couleur ETHFRAME sont des couples formés d'un ticket qui caractérise la trame Ethernet et d'une liste d'événements où sont encapsulés les événements à transmettre. Sur la figure 5, les marquages successifs de la place Ethernet_frame sont les suivants :

Ma1 = ((0,8,3,[4,15,3],0,«»), [(482,27,28,[3,2,3,11,28],0,«»)])
Ma2 = ((0,8,3,[3],0,«»), [(482,27,28,[3,2,3,11,28],0,«»)])
Mb1 = ((0,3,8,[4,16,5,23,11],0,«»), [(482,27,28,[11,28],0,«»)])
Mb2 = ((0,3,8,[5,23,11],0,«»), [(482,27,28,[11,28],0,«»)])
Mb3 = ((0,3,8,[11],0,«»), [(482,27,28,[11,28],0,«»)])

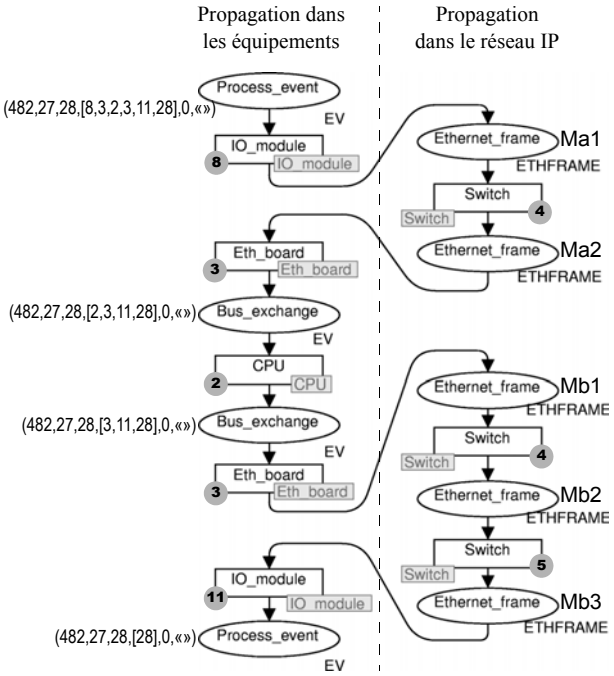

Figure 5. Exemple de propagation d'un jeton EV

3.3. Paramétrage du modèle

Examinons l'aspect paramétrage des modèles au travers des modules d'entrées-sorties déportés (IO_module) sur la figure 6c. La transition Input_filter assure l'acquisition des événements issus du processus. Comme les événements du processus ne doivent pas être lus par n'importe quel module, la condition de transition [isIOM(*loc*)] assure la mise en adéquation des bons événements avec les bons modules. Pour que la transition soit franchissable, il faut que la condition de transition soit vraie, cette dernière étant exprimée sous la forme d'une fonction à une variable. Cette variable *loc* (pour Location) est associée à l'identifiant de l'équipement dans lequel le jeton événement doit se diriger, c'est-à-dire au premier élément de sa liste route. La fonction *isIOM()* est adaptée à chaque nouvelle architecture modélisée. Pour l'exemple de la figure 4, cette fonction doit retourner le booléen vrai chaque fois que la variable *loc* contient l'identifiant d'un module d'entrées-sorties, sinon elle doit retourner le booléen faux.

```
fun isIOM(loc) = case loc of
        6 => true |  7 => true |  8 => true |
        9 => true | 10 => true | 11 => true |
       12 => true | 13 => true | 14 => true |
        _ => false ;
```

Ainsi pour que la transition Input_filter soit franchissable il faut qu'il existe un jeton dans la place Process_event de la forme (*id*, *src*, *dest*, *loc*::*route*, *t*, *data*) avec *loc* tel que isIOM(*loc*) retourne le booléen vrai. La notation *loc*::*route* a pour effet d'associer le contenu d'une liste à deux variables distinctes. La variable *loc* contiendra le premier élément de la liste, tandis que la variable *route* contiendra la liste privée de son premier élément. Lorsque la transition est franchie, le jeton de la forme (*id*, *src*, *dest*, *loc*::*route*, *t*, *data*) est supprimé de la place Process_event tandis qu'un jeton identique est alors ajouté dans la place P1. Ceci modélise la prise en compte d'un événement par un module d'entrée-sorties.

