Évaluation des bornes des performances temporelles des Architectures d'Automatisation en Réseau par preuves itératives de propriétés logiques

Silvain Ruel

N° ENSC-2009/168

THESE DE DOCTORAT
DE L'ECOLE NORMALE SUPERIEURE DE CACHAN

Présentée par

Monsieur Silvain RUEL

pour obtenir le grade de

DOCTEUR DE L'ECOLE NORMALE SUPERIEURE DE CACHAN

Domaine :
ELECTRONIQUE–ELECTROTECHNIQUE–AUTOMATIQUE

Sujet de la thèse :

Evaluation des bornes des performances temporelles des Architectures d'Automatisation en Réseau par preuves itératives de propriétés logiques

Thèse présentée et soutenue à Cachan le 09 juillet 2009 devant le jury composé de :

Jean-Louis BOIMOND	Professeur - Université d'Angers - LISA	Rapporteur
Thierry DIVOUX	Professeur - Université de Nancy - CRAN	Rapporteur
Hassane ALLA	Professeur - Université de Grenoble - GIPSA	Examinateur
Gaëlle MARSAL	EDF R&D	Invitée
Olivier de SMET	Maître de conférences - CNAM Paris - LURPA	Encadrant
Jean-Marc FAURE	Professeur - SUPMECA Paris - LURPA	Directeur de thèse

Laboratoire Universitaire de Recherche en Production Automatisée
(ENS CACHAN / EA 1385)
61, avenue du Président Wilson - 94 235 Cachan cedex

Remerciements

La durée d'une thèse est une performance temporelle qu'il est parfois difficile d'estimer a priori et qui dépend de nombreux "composants" interagissant entre eux. Pour des raisons de productivité (poursuite de la carrière) et de sécurité (santé mentale du doctorant), il est préférable qu'elle n'excède pas une borne supérieure fixée à trois ans. Ces quelques mots sont donc là pour remercier tous ceux qui m'ont permis de mener à bien ce challenge.

Je tiens donc tout particulièrement à remercier le Professeur Jean-Marc Faure pour avoir assumé la direction de ma thèse. Les échanges de données et les nombreuses synchronisations que nous avons pu avoir, malgré la présence de nombreux processus concurrents, ont été primordiaux dans le respect des délais que nous nous étions fixés en commençant cette thèse.

Par ailleurs, j'ai été encadré par Olivier de Smet que je tiens à remercier pour ses compétences scientifiques et techniques mais aussi pour ses qualités humaines.

Il m'est impossible de ne pas remercier Bruno Denis qui, même s'il n'intervenait pas dans mon encadrement direct, m'a bien aidé tout au long de ces trois années grâce à ses grandes compétences et à sa disponibilité.

J'ai également une pensée pour tous les membres du projet SIMOP qui m'ont "forcé" à progresser au début de ma thèse.

Je tiens également à remercier les Professeurs Jean-Louis Boimond et Thierry Divoux pour le temps passé à la relecture de mon mémoire et pour leur participation à mon jury de thèse.

Je remercie également les autres membres du jury pour l'intérêt qu'ils ont porté à mes travaux.

Pour finir, je désire remercier tous ceux qui, de près ou de loin, ont rendu ces trois années les plus agréables possibles. Dans cette très large catégorie de personnes (que ceux que je ne cite pas ne m'en veuillent pas, ce serait bien trop long), se trouvent notamment :

- mes collègues doctorants au LURPA comme Yann, Guillaume et Matthieu grâce à qui le jeudi était très certainement la journée la plus vivante de la semaine dans notre bureau commun.
- mes parents et ma soeur qui m'ont soutenu dans les moments plus difficiles et ont porté de l'intérêt à mes travaux, même s'il est vrai qu'ils ont parfois eu bien du mal à comprendre en quoi ils consistaient.
- Nancy qui m'a soutenu, encouragé et motivé à finir cette thèse même si en sa présence j'ai plutôt tendance à ne plus y penser.

Table des matières

Introduction

Les architectures d'automatisation conçues autour de réseaux de communication de type Ethernet industriel sont de plus en plus répandues, et ceci même pour des applications critiques, telles que la production d'énergie, les transports ou les industries chimiques. Pour de telles applications, qui sont celles visées par ces travaux, il est primordial de pouvoir garantir non seulement que le comportement logique de l'architecture est conforme aux spécifications, mais encore que ses performances temporelles permettent d'assurer la sûreté du processus commandé.

Chaque performance temporelle d'une telle architecture, telle que son temps de réponse, est caractérisée non pas par une valeur unique mais par une distribution de valeurs ; ceci provient des mécanismes de consommation de temps (traitement de données, synchronisation entre processus) au sein de l'architecture qui induisent des délais variables. Il est évident que, lorsque des applications critiques sont envisagées, seules les bornes de cette distribution importent ; il ne suffit pas en effet dans ces cas-là de savoir qu'en moyenne une performance donnée satisfait aux exigences mais bien qu'elle sera toujours supérieure ou inférieure à une valeur exigée.

L'objectif de cette thèse est donc de proposer une méthode d'évaluation de ces bornes qui puisse être utilisée lors de la conception d'une architecture d'automatisation, en s'attachant à garantir :

- l'exhaustivité de l'analyse, de façon à ce que les résultats obtenus soient dignes de confiance ;
- l'applicabilité de cette proposition pour des architectures de taille significative ;
- la précision des valeurs obtenues, qui ne doivent pas trop différer des valeurs mesurables une fois l'architecture réalisée.

Ces contraintes sont indispensables si l'on veut pouvoir, à terme, utiliser les résultats de cette recherche pour des applications industrielles critiques.

Ce mémoire de thèse comporte cinq chapitres qui nous permettent de présenter la problématique scientifique de ces travaux, puis d'exposer nos contributions formelles et méthodologiques, et enfin de valider expérimentalement ces propositions au travers de plusieurs cas d'étude.

Plus précisément, le premier chapitre présente tout d'abord de manière détaillée les systèmes technologiques étudiés : architectures d'automatisation dans lesquelles des contrôleurs logiques communiquent avec des modules d'entrées-sorties déportées via un réseau Modbus TCP/IP. Les performances temporelles de ces architectures sont ensuite définies et reliées aux performances du processus commandé exigées. La seconde partie de ce chapitre est consacrée à une synthèse des résultats de recherches récents dans le domaine. Ces

propositions s'appuient sur des analyses multiples ou fournissent directement un domaine de fonctionnement. Leurs avantages et limitations (exhaustivité de l'analyse non garantie, obtention de majorants/minorants trop pessimistes, passage à l'échelle impossible) respectifs sont recensés ; quelle que soit l'importance de ces contributions, aucune ne répond malheureusement complètement aux contraintes fixées.

Face à ce constat, notre première contribution est présentée dans le second chapitre. Il s'agit d'une méthode d'évaluation des bornes des performances temporelles qui repose sur des preuves itératives de propriétés d'atteignabilité, à l'aide de techniques de model-checking temporisé ; ces propriétés sont définies sur un automate observateur paramétré, dont certaines gardes sont fonction d'un paramètre temporel. A l'issue de chaque itération, les résultats des preuves de propriétés permettent de définir la valeur de ce paramètre pour la prochaine itération, au moyen d'un algorithme de recherche par dichotomie assurant la convergence des itérations.

La mise en oeuvre de cette méthode requiert évidemment de se doter d'un modèle formel de l'architecture d'automatisation étudiée. Le but du chapitre 3 est alors de décrire la construction de ce modèle, basée sur l'instanciation de modèles génériques de composants d'architecture. Ces modèles génériques, sous forme d'automates temporisés, sont ensuite détaillés. Le modèle formel d'une architecture est donc un réseau d'automates temporisés communicants, chacun de ces automates étant une instance d'un modèle générique.

Le passage à l'échelle de la méthode d'évaluation des bornes proposée s'avérant impossible en raison du problème usuel d'explosion combinatoire lié aux techniques de model-checking, une technique d'abstraction de modèles d'architectures est définie dans le chapitre 4. Cette technique comporte deux étapes :

- simplification de la structure du modèle initial d'architecture,
- modification des modèles de composants subsistant dans la structure simplifiée.

L'objectif de la première étape est de supprimer les modèles des composants qui n'influent pas directement sur la performance temporelle étudiée ; la seconde étape vise à prendre en compte dans les modèles de composants conservés l'impact potentiel des modèles supprimés.

Enfin, le dernier chapitre s'intéresse à la validation expérimentale des propositions présentées dans les trois chapitres précédents, sur la base de six cas d'étude de complexité croissante. De manière synthétique, ces études montrent :

- que la technique d'abstraction développée lors de ces travaux augmente très nettement les possibilités de passage à l'échelle,
- que les valeurs de bornes obtenues par preuves itératives de propriétés sur un modèle formel sont très proches de celles mesurées sur une architecture réelle, ce qui valide nos apports.

En conclusion, une synthèse des principaux résultats de ces travaux de thèse est effectuée ; quelques perspectives d'extensions sont enfin proposées.

Chapitre 1

Contexte des travaux

Ce chapitre présente le contexte de ces travaux, tant du point de vue technique que scientifique. Les particularités des systèmes étudiés, à savoir les architectures d'automatisation en réseau basées sur Ethernet industriel, seront tout d'abord présentées. Leurs performances temporelles, en liaison avec les performances attendues du système commandé, seront également détaillées. Puis une synthèse des différentes techniques d'évaluation des bornes des performances temporelles préalablement proposées sera réalisée afin d'en déterminer les possibilités et les limites. Cette synthèse permettra de définir l'objectif de nos travaux.

Sommaire

Ce chapitre va se décomposer en trois parties présentant respectivement, les architectures d'automatisation en réseau (AAR) (partie 1.1), les performances temporelles de ces architectures (partie 1.2) et les méthodes d'évaluation des bornes de ces performances temporelles (partie 1.3).

Dans la première partie, une présentation générale des AAR utilisant Ethernet industriel et la description de la solution technologique particulière retenue pour ces travaux seront réalisées.

La partie suivante permettra de présenter de façon détaillée les différentes performances temporelles qui sont attendues pour une AAR, ainsi que les mécanismes de consommation du temps à l'intérieur des AAR, qui sont les causes de ces performances.

Le but final de ces travaux étant de pouvoir évaluer les bornes des performances temporelles des AAR, différentes méthodes précédemment développées dans cette perspective seront présentées, que ce soit des méthodes basées sur des analyses multiples (comme la simulation) ou basées sur l'évaluation des domaines de fonctionnement des AAR. L'analyse des forces et des faiblesses de ces propositions permettra de définir l'objectif de nos travaux, en conclusion de ce chapitre.

1.1 Architectures d'Automatisation en Réseau à base d'Ethernet industriel

1.1.1 Caractéristiques générales

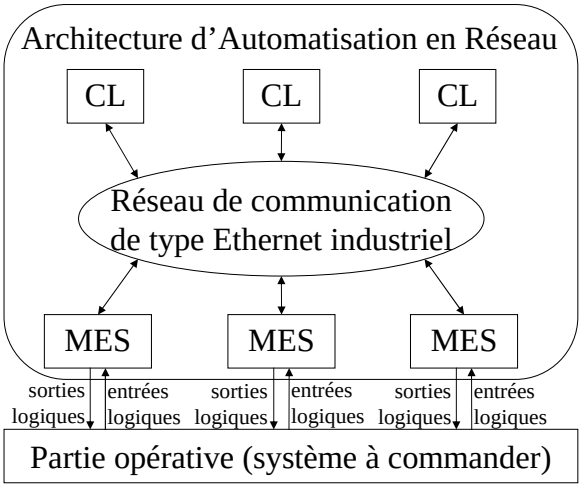


FIG. 1.1 – Structure générale d'une AAR

Comme le montre la figure 1.1, les architectures d'automatisation en réseau (AAR) sont constituées d'un ensemble de composants d'automatisation :

- des contrôleurs logiques CL (automates programmables industriels ou calculateurs industriels),
- des modules d'entrées/sorties (MES)

connectés à un réseau de communication (aussi appelé réseau de terrain) par lequel ces différents composants échangent des données. Le but de ce système est au final de permettre les échanges entre les contrôleurs logiques et la partie opérative.

Les contrôleurs logiques jouent deux rôles dans l'AAR. Ils exécutent un programme qui permet de déterminer le nouvel état des sorties en fonction de l'état antérieur de ses variables et de l'état courant des entrées. Ils doivent également gérer les échanges avec les MES pour récupérer les valeurs des entrées et transmettre les valeurs de sorties. Leur fonctionnement, ainsi que l'organisation des échanges de données seront détaillés dans la partie 1.2.2.1, uniquement pour la solution technique présentée dans la partie 1.1.2.

Bien que des réseaux de terrain dédiés aient été développés dans le passé, notamment FIP, Modbus ou Profibus, la (quasi-)totalité des AAR développées à l'heure actuelle utilise un réseau de type Ethernet industriel, et ceci même pour des utilisations qui peuvent être considérées particulièrement à risques, comme dans la production d'énergie ou l'industrie chimique.

Différentes solutions utilisant Ethernet industriel existent et offrent des caractéristiques technologiques ainsi que des performances très variables. Deux classifications ont été proposées pour classer ces différentes solutions techniques. La première, notamment utilisée par [Neu07], est basée sur la représentation OSI (Open Systems Interconnection) proposée par [Zim88] (figure 1.2.a). Elle permet de classer les différentes solutions du point de vue des choix technologiques faits pour leur réalisation. Une deuxième classification, notamment utilisée par [FFV06], est basée sur les performances de ces réseaux : périodicité de la scrutation des MES et variation de cette périodicité (gigue). Elle permet de classer les différentes solutions en fonction de leurs performances et non des choix technologiques effectués pour y parvenir. La suite de cette discussion s'appuie sur la première classification basée sur la représentation OSI.

	a)	b)	c)	d)
7	Application	Modbus	Powerlink	Profinet
6	Présentation	Non utilisée	Non utilisée	Non utilisée
5	Session	Non utilisée	Powerlink Slot Communication Network Management	Non utilisée
4	Transport	TCP		Isochronous Real Time
3	Réseau	IP		
2	Liaison de données	CSMA/CD	CSMA/CD	Découpage temporel
1	Physique	10 ou 100baseT	10 ou 100baseT	10 ou 100baseT

FIG. 1.2 – La représentation OSI a), appliquée à Modbus TCP/IP b), à Ethernet PowerLink c) et à PROFINET IRT d)

La représentation OSI (figure 1.2.a) permet de représenter les réseaux de communication de façon abstraite et peut être utilisée pour tous les réseaux de communication. Cette représentation est composée de sept niveaux allant du niveau physique, le plus bas, au niveau application, le plus haut. Un niveau dépend du niveau immédiatement inférieur

et le complète mais est indépendant des niveaux supérieurs. Il est possible de classer les réseaux de terrain utilisant Ethernet industriel en trois catégories en fonction du niveau OSI sur lequel s'appuie le protocole de communication temps réel utilisé ([Fel05]) :

- **Les solutions basées sur TCP/UDP** (couche transport) telles que Modbus TCP (figure 1.2.b), ETHERNET/IP, P-NET et VNET/IP,
- **Les solutions basées sur le niveau MAC** (couche liaison de données) telles que Ethernet PowerLink (figure 1.2.c), TCnet, EPA et PROFINET CBA,
- **Les solutions utilisant des composants réseau spécifiques ou une version modifiée d'Ethernet**, comme SERCOS, ETHERCAT, PROFINET IRT (figure 1.2.d).

Globalement, plus le niveau sur lequel s'appuie le protocole temps réel sera bas dans le modèle OSI, plus il pourra être spécifique et meilleures pourront être les caractéristiques du réseau (fréquence de scrutation des MES, temps de traversée du réseau, gigue, ...). En contrepartie, la solution sera plus chère à développer et moins ouverte pour d'éventuels trafics de données autres que ceux utiles pour la commande du système. Par exemple, une solution basée sur TCP/UDP peut cohabiter sur un même réseau avec des échanges de type FTP, HTTP, ..., utilisés pour de la navigation web par exemple.

1.1.2 AAR basées sur Modbus TCP/IP

Modbus TCP/IP[1] est une solution technique industrielle proposée par Schneider Electric. Elle a été développée à partir de Modbus (marque déposée par Modicon) qui est un protocole d'échange de données de type client/serveur prévu initialement pour des liaisons série.

Modbus TCP/IP (figure 1.2) est un protocole d'échange de messages positionné au niveau 7 dans la représentation OSI (niveau application) et utilisant TCP/IP. Il utilise les requêtes Modbus standards (lecture, écriture et lecture/écriture) qui sont encapsulées dans une trame Ethernet. Il permet des communications de type **client/serveur** entre les différents composants connectés au réseau, les contrôleurs logiques étant les clients, et les MES les serveurs. Ces échanges de données sont ordonnés par un algorithme de scrutation périodique des modules d'entrées/sorties appelé IOscanning. La période de ces scrutations est configurable. Une présentation plus détaillée des mécanismes d'échanges de données dans ce type d'AAR sera effectuée dans la partie 1.2.2.

Dans le cadre de ces travaux, le choix de Modbus TCP/IP comme solution Ethernet industriel a été dicté par deux observations. Tout d'abord, Modbus TCP/IP est fortement connu et implanté dans le milieu industriel. Par exemple, la revue scientifique "Mesures" de septembre 2008 considère que 30% des solutions Ethernet industriel réellement utilisées dans l'industrie d'automatisation sont basées sur Modbus TCP/IP[2], ce qui le place deuxième derrière Ethernet/IP (48%) et devant Profinet (7%). D'un autre côté, cette solution technique est intéressante parce que se trouvant en porte à faux entre les utilisations "temps réel classe 1" (temps de cycle réseau de l'ordre de 10 à 100 ms et variable autour de la valeur moyenne, utilisation pour l'automatisation de processus) et "temps réel classe

[1] www.modbus.org

[2] http ://www.mesures.com/actu-livre-technique-3867.html, consulté le 27/04/09

2" (temps de cycle réseau de l'ordre de 1 et 10 ms et très constant, utilisation pour le contrôle), (selon la classification de [Neu07]). En effet, la gestion du trafic des données n'étant pas basée sur le niveau 2 (niveau liaison de données) de la représentation OSI, Modbus TCP/IP ne devrait pas pouvoir entrer dans la catégorie des solutions "temps réel classe 2" mais d'un autre coté, son temps de cycle minimal de 10 ms semble lui permettre cette utilisation.

1.2 Performances temporelles des AAR

Avant d'évaluer les performances d'une AAR, comme celle représentée sur la figure 1.3, il est nécessaire de déterminer quelles sont celles qui sont réellement intéressantes pour l'utilisateur. Elles seront présentées dans la partie suivante en relation directe avec les performances attendues du système à commander. Par la suite, il sera nécessaire de rechercher quels comportements, au sein de l'AAR, influencent ces performances temporelles. Les différents mécanismes consommateurs de temps seront explicités pour tous les composants de l'AAR (contrôleurs, réseau, MES).

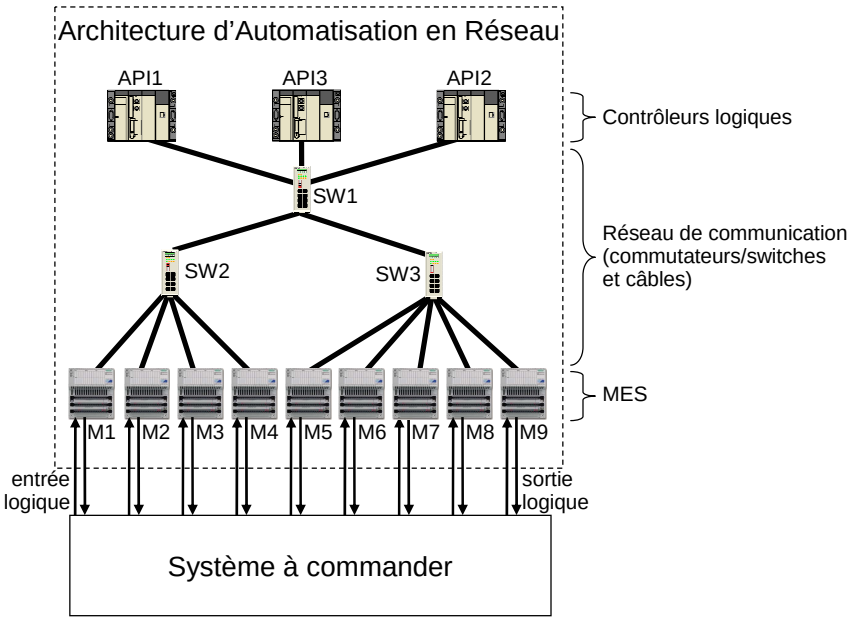


FIG. 1.3 – Exemple d'une architecture d'automatisation en réseau

1.2.1 Performances temporelles globales d'une AAR

Dans un système automatisé, les performances attendues du système à commander sont liées à des grandeurs physiques variées (position, vitesse, masse, volume, pression, temperature, ...). D'un autre coté, un système de commande ne peut que générer des signaux de sortie en fonctions de signaux d'entrée et de son état courant et donc ses performances ne peuvent être évaluées qu'en mesurant des délais entre différents évènements.

Ces délais (au niveau de la commande) représentent donc des grandeurs physiques (au niveau de la partie opérative) et les variations de ces délais correspondent aux intervalles de tolérance qu'il faut accepter sur ces grandeurs physiques.

Plus précisément, les performances temporelles des AAR doivent caractériser les **propriétés de réactivité, de synchronisation et la capacité de détection** de ces AAR. Trois performances temporelles permettent de quantifier ces propriétés et d'obtenir de façon indirecte les performances attendues du système à commander :

- La position d'arrêt d'un convoyeur par rapport à la position théorique, le volume réel du remplissage d'un réservoir par rapport à la consigne, ..., toutes ces performances, liées à la **réactivité** du système, peuvent être ramenée au **temps de réponse** de la commande, c'est à dire au délai qui s'écoule entre la variation d'une entrée (évènement déclencheur) et sa conséquence sur le système à commander (émission de la consigne de sortie correspondante).
- La **synchronisation** de plusieurs processus en parallèle est importante pour la sécurité et la productivité du système. Par exemple, une mauvaise synchronisation entre un bras manipulateur et un convoyeur à bande peut amener à essayer de saisir la pièce transportée avant que le convoyeur ne soit arrêté (bras manipulateur en avance) alors qu'ajouter une période d'attente trop importante entre l'arrêt du convoyeur et la saisie de la pièce par le bras va réduire la productivité du système. Cette synchronisation sera évaluée au travers de la **différence des temps de réponse** entre les émissions des sorties commandant les différents processus et provoquées par le même évènement d'entrée.
- Il est également intéressant de savoir quelle est la **durée minimale d'un signal d'entrée** pour qu'il soit pris en compte par la commande. Du point de vue de la partie opérative, cela peut correspondre à la **capacité de détection** d'un objet qui coupe un faisceau lumineux, passe devant un capteur, ..., sans pour autant s'arrêter. Cette durée pourra par exemple imposer la vitesse maximale de déplacement de l'objet ou la portée du capteur, pour être sûr de bien détecter.

Pour l'évaluation de ces trois performances, deux points sont à prendre en compte.

Tout d'abord, une performance temporelle d'une AAR ne peut pas être réduite à une seule valeur, ce point ayant déjà été abordé par de nombreux travaux ([Mar06], [LF07] et [JO08]). Il faut donc considérer qu'une performance temporelle correspond à une distribution de valeurs (figure 1.4) dont l'étendue englobe toutes les évolutions possibles du système. L'origine de ces variations sera détaillée dans la partie suivante.

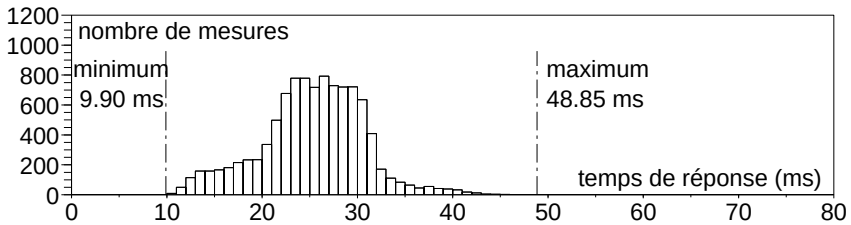


FIG. 1.4 – Exemple d'une distribution de temps de réponse ([DRF+07])

Une distribution peut être caractérisée par différents paramètres, comme sa moyenne et son écart type. Pour des systèmes de commande, notamment ceux utilisés pour des **applications critiques particulièrement à risques** (commande d'une turbine dans une centrale de production d'énergie, par exemple) il apparaît que cette représentation n'est pas la plus pertinente. En effet, ce qui importe, ce n'est pas tant de savoir si globalement le système va bien fonctionner mais bien de garantir que pour toutes les évolutions possibles du système, le comportement correspond bien au fonctionnement attendu. Dans cette optique, ces travaux vont s'attacher à déterminer **les bornes des performances temporelles des AAR** avec le souci permanent de toujours englober tous les comportements possibles de l'AAR étudiée.

1.2.2 Mécanismes de consommation du temps au sein des AAR

Les mécanismes de consommation de temps dans les AAR peuvent se décomposer en deux grandes familles, ceux liés à un seul composant (CL, MES, composants réseau) et ceux liés aux synchronisations entre les composants. Le fonctionnement de chaque composant sera présenté en premier lieu (indépendamment des autres composants) avant de s'intéresser aux interactions entre composants.

1.2.2.1 Fonctionnement des contrôleurs logiques

Les contrôleurs logiques doivent réaliser deux fonctions distinctes : l'exécution du programme de commande et la scrutation des MES. Ils sont de ce fait décomposés en deux entités : le processeur de calcul (noté CAL), qui exécute le programme de commande, et la carte de communication (notée COM), qui communique avec les MES.

Le processeur de calcul a un comportement cyclique et la carte de communication un comportement périodique. Ces deux cycles, appelés respectivement cycle de calcul et cycle d'IOscanning ne sont pas synchronisés. Les documents constructeurs n'étant pas très explicites sur les échanges entre le processeur de calcul et la carte de communication, des essais ont été réalisés au LURPA afin de vérifier quand le processeur de calcul lit ses entrées, quand il émet ses sorties, quand la carte de communication met à disposition du processeur les données reçues des MES et quand elle récupère les nouvelles valeurs de sorties pour les envoyer aux MES. Ces essais ont été réalisés avec un automate programme industriel (API) de la marque Modicon, composé d'un processeur de calcul TSX premium (TSX57203) et d'une carte de communication Ethernet ETY5102.

Comme le montre la figure 1.5, il y a quatre points à vérifier pour savoir si :

- le processeur de calcul copie toutes les entrées avant d'exécuter le programme de commande (cas 1) ou les prend en compte au cours du calcul lorsqu'il en a besoin (cas 1'). Il convient de noter que dans le deuxième cas, la phase de lecture n'existe plus pour le processeur de calcul.
- le processeur de calcul émet toutes ses sorties à la fin du programme de commande (cas 2) ou les émet au cours du calcul (cas 2'). Il convient de noter que dans le deuxième cas, la phase d'écriture n'existe plus pour le processeur de calcul.

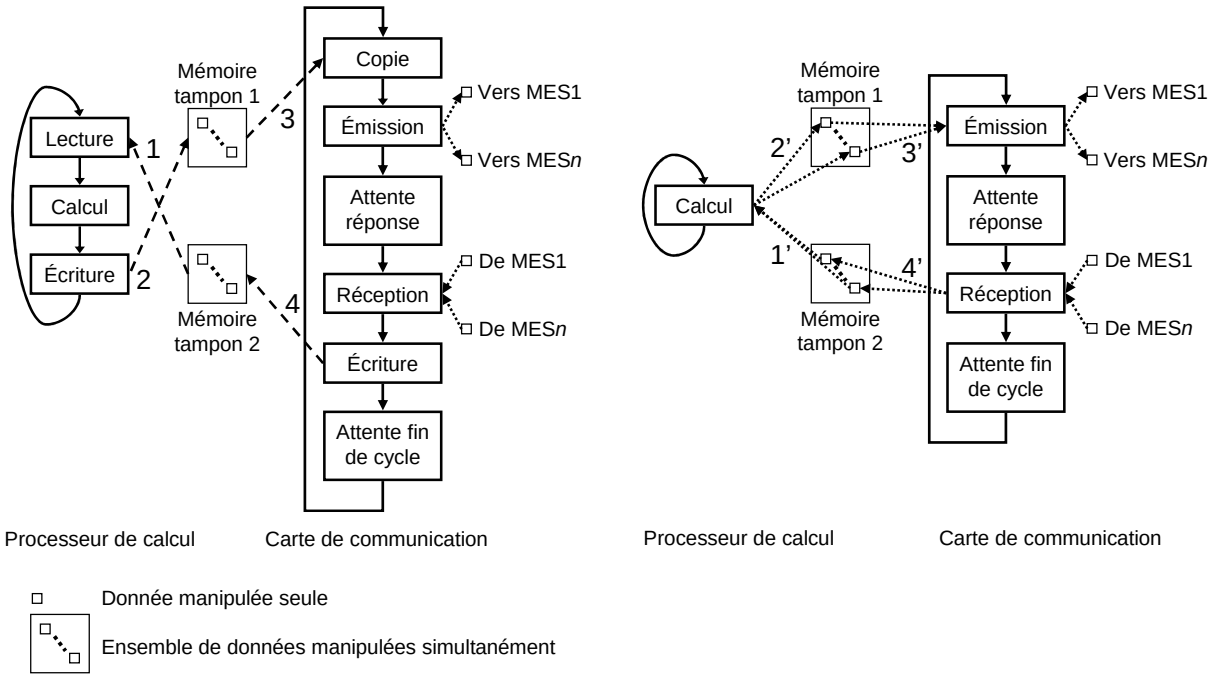


FIG. 1.5 – Possibilités d'échanges de données entre le processeur de calcul et la carte de communication

- la carte de communication prend en compte les nouvelles sorties à envoyer aux MES avant la phase d'émission (cas 3) ou au cours de cette phase (cas 3'). Il convient de noter que dans le deuxième cas, la phase de copie n'existe plus pour la carte de communication.
- la carte de communication met à disposition du processeur de calcul les nouvelles valeurs des entrées à la fin de la phase de réception (une fois qu'elle sont toutes arrivées) (cas 4) ou au fur et à mesure de leur arrivée (cas 4'). Il convient de noter que dans le deuxième cas, la phase d'écriture n'existe plus pour la carte de communication.

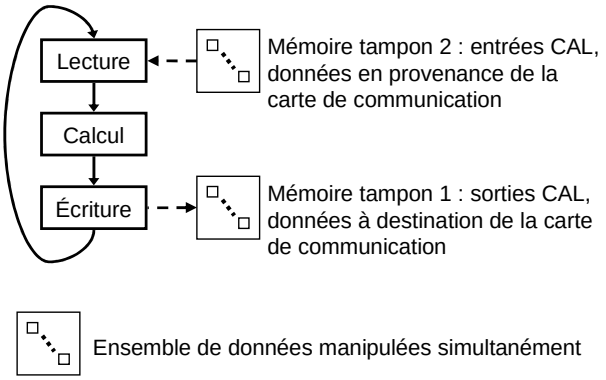


FIG. 1.6 – Fonctionnement d'un processeur de calcul

Il résulte de ces essais, détaillés en annexe A, que le processeur de calcul (figure 1.6)

a un fonctionnement cyclique qui se décompose en trois étapes : lecture des valeurs des entrées dans la mémoire tampon 2 située entre la carte de communication et le processeur de calcul (toutes ensemble), calcul des nouvelles valeurs des sorties, écriture des nouvelles valeurs des sorties dans la mémoire tampon 1 située entre le processeur de calcul et la carte de communication (toutes ensemble).

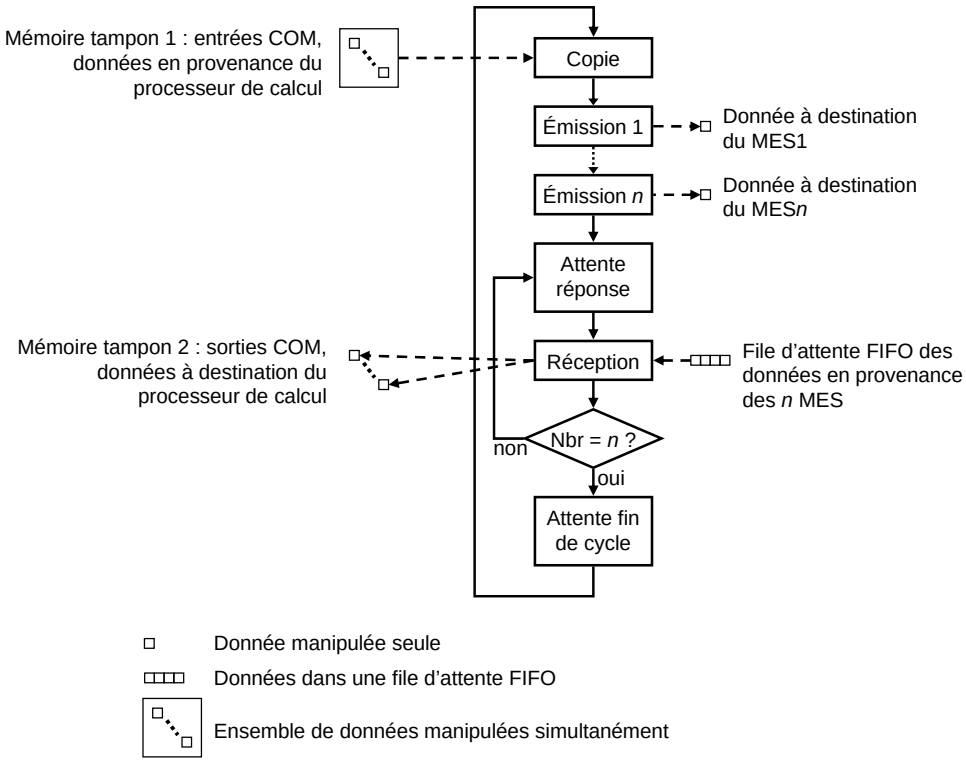


FIG. 1.7 – Fonctionnement d'une carte de communication

La carte de communication (figure 1.7) a, pour sa part, un comportement périodique. Elle commence par copier toutes les sorties calculées par le processeur de calcul de la mémoire tampon entre le processeur de calcul et la carte de communication dans sa mémoire propre puis elle émet les requêtes (une à une) aux n MES auxquels elle est connectée dans un ordre fixé par l'utilisateur. Après ces émissions, elle attend les réponses des MES, les traite une à une dans leur ordre d'arrivée et écrit les nouvelles valeurs des entrées au fur et à mesure dans la mémoire tampon 2 située entre la carte de communication et le processeur de calcul. Une file d'attente FIFO gère la réception des réponses pour les traiter dans leur ordre d'arrivée et n'en perdre aucune. Une fois toutes les réponses reçues, il y a une période d'attente pour garder un temps de cycle constant. Il est important que la durée du cycle d'IOscanning soit bien configurée pour que toutes les réponses puissent bien être reçues avant la fin du cycle de la carte de communication.

1.2.2.2 Fonctionnement du réseau

Le réseau est constitué, dans notre cas, uniquement de câbles Ethernet et de commutateurs fonctionnant en 100baseT ; ces commutateurs ne permettent pas de définir une

classification de service. Tous les éléments du réseau supportent des flux de données en full duplex avec un débit maximal de 100 Mbit/s. Il convient de souligner de plus que ces travaux s'intéressent uniquement à des AAR dans lesquelles des CL communiquent avec des MES. Aucun autre flux (communications entre CL, flux émis par ou envoyé à un composant externe à l'AAR) ne traverse le réseau.

Différentes études ont été menées dans la communauté pour déterminer l'impact de la charge du réseau sur les temps de traversée, l'impact des collisions sur le débit réel de l'architecture ([Geo05], [OB09], ...). Des modèles fins des composants du réseau, notamment les commutateurs, ont bien évidemment été réalisés pour mener à bien ces études. D'autre part, les travaux relatés dans [Mar06] ont montré que, **pour la classe d'AAR que nous considérons**, les mécanismes de consommation du temps dans le réseau ont peu d'influence sur les performances temporelles ; d'autres mécanismes, tels que ceux présentés dans la partie précédente, interviennent de façon prépondérante. Afin de proposer une modélisation du réseau adaptée à notre objectif, nous avons donc réalisé une campagne d'essais décrite ci-après.

Analyse expérimentale de l'influence de la charge des composants réseau sur une performance temporelle

Des expériences ont été menées par [DRF+07] sur une AAR constituée de 2 API et 9 MES et sur laquelle le temps de réponse de l'API1 (délai entre l'apparition l'évènement d'entrée E et l'émission de la sortie S) est évalué, pour mesurer l'impact qu'ont des trafics de données, annexes à ceux générés par l'AAR, sur ce temps de réponse. Pour ces expériences, les API, de marque Modicon, étaient composés d'un processeur de calcul TSX premium (TSX57203) et d'une carte de communication Ethernet ETY5102 et les commutateurs étaient des composants Schneider Electric de type 499NES18100.

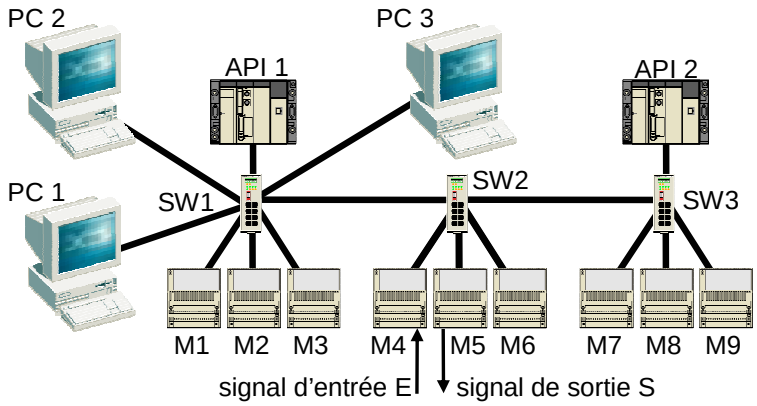


FIG. 1.8 – Analyse expérimentale de l'influence de la charge d'un commutateur sur le temps de réponse

Deux expériences ont été réalisées. La première consiste à évaluer l'impact sur le temps de réponse de ces trafics dans le cas où ils transitent par un commutateur (ici SW1) sans passer par les ports utilisés par les flux générés par l'AAR (figure 1.8). Pour cela trois

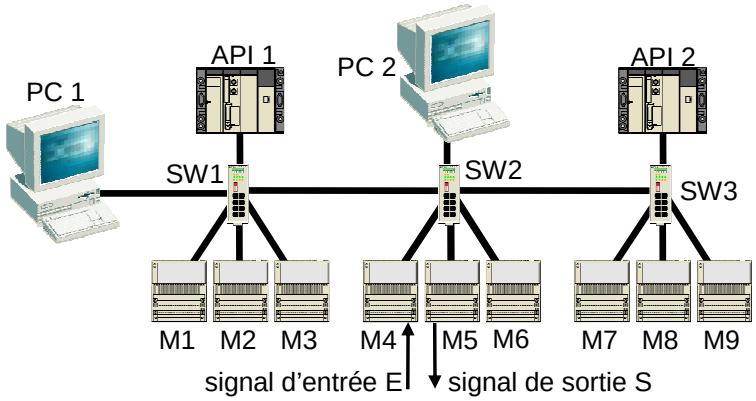


FIG. 1.9 – Analyse expérimentale de l'influence de la charge d'un câble Ethernet sur le temps de réponse

ordinateurs (PC1, PC2 et PC3) sont utilisés pour charger le commutateurs SW1. La seconde consiste à évaluer l'impact sur le temps de réponse de ces trafics dans le cas où ils passent par un câble Ethernet utilisé par les flux générés par l'AAR (figure 1.9). Seulement deux ordinateurs (PC1 et PC2) sont utilisés dans ce cas pour charger le câble entre les commutateurs SW1 et SW2. Dans les deux cas, différentes tailles de trames ont été essayées sans modification du comportement observé.

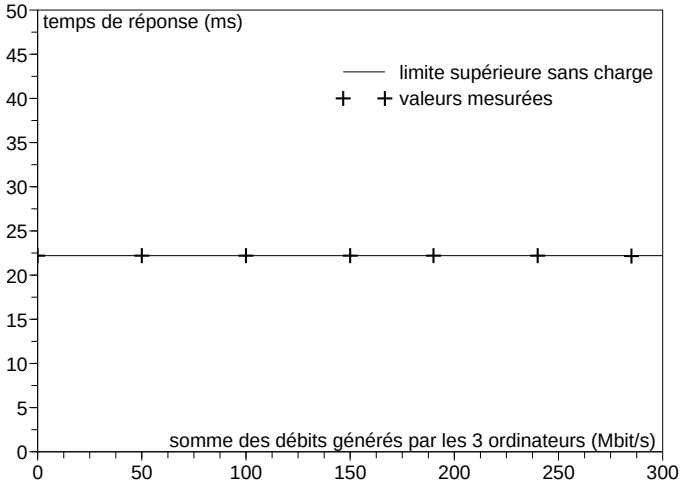


FIG. 1.10 – Temps de réponse en fonction de la charge d'un commutateur

La figure 1.10 montre l'impact sur le temps de réponse occasionné par la charge du commutateur SW1, charge représentée par la somme des trafics générés par les ordinateurs PC1, PC2 et PC3. Il est facile de constater que **le temps de réponse n'a pas été modifié par ces trafics** et ceci même quand chacun des ordinateurs générait un trafic de 95 Mbit/s (limite que les ordinateurs n'arrivaient pas à dépasser).

La figure 1.11 montre l'impact sur le temps de réponse occasionné par la charge du câble Ethernet situé entre le commutateur SW1 et le commutateur SW2, charge représentée par la somme des trafics générés par les ordinateurs PC1 et PC2. Ces trafics ne

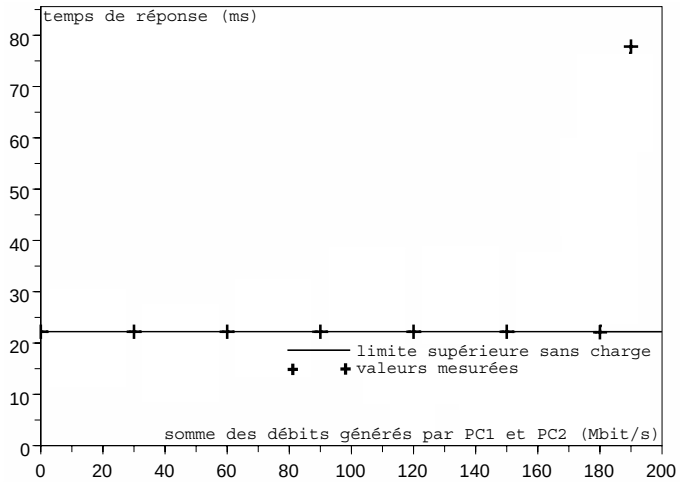


FIG. 1.11 – Temps de réponse en fonction de la charge d'un câble Ethernet

modifient pas le temps de réponse observé tant qu'ils sont chacun inférieurs ou égaux à 90 Mbit/s. Au delà de cette valeur, le temps de réponse est très fortement perturbé puisqu'il est quasiment multiplié par quatre.

Ces résultats peuvent être interprétés de la façon suivante si l'on considère un commutateur comme un ensemble de buffers d'entrée et de sortie reliés par une matrice de commutation. La première expérience permet simplement de valider l'absence de concurrence entre les ports d'entrée/sortie du commutateur. Pour la seconde expérience, la concurence sur un port de sortie peut avoir un impact sur le buffer de sortie du port en question et sur la matrice de commutation. L'expérience ayant été menée avec peu de générateurs de paquets (mais avec chacun des débits élevés), la matrice de commutation n'est donc pas saturée alors qu'il est possible de constater une saturation du buffer de sortie (pour des débits supérieurs à 90 Mbit/s). Un lecteur averti pourra donc s'étonner de l'absence d'une troisième expérience visant à mettre en défaut la matrice de commutation. Cette expérience n'a pas été réalisée, car, dans le cas des AAR étudiées, la configuration des communications entre contrôleurs logiques et MES est fixe et il n'y a pas d'autres générateurs de trafic. Pour des AAR où ces conditions ne sont plus vraies, il conviendrait de s'intéresser à la classification de service au sein des commutateurs, sous réserve que ceci soit possible, pour que les flux induits par les échanges entre les contrôleurs logiques et les modules d'E/S soient prioritaires par rapport aux autres.

A **la vue de ces résultats, il est possible de conclure que les trafics annexes à ceux générés par l'AAR ne perturbent pas les performances temporelles de l'AAR, tant qu'aucun des câbles Ethernet n'est chargé au delà de 90 Mbit/s (en full duplex).**

Utilisation des résultats expérimentaux pour la modélisation du fonctionnement du réseau.

Pour savoir s'il y a un risque de saturation du réseau Ethernet dans la configuration

étudiée dans ces travaux, le trafic sur le réseau a été estimé pour une architecture standard, celle de la figure 1.3. Lors d'un échange entre un API et un MES, une requête de type "Read/Write register" est générée par l'API pour transmettre les nouvelles valeurs des sorties et simultanément demander les nouvelles valeurs des entrées, et une réponse de type "Write register" est envoyée par le MES. Les requêtes sont contenues dans des trames Ethernet de 89 octets (19 octets pour la couche application, 24 pour la couche TCP, 20 pour la couche IP et 26 pour la couche Ethernet) alors que les réponses sont contenues dans des trames de 81 octets[3]. Ainsi, pour chaque échange complet (une requête et sa réponse), 170 octets sont échangés entre le client (l'API) et le serveur (le MES). Pour l'AAR de l'exemple, si l'on considère par exemple que l'API1 communique avec quatre MES toutes les 10 ms, que l'API2 scrute cinq MES toutes les 10 ms et que l'API3 communique avec neuf MES toutes les 50 ms, il y a 400 échanges par seconde à l'initiative de l'API1, 500 échanges par seconde à l'initiative de l'API2 et 180 échanges par seconde à l'initiative de l'API3. L'ensemble de ces échanges crée un trafic de 1.4688 Mbits/s, ce qui correspond à 1.5% du débit théorique maximum du réseau. L'influence de ce trafic sur les performances temporelles peut donc être considérée comme constante (cf. figure 1.10 et 1.11). En particulier, il n'y a aucun risque de congestion du réseau. Pour atteindre la valeur critique de 180 Mbits/s dans un câble Ethernet (en full duplex), il faut générer un peu plus de 132000 requêtes par seconde, ce qui correspond par exemple à 66 API, dont le cycle d'IOscanning est de 10 ms, qui communiquent avec 20 MES chacun.

Nous pouvons enfin noter que les collisions et les pertes de trames ne sont pas envisagées dans ces travaux de par l'utilisation de commutateurs (et non de concentrateurs) et de composants réseaux fonctionnant tous en full-duplex. Comme le trafic dû aux échanges entre les clients et les serveurs Modbus est peu important au regard des possibilités des composants réseau utilisés, la taille des différentes files d'attente (dans les commutateurs notamment) a été considérée comme suffisante pour que ces files ne soient jamais saturées.

Il a donc été décidé de représenter l'impact du réseau sur un échange API/MES ou MES/API uniquement par un délai constant et de ne pas prendre en compte les interactions entre les différents échanges au niveau du réseau. Ceci nous amène à modéliser chaque échange indépendamment des autres et ainsi à décomposer le réseau en **un ensemble de fonctions de communication indépendantes**. Chacune de ces fonctions de communication représente un échange entre une carte de communication et un MES. Le modèle du réseau correspond ainsi à l'ensemble des modèles des fonctions de communication.

La figure 1.12 présente de manière informelle le comportement d'une fonction de communication (FC). Son état initial est un état d'attente dont elle sort lors de la réception d'une requête en provenance de la carte de communication (client). La FC transmet alors cette requête au MES (serveur de données) après un délai correspondant au temps de traversée du réseau. Le même comportement se répète pour transmettre la réponse en provenance du MES. La FC sort de sa phase d'attente lors de la réception de la réponse avant de la transmettre à la carte de communication au bout du délai correspondant au temps de traversée du réseau.

Le modèle du réseau est composé alors de N modèles identiques à celui de la figure 1.12, N étant le nombre de communications API/MES et le temps de traversée étant une

[3] www.modbus.org

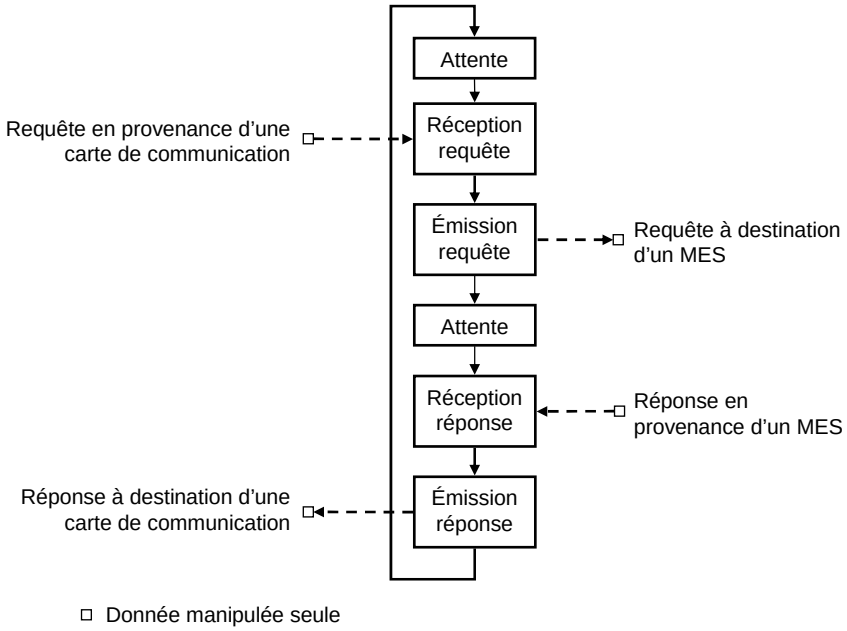


FIG. 1.12 – Fonctionnement d'une fonction de communication

constante obtenue par mesure (10 μs par commutateurs traversés pour les commutateurs utilisés). Dans le cas de l'exemple utilisé ci-dessus, le modèle du réseau comporte donc 18 FC.

