Évaluation de performance d'architectures de commande de systèmes automatisés industriels

Pascal Meunier

THÈSE DE DOCTORAT
DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

présentée par

Monsieur Pascal MEUNIER

pour obtenir le grade de

DOCTEUR DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

Domaine :
Électronique Électrotechnique et Automatique

Sujet de la thèse :

ÉVALUATION DE PERFORMANCE D'ARCHITECTURES DE COMMANDE DE SYSTÈMES AUTOMATISÉS INDUSTRIELS

Thèse présentée et soutenue à Cachan le jeudi 30 mars 2006 devant le jury composé de :

Thierry DIVOUX	Professeur à l'Université Nancy I (CRAN)	Président
Lothar LITZ	Professeur à l'Université de Kaiserslautern (EIT)	Rapporteur
Zineb SIMEU-ABAZI	Maître de Conférences HDR à l'INPG (LAG)	Rapporteur
Ludovic GUILLEMAUD	Docteur Ingénieur ALSTOM Transport	Invité
Jean-Jacques LESAGE	Professeur à l'ENS de Cachan (LURPA)	Directeur de thèse
Bruno DENIS	Maître de Conférences à l'ENS de Cachan (LURPA)	Co-encadrant

Laboratoire Universitaire de Recherche en Production Automatisée
ENS de Cachan/ EA 1385/Université de Paris Sud 11
61 avenue du Président Wilson - 94235 Cachan Cedex

à mes Parents

à mes amis

à tous ceux qui n'ont pu être présents

Avant-propos

Le travail présenté dans ce mémoire a été effectué au sein du Laboratoire Universitaire de Recherche en Production Automatisée (LURPA - EA 1385) de l'École Normale Supérieure de Cachan.

J'adresse mes sincères remerciements au Professeur Jean-Jacques Lesage pour m'avoir accueilli dans le laboratoire, au sein de l'équipe ISA et pour avoir assumé la direction de mes travaux et pour avoir oeuvré afin que je puisse soutenir mes travaux dans de bonnes conditions.

Merci à Bruno Denis pour son encadrement, ses conseils avisés qui ont permis de faire évoluer le champ applicatif de mes travaux ainsi que pour les heures, les jours et les semaines passés à la mise en oeuvre des expérimentations nécessaires à la validation de mes travaux.

Merci aux membres du jury pour le temps qu'ils m'ont consacré.

Merci à tous les membres du LURPA, partis ou en poste, pour avoir fait de ces nombreuses années une période que je n'oublierai pas. Merci à mes anciens collègues thésards : Bruno, Christophe, Nicolas et Olivier pour leur amitié et leur aide. Merci tout particulièrement à Olivier, Jean-Marc, Christophe, Maryvonne, Marie-France et Françoise sans qui le LURPA ne serait pas vraiment le LURPA. Pardon à tous ceux que j'ai oubliés.

Merci à mes proches pour m'avoir supporté dans mon travail, pour avoir supporté mes humeurs et pour m'avoir permis de continuer dans les moments de doute. Sans eux, ce document n'aurait vraisemblablement jamais vu le jour. Merci à ma mère et à mon père d'avoir essayé de comprendre à quoi j'occupais mon temps durant ces années et de m'avoir encouragé à faire aboutir mes travaux.