3.4. Modélisation du temps

En plus de disposer d'une condition de transition paramétrée, la transistion Input_filter est temporisée par une durée elle-même paramétrée *@+IOM_Ifilter(loc)*. Lors du franchissement de cette transition, le marqueur temporel des jetons produits sera augmenté de la valeur *IOM_Ifilter(loc)*. Ces jetons ne pourront plus franchir de transitions avant que la valeur courante du temps simulé soit supérieure ou égale à la valeur de leur marqueur temporel. Cela nous permet de modéliser des temps de traitement, comme ici le retard dû au filtrage des entrées. CPNtools propose une représentation du temps par un nombre entier. L'unité de temps choisie pour notre modèle est la microseconde. Le retard de filtrage des entrées allant de 0,06 à 2,2ms selon les modules, la fonction *IOM_Ifilter(loc)* prend la forme suivante :

```
fun IOM_Ifilter(iom) = case iom of
         6 =>   60 |  7 => 2200 | 8 =>    60 |
         9 => 2200 | 11 =>   60 | 12 => 2200 |
        13 =>   60 | 14 => 2200 ;
```

3.5. Modélisation des protocoles

Les sous-modèles des composants ayant une connexion au réseau Ethernet sont regroupés figure 6. Il s'agit des sous modèles représentant les cartes Ethernet des API (Eth_board), des commutateurs Ethernet (Switch) et des modules d'entrées-sorties déportés (IO_module). La structuration retenue est celle en couche du modèle OSI. La place Eth_frame, en bas de chaque modèle, stocke les jetons représentant des trames Ethernet entre deux composants. Chaque couche est modélisée sous forme d'une transition de substitution, derrière laquelle se trouve son modèle de comportement. La dernière couche de chaque modèle lui est spécifique. En effet, ETHB_Application_layer décrit le processus IO scanning, IOM_Application_layer décrit le serveur Modbus du module, et SWITCH_commutation_matrix décrit le comportement de la matrice de commutation du commutateur Ethernet.