1.2.2.3 Fonctionnement des MES

L'état normal d'un MES est un état d'attente (figure 1.13), dont il ne sort que lors de la réception d'une requête en provenance d'une carte de communication (via le réseau). Le traitement de cette requête consiste :

- à lire les valeurs des entrées physiques connectées au MES,
- à émettre sur les sorties physiques du MES les valeurs de sortie contenues dans la trame reçue,
- et enfin à envoyer la réponse contenant les nouvelles valeurs des entrées à la carte de communication à l'origine de la requête.

Si une ou plusieurs requêtes arrivent alors que le MES est en train de traiter une requête antérieure, ces requêtes sont mises en attente dans une file FIFO, puis traitées dans leur ordre d'arrivée.

1.2.2.4 Synchronisation entre les différents composants

Les échanges entre les différents composants de l'AAR sont représentés sur la figure 1.14 où seulement un API et n MES sont considérés.

Il apparaît qu'il peut y avoir un délai très variable entre le moment où la carte de communication met les données en provenance des MES à disposition du processeur de calcul et celui où il les prend en compte. Ce délai est compris entre un délai nul (le

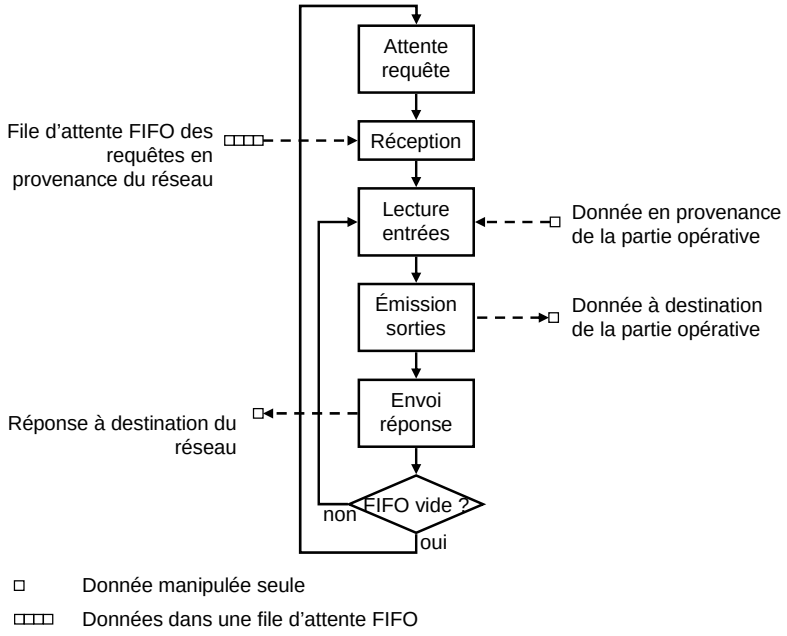


FIG. 1.13 – Fonctionnement des modules d'entrées/sorties

processeur lit ses entrées dans la zone tampon 2 juste après qu'elles aient été mises à sa disposition) et le temps de cycle maximal du processeur de calcul (le processeur vient juste de lire ses entrées quand elles sont remises à jour). De même, la prise en compte des nouvelles valeurs des sorties par la carte de communication dans la zone tampon 1 peut être instantanée comme atteindre jusqu'à un cycle d'IOscanning.

Pour les MES, entre la date d'arrivée d'une requête et son traitement, il peut y avoir un temps égal au temps de traitement de toutes les autres requêtes reçues des autres cartes de communication qui communiquent avec le MES.

Cette analyse rapide montre cependant bien que les performances temporelles des AAR sont très variables et que leur évaluation nécessite des méthodes précises, exposées dans la partie suivante de ce chapitre.

1.3 Evaluation des performances temporelles des systèmes en réseau

De nombreuses méthodes ont été développées pour évaluer les performances temporelles des "systèmes en réseau". Certaines d'entre elles se focalisent uniquement sur des performances du réseau de communication (délai de bout en bout, comportement en cas de défaillance pour des réseaux redondants, ...), tandis que d'autres considèrent les performances globales du système. Bien que notre objectif soit relatif à ce dernier type, nous n'avons pas voulu exclure de notre analyse bibliographique les travaux motivés par le premier objectif et présenterons les résultats obtenus antérieurement en adoptant la classification suivante : méthodes basées sur des analyses multiples et méthodes permettant

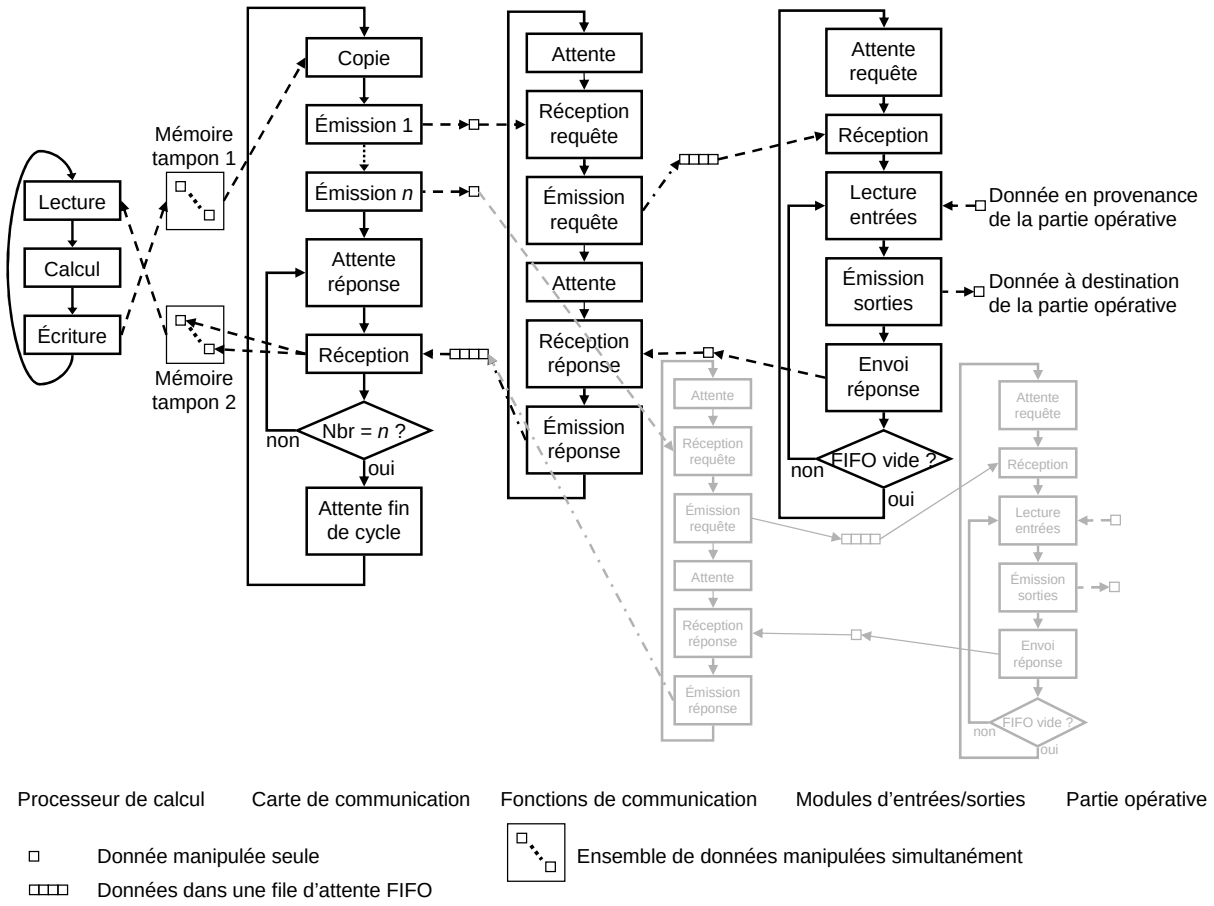


FIG. 1.14 – Synchronisation des différents composants d'une AAR à 1 API et n MES

d'obtenir directement le domaine de fonctionnement. Les travaux de la première catégorie fournissent une distribution de valeurs dont les valeurs extrêmes sont considérées comme les bornes de la performance étudiée ; les méthodes relevant de la seconde approche délivrent par contre directement les bornes de la performance.

1.3.1 Évaluation par analyses multiples

Les méthodes pour l'évaluation des performances temporelles des AAR par analyses multiples sont de deux types. Soit elles s'appuient sur un modèle de la future architecture ; il s'agit alors de simulation. Soit elles utilisent une AAR existante ; il s'agit dans ce cas de mesure.

1.3.1.1 Simulation

De nombreux travaux ont abordé l'évaluation des performances temporelles des AAR par le moyen de la simulation. Plusieurs approches coexistent, certaines basées sur la théorie des SED et en particulier les réseaux de Petri, d'autres sur des outils dédiés à la

simulation.

Utilisation d'une classe particulière de réseaux de Petri

Les travaux basés sur la théorie des SED utilisent surtout des représentations par réseaux de Petri colorés et temporisés comme [Zai04], qui évalue le temps de réponse d'un réseau, délai entre l'émission d'une requête par une station de travail (WS sur la figure 1.15) et la réception de la réponse envoyée par un serveur de données (S sur la figure 1.15).

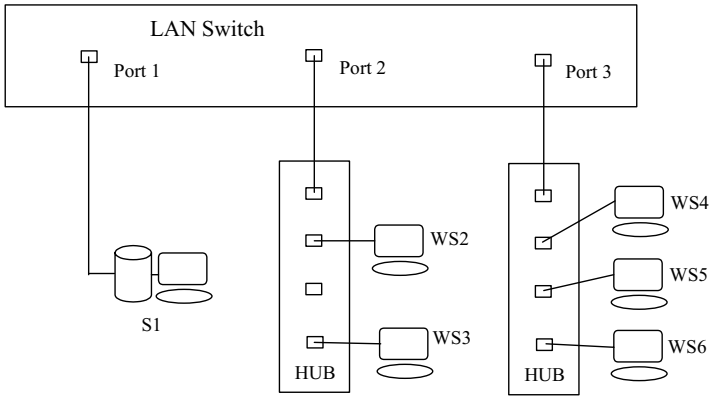


FIG. 1.15 – Exemple de système étudié par Zaitsev [Zai04]

Les réseaux de Petri colorés et temporisés sont également utilisés par [Mar06] pour l'évaluation de temps de réponse dans des architectures d'automatisation utilisant un réseau basé sur Ethernet et la comparaison des modèles de coopération (maitre/esclave, client/serveur et producteur/consommateur). POur cette étude, c'est bien des performances globales de l'AAR qui sont étudiées et non pas les performances du réseau uniquement.

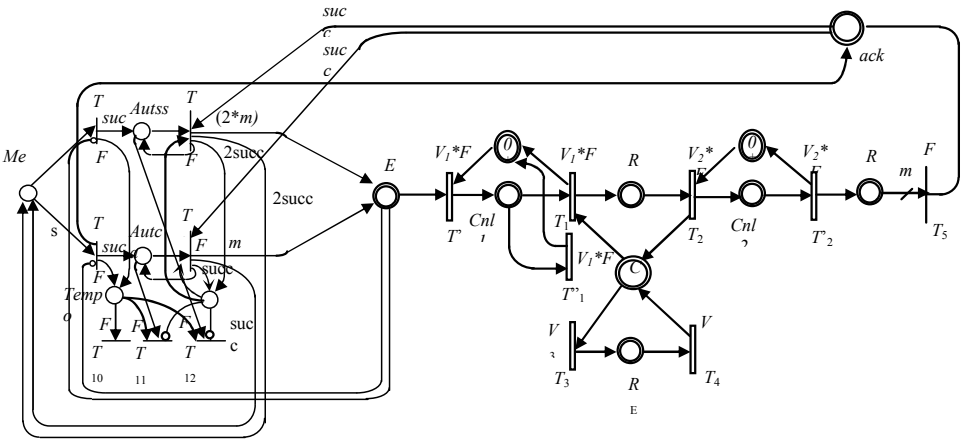


FIG. 1.16 – Modèle d'une ligne de transmission utilisant TCP proposé par Bitam [BA05a]

Des réseaux de Petri hybrides sont par ailleurs utilisés comme dans [BA05a]. Ces réseaux de Petri permettent de représenter à la fois des comportements continus et des comportement séquentiels (figure 1.16). Une représentation continue sera utilisée pour modéliser le comportement des flux de messages alors qu'une représentation logique permet de modéliser les protocoles de contrôle du réseau (slow start, congestion avoidance, ...). Les différentes dynamiques des réseaux peuvent être observées (charges des buffers, retards, pertes de trame, ...) et ainsi les effets des changements des paramètres du réseau et des protocoles sur les transmissions sont visibles.

Utilisation d'un outil logiciel de simulation

[LF07] a proposé un outil pour l'évaluation de temps de réponse basé sur le logiciel Modelica([FE98])/Dymola([Cel91]). Comme le montre la figure 1.17, le cas d'étude est simple mais les résultats obtenus sont intéressants car les valeurs obtenues sont comparables à celles fournies par le logiciel TrueTime ([EC99]) mais ceci avec des temps de calcul plus de deux fois plus faibles.

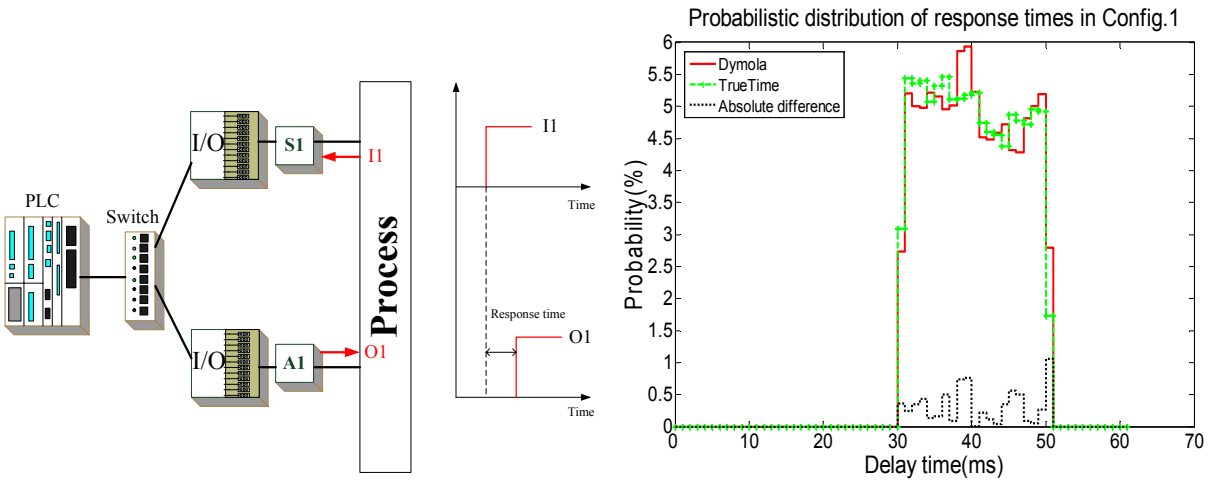


FIG. 1.17 – Cas d'étude traité par [LF07] et résultats obtenus

Le but des travaux de [PTP04] est de réaliser, à partir de l'étude des temps de réponse d'une AAR obtenus à l'aide du simulateur de réseau OMNet++ [Var01], des hypothèses raisonnables qui seront utilisées pour des modèles analytiques et particulièrement pour des analyses au pire des cas.

Il convient de souligner que ces outils de simulation de réseaux sont le plus souvent utilisés pour l'évaluation de performances propres aux réseaux (temps de traversée, temps de cycle, ...) et non d'une architecture d'automatisation complète utilisant un réseau pour communiquer.

1.3.1.2 Mesure

Peu d'équipes de recherche travaillent vraiment sur l'évaluation de performances au moyen de mesures mais cette technique est souvent utilisée en complément d'autres méthodes (simulation, calcul au pire des cas, ...) pour l'analyse du comportement d'un élément de l'architecture, l'estimation des valeurs de paramètres caractéristiques de modèles, ...

[CBV06] a travaillé plus particulièrement sur la mesure de performances de composants (cartes réseau) et non sur les performances d'un système complet.

[DFF+06], [FFV06] ont développé un outil de mesure, basé sur la capture de trames Ethernet en différents points du réseau, et l'ont utilisé notamment pour la mesure de performances de réseaux PROFINET.

[PMT06] a étudié plus en détail les délais engendrés par la couche application pour différents types de communications (OPC, VPN).

[DRF+07] a montré expérimentalement, pour des AAR à base de réseau Modbus TCP/IP, l'impact sur les performances temporelles de l'ajout de nouveaux services et de nouveaux flux de données ne concernant pas directement les tâches d'automatisation (intégration verticale).

1.3.1.3 Limitations de ces méthodes

Pour la mesure, la première limitation est évidente. L'AAR doit déjà exister pour pouvoir la valider, ce qui dans la majorité des cas n'est pas envisageable. De ce fait, la mesure est souvent utilisée sur des plateformes expérimentales (ne correspondant pas complètement à l'AAR étudiée) pour évaluer les performances d'un élément de l'AAR, afin de pouvoir le modéliser par la suite, ou valider des hypothèses de modélisation.

De plus, les deux méthodes ont une limitation commune qui est le problème de l'exhaustivité des observations. La multiplication des analyses permet d'obtenir l'allure de la distribution de la performance étudiée mais ne permet en aucun cas d'en déterminer les bornes avec certitude. Des méthodes pour évaluer la précision des "bornes" obtenues ainsi que le nombre d'analyses nécessaires pour les obtenir avec cette précision ont été développées ([Meu06]) mais sans pour autant supprimer cette difficulté.

1.3.2 Obtention directe du domaine de fonctionnement

Les méthodes formelles permettant d'obtenir directement le domaine de fonctionnement de systèmes en réseau et, en conséquence, les bornes ou les majorants/minorants des performances temporelles étudiées peuvent se classer en deux familles, celles utilisant des méthodes analytiques et celles basées sur le model-checking.

1.3.2.1 Méthodes analytiques

Le calcul de délai au pire des cas peut être utilisé pour l'évaluation de temps de cycle réseau comme dans [TV01] et [Vit01]. Ce calcul au pire des cas consiste à sommer des

délais élémentaires constants (durée d'une tâche) et non constants (attente au pire des cas de synchronisation entre deux cycles).

Une autre approche bien connue et importante pour calculer analytiquement des délais dans des systèmes en réseau est le calcul réseau (network calculus) initié par [Cru91] et développé par [BT01]. Le but de cette approche est d'obtenir les délais minimum et maximum dans des composants en réseau en utilisant l'algèbre Min-Plus.

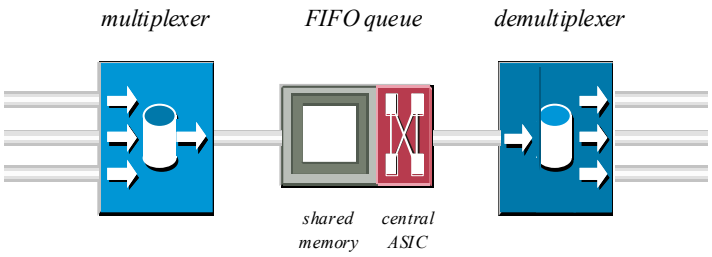


FIG. 1.18 – Modèle de commutateur proposé par [GRD02]

Une application de cette méthode a été réalisée par [GRD02] pour le calcul du temps de traversée maximum d'un ensemble de commutateurs en cascade. La figure 1.18 représente la modélisation retenue pour un commutateur qui sera traversé par des trafics périodiques et apériodiques. [GKDR06] fait une application de ces travaux pour la conception et l'optimisation de réseaux commutés pour satisfaire aux contraintes d'applications temps réel.

Les résultats obtenus avec les méthodes précédentes sont des majorants ou minorants des performances étudiées qui peuvent parfois être relativement éloignés des valeurs réelles su système étudié. La méthode des trajectoires ([BT97]) permet d'obtenir des valeurs bien plus proches sous réserves d'hypothèses simplificatrices. Elle consiste à étudier l'ordonnancement produit par tous les noeuds traversés par le flux de données. Seulement les scénarii possibles sont examinés, ce qui permet d'obtenir des résultats moins pessimistes. [Mar04] annonce des résultats très intéressants pour l'obtention de délais de bout en bout par la méthode des trajectoires. Ceux-ci surestiment dans le pire des cas de seulement 7% les valeurs exactes ; alors que dans la même configuration, la méthode de calcul réseau traditionnelle amène à des valeurs quasiment deux fois plus élevées que les valeurs exactes.

Ces différentes méthodes ont cependant une limitation commune dans le fait qu'elles ne s'intéresent qu'à des performances propres aux réseaux de communication et non à des performances globales d'AAR. Des travaux [AA08] commencent à étudier ces types de perfomances au travers d'une modélisation par graphes d'évènements temporisés (figure 1.19) et de l'utilisation de l'algèbre Max-Plus.

Les résultats obtenus avec cette méthode sont très intéressants car ils n'engendrent pas d'écarts trop importants par rapport aux bornes mesurées, mais restent, pour le moment, limités à des AAR ne comportant qu'un seul contrôleur logique, ce qui est bien trop peu.

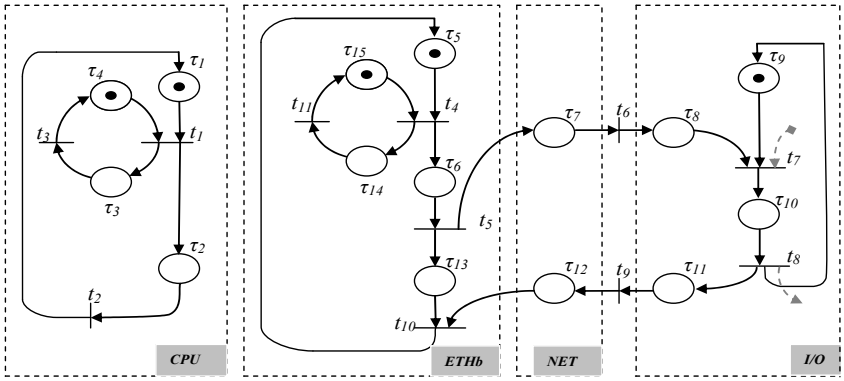


FIG. 1.19 – Graphe d'évènements temporisé utilisé par [AA08]

1.3.2.2 Vérification formelle par model-checking

Il existe différentes techniques de model-checking : model-checking temporisé ou non, probabiliste ou même paramétré.

Le model-checking non temporisé est clairement inadapté pour atteindre l'objectif de ces travaux. L'évaluation de performances temporelles d'AAR impose de pouvoir modéliser l'écoulement du temps.

Le model-checking probabiliste, utilisé notamment par [GF07] ne semble pas non plus particulièrement adapté pour ces travaux vu que la perte de trame dans le réseau n'est pas considérée et que seules les bornes des performances temporelles étudiées sont importantes, et non leur répartition. De plus, cette approche semble plus sensible à l'explosion combinatoire que le model-checking non temporisé ou temporisé.

Deux familles de model-checking semblent donc être plus intéressantes pour ces travaux, le model-checking temporisé et le model-checking paramétré. Les principaux résultats obtenus avec ces techniques pour l'analyse des systèmes en réseau sont rappelés ci-après.

Utilisation du model-checking temporisé pour l'analyse des systèmes en réseau

Le model-checking temporisé a pour but la vérification de propriétés logiques sur des modèles qui sont temporisés. [BA05b] applique cette technique à la gestion automatique de la qualité de service pour des réseaux sans fil. Bien que les grandeurs à surveiller sont le débit du réseau, ainsi que la latence et la gigue que subissent les trames pour traverser ce réseau, la validation du comportement se fait simplement en vérifiant que ces caractéristiques restent dans une plage de valeur prédéfinie. Si ce n'est pas le cas, une situation "défaut" est atteinte ce qui ramène la vérification du comportement de ce réseau à la vérification que cette situation n'est jamais atteinte.

De son coté, [RNPH06] traite du problème de la synchronisation d'horloge entre maître et esclave dans un réseau CAN (Controller Area Network). De la même manière, la vérification de ce système a été ramenée à une recherche d'atteignabilité d'une situation de défaut (situation "FAILURE") d'un automate observateur (figure 1.20). Cet automate

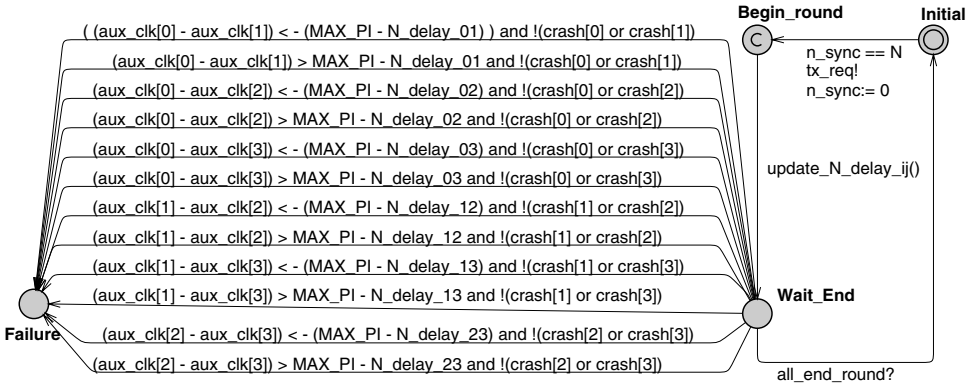


FIG. 1.20 – Automate observateur utilisé par [RNPH06]

observateur est assez compliqué car il doit prendre en compte toutes les possibilités de désynchronisation des horloges. Le problème résolu n'est donc pas là non plus de trouver quelle est la désynchronisation des horloges mais si cette désynchronisation reste inférieure à une limite de bon fonctionnement.

Les travaux de [Lim09] ont porté sur la validation d'architectures de contrôle-commande redondantes à base d'Ethernet industriel. Les différentes propriétés à vérifier sur ces architectures (présence d'un seul maître sur le réseau, absence de collision, de perte de trame, ...) peuvent l'être notamment en vérifiant que les situations "défaut" des différents automates ne sont pas atteintes. Ces travaux sont très intéressants du point de vue du passage à l'échelle du model-checking temporisé. Ayant été réalisés en partenariat avec un industriel, les cas d'application (réels) sont donc de grande taille ce qui implique d'utiliser des abstractions appropriées pour réussir le passage à l'échelle. Ainsi, des systèmes composés de 4 à 10 noeuds (suivant le nombre et le type de défaillances possibles), noeuds dont le modèle générique est donné figure 1.21, ont pu être vérifiés.

En conclusion, il est possible de dire que l'objectif du model-checking temporisé est clairement de prouver si une propriété logique est vraie ou fausse sur un modèle temporisé. De ce fait, cette technique ne donne pas un domaine de fonctionnement. S'il existe des applications sur des AAR, elles n'ont pas pour objectif de déterminer les bornes de performances temporelles mais de vérifier un critère de bon fonctionnement de l'AAR pour un paramétrage donné.

Utilisation des techniques de model-checking paramétré

L'objectif du model-checking paramétré est de déterminer le domaine de valeurs que peut prendre un paramètre (inconnu) en fonction des autres paramètres (connus) du système. En première approche, cette technique semble la plus appropriée pour l'évaluation des bornes de performances temporelles d'AAR. Pour ce faire, il suffit de considérer la performance temporelle à étudier comme le paramètre inconnu du système et de rechercher le domaine de valeurs qu'il peut prendre en fonction des paramètres connus du système.

Le logiciel Hytech [HHWT97] a notamment été utilisé pour l'évaluation de grandeurs

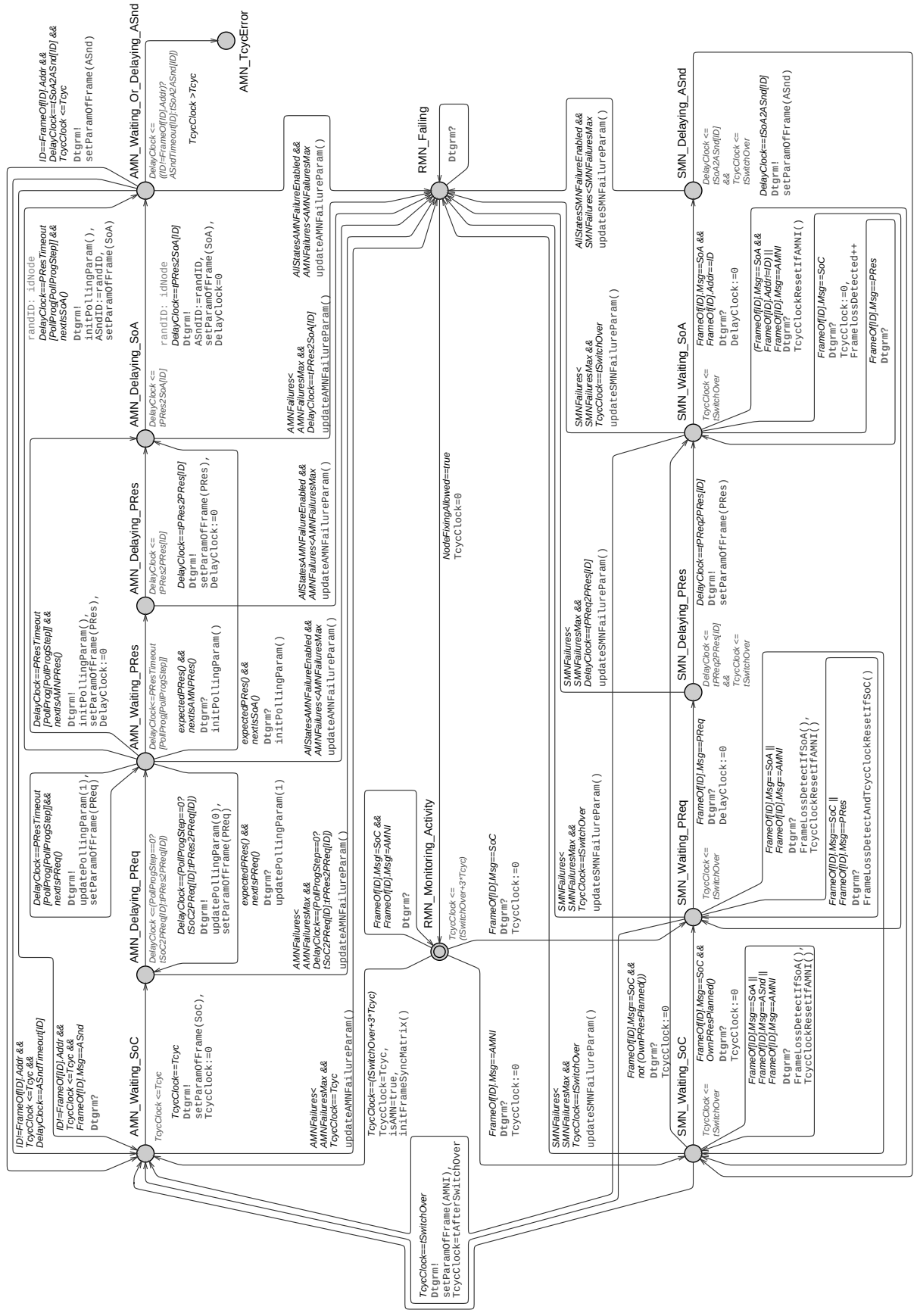


FIG. 1.21 – Modèle d'un noeud utilisé par [Lim09]

physiques par [SMF97] dont l'étude porte sur la hauteur d'une suspension pneumatique pour voiture.

Une application entièrement paramétrée pour l'obtention de délais de propagation de signaux dans un circuit mémoire a été réalisée [CEFX06]. L'élément de base du circuit mémoire est une bascule (dont le modèle est donné figure 1.22) qui propage la valeur de d (donnée à mémoriser) vers q (la sortie de la bascule) en fonction du signal e (enable). Le passage d'une situation où la donnée vaut 0 (noté d0) à une autre où elle vaut 1 (situation notée d1) se fait par une transition franchie simultanément à un front montant du signal d (d↑). Le temps (modélisé par l'horloge c) ne s'écoule que dans deux situations (notées e1d0 et e1d1) atteintes suite à un front montant sur le signal e. Le modèle ne comporte que très peu d'indéterminisme puisque seules ces deux situations ont un indéterminisme temporel (l'évolution du modèle se fait quand la valeur de d'horloge c est supérieure à la valeur l – lower bound – et inférieure à la valeur u – upper bound –) et toutes les évolutions se synchronisent avec d'autres modèles (propagation de fronts des signaux e, d et q).

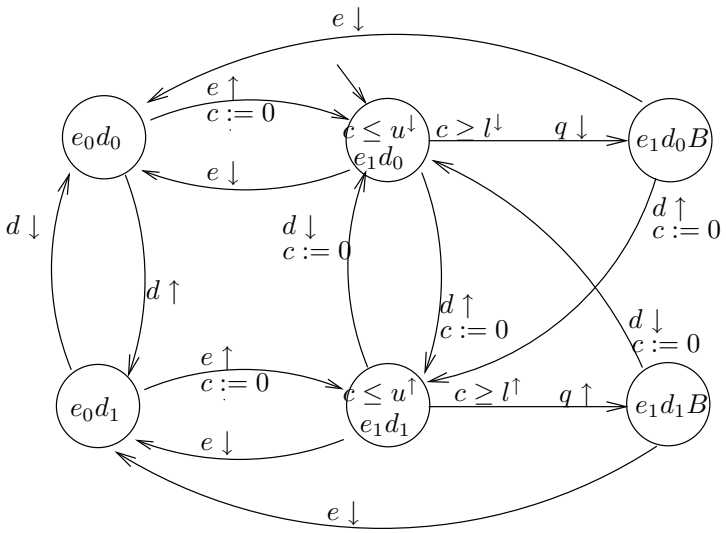


FIG. 1.22 – Modèle de bascule retenu par [CEFX06]

[JO08] utilise des modèles hybrides et le model-checker Phaver [Fre05] pour la vérification de propriétés quantitatives sur des systèmes logiques. Les systèmes représentés sont composés à la fois d'une partie commande et d'une partie opérative et les propriétés portent sur des grandeurs physiques telles que la position d'arrêt d'un convoyeur. Le modèle du contrôleur logique est composé de plusieurs automates hybrides (figure 1.23) mais dont les évolutions sont synchronisées. Ce modèle ne comporte de ce fait que très peu d'indéterminisme.

Sur la base de ces résultats prometteurs, une collaboration entre le LURPA et le LSV (Laboratoire Spécification et Vérification de l'ENS de Cachan) a été mise en place en 2007 sous la forme d'un projet labellisé par l'Institut Farman[4] et dénommé SIMOP : Synergie

[4] www.farman.ens-cachan.fr

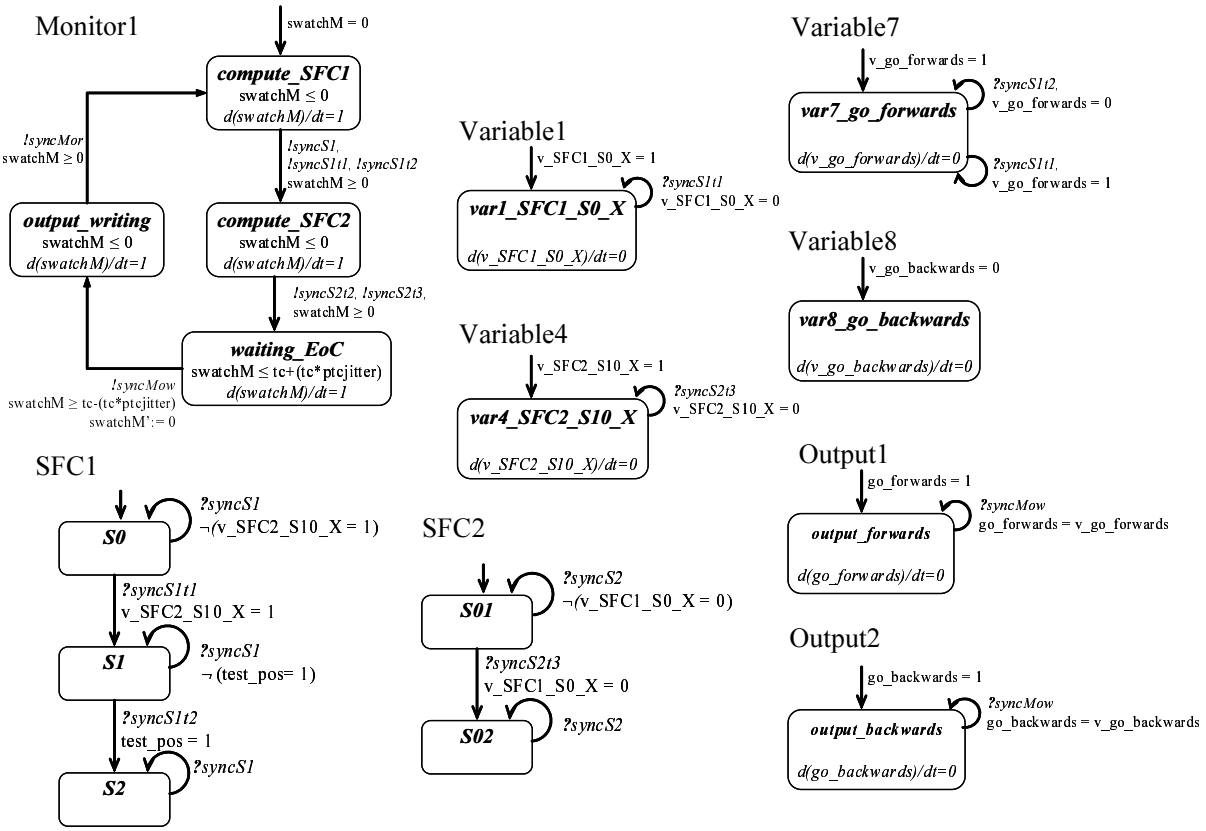


FIG. 1.23 – Modèle d'un contrôleur logique proposé par [JO08]

Simulation Model-checking Paramétré, auquel j'ai participé. L'un des objectifs principaux de ce projet était d'évaluer les capacités du model-checking paramétré pour l'évaluation des bornes des performances temporelles des AAR. Les principaux résultats de cette étude sont résumés ci dessous.

En utilisant l'outil Hytech et pour des analyses avec un seul paramètre libre, le temps de réponse de l'AAR, les autres paramètres des modèles (durée des différentes tâches) étant instanciés, il faut compter environ 1 heure de calcul pour une AAR constituée que d'un API et d'un MES. Cette valeur de 1 heure correspond à une moyenne puisque suivant la valeur des différents délais introduits dans le modèle de l'AAR, certains calculs prennent 20 minutes alors que d'autres n'aboutissent pas. Le passage à des AAR plus complexes n'est donc pas réalisable pour une résolution avec un seul paramètre libre et donc à plus forte raison pour une étude entièrement paramétrée (aucune instanciation des paramètres). Cependant, sur ce point, [ACEF08] propose une méthode pour construire l'espace des solutions en concaténant des portions de cet espace chacune définie de manière entièrement paramétrée.

A l'aide de l'outil Phaver, dont les performances en matière de temps d'analyse semblent bien meilleures, il faut 5 minutes de temps de calcul pour obtenir le domaine des temps de réponse de la même architecture constituée d'un API et d'un seul MES. Cependant, le calcul ne peut aboutir faute de mémoire (malgré les 4 Go installés sur l'ordinateur)

pour une AAR comportant un API et deux MES.

Le model-checking paramétré est bien trop sensible à l'explosion combinatoire pour être utilisé sur des modèles de taille non triviale et aussi indéterministes que ceux nécessaires pour représenter les AAR traitées dans ces travaux.

1.3.3 Synthèse

TAB. 1.1 – Synthèse des techniques présentées

Techniques	Système	Performances globales des AAR	Résultat fourni	Taille des systèmes analysés
Mesure	réel	oui	distribution de valeurs	grande
Simulation	modèle	oui	distribution de valeurs	grande
Calcul réseau	modèle	non	majorant / minorant	grande
Model-checking temporisé	modèle	oui	propriété logique, vraie / fausse	grande
Model-checking paramétré	modèle	oui	domaine de valeurs	petite

En conclusion de cette partie, le tableau 1.1 reprend les principales caractéristiques des différentes techniques présentées. Aucune de ces techniques ne répond aux critères fixés pour ces travaux car aucune ne fournit réellement des bornes avec certitude. Les plus séduisantes (calcul réseau et model-checking paramétré) présentent respectivement l'inconvénient de traiter des performances des réseaux et non des performances globales des AAR ou de ne pas permettre le passage à l'échelle.

1.4 Objectif des travaux

Comme le tableau 1.1 l'a montré, aucune des techniques présentées dans la partie 1.3 n'est utilisable directement pour obtenir les bornes des performances temporelles d'AAR de taille non triviale. Cependant les solutions de la famille du model-checking semblent être les plus à même de répondre à ces problèmes. Elles permettent en effet d'englober toutes les évolutions possibles, contrairement aux solutions par analyses multiples, et arrivent à prendre àtraiter des problèmes qui ne soient pas uniquement liés au réseau de communication mais bien à l'ensemble de l'AAR (réseau + contrôleurs logiques + modules d'E/S).

Parmi cette famille de méthodes, le model-checking paramétré, qui permet d'obtenir directement le domaine des valeurs prises par la performance étudiée, est trop sensible à l'explosion combinatoire pour réaliser le passage à l'échelle alors que le model-checking

temporisé ne peut donner que des réponses booléennes alors qu'une réponse quantitative est attendue. Pour palier à ces difficultés, une méthode, esquissée sur la figure 1.24, sera proposée dans la suite de ce mémoire.

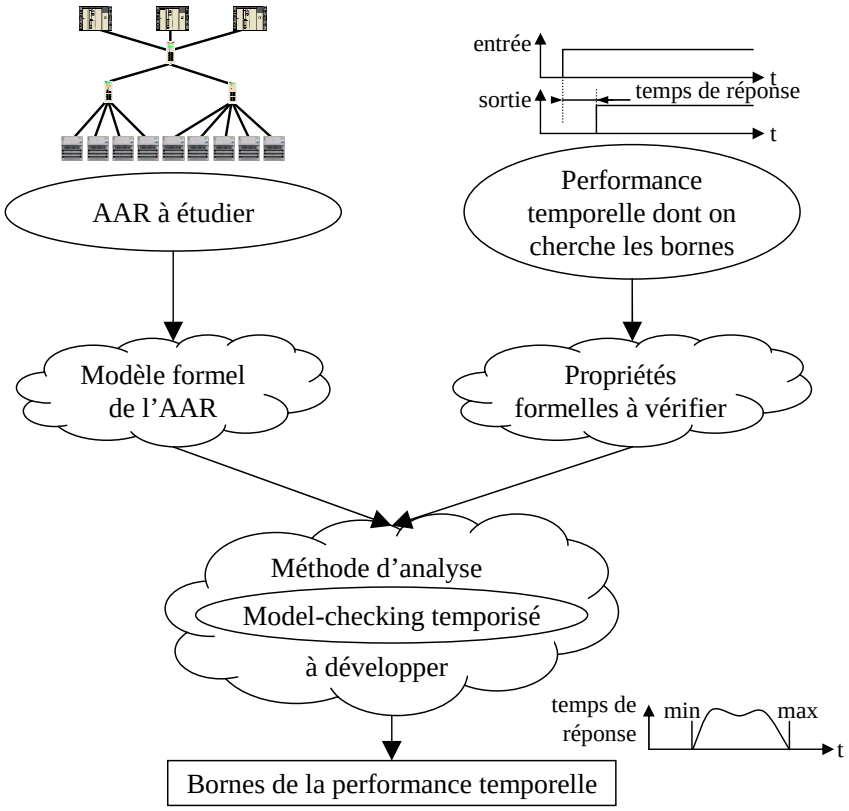


FIG. 1.24 – Objectif des travaux

Cette méthode d'analyse sera basée sur les techniques de model-checking temporisé afin de pouvoir réaliser une analyse exhaustive d'un modèle de grande taille. L'obtention de valeurs quantitatives sera possible grâce à l'interprétation des résultats booléens donnés par le model-checker pour différentes propriétés. Pour cela, il sera nécessaire de développer un modèle formel de l'AAR et de transcrire la performance temporelle à étudier en propriétés formelles compréhensibles par le model-checker et adaptés à la méthode. Cette dernière, ainsi que les propriétés formelles utilisées, vont être détaillées dans le chapitre suivant. La modélisation de l'AAR fera l'objet du chapitre 3.

Chapitre 2

Méthode d'évaluation des bornes des performances temporelles proposée

Ce chapitre présente notre première contribution : une méthode développée pour l'obtention des bornes des performances temporelles au moyen de techniques de model-checking temporisé. Ces techniques n'étant pas prévues pour délivrer des valeurs numériques mais bien des résultats booléens, une méthode par itérations a été mise en place pour y arriver.
Après avoir rappelé les principales caractéristiques de la vérification formelle par model-checking temporisé, le principe de notre proposition est décrit. Les deux dernières parties du chapitre en détaillent les éléments majeurs : automate observateur paramétré et algorithme de modification du paramètre temporel en fonction des résultats de preuves.

Sommaire

2.1 Mise en oeuvre des techniques de model-checking temporisé

2.1.1 Model-checking temporisé

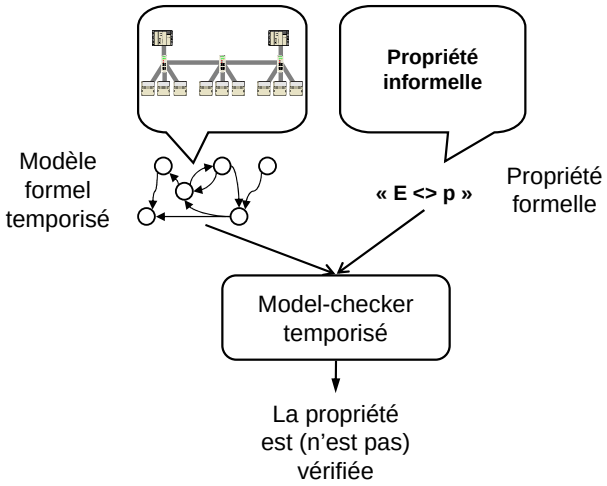


FIG. 2.1 – Principe du model-checking temporisé

Comme l'indique la figure 2.1, le principe du model-checking est de vérifier des propriétés formelles (des assertions écrites en logique temporelle) sur des modèles formels. Le model-checking est dit temporisé dès que les modèles formels sont temporisés alors que les propriétés à vérifier peuvent être écrites en logique temporelle temporisée ou non temporisée. Les logiques temporelles non temporisées, telles que PLTL (Propositional Linear Temporal Logic) ([Pnu81]) ou CTL (Computation Tree Logic) ([CE81] et [EH82]), permettent d'exprimer, par exemple, qu'une formule Booléenne est toujours vraie pour toutes les évolutions possibles du modèle, vraie pour au moins une évolution, etc. Ces deux logiques temporelles non temporisées peuvent être vues comme des fragments de la logique CTL* ([EH86]). Les logiques temporelles non temporisées ne permettent pas de quantifier du temps dans les propriétés comme c'est le cas par exemple dans l'assertion : "il s'écoule moins de 10 secondes entre l'apparition du signal S1 et l'apparition du signal S2". C'est pourquoi [Koy90] a par la suite proposé l'extension de la logique temporelle pour permettre la prise en compte d'informations quantitatives sur l'écoulement du temps, ce qui a abouti aux logiques temporelles temporisées. Ainsi, cette approche appliquée à la logique CTL a donné la logique temporisée TCTL (timed CTL).

Kronos est un des rares model-checkers à supporter des propriétés écrites en logique TCTL ; malheureusement il est très sensible au phénomène d'explosion combinatoire et est donc beaucoup moins efficace (rapidité et taille limite des modèles) que des model-checkers utilisant une logique plus limitée comme UPPAAL. D'après [LPY95], Kronos est 70 fois plus lent que UPPAAL pour la vérification du protocole d'exclusion mutuelle de Fischer et pour [MLAH99], il est même 150 fois plus lent lors de la vérification de l'ordonnanceur de Milner.