Table des matières

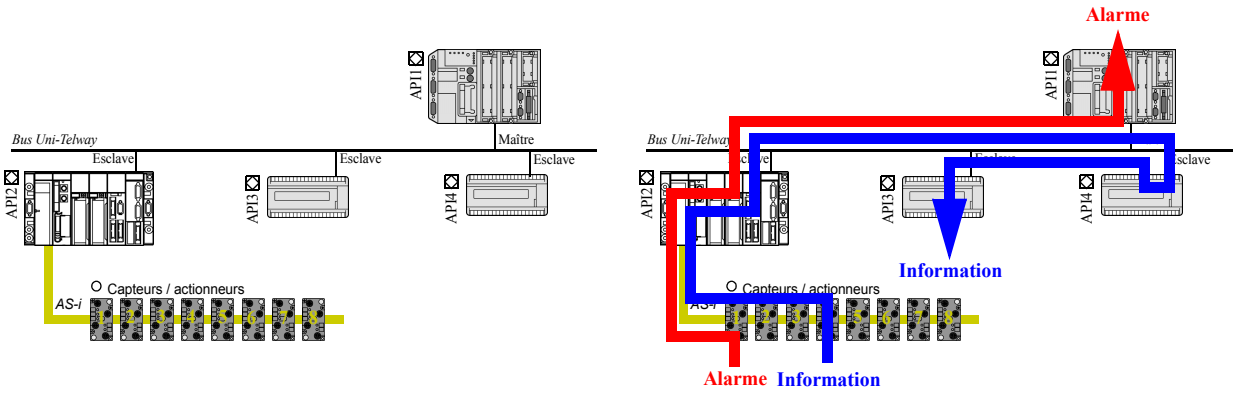
Introduction

Depuis de nombreuses années, les industriels ont automatisé leurs usines et moyens de production pour améliorer la qualité de leurs procédés et de leurs produits. Tout système industriel automatisé comporte une architecture de contrôle-commande dont il est nécessaire de connaitre les performances qui ont un impact direct sur les performances des processus commandés. L'ingénieur en charge de ces architectures de contrôle-commande a un besoin vital d'en évaluer les performances tout au long du cycle de vie :

- lors de sa conception, en tout début du cycle de vie, l'architecte doit être capable de proposer des architectures cohérentes avec le cahier des charges qui lui a été fourni et cela sans pouvoir faire de test sur l'installation finale (qui n'a pas alors été construite). Les choix effectués par l'architecte à ce niveau sont primordiaux car des changements futurs d'architectures seraient fort coûteux. Une *évaluation* des performances attendues des architectures ainsi conçues est donc indispensable ;
- lors de la période d'installation et de mise au point, l'automaticien installateur doit paramétrer les différents équipements de l'architecture de contrôle-commande. Afin de livrer au client le système au plus tôt, il ne lui est pas possible de d'effectuer tous les tests nécessaires pour affiner les paramétrages de l'architecture. Ainsi, pour éviter que de mauvais réglages ne viennent dégrader les performances générales de l'architecture, l'automaticien doit avoir *évalué* l'impact des différents réglages sur les performances de la commande avant la mise au point de l'architecture ;
- lors des phases de modification ou d'évolution, le système de production est généralement en exploitation. Or l'architecte peut être amené à devoir améliorer les performances du contrôle-commande, que ce soit par amélioration des paramétrages ou par adjonction de nouveaux matériels ou programmes. L'*évaluation* hors-ligne des performances du système modifié est alors indispensable si l'on veut éviter de perturber le fonctionnement du système de production.

Pour évaluer les performances temporelles (retards de causalité) d'une architecture de contrôle-commande, l'approche courament pratiquée consiste en une simple somme de retards (simulation numérique) occasionnés par chacun des équipements de commande traversé par un flux d'information. Par exemple sur l'architecture de la figure 1, comment peut-on évaluer les temps de remontée d'une alarme ou de transmission d'une information ?

La plupart des comportements décrits par les documentations des constructeurs d'équipements de commande, peuvent être assimilées à des retards. L'hypothèse courante est que ces retards sont indépendants les uns des autres et qu'ils sont uniformément répartis entre deux bornes, on peut ainsi facilement obtenir par simulation numérique des répartitions des temps de transmission d'alarmes ou d'informa-



-a- Architecture matérielle

-b- Deux flux d'informations particuliers

Figure 1 : Exemples d'architecture de contrôle-commande

tions. Or ces documentations ne procurent que les informations relatives aux comportements des équipements à vide, c'est-à-dire sans sa charge correspondant à son fonctionnement dans son architecture.

Afin d'illustrer les limites de cette approche, nous donnons sur la figure 2 les performances calculées, comparées aux performances que nous avons mesurées sur l'architecture de la figure 1 hors charge (une seule information transite dans l'architecture) et en charge (l'ensemble des informations nécessaires au fonctionnement du système transitent dans l'architecture).