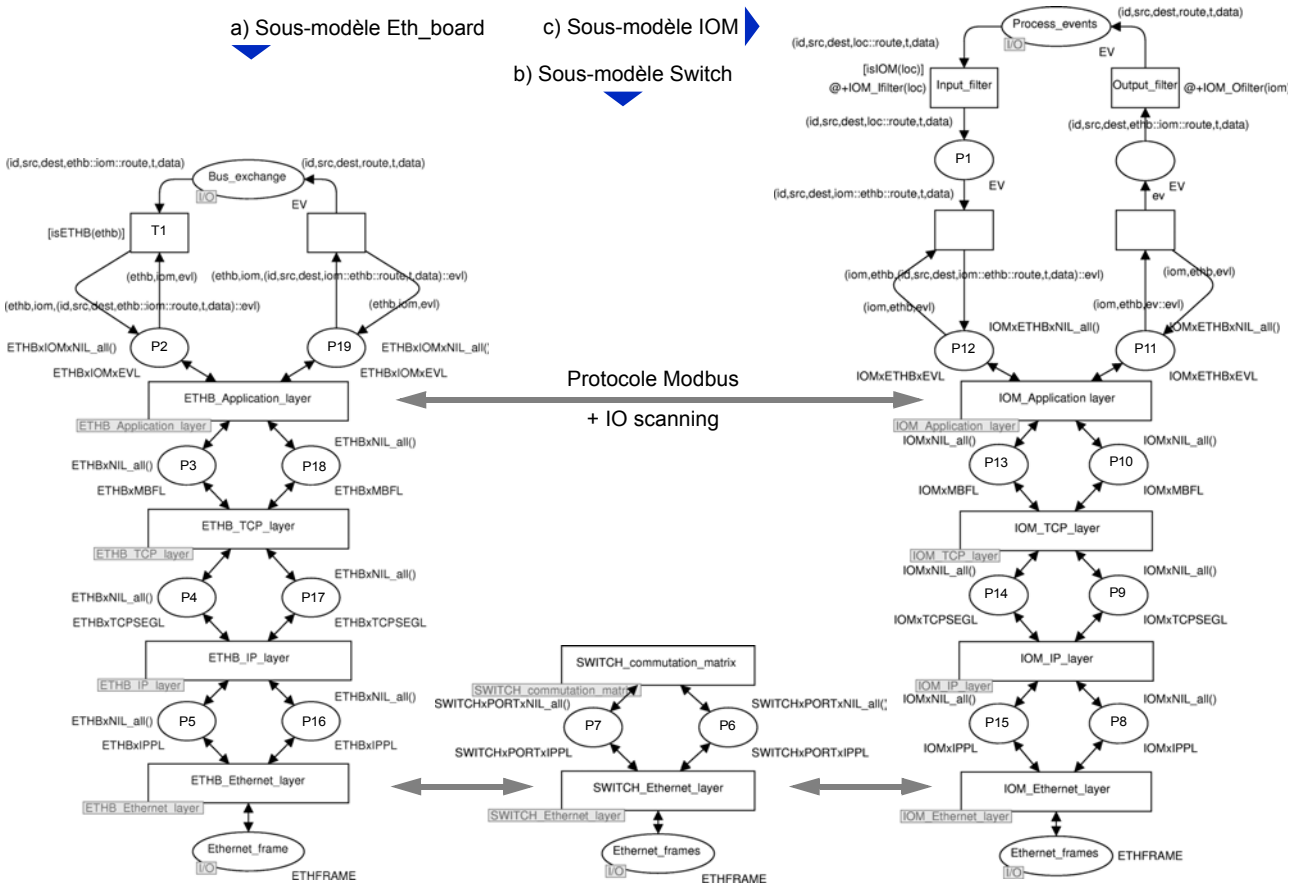

Figure 6. Sous-modèles réseau de Petri structurés en couches

La figure 6 met en évidence la manière dont le protocole Modbus est modélisé au travers des deux sous-modèles Eth_board et IO_module.La transition T1 du sous-modèle Eth_board organise les événements qui se présentent sur le bus des automates en autant de listes qu'il existe de couples de clients-serveurs Modbus. Dans nos architectures les serveurs Modbus sont les modules d'entrées-sorties déportées, alors que les clients sont les cartes Ethernet des automates.

Cette organisation des événements en liste se traduit par les déclarations suivantes :

```
color EVL = list EV timed;
color ETHBxIOMxEVL = product ETHB*IOM*EVL timed;
fun IOM_all() = 1`6++1`7++1`8++1`9++1`10++
                1`11++1`12++1`13++1`14;
fun ETHB_all() = 1`3;
fun ETHBxIOMxNIL_all() =
  ETHBxIOMxEVL.mult(ETHB_all(),IOM_all(),1`nil);
```

Les jetons de la place P2 (couleur ETHBxIOMxEVL) sont des triplets formés d'un client Modbus (une carte Ethernet), d'un serveur Modbus (un module d'entrées-sorties) et d'une liste d'événements à propager du client vers le serveur. Pour le bon fonctionnement des tirs de la transition T1, le marquage initial de la place P2 est constitué de l'ensemble de tous les jetons avec une liste vide (pour CPNtools un liste vide est notée *nil*). Ainsi, lors du tir de T1, un événement dont les deux éléments du sommet de la *route* sont *ethb* et *iom*, est empilé dans la liste (*ethb*,*iom*,*evl*). Le marquage initial de la place P2 est obtenu grâce à la fonction *ETHBxIOMxNIL_all()* qui utilise les fonctions *ETHB_all()* et *IOM_all()* qui retourne respectivement l'ensemble des cartes Ethernet et l'ensemble des modules d'entrées-sorties.