Pour ces travaux, le model-checker temporisé Uppaal ([LPY97]) a été choisi pour différents critères (cf. paragraphe précédent notamment), issus de l'expérience acquise au sein du LURPA comme de celle de la communauté ([BS00], [Wan04]). Il supporte des modèles temporisés (la sémantique utilisée sera présentée en partie 3.1) et utilise une logique temporelle non temporisée (voir partie 2.1.2), ce qui obligera à utiliser un observateur (dont le fonctionnement sera détaillé à la partie 2.1.3). Comme il est encore développé, ses performances ne cessent de s'améliorer, ce qui lui permet d'être moins sensible au phénomène d'explosion combinatoire si fréquent en model-checking temporisé lors du passage à l'échelle. Il est par ailleurs très stable ; au cours de ma thèse, les seuls calculs n'ayant pu aboutir l'ont été par saturation de la mémoire de l'ordinateur et non par une erreur lors de la vérification (arrêt intempestif du logiciel). Ce model-checker offre également des facilités pour le développement de modèles par l'intermédiaire de son interface graphique qui permet de concevoir des modèles plus explicites que des modèles bruts en lignes de texte et surtout par son outil de simulation qui permet de visualiser facilement les évolutions possibles des modèles et ainsi de trouver d'éventuelles erreurs dans la modélisation.

2.1.2 Écriture de propriétés avec Uppaal

Malgré ses qualités, le model-checker temporisé Uppaal a des limites, notamment pour l'expression des propriétés à vérifier. Il utilise en effet pour cela une sous-classe de la logique CTL qui limite le langage de spécification des propriétés à seulement cinq types de propriétés. Une description détaillée des possibilités d'Uppaal et de la sémantique de représentation des modèles sera donnée au chapitre 3.

Il est bon de rappeler en préambule que, contrairement à la simulation où une seule évolution est observée, le model-checking temporisé considère, lors d'une seule analyse, toutes les évolutions possibles du modèle. Les différentes propriétés qui seront exprimées par la suite seront donc toujours à vérifier pour l'ensemble des évolutions possibles du système étudié.

En considérant p et q, deux propositions logiques[5] sur les états du modèle formel (par exemple l'automate A est dans la situation S1) et les valeurs des variables ou des horloges (la variable ou l'horloge est égale ($=$), inférieure strictement ($<$), supérieure strictement ($>$), inférieure ou égale ($\leq$) ou supérieure ou égale ($\geq$) à n avec n $\in \mathbb{N}$), il est possible de définir cinq propriétés logiques supportées par Uppaal, basées sur les quantificateurs de chemins (A : quel que soit le chemin et E : il existe un chemin) et d'états (<> : pour un état et [ ] : pour tous les états).

- Possibility : La propriété $E <> p$ est vraie si et seulement s'il existe au moins une séquence d'évolutions partant de la situation initiale $S0$ et atteignant une situation Sn tel que Sn satisfait p.
- Invariantly : La propriété $A[]p$ est vraie si et seulement si toutes les situations atteignables à partir de la situation initiale $S0$ satisfont p.

[5] Notons au passage qu'il est possible de combiner ces propositions au moyen des opérateurs Booléens *not*, *and*, *or* et *imply*.

- Potentially always : La propriété $E[]p$ est vraie si et seulement s'il existe au moins une séquence d'évolutions partant de la situation initiale $S0$ pour laquelle p est vérifiée pour tous les états de cette séquence. De plus cette séquence doit soit être infinie soit aboutissant à un état bloquant.
- Eventually : La propriété $A <> p$ est vraie si et seulement si toutes les séquences d'évolutions possibles partant de la situation initiale $S0$ atteignent un état vérifiant p.
- Leads to : La syntaxe $p \rightarrow q$ décrit une propriété signifiant que si p est vérifiée alors q devra aussi être vérifiée dans le futur, pour toutes les exécutions

La figure 2.2 illustre des exemples d'évolutions vérifiant ces cinq types de propriétés.

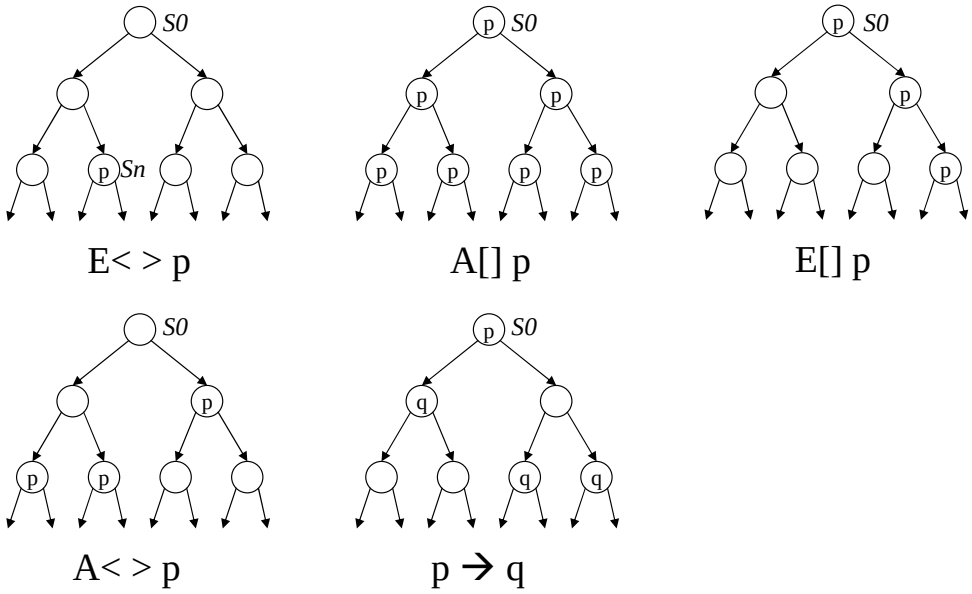


FIG. 2.2 – Les propriétés logiques supportées par UPPAAL

Ces cinq propriétés ne sont pas indépendantes les unes des autres. En effet, elles peuvent toutes être déduites à partir de deux d'entre elles seulement (E<> et E[ ] ou A<> et A[ ]) comme l'illustre le tableau 2.1[6].

TAB. 2.1 – Équivalence des propriétés temporelles

Nom	Propriété	Équivalent à
Possibility	$E <> p$	
Invariantly	$A[]p$	$notE <> notp$
Potentially always	$E[]p$	
Eventually	$A <> p$	$notE[]notp$
Leads to	$p \rightarrow q$	$A[](p\ imply\ A <> q)$

[6]La propriété "leads to" peut également s'exprimer en n'utilisant que les propriétés E<> et E[ ] grâce aux équivalences présentées dans ce tableau.

Il importe de souligner que ces propriétés sont insuffisantes pour pouvoir exprimer directement les performances temporelles qui nous intéressent dans ces travaux (1.2) puisqu'elles ne font jamais référence au temps physique. Il faudrait en effet pouvoir exprimer de façon formelle ce type de propriété : "quel est le délai qui s'est écoulé entre l'évènement $E1$ et l'évènement $E2$?", avec, par exemple dans le cas d'un temps de réponse, $E1$ un évènement d'entrée et $E2$ un évènement de sortie. La limitation d'expression imposée par les propriétés logiques supportées par UPPAAL nous conduit à utiliser un automate observateur qui pourra être vu comme un compteur du temps s'écoulant entre deux évènements.

2.1.3 Utilisation d'un automate observateur

Pour [BBF+01], l'utilisation d'un automate observateur ou automate de test permet de simplifier un système en restreignant les comportements autorisés aux seuls chemins acceptés par un automate extérieur au système. Les propriétés à étudier sont alors beaucoup plus simples à exprimer car alors "l'automate observateur est lui-même la spécification formelle de la propriété souhaitée". Dans notre cas, suivant l'état final atteint par cet automate observateur, il sera possible d'avoir des informations sur le délai observé. Il suffit donc de chercher quelle situation est atteinte par l'observateur au moyen d'une (ou plusieurs) propriété(s) d'atteignabilité, propriété(s) détaillée(s) dans la partie 2.3. Dans la syntaxe utilisée par UPPAAL, cette propriété d'atteignabilité se note :

$$E <> situation\_name \tag{2.1}$$

ce qui signifie : " il existe au moins une évolution du modèle formel telle que la situation situation_name soit atteinte à partir de l'état initial ".

Comme l'ensemble des évolutions possibles du modèle formel étudié sont obtenues en une seule vérification, il a été choisi de n'étudier le comportement de l'architecture d'automatisation que pour une seule variation des entrées, variation pouvant intervenir dans n'importe quelle configuration du système, ceci afin qu'elle soit représentative de toutes ses évolutions possibles. Cette variation des entrées est générée par un automate ENV (environnement) qui a pour tâche, outre la génération de l'évènement d'entrée (représentant le signal provenant d'un capteur), de recevoir les évènements de sortie (représentant les ordres envoyés aux actionneurs).

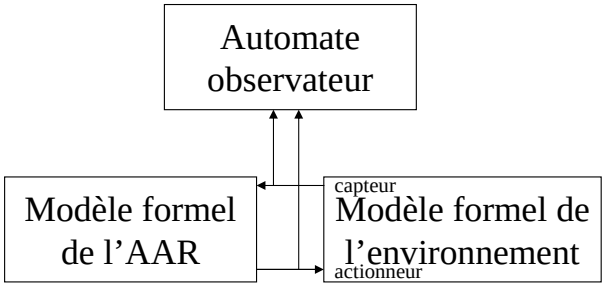


FIG. 2.3 – Éléments constituant le modèle formel à vérifier

Au final, le modèle formel à vérifier par UPPAAL sera composé du modèle de l'AAR, du modèle de l'environnement et de l'automate observateur (figure 2.3).

Il est bon de noter que, malgré l'utilisation d'un automate observateur associé à un ensemble de propriétés d'atteignabilité, il est toujours impossible de déterminer en une seule étape de vérification les bornes d'une performance temporelle. Dans le but d'obtenir ces bornes, la méthode présentée en partie 2.2 a été développée.

2.2 Principe de la méthode proposée

Le model-checking temporisé ne fournit que des résultats Booléens (la propriété logique étudiée est ou non vraie), et malheureusement pas de valeurs numériques. C'est pourquoi, une méthode itérative ([RdSF09]), décrite dans la figure 2.4[7], a été développée pour obtenir les bornes de performances temporelles des AAR, performances représentées par des distributions de valeurs.

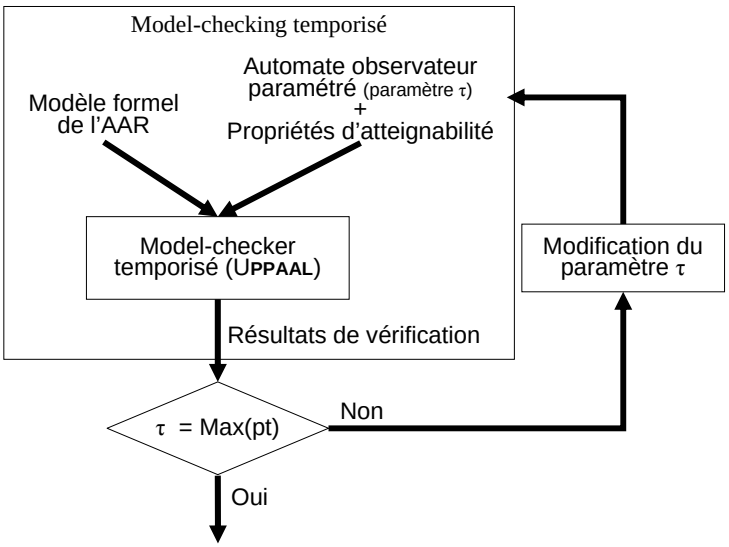


FIG. 2.4 – Principe de la méthode itérative pour obtenir la borne supérieure d'une performance temporelle pt

Cette méthode est basée sur les techniques de model-checking temporisé et repose sur l'enchaînement de plusieurs vérifications jusqu'à l'obtention de la borne recherchée. Tant que celle-ci n'est pas atteinte, le résultat de vérification sert à modifier l'automate observateur utilisé pour la vérification suivante.

Cet automate observateur est paramétré, c'est-à-dire qu'un paramètre temporel τ est introduit dans certaines gardes de ses transitions. Pour chaque vérification, il sera recherché où se situe ce paramètre τ par rapport aux bornes de la distribution représentant la performance temporelle (pt) étudiée.

[7]Cette figure décrit le principe d'obtention de la borne supérieure. La borne inférieure est obtenue en suivant le même principe mais en remplaçant dans le test $\tau = Max(pt)$ par $\tau = Min(pt)$

Si τ est égal à la borne recherchée, la recherche est terminée. Dans le cas contraire (si τ est strictement inférieur à la borne recherchée ou si τ est strictement supérieur à la borne recherchée), la valeur de ce paramètre sera modifiée et une nouvelle vérification sera réalisée.

La partie 2.3 détaillera la structure et le comportement de cet automate observateur ainsi que les propriétés qui seront vérifiées lors de chaque itération de la méthode. L'algorithme développé pour modifier la valeur de paramètre τ et enchaîner les itérations de manière à assurer la convergence de l'étude sera présenté dans la partie 2.4.

2.3 Définition de l'automate observateur et expression des propriétés à vérifier

2.3.1 Principe et structure de l'automate observateur

L'automate observateur paramétré va avoir une structure différente suivant la performance temporelle étudiée même si son principe demeure le même[8]. Il doit dans tous les cas observer l'évènement déclencheur (évènement d'entrée) et attendre le ou les évènements de sortie (conséquence(s)) en mesurant le temps qui s'écoule.

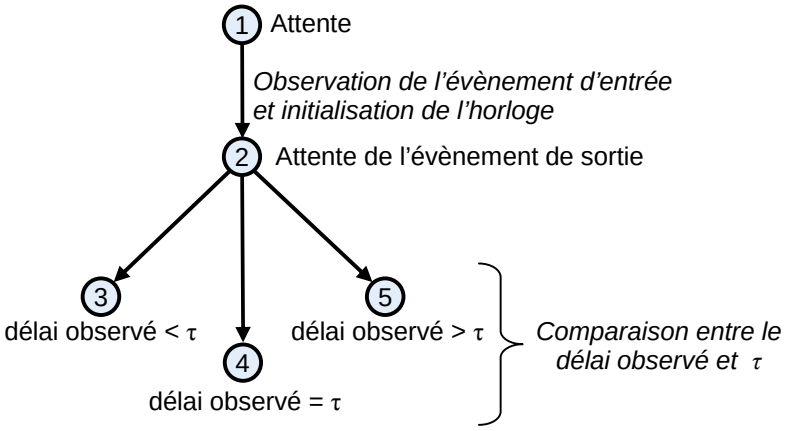


FIG. 2.5 – Structure de l'automate observateur pour un temps de réponse

L'évolution de l'automate observateur pour l'évaluation d'un temps de réponse (figure 2.5) comporte trois étapes :

– Tout d'abord il y a une phase d'attente (situation 1) qui prend fin lors de l'observation de l'évènement d'entrée. Lors de cette observation, l'horloge de l'automate observateur est mise à 0.

[8]Les travaux réalisés dans le cadre de cette thèse n'ont porté que sur le temps de réponse et la différence de temps de réponse. Ceci explique que seuls les observateurs utilisés pour la détermination des bornes de ces deux performances temporelles sont présentés. Une approche similaire peut cependant être conduite pour la conception d'un automate observateur dédié à la détermination des bornes de la durée minimale d'un signal d'entrée pour qu'il soit toujours pris en compte par l'AAR

– Une nouvelle phase d'attente (situation 2) a lieu jusqu'à l'observation de l'évènement de sortie.
– Suite à cette observation, l'automate observateur évolue vers une de ses 3 situations finales (situations 3, 4 ou 5) en fonction de la valeur de l'horloge (par rapport à τ) à l'instant d'observation.

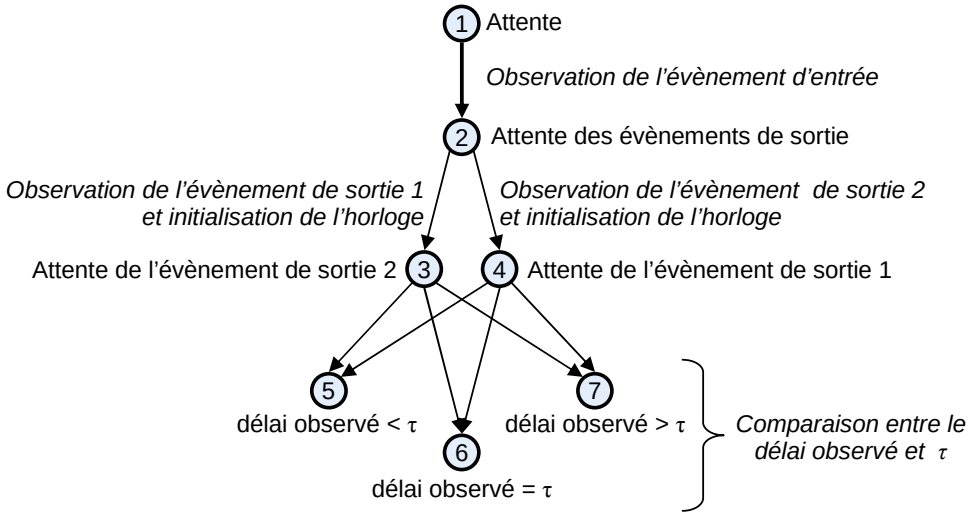


FIG. 2.6 – Structure de l'automate observateur pour une différence de temps de réponse

L'évolution de l'automate observateur pour l'évaluation d'une différence de temps de réponse (figure 2.6) comporte quatre étapes :
– Tout d'abord il y a une phase d'attente (situation 1) qui prend fin lors de l'observation de l'évènement d'entrée.
– Une seconde phase d'attente (situation 2) a lieu jusqu'à l'observation d'un des deux évènements de sortie, ce qui fera évoluer l'automate observateur vers la situation 3 (observation de l'évènement de sortie 1) ou vers la situation 4 (observation de l'évènement de sortie 2). Lors de ces évolutions, l'automate observateur met son horloge à 0.
– Une dernière phase d'attente (situation 3 ou 4) a alors lieu jusqu'à l'observation du deuxième évènement de sortie.
– Lors de cette observation, l'automate observateur évolue vers une de ses 3 situations finales (situations 5, 6 ou 7) en fonction de la valeur de l'horloge (par rapport à τ) à l'instant d'observation.

La présentation des modèles détaillés de ces automates (dans la syntaxe UPPAAL) sera faite en même temps que la présentation des modèles des composants de l'AAR, c'est-à-dire dans le chapitre 3.

2.3.2 Formes générales des propriétés à vérifier

Une fois que l'automate observateur (noté OBS) utilisé pour l'évaluation de la performance temporelle à étudier a été défini, trois propriétés formelles d'atteignabilité peuvent

être construites. Dans le cas de l'évaluation d'un temps de réponse (automate observateur de la figure 2.5), elle s'écrivent ainsi :

$$P1 : E <> OBS.3 \tag{2.2}$$

$$P2 : E <> OBS.4 \tag{2.3}$$

$$P3 : E <> OBS.5 \tag{2.4}$$

Pour l'évaluation de la différence de temps de réponse (automate observateur de la figure 2.6), les situations finales sont les situations 5, 6 et 7. Il faut donc remplacer 3, 4, 5 par 5, 6, 7 dans ces expressions formelles.

Ces trois propriétés peuvent être traduites en langage naturel par : " il existe au moins une évolution du système qui permet d'atteindre la situation 3 (respectivement situation 4 et situation 5) de l'automate observateur ".

Par suite, les résultats de la vérification de ces trois propriétés permettent de comparer la valeur du paramètre temporel τ avec les bornes supérieure et inférieure de la performance temporelle (pt) étudiée (tableau 2.2). Dans ce tableau, les notations Min(pt) et Max(pt) représentent respectivement les bornes inférieure et supérieure de la performance temporelle considérée.

TAB. 2.2 – Signification des résultats de preuves

Valeur de τ	P1 est	P2 est	P3 est	signifie que
$\tau = \tau_1$	fausse	fausse	vraie	τ est inférieur à Min(pt)
$\tau = \tau_2$	fausse	vraie	vraie	τ est égal à Min(pt)
$\tau = \tau_3$	vraie	vraie	vraie	τ est entre Min(pt) et Max(pt)
$\tau = \tau_4$	vraie	vraie	fausse	τ est égal à Max(pt)
$\tau = \tau_5$	vraie	fausse	fausse	τ est supérieur à Max(pt)

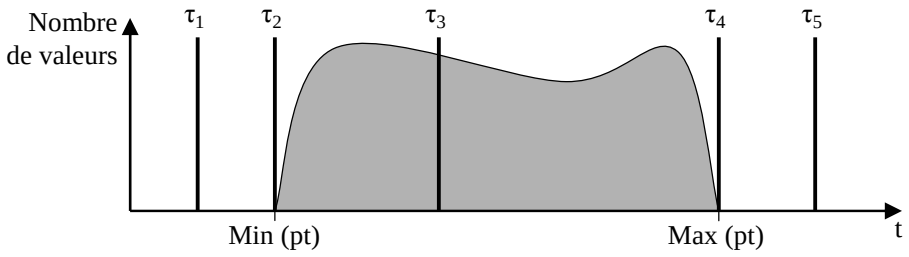


FIG. 2.7 – Positionnements possibles de τ par rapport aux bornes de la performance temporelle étudiée

Il est possible d'illustrer ce tableau en choisissant des valeurs de τ représentatives des différents cas de figure possibles (figure 2.7). Par exemple, si $\tau = \tau_1$, les trois propriétés P1, P2 et P3 sont respectivement fausse, fausse et vraie.

– P1 est fausse signifie qu'aucune évolution du modèle de l'AAR ne peut amener à une valeur de la performance temporelle étudiée qui soit strictement inférieure à τ.

- P2 est fausse signifie qu'aucune évolution du modèle de l'AAR ne peut amener à une valeur de la performance temporelle étudiée qui soit égale à τ.
- P3 est vraie indique qu'il y a au moins une évolution du modèle de l'AAR pour laquelle la valeur de la performance temporelle étudiée est strictement supérieure à τ.

De la vérification ces trois propriétés logiques il est possible de déduire que τ est inférieur strictement à Min(pt), pour $\tau = \tau_1$.

Des raisonnements identiques peuvent être faits pour les autres valeurs de τ.

Le tableau 2.2 peut sembler incomplet en première approche car toutes les combinaisons des trois propriétés ne sont pas représentées (il y a 8 combinaisons possibles). Ceci est volontaire car les combinaisons non représentées ne sont pas réalistes :

- La combinaison P2 est vraie et P1 et P3 sont fausses signifierait qu'il n'y a qu'une seule valeur pour la performance temporelle étudiée. Cette situation ne semble pas possible à cause du fort indéterminisme des modèles d'AAR utilisés et ne correspond à aucun résultat de mesure sur les AAR.
- La combinaison avec toutes les propriétés fausses correspondrait à une erreur de modélisation car le modèle n'évoluerait pas correctement. En effet, il n'y aurait pas d'émission de la sortie alors que l'évènement d'entrée a bien eu lieu.
- La combinaison P2 est fausse et P1 et P3 sont vraies indiquerait une performance temporelle dont la distribution n'est pas continue. Cette possibilité n'est pas envisagée, car non réaliste pour les AAR étudiées.

2.4 Modification du paramètre τ

Comme nous l'avons vu précédemment (figure 2.4), le paramètre temporel τ doit être modifié après chaque vérification, si la borne n'a pas été obtenue, ce qui correspond aux lignes 1, 2, 3 et 5 du tableau 2.2 si la borne recherchée est la borne supérieure de la performance temporelle et aux lignes 1, 3, 4 et 5 du tableau si la borne recherchée est la borne inférieure. Pour modifier ce paramètre, deux algorithmes de **recherche par dichotomie** ont été proposés, l'un pour obtenir la borne supérieure, l'autre pour la borne inférieure.

2.4.1 Présentation des algorithmes

Pour les deux algorithmes, la valeur du paramètre τ est calculée au moyen de deux valeurs limites L (limite inférieure) et H (limite supérieure). A partir des valeurs initiales Linit et Hinit qui correspondent aux limites du domaine de recherche, les valeurs de L et de H sont modifiées à chaque itération en fonction du résultat de preuve obtenu lors de la vérification précédente.

Dans le cas de la recherche de la borne maximale de la performance temporelle étudiée, il suffit d'analyser les résultats de vérification des propriétés P2 et P3. Quand τ est inférieur à la borne maximale de la performance temporelle étudiée (cas pour lesquels P3 est vraie), la nouvelle valeur de L correspond à la valeur courante de τ, tandis que la valeur de H

Algorithm 1 Algorithme de recherche par dichotomie pour obtenir la borne supérieure d'une performance temporelle

$L \leftarrow Linit$, $H \leftarrow Hinit$
while $H \neq L$ **do**
  $\tau \leftarrow L + \left\lfloor \frac{H-L}{2} \right\rfloor$
  vérifier les propriétés P2 et P3
  if (P3 est vraie) **then**
    $L \leftarrow \tau$
  end if
  if (P3 est fausse) $\wedge$ (P2 est fausse) **then**
    $H \leftarrow \tau$
  end if
  if (P3 est fausse) $\wedge$ (P2 est vraie) **then**
    $H \leftarrow \tau$
    $L \leftarrow \tau$
  end if
end while
$Max(pt) \leftarrow \tau$
return Max(pt) la valeur maximale de la performance temporelle étudiée

reste inchangée. Quand τ est strictement supérieur à la borne maximale de la performance temporelle étudiée (P3 et P2 sont tous les deux fausses), la nouvelle valeur de H correspond à la valeur courante de τ tandis que la valeur de L reste inchangée. L'algorithme s'arrête quand P3 est fausse et P2 est vraie.

De la même manière, pour obtenir la borne inférieure d'une performance temporelle, l'algorithme 2 a été développé. Il fonctionne sur le même principe mais utilise uniquement les résultats obtenus lors de la vérification des propriétés P1 et P2.

Pour cet algorithme, quand τ est supérieur à la borne minimale de la performance temporelle étudiée (cas pour lesquels P1 est vraie), la nouvelle valeur de H correspond à la valeur courante de τ, tandis que la valeur de L reste inchangée. Quand τ est strictement inférieur à la borne minimale de la performance temporelle étudiée (P1 et P2 sont tous les deux fausses), la nouvelle valeur de L correspond à la valeur courante de τ tandis que la valeur de H reste inchangée. L'algorithme s'arrête quand P1 est fausse et P2 est vraie.

Si les deux algorithmes donnent la même valeur, cela signifie que les bornes supérieure et inférieure de la distribution sont confondues. La performance temporelle étudiée se limiterait donc à une seule valeur et non à une distribution de valeurs. Cela correspondrait au cas où la propriété P2 serait vraie et les propriétés P1 et P3 seraient fausses, situation qui n'a pas été prise en compte précédemment (tableau 2.2) mais qui peut donc quand même être détectée.

Algorithm 2 Algorithme de recherche par dichotomie pour obtenir la borne inférieure d'une performance temporelle

$L \leftarrow Linit$, $H \leftarrow Hinit$
while $H \neq L$ **do**
  $\tau \leftarrow L + \left\lfloor \frac{H-L}{2} \right\rfloor$
  vérifier les propriétés P1 et P2
  if (P1 est vraie) **then**
    $H \leftarrow \tau$
  end if
  if (P1 est fausse) $\wedge$ (P2 est fausse) **then**
    $L \leftarrow \tau$
  end if
  if (P1 est fausse) $\wedge$ (P2 est vraie) **then**
    $H \leftarrow \tau$
    $L \leftarrow \tau$
  end if
end while
$Min(pt) \leftarrow \tau$
return Min(pt) la valeur minimale de la performance temporelle étudiée

2.4.2 Type des nombres et résolution temporelle

Il faut noter que, dans ces deux algorithmes, tous les nombres doivent être des entiers naturels car Uppaal ne supporte que ce format de nombres pour l'instanciation des paramètres. Il sera donc nécessaire de faire particulièrement attention dans les algorithmes de recherche par dichotomie de toujours prendre la partie entière des nombres pour ne pas risquer d'avoir une valeur de τ non entière.

Il faut également que les valeurs des paramètres des modèles d'AAR (période d'IOscanning, temps de réponse à une requête, ...) soient exprimées par des entiers. Pour cela il faut avoir une résolution temporelle (valeur réelle d'une unité de temps du model-checker) qui permette de représenter toutes les valeurs utiles sans pour autant obliger le model-checker à manipuler des nombres trop grands. Ceux-ci peuvent poser des problèmes de représentation et de stockage pour un système 32 bits. De plus, plus la résolution sera fine, plus le nombre d'itérations des algorithmes avant de converger sera grand et, par suite, plus la durée de calcul sera importante.

Il a donc été choisi de prendre la résolution temporelle la plus grande possible qui permette de représenter toutes les valeurs des paramètres des modèles d'AAR. Pour ces travaux, cela revient à **fixer la résolution temporelle à 10 μs**, ce qui signifie que si, par exemple, la borne supérieure de la performance temporelle étudiée vaut 3124 unités de temps, elle correspond en réalité à 31,24 ms. Cette résolution a permis de représenter toutes les valeurs des paramètres utilisés sans pour autant que le model-checker n'ait à manipuler des nombres trop grands.

2.4.3 Choix des valeurs initiales Linit et Hinit

Pour ces deux algorithmes, le nombre d'itérations nécessaires pour converger vers la borne supérieure (ou inférieure) de la performance temporelle étudiée dépend des valeurs initiales prises pour les deux limites Linit et Hinit. Il est donc préférable de pouvoir estimer au préalable les valeurs d'initialisation des algorithmes afin d'être sûr que la borne puisse bien être trouvée (elle doit être entre les deux valeurs d'initialisation) et afin que le calcul ne soit pas inutilement allongé.

Pour avoir les temps de calculs les plus réduits, les limites du domaine de recherche par dichotomie doivent donc être choisies en fonction de la performance étudiée, de la borne étudiée (inférieure ou supérieure) ainsi que du modèle.

La solution la plus simple et la plus efficace consiste à définir ces limites à partir de résultats d'études précédentes (simulation, mesure, ...). Si ce type d'étude a déjà été réalisé sur le système à étudier, il est facile de limiter très fortement le domaine de recherche, sous réserve d'avoir une équivalence entre les modèles de simulation et les modèles de vérification ou entre les modèles de vérification et la réalité. De par leur origine, les valeurs obtenues par simulation ou par mesure sont toutes comprises entre les bornes inférieure et supérieure de la distribution. Il est ainsi facile de déterminer la limite maximale de la zone de recherche de la borne inférieure (la plus petite valeur observée ou simulée) ainsi que la limite minimale de la zone de recherche de la borne supérieure (la plus grande valeur observée ou simulée). Par contre, les deux autres limites ne sont pas aussi simples à fixer. Suivant la confiance accordée aux résultats de simulation ou de mesure, il est possible de déterminer ces limites à partir des valeurs extrêmes observées ou simulées. Par exemple la limite minimale initiale de la zone de recherche de la borne inférieure peut être prise égale à 80% de la valeur minimale observée (ou simulée) et la limite maximale de la zone de recherche de la borne supérieure peut être prise égale à 120% de la valeur maximale observée (ou simulée).

Il est par ailleurs évident que si une des bornes a déjà été déterminée, sa valeur peut servir pour limiter la zone de recherche pour la deuxième borne de la performance temporelle étudiée.

Dans le cas particulier de l'étude de différences de temps de réponse, si les deux automates ont des configurations proches, il est tout a fait possible que la valeur minimale de cette différence soit nulle. A cause de cela, la limite inférieure du domaine de recherche, a été systématiquement choisie égale 0. En ce qui concerne la borne maximale, elle sera dans tous les cas inférieure au plus grand des temps de réponse.

Pour faciliter le choix de Hinit, l'annexe B présente deux aides pour le choix de cette valeur initiale quand aucune information n'est disponible sur l'ordre de grandeur des valeurs des performances temporelles étudiées. La première correspond à une méthode de recherche des bornes qui ne nécessite pas de valeur d'initialisation pour Hinit (limite supérieure initiale de la recherche par dichotomie) tandis que la seconde est une représentation au pire des cas de l'évolution d'une AAR permettant d'estimer la valeur maximale d'un temps de réponse.

Les deux algorithmes proposés dans ce chapitre ont été programmés à l'aide

du langage Python afin d'automatiser la méthode présentée à la figure 2.4. Le prototype logiciel réalisé a été interfacé avec le model-checker Uppaal. Ceci nous a permis de traiter plusieurs cas d'AAR, présentés au chapitre 5, afin d'étudier l'intérêt de notre contribution.

Chapitre 3

Modélisation des Architectures d'Automatisation en Réseau

L'OBJECTIF principal de ce chapitre est de présenter les modèles génériques que nous avons développés et qui nous ont servi pour construire des modèles formels d'AAR qui pourront être analysés par la méthode proposée au chapitre précédent. En préambule à cette présentation, nous rappellerons la syntaxe et la sémantique des automates temporisés utilisés dans ces travaux et énoncerons nos principes de construction de modèles d'AAR. Le chapitre se termine par une évaluation de la capacité du model-checker choisi à traiter les modèles d'AAR.

Sommaire

3.1 Syntaxe et sémantique des modèles formels développés

Ayant choisi UPPAAL ([LPY97]) comme outil de vérification formelle pour ces travaux, les modèles formels à développer devront adopter la syntaxe et la sémantique de cet outil, et non celles proposées initialement par [AD94] pour décrire des systèmes temporisés. Le but de cette partie est de présenter ces caractéristiques.

3.1.1 Model-checker Uppaal

Dans ces travaux, le model-checker temporisé UPPAAL a été utilisé. C'est un outil pour la modélisation, la simulation et la vérification de systèmes temps réel qui a été développé conjointement par Uppsala University, Suède et Aalborg University, Danemark, principalement par W. Yi, K. G. Larsen et P. Pettersson. La première version de cet outil date de 1995 et depuis il n'a cessé d'être amélioré dans le but de réduire sa sensibilité au phénomène d'explosion combinatoire bien connu lors du passage à l'échelle.

UPPAAL est composé de trois éléments principaux : un langage pour la représentation des systèmes, un simulateur, et un model-checker. Le langage de représentation est un langage non-déterministe avec gardes et systèmes de données. Il permet la représentation de comportements de systèmes par le moyen de réseaux d'automates communicants possédant des horloges et des variables. Le model-checker permet de vérifier des propriétés d'atteignabilité ($E <>$), d'invariance ($A[]$), ...cf. partie 2.1.2 par exploration de l'ensemble de l'espace d'états du système. Ce model-checker offre également des facilités pour le développement de modèles par l'intermédiaire de son interface graphique, souvent plus explicite que des modèles textuels et surtout par son outil de simulation qui permet de visualiser facilement certaines évolutions du modèle de l'AAR (par exemple une évolution qui aboutirait à la borne supérieure de la performance temporelle étudiée) et ainsi de juger de leur cohérence par rapport au comportement réel du système.

Pour automatiser la méthode d'obtention des bornes des performances temporelles des AAR (figure 2.4), il était nécessaire que l'outil développé dans ce but puisse éditer le modèle de l'AAR et les propriétés à vérifier, lancer la vérification et enfin lire et interpréter les résultats donnés par le model-checker. Tout cela est possible avec UPPAAL car les modèles formels à étudier sont représentés en XML et les propriétés à vérifier peuvent être éditées avec un éditeur de texte standard. Le model-checker *verifyta* (l'outil de vérification de UPPAAL) peut être lancé en ligne de commande et le résultat de l'analyse peut être transcrit dans un fichier texte standard et sous une forme compréhensible par un humain.

Une fois cet outil de model-checking choisi, il est nécessaire de connaitre les spécificités de la représentation qu'il utilise.

3.1.2 Automates temporisés et réseaux d'automates temporisés communicants

Dans cette partie, la syntaxe et la sémantique des automates temporisés et réseaux d'automates temporisés utilisés par UPPAAL sont présentées. Seules les spécificités utiles pour ces travaux seront détaillées.

3.1.2.1 Automates temporisés

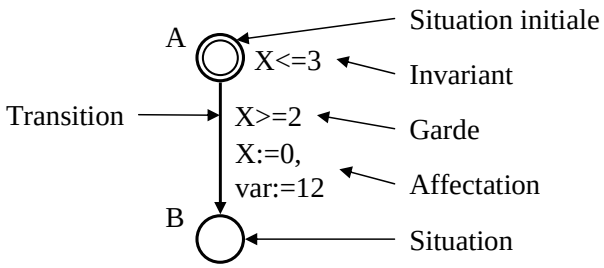


FIG. 3.1 – Exemple d'un automate temporisé élémentaire

La figure 3.1 représente un automate temporisé composé de deux situations (A et B) et d'une transition. Une horloge X (à valeur réelle positive) est associée à cet automate. Le temps, mesuré par cette horloge, ne s'écoule que dans les situations ; le franchissement des transitions se fait à temps nul. Des variables (entières ou booléennes) peuvent également être utilisées. Ces variables et horloges peuvent être utilisées dans plusieurs types d'expressions :

- Une **garde** est une expression particulière qui satisfait les conditions suivantes : elle n'a pas d'effet sur les variables ; elle est évaluée comme un booléen ; seules des horloges, des variables et des constantes peuvent y être utilisées ; des horloges ou des différences d'horloges sont seulement comparées à des expressions entières ; les gardes sur des horloges sont essentiellement des conjonctions.
- Une **affectation** est une liste d'expressions séparées par des virgules ayant un effet sur les variables ; ces expressions ne peuvent faire référence qu'à des horloges, des variables ou des constantes, sachant qu'il est possible d'affecter uniquement des valeurs entières aux horloges.
- Un **invariant** est une expression particulière qui satisfait les conditions suivantes : il n'a pas d'effet sur les variables ; seules des horloges, des variables et des constantes peuvent y être utilisées ; c'est une conjonction de conditions de la forme $x < e$ or $x \leq e$ où x est une horloge et e une variable ou une constante entière.

Il faut remarquer que le temps s'écoule de façon continue car les horloges ont des valeurs réelles positives mais celles-ci ne peuvent être comparées (dans les gardes et les invariants) qu'à des entiers et n'être affectées qu'avec des valeurs entières.

Pour l'automate de la figure 3.1, l'horloge X est utilisée dans l'invariant associé à la situation A et la garde de la transition. Elle est également remise à 0 dans l'affectation liée au franchissement de la transition. L'évolution de cet automate élémentaire se fait ainsi :

- Attente (évolution de l'horloge X) dans la situation initiale A que la garde associée à la transition soit vraie pour que cette dernière puisse être franchie.
- Franchissement de la transition quand $X \geq 2$ (imposé par la garde) et $X \leq 3$ (imposé par l'invariant de la situation A). Lors de ce franchissement, l'horloge X est remise à 0 et la variable var est affectée à 12.

D'un point de vue formel, un automate temporisé est un graphe orienté fini annoté avec des conditions et des réinitialisations d'horloges. L'ensemble de ses situations est noté L avec $l0$ la situation initiale. Les notations suivantes seront utilisées : C est un ensemble d'horloges, $B(C)$ est un ensemble de conjonctions sur des conditions simples de la forme $x \bowtie c$ ou $(x - y) \bowtie c$, avec $x, y \in C$, $c \in \mathbb{N}$ et $\bowtie \in \{<, \leq, =, \geq, >\}$. On notera I l'ensemble des invariants associés aux situations, A l'ensemble d'actions associées aux transitions (franchissements, affectations) et E l'ensemble des transitions.

Une valuation d'horloge est une fonction u : $C \rightarrow \mathbb{R}^+$ de l'ensemble des horloges dans les réels positifs. Soit $\mathbb{R}^C$ l'ensemble de toutes les valuations d'horloges. Soit $u_0(x) = 0$ pour tout $x \in C$. La notation $u \in I(l)$ sera utilisée pour signifier que u satisfait $I(l)$. Il est maintenant possible de définir la sémantique associée aux automates temporisés.

Définition 1 (Sémantique des Automates Temporisés). Soit $(L, l0, C, A, E, I)$ un automate temporisé. Sa sémantique est définie comme un système de transitions étiquetées $\langle S, s_0, \rightarrow \rangle$, avec $S \subseteq L \times \mathbb{R}^C$ l'ensemble des états, $s_0 = (l_0, u_0)$ l'état initial, et $\rightarrow \subseteq S \times S$ la relation telle que :

- $(l, u) \xrightarrow{d} (l, u + d)$ si $\forall d' : 0 \leq d' \leq d \Rightarrow u + d' \in I(l)$, et
- $(l, u) \xrightarrow{a} (l', u')$ s'il existe $e = (l, a, g, r, l') \in E$ telle que $u \in g$, $u' = [r \mapsto 0]\, u$ et $u' \in I(l')$,

où pour $d \in \mathbb{R}^+$, $u + d$ affecte à chaque horloge x de C la valeur $u(x) + d$ et $u' = [r \mapsto 0]\, u$ représente la valuation d'horloge qui affecte chaque horloge x à 0 et satisfait u sur $C \setminus r$.

La première partie de cette relation représente l'évolution du temps sans changement de situation ; la seconde le franchissement d'une transition conduisant à une nouvelle situation.

3.1.2.2 Réseaux d'automates temporisés

Les modèles temporisés sont souvent structurés sous la forme d'un réseau d'automates temporisés qui peuvent évoluer indépendamment les uns des autres ou au contraire se synchroniser, ceci au moyen d'un canal de communication qui permet de synchroniser les évolutions de deux automates. Une **étiquette de synchronisation** peut être de la forme expression ! (émission du message de synchronisation), expression ? (réception du message de synchronisation) ou être vide (pas de synchronisation).

Les évolutions des réseaux d'automates temporisés communicants sont illustrées par la figure 3.2.

Les trois types d'évolutions possibles sont les suivantes :

- **Évolution des horloges sans changement de situation.** Dans l'état où les automates sont dans les situations A et D et l'horloge X vaut 0, seule l'horloge peut évoluer (jusqu'à 3 pour ne pas violer l'invariant).

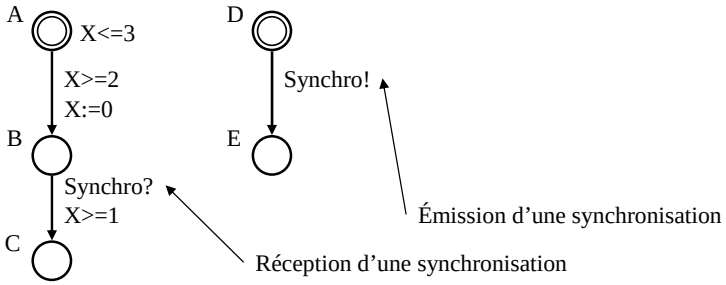


FIG. 3.2 – Illustration des évolutions des réseaux d'automates temporisés communicants

- **Franchissement d'une transition dans un automate.** Dans l'état où les automates sont dans les situations A et D et l'horloge X vaut entre 2 et 3, le franchissement de la transition entre les situations A et B peut s'effectuer (le franchissement devra avoir eu lieu au plus tard quand l'horloge vaudra 3). Il aura pour effet de remettre l'horloge X à 0.
- **Synchronisation des franchissements de deux transitions dans deux automates différents.** Dans l'état où les automates sont dans les situations B et D et l'horloge X est supérieure ou égale à 1, le franchissement synchrone des transitions menant dans les situations C et E est possible.

Il est possible de définir un réseau d'automates temporisés régi par un même ensemble d'horloges et d'actions, comme un ensemble de n automates temporisés $A_i = (L_i, l_i^0, C, A, E_i, I_i)$, $1 \leq i \leq n$ dont les situations sont représentées par un vecteur $\bar{l} = (l_1, \ldots, l_n)$. Les fonctions d'invariants sont composées en une fonction commune sur les vecteurs de situations $I(\bar{l}) = \wedge_i I_i(l_i)$. Le vecteur pour lequel le $i^{ème}$ élément l_i de $\bar{l}$ est remplacé par l_i' est noté : $\bar{l}[l_i'/l_i]$. Nous avons alors :

Définition 2 (Sémantique d'un réseau d'Automates Temporisés). Soit $A_i = (L_i, l_i^0, C, A, E_i, I_i)$ un réseau de n automates temporisés. Soit $\bar{l}_0 = (l_i^0, \ldots l_n^0)$ le vecteur des situations initiales. La sémantique est définie comme un système de transitions $\langle S, s_0, \rightarrow \rangle$, où $S = (L_1 \times \cdots \times L_n) \times \mathbb{R}^C$ est l'ensemble des états, $s_0 = (\bar{l}_0, u_0)$ est l'état initial, et $\rightarrow \subseteq S \times S$ est la relation de transitions définie par :

- $(\bar{l}, u) \xrightarrow{d} (\bar{l}, u + d)$ si $\forall d' : 0 \leq d' \leq d \Longrightarrow u + d' \in I(\bar{l})$.
- $(\bar{l}, u) \xrightarrow{a} (\bar{l}[l_i'/l_i], u')$ s'il existe $l_i \xrightarrow{\tau gr} l'i$ tel que $u \in g$, $u' = [r \mapsto 0]u$ et $u' \in I(\bar{l}[l_i'/l_i])$.
- $(\bar{l}, u) \xrightarrow{a} (\bar{l}[l_j'/l_j, l_i'/l_i], u')$ s'il existe $l_i \xrightarrow{c?g_i r_i} l'i$ et $l_j \xrightarrow{c!g_j r_j} l'j$ tel que $u \in (g_i \wedge g_j$, $u' = [r_i \cup r_j \mapsto 0]u$ et $u' \in I(\bar{l}[l_j'/l_j, l_i'/l_i])$.

La première partie de cette relation représente l'évolution du temps sans changement de situation ; la seconde le franchissement d'une transition conduisant à une nouvelle situation et enfin la troisième le franchissement simultané de deux transitions par synchronisation des évolutions de deux automates.

Enfin, UPPAAL permet également de définir des situations dites "urgentes" et "obligées" :

- **Les situations de type urgentes** sont sémantiquement équivalentes à l'ajout

d'une horloge supplémentaire x qui est mise à 0 sur tous les arcs atteignant cette situation, situation à laquelle est ajoutée un invariant ≤ 0. **Le temps ne s'écoule pas** quand le système est dans une situation urgente. Cette représentation n'est au final qu'une facilité d'écriture.

– **Les situations "obligées", notées committed**, sont encore plus restrictives pour l'exécution que les situations urgentes. Un état est committed si l'une des situations dans l'état est committed. Un état committed ne peut être retardé (même en temps logique) et la prochaine évolution doit correspondre au franchissement d'une transition de sortie d'au moins une des situations committed. Ce type de situations, **qui impose une priorité**, n'entre pas dans la sémantique initiale des automates temporisés.

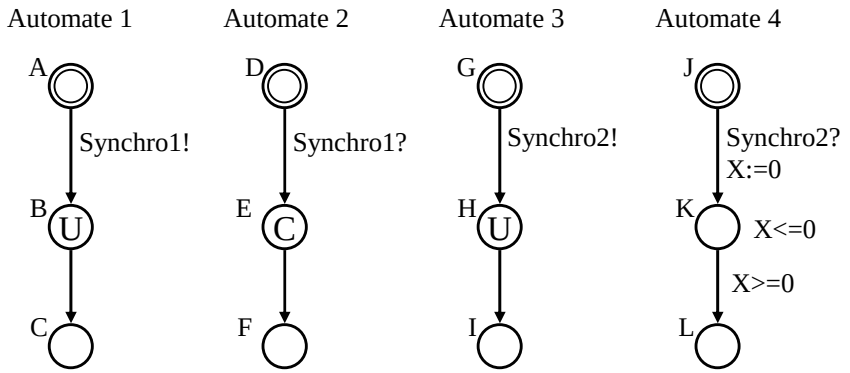


FIG. 3.3 – Représentation des situations spécifiques possibles dans UPPAAL

La figure 3.3 montre la représentation de situations "urgentes" (U) et "obligées" (C). La situation E est une situation "obligée" alors que les situations B et H sont des situations "urgentes". La situation K a le même comportement qu'une situation "urgente" mais est représentée en utilisant la syntaxe de base basée sur une horloge X.

Les évolutions des automates 1 et 2 sont les suivantes :
– attente dans les états initiaux A et D,
– évolution par le franchissement simultané des transitions de A vers B et de D vers E à cause de la synchronisation Synchro1,
– la situation E étant committed, la transition de E vers F doit être franchie de suite,
– la situation B étant urgente, la transition de B vers C doit être franchie sans que le temps s'écoule dans la situation B (mais après la transition de E vers F).

Pour les automates 3 et 4, les évolutions sont :
– attente dans les états initiaux G et J,
– évolution par le franchissement simultané des transitions de G vers H et de J vers K à cause de la synchronisation Synchro2 ; lors de ce franchissement, l'horloge X est réinitialisée,
– la situation H étant urgente et la situation K devant avoir une durée nulle (invariant associé), les deux transitions de H vers I et de K vers L peuvent être franchies dans n'importe quel ordre mais ces deux franchissements se feront à la même date (le

temps ne s'écoule pas entre ces deux franchissements).

Toutes les sémantiques présentées dans cette partie sont utilisées lors de la modélisation des composants d'AAR.

3.2 Principes de construction des modèles formels d'AAR

Pour faciliter la compréhension et la conception des modèles d'architectures d'automatisation, il a été choisi de modéliser chaque composant indépendamment des autres et, dans ce but, deux principes ont été mis en oeuvre :

- création d'une bibliothèque de modèles génériques de composants,
- synchronisation entre les différents modèles de composants pour réaliser le modèle de l'AAR.