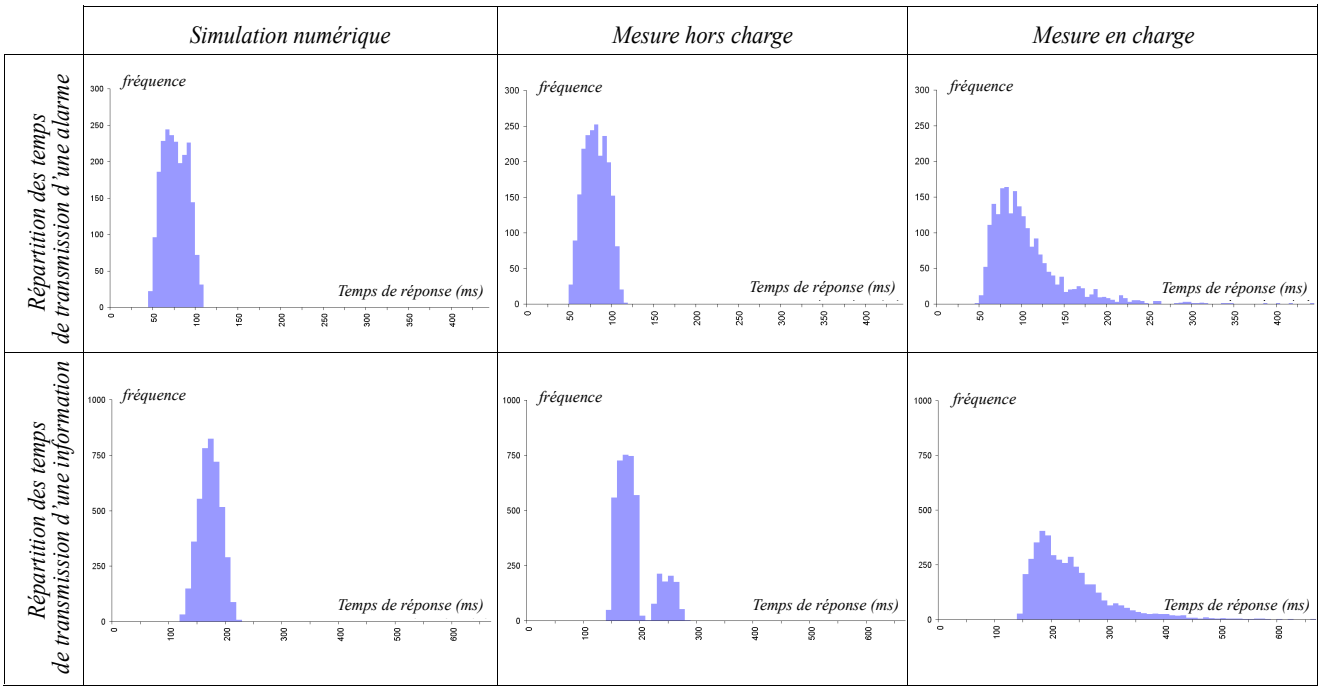


Figure 2 : Évaluation des performances obtenues par simulation et par mesure en charge et hors-charge

On peut remarquer que dans le cas d'une alarme, les répartitions et les données statistiques (minimum, moyenne, maximum) de la simulation numérique et des mesures hors charge sont très proches. En revanche, cela n'est plus le cas pour la transmission d'informations. Cela s'explique par le fait que l'hypothèse d'indépendance des retards est fausse, notamment dans le cas d'une transmission d'infor-

mation puisque le réseau de communication étant utilisé à deux reprises, ces deux utilisations sont temporellement corrélées.

Lorsque l'on observe les répartitions des temps de transmission en charge, les différences entre simulation numérique et mesure portant sur la répartition, la moyenne et la valeur maximale deviennent fortement marquées : par exemple, le rapport entre valeur maximale de la simulation numérique et de la mesure en charge d'une alarme est proche de 400%.
Une approche uniquement basée sur une modélisation par somme de retards indépendants ne peut donc pas répondre de manière suffisante au besoin de l'architecte.