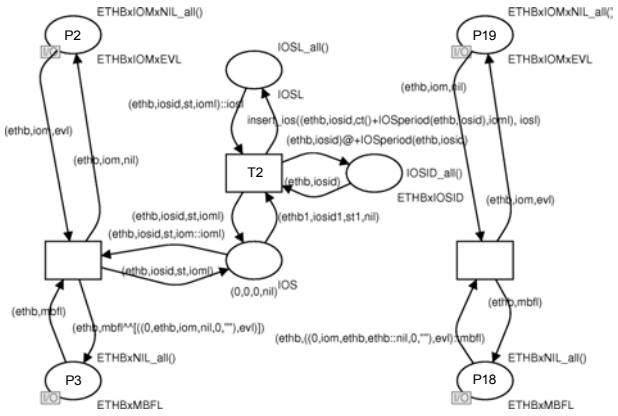

Figure 7. Sous-modèle ETHB_application_layer

Lorsque le moment est venu, le jeton de P2 qui contient la liste les événements à transmettre de *ethb* vers *iom* est encapsulée dans un jeton de couleur MBFRAME, c'est-à-dire dans une trame Modbus. La logique de séquencement des trames (transition T2 de la figure 7) ne sera pas détaillée. Un jeton de couleur MBFRAME est un couple formé d'un *ticket* et d'une liste d'événement. Le ticket d'une trame Modbus indique la source et la destination de la trame. L'identifiant, la route, le type et les données du *ticket* d'une trame ne sont pas utilisés dans la version du modèle présenté dans ce papier, il sont par conséquent inialisés respectivement à 0, nil, 0 et «». Une fois l'encapsulation effectuée, la trame Modbus est empilée dans le jeton de la place P6 qui est une liste de trame Modbus. Suite à cette encapsulation, la trame Modbus va être transmise à la couche application d'un module d'entrées-sorties déportées est utilisant les services des couches de plus bas niveau. Dans la carte Ethernet dont elle est issue, une trame Modbus va successivement occuper les places P3, P4 et P5 pour en sortir par la place Ethernet_frame. Si nécessaire la trame Modbus traversera un à plusieurs commutateurs en empruntant le chemin P6, P7 et Ethernet_frame autant de fois qu'il y aura de commutateurs à traverser. Désormais présent à l'entrée Ethernet du module d'entrées-sorties déportées, la trame Modbus remonte les couches du modèle en occupant successivement les places P8, P9 et P10 où elle se retrouve dans une liste, prête à être traitée par la couche application du module d'entrées-sorties déportées. Ces couches applications utilisent les déclarations suivantes :

```
color MBFRAME = product TICKET * EVL timed;
color MBFL = list MBFRAME timed;
color ETHBxMBFL = product ETHB * MBFL timed;
color ETHBxIOMxEVL = product ETHB*IOM*EVL timed;
color IOMxMBFL = product IOM * MBFL timed;
color IOMxETHBxEVL = product IOM*ETHB*EVL timed;
fun ETHBxNIL_all() =
  ETHBxMBFL.mult(ETHB_all(),1`nil);
fun IOMxNIL_all() =
  IOMxMBFL.mult(IOM_all(),1`nil);
fun IOMxETHBxNIL_all() =
  IOMxETHBxEVL.mult(IOM_all(),ETHB_all(),1`nil);
fun ETHBxIOMxNIL_all() =
  ETHBxIOMxEVL.mult(ETHB_all(),IOM_all(),1`nil);
```