3.2.1 Bibliothèque de modèles génériques de composants

La bibliothèque de modèles génériques de composants (figure 3.4) est composée d'un modèle générique de chaque famille de composant (un modèle générique pour les processeurs de calcul, un pour les cartes de communication, ...).

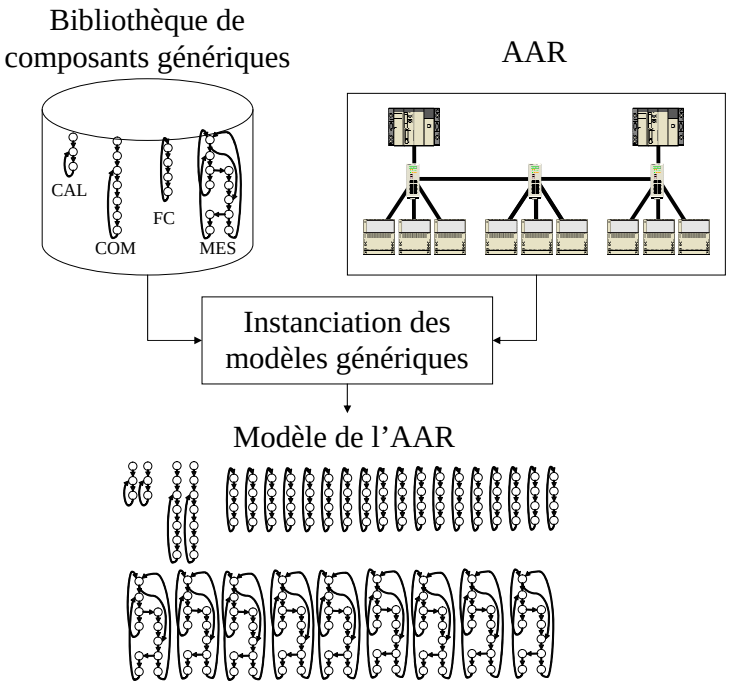


FIG. 3.4 – Utilisation de la bibliothèque de composants lors de l'instanciation d'un modèle d'AAR

Tous les composants d'une même famille ont le même comportement mais la durée de chacune de leurs évolutions peut être différente. Des paramètres (de type entier) sont,

pour cela, utilisés dans les invariants et les gardes pour pouvoir représenter toutes les durées possibles pour les évolutions. Les valeurs de ces paramètres sont indiquées lors de l'instanciation du modèle de composant.

Cette représentation par famille de composants permet de définir simplement différentes architectures d'automatisation. Lors de la conception du modèle de l'architecture, il est possible de choisir quels sont les composants à instancier (par exemple 3 composants de type A, 2 de type B et 4 de type C) et de choisir les valeurs de chaque paramètre pour les différents modèles (par exemple pour les composants de type B, la durée d vaudra 3 ms pour le premier composant et 4 ms pour le second). De cette façon, il est possible de représenter facilement et rapidement toutes les architectures d'automatisation à partir d'une bibliothèque assez réduite de modèles génériques de composants qui sont décrits dans la suite de ce chapitre.

3.2.2 Synchronisation des modèles instanciés de composants

L'utilisation de modèles de composants indépendants va nécessiter des échanges de données et des synchronisations entre modèles de composants. Les échanges de données (trames, variables d'entrée/sortie de l'AAR, ...) sont modélisés par des variables partagées, les synchronisations par des étiquettes de synchronisation.

Pour la synchronisation des évolutions de deux automates, des canaux de communication sont utilisés. Comme ils ne peuvent synchroniser plus de deux automates, il sera parfois nécessaire de réaliser plusieurs synchronisations pour synchroniser plus de deux automates entre eux. Pour qu'aucune autre évolution ne puisse se faire au cours de ces synchronisations "simultanées", les situations entre deux synchronisations "simultanées" devront être de type "committed" (figure 3.5). Sur cet exemple, les automates évoluent depuis l'état initial (situations A, D et F) par le franchissement simultané des transitions entre A et B et entre D et E sur l'occurrence de la synchronisation Synchro1. Comme la situation B est une situation "committed", l'automate 1 est obligé d'évoluer. Ainsi la transition entre les situations B et C est franchie, ce qui impose l'évolution de l'automate 3 vers la situation G au travers de le synchronisation Synchro2.

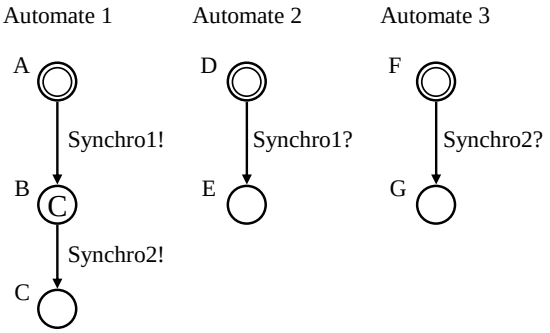


Fig. 3.5 – Synchronisation de trois automates

L'utilisation de variables partagées permet l'échange de données (données écrites par un automate et lues par un autre). Il sera nécessaire de prendre garde de ne pas affec-

ter une valeur à une variable et de simultanément tester cette valeur (dans une garde notamment), par exemple si les deux évolutions sont synchronisées par un canal de synchronisation (figure 3.6). En effet l'ordre des opérations dans UPPAAL est de vérifier en premier les gardes (pour tester si une transition est franchissable) puis, si elle est franchissable, d'affecter les valeurs aux variables. De cette façon, le test de la garde se fait systématiquement sur la valeur précédente de la variable et non sur la valeur mise à jour.

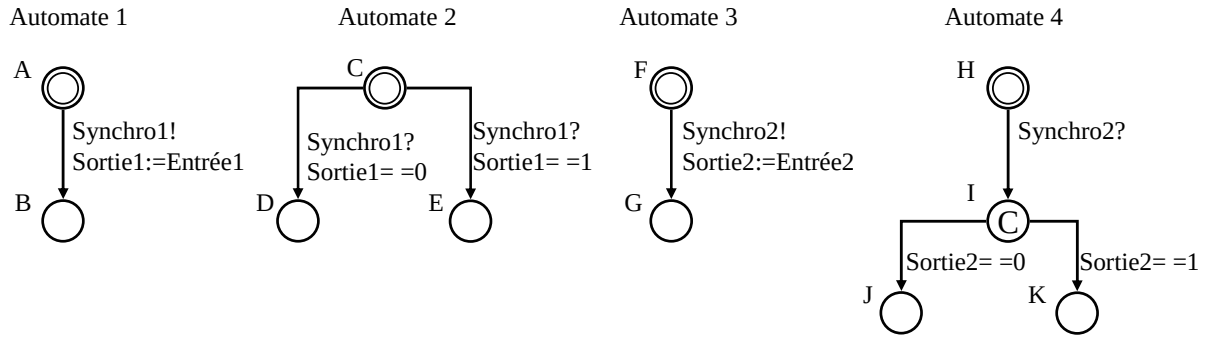


FIG. 3.6 – Synchronisation d'automates avec échanges de données

Concrètement, dans le cas représenté par la figure 3.6, si l'état initial du système correspond à ce que l'automate 1 soit dans la situation A, l'automate 2 dans la situation C, que Sortie1 = 0 et Entrée1 = 1, le système évoluera ainsi :

- état initial : A, C, Sortie1 = 0 et Entrée1 = 1,
- vérification des gardes des transitions : les gardes des transitions de A vers B et de C vers D sont vérifiées, pas celle de la transition de C vers E,
- franchissement simultané (par le moyen de la synchronisation Synchro1) des transitions dont les gardes sont vérifiées et affectation des variables partagées (Sortie1 passe à 1),
- état final : B, D, Sortie1 = 1 et Entrée1 = 1.

Cette situation finale peut sembler surprenante et souvent ne correspond pas au comportement que l'on souhaite modéliser (affectation de la nouvelle valeur de Sortie avant la vérification des gardes franchissables). Il faut alors utiliser la modélisation faite avec les automates 3 et 4. L'évolution du système sera la suivante :

- état initial : F, H, Sortie2 = 0 et Entrée2 = 1,
- vérification des gardes des transitions : les gardes des transitions de F vers G et de H vers I sont vérifiées,
- franchissement simultané (par le moyen de la synchronisation Synchro2) des transitions dont les gardes sont vérifiées et affectation des variables partagées (Sortie2 passe à 1),
- vérification des gardes des transitions : la garde de la transition de I vers K est vérifiée,
- franchissement de cette transition,
- état final : G, K, Sortie2 = 1 et Entrée2 = 1.

3.3 Modélisation des composants des AAR

Dans cette partie, les choix de modélisation amenant aux modèles génériques des composants seront présentés. Ces composants génériques représentent le comportement des composants des AAR présentés dans le chapitre 1. Pour bien comprendre les modèles des différents composants, il est important de connaitre les échanges de données et les synchronisations qu'il existe entre eux. Dans ce but, la figure 3.7 les représente dans le cas où seulement un API et un MES sont considérés.

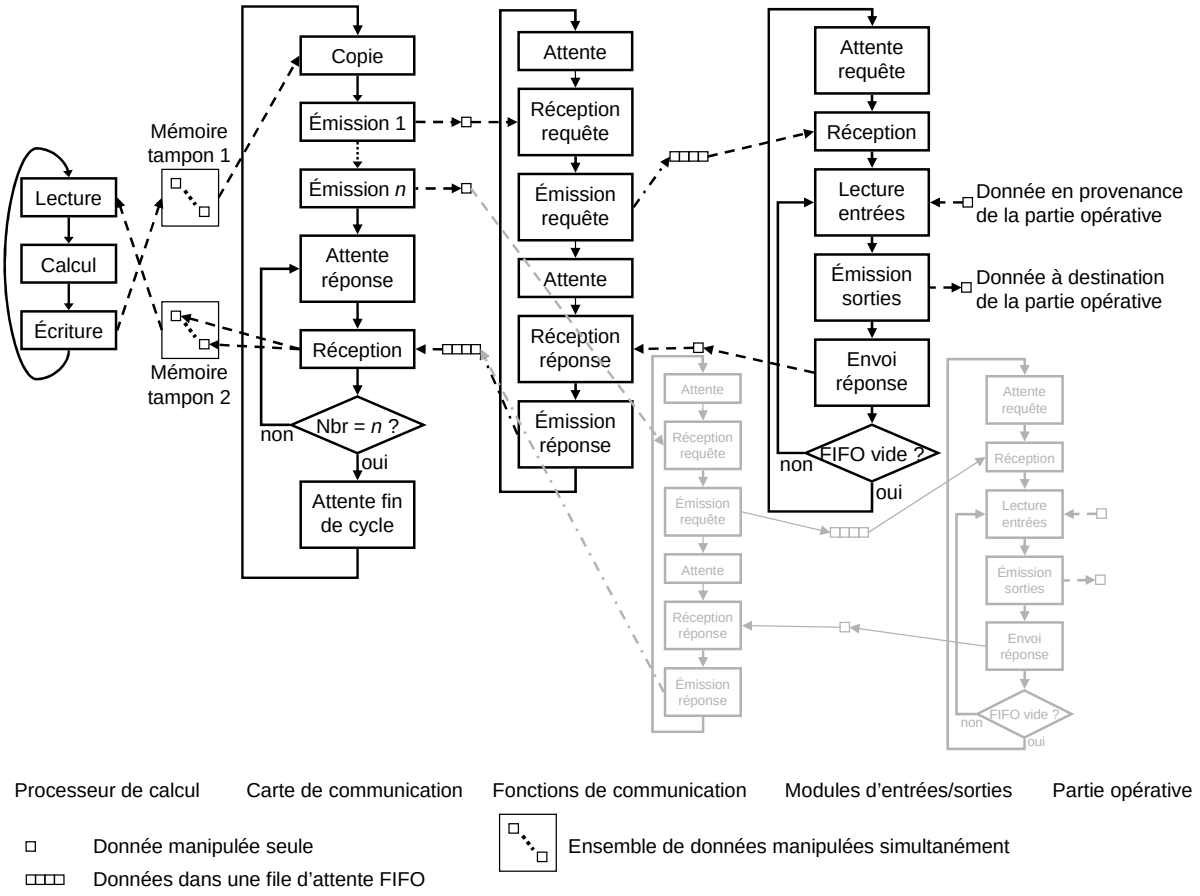


FIG. 3.7 – Synchronisations et échanges de données entre composants d'AAR

Avant de présenter les modèles génériques des composants, il est nécessaire d'indiquer une précaution à prendre pour ne pas obtenir des résultats aberrants lors de la vérification des modèles d'AAR. Physiquement, l'AAR est composée de différents éléments interagissant entre eux. Il est évident que les performances attendues d'un tel système, système qui n'est pas sensé être arrêté au cours de son utilisation, sont celles obtenues en **régime permanent**, c'est à dire quand tous les composants sont dans leur fonctionnement nominal (en dehors des phases de démarrage et d'arrêt). Il en est donc de même lors de la vérification du modèle d'AAR ; les différents automates modélisant le comportement des composants doivent tous être dans un état correspondant au fonctionnement en régime établi du système. Ceci impose d'initialiser convenablement ces modèles.

3.3.1 Initialisation des modèles des composants d'une AAR

Il a été présenté dans le chapitre 1 les comportements des différents composants d'AAR. Ainsi, nous avons pu voir que le processeur de calcul et la carte de communication ont des comportements cycliques[9] et totalement désynchronisés l'un de l'autre. Au contraire, l'état normal des modules d'entrée/sortie est un état d'attente dont ils ne sortent qu'au moment où ils reçoivent une requête. Les MES n'ont donc pas besoin d'initialisation particulière puisqu'ils commenceront à évoluer lors de la réception de leur première requête. Les fonctions de communications n'ont, elles n'ont plus, pas besoin d'initialisation particulière puisqu'elles commenceront à évoluer lors de la réception de leur première requête. Par contre, pour les processeurs de calcul et les cartes de communication, il est impératif de **garantir l'absence de synchronisation** dans leurs évolutions. Leurs cycles de fonctionnement doivent donc débuter indépendamment l'un de l'autre.

L'évènement d'entrée, généré par l'automate d'environnement, doit également être émis n'importe quand pour être sûr d'explorer l'ensemble des évolutions possibles. Pour éviter que cette émission ne se fasse avant que les processeurs de calcul et les cartes de communication ne soient en régime établi, il a été choisi que :

- un processeur de calcul commence son cycle à une date quelconque,
- une carte de communication commence son cycle à une date quelconque postérieure à l'initialisation du processeur de calcul qui lui est associé,
- l'émission de l'évènement d'entrée ne peut se faire que si tous les MES liés à la performance temporelle étudiée (MES qui reçoit l'évènement d'entrée et MES qui émet(tent) l'(les) évènement(s) de sortie) ont été scrutés au moins une fois.

Tout cela permet de garantir que l'initialisation des modèles de l'AAR a bien été terminée avant l'émission de l'évènement d'entrée. En effet, le fait que les MES aient déjà été scrutés implique que les fonctions de communication ont déjà été sollicitées, ceci implique que les carte de communication sont en régime établi et par suite que c'est également le cas des processeurs de calcul.

La figure 3.8 illustre l'ordre d'initialisation des composants dans un cas simple, l'AAR n'étant constituée que d'un processeur de calcul, une carte de communication et un module d'entrée/sortie. Il est possible de voir les trois délais (variables) garantissant l'absence totale de synchronisation entre les composants et entre l'évènement d'entrée et les cycles des composants. Ces trois délais permettent de garantir que les résultats de vérification seront représentatifs du comportement en régime établi.

3.3.2 Modélisation du processeur de calcul

La représentation du processeur de calcul est relativement simple car l'exécution du programme n'est pas modélisée de façon détaillée. Pour l'étude qui nous intéresse, il est suffisant de savoir quand les informations d'entrée sont lues et quand les informations de sortie sont écrites.

[9] Nous rappelons que le processeur de calcul a un comportement cyclique et la carte de communication un comportement périodique. Par souci de concision, nous n'exprimerons pas cette différence dans cette sous-partie et regrouperons "cyclique" et "périodique" sous le même vocable "cyclique".

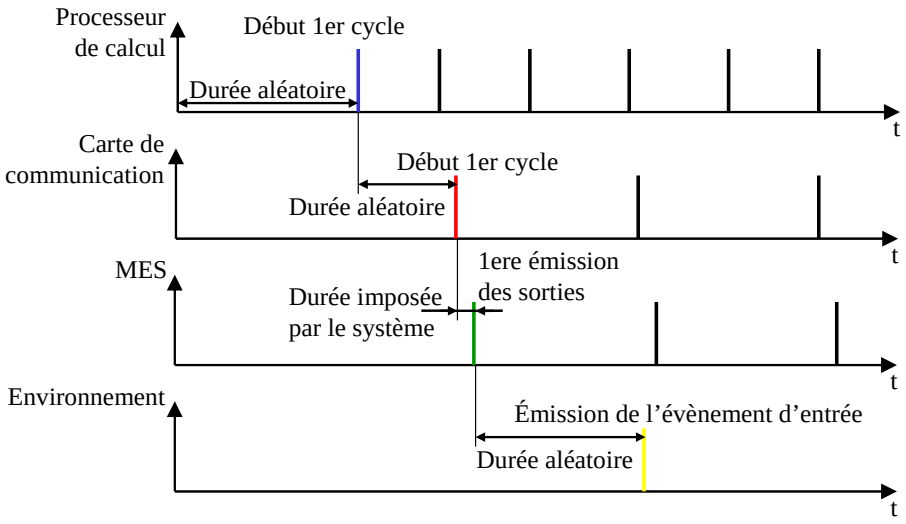


FIG. 3.8 – Ordre d'initialisation des composants

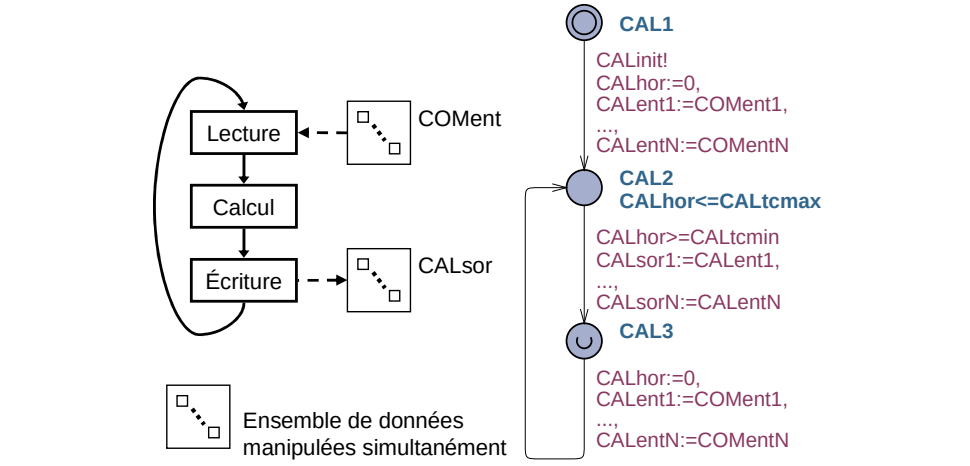
Comme le montre la figure 3.9, le fonctionnement d'un processeur de calcul est cyclique et comporte trois étapes, la lecture des entrées (le processeur de calcul copie l'ensemble des valeurs des données mises à disposition par la carte de communication COMent dans sa mémoire interne CALent), le calcul des sorties et l'écriture des sorties (le processeur de calcul met à disposition de la carte de communication les nouvelles valeurs des sorties CALsor).

Les phases de lecture et d'écriture sont considérées de durée nulle car leurs durées sont bien inférieures à la durée d'exécution du programme (quelques μs contre quelques ms), ceci permettant de simplifier le modèle sans perdre trop de sens dans les résultats. Elles sont représentées sur le modèle par les affectations liées aux transitions de CAL1 vers CAL2 et de CAL3 vers CAL2 pour la lecture des entrées (CALent :=COMent), de CAL2 vers CAL3 pour l'écriture des sorties (CALsor :=CALent). La lecture des entrées se fait à deux endroits dans le modèle car elle doit avoir lieu lors de l'initialisation du modèle (transition CAL1 vers CAL2) et lors de chaque cycle en fonctionnement nominal (transition CAL3 vers CAL2).

La phase de calcul des sorties qui correspond à l'exécution du programme a une durée variable. Elle est représentée sur le modèle par la situation CAL2 dont l'invariant (CALhor<=CALtcmax) garantit l'évolution avant la durée maximale du temps de calcul. L'attente du temps de calcul minimal avant de commencer un nouveau cycle est garantie par la garde de la transition CAL2 vers CAL3 (CALhor>=CALtcmin).

Il est également bon de noter que, quel que soit le nombre d'entrées lues et de sorties écrites, la structure du modèle reste la même, il y a juste plus ou moins d'affectations attachées aux transitions représentant les phases de lecture et d'écriture.

En ce qui concerne l'initialisation, problème présenté en partie 3.3.1, le modèle de processeur de calcul débute par une transition (entre CAL1 et CAL2) franchie à une date quelconque, à partir de l'instant initial, pour représenter l'ensemble des évolutions possibles.



CALhor : horloge du modèle
COMent : valeur des entrées logiques mises à disposition par la carte de communication (booléen)
CALent : copie des entrées logiques dans la mémoire du processeur de calcul (booléen)
CALsor : valeur des sorties logiques mises à la disposition de la carte de communication (booléen)
CALtcmin , **CALtcmax** : durée minimale et maximale de l'exécution du programme (entiers)
CALinit! : synchronisation indiquant la fin d'initialisation du processeur de calcul

FIG. 3.9 – Comportement et modèle formel générique d'un processeur de calcul cyclique

3.3.3 Modélisation de la carte de communication

Son cycle de communication avec les MES (IOscanning) (figure 3.10) consiste à copier dans la mémoire de la carte de communication toutes les valeurs des sorties à envoyer aux MES, envoyer toutes les requêtes contenant ces sorties aux MES, attendre les réponses des MES (qui sont stockées dans une file d'attente fifo lors de leur arrivée) et traiter ces réponses dans leur ordre d'arrivée. Comme le fonctionnement est périodique, il y a toujours une phase d'attente après le traitement des réponses pour garantir cette périodicité.

Le modèle générique de carte de communication connectée à N MES se compose de quatre ensembles de situations. L'ensemble utile pour l'initialisation du modèle (COM1 et COM2), les situations représentant l'envoi des N requêtes vers les N modules d'entrée/sortie connectés à la carte de communication (COM3 à COM(N+2)), la situation représentant le traitement des réponses (stockées dans la file fifo) reçues avant la fin de la phase d'envoi des requêtes (COM(N+3)) et enfin la situation représentant l'attente de la fin du cycle d'IOscanning en traitant les réponses au fur et à mesure de leur arrivée (COM(N+4)).

Comme pour le modèle du processeur de calcul, la phase de copie des valeurs des sorties dans la mémoire de la carte (transition de COM2 à COM3) est considérée de durée nulle car sa durée est bien inférieure à la durée des autres tâches exécutées au cours du cycle d'IOscanning.

Il a été choisi pour ce modèle que **l'émission d'une requête avait une durée fixe COMd** (représentant le temps d'émission sur le réseau de la trame Ethernet contenant la requête), ce qui est modélisé par l'usage conjoint d'un invariant et d'une garde. Par exemple pour l'émission de la première requête (situation COM3) l'invariant de COM3 (COMhor<=COMd) oblige à quitter cette situation au plus tard quand l'horloge vaut

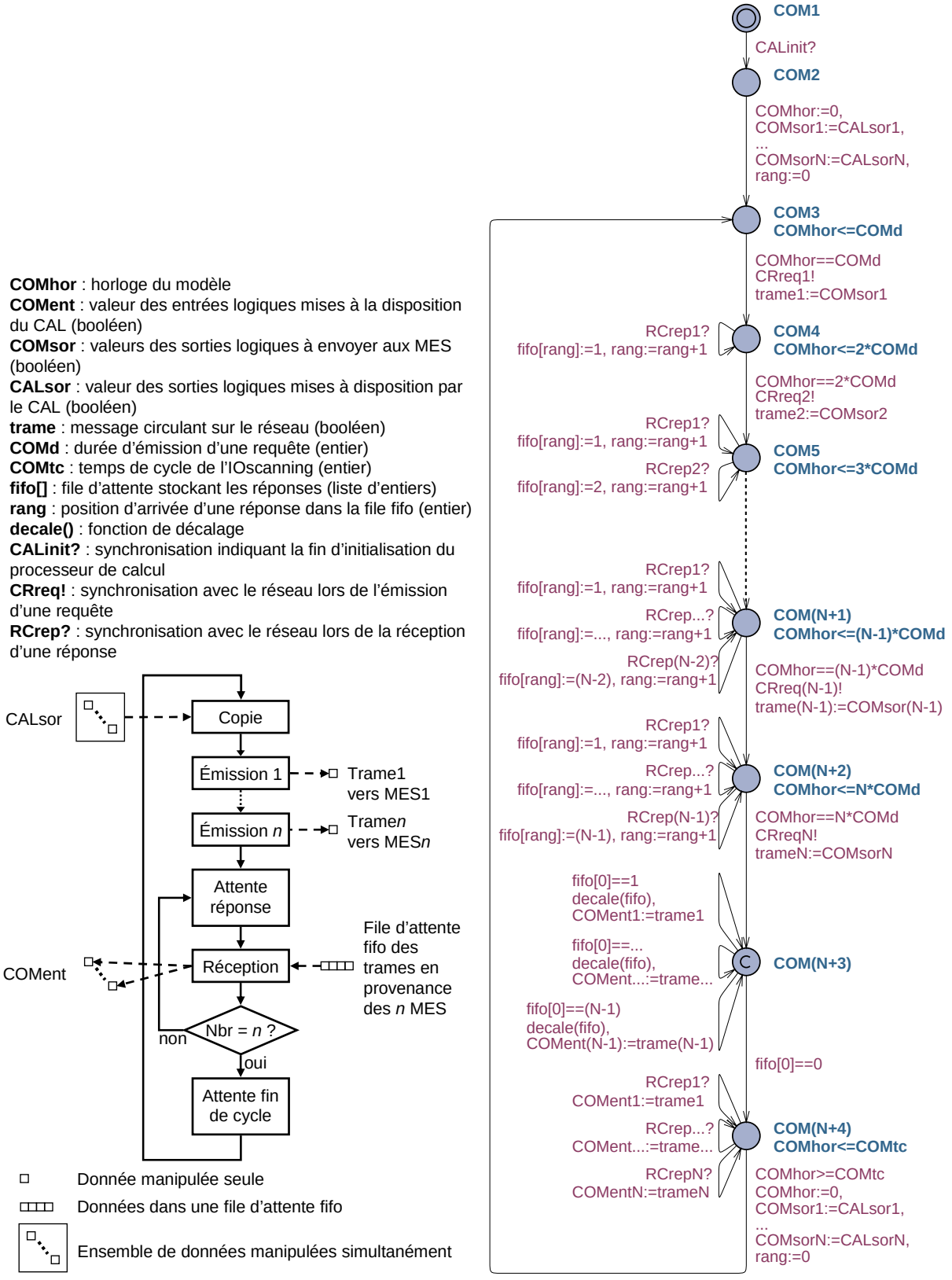


FIG. 3.10 – Comportement et modèle générique d'une carte de communication communiquant avec N MES

COMd et la garde de la transition entre COM3 et COM4 (COMhor==COMd) oblige cette transition à n'être franchie que quand l'horloge vaut COMd. A contrario, **la réception d'une réponse se fait à temps nul** (la trame Ethernet contenant la réponse est récupérée en temps masqué) ce qui permet de modéliser la réception d'une réponse uniquement par une transition de type "self-loop" (par exemple la transition entre COM4 et COM4).

La durée du cycle d'IOscanning est ici fixée par le paramètre COMtc (temps de cycle de la carte de communication), présent dans l'invariant de la situation COM(N+4) et dans la garde de la transition entre COM(N+4) et COM2. Comme ici la garde et l'invariant ont exactement la même valeur, la durée du cycle de l'IOscanning est considérée comme constante.

Il est également nécessaire de prévoir la mise en attente des réponses qui arriveraient des MES avant la fin du cycle d'émission des requêtes. Celles-ci doivent, de plus, être prises en compte selon leur ordre d'arrivée. Ainsi, dès la situation suivant l'émission de la première requête (COM4), il est nécessaire de prévoir une transition (avec une étiquette de synchronisation RCrep1 ? correspondant à la réception de la première réponse) qui ramène le modèle dans la même situation mais qui a permis de noter l'arrivée de la première réponse (la valeur de cette réponse est stockée dans la liste fifo et le nombre rang, représentant le nombre de requêtes en attente, est incrémenté) ; la valeur de cette réponse sera quant à elle transmise au processeur de calcul uniquement après l'émission de la dernière requête. Il en est de même pour toutes les situations suivantes (avec une multiplication des transitions pour permettre la prise en compte des réponses de toutes les requêtes émises), jusqu'à l'envoi de la dernière requête. Après l'envoi de la dernière requête (situation COM(N+2), le traitement des réponses en attente doit avoir lieu. Ce traitement est représenté par la situation (COM(N+3)). Le modèle comporte une dernière situation (COM(N+4)) qui permet de prendre en compte les réponses qui ne sont pas encore arrivées dès qu'elles arrivent et de les traiter, en attendant le début du cycle suivant.

Pour une carte de communication communiquant avec N MES, la liste fifo est une liste d'entiers de longueur N-1 car il est impossible d'avoir la réponse à la N^{ime} requête juste après son émission. Lors de la prise en compte d'une réponse, l'entier rang est décrémenté (il y a une réponse de moins en attente) et les valeurs dans la liste fifo sont décalées : la réponse qui était en première position ayant été traitée, la réponse précédemment en deuxième position passe en première position, et ainsi de suite. Une fonction de décalage a donc été créée pour réaliser cette opération. Elle est présentée ci-dessous avec un paramètre l représentant la longueur de la file fifo.

```
void decale_l(int& a[l])
{
  for (i : int[0,l-2])
  {
    a[i] = a[i+1];
  }
  a[l-1]=0;
}
```

3.3.4 Modélisation du réseau

Comme cela a été présenté dans le chapitre 1, l'impact du réseau sur la transmission des données entre une carte de communication et un MES peut être représenté par un délai constant. De ce fait, une modélisation à l'aide de fonctions de communication (représentant chacune les échanges entre une carte de communication et un MES) a été choisie. Le modèle du réseau correspond donc à l'ensemble des modèles des fonctions de communication indépendantes (figure 3.11).

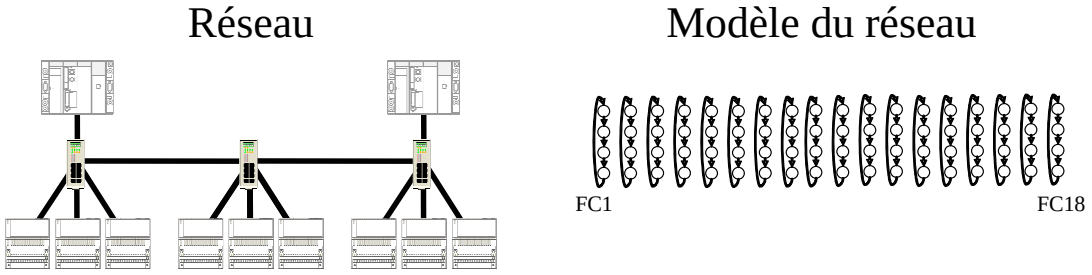


FIG. 3.11 – Modèle du réseau

Le modèle retenu pour représenter une fonction de communication générique est celui de la figure 3.12. Il est composé de quatre situations :

- FC1 : attente d'une requête provenant de la carte de communication
- FC2 : transfert de la requête provenant de la carte de communication vers le MES
- FC3 : attente d'une réponse provenant du MES
- FC4 : transfert de la réponse provenant du MES vers la carte de communication

Les canaux de communication représentent :

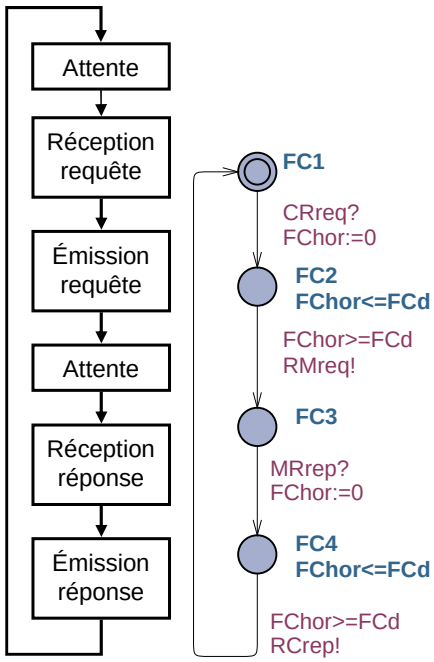
- CRreq : l'émission d'une requête de la carte de communication vers le réseau
- RMreq : l'émission d'une requête du réseau vers le MES
- MRrep : l'émission d'une réponse du MES vers le réseau
- RCrep : l'émission d'une réponse du réseau vers la carte de communication

Le modèle de fonction de communication ne modifie aucune donnée, il sert juste à synchroniser les évolutions des modèles de carte de communication et de MES.

Un MES doit recevoir la requête envoyée par la carte de communication au bout d'**une durée égale à FCd (retard dû au réseau)**. De la même façon, la réponse fournie par une MES doit être transmise à la carte de communication au bout d'une durée égale à FCd. Cette durée est définie par les invariants des situations FC2 et FC4 (ces situations doivent être quittées au plus tard quand la valeur de l'horloge FChor vaut FCd, FChor<= FCd) et les gardes associées aux transitions FC2 vers FC3 et FC4 vers FC1 (ces transitions ne peuvent être franchies que si la valeur de l'horloge est supérieure ou égale à FCd, FChor>=FCd).

3.3.5 Modélisation des modules d'entrées/sorties déportées

Le fonctionnement d'un module d'entrées/sorties est simple. Dès qu'il reçoit une requête, il copie ses entrées (action considérée à temps nul), encode ces valeurs dans une

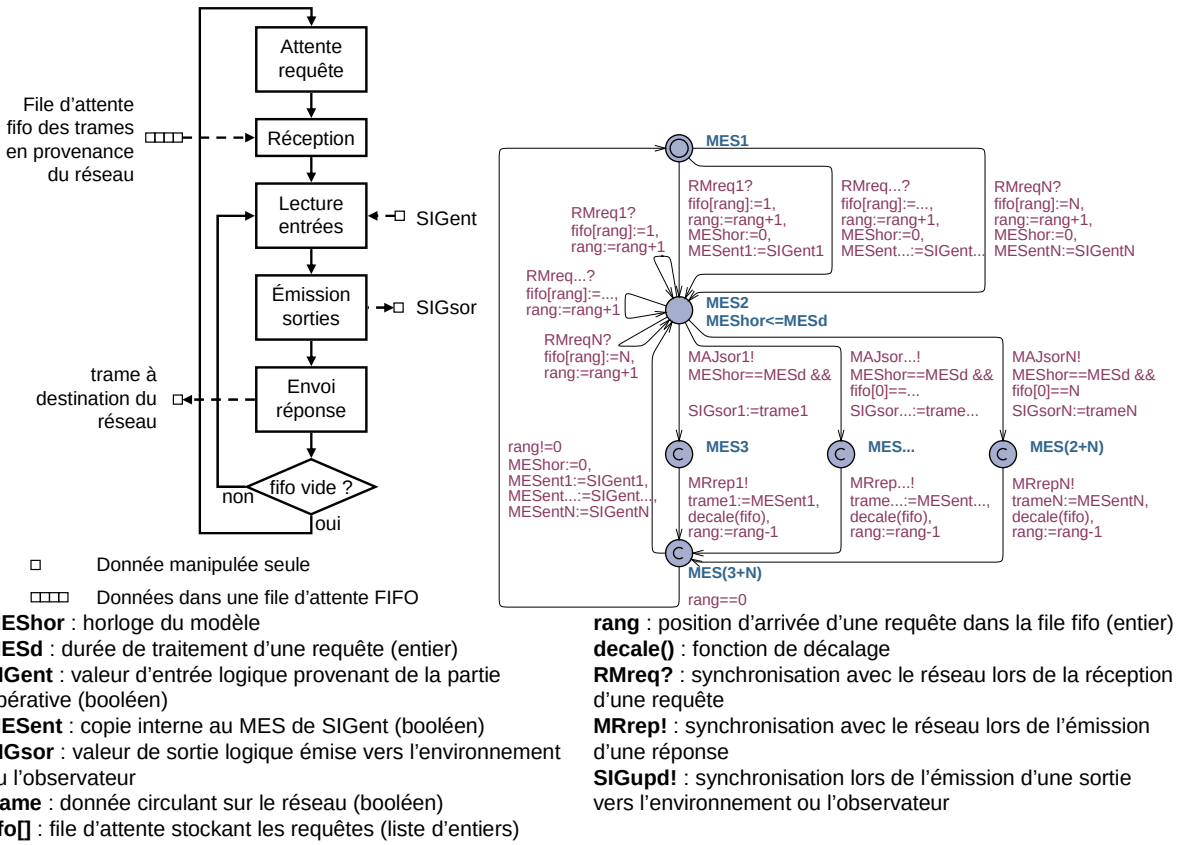


FChor : horloge du modèle
FCd : durée de transmission d'une requête
CRreq? : synchronisation avec la carte de communication lors de la réception d'une requête
RMreq! : synchronisation avec le MES lors de l'émission d'une requête
MRrep? : synchronisation avec le MES lors de la réception d'une réponse
RCrep! : synchronisation avec la carte de communication lors de l'émission d'une réponse

FIG. 3.12 – Modèle générique d'une fonction de communication

trame et envoie sa réponse (durée fixe). S'il reçoit une ou plusieurs requêtes avant la fin du traitement de la requête précédente, la requête en cours est traitée tandis que les suivantes sont mises en attente avant d'être traitées dans leur ordre d'arrivée (fonctionnement FIFO). Contrairement aux modèles des processeurs de calcul et des cartes de communication, le fonctionnement sur évènement permet de s'affranchir d'une étape d'initialisation. L'état normal lors de la mise en route de l'architecture est d'attendre une requête.

Une hypothèse a été faite pour réaliser le modèle des MES, c'est que **la durée de traitement de l'ensemble des N requêtes provenant des N cartes de communication communicant avec le MES est inférieure au plus court des cycles d'IOscanning de ces cartes de communication**. Cette hypothèse permet de simplifier le modèle en étant sûr qu'il y aura au plus N requêtes en attente de traitement (une pour chaque carte de communication). Cette hypothèse impose que $N * MESd \leq min(COMtc)$ avec N le nombre de cartes de communication en relation avec le MES, $MESd$ la durée du traitement d'une requête et $min(COMtc)$ le temps de cycle de l'IOscanning le plus court. Cela semble tout à fait réaliste pour les AAR étudiées (Modicon Modbus TCP/IP) pour lesquelles la durée d'IOscanning la plus courte est de 10 ms et la durée du traitement d'une requête est de 0.7 ms, ce qui permet quand même à 14 cartes de communication (valeur très supérieure à celle rencontrée pour les cas industriels) d'échanger avec le même MES tout en respectant cette contrainte.



MEShor : horloge du modèle
MESd : durée de traitement d'une requête (entier)
SIGent : valeur d'entrée logique provenant de la partie opérative (booléen)
MESent : copie interne au MES de SIGent (booléen)
SIGsor : valeur de sortie logique émise vers l'environnement ou l'observateur
trame : donnée circulant sur le réseau (booléen)
fifo[] : file d'attente stockant les requêtes (liste d'entiers)

rang : position d'arrivée d'une requête dans la file fifo (entier)
decale() : fonction de décalage
RMreq? : synchronisation avec le réseau lors de la réception d'une requête
MRrep! : synchronisation avec le réseau lors de l'émission d'une réponse
SIGupd! : synchronisation lors de l'émission d'une sortie vers l'environnement ou l'observateur

FIG. 3.13 – Comportement et modèle générique d'un module d'entrées/sorties communiquant avec N cartes de communication

La figure 3.13 présente le modèle générique de MES communiquant avec N cartes de communication. En faisant le parallèle entre le comportement d'un MES et le modèle proposé, nous retrouvons :

- MES1 : attente des requêtes,
- transitions entre MES1 et MES2 : réception d'une requête alors que le MES est en phase d'attente et lecture des entrées,
- MES2 : traitement d'une requête,
- transitions entre MES2 et MES2 : réception d'une requête i alors que le MES traite déjà une autre requête j,
- transitions entre MES2 et MES3 (respectivement MES4 ...MES(2+N)) : émission des sorties vers la partie opérative,
- transitions entre MES3 (respectivement MES4 ...MES(2+N)) et MES(3+N) : envoi de la réponse à la carte de communication,
- MES(3+N) : test pour savoir si la file d'attente FIFO est vide
- transition MES(3+N) vers MES1 : si la file d'attente FIFO est vide, retour en situation d'attente de requêtes,
- transition MES(3+N) vers MES2 : si la file d'attente FIFO n'est pas vide, lecture des entrées venant de la partie opérative.

Pour garder cette compacité et cette facilité de compréhension du modèle, comme pour le modèle de carte de communication, il a été nécessaire de définir une liste pour représenter la file fifo des requêtes en attente et une fonction pour décaler les requêtes dans cette file.

3.4 Modélisation de l'environnement et de l'automate observateur

3.4.1 Modélisation de l'environnement

Le modèle de l'environnement est une représentation très abstraite de la partie opérative sur laquelle agit l'AAR étudiée. Ce qui nous intéresse dans notre étude n'étant pas le comportement de cette partie opérative, il a été choisi d'en donner un modèle très abstrait qui comporte simplement un générateur d'évènement d'entrée et des consommateurs d'évènements de sortie.

Deux modèles d'environnement ont été développés : un pour l'étude du temps de réponse, un pour l'étude de la différence de temps de réponse. La différence essentielle entre ces deux modèles réside dans leurs évolutions initiales (transition entre ENV1 et ENV2 pour le premier figure 3.14, transitions entre ENV1 et ENV4 via ENV2 ou ENV3 pour le second figure 3.15). Cette différence provient du fait que :

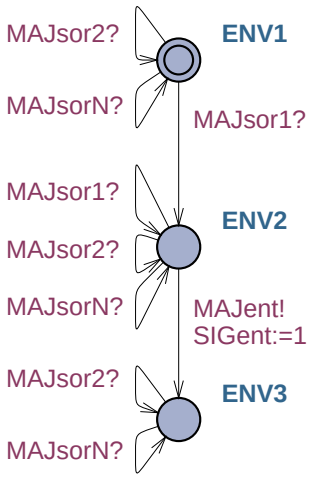
- pour l'étude du temps de réponse, il faut attendre l'initialisation d'un seul API (processeur de calcul et carte de communication) et d'un seul MES (qui génère l'évènement de sortie intervenant dans la définition du temps de réponse) pour générer l'évènement d'entrée (c.f. partie 3.3.1) ;
- pour l'étude de la différence de temps de réponse, il faut attendre l'initialisation de deux API et de deux MES car deux évènements de sortie, que nous supposerons générés par deux API différents, sont à considérer.

Ces deux modèles sont détaillés dans ce qui suit.

3.4.1.1 Pour l'étude d'un temps de réponse

La figure 3.14 représente le modèle d'environnement proposé dans le cas d'une AAR à N sorties (toutes les transitions de type "boucle" ne sont pas représentées par souci de lisibilité).

La situation ENV1 correspond à l'attente de l'initialisation de l'API et du MES générant l'évènement de sortie. Une fois la première mise à jour de la sortie observée effectuée (MAJsor1 ?), le modèle d'environnement peut générer l'évènement d'entrée à n'importe quel moment (transition ENV2 vers ENV3). Dans la situation ENV2, les mises à jour de toutes les sorties doivent pouvoir être prises en compte. Une fois l'évènement d'entrée généré (situation ENV3), l'environnement n'a plus qu'à accepter les mises à jour de toutes les sorties, sauf celle observée ; cette sortie utilisée pour l'évaluation du temps de réponse doit en effet n'être vue que par l'automate observateur.



SIGent : valeur du signal d'entrée (booléen)
MAJent! : synchronisation avec l'observateur lors de l'émission de l'évènement d'entrée
MAJsor*i*? : synchronisation avec un MES lors de l'émission de la sortie *i*

FIG. 3.14 – Modèle de l'environnement pour l'étude de temps de réponse

Dans toutes les situations, les synchronisations avec les sorties qui ne sont pas observées (MAJsor2 ? à MAJsorN ?) doivent être acceptées pour ne pas bloquer l'évolution du modèle de l'AAR.

Le modèle ne comporte pas d'horloge puisqu'il n'est soumis à aucune contrainte de temps physique et affecte une seule variable (SIGent, la valeur du signal d'entrée) mais reçoit de nombreux canaux de communication : les signaux de mise à jour des sorties des MES MAJsor1 à MAJsorN, N correspondant au nombre de MES multiplié par le nombre de sorties par MES.

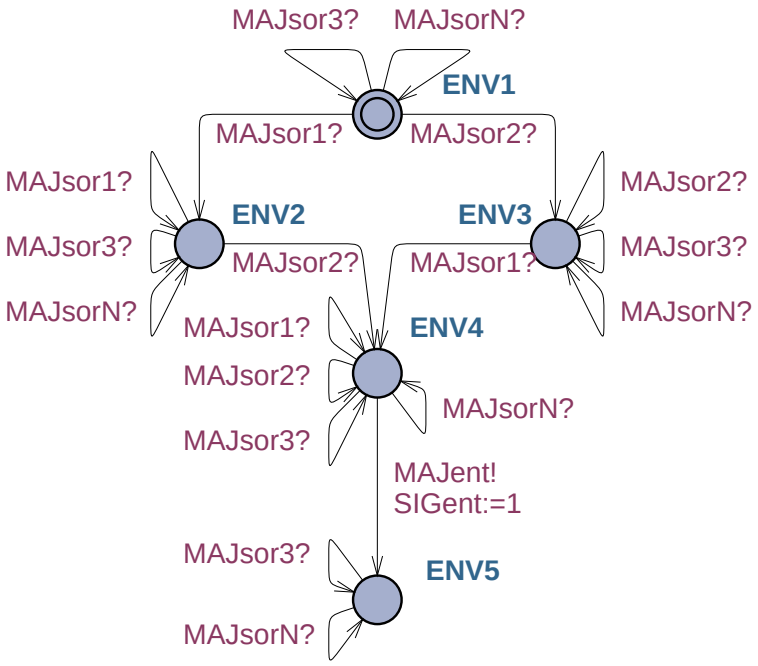
3.4.1.2 Pour l'étude d'une différence de temps de réponse

Le modèle (figure 3.15), représente un environnement pour l'étude d'une différence de temps de réponse dans le cas d'une AAR à N sorties.

Les situations ENV1 à ENV3 correspondent à la phase d'initialisation des deux API qui reçoivent la valeur de l'entrée et calculent les valeurs des deux sorties utilisées ainsi que des deux MES qui appliquent ces sorties à la partie opérative. La fin de l'initialisation correspond à la situation ENV4, situation qui ne peut être atteinte que si les deux synchronisations MAJsor1 ? et MAJsor2 ? ont eu lieu au moins une fois. L'environnement peut alors générer l'évènement d'entrée à n'importe quel moment (transition ENV4 vers ENV5). Une fois que l'évènement d'entrée a été généré (situation ENV5), l'environnement n'a plus qu'à accepter les mises à jour de toutes les sorties sauf celles qui interviennent dans la définition de la différence de temps de réponse.

Dans toutes les situations, les synchronisations avec les sorties qui ne sont pas observées (MAJsor3 ? à MAJsorN ?) doivent être acceptées pour ne pas bloquer l'évolution du modèle de l'AAR.

Comme le modèle précédent, ce modèle ne comporte pas d'horloge, affecte une seule



SIGent : valeur du signal d'entrée (booléen)
MAJent! : synchronisation avec l'observateur lors de l'émission de l'évènement d'entrée
MAJsor*i*? : synchronisation avec un MES lors de l'émission de la sortie *i*

FIG. 3.15 – Modèle de l'environnement pour l'étude de la différence de temps de réponse

variable (SIGent, la valeur du signal d'entrée) et reçoit N signaux de mise à jour des sorties (MAJsor1 à MAJsorN), avec N correspondant au nombre de MES multiplié par le nombre de sorties par MES.

3.4.2 Modélisation de l'automate observateur

Le principe général de fonctionnement des automates observateurs a été présenté dans la partie 2.2 ; nous allons à présent détailler les modèles développés dans le cadre de ces travaux.

3.4.2.1 Pour l'étude d'un temps de réponse

Nous retrouvons la situation OBS1 pour modéliser l'attente de l'évènement d'entrée dont la réception (modélisée par la synchronisation MAJent ?) fait franchir la transition entre OBS1 et OBS2. Lors de ce franchissement, l'horloge de l'automate observateur (OBShor) est réinitialisée. Elle ne sera plus affectée dans le modèle, sa valeur sera simplement comparée à la valeur du paramètre τ lors de la mise à 1 de la sortie observée.