Notre apport consiste à proposer une méthode d'évaluation des performances d'architecture de commande par simulation de modèles réseaux de Petri qui prennent non seulement en compte la charge du système complet mais qui réponde aussi aux besoins de l'architecte tout au long du cycle de vie de l'architecture. Nous montrerons que cette méthode est précise, peu sensible aux imprécisions de paramétrages et qu'elle permet d'obtenir des bons résultats rapidement.

Pour cela, nous présenterons dans un premier temps le contexte scientifique de notre étude en décrivant la structure générale d'une architecture de contrôle-commande. Puis nous proposerons une revue de littérature des travaux d'évaluation de performances de la communauté scientifique.

Nous présenterons ensuite notre méthodologie, en commençant par le choix de notre outil d'évaluation. Puis nous exposerons la modélisation des Architectures Fonctionnelles et Matérielles ainsi que leur composition pour constituer le modèle de l'Architecture Opérationnelle. Nous compléterons enfin ces modèles pour les préparer à la simulation qui nous permettra d'obtenir des prédictions de performances.

Pour que l'ingénieur architecte soit correctement accompagné dans son processus d'évaluation de performances, nous lui proposerons ensuite un ensemble de guides de modélisation qui lui facilitera l'application de notre méthode.

Dans le cas où l'architecte doit concevoir un modèle pour un nouvel équipement matériel, nous présenterons, la méthodologie qui lui est proposée, en utilisant des réseaux de Petri et en l'appliquant à un exemple courant : un Automate Programmable Industriel.

Enfin, nous validerons nos travaux au travers du traitement d'un exemple comportant les problèmes significatifs dans les architectures de contrôle-commande. Nous confronterons les résultats de mesure sur une architecture réelle aux résultats de simulation d'un modèle correctement paramétré afin d'évaluer la pertinence de nos modèles. Enfin, nous étudierons l'influence des erreurs de paramétrages sur les prédictions de performances afin d'évaluer la sensibilité de notre méthode.

Chapitre 1

Contexte scientifique

« Les performances individuelles, ce n'est pas le plus important. On gagne et on perd en équipe »

Zinedine Zidane - 18 novembre 2001

L'évaluation de performances d'architectures de contrôle-commande est un domaine extrêmement vaste, c'est pourquoi dans ce premier chapitre, nous précisons le cadre méthodologique et technique dans lesquels nos travaux s'inscrivent.
Ainsi, nous définissons les systèmes cibles de l'architecture de contrôle-commande qui sont l'objet de notre étude, les besoins en évaluation de performances des « Architectures Opérationnelles » durant leur cycle de vie et les méthodes de modélisation retenues pour ce faire.
Enfin, nous montrons que les travaux et résultats de la communauté ne répondent pas à l'ensemble des besoins et qu'il est donc nécessaire de proposer une démarche originale d'évaluation (objet du Chapitre 2) qui permette d'accompagner l'architecte tout au long du cycle de vie de l'architecture de contrôle-commande.

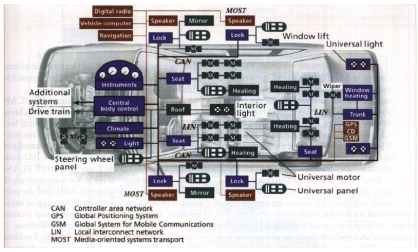
1.1 Problématique

1.1.1 Définitions générales

Dans les sociétés spécialisées dans le développement de systèmes automatisés complexes, on nomme souvent «architecte» l'ingénieur chargé de la conception et du développement de l'architecture de commande. Nous retiendrons cette dénomination pour la suite de ce mémoire.
Afin de définir ce que l'on entend par architecture de contrôle-commande, on se propose de présenter un panorama rapide des différents types d'architectures de commande que l'on est à même de rencontrer dans le monde industriel. Ce panorama peut être brossé en retenant trois familles d'architectures qui sont fonction du volume de production.