La figure 8 montre le comportement du serveur Modbus des modules. En franchissant la transition une trame Modbus est dépilée de la place P10 et la liste d'événement encapsulée dans la trame est envoyée dans la place P11. Dans ce même franchissement, et en réponse à la requête portée par la trame Modbus une réponse prenant la forme d'une nouvelle trame Modbus est empilée dans la place P13. Dans cette réponse est encapsulé la liste des événements issus du processus qui était présente dans la place P12. On remarquera que les réponses Modbus sont empilées selon une pile de type FIFO. La seule manière de dépiler un élément d'une liste dans CPNtools étant d'extraire le premier élément, il faut dont empiler un élément en l'ajoutant à la fin de la liste pour obtenir un comportement FIFO. Si pour ajouté en élément *el* en début d'une liste *liste* on utilisait la syntaxe *el*::*liste*, pour ajouté *el* en fin de *liste* il faut utilisé la primitive de concaténation de liste et concaténer la liste *liste* à la liste de un élément [*el*] en utilisant la syntaxe *liste*^^[*el*].

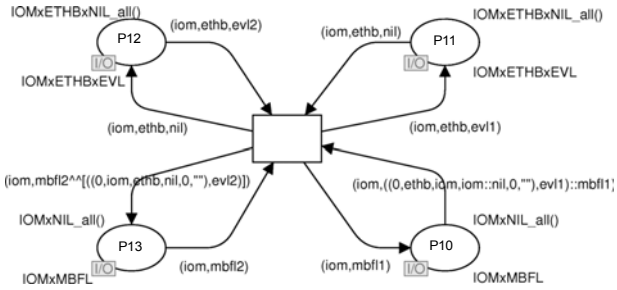

Figure 8. Sous-modèle IOM_application_layer

A partir de la place P13 la réponse suit un parcours analogue à la requête Modbus. Après avoir parcouru les places P14, P15 et Ethernet_frame dans le sous-modèle IOM, la trame emprunte le chemin P6, P7 et Ethernet_frame autant de fois qu'il y aura de commutateurs à traverser, puis remonte vers la couche application du sous-modèle Eth_board en passant par P16, P17 et P18. A l'instar de la couche application des IOM, la couche application des Eth_board extrait de la réponse Modbus située place P18 la liste des événements à transmettre sur le bus de l'automate.

Les modèles des différentes couches TCP/IP et Ethernet se comporte globalement comme des piles FIFO. Cela est suffisant pour les sous-modèles TCP car dans nos architectures les requêtes Modbus ne donne lieu qu'à un segment TCP. Pour les sous-modèles des piles IP il faut noter un enrichissement de la trame Modbus à laquelle on ajoute la route à travers le réseau IP. Cela se fait de manière paramétrique grâce à une fonction à 2 paramètres *IProute(src,dest)*. A titre d'exemple, pour la requête Modbus du client Eth_board n°3 vers le serveur IO_module n°11 on a :

```
IProute(3,11) = [3,15,4,16,5,23,11]
```

4. VALIDATION DU MODÈLE

Pour valider le modèle développé dans la section précédente, nous allons confronter des résultats de simulation avec une campagne de mesure sur une architecture réelle. L'architecture choisie est celle de la figure 4 et l'on s'intéressera à une exigence de réactivité en étudiant le retard de causalité de la figure 9, entre une entrée sur un module déporté (IO_module 11) et une sortie sur la carte de l'API (IO_board 1).

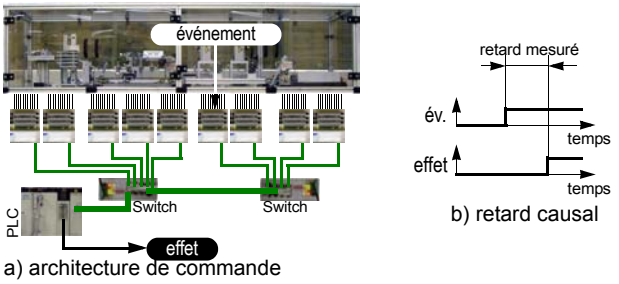

Figure 9. mesure de validation

Ce retard n'étant pas identique pour chaque occurrence de l'événement, la confrontation ne pourra être effectuée que sur un échantillon représentatif. Dans notre cas, la convergence des minima et maxima entre plusieurs échantillons a nécessitée au moins 700 mesures. La simulation de la propagation de 700 occurrences d'événements dans l'architecture a nécessité environ 2 minutes et 20 secondes de calcul pour le logiciel CPNTools sur un PC muni d'un processeur Pentium 4 à 2,4 GHz et de 500 Mo de RAM. Pour les 700 relevés à mesurer sur l'architecture, nous avons utilisé une plateforme dédiée à l'évaluation des performances des SED mise au point au LURPA. Il s'agit de la plateforme brevetée PRISME [2].