Les situations OBS2 et OBS3 modélisent l'attente de l'évènement de sortie dont l'occurrence amène dans une des situations finales (OBS4, OBS5 et OBS6) suivant la valeur de l'horloge lors de la mise à 1 de la sortie observée. Il est bon de noter qu'il n'est pas possible de tester la valeur du signal de sortie SIGsor lors du franchissement de la transition com-

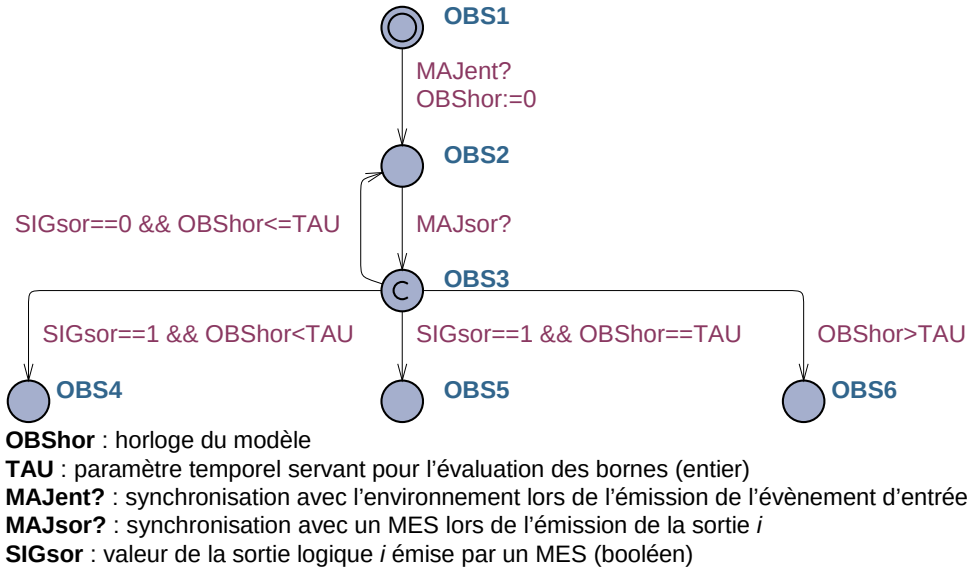


FIG. 3.16 – Modèle de l'automate observateur utilisé pour l'étude de temps de réponse

portant l'étiquette de synchronisation MAJsor ? car le modèle du module d'entrées/sorties déportées met à jour la valeur de SIGsor en même temps qu'il émet le message de synchronisation MAJsor !. UPPAAL vérifiant premièrement les gardes puis ensuite effectuant les affectations, le test de la valeur de SIGsor par l'automate observateur se ferait alors avec l'ancienne valeur de SIGsor et non la nouvelle. Ceci explique la modélisation proposée, qui utilise une situation committed supplémentaire et non trois transitions issues de OBS2 et arrivant dans les trois situations terminales.

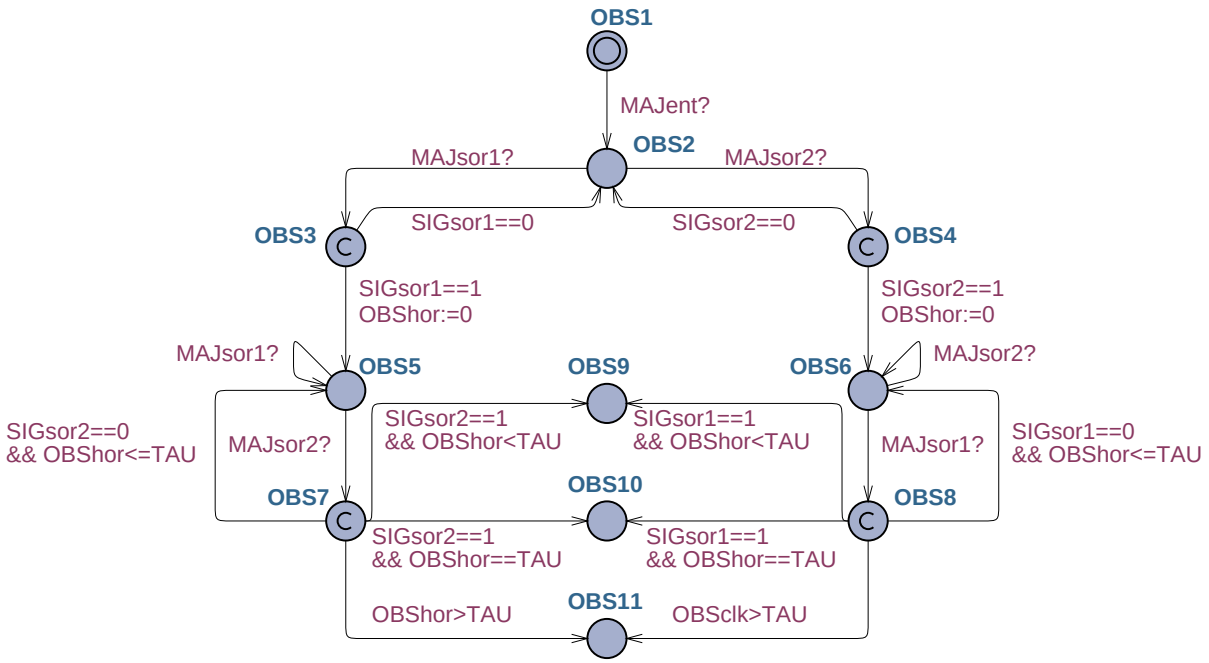
La transition de la situation OBS3 vers OBS2 a une garde qui porte en partie sur l'horloge OBShor. Cette contrainte (OBShor $\leq \tau$) permet d'éviter l'attente du prochain évènement de sortie (MAJsor ?) si le délai τ est déjà atteint. Dans ce cas l'observateur évolue vers OBS5 même si la sortie n'a pas encore été mise à 1.

3.4.2.2 Pour l'étude d'une différence de temps de réponse

Pour l'étude d'une différence de temps de réponse, l'automate observateur utilisé est un peu plus compliqué parce qu'il est nécessaire de surveiller les évolutions de deux sorties différentes.

Le franchissement de la transition entre OBS1 et OBS2 par synchronisation MAJent ? déclenche le début de l'observation des deux signaux de sortie MAJsor1 ? et MAJsor2 ? mais ne réinitialise pas l'horloge du modèle (OBShor) car la performance étudiée n'est pas basée sur la date d'apparition de l'évènement d'entrée mais sur la date d'apparition du premier évènement de sortie.

Dès que la sortie du premier (resp. deuxième) module d'entrées/sorties est émise (signalée par la synchronisation MAJsor1 ?, resp. MAJsor2 ?), la situation OBS3 (resp. OBS4) est atteinte, situation transitoire pour soit retourner dans la situation OBS2 si la valeur de la sortie émise est 0, soit pour évoluer vers la situation OBS5 (resp. OBS6) si elle est



OBShor : horloge du modèle
TAU : paramètre temporel servant pour l'évaluation des bornes (entier)
MAJent? : synchronisation avec l'environnement lors de l'émission de l'évènement d'entrée
MAJsor*i*? : synchronisation avec un MES lors de l'émission de la sortie *i*
SIGsor*i* : valeur de la sortie logique *i* émise par un MES (booléen)

FIG. 3.17 – Modèle de l'automate observateur utilisé pour l'étude d'une différence de temps de réponse

égale à 1. L'horloge de l'automate observateur est réinitialisée lors du franchissement de la transition menant à OBS5 ou à OBS6. Elle ne sera plus affectée dans le modèle, sa valeur sera simplement comparée à la valeur du paramètre τ lors de l'observation de la mise à 1 de la deuxième sortie.

Les situations OBS5 et OBS7 (resp. OBS6 et OBS8) modélisent l'attente de la mise à 1 de la deuxième sortie et le test de la valeur de l'horloge de l'automate observateur lors de cette mise à 1. Une des situations finales (OBS9, OBS10 et OBS11) sera atteinte en fonction de la valeur de l'horloge par rapport au paramètre τ lors de la mise à 1 de la deuxième sortie.

Les transitions des situation OBS7 vers OBS5 et OBS8 vers OBS6 ont des gardes qui portent en partie sur l'horloge OBShor. Cette contrainte (OBShor $\leq \tau$) permet d'éviter l'attente des prochains évènements de sortie (MAJsor1 ? ou MAJsor2 ?) si le délai τ est déjà atteint. Dans ce cas l'observateur évolue vers OBS11 même si la deuxième sortie émise n'a pas encore été mise à 1.

3.5 Évaluation de la capacité d'Uppaal à traiter des modèles de taille non triviale

Pour évaluer les possibilités de passage à l'échelle de la méthode proposée au chapitre 2, et en particulier les capacités du model-checker retenu à traiter des modèles d'AAR construits à partir des modèles génériques proposés au cours de ce chapitre, une architecture d'automatisation de taille intermédiaire, qui ne soit ni un cas d'étude trivial, ni une application de taille industrielle, a été étudiée. Elle est représentée par la figure 3.18. Elle se compose de trois API, deux utilisés pour du contrôle commande et un pour de la supervision. Ces trois API communiquent avec neuf modules d'entrées/sorties au travers d'un réseau composé de trois commutateurs.

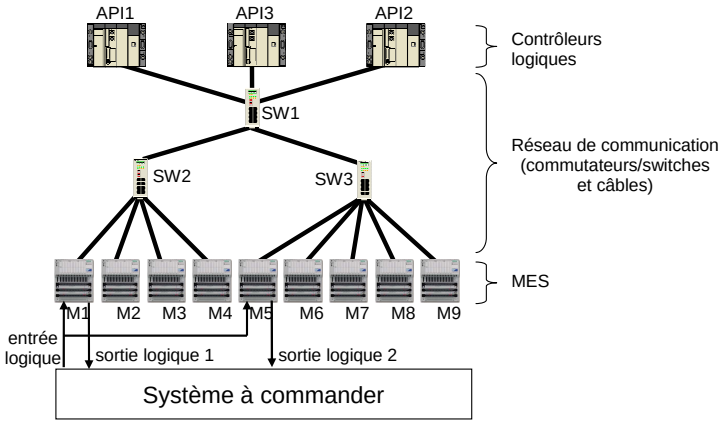


Fig. 3.18 – Cas d'étude de taille représentative

La configuration de ces différents composants de l'architecture d'automatisation est donnée dans le tableau 3.1.

Tab. 3.1 – Valeurs types pour les paramètres des API

	API1	API2	API3
durée d'exécution du programme	2 à 3 ms	3 à 4 ms	5 à 6 ms
durée du cycle d'I/O scanning	10 ms	10 ms	50 ms
MES scrutés	M1 à M4	M5 à M9	M1 à M9

Le modèle de cette AAR comporte ainsi trois instances du modèle générique de processeur de calcul, trois modèles de cartes de communication (un modèle communiquant avec quatre MES, un modèle communiquant avec cinq MES et un modèle communiquant avec neuf MES), dix-huit instances du modèle générique de fonction de communication, neuf instances du modèle générique de MES. A ce modèle d'AAR, il convient d'ajouter un modèle d'environnement générant l'évènement d'entrée et un automate observateur pour l'étude de la différence de temps de réponse.

Différentes études ont pu être menées sur ce cas, notamment en changeant la représentation de l'espace d'états utilisée par Uppaal (DBM, compact data structure ou

approximations)[10]. Malheureusement, toutes ces études ont abouti au même résultat : la résolution est impossible faute de mémoire, malgré l'utilisation d'un ordinateur muni de 4 Go de RAM. La mémoire vive de l'ordinateur est saturée quasiment instantanément (quelques secondes) quelle que soit la représentation utilisée pour l'espace d'états./newline

Ces études nous montrent que malgré toutes les hypothèses simplificatrices utilisées lors de la conception des modèles génériques de composants, les modèles d'architectures d'automatisation déduits de ces modèles génériques restent trop complexes pour pouvoir être traités par le model-checker utilisé. Pour résoudre ce problème, nous avons dû développer plusieurs abstractions, visant à améliorer le passage à l'échelle. Ces abstractions sont décrites dans le chapitre suivant.

[10] Ces différentes représentations seront explicitées dans le chapitre 5

Chapitre 4

Abstraction des modèles d'architectures d'automatisation en réseau

Ce chapitre est consacré à l'abstraction des modèles d'architectures d'automatisation en réseau afin de réduire les effets de l'explosion combinatoire liée aux techniques de model-checking. Une modification de certains modèles génériques de composants est tout d'abord présentée mais, devant l'insuffisance de cette tentative pour réussir un passage à l'échelle, une méthode d'abstraction plus globale du modèle de l'AAR sera développée. Cette méthode s'inspire de méthodes existantes telles que les cônes d'influence ou les points de coupe. Elle se compose de deux étapes. Nous verrons ainsi premièrement comment il est possible de simplifier la structure des modèles d'architectures d'automatisation en réseau en fonction de la performance temporelle étudiée puis pourquoi il est nécessaire de modifier les modèles de composants pour garder une équivalence, du point de vue de la performance temporelle étudiée, entre le modèle initial et le modèle abstrait.

Sommaire

4.1 Première tentative portant sur certains modèles génériques

Afin de lever le verrou indiqué au chapitre précédent : incapacité de l'outil de vérification Uppaal à traiter des modèles d'AAR de taille non triviale, nous avons, dans un premier temps, tenté de **modifier trois modèles génériques** : MES, environnement et automate observateur, qui nous paraissaient coûteux en temps d'analyse. Ces propositions, ainsi que leurs effets sur le temps de vérification, sont décrits dans cette partie.

4.1.1 Modifications du modèle de MES

La liste fifo (une liste d'entiers pour Uppaal) utilisée par le modèle de MES, ainsi que la fonction de décalage qui lui est associée, semblent impacter fortement le temps de vérification. Ce constat nous a conduit à investiguer deux possibilités de modification :

- la représentation de l'ordre d'arrivée des requêtes au moyen d'un entier seulement,
- la prise en compte de cet ordre d'arrivée au travers de la structure du modèle.

Trois modèles différents de MES communiquant avec trois cartes de communications ont été comparés, le premier utilise la liste fifo (modèle présenté dans la partie 3.3.5), le deuxième en est dérivé et utilise le codage de l'ordre d'arrivée des requêtes sur un entier et enfin le dernier est un modèle dont la structure gère l'ordre d'arrivée des requêtes. Ces modèles sont représentés respectivement sur les figures 3.13, 4.1 et 4.2.

Représentation de l'ordre d'arrivée des requêtes au moyen d'un entier

Cette modification consiste en la suppression de la liste fifo en tant que telle. Celle-ci est remplacé par un entier *ordre_requetes* (et non plus une liste d'entiers) pour représenter l'ordre d'arrivée des requêtes sur le MES.

Soit :

- n le nombre de cartes de communication communiquant avec le MES,
- i le numéro de la carte qui a émis une requête.

L'entier *ordre_requetes* est initialisé à zéro puis est calculé ainsi, lors de l'arrivée de la requête en provenance de la carte i : $ordre\_requetes = (ordre\_requetes \times (n+1)) + i$. En parallèle, un entier *nbr_att* contient le nombre de requêtes en attente. Il est donc incrémenté lors de la réception d'une requête.

Ainsi, si un MES communiquant vec 9 cartes de communications reçoit une requête de la carte numéro 2, *ordre_requetes* devient égal à 2 et *nbr_att* devient égal à 1. Si avant la fin du traitement de cette requête, deux autres requêtes en provenance des cartes 1 et 3 arrivent (dans cet ordre), *ordre_requetes* passe tout d'abord à 21 et *nbr_att* devient égal à 2, puis *ordre_requetes* passe à 213 et *nbr_att* devient égal à 3. Cela correspond à une représentation décimale d'un codage en base $n+1$.

Lors du traitement d'une requête, la valeur de *ordre_requetes* est testée de manière à savoir quelle est la première requête à traiter. Après le traitement de cette requête,

ordre_requetes et *nbr_att* seront modifiés pour que la requête ne soit plus considérée en attente de traitement.

Ce traitement peut se décrire ainsi :

si $\exists i,\ 0 < i \leq n$ tel que $ordre\_requetes = i \vee i * (n+1) \leq ordre\_requetes < (i+1) * (n+1) \vee \cdots \vee i * (n+1)^{n-1} < ordre\_requetes \leq (i+1) * (n+1)^{n-1}$

alors émission de la requête i

$ordre\_requetes = ordre\_requetes - i * (n+1)^{nbr\_att-1}$

$nbr\_att = nbr\_att - 1$

Cette représentation est plus compliquée à mettre en oeuvre que de créer une liste, y associer une fonction de décalage (fonction valable pour toutes les listes) et ainsi pouvoir utiliser toujours les mêmes transitions. De plus, cette représentation est limitée par la nature du nombre ordre_requetes (un entier) dont Uppaal limite la valeur à 32767, ce qui ne permet de représenter que 5 requêtes en attente.

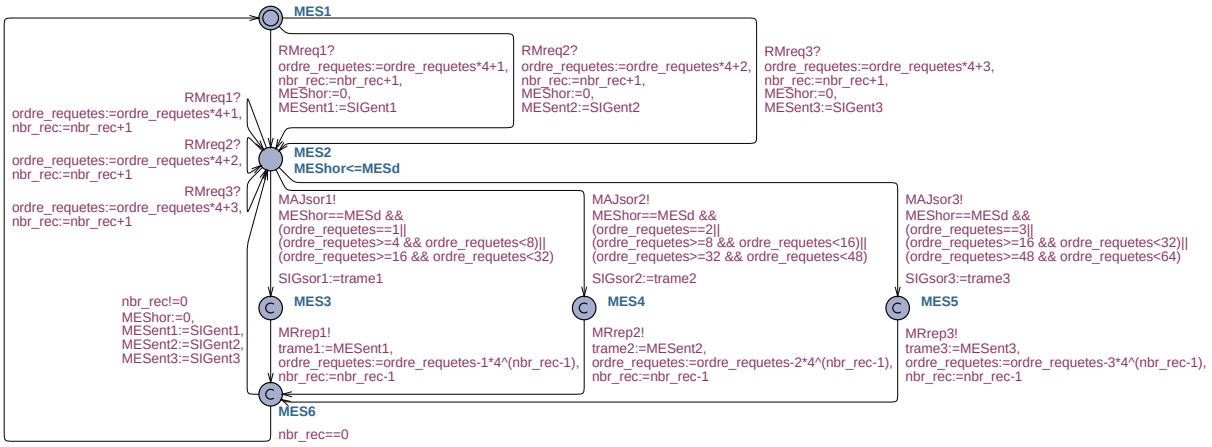


Fig. 4.1 – Modèle d'un MES communiquant avec 3 API, avec prise en compte de l'ordre d'arrivée des requêtes par un entier

Cette limitation sur la longueur de la file d'attente avec notre choix de représentation nous nous a pas permis de réaliser cette même modification sur le modèle de la carte de communication. En effet qu'une carte de communication communique avec plus de 5 MES est une situation très courante dans les AAR alors que le partage d'un même MES par plus de 5 cartes de communication est plus qu'extrêmement rare.

Représentation de l'ordre d'arrivée des requêtes au travers de la structure du modèle

Le modèle de la figure 4.2 montre comment prendre en compte par construction l'ensemble des possibilités d'arrivées des requêtes sur le MES. Avec trois requêtes provenant de trois API différents, il y a au plus trois requêtes en attente de traitement. En effet, le fonctionnement du cycle d'IOscanning des cartes de communication empêche celles-ci d'envoyer une nouvelle requête si la précédente n'a pas encore eu de réponse (cf partie 1.2.2.1).

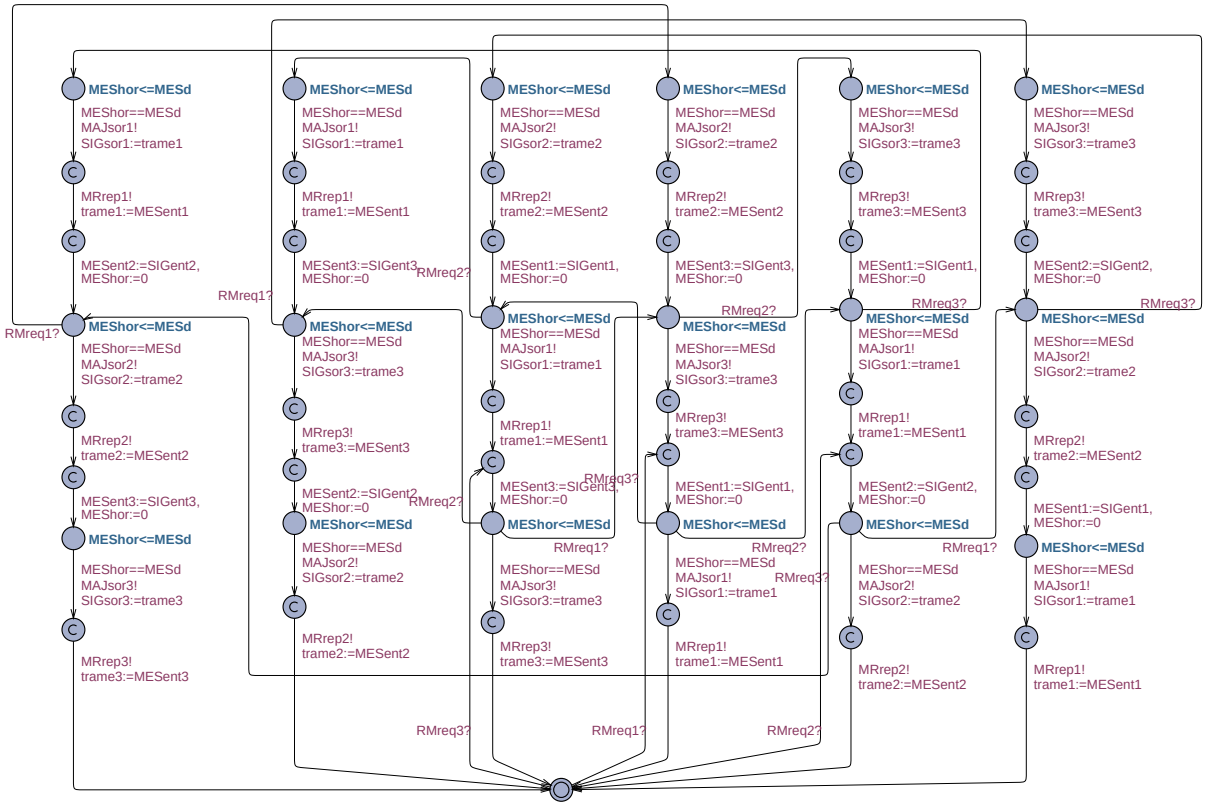


FIG. 4.2 – Modèle d'un MES communiquant avec trois API, avec prise en compte de l'ordre d'arrivée des requêtes dans la structure

Il y a donc trois possibilités d'avoir une requête en cours de traitement et aucune en attente, six (trois possibilités pour la première requête et deux pour la deuxième) possibilités d'avoir une requête en cours de traitement et une en attente mais également six possibilités d'avoir une requête en cours de traitement et deux en attente (trois possibilités pour la première requête, deux pour la deuxième et une pour la troisième). Le modèle de la figure 4.2 décrit ces cas de figure. Le passage à un modèle de MES communiquant avec quatre API différents n'est pas compliqué mais rend le modèle de beaucoup plus grande taille vu qu'il y a dans ce cas 24 possibilités d'avoir une succession de quatre requêtes différentes et donc 24 branches différentes en parallèle. Cette solution pourrait être envisagée en utilisant un générateur automatique de modèles UPPAAL, outil qui n'a pas été développé dans ces travaux.

Comparaison des trois modélisations

La comparaison a été effectuée avec la configuration d'essai suivante : trois cartes de communication communiquant chacune avec trois MES, un environnement et un automate observateur pour déterminer les bornes d'un temps de réponse. Il convient de noter que, pour simplifier le modèle (gain de temps de calcul) et pour éviter que l'influence du modèle de MES utilisé soit réduite par les autres modèles, il n'y a dans cette configuration

ni processeur de calcul (les cartes de communication émettent directement en sortie les valeurs qu'elles ont en entrée), ni réseau (les cartes de communication communiquent "directement" avec les MES).

Les trois essais ont bien sûr été réalisés dans les mêmes conditions (même ordinateur et même choix dans les options de vérification proposées par UPPAAL). Les résultats obtenus sont les suivants (tableau 4.1) :

TAB. 4.1 – Comparaison des modèles de MES

Modèle	Figure 3.13	Figure 4.1	Figure 4.2
Prise en compte de l'ordre d'arrivée des requêtes	liste	entier	stucture
Temps de vérification	1 h 5 min	31 min	35 min

Il apparaît clairement que le modèle avec l'ordre d'arrivée des requêtes contenu dans un entier est le plus efficace en terme de temps de calcul (gain de 52% par rapport au modèle initial dans notre cas d'étude). Il est cependant limité par la taille d'un entier par UPPAAL. La prise en compte de l'ordre d'arrivée des requêtes directement par la structure du modèle est elle aussi beaucoup plus efficace que la représentation avec la file fifo mais reste un peu moins intéressante que la représentation de l'ordre d'arrivée des requêtes dans un entier. Cette représentation sera donc utilisée autant que possible.

4.1.2 Fusion du modèle d'environnement et de l'automate observateur

Une autre voie d'abstraction est envisageable en fusionnant le modèle de l'environnement et l'automate observateur. En effet l'environnement évolue jusqu'à sa synchronisation avec l'automate observateur, puis ne change plus de situation alors que pour l'observateur cette synchronisation correspond à se première évolution. La fusion de ces deux automates, en vue de réduire le temps de vérification, supprime donc une synchronisation.

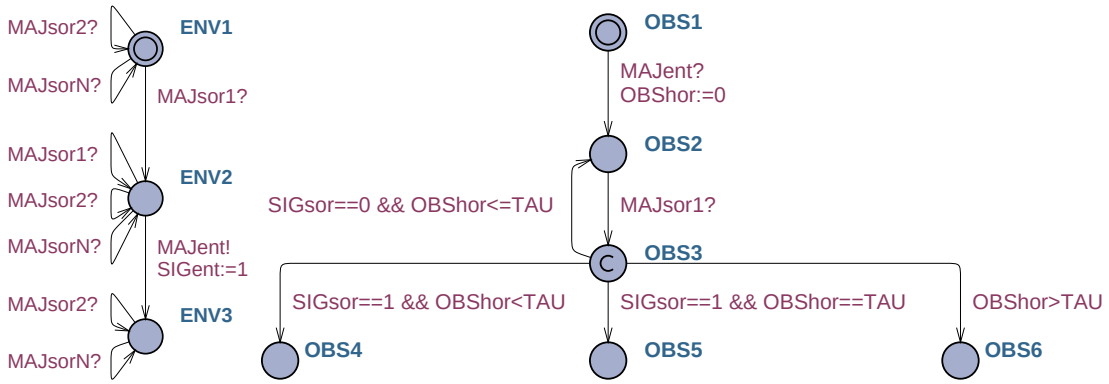


FIG. 4.3 – Modèles de l'environnement et de l'observateur avant leur fusion

Dans l'automate fusionné, le nom d'une situation est la concaténation des noms des situations des deux automates indépendants. Dans les situations ENV3OBS4 à ENV3OBS6,

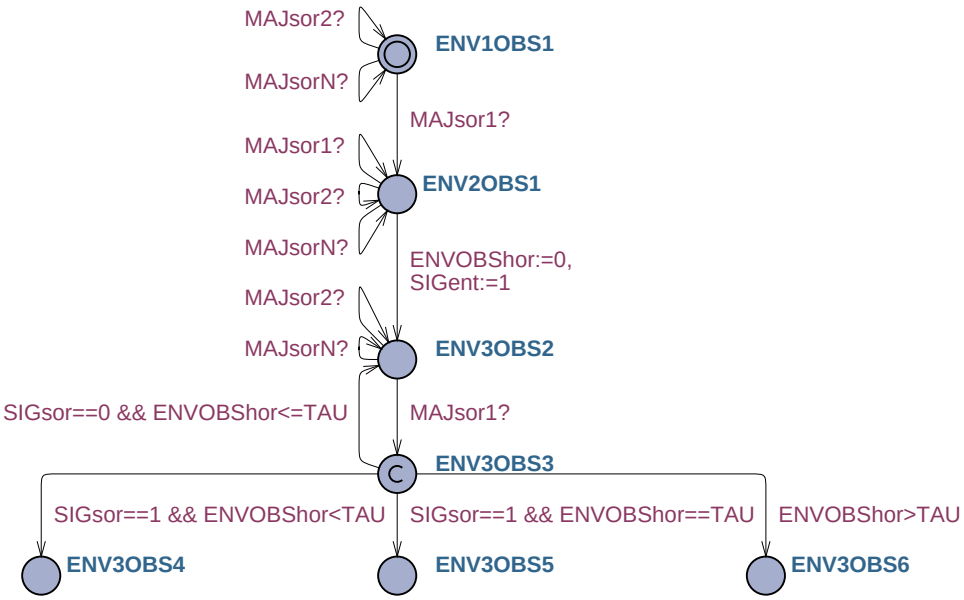


FIG. 4.4 – Modèle fusionné de l'environnement et de l'observateur

les synchronisations pour la prise en compte des nouvelles sorties (MAJsor2 à MAJsorN) ont été supprimées car les évolutions du système après qu'une de ces trois situations ait été atteintes ne sont pas utiles pour l'obtention des bornes de la performance temporelle étudiée.

Le gain en performance entre ces deux solutions a été quantifié par une configuration de test comportant deux cartes de communication communiquant chacune avec neuf MES, un environnement et un automate observateur pour déterminer les bornes d'un temps de réponse. Comme pour la configuration de test des modèles de MES, il n'y a ni processeur de calcul (les carte de communications émettent directement en sortie les valeurs qu'elles ont en entrée) ni réseau (les cartes de communication communiquent "directement" avec les MES).

Les temps de calculs observés sont contenus dans le tableau 4.2.

TAB. 4.2 – Comparaison des modèles de MES

Modèle	ENV et OBS séparés	ENV et OBS fusionnés
Temps de vérification	2 min 43 s	2 min 57 s

La fusion des deux modèles de l'environnement et de l'observateur n'est pas efficace puisque le temps de vérification est sensiblement le même que sans la fusion (légèrement supérieure).

4.1.3 Simplification de l'automate observateur

Cette simplification consiste à ne plus effectuer le crible des valeurs de l'horloge par rapport à τ dans le modèle mais à l'indiquer dans les propriétés à analyser. Elle conduit

donc à modifier les propriétés d'atteignabilité.

Dans le cas de l'étude de temps de réponse, le modèle de l'observateur passe donc de celui de la figure 4.3 à celui de la figure 4.5. L'état OBS3 est un état bloquant dès que la sortie SIGsor vaut 1, ce qui signifie que dès que la valeur de sortie observée est passée à 1 (la réponse à l'évènement d'entrée est arrivée) le système ne peut plus du tout évoluer. Il est également bloquant quand la valeur de l'horloge est strictement supérieure à τ lors de la mise à jour de la sortie puisqu'il ne sert plus à rien de laisser évoluer le système, la valeur de τ ayant déjà été dépassée.

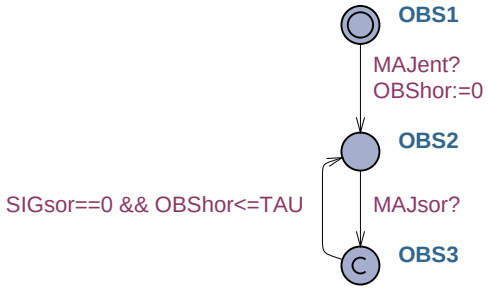


FIG. 4.5 – Modèle simplifié de l'automate observateur

Dans le même temps, les propriétés qui étaient :

$$P1 : E <> OBS.3 \tag{4.1}$$

$$P2 : E <> OBS.4 \tag{4.2}$$

$$P3 : E <> OBS.5 \tag{4.3}$$

deviennent maintenant :

$$P4 : E <> OBS.3 \wedge SIGsor == 1 \wedge horloge < \tau \tag{4.4}$$

$$P5 : E <> OBS.3 \wedge SIGsor == 1 \wedge horloge = \tau \tag{4.5}$$

$$P6 : E <> OBS.3 \wedge SIGsor == 1 \wedge horloge > \tau \tag{4.6}$$

L'influence de cette modification sur le temps de vérification a été quantifiée sur la même configuration que celle utilisée pour la fusion de l'environnement et de l'observateur. Les valeurs obtenues avec :

- ENV et OBS original séparés
- ENV et OBS simplifié séparés
- ENV et OBS simplifié fusionnés

sont contenues dans le tableau 4.3.

La comparaison entre la première colonne et les deux suivantes montre que cette simplification de l'automate observateur est très intéressante (gain de 67%), et ceci malgré le changement de propriétés à analyser. Les temps de vérification avec le modèle simplifié de l'observateur fusionné ou non avec l'environnement sont très proche. Comme c'est le modèle fusionné qui est le plus efficace, il sera donc utilisé pour la suite des travaux présentés.

Tab. 4.3 – Comparaison des modèles de MES

Modèle	ENV et OBS original séparé	ENV et OBS simplifié séparés	ENV et OBS simplifié fusionnés
Temps de vérification	2 min 43 s	53 s	44 s

Ces différentes simplifications proposées, si elles ne sont pas toutes efficaces, ont permis l'accroissement du savoir-faire sur les objets modélisés. Deux d'entres elles (la représentation de l'ordre d'arrivée des requêtes dans un entier et l'automate observateur simplifié avec les nouvelles propriétés correspondantes) continueront à être utilisées par la suite. Ces simplifications apportées aux différents modèles de composants ne changent malheureusement pas fondamentalement le problème de l'explosion combinatoire lié aux techniques de model-checking temporisé. Il est donc nécessaire de développer une méthode d'abstraction plus globale des modèles d'architectures d'automatisation en réseau afin de pouvoir enfin arriver à obtenir des résultats sur des cas d'études non triviaux. Cette méthode fait l'objet de la partie suivante.

4.2 Proposition d'une méthode globale d'abstraction de modèles d'AAR

4.2.1 Description générale

Lors de l'étude des composants constituant l'AAR, il apparaît que peu d'entre eux agissent directement sur les évènements observés lors de l'étude d'une performance temporelle. La première idée de cette méthode d'abstraction (figure 4.6) est de supprimer tous les composants "inutiles" pour la performance temporelle étudiée. Cette suppression, explicitée dans la partie 4.3, n'est pas forcément anodine. En effet, vu que le modèle de l'AAR est un ensemble d'automates communicants, la suppression de certains d'entre eux impose de supprimer toutes les anciennes communications/synchronisations entre les modèles de composants restants et ceux qui ont été supprimés. De plus, même si les composants supprimés n'agissaient pas directement sur les évènements observés, ils pouvaient avoir une action indirecte sur la performance temporelle étudiée simplement en chargeant (et donc ralentissant) les composants qui agissaient réellement sur les évènements observés.

La méthode globale d'abstraction proposée (figure 4.6) ([RdSF08b] et [RdSF08a]) utilise le modèle de l'AAR initial, dont l'obtention a été explicitée au chapitre 3, et se décompose en deux étapes :

- **Simplification de la structure** du modèle de l'AAR initial, en recherchant un chemin de coût minimum dans cette structure,
- **Modification des modèles de composants** restants (sur ce chemin de coût minimum), de façon à conserver un comportement équivalent.

Ces deux étapes de la méthode d'abstraction s'appuient sur des travaux antérieurs de la communauté qui sont détaillés dans la partie suivante.

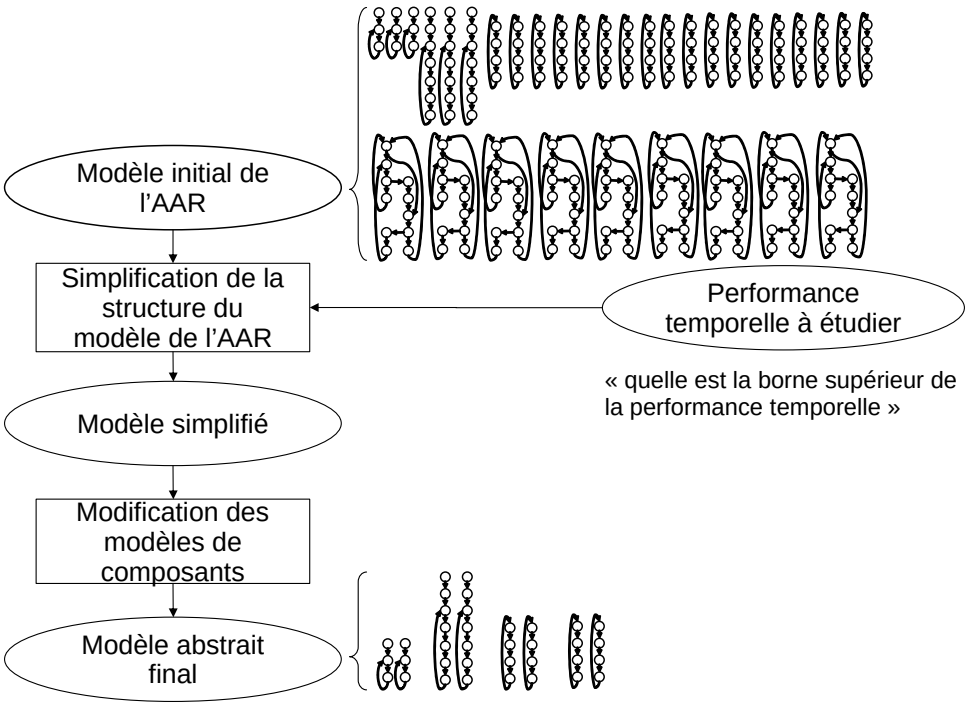


FIG. 4.6 – Méthode d'abstraction proposée

4.2.2 Revue bibliographique de résultats antérieurs

4.2.2.1 Techniques visant à simplifier la structure d'un modèle formel

La **réduction par cône d'influence**, Cone of Influence Reduction (COI) en anglais [CGP99], est une technique d'abstraction fondamentale pour réduire la taille des modèles utilisés en model-checking. De manière générale, cette technique est utilisée dans le domaine du logiciel (vérification de code C, de Ladder Diagram, ...) où elle est connue sous le nom de "slicing" [Hol04] comme dans le domaine du matériel où elle est plutôt appelée "localization reduction". C'est la technique d'abstraction la plus usitée dans le domaine de la vérification de circuits électroniques constitués de nombreux composants ([Kur94], [WJK+01], [CCK+02]).

Le modèle abstrait est créé à partir du circuit à étudier en supprimant un nombre important de portes dans l'optique de ne garder que celles utiles pour calculer le prochain état. Ceci est rendu possible par l'utilisation d'un graphe de dépendance qui permet d'éliminer tous les composants du circuit qui n'apparaissent pas dans ce graphe. Cette technique est une sur-approximation conservative du circuit original pour des propriétés d'atteignabilité. Cela signifie, que si le modèle abstrait satisfait une propriété, la propriété est aussi vérifiée dans le modèle original. Cependant, il est possible que le modèle abstrait amène à un contre-exemple qui ne corresponde à aucun comportement réel du circuit. Pour éviter ce désagrément, des techniques de raffinement de modèle ont été développées pour créer un nouveau modèle abstrait, plus riche que le précédent pour se prémunir de ces contre-exemples non réalistes. Une de ces techniques, basée sur l'utilisation du

contre-exemple pour raffiner le modèle est la méthode CEGAR (CounterExample Guided Abstraction Refinement) ([Kur94], [BR00], [CFH+03], [DD01], [WJK+01]).

La difficulté pour appliquer cette méthode sur des modèles d'AAR est essentiellement de savoir quels sont les composants de l'architecture (et non plus les portes du circuit) qu'il est nécessaire de conserver, sachant qu'ils introduisent tous des délais dans le modèle de l'architecture. De ce fait, ils sont tous dans le graphe de dépendance représentant les composants influant sur la performance étudiée et la méthode de réduction par cône d'influence ne peut pas s'appliquer directement.

La solution retenue pour cela est de ne garder qu'un chemin de coût minimum dans le graphe de dépendance. Pour trouver ce chemin, la notion de **classes d'influence des composants** proposée par [SF07] a été utilisée. Cette étude, appliquée à de la simulation d'un réseau AFDX (Avionics Full Duplex Switched Ethernet), a permis de mettre en avant que la simplification du réseau aux seuls composants ayant au moins une entrée/sortie sur le chemin emprunté par le flux de données permet d'étudier des systèmes bien plus grands tout en gardant des résultats cohérents (95% des évolutions possibles sont estimées avec une erreur inférieure à 1.5%).

Il a donc été choisi de ne garder que les modèles de composants qui sont sur la route des données utiles pour la détermination de la performance temporelle étudiée. En d'autres termes, ceux qui génèrent, modifient ou propagent les données (que ce soient des trames Ethernet ou des variables logiques) qui sont fonctions des évènements d'entrée et de sortie sur lesquels la performance temporelle considérée est basée. La mise en oeuvre de cette solutions sera détaillée dans la partie 4.3 avec notamment la présentation de l'algorithme de parcours de la structure du modèle pour en extraire les composants à conserver.

Il ne faut cependant pas oublier que, dans le cas de l'évaluation de bornes de performances, il est totalement impossible de supprimer ainsi des composants sans en conserver l'impact dans le modèle abstrait. En effet, ceci amènerait à faire une sous-approximation du modèle, approximation qui n'engloberait pas tous les comportements possibles. Dans le cadre de cette thèse, il est donc nécessaire de trouver un moyen de conserver l'impact des composants supprimés, sans pour autant garder ces composants dans le modèle abstrait de l'AAR.

4.2.3 Techniques visant à modifier les modèles de composants d'un modèle formel

Une partie de la solution à cette difficulté de ne pas sous-approximer le comportement de l'AAR étudiée est contenue dans les travaux de [Kro06] qui propose l'**insertion de points de coupe** (cut-point). Le principe de cette méthode (figure 4.7) est de remplacer un signal secondaire (ici C=f(A,B) qui est fonction de A et B), obtenu normalement à partir d'autres signaux en amont dans l'architecture, par un nouveau signal primaire (C) ne dépendant d'aucun autre signal. L'idée sous-jacente est qu'il peut être plus efficace de considérer qu'un signal (dont le calcul du comportement réel est compliqué à faire) peut évoluer n'importe quand (sur-approximation du modèle) plutôt que de devoir calculer son évolution réelle. Toutes les évolutions initiales du modèle détaillé sont conservées mais des

évolutions impossibles dans ce modèle peuvent être introduites et perturber la vérification. Il conviendra donc de s'assurer que les résultats de preuve négatifs ne proviennent pas de ces évolutions dues à la sur-approximation.

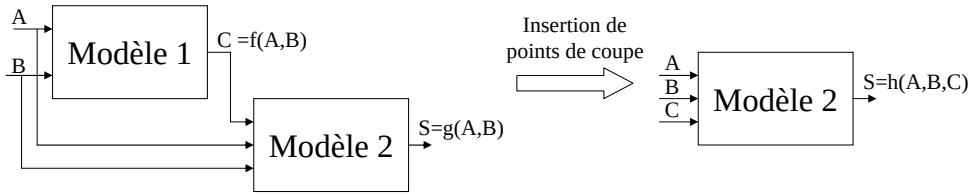


FIG. 4.7 – Principe de l'insertion de points de coupe

La modification des modèles de composants ira dans ce sens. L'impact de composants supprimés dans la modélisation abstraite de l'AAR sera conservé au moyen d'une modélisation au pire des cas. Par exemple, pour un MES en communication avec deux cartes de communication, il est beaucoup plus rapide de vérifier le comportement de l'AAR en considérant que l'une des requêtes peut arriver à n'importe quelle date, que de devoir modéliser le système complet afin de savoir quand cette requête peut arriver réellement.

La modification des modèles de composants sera détaillée dans la partie 4.4.

4.3 Simplification de la structure du modèle d'AAR

4.3.1 Principe

La première étape de la méthode proposée consiste à simplifier la structure du modèle initial en conservant uniquement les composants qui introduisent des délais (temps de traitement, temps d'attente de disponibilité d'une ressource ou attente de synchronisation) qui impactent **directement** la performance temporelle étudiée. Tous les autres composants seront retirés du modèle de l'AAR.

Les modèles de composants à conserver sont faciles à déterminer : ils sont situés sur la route des données utiles pour la détermination de la performance temporelle étudiée. En d'autres termes, ils génèrent, modifient ou propagent les données (que ce soient des trames Ethernet ou des variables logiques) qui sont fonctions des évènements d'entrée et de sortie sur lesquels la performance temporelle considérée est basée.

Ceci peut être illustré sur le modèle de la figure 4.8. Cette figure représente la structure de l'AAR de la figure 3.18 ; les rectangles représentent des modèles de composants, les flèches des échanges de données. La performance temporelle dont les bornes sont cherchées est la différence des temps de réponse entre l'émission des sorties S1 et S2 lorsqu'un évènement d'entrée E est observé simultanément sur les modules d'entrée/sortie M1 et M5 et que les évènements de sortie sont générés également par ces mêmes modules d'entrée/sortie MES1 et MES5. Dans ce cas, seulement les composants suivants restent dans le modèle à la fin de la simplification de la structure :

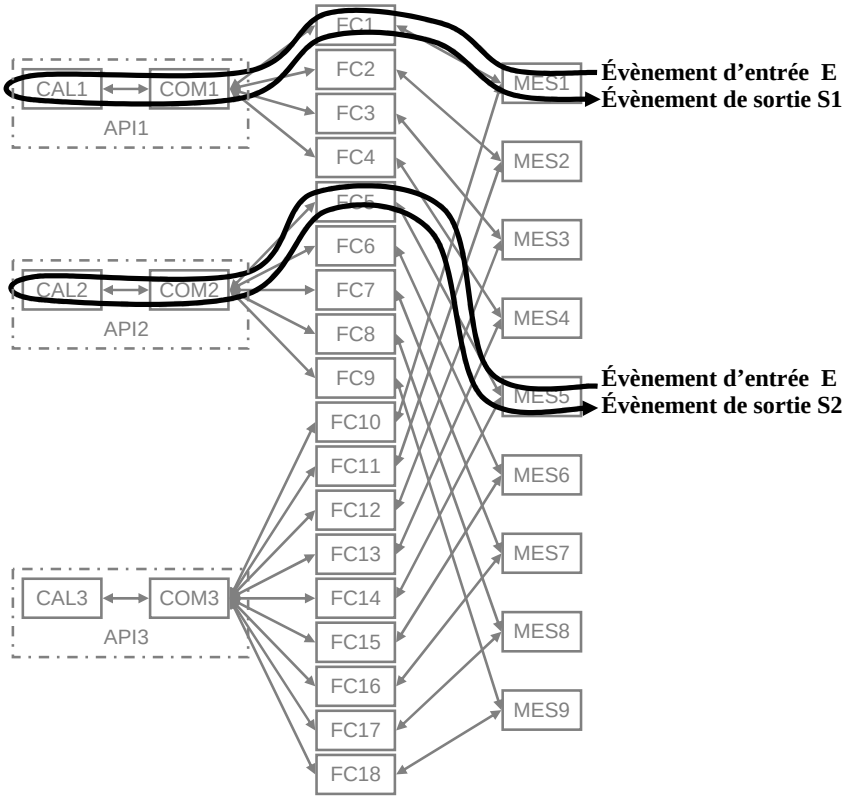


FIG. 4.8 – Route des données lors de l'étude de la différence de temps de réponse

- les modules d'entrée/sortie M1 et M5 qui reçoivent l'évènement d'entrée E, envoient des trames Ethernet qui contiennent la valeur de cet évènement, et génèrent également, à partir des trames émises par les cartes de communications COM1 et COM2, les évènements de sorties S1 et S2 ;
- les automates programmables industriels API1 et API2, constitués respectivement de CAL1 et COM1, CAL2 et COM2, qui reçoivent les trames contenant la valeur de l'évènement d'entrée E, exécutent leurs programmes avec cette valeur de E et enfin répondent respectivement aux modules d'entrée/sortie M1 et M5 par le moyen de nouvelles trames contenant les valeurs des évènements de sortie S1 et S2 ;
- les fonctions de communication FC1 et FC5 qui relient les deux couples de modèles (M1, COM1) et (M5, COM2).

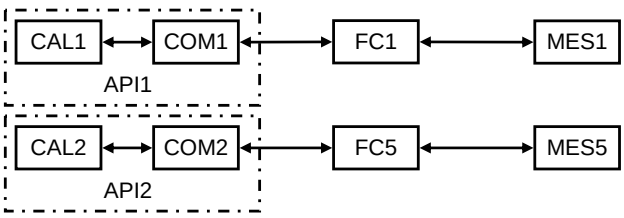


FIG. 4.9 – Structure simplifiée de l'AAR de la figure 4.8

Tous les autres modèles de composants, à savoir ceux de l'API3 (processeur de calcul

CAL3 et carte de communication COM3), des modules d'entrée/sortie MES2, MES3, MES4, MES6, MES7, MES8 et MES9 et des fonctions de communication FC2, FC3, FC4, FC6, FC7, FC8, FC9, FC10, FC11, FC12, FC13, FC14, FC15, FC16, FC17 et FC18 ne sont pas conservés lors de la simplification de la structure car ils ne sont pas sur la route des données. La structure simplifiée de ce modèle est représentée figure 4.9. Il est intéressant de remarquer que ce modèle abstrait est maintenant constitué de deux sous-modèles indépendants (CAL1, COM1, FC1 et MES1) et (CAL2, COM2, FC5 et MES5). Les deux API API1 et API2, liés par les modules d'entrée/sortie qu'ils partageaient dans le modèle initial sont maintenant découplés. Ce découplage serait une sous-approximation du modèle initial si les modèles de composants n'étaient pas adaptés pour en compenser les effets, ce qui est le cas dans la méthode globale d'abstraction que nous proposons (figure 4.6).