1.1.1.1 Les architectures produites en grandes séries

Une grande série est caractérisée par un grand nombre d'exemplaires (plusieurs milliers) ce qui permet l'utilisation de composants spécifiques (cartes électroniques, calculateurs industriels dédiés, …) à fonctions ciblées et à faible coût. En général, la partie matérielle de ce type d'architecture est réalisée en Codesign [ERNST 1998] et la partie logicielle est souvent développée en assembleur ou en langage C [ISO/IEC 9899].
On peut citer dans cette catégorie les systèmes de contrôles embarqués en automobile [CAVALIERI & al 1996](fig. 3).



Réseaux de contrôle-commande embarqués sur automobile

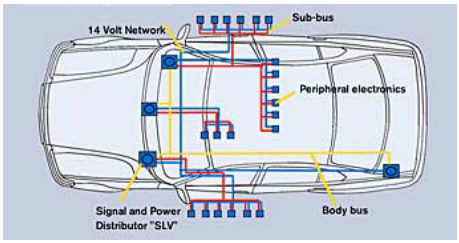


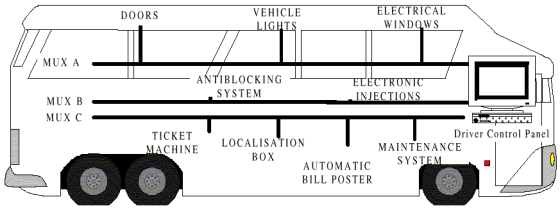
Figure 3 : Exemples d'architectures de commande produites en grande série

1.1.1.2 Les architectures produites en moyennes et petites séries

Une moyenne ou petite série est caractérisée par un petit volume de production (d'une dizaine à quelques centaines d'exemplaires) qui amène souvent à retenir l'utilisation conjointe de composants standards (Bus CAN [ISO 11898:1993]) et de composants spécifiques (cartes électroniques propriétaires) et à l'utilisation de langages de programmations d'informatique générale (langage C) ou industriel (langages d'automates programmables industriels [ISO/IEC 1131-3]).
On peut citer dans cette catégorie, les systèmes de contrôle-commande ferroviaires (embarqués ou au sol) dans le ETCS (European Train Control System) [JANSEN & al 1998] ou les systèmes de commande d'autobus embarqués [HUMBERT & al 1997] [HUMBERT & al 1998](fig. 4).

1.1.1.3 Les architectures produites de façon unitaire

Une production unitaire souvent implique une utilisation quasiment exclusive d'éléments standards, que ce soit au niveau matériel (Automates Programmables Industriel, Réseaux Locaux Industriels, …) ou au niveau logiciel (langages de la norme IEC 61131-3 [ISO/IEC 1131-3]).



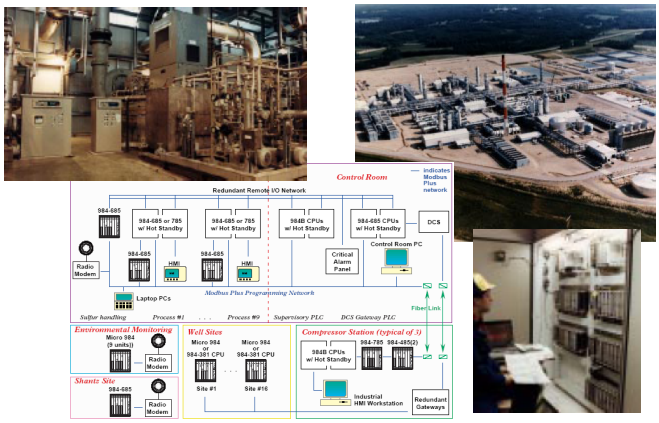
Commande embarquée sur autobus
[HUMBERT & al 1998]

Commandes ferroviaires embarquées ou au sol
[JANSEN & al 1998]

Figure 4 : Exemples d'architectures de commande produites en petites ou moyennes séries

On parlera de production unitaire par exemple pour les systèmes de contrôle-commande d'usines [KOWALEWSKI & al 1999], le bâtiment [MATEOS & al 2001] ou bien encore des systèmes plus spécifiques comme l'acquisition de données [BARANA & al 2002] ou accélérateurs de particules [SOUMBAEV & al 2002](fig. 5).