La figure 10 donne les histogrammes des échantillons obtenus pour deux configurations de paramètres : la première, très réactive, avec un temps de cycle API de 5 ms et un temps de cycle IO scanning de 10ms, la deuxième, beaucoup moins réactive, avec un temps de cycle API de 100 ms et un temps de cycle IO scanning de 60ms. La validation porte sur trois valeurs significatives des distributions de ces retards qui sont leur minimum, leur moyenne et leur maximum. Le tableau 2 donne ces indicateurs pour les délais mesurés (mes), prévus par simulation (sim) et également l'écart relatif en pourcentage du délai simulé par rapport au mesuré (éc rel).

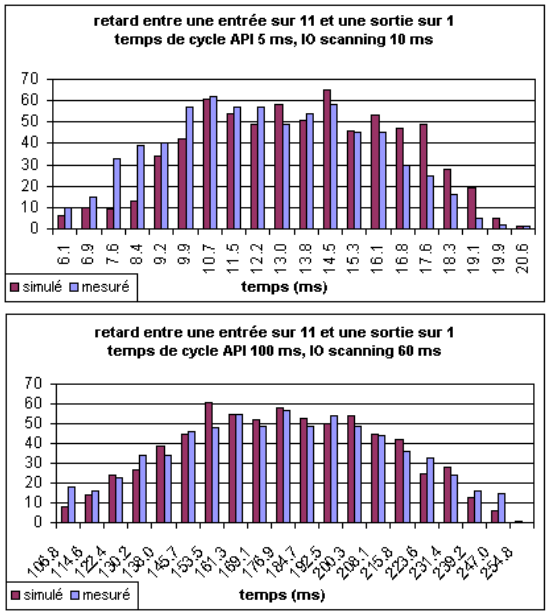

Figure 10. Histogrammes des retards mesurés et simulés

On peut mener une première analyse qualitative sur le profil des distributions montrées sur les histogrammes figure 10. Pour chaque configuration, les échantillons relevés par mesure et ceux prévus par simulation ont été tracés sur le même graphe. En haut figurent les échantillons obtenus pour la première configuration, très réactive, et en bas, ceux de la deuxième configuration. Pour ces deux cas, les formes des distributions des échantillons mesurés et simulés sont très proches, ce qui est satisfaisant. La seconde analyse qui a été menée porte sur le meilleur estimateur de la population, sa

moyenne. Le tableau 2 nous donne l'écart relatif du retard moyen prévu par simulation avec celui mesuré. Les résultats obtenus sont très satisfaisant puisqu'on a seulement 2 à 3% d'écart.

	Paramètres (temps de cycle en ms)		retard causal (en ms)			
	API	IO scanning		min	moy	max
cas 1	5	10	mes	6,6	13,6	21,6
			sim	6,1	13,9	21,4
			éc rel	7,5%	2,2%	0,9%
cas 2	100	60	mes	111,9	181,9	259,8
			sim	106,8	186,7	262,6
			éc rel	4,6%	2.6%	1,1%

Tableau 2: résultats de mesure et simulation

Cependant, les données qui importent à l'automaticien sont les extrema, et plus particulièrement le retard maximum. Ces paramètres doivent être interprétés avec précaution car ils sont difficilement mesurables. En effet, ils dépendent beaucoup de la taille de l'échantillon choisi. Il sera donc nécessaire de définir des campagnes de mesures complémentaires. Il sera également nécessaire de conduire de nouvelles mesures et simulations sur d'autres architectures pour valider complètement le modèle. Néanmoins, cette première confrontation est globalement encourageante.