Les conséquences de ce découplage sur les modèles de composants et sur la méthode d'obtention des bornes de la différence des temps de réponse seront discutés respectivement dans la partie 4.4 et le chapitre 5).

4.3.2 Construction automatique du modèle simplifié

Sur la base du principe précédent, le modèle simplifié peut donc être construit automatiquement en recherchant dans le graphe représentant la structure du modèle d'AAR le chemin le plus court entre l'évènement d'entrée et l'évènement de sortie, chemin qui passe obligatoirement par le noeud correspondant au modèle du processeur de calcul qui sert à déterminer la valeur du signal de sortie. Seulement les modèles de composants qui correspondent aux noeuds le long de ce chemin sont gardés dans le modèle simplifié.

Cette recherche peut être réalisée au moyen de l'algorithme de Dijsktra [Dij59]. Il n'est cependant pas possible d'appliquer directement cet algorithme car il ne permet pas, initialement, de contraindre le chemin à passer par le processeur de calcul de l'API. Pour compenser ce manque, il est possible de faire deux recherches successives des plus courts chemins : la première pour déterminer le plus court chemin entre le module d'entrée/sortie où se situe l'évènement d'entrée et le processeur de calcul associé, la seconde pour le chemin entre le processeur de calcul et le module d'entrée/sortie où est émis l'évènement de sortie. Le chemin global le plus court est la concaténation de ces deux chemins. Les modèles de composants à garder correspondent à l'ensemble des noeuds sur ce chemin global.

Il est bon de noter que, pour cette étude, la longueur du chemin se détermine en nombre de composants à traverser ; il n'y a aucun poids sur les arcs, ou plutôt tous les poids des arcs sont de 1, car le chemin sera toujours du même type : environnement, MES, FC, carte de communication, processeur de calcul, carte de communication, FC, MES, observateur.

4.4 Modification des modèles de composants

Comme nous l'avons vu précédemment, la structure du modèle de l'architecture d'automatisation en réseau a été simplifiée par la suppression de tous les composants n'inter-

venant pas directement sur les données utiles pour la détermination de la performance temporelle étudiée. Ces suppressions de modèles de composants ont un impact direct et majeur sur le modèle de l'AAR : sans aucune autre modification, ce modèle ne peut plus fonctionner. Par exemple, certains canaux de synchronisation émetteurs (de type synchro !) n'ont plus de récepteur (de type synchro ?) et deviennent donc bloquants. Par ailleurs, outre ces problèmes de blocage, le modèle abstrait ne représente plus la même architecture car, si les composants (supprimés) n'agissaient pas directement sur les données utiles pour la détermination de la performance temporelle étudiée, ils avaient un impact indirect (impact sur les composants restant dans le modèle de l'AAR) et cet impact ne peut être totalement ignoré sous peine de ne plus pouvoir avoir confiance dans les résultats de vérification obtenus. Il convient donc de modifier les modèles des composants restants selon les règles explicitées dans la partie 4.4.1 et illustrées par des exemples concrets dans la partie 4.4.2.

4.4.1 Règles de modification des modèles de composants

L'analyse des communications entre les modèles de composants montre qu'elles peuvent être de deux types :

- échanges de données au moyen de variables partagées,
- synchronisations pour forcer des évolutions simultanées.

Le premier cas est simple alors que le second est un peu plus particulier puisque toute synchronisation devient impossible dès que l'un des deux modèles n'est plus présent. Il faudra donc les supprimer tout en prenant en compte l'impact du composant supprimé sur le comportement du composant restant. Il faudra donc distinguer les cas où le composant supprimé est récepteur ou émetteur de la synchronisation. Dans ce dernier cas, il faudra également étudier la concurrence entre des signaux de synchronisation.

4.4.1.1 Traitement des variables partagées

Dans le cas des variables partagées, le problème est relativement simple et peut se décomposer en trois cas :

- la variable partagée n'était utilisée que dans des modèles de composants qui ont été supprimés,
- la variable était affectée par un des modèles restants et lue par un des modèles supprimés,
- la variable était affectée par un des modèles supprimés et lue par un des modèles restants.

Il est évident qu'une variable qui n'était utilisée que dans des modèles de composants qui ont été supprimés est purement et simplement supprimée.

Une variable partagée affectée par un des modèles restants et lue par un des modèles supprimés peut être supprimée, sous réserve qu'elle ne soit lue par aucun modèle restant.

Dans le cas d'une variable affectée par un des modèles supprimés et lue par un des modèles restants, il peut sembler difficile, en première approche, de la supprimer. En effet, si elle intervient dans des gardes, sa suppression enlèverait tout sens au modèle. Il

est alors nécessaire de remarquer, qu'à part pour le modèle de l'observateur, les variables partagées n'interviennent jamais dans l'écriture des gardes, celles-ci ne comportant que des références à des horloges ou à des variables locales (variables propres à un modèle de composant). Dans le cas particulier de l'automate observateur, les variables utilisées dans les gardes font partie des données utiles pour la détermination de la performance temporelle étudiée. Elles ne seront donc jamais supprimées ainsi que les modèles de composants qui les manipulent. Dans tous les autres modèles, les variables partagées issues de modèles de composants supprimés sont uniquement lues puis leur valeurs sont affectées à d'autres variables partagées, elles-mêmes destinées à être lues par des modèles de composants ayant été supprimés. Elles peuvent donc, elles aussi, être supprimées.

Pour conclure, il est possible de supprimer :
- les variables uniquement utilisées dans des modèles de composants supprimés,
- les variables lues par des modèles de composants supprimés, si elles ne sont pas utilisées dans des modèles de composants restants,
- les variables affectées par des modèles de composants supprimés.

Au final, seules les variables à la fois affectées et lues par des composants non supprimés doivent être gardées.

4.4.1.2 Synchronisation avec un modèle de composant supprimé agissant en tant que récepteur

Dans les différents modèles de composants proposés dans le chapitre 3, les récepteurs sont toujours prêts à recevoir un message de synchronisation, ils ne sont donc jamais bloquants. Ce sont toujours les émetteurs qui imposent les évolutions et les synchronisations.

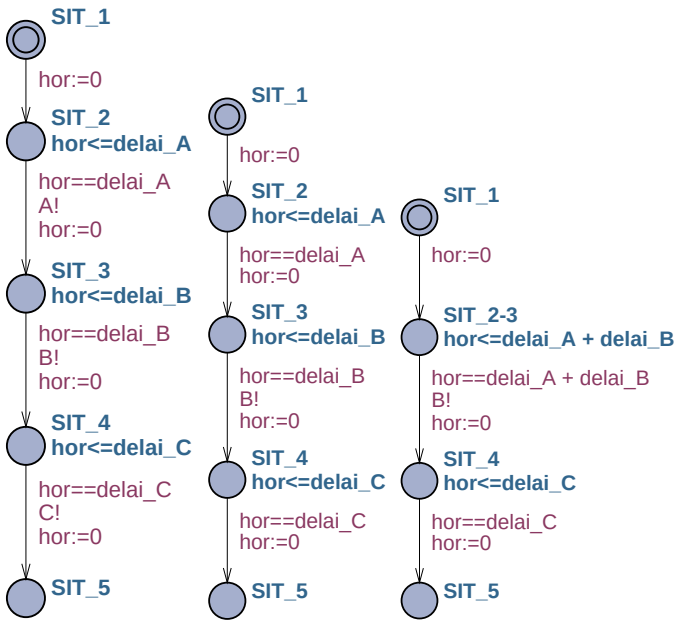


FIG. 4.10 – Modification par suppression de synchronisations

Par suite, quand un modèle de composant représente une tâche qui se termine par

l'émission d'un signal de synchronisation avec un autre modèle de composant (récepteur du signal de synchronisation), cette émission se fait dès que le modèle de l'émetteur le permet (validation des gardes et des invariants du modèle). Si le récepteur de cette synchronisation vient à être supprimé lors de la simplification de l'AAR, il n'y a aucune raison que l'évolution du modèle de composant émetteur de la synchronisation soit retardé ou bloqué par ce signal qui ne synchronise plus aucun autre modèle. Cette synchronisation peut donc être supprimée.

Ainsi sur le modèle de la figure 4.10 (à gauche), si seule la synchronisation B ! a un sens (les modèles qui devraient être synchronisés par A ! et C ! ne sont plus présents dans le modèle de l'AAR), la modification du modèle amène au modèle du centre. Ce qui peut être encore simplifié en fusionnant les situations SIT_2 et SIT_3 pour obtenir le modèle de droite.

4.4.1.3 Synchronisation avec un modèle de composant supprimé agissant en tant qu'émetteur ; modélisation de la concurrence entre synchronisations

Quant au contraire du cas précédent, le modèle de composant supprimé est le modèle émetteur de la synchronisation qui déclenche une évolution dans un des modèles de composant restants (le récepteur), le principe de modification de ce modèle de composant est tout autre. La réception de la requête peut intervenir n'importe quand (c'est l'émetteur qui impose la date de la synchronisation et non le récepteur) ; par suite, il n'est plus possible de considérer que la réception se fait "au bon moment", comme précédemment, mais "à n'importe quel moment".

Si cette synchronisation est la seule figurant dans le modèle de composant restant, elle peut être simplement supprimée. Ceci n'est pas possible s'il y a concurrence entre plusieurs synchronisations. Ainsi dans le cas ou plusieurs synchronisations différentes (avec plusieurs modèles de composants différents) peuvent avoir lieu simultanément, si l'une des synchronisation se fait avec un modèle de composant ayant été supprimé, il est nécessaire de prendre en compte l'impact de cette synchronisation.

Sur les différents modèles de composants, la concurrence entre deux synchronisations consiste toujours à la prise en compte d'une synchronisation avant une autre, ce qui a pour conséquence de retarder la prise en compte de la seconde (et non de l'empêcher) et par suite le traitement qui lui est associé.

Pour illustrer ce problème de concurrence entre deux synchronisations, l'exemple de la figure 4.11 sera utilisé. Ce modèle représente un composant qui doit, au cours d'une phase d'attente (représentée par la situation SIT_2), se synchroniser avec deux autres modèles. Ces deux synchronisations (A ? et B ?) peuvent se faire dans n'importe quel ordre et à n'importe quelle date, pourvu que ce soit avant la fin de cette phase d'attente.

Faisons l'hypothèse que, après simplification de la structure, le composant émetteur de la synchronisation B ! n'est plus présent dans le modèle de l'AAR et donc la synchronisation B ? ne peut plus avoir lieu. Dans le modèle initial, deux évolutions dues à des synchronisations sont possibles à partir de SIT_2 : soit la synchronisation A ? se fait avant B ? soit c'est l'inverse qui se produit. Si A ? a lieu avant B ?, la tâche (de durée delai_A) liée

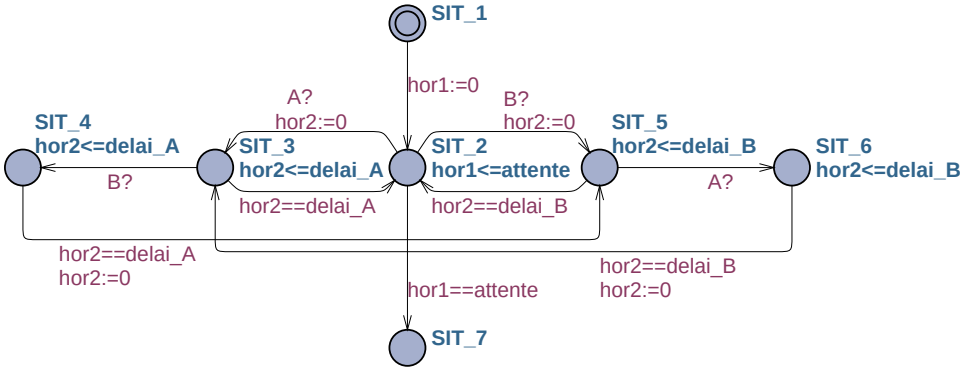


FIG. 4.11 – Cas de concurrence entre deux synchronisations (modèle initial)

à cette synchronisation a lieu dès la réception du signal de synchronisation (A ?), la tâche (de durée delai_B) liée à la synchronisation B ? aura lieu après, elle n'a aucun impact sur la synchronisation A ?. Dans le modèle abstrait, et pour cette évolution, la synchronisation B ? peut ne pas être représentée (elle n'a donc aucun impact sur la suite des évolutions). Par contre, si initialement la synchronisation B ? a lieu avant la A ?, la tâche liée à B ? peut retarder la tâche liée à A ? d'une durée comprise entre 0 (la synchronisation A ? arrive après la fin du traitement de la tâche liée à B ?) et delai_B (la synchronisation A ? arrive juste après la synchronisation B ?). Ce comportement peut être représenté comme sur la figure 4.12 à gauche, ce qui peut être réduit au modèle de droite simplement en fusionnant les situations SIT_13 et SIT_14 et en modifiant convenablement une garde et un invariant.

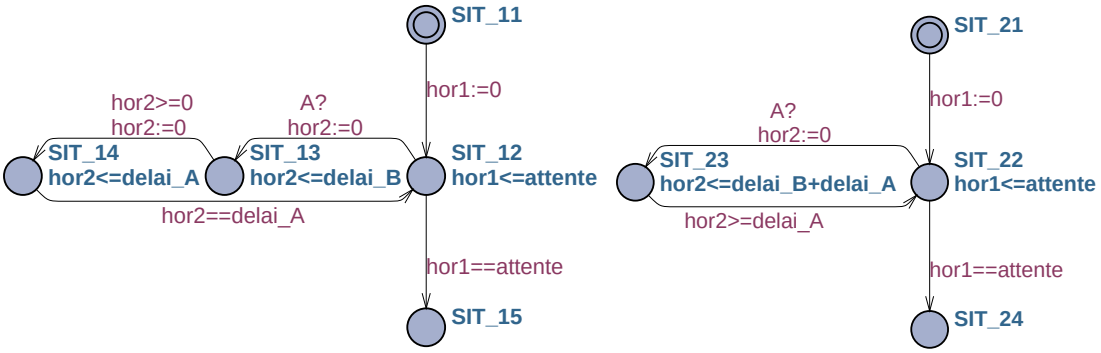


FIG. 4.12 – Modélisation de la concurrence entre synchronisations émises par un composant conservé et un composant supprimé

Ce raisonnement peut être généralisé à la concurrence entre une synchronisation provenant d'un composant conservé et n synchronisations provenant de composants supprimés. Lors de la synchronisation avec le composant conservé, avant de traiter la tâche qui y est associée, une attente de durée variable doit avoir lieu pour représenter les n synchronisations qui ne sont plus représentées. Cette attente peut varier entre 0 (la synchronisation avec le composant conservé est arrivée avant les autres synchronisations) et la somme des délais des tâches associées aux n synchronisations avec les composants supprimés

(la synchronisation avec le composant conservé est arrivée juste après toutes les autres synchronisations sans qu'aucune des tâches n'ait été effectuée).

4.4.2 Applications de ces règles sur les modèles de composants

Tous les modèles de composants ont été modifiés en utilisant les règles énoncées précédemment. Comme nous pourrons le voir, pour certains modèles (processeur de calcul et fonctions de communication), les modifications apportées par ces règles sont mineures alors que pour d'autres (carte de communication et MES), les modifications sont importantes.

4.4.2.1 Processeur de calcul

Le processeur de calcul ne communique qu'avec un seul autre modèle : la carte de communication qui lui est associée. Ces deux modèles sont intimement liés lors de la phase de simplification de la structure du modèle de l'AAR. Ils ne peuvent pas être dissociés ; si l'un est conservé, l'autre le sera forcément.

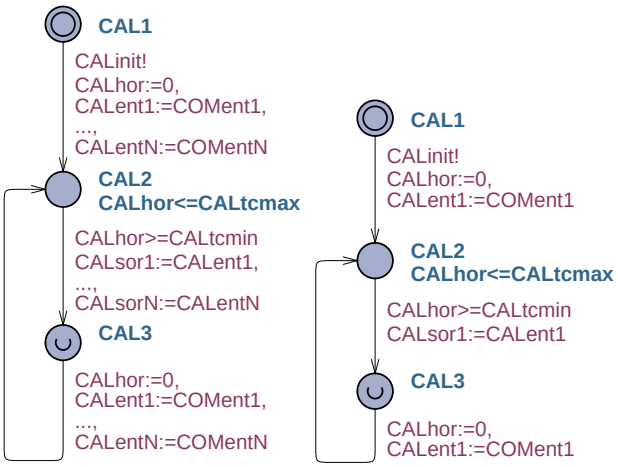


FIG. 4.13 – Modèle d'un processeur de calcul avant et après modification pour recevoir une seule valeur d'entrée et ne calculer qu'une sortie

Les modifications à apporter au modèle de processeur de calcul (figure 4.13) vont porter sur le nombre de variables partagées. En effet, seules celles intervenant directement dans la performance temporelle étudiée seront conservées. Les N variables CALent et CALsor, qui représentent les valeurs en provenance ou à destination des N MES connectés au processeur de calcul via la carte de communication et le réseau ne sont plus toutes nécessaires, seules celles correspondantes aux MES conservés dans le modèle abstrait sont utiles.

En ce qui concerne les synchronisations, le modèle du processeur de calcul émet une seule synchronisation (CALinit !) vers la carte de communication qui lui est associée et n'en reçoit aucune. Comme la présence du modèle de la carte de communication est indissociable de celle du processeur de calcul, il n'y a aucune modification à faire pour les synchronisations.

4.4.2.2 Carte de communication

Les modifications apportées sur le modèle de carte de communication vont être illustrées par la figure 4.14 dans le cas d'une carte de communication communiquant initialement avec N modèles de MES mais dont deux seulement sont conservés dans le modèle abstrait de l'AAR (modèle utilisé quand l'entrée est lue sur un MES et la sortie émise sur un autre). Le modèle d'une carte de communication communiquant initialement avec N modèles de MES mais dont un seul est conservé dans le modèle abstrait de l'AAR (modèle utilisé notamment quand l'entrée est lue et la sortie émise sur le même MES) pourra être facilement déduit de cette analyse.

En ce qui concerne les variables partagées, les modifications vont porter sur la suppression de toutes les variables liées aux N-2 MES supprimés dans le modèle abstrait de l'AAR (CALsor3 à CALsorN, COMsor3 à COMsorN, COMent3 à COMentN et trame3 à trameN).

Pour les synchronisations émises (représentant les émissions des requêtes vers les MES) par le modèle de carte de communication, les délais associés à chaque émission sont conservés mais seules les deux synchronisations avec les 2 modèles de MES conservés dans le modèle abstrait de l'AAR sont émises (CRreq1 ! et CRreq2 !). Trois attentes (COM3, COM5 et COM7) sont ainsi introduites pour représenter la durée des émissions des requêtes avant l'émission de la requête vers le premier modèle de MES conservé, entre les émissions des deux requêtes vers les modèles de MES conservés et après l'émission de la requête vers le deuxième modèle de MES conservé. Ces attentes sont entièrement configurables au moyen des paramètres nbrAV (nombre de requêtes normalement émises avant celle vers le premier modèle de MES conservé), nbrEN (nombre de requêtes normalement émises entre les celles vers les deux modèles de MES conservés) et nbrAP (nombre de requêtes normalement émises après celle vers le deuxième modèle de MES conservé). Il faut juste s'assurer que $nbrAV + nbrEN + nbrAP + 2 = N$.

Pour les synchronisations reçues par le modèle de carte de communication (représentant les réceptions des réponses), seules les deux synchronisations avec les 2 modèles de MES conservés dans le modèle abstrait de l'AAR sont gardées (RCrep1 ? et RCrep2 ?). Comme la réception des réponses se fait à temps nul, les N-2 autres synchronisations peuvent être supprimées directement, elles n'ont pas d'impact sur les deux synchronisations qui restent car il n'y a pas de délai lié à la réception de ces synchronisations à prendre en compte.

Au passage, il est possible de constater, que dans ce modèle modifié, la représentation de la file d'attente des réponses à l'aide d'une liste d'entiers a été remplacée par une prise en compte de l'ordre d'arrivée des réponses directement par la structure du modèle pour gagner en efficacité lors de la phase de vérification (cf discussion sur ce sujet dans la partie 4.1.1).

4.4.2.3 Fonction de communication

Le modèle du réseau, représenté par un ensemble de fonctions de communication, liant chacune une carte de communication à un MES, change de par le fait que certaines

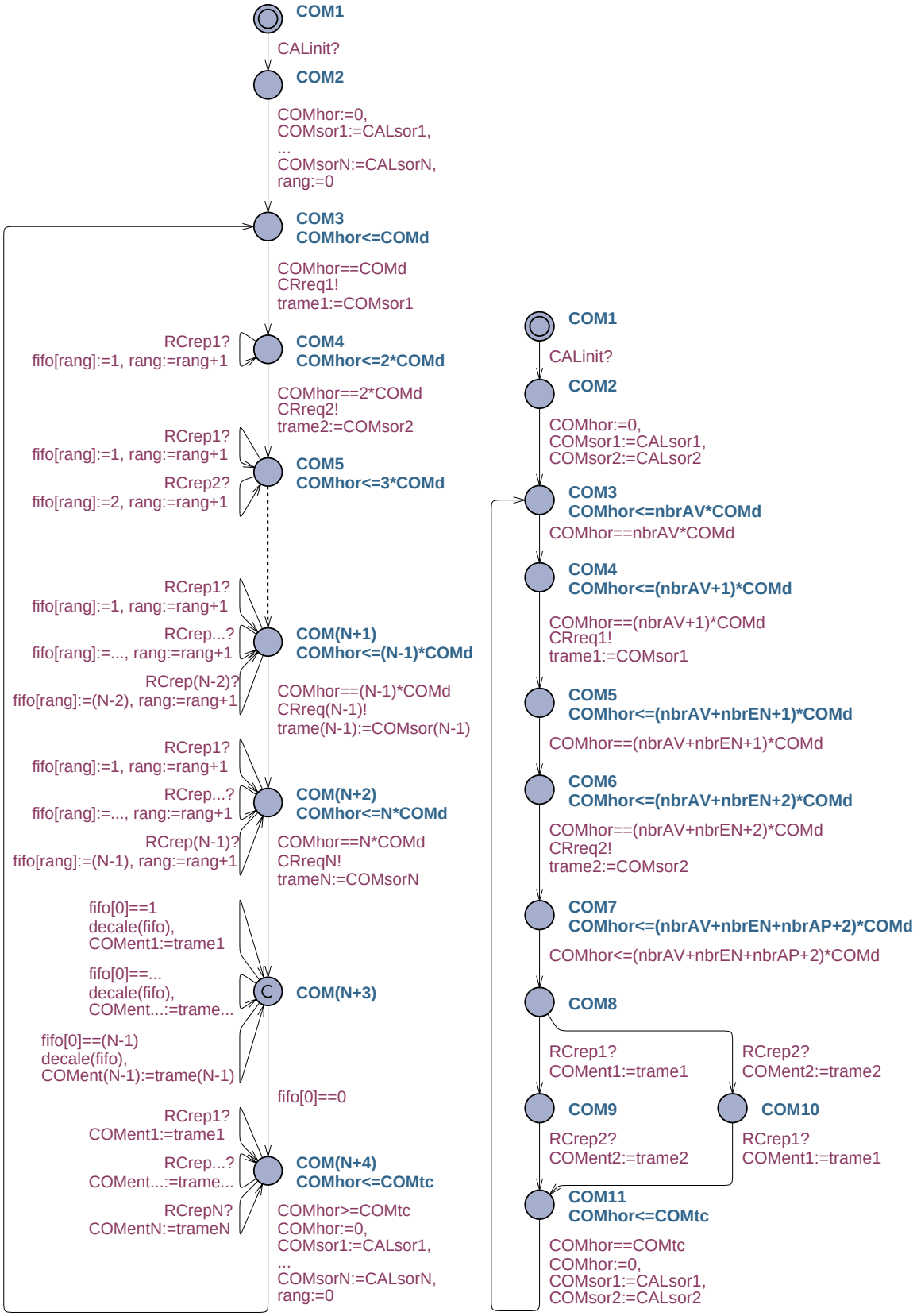


FIG. 4.14 – Modèle initial et modifié d'une carte de communication scrutant N MES dont 2 sont conservés dans le modèle abstrait

fonctions de communications ne sont plus représentées dans le modèle abstrait. Par contre, le modèle d'une fonction de communication ne change absolument pas.

Le modèle d'une fonction de communication ne comporte aucune variable partagée.

Il comporte par contre des émissions et des réceptions de synchronisations. Celles-ci ne se font qu'avec des modèles de composants qui sont conservés dans le modèle abstrait de l'AAR car, si le MES ou la carte de communication entre lesquelles se positionne la fonction de communication était supprimé, la fonction de communication le serait également.

Le modèle modifié d'une fonction de communication est donc identique à celui présenté dans la partie 3.3.4.

4.4.2.4 Module d'entrées/sorties déportées

Deux cas de modification seront présentés pour le modèle de MES. Le premier correspond à un MES communiquant avec N cartes de communication dont une seule est conservée dans le modèle abstrait de l'AAR (modèle utilisé dans le cas de l'étude d'un temps de réponse). Le second correspond à un MES communiquant avec N cartes de communication dont deux sont conservées dans le modèle abstrait de l'AAR (modèle utilisé dans le cas de l'étude d'une différence de temps de réponse, pour le MES qui reçoit l'évènement d'entrée).

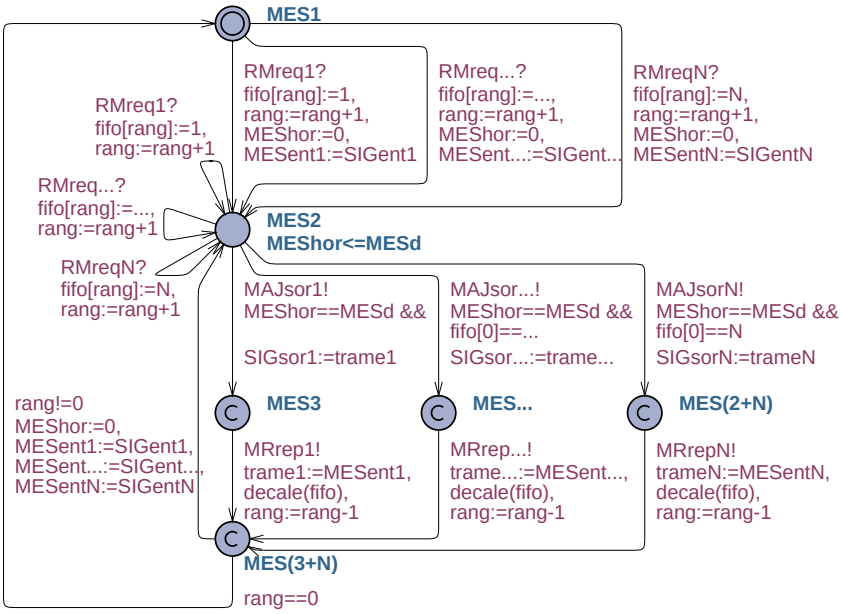


FIG. 4.15 – Modèle initial de MES communiquant avec N cartes de communication

Par souci de clarté, les modifications du modèle de MES sont expliquées sur le modèle de MES avec une représentation de la file d'attente des requêtes par une liste d'entiers (figure 4.15).

Dans le cas des modifications faites pour un MES communiquant avec N cartes de communication dont une seule est conservée dans le modèle abstrait de l'AAR (figure 4.16

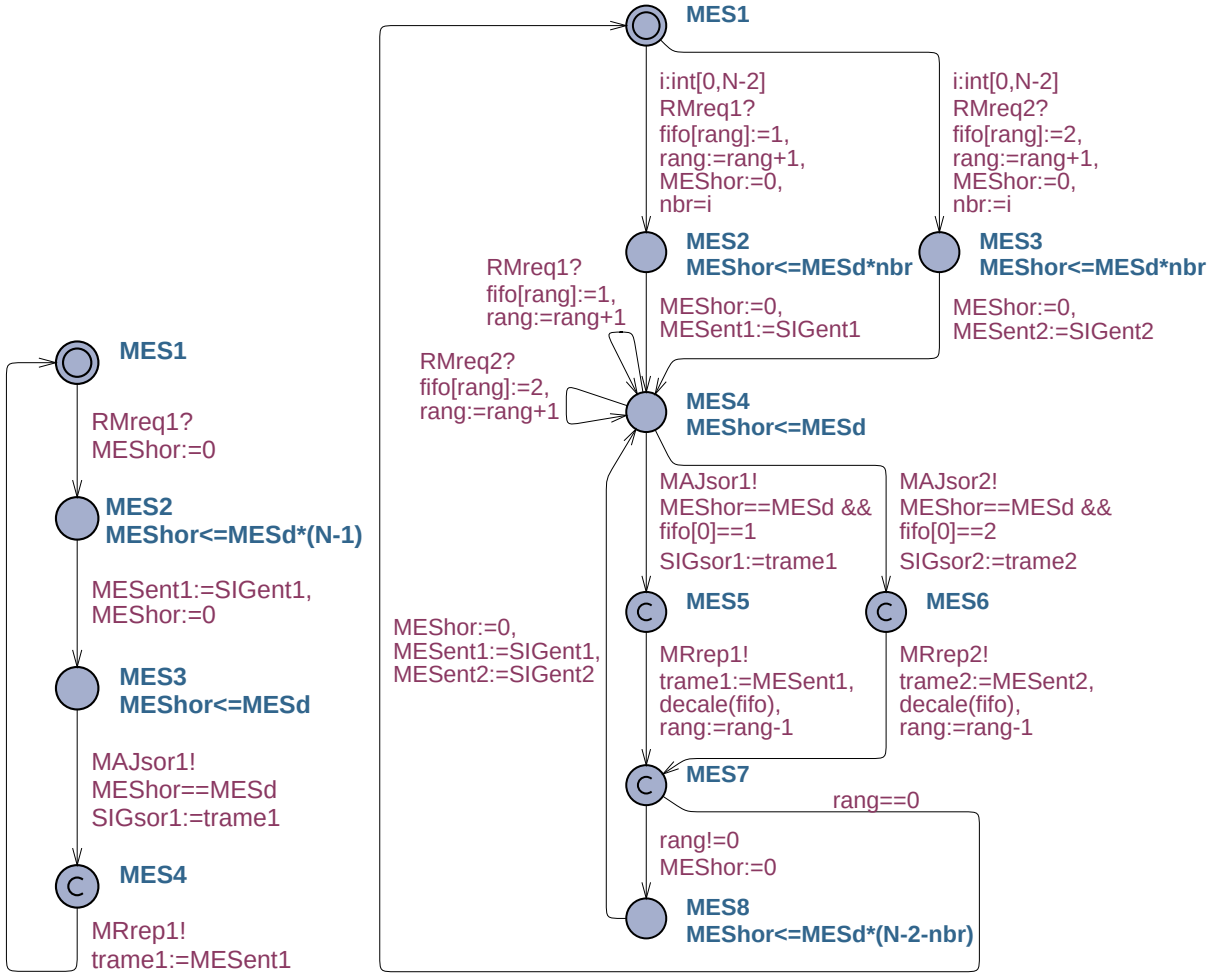


FIG. 4.16 – Modèle de MES communiquant avec N cartes de communication dont 1 (à gauche) ou 2 (à droite) sont conservées dans le modèle abstrait de l'AAR

à gauche), la première modification traite des variables partagées. Toutes les variables partagées peuvent être supprimées sauf celle lue ou modifiée par la carte de communication conservée (trame1) et les variables partagées qui lui sont directement liées (SIGent1, MESent1 et SIGsor1)[11].

Pour les synchronisations émises par le MES, seules MAJsor1 ! et MRrep1 ! sont conservées. Les autres (MRrep2 ! à MRrepN !) étaient destinées à des fonctions de communication qui ne sont pas présentes dans le modèle abstrait. MAJsor2 ! à MAJsorN ! sont simplement des doubles des synchronisations MRrep2 ! à MRrepN ! à destination de l'environnement ou de l'observateur qui étaient utilisées à cause de l'impossibilité de synchroniser plus de 2 modèles simultanément dans UPPAAL. Elles peuvent donc également être supprimées.

Pour les synchronisations reçues par le MES, seule RMreq1 ? est conservée mais l'impact de la concurrence avec les synchronisations RMreq2 ? à RM-

[11] On rappelle que les valeurs des entrées lues et des sorties émises par un MES sont encapsulées dans une trame transmise à ou envoyée par une carte de communication.

reqN ? doit être préservé. Lors de la réception de la synchronisation RMreq1 ?, un délai variable, compris entre 0 et (N-1)*MESd, doit être introduit dans le modèle avant de traiter la requête reçue, modélisé sous forme d'un invariant associé à la situation MES2.

La figure 4.16 à droite représente un modèle de MES communiquant avec N cartes de communication dont 2 sont conservées dans le modèle abstrait de l'AAR. Pour faciliter la compréhension, ces deux requêtes seront appelées requête1 et requête2.

Les modifications apportées au modèle initial (figure 4.15) portent tout d'abord sur les variables partagées. Comme pour le premier cas de modification, toutes les variables partagées peuvent être supprimées sauf celles lues ou modifiées par les cartes de communication conservées (trame1 et trame2) ainsi que les variables partagées qui leur sont directement liées (SIGent1, MESent1 et SIGsor1 pour les échanges avec la première carte de communication et SIGent2, MESent2 et SIGsor2 pour les échanges avec la deuxième carte de communication).

Pour les synchronisations émises par le MES, seules MAJsor1 !, MAJsor2 !, MRrep1 ! et MRrep2 ! sont conservées. Les autres (MRrep3 ! à MRrepN !) étaient destinées à des fonctions de communications qui ne sont pas présentes dans le modèle abstrait. Les autres synchronisations émises par le MES (MAJsor3 ! à MAJsorN !) sont directement liées aux précédentes et peuvent être supprimées en même temps qu'elles.

Pour les synchronisations reçues par le MES, seules RMreq1 ? et RMreq2 ? sont conservées mais l'impact de la concurrence avec les synchronisations RMreq3 ? à RMreqN ? doit être préservé. Dans le cas où deux cartes de communications sont encore présentes dans le modèle abstrait de l'AAR, la situation est un peu plus compliquée que pour le premier cas de modification. En effet, l'impact de la concurrence entre les requêtes reçues peut intervenir lors de la réception de requête1, de requête2 ou être réparti sur les deux réceptions.

De plus, il faut tenir compte du fait que seulement une requête par carte de communication peut être en attente de traitement (hypothèse faite dans le chapitre 1). Dans le cas où requête2 est reçue alors que requête1 n'a pas encore fini d'être traitée, il est nécessaire de tenir compte du fait que les k (k<=N-2) requêtes qui ont retardées le traitement de requête1 ne peuvent plus retarder requête2. Ceci est pris en compte, lors du franchissement des transitions entre MES1 et MES2 et entre MES1 et MES3, par un tirage aléatoire (entre 0 et le nombre maximum de requêtes issues des cartes de communication non présentes dans le modèle abstrait, N-2) du nombre *nbr* de requêtes retardant requête1, ce qui réduit à N-2-*nbr* le nombre de requêtes pouvant retarder requête2 (si elle est elle-même retardée par requête1). Dans le cas où requête1 et requête2 ne se retardent pas mutuellement, l'ensemble des autres requêtes peut retarder chacune d'elles.

4.4.2.5 Environnement

Les modifications à apporter au modèle de l'environnement sont présentées dans le cas d'un modèle d'environnement connecté à N sorties de l'AAR et servant à l'évaluation d'un temps de réponse (figure 4.17).

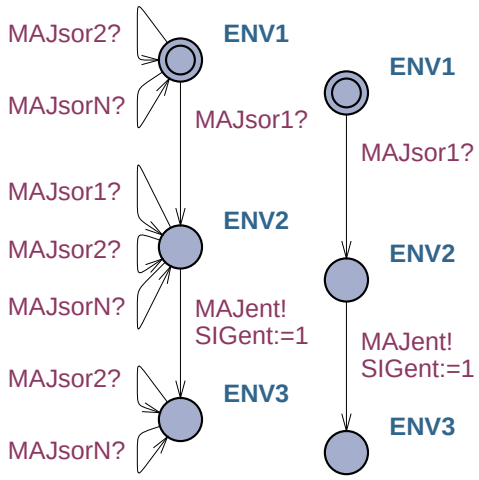


FIG. 4.17 – Modèle initial (à gauche) et modifié (à droite) de l'environnement pour l'évaluation d'un temps de réponse dans le cas d'une AAR à N sorties dont une seule est conservée dans le modèle abstrait de l'AAR

En ce qui concerne les variables partagées, il n'y a aucun changement car initialement il n'y en a qu'une, SIGent1, qui est la valeur du signal d'entrée observé par le système. Elle est forcément conservée.

Pour les synchronisations émises par l'environnement, il n'y en a qu'une MAJent ! pour signaler à l'observateur (qui est conservé dans le modèle abstrait de l'AAR) que l'évènement d'entrée a eu lieu. Cette synchronisation est indispensable dans le modèle, elle ne peut pas être supprimée.

Pour les synchronisations reçues par l'environnement, MAJsor1 ? provient du MES qui est conservé dans le modèle abstrait de l'AAR ; cette synchronisation est donc elle aussi conservée. Pour les autres (MAJsor2 ? à MAJsorN ?) elles proviennent de modèles de MES non présents dans le modèle abstrait de l'AAR, et **la concurrence entre ces synchronisations ne se traduit pas par une consommation de temps dans le modèle initial de l'environnement**. Toutes ces synchronisations peuvent donc être supprimées sans aucun problème.

Les modifications à apporter sur le modèle de l'environnement utilisé dans le cas d'une différence de temps de réponse (figure 4.18) sont les mêmes. Pour les synchronisations reçues par l'environnement, il faut simplement en conserver deux (MAJsor1 ? et MAJsor2 ?) au lieu d'une seule.

4.4.2.6 Observateur

Le modèle initial de l'observateur est construit pour ne communiquer qu'avec l'environnement et le ou les MES qui lisent l'entrée et écrivent la (les) sortie(s). Tous ces modèles de composants sont conservés dans le modèle abstrait de l'AAR. Le modèle de l'observateur ne subit donc aucune modification.

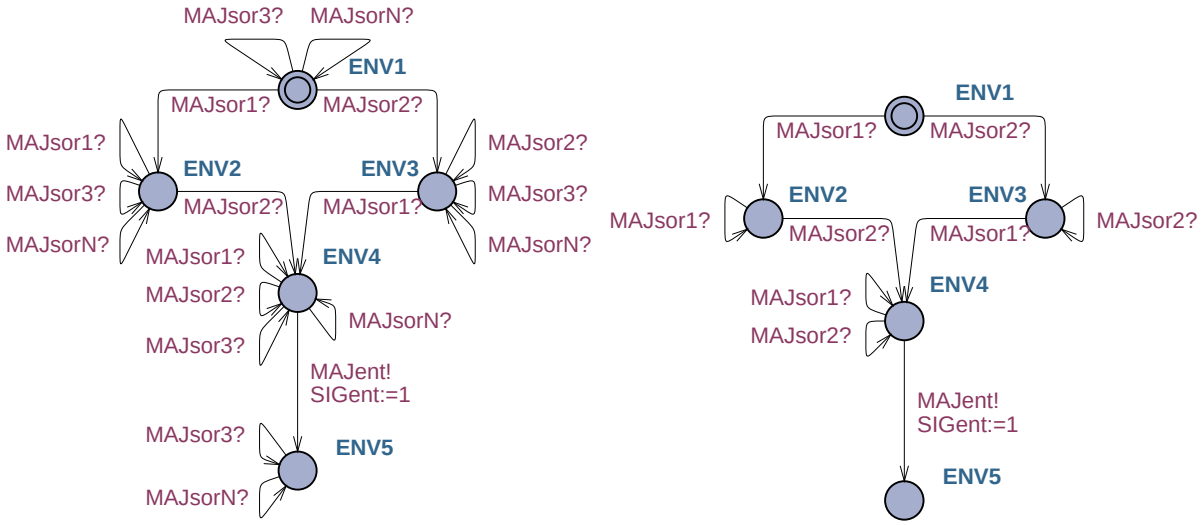


FIG. 4.18 – Modèle initial (à gauche) et modifié (à droite) de l'environnement pour l'évaluation d'un temps de réponse dans le cas d'une AAR à N sorties dont une seule est conservée dans le modèle abstrait de l'AAR

4.4.3 Validation de l'équivalence de comportement entre modèles initiaux et modèles modifiés

Les modifications apportées sur les modèles de composants nécessitent de vérifier que, pour le problème étudié, l'évaluation de bornes de performances temporelles, les deux modèles conduisent aux mêmes résultats.

La suppression de variables partagées et la suppression de l'émission de signaux de synchronisation (l'émission n'est jamais bloquante) ne modifient pas le comportement temporel du modèle modifié (cas des modèles de processeurs de calcul, des fonctions de communications et de l'observateur). Par contre, la prise en compte de la concurrence lors de la réception de signaux de synchronisation (modèles de carte de communication, de MES et de l'environnement) doit être vérifiée pour garantir qu'il n'y ait pas de sous-approximation du comportement de l'AAR.

4.4.3.1 Cas des modèles de carte de communication et de l'environnement

Ces deux cas peuvent être traités simultanément car, en ce qui concerne la prise en compte de la concurrence générée par les signaux de synchronisation reçus, ils ont le même comportement. Tous les deux comportent des situations d'attente de signaux de synchronisation, mais ces situations ont une durée nulle (voir les synchronisations reçues dans les situations COM4 à COM(N+4) pour le modèle initial de carte de communication à la figure 4.14 et dans toutes les situations pour les modèles d'environnement des figures 4.17 et 4.18).

La concurrence entre signaux de synchronisation n'impacte pas la durée d'évolution de ces modèles. La vérification de l'équivalence temporelle entre leurs versions initiale et modifiée est par conséquent inutile.

4.4.3.2 Cas du modèle de MES

Les comportements temporels des modèles initial et modifié ont été comparés sur la base des architectures schématisées figure 4.19, le modèle initial correspondant à un MES communiquant avec trois cartes de communication, COM1, COM2 et COM3 (cas représentatif d'une application industrielle très contrainte).

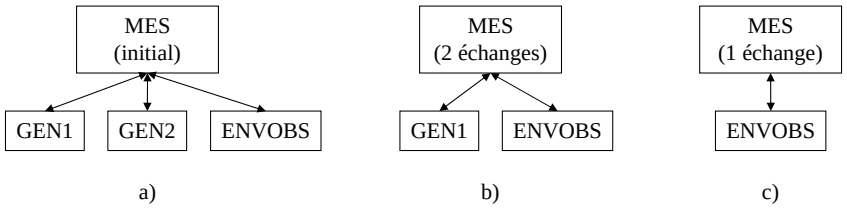


FIG. 4.19 – Architectures utilisées pour la comparaison des modèles de MES initial (trois échanges modélisés de façon explicite) (a), modifié avec deux échanges modélisés de façon explicite et un échange abstrait (b) et modifié avec un échange modélisé de façon explicite et deux échanges abstraits (c)

Le but de ces expériences est de **prouver** (avec la méthode présentée en partie 2.2) que pour les trois modèles (initial, modifié avec deux échanges modélisés de façon explicite et un échange abstrait, modifié avec un échange modélisé de façon explicite et deux échanges abstraits), les bornes du temps de réponse du MES à une requête de COM1 étaient identiques.

Il a été décidé, pour ces expériences, de remplacer les modèles de cartes de communication (trop lourds à utiliser dans ce cas) par des automates générateurs, pour représenter l'émission des requêtes (provenant de COM2 et COM3 normalement), et un automate ENVOBS, pour générer la requête "principale" (provenant de COM1) et mesurer le temps qui s'écoule jusqu'à la réception de la réponse du MES (figure 4.20).

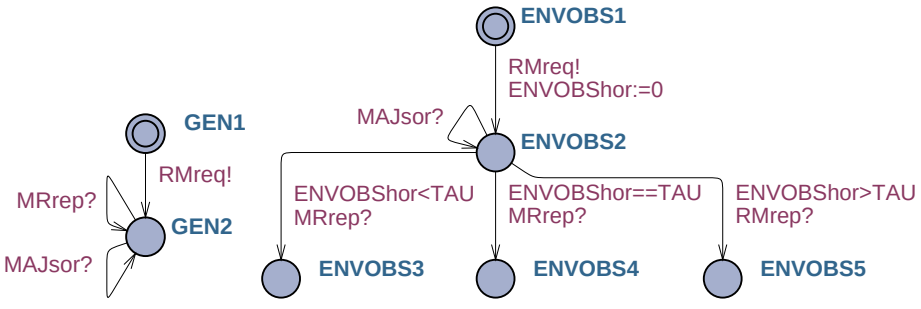


FIG. 4.20 – Modèles de générateur (à gauche) et d'ENVOBS (à droite) utilisés pour la validation des modèles modifiés de MES

Les résultats obtenus lors de ces trois vérifications sont contenus dans le tableau 4.4. La durée de traitement d'une seule requête, MESd, était de 0.7 ms pour les trois modèles de MES.

TAB. 4.4 – Bornes des temps de réponse d'un MES communiquant avec 3 cartes de communication

	borne minimale	borne maximale
cas 1	0,7 ms	2,1 ms
cas 2	0,7 ms	2,1 ms
cas 3	0,7 ms	2,1 ms

Ces résultats montrent que l'abstraction proposée pour ce modèle de composant ne modifie pas les bornes de son temps de réponse à une requête. La modélisation de la concurrence proposée est donc pertinente.

4.5 Conclusion

Dans ce chapitre, il a été montré tout d'abord que la modification de certains modèles de composants n'était pas suffisante pour permettre d'analyser des systèmes de grande taille. Pour lever cet obstacle, une méthode d'abstraction globale des modèles d'AAR a été présentée. Elle se base sur la simplification de la structure du modèle de l'AAR, puis sur la modification des modèles de composants restants pour prendre en compte les impacts indirects causés par les composants dont les modèles ont été supprimés. La comparaison des comportements temporels des modèles de composants initiaux et modifiés a montré que ces derniers traduisaient bien les mécanismes de consommation de temps dus à des attentes de signaux de synchronisation. Ces modèles modifiés pourront donc être utilisés pour l'évaluation des bornes des performances temporelles étudiées. Pour étudier l'impact de la simplification de la structure du modèle de l'AAR, il sera nécessaire de réaliser des analyses sur des systèmes complets et non sur des systèmes élémentaires comme pour la validation des modèles de composants. Ceci sera réalisé dans le chapitre suivant.

Chapitre 5

Validation expérimentale de nos propositions

Ce chapitre a pour but de valider l'ensemble de nos propositions, sur la base de six cas d'étude, en montrant que :
- la méthode globale d'abstraction augmente nettement les possibilités de passage à l'échelle ;
- les valeurs de bornes obtenues par preuves itératives de propriétés logiques sur un modèle formel sont très proches de celles mesurées sur une AAR réelle.

Sommaire

Ce chapitre va permettre de valider les propositions présentées aux chapitres 2, 3 et 4, sur la base de six cas d'étude. Ces cas d'étude sont présentés dans la partie suivante, et la structure de leurs modèles abstraits, obtenus comme indiqué au chapitre 4, dans la partie 5.2.

Notre proposition de méthode d'évaluation des bornes des performances temporelles reposant sur un outil de preuves de propriétés, les conditions des expériences réalisées avec cet outil sont détaillés en partie 5.3 ; pour chaque cas d'étude, quatre configurations d'analyse sont finalement retenues.

Les résultats expérimentaux obtenus nous permettent d'évaluer l'intérêt de la méthode globale d'abstraction proposée pour le passage à l'échelle, dans les parties 5.4 et 5.5.

Enfin, la validation de propositions théoriques ne pouvant être faite réellement que par confrontation au réel, la partie 5.6 permet de comparer les bornes obtenues par preuves itératives sur un modèle d'AAR à celles déduites de mesures sur une architecture réelle.

5.1 Présentation des différents cas d'étude

Deux architectures sont utilisées pour définir ces différents cas d'étude. La première (figure 5.1) est une architecture caractéristique d'AAR industrielles où deux API partagent plusieurs MES à des fins de synchronisation de processus et où un troisième API scrute un large ensemble de MES pour assurer la surveillance/supervision de ces processus. Cinq cas d'étude seront extraits de cette architecture. La deuxième (figure 5.7) est une architecture existante au LURPA, qui nous permettra également de réaliser une confrontation au réel présentée en partie 5.6.