Contrôle-commande d'un moulin
[SCHNEIDER 2000]



Contrôle-commande d'une installation de production de gaz
[SCHNEIDER CA 2000]

Figure 5 : Exemples d'architectures de commande produites de manière unitaire

1.1.1.4 Synthèse

La figure 6 représente une vue synthétique des différentes familles d'architectures de contrôle-commande classifiées selon le volume de production.

Nombre d'exemplaires à fabriquer	**grande série**	**moyenne et petites séries**	**unitaire**
Type des composants utilisés	**spécifique**		**standard**
Exemples de systèmes concernés	- Automobile	- Autobus - Ferroviaire - Avionique	- Manufacturier - Bâtiment - Agro-alimentaire - Production d'énergie

Figure 6 : Familles de cibles possédant un contrôle-commande

Notre travail se concentre sur les architectures de contrôle-commande produites en petites séries ou unitaires, constituées de composants standards. En effet, comme nous allons le montrer dans les pages qui suivent, d'assez nombreux travaux de recherche ont porté sur l'optimisation et le développement d'architecture de contrôle-commande de grandes séries mais peu se sont intéressés au domaines des petites séries et de l'unitaire.

1.1.2 Place de l'évaluation de performance dans le cycle de vie d'architecture de contrôle-commande

Le besoin en évaluation de performance se place souvent après la phase de conception comme nous le montrent les études réalisées sur les architectures de contrôle-commande.

Dans les travaux de Bruno DENIS [DENIS & al 1993] portant sur la conception des architectures de contrôle-commande, l'évaluation de performances est placée à la fin de la procédure de conception. Il utilise la carte des activités de développement d'un système automatisé pour représenter l'emplacement de ses travaux (fig. 7).

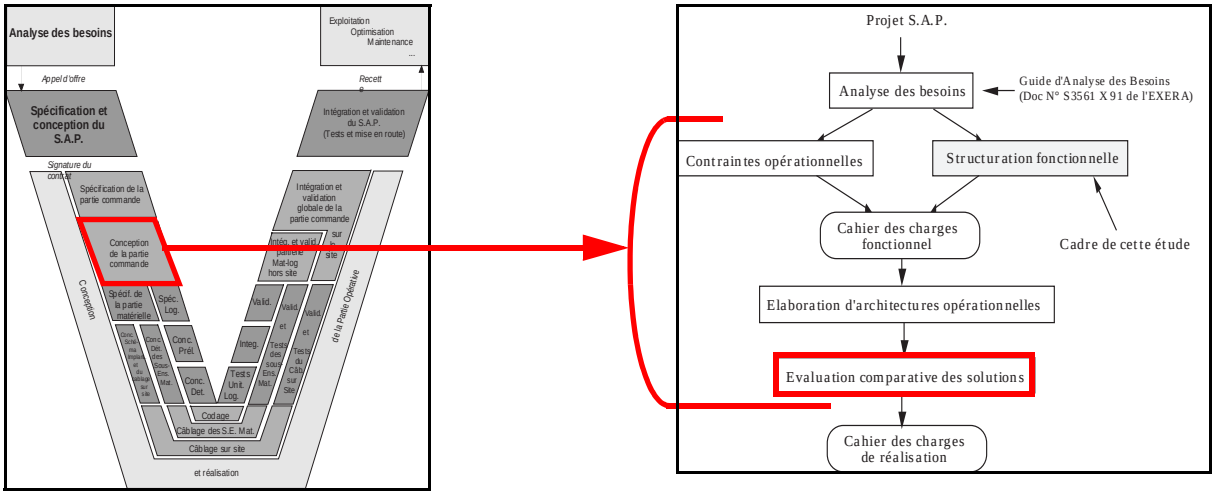


Figure 7 : Carte des activités de développement d'un système automatisé [DENIS & al 1993]

Dans le cadre des systèmes embarqués, nous trouvons par exemple les travaux de Françoise Simonot-Lion et Jean-Pierre Elloy [ELLOY & SIMONOT-LION 2002]. Ils proposent entre autre une description du cycle de développement de l'architecture de contrôle-commande (fig. 8).