5. CONCLUSION

Notre travail était de construire un modèle de simulation paramétrable des architectures de commande sur réseau de terrain Ethernet et TCP/IP dans le but d'évaluer leurs performances temporelles. Le modèle construit permet de modéliser des architectures avec API à moniteur périodique communiquant sur Ethernet commuté avec les protocoles Modbus et IO scanning. Le choix des réseaux de Petri colorés temporisés avec hiérarchie s'est avéré être adéquat pour mener à bien l'élaboration du modèle. Ce modèle a ainsi pu être simulé grâce à l'outil CPNTools. Les résultats de simulations ont été validés par confrontation avec des mesures grâce à l'architecture réelle à notre disposition au LURPA. Cette première validation est extrêmement satisfaisante quand à la qualité des performances temporelles obtenues par simulation. En effet, les résultats sont bons aussi bien qualitativement, profils des distributions, que quantitativement, prévision du retard moyen à 2,5%. Il reste maintenant à approfondir la validation du modèle sur différentes architectures. Ensuite, ce modèle pourra être utilisé pour de l'aide à la structuration et au dimensionnement des architectures de commande via Ethernet et TCP/IP. Enfin, il est prévu de prendre en compte les réseaux Ethernet segmentés par l'introduction d'un modèle de routeur IP.

REFERENCES

[1] P. Bhowal, R. Mall and A. Basu. Estimating micro-PLC execution time for time critical system design. *Journal of Systems Architecture*, 45(14):1245-1248, 1999.

[2] B. Denis, O. De Smet, J.-J. Lesage, J.-M. Roussel. Dispositif et procédé d'analyse de performances et d'identification comportementale d'un système en tant qu'automate à événements discrets et finis. Brevet N° 01 110 933, Août 2001.

[3] IEEE standard, Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) and Physical Layer Specifications. IEEE 802-3, 2000.

[4] J.C.A de Figueiredo and L.M. Kristensen. Using Coloured Petri Nets to Investigate Behavioural and Performance Issues of TCP Protocols. *CPN Workshop*, Aarhus, Denmark, Oct. 1999.

[5] A. Flammini, P. Ferrari, E. Sisinni, D. Marioli and A. Taroni. Sensor integration in industrial environment: from fieldbus to web sensors. *Computer Standards & Interfaces*, 25(2):183-194, 2003.

[6] J.P. Georges, E. Rondeau and T. Divoux. Evaluation des performances d'architectures Ethernet commuté. *TSI 2003,* 2003.

[7] A. Koubaa and Y. Song. Evaluation de performances d'Ethernet commuté pour des applications temps réel. In *10th Int. Conf. on Real Time and Embedded Systems*, Paris, France, 2002.

[8] K.C. Lee and S. Lee. Performance evaluation of switched Ethernet for real-time. *Computer Standards & Interfaces,* (24):411- 423, 2002.

[9] *Modbus Messaging Implementation Guide V1*, May 8, 2002.

[10] J. Postel (Eds.). *RFC 793, Transmission Control Protocol*. Information Sciences Institute, University of Southern California, September 1981.

[11] J. Postel (Eds.). *RFC 791, Internet Protocol*. Information Sciences Institute, University of Southern California, January 1981.

[12] A. V. Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen, K. Jensen. CPN Tools for Editing, Simulating, and Analysing Coloured Petri Net. *LNCS,* 2679:450 - 462, 2003.

[13] S. Rüping, E. Vonnahme, J. Jasperneite. Analysis of Switched Ethernet Networks with different Topologies used in Automation Systems. *Proc. of the Fieldbus Technology Conference* (FeT'99), 351-358, Magdeburg, Springer-Verlag, 1999.

[14] S. Vitturi. Some features of two fieldbuses of the IEC 61158 standard. *Computer Standards & Interfaces*, 22 (3):203-215, 2000.

[15] S. Vitturi. DP-Ethernet: the Profibus DP protocol implemented on Ethernet. *Computer Communications*, 26:1095-1104, 2003.