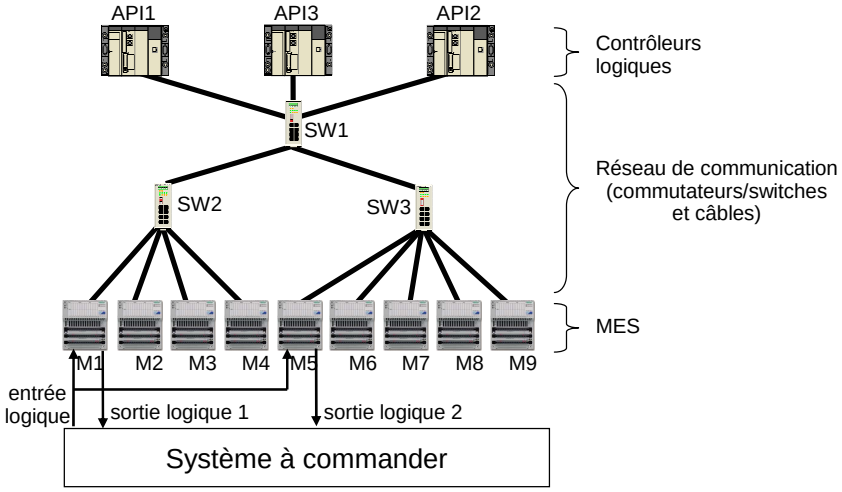


Fig. 5.1 – AAR utilisée dans les cas d'étude 1 à 5

La configuration des différents composants de l'architecture d'automatisation de la figure 5.1 est donnée dans le tableau 5.1.

Cette architecture peut être décomposée en plusieurs cas d'étude de taille et de complexité différentes. Ces cinq cas d'études sont explicités dans le tableau 5.2 et la structure

TAB. 5.1 – Valeurs des paramètres des API

	API1	API2	API3
durée d'exécution du programme	2 à 3 ms	3 à 4 ms	5 à 6 ms
durée du cycle d'I/O scanning	10 ms	10 ms	50 ms

des modèles initiaux correspondants est représentée par les figures 5.2, 5.3, 5.4, 5.5 et 5.6. Pour les trois premiers cas, le temps de réponse (noté TR dans le tableau) de l'AAR est étudié alors que pour les cas 4 et 5 c'est une différence de temps de réponse (notée DTR dans le tableau) qui est considérée.

TAB. 5.2 – Configurations des 5 cas d'étude

	cas 1	cas 2	cas 3	cas 4	cas 5
API	1	1	3	1 et 2	1, 2 et 3
MES	1	1 à 4	1 à 9	1 à 9	1 à 9
nombre de FC dans le modèle initial	1	4	9	9	18
performance temporelle	TR	TR	TR	DTR	DTR
signal d'entrée en	MES1	MES1	MES1	MES1 et MES5	MES1 et MES5
signal de sortie en	MES1	MES1	MES1	MES1 et MES5	MES1 et MES5

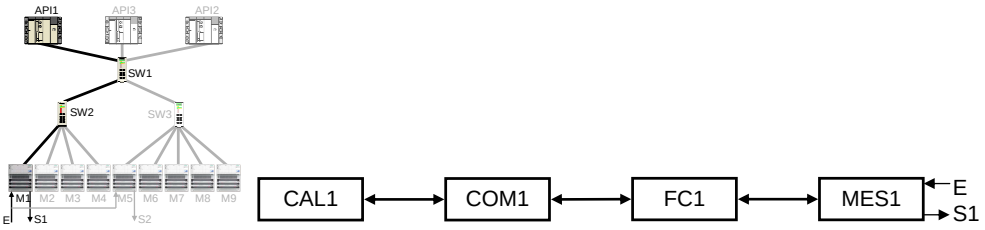


FIG. 5.2 – Cas d'étude 1 et structure de son modèle initial

Le cas d'étude n°1, dont la structure du modèle initial est représentée figure 5.2, est le modèle d'architecture le plus simple que l'on puisse trouver puisqu'il n'est composé que d'un API (un modèle de processeur de calcul, CAL et un modèle de carte de communication, COM) d'une fonction de communication, FC et d'un modèle de module d'entrée/sortie, MES. Ce modèle est en soit trop simple pour montrer l'intérêt de la méthode de simplification mais il correspond, dans plusieurs autres cas, au modèle simplifié de l'architecture étudiée.

Il est rappellé que, dans la représentation de la structure des modèles, un rectangle représente un automate temporisé et une double flèche des communications entre 2 automates temporisés au moyen de variables partagées et de canaux de communication.

Les cas d'étude n°2 et 3 ne comportent eux aussi qu'un seul API mais qui scrutent cette fois respectivement quatre et neuf modules d'entrées/sorties.

Le quatrième cas d'étude comporte deux API, ce qui complexifie beaucoup le comportement de l'architecture car ces deux APIs sont totalement indépendants l'un de l'autre et

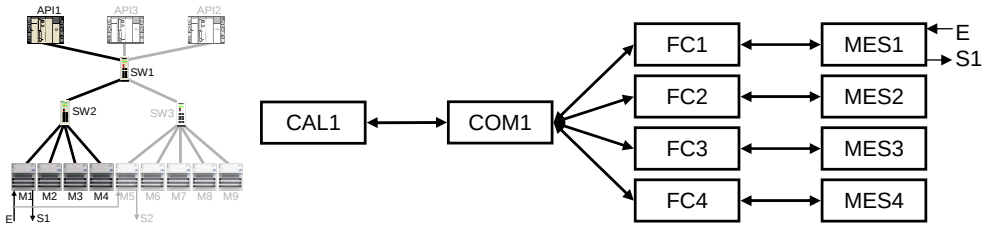


FIG. 5.3 – Cas d'étude 2 et structure de son modèle initial

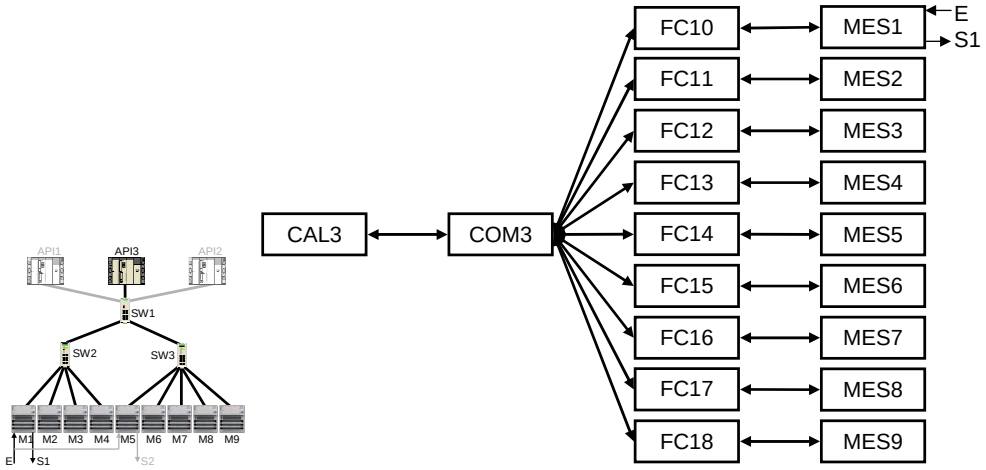


FIG. 5.4 – Cas d'étude 3 et structure de son modèle initial

donc non synchronisés. L'éventail des possibilités d'évolution est donc bien plus important que pour les trois cas précédents.

Enfin le cinquième cas d'étude tiré de cette architecture d'automatisation correspond à l'ensemble de l'architecture avec ses trois API et ses neuf MES, ce qui fait un total de dix-huit fonctions de communication.

Le sixième cas d'étude est basé sur une seconde architecture, qui correspond à un cas réel existant au LURPA, se composant de deux API, neuf MES dont trois sont partagés par les deux APIs. Enfin, sur ce cas d'étude, la performance étudiée est à nouveau un temps de réponse qui correspond à l'écart temporel entre un évènement reçu par un MES (le MES4) et l'évènement conséquence émis par un autre MES (le MES5). D'après le logiciel PL7pro utilisé pour programmer les APIs, les deux processeurs de calcul ont des temps de cycle compris entre 2 et 3 ms et les cartes de communication sont configurées avec des cycle d'IOscanning de 10 ms. La structure de ce sixième cas d'étude est représentée sur la figure 5.8.

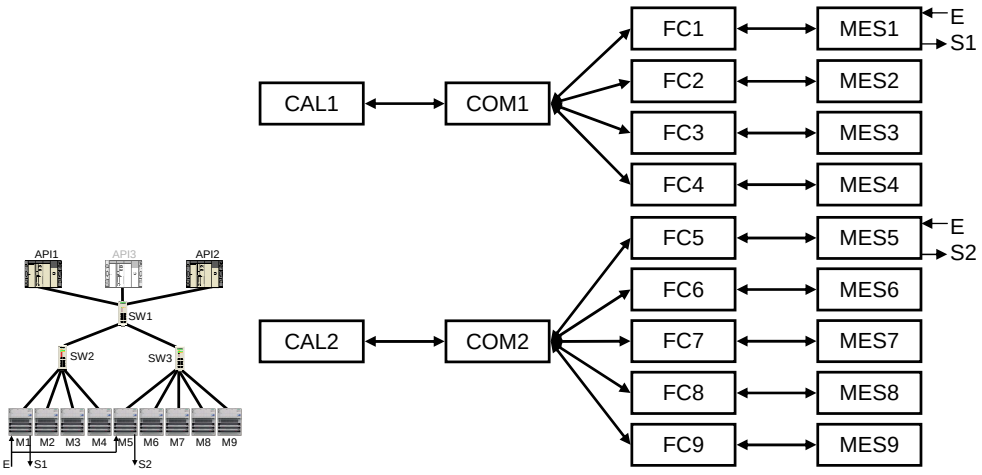


FIG. 5.5 – Cas d'étude 4 et structure de son modèle initial

5.2 Application de la méthode de simplification sur les cas d'étude

5.2.1 Présentation de la structure des modèles abstraits

La structure des modèles abstraits des six cas d'étude est présentée ci-après. Il est évident qu'une simplification de la structure implique une modification des modèles des composants restants, comme exposé au chapitre précédent. Nous n'avons pas souhaité, par souci de concision, présenter les modèles modifiés de composants pour les six cas d'étude ; ces modèles peuvent se déduire aisément de ceux décrits précédemment.

5.2.1.1 Cas d'étude n°1

Comme cela l'a déjà été signalé lors de la présentation de ce cas d'étude, la structure de ce modèle d'AAR est minimale (cf figure 5.9) : un processeur de calcul, une carte de communication, une fonction de communication et un module d'entrées/sorties déportées. Elle ne peut donc pas être simplifiée. Ce modèle n'est pas modifié par la méthode d'abstraction.

5.2.1.2 Cas d'étude n°2 et 3

Les cas d'étude 2 et 3, bien que différents, ont des structures qui se simplifient de la même manière. Quand seuls les composants sur la route des données sont conservés, il ne reste qu'un processeur de calcul, une carte de communication, une fonction de communication et un MES. Ils ont ainsi la même structure simplifiée que le cas d'étude 1 (figure 5.9). Pour le cas d'étude 3, il faut cependant modifier les noms des composants dans le modèle puisque ce sont le processeur de calcul 3, la carte de communication 3, la fonction de communication 10 et le MES 1 qui sont concernés par le temps de réponse étudié.

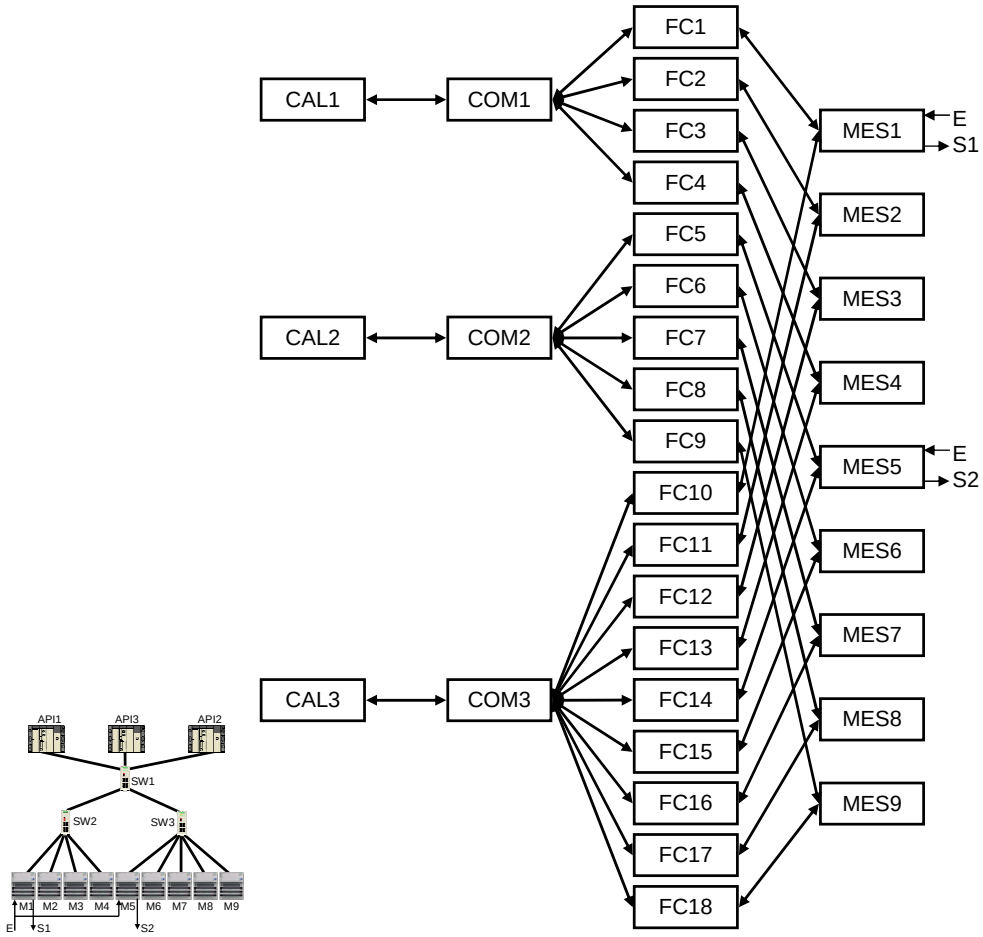


FIG. 5.6 – Cas d'étude 5 et structure de son modèle initial

Il convient de préciser également que, dans le modèle abstrait du cas 2, l'automate modélisant la carte de communication devra prendre en compte les échanges avec les trois MES (MES2, MES3 et MES4) qui ne figurent pas dans le modèle abstrait de l'AAR. De même, dans le cas 3, ce sont huit MES qui ne figurent pas dans le modèle abstrait de l'AAR et dont il faudra tenir compte dans le modèle modifié de COM3 (cf. partie 4.4.2.2).

5.2.1.3 Cas d'étude n°4 et 5

Les cas d'étude 4 et 5 ont également des structures qui se simplifient de la même manière. Cette structure simplifiée est représentée sur la figure 5.10. Cette structure simplifiée se compose de **deux branches totalement indépendantes**, chacune composée d'un processeur de calcul, d'une carte de communication, d'une fonction de communication et d'un MES.

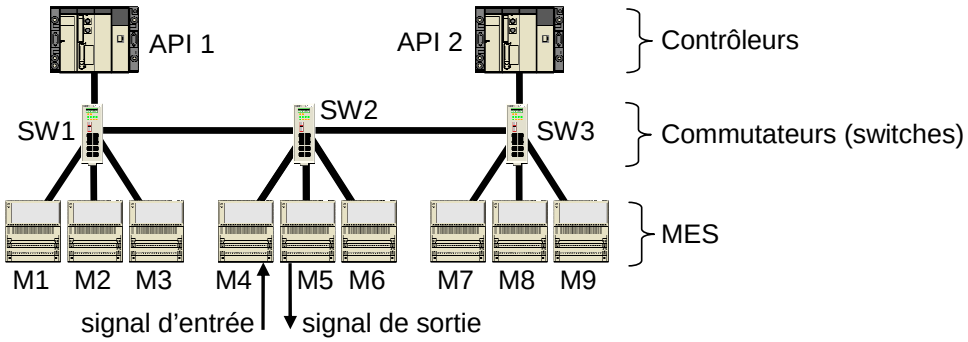


FIG. 5.7 – AAR utilisée pour le cas 6

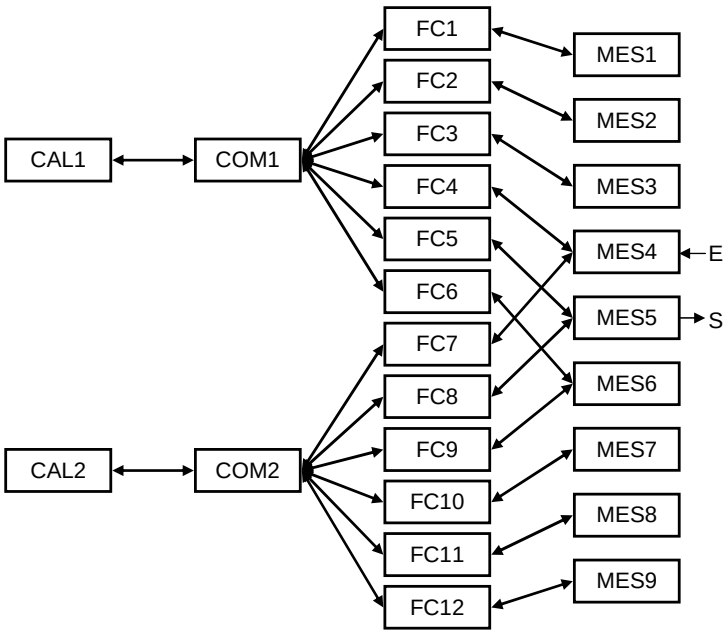


FIG. 5.8 – Structure du modèle initial du cas d'étude 6

5.2.1.4 Cas d'étude n°6

Pour ce dernier cas d'étude, la structure du modèle simplifié, représentée sur la figure 5.11, ne comporte qu'un processeur de calcul et une carte de communication mais dispose de deux fonctions de communication et de deux MES. Il est également bon de remarquer que, contrairement au cas 1 à 4, ces deux MES sont, dans le modèle initial (figure 5.8), partagés par les deux API. Leurs modèles modifiés devront donc prendre en compte l'impact des échanges avec la carte de communication (ici avec COM2) qui n'apparaît plus dans le modèle abstrait de l'AAR, comme indiqué au chapite 4.

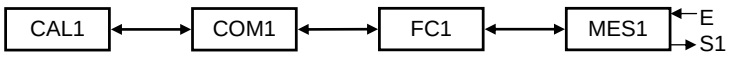


FIG. 5.9 – Structure du modèle abstrait du cas d'étude 1

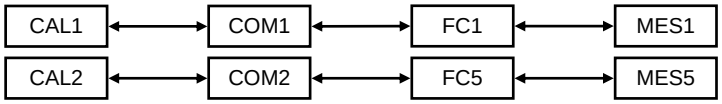


FIG. 5.10 – Structure des modèles abstraits des cas d'étude 4 et 5

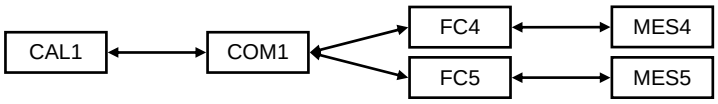


FIG. 5.11 – Structure du modèle abstrait du cas d'étude 6

5.2.2 Conséquence pour l'évaluation de la différence de temps de réponse

Les bornes de la performance temporelle étudiée peuvent être obtenues par *analyse directe*, c'est à dire selon la méthode d'évaluation par preuves itératives proposée au chapitre 2, en utilisant un modèle abstrait de l'AAR. Lorsque la performance temporelle étudiée est une différence de temps de réponse, la structure du modèle abstrait permet également d'évaluer ses bornes par *analyse indirecte*. Cette possibilité est explicitée ci-dessous.

Pour les cas d'étude 4 et 5, l'indépendance des branches de la structure des modèles abstraits incite à réaliser une étude séparée de ces deux branches. En effet, en obtenant les bornes minimale et maximale des temps de réponse de ces deux branches, notées respectivement TR1m et TR1M pour la première branche du modèle (CAL1, COM1, CF1 et MES1) et TR2m et TR2M pour la deuxième branche du modèle (CAL2, COM2, CF5 et MES5), il est possible de calculer les bornes minimale et maximale de la différence des temps de réponse DTRm et DTRM.

En effet, comme indiqué figure 5.12, trois cas sont possibles :
- histogrammes disjoints $TR1M < TR2m$ (cas 1)
- intersection non vide mais n'englobant pas un des histogrammes en entier $TR1m \leq TR2m$ et $TR2m \leq TR1M \leq TR2M$ (cas 2)
- un histogramme inclus dans l'autre $TR1m > TR2m$ et $TR1M < TR2M$ (cas 3)

A partir de ces observations sur les relations entre les bornes des temps de réponse, la valeur maximale de la différence des temps de réponse peut être estimée ainsi :

$$DTRM \leq Max((TR1M - TR2m), (TR2M - TR1m)) \tag{5.1}$$

Pour la valeur minimale de la différence des temps de réponse, il est nécessaire de faire une hypothèse supplémentaire : les distributions représentant les temps de réponse sont continues. Cette hypothèse semble très raisonnable à la vue des histogrammes des temps de réponse obtenus par mesure ou par simulation. Avec cette hypothèse, la valeur minimale de la différence des temps de réponse peut être évaluée ainsi :

Pour les cas 2 et 3

$$dtrM = 0 \tag{5.2}$$

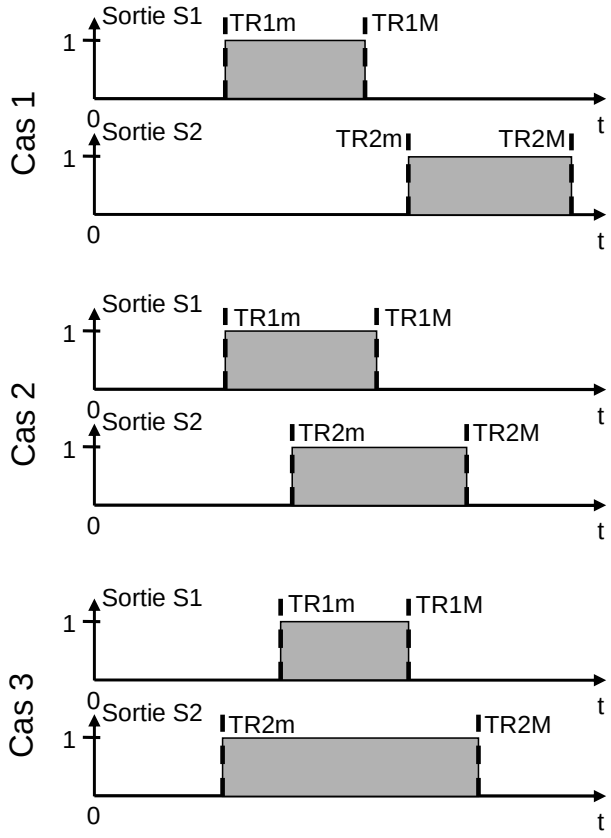


FIG. 5.12 – Position relative des bornes des temps de réponse des deux branches de la figure 5.10

Pour le cas 1

$$DRTm \geq Max((TR1m - TR2M), (TR2m - TR1M)) \tag{5.3}$$

Ce qui peut être résumé en une seule relation :

$$DTRm \geq Max((TR1m - TR2M), (TR2m - TR1M), 0) \tag{5.4}$$

Avec cette deuxième approche d'*analyse indirecte*, les quatre bornes des temps de réponse doivent être trouvées en utilisant la méthode de preuves itératives et l'automate observateur présenté par la figure 3.16. Cela peut sembler coûteux de chercher quatre valeurs quand initialement il n'y en a que deux à trouver mais, comme l'espace d'état à analyser pour étudier les temps de réponse des branches indépendamment l'une de l'autre est bien moins grand que celui nécessaire à l'étude de la différence des temps de réponse de deux branches, au final, le temps de vérification est réduit, comme montré dans la partie 5.5.

5.3 Conditions expérimentales pour la preuve de propriétés

Pour que les résultats obtenus soient comparables, tous les calculs ont évidemment été réalisés sur le même ordinateur. Cet ordinateur est doté d'un processeur Core2duo E6400, possède 4 Go de mémoire vive et fonctionne sous Windows XP professionnel. La version de développement du logiciel UPPAAL (version 4.1.0) a par ailleurs été choisie pour réaliser ces analyses, car elle offre des performances bien supérieures à la version 4.0.7 qui est la dernière version officielle du logiciel.

Les caractéristiques retenues pour juger de l'efficacité de la méthode d'abstraction ou des réglages du logiciel seront toujours la quantité de mémoire consommée et le temps de calcul. Dans la suite de ce chapitre, quand des temps de calcul seront donnés, ils correspondront toujours au temps nécessaire pour vérifier les trois propriétés P4, P5 et P6, présentée dans le partie 4.1.3 et rappelées ci-dessous, pour la dernière itération de la méthode d'évaluation.

$$P4 : E <> OBS.SIT_{finale} \wedge SIGsor == 1 \wedge horloge < \tau \tag{5.5}$$

$$P5 : E <> OBS.SIT_{finale} \wedge SIGsor == 1 \wedge horloge = \tau \tag{5.6}$$

$$P6 : E <> OBS.SIT_{finale} \wedge SIGsor == 1 \wedge horloge > \tau \tag{5.7}$$

Ce choix a été réalisé car il n'est pas possible de juger de l'intérêt de modèles (modèle abstrait/modèle détaillé) ou de techniques d'analyse (directe/indirecte) sur l'ensemble de la recherche itérative car le nombre d'itérations nécessaires pour converger vers une des bornes de la performance étudiée et donc le temps de calcul dépend des bornes initiales choisies. D'autre part, cette dernière itération de la méthode sera toujours réalisée, quelles que soient les bornes initiales choisies, puisque c'est elle qui met fin à la recherche.

5.3.1 Choix du mode de représentation et d'analyse de l'espace d'états

Avant de réaliser les essais sur les différents cas d'étude, il est nécessaire de définir les réglages du logiciel UPPAAL qui offre de nombreuses possibilités de paramétrage. Tout d'abord UPPAAL propose quatre **représentations internes** possibles de l'espace d'états d'un modèle formel temporisé : deux sans approximation et deux avec approximation. La représentation standard proposée par UPPAAL est basée sur l'utilisation de DBM (Difference Bound Matrices) ([BBD+02]), tandis qu'une autre représentation, basée sur une structure de données compacte (CDS) ([LLPY97]), permet de réduire la consommation de mémoire mais au détriment de l'augmentation des temps de calcul. Les deux dernières représentations utilisent des sur-approximations ([Bal96]) et des sous-approximations ([WL93]) de l'espace d'état. Dans le contexte de ces travaux, une sous-approximation de l'espace d'état peut amener à ne pas détecter certaines évolutions qui pourraient étendre la plage de valeurs des performances étudiées. Cette représentation n'est donc absolument pas utilisable dans notre cas et ne sera donc plus abordée dans la suite de ce chapitre.

Uppaal permet également de **réduire l'espace d'état conservé en mémoire** puisqu'il n'est pas forcément nécessaire de conserver tous les états pour vérifier une propriété, cette suppression permettant de réduire la consommation de mémoire et ainsi d'éventuellement mener à son terme un calcul qui n'aurait pas abouti sans cela. Il y a trois possibilités de réglage pour ce paramètre : sans réduction de l'espace d'état en mémoire, avec une réduction "conservative" (l'outil évite de conserver en mémoire les états "committed") ou avec une réduction "agressive" (l'outil essaie de ne créer qu'un état par pas de calcul). Par défaut, c'est la réduction "conservative" de l'espace d'états qui est utilisée.

Enfin, Uppaal propose différents types de recherche : en largeur, en profondeur, en profondeur avec un choix aléatoire de la branche à explorer et enfin une approche appelée "Closest to target first" et que l'on peut considérer comme "au plus proche de la propriété à valider". Une recherche en largeur (réglage de base dans Uppaal) est généralement la plus efficace quand l'ensemble de l'espace d'état doit être exploré. Au contraire, une recherche en profondeur peut permettre de trouver plus rapidement un contre-exemple qu'une recherche en largeur. La recherche en profondeur aléatoire est sensée être encore plus rapide dans ce cas. Enfin le dernier type de recherche est apparu sur cette version de développement d'Uppaal mais n'est pas implanté dans la version officielle du logiciel. Cette recherche utilise des heuristiques basées sur les situations et les variables contenues dans la propriété à vérifier afin de donner la priorité lors de la recherche aux situations les plus proches de la situation cible et aux transitions susceptibles d'amener les variables dans la configuration recherchée.

Pour savoir quels étaient les réglages optimaux à utiliser pour les cas d'étude et comme il n'a pas été possible de tester toutes les possibilités de représentation, de réduction de l'espace d'états et d'ordre de recherche proposés par Uppaal pour tous les cas d'étude, une étude comparative a été menée sur un seul modèle, celui correspondant au cas d'étude 4 après application de la méthode d'abstraction. Ce modèle a été choisi car il permet de bien mettre en évidence les différences de réglage (les temps de calculs sont suffisamment longs) sans qu'il soit trop long à vérifier (tous les calculs aboutissent).

Les paramètres de réglage sont donc :

- la représentation interne de l'espace d'états (3 possibilités : DBM, structure de données compactes et sur-approximation)
- la réduction de l'espace d'états (3 possibilités : sans réduction, réduction conservative, réduction agressive)
- l'ordre de recherche (4 possibilités : largeur, profondeur, profondeur aléatoire et "au plus proche de la propriété à valider")

Pour tester toutes les combinaisons de paramètres il faudrait faire 36 essais. Par souci d'efficacité, il a été décidé de ne faire varier qu'un des paramètres de réglage en gardant les deux autres fixes, et ceci pour les trois réglages possibles.

5.3.2 Influence de la technique de réduction de l'espace d'états

Le tableau 5.3 reprend les valeurs obtenues en gardant les réglages de base pour la représentation de l'espace d'états (DBM) et l'ordre de recherche (en largeur) tandis que l'impact de la réduction de l'espace d'état est analysé.

TAB. 5.3 – Comparaison des possibilités de réduction de l'espace d'état

	temps de calcul	mémoire consommée
sans réduction	4'01"9	30,124 Mo
réduction conservative	3'59"6	28,868 Mo
réduction agressive	4'43"3	24,292 Mo

Il est possible de voir que le réglage de base d'UPPAAL (réduction conservative) semble le plus efficace, du moins pour le temps de calcul. La réduction agressive réduit bien la consommation de mémoire (de 16%) mais augmente (de 19%) le temps de calcul. Cette solution ne sera pas choisie par défaut mais sera néanmoins essayée si la mémoire est saturée lors d'un calcul.

5.3.3 Influence de l'ordre de recherche

La deuxième possibilité de réglage est l'ordre de la recherche dans l'espace d'états. Les résultats obtenus avec une représentation de l'espace d'états réalisé par DBM et une réduction de l'espace d'états conservative sont donnés dans le tableau 5.4.

TAB. 5.4 – Comparaison des ordres de recherche

	temps de calcul	mémoire consommée
en largeur	3'59"6	28,868 Mo
en profondeur	2h54'23"7	77,388 Mo
en profondeur avec choix aléatoire	1h48'53"6	79,480 Mo
au plus près de la propriété	3'54"8	29,036 Mo

Comme cela était annoncé par UPPAAL[12], la recherche en profondeur a été beaucoup moins efficace que la recherche en largeur pour nos analyses qui ne nécessitent pas de rechercher un contre-exemple mais bien d'explorer l'ensemble de l'espace d'état. La méthode au plus près de la propriété a été légèrement plus efficace que la méthode de recherche en largeur (gain de 2% en temps de calcul et consommation mémoire identique). Cette possibilité n'étant pas encore implantée dans les versions stables du logiciel, il a été choisi de ne pas l'utiliser pour ne pas courir le risque d'avoir des erreurs pour seulement 2% de gain en temps. La recherche en largeur (le réglage par défaut d'UPPAAL) sera donc utilisée pour les différents cas d'étude.

5.3.4 Influence de la représentation de l'espace d'états

Le dernier réglage possible est le choix de la représentation de l'espace d'états. Pour cela, une étude comparative a été faite en gardant comme réglages imposés une recherche en largeur et une réduction de l'espace d'état conservative. Les résultats sont regroupés dans le tableau 5.5.

[12] dans l'aide en ligne disponible sur le site www.uppaal.com

TAB. 5.5 – Comparaison des choix de représentation de l'espace d'états

	temps de calcul	mémoire consommée
DBM	3'59"6	28,868 Mo
CDS	6'20"0	14,600 Mo
sur-approximation	1"1	5,940 Mo

Pour les méthodes de représentation de l'espace d'états de manière exacte (DBM et CDS), il y a clairement un choix à faire entre rapidité et consommation de mémoire. La représentation par CDS permet de diviser par deux la consommation de mémoire par rapport à la représentation par DBM mais au prix d'un temps de calcul allongé de près de 60%. La représentation par DBM sera donc privilégiée tant qu'elle permettra aux calculs d'aboutir. Si ce n'est pas le cas, la représentation par CDS sera essayée. L'efficacité de la méthode de sur-approximation de l'espace d'état est évidente (division du temps de calcul par 240 et de la mémoire consommée par 4,9) mais l'exactitude des résultats fournis devra être évaluée. Ce point sera discuté dans la partie 5.4.

5.3.5 Définition des configurations d'analyse des cas d'étude

Il a donc été choisi de réaliser, pour chaque cas d'étude, quatre analyses. Le modèle initial et le modèle abstrait (obtenu à l'aide de la méthode présentée au chapitre 4) correspondant à chaque cas d'étude seront chacun analysés avec deux représentations différentes de l'espace d'états, l'une sans approximation (en utilisant des DBM), l'autre avec une sur-approximation de l'espace d'états. Ces quatre configurations d'analyse sont données dans le tableau 5.6. Pour toutes ces configurations, l'analyse est faite avec réduction de l'espace d'état conservative et recherche en largeur.

TAB. 5.6 – Configurations des analyses

	modèle initial	modèle abstrait
sans approximation	configuration 1	configuration 3
avec sur-approximation	configuration 2	configuration 4

5.4 Comparaison des valeurs des bornes obtenues avec les quatre configurations

5.4.1 Présentation des résultats

Avant de quantifier les gains apportés par la méthode d'abstraction proposée au chapitre 4, il est indispensable de s'assurer que cette méthode ne dégrade pas les valeurs des bornes obtenues. Les différentes valeurs des bornes des performances temporelles étudiées, obtenues avec les quatre configurations d'analyse, sont regroupées dans les tableaux 5.7 et 5.8. Elles sont notées TRm, TRM, DTRm et DTRM et représentent respectivement les

bornes minimale et maximale d'un temps de réponse et les bornes minimale et maximale d'une différence de temps de réponse. Quand aucune valeur n'est indiquée (notation KO), cela correspond à un calcul qui n'a pas pu aboutir à cause de la saturation de la mémoire disponible. Il convient de remarquer dès à présent que seule l'utilisation d'un modèle abstrait (configurations 3 et 4) permet d'obtenir des résultats pour tous les cas d'étude. Ceci constitue une première validation de l'intérêt de cette proposition.

TAB. 5.7 – Valeurs des bornes des performances temporelles étudiées

	cas 1	cas 2	cas 3
configuration 1	$TRm = 10,70\ ms$ $TRM = 20,70\ ms$	$TRm = 10,70\ ms$ $TRM = 20,70\ ms$	$TRm = 50,70\ ms$ $TRM = 100,70\ ms$
configuration 2	$TRm \geq 10,70\ ms$ $TRM \leq 20,70\ ms$	$TRm \geq 10,70\ ms$ $TRM \leq 20,70\ ms$	$TRm \geq 50,70\ ms$ $TRM \leq 100,70\ ms$
configuration 3	$TRm = 10,70\ ms$ $TRM = 20,70\ ms$	$TRm = 10,70\ ms$ $TRM = 20,70\ ms$	$TRm = 50,70\ ms$ $TRM = 100,70\ ms$
configuration 4	$TRm \geq 10,70\ ms$ $TRM \leq 20,70\ ms$	$TRm \geq 10,70\ ms$ $TRM \leq 20,70\ ms$	$TRm \geq 50,70\ ms$ $TRM \leq 100,70\ ms$

	cas 4	cas 5	cas 6
configuration 1	$DTRm = 00,00\ ms$ $DTRM = 10,00\ ms$	KO KO	KO KO
configuration 2	$DTRm \geq 00,00\ ms$ $DTRM \leq 10,00\ ms$	KO KO	$TRm \geq 10,25\ ms$ $TRM \leq 21,65\ ms$
configuration 3	$DTRm = 00,00\ ms$ $DTRM = 10,00\ ms$	$DTRm = 00,00\ ms$ $DTRM = 21,40\ ms$	$TRm = 10,25\ ms$ $TRM = 21,65\ ms$
configuration 4	$DTRm \geq 00,00\ ms$ $DTRM \leq 10,00\ ms$	$DTRm \geq 00,00\ ms$ $DTRM \leq 21,40\ ms$	$TRm \geq 10,25\ ms$ $TRM \leq 21,65\ ms$

5.4.2 Discussion

5.4.2.1 Sur-approximation proposée par Uppaal

Les résultats obtenus pour les configurations 2 et 4 reposent sur la représentation de l'espace d'états par sur-approximation proposée par UPPAAL. Cette représentation ne permet pas de garantir qu'une évolution observée soit réellement existante et ne provient pas en réalité de la sur-approximation. Seule une réponse négative à l'existence d'une évolution peut être considérée comme sûre. En conséquence, les valeurs obtenues par preuves itératives ne peuvent pas être considérées strictement comme des bornes mais doivent être vues comme des majorants ou minorants. Ceci explique l'utilisation des symboles $\leq$ et $\geq$ dans les tableaux.

TAB. 5.8 – Valeurs des DTR obtenues par la méthode indirecte (déduites des TR élémentaires)

	cas 4	cas 5
configuration 3	$TR1m = 10,70\ ms$	$TR1m = 10,00\ ms$
	$TR1M = 20,70\ ms$	$TR1M = 21,40\ ms$
	$TR2m = 10,70\ ms$	$TR2m = 10,00\ ms$
	$TR2M = 20,70\ ms$	$TR2M = 31,40\ ms$
	$DTRm = 0,00\ ms$	$DTRm = 0,00\ ms$
	$DTRM = 10,00\ ms$	$DTRM = 21,40\ ms$
configuration 4	$TR1m \geq 10,70\ ms$	$TR1m \geq 10,00\ ms$
	$TR1M \leq 20,70\ ms$	$TR1M \leq 21,40\ ms$
	$TR2m \geq 10,70\ ms$	$TR2m \geq 10,00\ ms$
	$TR2M \leq 20,70\ ms$	$TR2M \leq 31,40\ ms$
	$DTRm \geq 0,00\ ms$	$DTRm \geq 0,00\ ms$
	$DTRM \leq 10,00\ ms$	$DTRM \leq 21,40\ ms$

5.4.2.2 Modèles abstraits de l'AAR

Dans le cas de l'analyse d'un modèle abstrait sans utilisation de la sur-approximation (configuration 3), les valeurs obtenues par preuves itératives correspondent bien aux bornes des performances temporelles du modèle abstrait. Les relations figurant dans le tableau 5.7, pour cette configuration, sont donc des égalités, contrairement aux configurations 2 et 4.

5.4.2.3 Comparaison des résultats fournis par les quatre configurations d'analyse

La comparaison des résultats obtenus pour les cas d'étude 1 à 4 montre que les quatre configurations d'analyse conduisent à des valeurs identiques. Ceci montre que les techniques visant à simplifier le modèle formel à analyser – sur-approximation proposée par UPPAAL ou méthode globale d'abstraction exposée au chapitre 4 – permettent d'obtenir des résultats dignes de confiance, dans ces cas d'étude. Cette conclusion sera extrapolée pour les cas d'étude pour lesquels l'analyse d'un modèle détaillé, sans sur-approximation, est impossible.

5.4.2.4 Méthode indirecte

Enfin, lorsque la différence de temps de réponse est évalué par analyse indirecte (cd. partie 5.2.2), on constate, en comparant les tableaux 5.7 et 5.8, que les résultats obtenus sont identiques à ceux obtenus par évaluation directe. La décomposition du modèle abstrait en deux modèles partiels et l'utilisation d'un automate observateur plus simple n'apportent pas, dans les cas étudiés, de dégradation des valeurs de bornes obtenues.

5.5 Quantification des gains dus à l'utilisation d'un modèle abstrait

L'objectif de cette partie est de comparer les temps de calcul (tableau 5.9) et la consommation de mémoire – pic de consommation – (tableau 5.10) pour les différents cas d'étude en fonction de la configuration d'analyse retenue, lors d'une analyse directe. Il sera ainsi possible de déterminer l'apport de la méthode d'abstraction. Comme il a été montré précédemment que la sur-approximation proposée par Uppaal ainsi que l'utilisation d'un modèle abstrait ne dégradaient pas les valeurs des bornes obtenues, la comparaison de ces grandeurs pour les différents cas d'étude pourra se faire sans arrière-pensées.

Tab. 5.9 – Temps de calcul

	cas 1	cas 2	cas 3	cas 4	cas 5	cas 6
configuration 1	0"1	0"2	0"4	5h06'56"8	KO	KO
configuration 2	0"1	0"2	0"3	8"8	KO	34"6
configuration 3 (directe)	0"1	0"1	0"2	4'59"6	13'25"8	0"3
configuration 3 (indirecte)				0"2	0"2	
configuration 4	0"1	0"1	0"1	1"1	2"3	0"2

Tab. 5.10 – Consommation de mémoire vive

	cas 1	cas 2	cas 3	cas 4	cas 5	cas 6
configuration 1	4,464 Mo	5,168 Mo	7,096 Mo	1081,680 Mo	KO	KO
configuration 2	4,440 Mo	5,124 Mo	6,704 Mo	10,588 Mo	KO	43,240 Mo
configuration 3 (directe)	4,464 Mo	4,648 Mo	4,680 Mo	28,868 Mo	73,708 Mo	5,024 Mo
configuration 3 (indirecte)				4,648 Mo	4,648 Mo	
configuration 4	4,440 Mo	4,632 Mo	4,652 Mo	5,940 Mo	6,904 Mo	4,932 Mo

A la lecture de ces tableaux, il est évident que la méthode d'abstraction développée n'est pas utile pour les cas d'étude 1, 2 et 3. Ces trois cas sont suffisamment simples pour pouvoir être traités directement sans même utiliser les possibilités d'approximation offertes par Uppaal. Dès que les modèles initiaux comportent plus de deux APIs (cas 4, 5 et 6), les temps de calcul et la consommation de mémoire augmentent grandement et, dans certains cas, les calculs ne peuvent pas aboutir, comme indiqué précédemment.

Il convient de remarquer dès à présent que les cas 4 et 6 (représentés respectivement par les figures 5.5 et 5.8), montrent que deux modèles assez proches (même nombre d'API et de MES) peuvent donner des résultats bien différents, simplement à cause de quelques changements, comme ici le partage de certains MES.

Pour le cas 4, les calculs peuvent être menés à bout sur le modèle détaillé, sans utiliser de sur-approximation de l'espace d'états (configuration 1) mais le temps de calcul est alors très long : un peu plus de 5 h, pour, rappelons-le, uniquement une itération. La

représentation de l'espace d'états par sur-approximation, proposée par UPPAAL, est ici très efficace puisque le même calcul ne prend plus qu'environ 9 s (configuration 2). Cette valeur est inférieure à celle nécessaire pour analyser le modèle abstrait, sans approximation, par évaluation directe des bornes (configuration 3) : 5 min. Cependant, lorsque les bornes sont évaluées de façon indirecte, ce temps tombe à 0,2 s. Le gain apporté est d'autant plus intéressant que pour l'analyse complète, il faut une dizaine d'itérations pour trouver une borne (valeur dépendant des valeurs initiales des limites de la recherche par dichotomie) ce qui amène à un temps de calcul global de 2 s avec l'analyse indirecte sans approximation contre 1 min 30 s pour l'analyse directe et la sur-approximation. Ceci montre clairement l'intérêt de l'évaluation des bornes par analyse indirecte, rendue possible uniquement grâce à la méthode d'abstraction proposée.

Pour le cas 6, il n'est pas possible d'obtenir de résultat avec la représentation non approximée de l'espace d'état (configuration 1). Il est cependant possible d'aboutir à un résultat si on modifie la représentation de l'espace d'états, en utilisant une représentation par Compact Data Structure (CDS), qui est également une représentation non approximée. Dans ce cas, l'itération dure 179 h et consomme 907 Mo de mémoire vive. Ces valeurs, et notamment la première, sont incompatibles (ou peu compatibles) avec un processus industriel de conception, ce qui explique que nous ayons indiqué que le calcul était impossible (KO) avec un modèle ni abstrait, ni approximé. L'observation des trois lignes suivantes montre clairement que l'analyse du modèle abstrait conduit à des temps beaucoup plus courts (0,3 s contre 35 s) que l'analyse par sur-approximation proposée par UPPAAL.

Le cas d'étude 5 étant le plus complexe à traiter, les résultats obtenus sont particulièrement intéressants pour estimer l'efficacité de la méthode d'abstraction sur des architectures d'automatisation de type industriel. Dans ce cas, UPPAAL est totalement incapable de faire aboutir le calcul sur le modèle initial de l'AAR, et ceci même en utilisant la configuration la moins gourmande en mémoire, à savoir une représentation par sur-approximation de l'espace d'états et une réduction agressive de l'espace d'états. **La méthode d'abstraction prend ici tout son sens puisqu'elle permet :**

- **d'apporter une réponse au problème,**
- **et ceci assez rapidement (13 minutes et 26 secondes).**

Pour ce cas, l'évaluation des bornes ne peut donc se faire que sur un modèle abstrait en utilisant :

- la méthode d'évaluation par analyse directe et aucune sur-approximation (configuration 3) ; une itération dure alors 13 min et 26 s,
- la méthode d'évaluation par analyse directe et la sur-approximation (configuration 4) ; une itération dure alors 2,3 s,
- la méthode d'évaluation par analyse indirecte et aucune sur-approximation (configuration 3) ; une itération dure alors 0,2 s.

Ces valeurs de temps de calcul montrent clairement l'intérêt de l'évaluation par analyse indirecte, plus performante que la sur-approximation.

5.6 Confrontation au réel

5.6.1 Présentation du dispositif de mesure des performances temporelles

Pour comparer les valeurs des bornes obtenues par preuves itératives aux valeurs observées sur une AAR réelle, des mesures ont été réalisées à l'aide d'une plateforme de mesure, sur la base du cas d'étude 6. Cette plateforme pour mesurer les performances temporelles de systèmes automatisés qui peuvent être modéliser comme des SED (Systèmes à Evènements Discrets), dénommée PRISME[13], a été développée au LURPA il y a quelques années et comporte trois éléments principaux :

- un générateur de signaux carrés qui permet de délivrer des signaux de période variable pour lesquels les fronts montants et descendants peuvent être considérés comme des évènements ;
- un analyseur logique pour collecter des diagrammes temporels qui contiennent aussi bien l'ordre des occurrences d'évènements que la durée entre ces évènements ;
- un ordinateur temps réel qui coordonne la génération des évènements et la collecte des diagrammes temporels.

Pour mesurer les performances temporelles (temps de réponse ou différence de temps de réponse) d'une AAR (figure 5.13), le générateur de signaux est connecté à l'entrée considérée et l'analyseur logique collecte les diagrammes temporels de l'entrée et de (des) sortie(s) à partir desquelles le temps de réponse ou la différence des temps de réponse sont définis.

Comme nous l'avons vu précédemment, les performances temporelles étudiées sont caractérisées par des distributions de valeurs, ce qui impose de réaliser un nombre important de mesures. Ceci n'est pas un problème grâce au dispositif PRISME puisqu'il permet d'automatiser les mesures. Il est d'autre part indispensable que ces mesures représentent aussi fidèlement que possible le comportement réel de l'AAR. Pour cela, **il importe d'éviter toute synchronisation entre le signal d'entrée et le cycle de lecture des entrées contrôlé par la carte de communication**.

Pour éviter ce phénomène, la fréquence du signal d'entrée a été choisie afin de ne pas être un multiple de la fréquence de l'IOscanning et afin que les occurrences de l'évènement d'entrée soient réparties aussi uniformément que possible sur toute la durée de ce cycle (cf [DRF+07]). Dans le meilleur des cas, la distribution de la durée entre le début d'un cycle d'IOscanning et l'occurrence de l'évènement d'entrée (notée δ sur la figure 5.14) serait uniforme.

Dans ce but, la période du signal d'entrée $T_{entrée}$ a été définie en fonction de la période de l'IOscanning$T_{IOscanning}$ par la formule :

$$T_{input} = k * T_{IOscanning} + \frac{T_{IOscanning}}{100} \tag{5.8}$$

où le paramètre k est un entier qui doit être suffisamment grand pour permettre la génération du signal de sortie avant l'émission d'un nouvel évènement d'entrée tout en restant

[13] Brevet français # 01 110 933

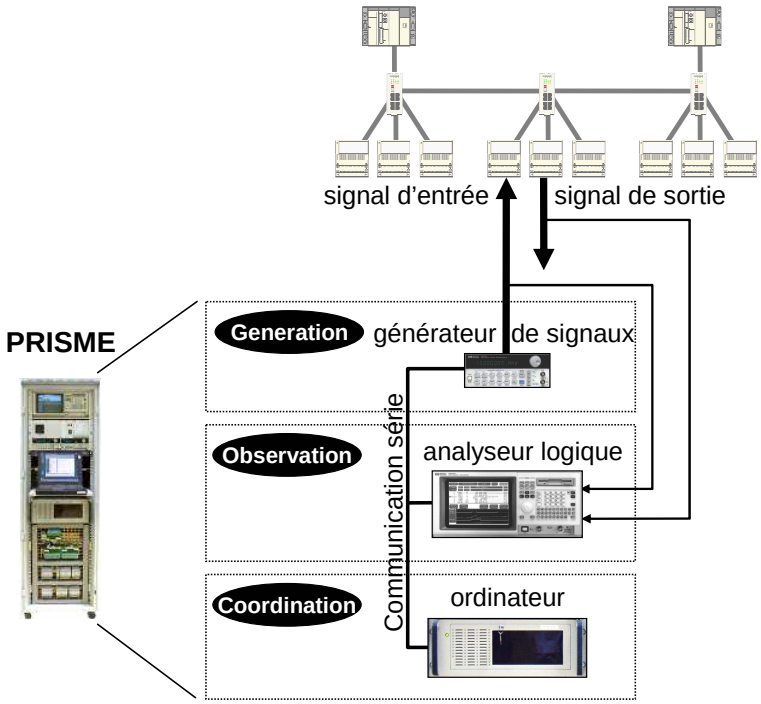


FIG. 5.13 – Dispositif expérimental pour la mesure de performances temporelles

suffisamment petit pour que la durée des mesures ne soit pas trop longue. Le second terme de l'équation est proportionnel au centième de la période d'IOscanning, afin de générer un décalage systématique de l'émission de l'évènement d'entrée par rapport au cycle de l'IOscanning. Pour ces mesures, comme la durée du cycle d'IOscanning était de 10 ms, il a été choisi de prendre k égal à 10, ce qui implique que la période du signal d'entrée soit de 100,1 ms.

La figure 5.15 présente le résultat de 10 000 mesures du temps de réponse du cas 6. **Compte tenu du grand nombre de mesures, les valeurs minimale et maximale de cette distribution seront considérées comme les bornes inférieure et supérieure du temps de réponse étudié.**

5.6.2 Comparaison des mesures aux valeurs obtenues sur le modèle

Les valeurs obtenues par mesure, 10,65 ms pour la borne inférieure et 22,25 ms pour la borne supérieure, sont à comparer avec les valeurs obtenues dans la partie 5.4 qui sont de 10,25 ms et 21,65 ms. Si les valeurs obtenues avec la modélisation retenue de l'AAR sont proches des valeurs expérimentales, elles ne sont pas entièrement satisfaisantes car elles devraient encadrer les bornes mesurées, ce qui n'est pas le cas. En effet, la modélisation retenue est censée englober l'ensemble des comportement pour permettre la validation du comportement de l'AAR et donc amener à avoir une borne inférieure plus base que la valeur minimale observée et une borne supérieure plus haute que la plus grande valeur mesurée.

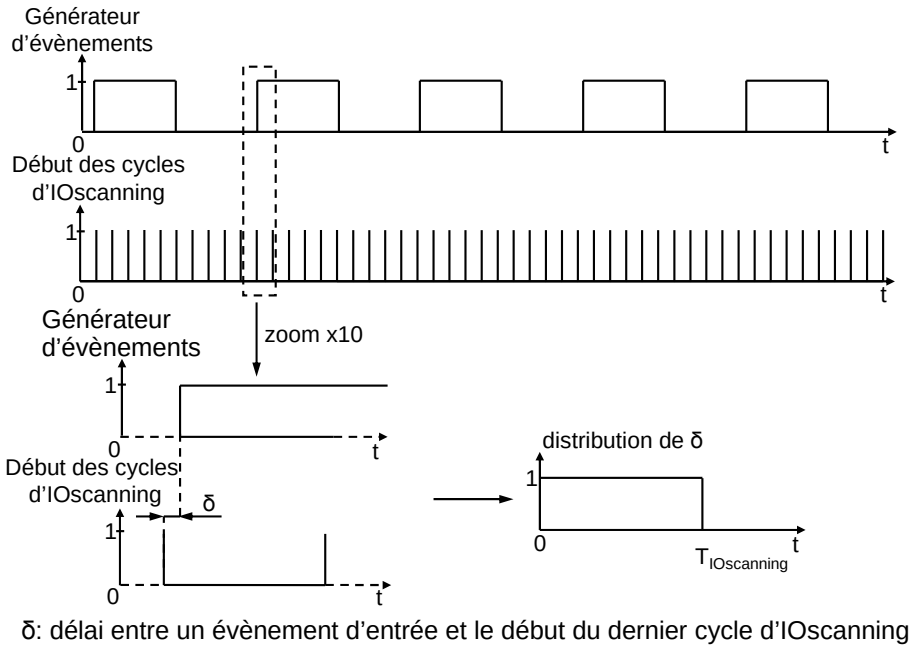


FIG. 5.14 – Chronogramme comparant la génération d'évènements d'entrée au cycle d'IOscanning

Cette inexactitude observée dans les valeurs obtenues a pu être corrigée en analysant les conditions expérimentales. En effet, les modèles formels utilisés pour la vérification ont été configurés avec les valeurs données par le logiciel de programmation des API PL7pro. Ce logiciel indiquait un temps de cycle du processeur de calcul compris entre 2 et 3 ms ainsi qu'un cycle **constant** d'IOscanning de 10 ms. C'est cette dernière valeur qui était erronée. Le cycle d'IOscanning était bien configuré pour être de 10 ms, mais dans la réalité, ce cycle n'est pas absolument constant. A l'aide du logiciel Wireshark ([Com07]), le temps de cycle réel de l'IOscanning a pu être mesuré (en mesurant le délai entre deux émissions successives de la première requête du cycle de scrutation) ; ces mesures ont fourni les valeurs de 9,24 ms et 10,74 ms comme bornes inférieure et supérieure de ce cycle.

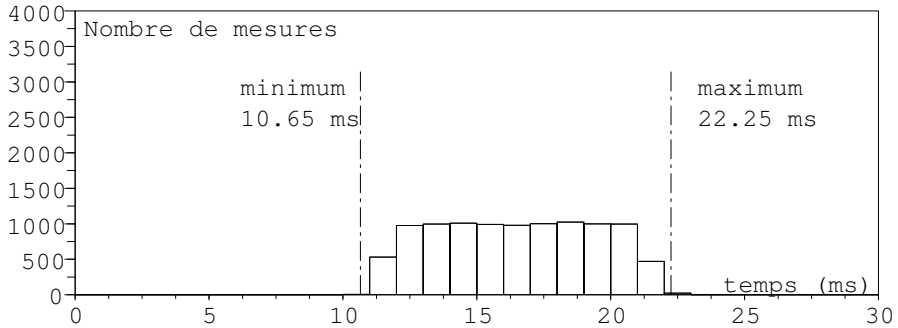


FIG. 5.15 – Histogramme d'un temps de réponse obtenu par mesure sur une AAR réelle

5.6.3 Correction des modèles

Pour prendre en compte cette variation de la durée du cycle d'IOscanning, il a été nécessaire de légèrement modifier le modèle de la carte de communication COM afin de représenter cette variation de durée. Ceci est illustré dans le cas du modèle modifié de carte de communication (en relation avec N MES dont deux seulement demeurent dans le modèle simplifié de l'AAR) par la figure 5.16). Les modifications portent sur la situation COM11 dont l'invariant devient $COMhor \leq COMtcmax$ et sur la transition reliant les situations COM11 et COM3 dont la garde devient $COMhor \geq COMtcmin$ avec COMtcmin = 9,24 ms et COMtcmax = 10,74 ms.

Les valeurs des bornes du temps de réponse de l'AAR obtenues avec ce modèle modifié sont 9,49 ms et 23,13 ms. Ces valeurs sont plus cohérentes car elles englobent bien les bornes des mesures et en plus, elles en sont assez proches avec environ 11% d'écart pour la borne inférieure et seulement 4% pour la borne supérieure.

Cette confrontation au réel montre clairement l'exactitude de notre modélisation, même abstraite, ainsi que l'intérêt de la méthode de détermination des performances temporelles par preuves itératives.

5.7 Conclusion

Ce chapitre a permis de valider expérimentalement nos propositions. La méthode globale d'abstraction, proposée au chapitre 4, ne dégrade pas les valeurs des bornes et améliore nettement les possibilités de passage à l'échelle. Cette contribution permet le traitement de cas impossible à résoudre précédemment, même en utilisant les techniques de sur-appromation existantes. De plus, lorsque la concurrence entre cette méthode d'abstraction et les techniques de sur-approximation existe, elle se révèle plus performante.

D'autre part, la confrontation des résultats obtenus par preuves itératives sur un modèle formel aux mesures réalisées sur une AAR réelle a montré le bien-fondé de notre proposition pour l'évaluation des bornes des performances temporelles.

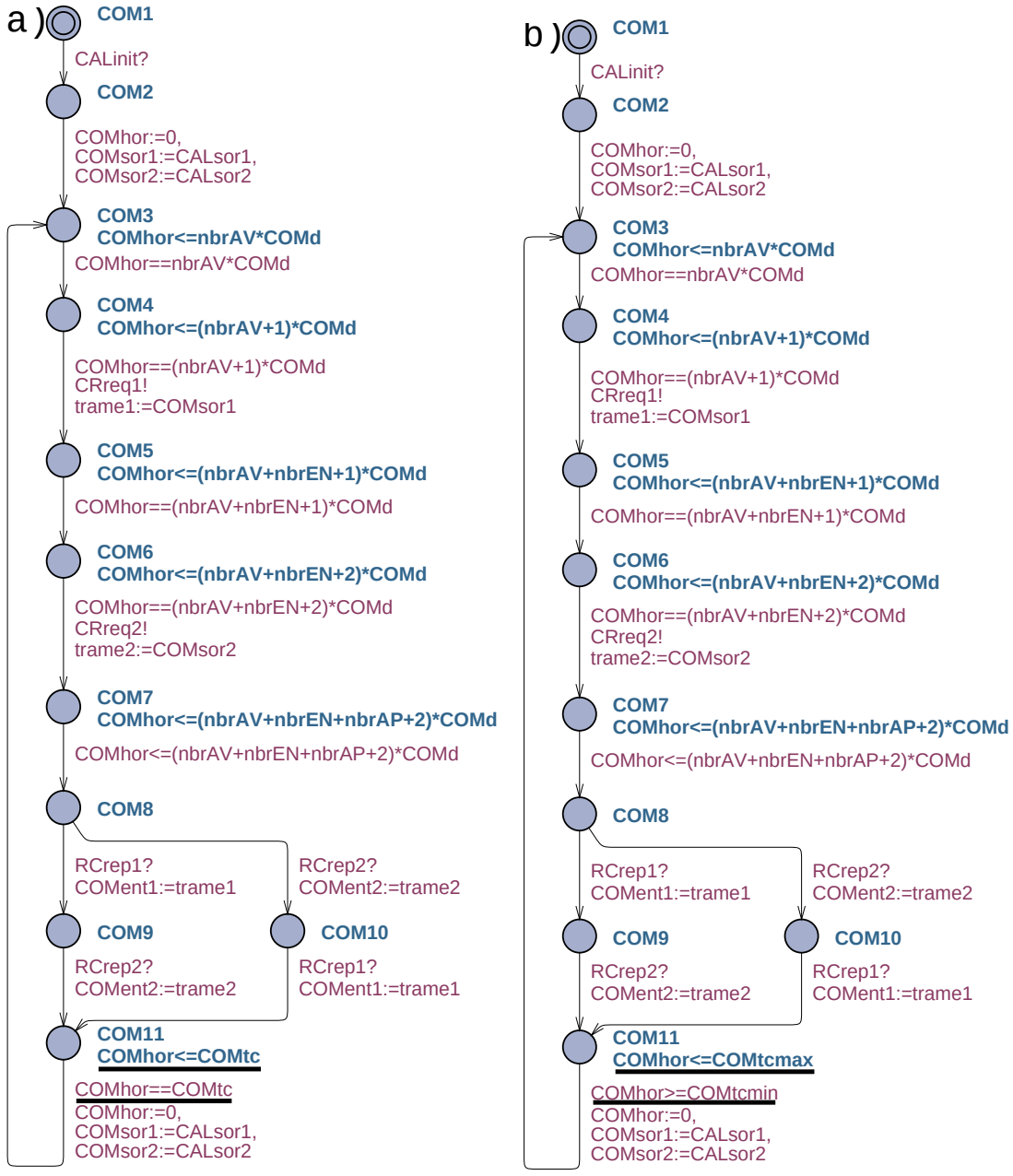


FIG. 5.16 – Modèles d'une carte de communication à temps de cycle constant (a), à temps de cycle variable (b)

Conclusions et Perspectives

Au cours de ce mémoire, nous avons exposé les contributions développées lors de nos recherches doctorales. L'ensemble de ces contributions constitue une méthode originale d'évaluation des bornes des performances temporelles des architectures d'automatisation étudiées, méthode qui garantit l'exhaustivité de l'analyse, la précision des résultats et autorise le passage à l'échelle.

Notre apport théorique majeur, de notre point de vue, est une proposition de recherche des bornes d'une distribution de valeurs par preuves itératives de propriétés logiques définies au moyen d'un automate temporisé observateur paramétré. La mise en oeuvre de cette proposition pour des architectures d'automatisation de taille non triviale nous a conduit tout d'abord à développer des modèles formels génériques des composants d'architectures, qui permettent la construction par instanciation de modèles d'architectures particulières, puis une méthode globale d'abstraction de ces derniers modèles.

L'ensemble de ces propositions a été validé sur six cas d'étude ainsi que par confrontation au réel. Ces expériences ont montré en particulier que les abstractions proposées ne nuisent pas à la précision des valeurs de bornes obtenues et améliorent très nettement les possibilités de passage à l'échelle.

Au terme de ces recherches, plusieurs perspectives de travaux peuvent être envisagées pour le court ou le moyen terme.

Il conviendrait en premier lieu d'étendre nos propositions à la troisième performance temporelle présentée au chapitre 1 : durée minimale d'un signal d'entrée observable. Cette extension nous paraît relativement aisée, car elle consiste principalement à définir un nouvel automate observateur.

Une deuxième perspective à court terme est l'automatisation complète de la méthode d'évaluation des bornes des performances temporelles. Pour ce faire, il conviendra d'ajouter aux outils logiciels développés dans le cadre de ces travaux, pour la recherche des bornes par preuves itératives ainsi que pour la simplification de la structure du modèle formel d'architecture, un outil de construction de ce modèle formel, à partir des modèles génériques de ses composants et d'une définition informelle de l'architecture considérée.

Enfin, pour ce qui concerne le court terme, la validation des aptitudes au passage à l'échelle gagnerait à être complétée par le traitement de cas fournis par des partenaires industriels, même si les derniers cas d'étude traités au chapitre 5 sont tout à fait représentatifs de certaines architectures de sites industriels.

A plus long terme, il serait intéressant de lever une partie des restrictions faites au début de ces travaux, en les étendant à d'autres classes d'architectures, basées sur

d'autres solutions que Modbus TCP/IP et/ou intégrant d'autres flux de données, comme des échanges entre contrôleurs logiques ou avec des composants externes à l'architecture. Nous pensons que les propositions présentées dans ce mémoire (recherche des bornes par preuves itératives, construction du modèle formel par instanciation de modèles génériques de composants, méthode globale d'abstraction) demeurent applicables pour ces recherches dont la première difficulté réside dans le développement de nouveaux modèles génériques de composants, caractéristiques du nouveau protocole et/ou des nouvelles sources de flux.

En dernier lieu, il nous paraît possible de reprendre les principes et certains résultats de ces travaux pour l'évaluation des bornes des distributions de valeurs d'autres grandeurs physiques que le temps, comme par exemple celles caractérisant le comportement d'un processus commandé (position, vitesse, température, ...). Là encore, la conduite de ces travaux devra s'appuyer sur une modélisation formelle précise du comportement du processus.

Bibliographie

[AA08] Boussad Addad and Saïd Amari. Modeling and response time evaluation of ethernet-based control architectures using timed event graphs and max-plus algebra. In *Proc. of the 4th annual IEEE Conference on Automation Science and Engineering*, Washington, USA, 2008.

[ACEF08] Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. An inverse method for parametric timed automata. In Vesa Halava and Igor Potapov, editors, *Proceedings of the 2nd Workshop on Reachability Problems in Computational Models (RP'08)*, volume 223 of *Electronic Notes in Theoretical Computer Science*, pages 29–46, Liverpool, UK, December 2008. Elsevier Science Publishers.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2) :183–235, 1994.

[BA05a] M. Bitam and H. Alla. Performance evaluation of a TCP/IP transmission using hybrid Petri nets. In *AICCSA '05 : Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications*, pages 54–I, Washington, DC, USA, 2005. IEEE Computer Society.

[BA05b] Behzad Bordbar and Rachid Anane. An architecture for automated QoS resolution in wireless systems. In *AINA '05 : Proceedings of the 19th International Conference on Advanced Information Networking and Applications*, pages 774–779, Washington, DC, USA, 2005. IEEE Computer Society.

[Bal96] F. Balarin. Approximate reachability analysis of timed automata. In *RTSS '96 : Proceedings of the 17th IEEE Real-Time Systems Symposium (RTSS '96)*, page 52, Washington, DC, USA, 1996. IEEE Computer Society.

[BBD+02] Gerd Behrmann, Johan Bengtsson, Alexandre David, Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Uppaal implementation secrets. In *FTRTFT '02 : Proceedings of the 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 3–22, London, UK, 2002. Springer-Verlag.

[BBF+01] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer, 2001.

[BR00] T. Ball and S. Rajamani. Boolean programs : A model and process for software analysis. Research report, Microsoft Research, 2000.

[BS00] Béatrice Bérard and Luis Sierra. Comparing verification with HyTech, Kronos and Uppaal on the railroad crossing example. Research Report LSV-00-2, Laboratoire Spécification et Vérification, ENS Cachan, France, January 2000.

[BT97] Jean-Yves Le Boudec and Patrick Thiran. A note on time and space methods in network calculus. Research Report 97/224, Laboratoire de Réseaux et Communication, Ecole Polytechnique Fédérale de Lausanne, April 1997.

[BT01] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus*. Springer-Verlag LNCS 2050, 2001.

[CBV06] Gianluca Cena, Ivan Cibrario Bertolotti, and Adriano Valenzano. Experimental analysis of latencies in Ethernet. In *Proc. of WFCS 2006*, 2006.

[CCK+02] Pankaj Chauhan, Edmund Clarke, James Kukula, Samir Sapra, Helmut Veith, and Dong Wang. Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis. In *in Proceedings of FMCAD*, pages 33–51, 2002.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1981. Springer-Verlag.

[CEFX06] Rémy Chevallier, Emmanuelle Encrenaz-Tiphène, Laurent Fribourg, and Weiwen Xu. Verification of the generic architecture of a memory circuit using parametric timed automata. In Eugène Asarin and Patricia Bouyer, editors, *Proceedings of the 4th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 113–127, Paris, France, September 2006. Springer.

[Cel91] Francois E. Cellier. *Continuous System Modeling*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1991.

[CFH+03] Edmund Clarke, Ansgar Fehnker, Zhi Han, Bruce Krogh, Joël Ouaknine, Olaf Stursberg, and Michael Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *International Journal of Foundations of Computer Science*, 14(4) :583–604, 2003.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. The MIT press, 1999.

[Com07] G. Combs. Wireshark. In *http ://www.wireshark.org*, 2007.

[Cru91] Rene L. Cruz. A calculus for network delay. *IEEE Transactions on Information Theory*, 37, January 1991.

[DD01] Satyaki Das and David L. Dill. Successive approximation of abstract transition relations. In *LICS '01 : Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, page 51, Washington, DC, USA, 2001. IEEE Computer Society.

[DFF+06] Alessandro Depari, Paolo Ferrari, Alessandra Flammini, Daniele Marioli, and Andrea Taroni. Multi-probe measurements instrument for real-time Ethernet networks. In *Proc. of WFCS 2006*, 2006.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1) :269–271, 1959.

[DRF+07] Bruno Denis, Silvain Ruel, Jean-Marc Faure, Gaëlle Marsal, and Georg Frey. Measuring the impact of vertical integration on response times in Ethernet fieldbuses. In *Proc. of ETFA'07*, pages 532–539, 2007.

[EC99] Johan Eker and Anton Cervin. A matlab toolbox for real-time and control systems co-design. In *RTCSA '99 : Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, page 320, Washington, DC, USA, 1999. IEEE Computer Society.

[EH82] E. Allen Emerson and Joseph Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *STOC '82 : Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 169–180, New York, NY, USA, 1982. ACM.

[EH86] E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "not never" revisited : on branching versus linear time temporal logic. *J. ACM*, 33(1) :151–178, 1986.

[FE98] Peter Fritzson and Vadim Engelson. Modelica – A Unified Object-Oriented Language for System Modelling and Simulation. In *ECCOP '98 : Proceedings of the 12th European Conference on Object-Oriented Programming*, pages 67–90, London, UK, 1998. Springer-Verlag.

[Fel05] Max Felser. Real-Time Ethernet - Industry Prospective. *Proceedings of the IEEE*, 93(6) :1118–1128, 2005.

[FFV06] Paolo Ferrari, Alessandra Flammini, and Stefano Vitturi. Performance analysis of PROFINET networks. *Computer Standards and Interfaces*, 28 :369–385, 2006.

[Fre05] Goran Frehse. PHAVer : Algorithmic Verification of Hybrid Systems Past HyTech. In *HSCC*, pages 258–273, 2005.

[Geo05] Jean-Philippe Georges. *Systèmes contrôlés en réseau : Evaluation de performances d'architectures Ethernet commutés*. PhD thesis, Université Henri Poincaré, Nancy 1, novembre 2005.

[GF07] J. Greifeneider and G. Frey. Probabilistic timed automata for modeling networked automation systems. In *Proc. 1st IFAC DCDS, Cachan*, pages 143 – 148, 2007.

[GKDR06] Jean-Philippe Georges, Nicolas Krommenacker, Thierry Divoux, and Eric Rondeau. A design process of switched ethernet architectures according to real-time application constraints. *Engineering Applications of Artificial Intelligence*, 19 :335–344, 2006.

[GRD02] J.-P. Georges, E. Rondeau, and T. Divoux. How to be sure that Ethernet networks will satisfy the real-time requirements ? In *Proc. of IEEE Int. Symposium on Industrial Electronics*, July 2002.

[HHWT97] T.A. Henzinger, P.H. Ho, and H. Wong-Toi. HYTECH : A Model Checker for Hybrid Systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1) :110–122, 1997.

[Hol04] Gerald J. Holzmann. *The SPIN Model Checker*. Addison Wesley Professional, 2004.

[JO08] Zulema Juarez-Orozco. *Vérification de propriétés quantitatives des systèmes logiques par model-checking hybride*. PhD thesis, ENS Cachan (France), 2008.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Syst.*, 2(4) :255–299, 1990.

[Kro06] D. Kroening. Computing over-approximations with bounded model checking. *Electronic Notes in Theoretical Computer Science*, 144 :79–92, 2006.

[Kur94] R. Kurshan. *Computer-aided verification of coordinating processes : the automata-theoretic approach*. Princeton University Press, 1994.

[LF07] Liu Liu and Georg Frey. Simulation approach for evaluating response times in networked automation systems. In *Proc. of ETFA'07*, pages 1061–1068, 2007.

[Lim09] Steve Limal. *Architectures de contrôle-commande redondantes à base d'Ethernet Industriel : Modélisation et validation par model-checking temporisé*. PhD thesis, ENS Cachan (France), janvier 2009.

[LLPY97] K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems : compact data structure and state-space reduction. In *RTSS '97 : Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97)*, pages 14–24, Washington, DC, USA, 1997. IEEE Computer Society.

[LPY95] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Model-checking for real-time systems. In *FCT '95 : Proceedings of the 10th International Symposium on Fundamentals of Computation Theory*, pages 62–88, London, UK, 1995. Springer-Verlag.

[LPY97] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1-2) :134–152, 1997.

[Mar04] Steven Martin. *Maîtrise de la dimension temporelle de la qualité de service dans les réseaux*. PhD thesis, Université paris XII (France), juillet 2004.

[Mar06] Gaëlle Marsal. *Evaluation of time performances of Ethernet-based automation systems by simulation of high-level Petri nets*. PhD thesis, ENS Cachan (France) and Kaiserslautern University (Germany), december 2006.

[Meu06] Pascal Meunier. *Evaluation de perfromances d'architectures de commande de systèmes automatisés industriels*. PhD thesis, ENS Cachan (France), mars 2006.

[MLAH99] Jesper Møller, Jakob Lichtenberg, Henrik Reif Andersen, and Henrik Hulgaard. Difference decision diagrams. *Computer Science Logic*, 1683/1999 :826–841, 1999.

[Neu07] Peter Neumann. Communication in industrial automation - What is going on ? *Control Engineering Practice*, 15 :1332–1347, 2007.

[OB09] Carolina Osorio and Michel Bierlaire. An analytic finite capacity queueing network model capturing the propagation of congestion and blocking. *European Journal of Operational Research*, 196 :996–1007, 2009.

[PMT06] J. T. Parrott, J. R. Moyne, and D. M. Tilbury. Experimental Determination of Network Quality of Service in Ethernet : UDP, OPC, and VPN. In *2006 American Control Conference, ACC'06*, 2006.

[Pnu81] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13 :45 – 60, 1981.

[PTP04] Nuno Pereira, Eduardo Tovar, and Luis Miguel Pinho. Timeliness in COTS factory-floor distributed systems : what role for simulation ? In *Proc. of IEEE International Workshop on Factory Communication Systems*, pages 13 – 21, September 2004.

[RdSF08a] Silvain Ruel, Olivier de Smet, and Jean-Marc Faure. Building effective formal models to prove time properties of networked automation systems. In *WODES'08 : Proceedings of the 9th International Workshop On Discrete Event Systems*, pages 334–339, 2008.

[RdSF08b] Silvain Ruel, Olivier de Smet, and Jean-Marc Faure. Efficient representation for formal verification of time performances of networked automation architectures. In *World Congress IFAC 08 : Proceedings of the 17th World Congress of The International Federation of Automatic Control*, pages 5119–5124, Seoul, Korea, 2008.

[RdSF09] Silvain Ruel, Olivier de Smet, and Jean-Marc Faure. Finding the bounds of response time of networked automation systems by iterative proofs. In *INCOM'09 : Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing*, page to be published, 2009.

[RNPH06] G. Rodriguez-Navas, J. Proenza, and H. Hansson. An Uppaal model for formal verification of master/slave clock synchronization over the controller area network. In *Proceedings of the International Workshop on Factory Communication Systems*, pages 3–12, june 2006.

[SF07] Jean-Luc Scharbarg and Christian Fraboul. Simulation for end-to-end delays distribution on a switched Ethernet. In *Proc. of ETFA'07*, pages 1092–1099, 2007.

[SMF97] Thomas Stauner, Olaf Müller, and Max Fuchs. Using HYTECH to Verify an Automative Control System. In *Hybrid and Real-Time Systems, International Workshop, HART'97*, volume 1201 of *Lecture Notes in Computer Science*, pages 139–153. Springer, 1997.

[TV01] Eduardo Tovar and Francisco Vasques. Distributed computing for the factory-floor : a real-time approach using worldfip networks. *Comput. Ind.*, 44(1) :11–31, 2001.

[Var01] A. Varga. The omnet++ discrete event simulation system. In *Proc. of the European Simulation Multiconference*, Prage, République Tchèque, June 2001.

[Vit01] Stefano Vitturi. On the use of Ethernet at low level of factory communication systems. *Computer Standards and Interfaces*, 23 :267–277, 2001.

[Wan04] F. Wang. Formal verification of timed systems : A survey and perspective. *Proceedings of the IEEE*, 92(8) :1283 – 1305, August 2004.

[WJK+01] Dong Wang, Pei-Hsin Jiang, James Kukula, Yunshan Zhu, Tony Ma, and Robert Damiano. Formal property verification by abstraction refinement with formal, simulation and hybrid engines. In *DAC '01 : Proceedings of the 38th conference on Design automation*, pages 35–40, New York, NY, USA, 2001. ACM.

[WL93] Pierre Wolper and Denis Leroy. Reliable hashing without collision detection. In *Computer Aided Verification. 5th International Conference*, pages 59–70. Springer-Verlag, 1993.

[Zai04] Dmitry A. Zaitsev. Switched LAN simulation by colored Petri nets. *Mathematics and Computers in Simulation*, 65(3) :245–249, 2004.

[Zim88] Hubert Zimmermann. OSI reference model—the ISO model of architecture for Open Systems Interconnection. *Innovations in Internetworking*, pages 2–9, 1988.

Annexe A

Détermination expérimentale du mode d'échange de données dans un API

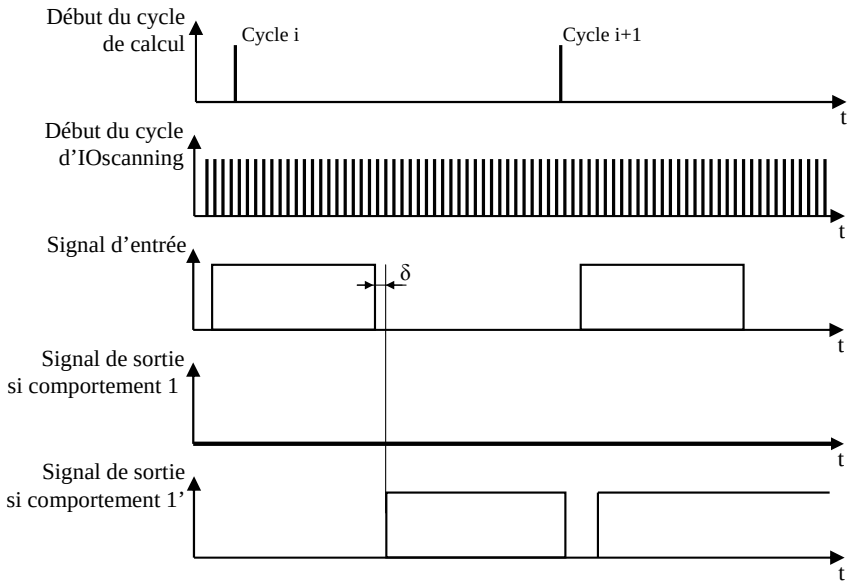


FIG. A.1 – Chronogramme explicatif de l'essai 1

Premier essai pour discriminer entre 1 et 1' :

- Le processeur de calcul prend-il prendre en compte les valeurs de ses entrées au cours du calcul ?
- L'essai (figure A.1) a été réalisé avec un cycle de calcul long (400 ms) et un IOscanning rapide (10 ms). Un signal périodique est fourni en entrée du système. Tout au long de l'exécution du programme, le processeur de calcul compare la valeur de cette entrée avec sa valeur au début du cycle de calcul. Tant que les deux valeurs sont identiques, la sortie émise vaut 0. Dès qu'une différence est détectée, elle est mise à 1 et est bloquée à cette valeur jusqu'à la fin du cycle de calcul. il faut noter que

dans le cas du comportement 1', le délai entre le changement de la valeur du signal d'entrée et celui de la valeur du signal de sortie (δ) dépend du type de comportement pour l'écriture des sorties (2 ou 2').

– La valeur de sortie a toujours été égale à 0. **Le processeur de calcul lit donc toutes ses entrées en zone tampon 2 uniquement au début de l'exécution du programme**.

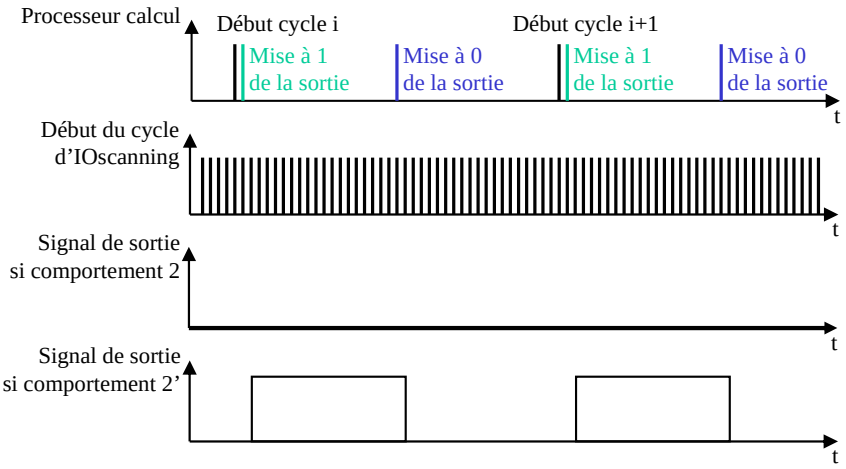


FIG. A.2 – Chronogramme explicatif de l'essai 2

Deuxième essai pour discriminer entre 2 et 2' :

– Le processeur de calcul émet-il ses sorties dès qu'elles sont calculées ?

– L'essai (figure A.2)a été réalisé avec un cycle de calcul long (400 ms) et un IOscanning rapide (10 ms). Une sortie est mise à 1 au début du programme puis est remise à 0 au milieu du programme.

– Aucune variation de la sortie n'a été observée, elle est toujours restée à 0. **Le processeur de calcul place donc en zone tampon 1 toutes ses sorties uniquement à la fin de l'exécution du programme**.

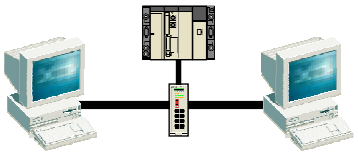


FIG. A.3 – Architecture utilisée pour les troisième et quatrième essais

Troisième essai pour discriminer entre 3 et 3' :

– La carte de communication prend-elle en compte les nouvelles valeurs des sorties à envoyer aux MES au cours de la phase d'émission de ces sorties ?

– L'essai (figure A.4) a été réalisé sur une architecture (figure A.3) constituée d'un API et de deux ordinateurs, chacun hébergeant plusieurs serveurs Modbus. L'API était configuré avec un cycle de calcul court (5 ms) et un IOscanning très lent (1 s). En envoyant 30 requêtes aux deux ordinateurs (la sortie dans la mémoire tampon 1 est envoyée dans chaque requête) au cours du cycle d'IOscanning, la phase d'émission

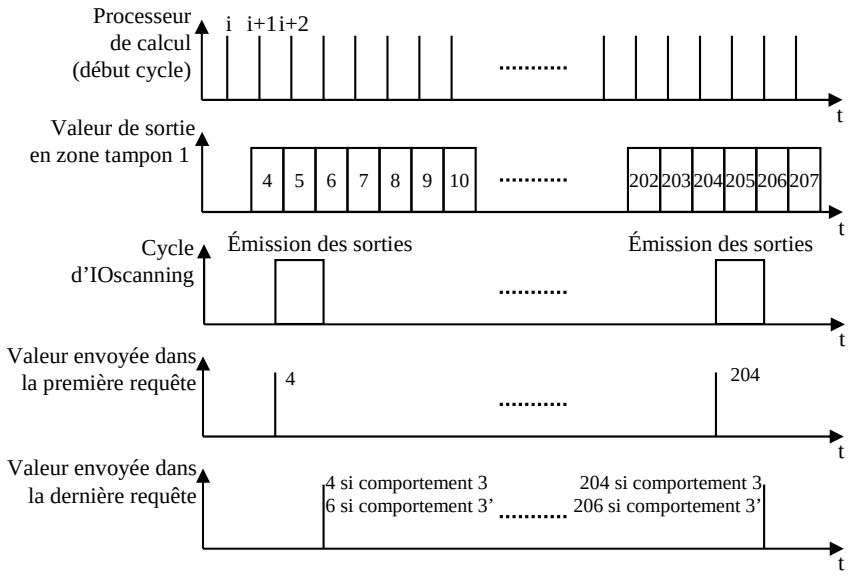


FIG. A.4 – Chronogramme explicatif de l'essai 3

des requêtes dure environ 7 ms (il reste donc 993 ms d'attente avant la prochaine phase d'émission). Le processeur de calcul incrémente la valeur de cette sortie à chaque cycle de calcul. Un des ordinateurs reçoit la première et la dernière requêtes émises aux cours du cycle d'IOscanning et compare les valeurs reçues. Toutes les autres requêtes sont envoyées vers l'autre ordinateur. Entre deux cycles d'IOscanning consécutif, la valeur de sortie va augmenter d'environ 200 alors qu'elle ne peut avoir évoluer que de 1 ou 2 entre la première et la dernière requête émise.

– Pour l'ordinateur qui reçoit la première et la dernière requêtes, les écarts entre deux valeurs consécutives étaient soit nuls soit de l'ordre de 200. **La carte de communication recopie donc toutes les valeurs des sorties à émettre en une seule fois, au début de son cycle**.

Quatrième essai pour discriminer entre 4 et 4' :

– La carte de communication met-elle à disposition du processeur de calcul les valeurs des entrées au fur et à mesure de leur arrivée ou en une seule fois, quand elles sont toutes arrivées ?
– L'essai (figure A.5) a été réalisé avec la même architecture que pour l'essai 3 (figure A.3) et l'API a été configuré avec un cycle de calcul court (5 ms) et un IOscanning très lent (1 s). Pour cet essai, seulement deux serveurs Modbus étaient implantés sur les deux ordinateurs. Le premier serveur (hébergé par le premier ordinateur) répond immédiatement alors que le second (hébergé par le second ordinateur) retarde sa réponse de 0,5 s. Ils comptent tous les deux le nombre de requêtes reçues et répondent par ce nombre. Le processeur de calcul compare les valeurs reçues des deux serveurs et met une sortie à 0 si elle sont identiques, à 1 si elles sont différentes. Si la sortie vaut tout le temps 0, le processeur de calcul voit toujours la même valeur pour les deux réponses des serveurs et donc la carte de communication ne les transmet donc qu'une fois qu'elles sont toutes arrivées. Dans le cas contraire, cela signifie que la carte de communication transmet les réponses au fur et à mesure de leur arrivée.

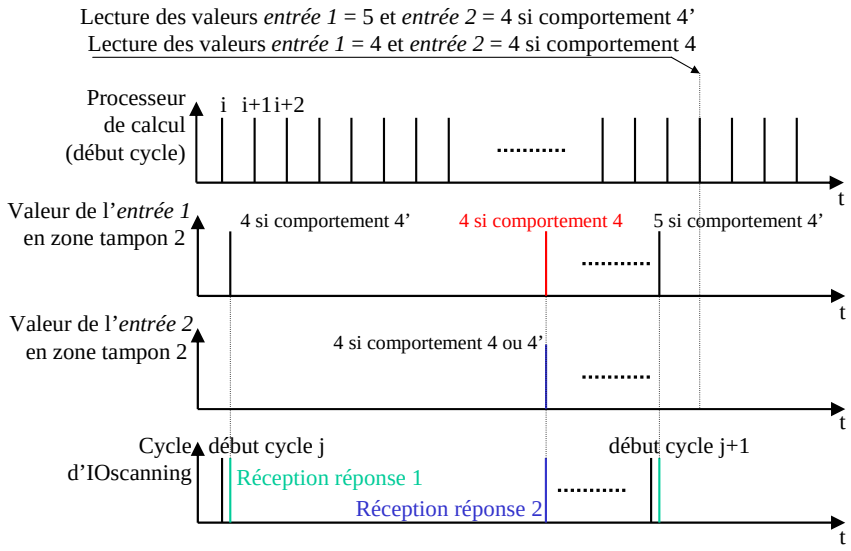


FIG. A.5 – Chronogramme explicatif de l'essai 4

– La valeur de la sortie a été alternativement 0 et 1, ce qui signifie que le processeur de calcul a observé des valeurs différentes pour les deux réponses et donc que **la carte de communication place dans la zone tampon 2 les réponses des MES au fur et à mesure de leur arrivée**.

Annexe B

Aide au choix de Hinit pour l'algorithme de recherche par dichotomie

Algorithm 3 Algorithme de recherche par dichotomie adaptative pour obtenir la borne supérieure d'une performance temporelle

$L \leftarrow Linit$, $H \leftarrow Linit + 2 * \tau init$, $Hlimite = 10^8$
while $H \neq L$ **do**
  $\tau \leftarrow L + \lfloor \frac{H-L}{2} \rfloor$
  vérifier les propriétés P2 et P3
  if (P3 est vraie) **then**
    $H \leftarrow Min(Hlimite; L + 4 * (\tau - L))$, $L \leftarrow \tau$
  end if
  if (P3 est fausse) $\wedge$ (P2 est fausse) **then**
    $H \leftarrow \tau$, $Hlimite \leftarrow \tau$
  end if
  if (P3 est fausse) $\wedge$ (P2 est vraie) **then**
    $H \leftarrow \tau$, $Hlimite \leftarrow \tau$
    $L \leftarrow \tau$
  end if
end while
$Max(pt) \leftarrow \tau$
return Max(pt) la valeur maximale de la performance temporelle étudiée

Pour obtenir la valeur de Hinit, une estimation rapide de la valeur maximale de la performance temporelle étudiée par une méthode au pire des cas peut être envisagée, mais elle risque de ne pas être facile à mettre en oeuvre sur des cas non triviaux. C'est pourquoi une méthode itérative pour déterminer rapidement les limites de la zone de recherche, méthode qui peut être directement couplée à la recherche par dichotomie, a été proposée. Cette méthode, appelée dichotomie adaptative, est illustrée par l'algorithme 3.

La différence principale avec l'algorithme 1 est que la limite maximale de la zone de recherche n'est pas fixée à l'initialisation de l'algorithme. Elle a bien une valeur au démarrage (Hlimite = 10^8), mais cette valeur est très grande pour convenir à tous les cas d'étude et ne sert pas directement pour la dichotomie. Linit est initialisé à 0 et τinit est fixé par l'utilisateur. La borne H augmente ($H \leftarrow Min(Hlimite; L+4*(\tau-L))$) tant qu'il n'est pas possible d'être sûr qu'elle soit supérieure à la borne supérieure de la performance temporelle étudiée, mais doit toujours rester inférieure à Hlimite. Le choix de rajouter 4 fois ($\tau - L=$ à L pour déterminer la nouvelle valeur de H permet de doubler le domaine de recherche. Quand la propriété P3 est fausse, H est supérieure à la valeur maximale de la perfromance temporelle étudiée. La limite Hlimite prend alors la valeur de τ et H est bornée par Hlimite. La suite correspond à une dichotomie normale entre L et H (avec H = Hlimite).

Il suffit, pour que cet algorithme converge rapidement, qu'à l'initialisation τinit soit du même ordre de grandeur que la borne à trouver, ceci afin d'éviter un trop grand nombre d'itérations pour trouver la limite maximale de la zone de recherche. Dans ce but, une estimation rapide de la performance temporelle étudiée est proposée. Il est possible de se ramener à une architecture minimale du système, même si cette architecture ne correspond pas vraiment à la réalité, puisque seul l'ordre de grandeur est important. Il est ainsi possible de se ramener à l'estimation du temps de réponse d'un système comportant uniquement le processeur de calcul, la carte de communication et un MES, sans aucune autre contrainte (imposée par le partage de certains composants de l'AAR).

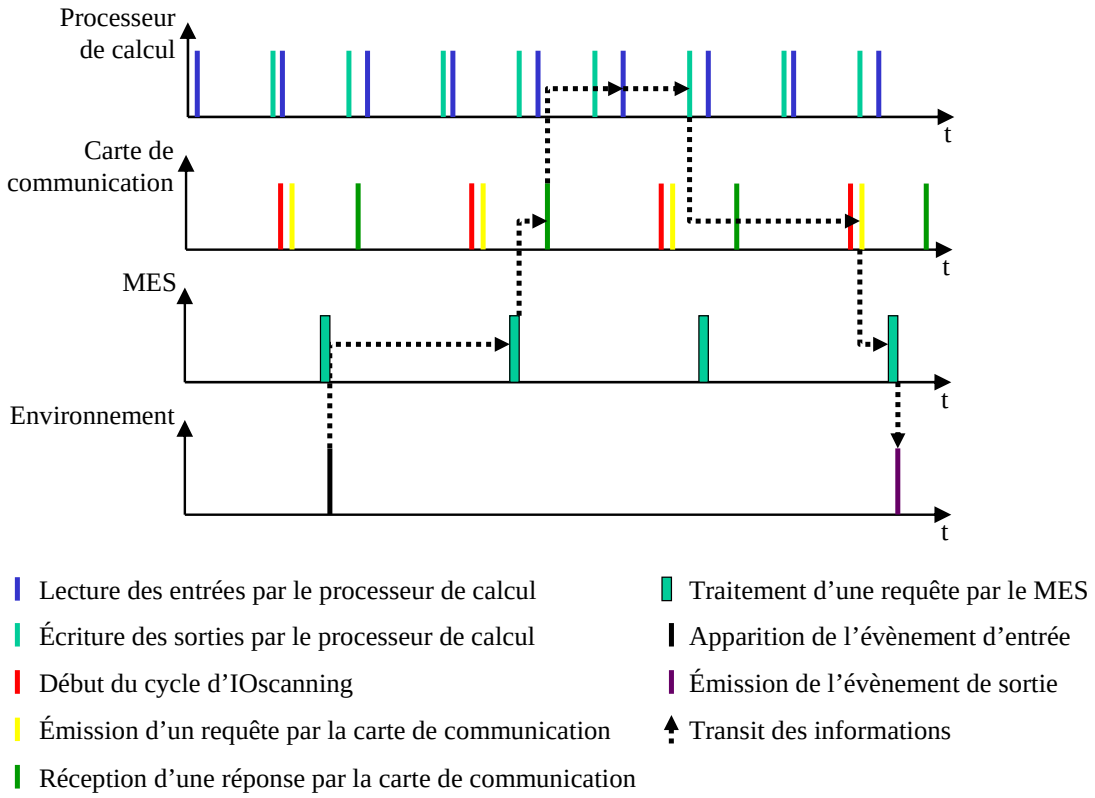


FIG. B.1 – Transit des informations entre les évènements d'entrée et de sortie

Pour une AAR comportant uniquement un processeur de calcul, une carte de communication et un MES, un cycle correspondant à la réponse à un évènement d'entrée peut être représenté (figure B.1). L'évènement d'entrée peut apparaitre n'importe quand au cours du cycle d'IOscanning de la carte de communication ; l'évènement d'entrée peut donc attendre un cycle d'IOscanning complet avant d'être pris en compte par le MES. L'information arrive au niveau de la carte de communication après un délai dû à la durée de traitement de la requête par le MES et au temps de traversée du réseau. Le processeur de calcul la prendra en compte lors de sa prochaine lecture des entrées, soit au plus tard dans un cycle du processeur de calcul. L'émission de la sortie se fera au plus tard un cycle du processeur de calcul plus tard. Si l'on ne corrèle pas la durée du cycle du processeur de calcul à la durée du cycle de l'IOscanning, on peut considérer que cette information peut n'être prise en compte que lors du prochain cycle d'IOscanning (ajout d'une attente de la durée du cycle d'IOscanning). La requête arrivera au MES après une durée fixée par le temps d'émission de la carte de communication et le temps de traversée du réseau. L'observateur, lui, ne verra la valeur de sortie qu'après le traitement par le MES. Nous avons donc dans le pire des cas :

$$TR_{max} = 2 * T_{IOscanning} + 2 * T_{processeur} + 2 * T_{MES} + 2 * T_{réseau} + T_{émission} \tag{B.1}$$

Par cette méthode, il est possible de déterminer rapidement un ordre de grandeur pour le plus grand des temps de réponse.

Résumé

Ce mémoire de thèse propose une approche pour l'obtention des bornes des performances temporelles d'une Architecture d'Automatisation en Réseau par preuves itératives de propriétés d'atteignabilité sur un modèle formel de l'architecture. Ces propriétés d'atteignabilité sont définies grâce à un automate observateur temporisé et paramétré, dont les gardes de certaines transitions sont fonction d'un paramètre temporel. A chaque itération, les résultats de preuves permettent de déterminer la valeur de ce paramètre pour la prochaine itération ; un algorithme de recherche par dichotomie assure la convergence des itérations.La mise en oeuvre de cette approche sur des architectures de taille non triviale a nécessité le développement d'une méthode d'abstraction qui comporte deux étapes : simplification de la structure et modification des modèles formels des composants figurant dans la structure simplifiée, ceci afin de prendre en compte les phénomènes de concurrence entre requêtes émises par différents composants.Ces contributions formelles et méthodologiques ont été validées expérimentalement par le traitement de plusieurs cas de taille et complexité croissantes, basés sur le protocole Modbus TCP/IP.

Mots-clés: Architectures d'automatisation en réseau, évaluation des performances temporelles, model-checking temporisé, abstraction, Modbus TCP/IP

Abstract

This thesis proposes an approach to obtain the bounds of temporal performances of a Networked Automation Architecture by iterative proofs of reachability properties on a formal model of the architecture. These reachability properties are defined by means of a parametric timed observer automaton, whose guards of transitions are based on a time parameter. At each iteration, the results of proofs are used to calculate the value of this parameter for the next iteration ; a dichotomy algorithm ensures the convergence of iterations. The implementation of this approach on nontrivial architectures has required the development of an abstraction method which comprises two steps : simplification of the structure and modification of formal models of the components contained in the simplified structure in order to take into account the phenomena of concurrency between requests emitted by different components. These formal and methodological contributions have been validated experimentally by the treatment of several case studies of increasing complexity and size, based on Modbus TCP / IP.

Keywords: Networked automation systems, evaluation of time performances, timed model-checking, abstraction, Modbus TCP/IP