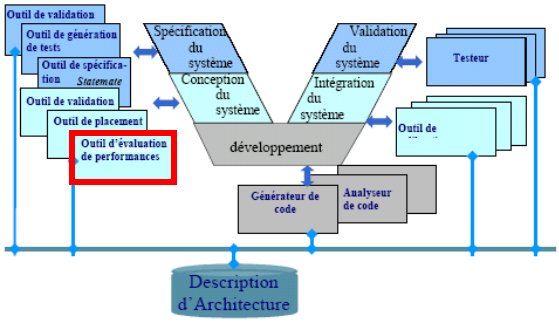


Figure 8 : Activités du cycle de développement selon [ELLOY & SIMONOT-LION 2002]

Cette description est très proche de celle de la carte de développement pour un Système Automatisé de Production (SAP). Elle place les outils d'évaluation de performance durant la phase de conception du système, donc de la conception du contrôle-commande en début de cycle, juste avant la phase de développement.

Lorsque nous nous intéressons aux travaux de recherche de domaines scientifiques connexes, nous remarquons que l'on trouve d'autres descriptions des cycles de développement. Par exemple dans le cas du Codesign, on trouve des cycles de développement spécifiques [ERNST 1998] (fig. 9).

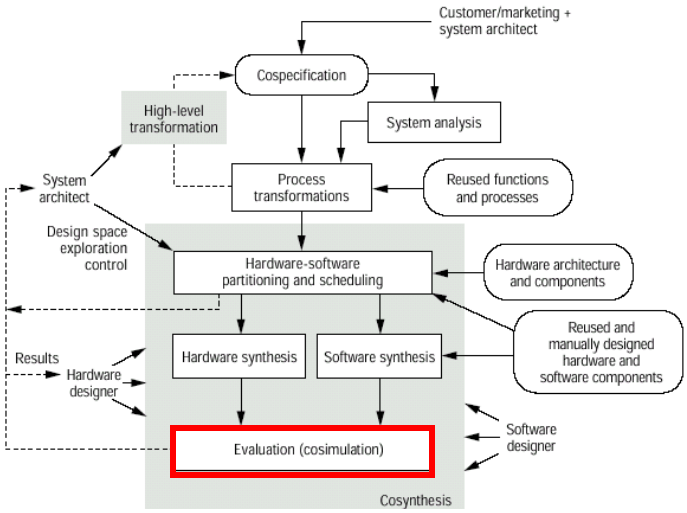


Figure 9 : Exploration de l'espace de conception des systèmes embarqués selon [ERNST 1998]

L'évaluation est dans ce cas obtenue par cosimulation de la partie matérielle et de la partie logicielle après partitionnement et ordonnancement entre le matériel et le logiciel.

On remarque que l'évaluation de performance est donc souvent placée immédiatement après la phase de conception de l'architecture de contrôle-commande. Or le besoin d'évaluation de l'architecte est présent tout au long du cycle de vie du Système Automatisé de Production (SAP) (fig. 10) :

- en début du cycle, l'architecte reçoit un appel d'offre pour la conception du SAP. Il doit en réponse définir, à partir d'une expression des besoins assez succincte, une ou plusieurs solutions d'architectures, et ce dans un laps de temps assez court. Il doit alors pouvoir évaluer si ses solutions répondront aux exigences du cahier des charges ;
- après signature du contrat, le cahier des charges détaillé est fourni par le client et l'architecte doit réaliser la conception de détail de l'ensemble du SAP (donc de l'architecture de contrôle-commande). L'architecte doit pouvoir alors vérifier si les performances de sa solution répondent bien aux besoins du cahier des charges détaillé ;
- lors de l'installation du SAP, l'architecte doit réaliser l'intégration et la validation globale du SAP. Le paramétrage des différents équipements de l'architecture est souvent complexe et l'architecte doit pouvoir évaluer s'ils n'auront pas un impact négatif sur les performances attendues ;
- Une fois le SAP en exploitation, l'architecte peut être amené à optimiser les performances de son installation. Un changement de paramétrage, un renouvellement des équipements, ou une évolution de l'implantation peuvent améliorer ou dégrader les performances. L'architecte doit donc pouvoir évaluer les performances qu'impliqueront ses modifications.

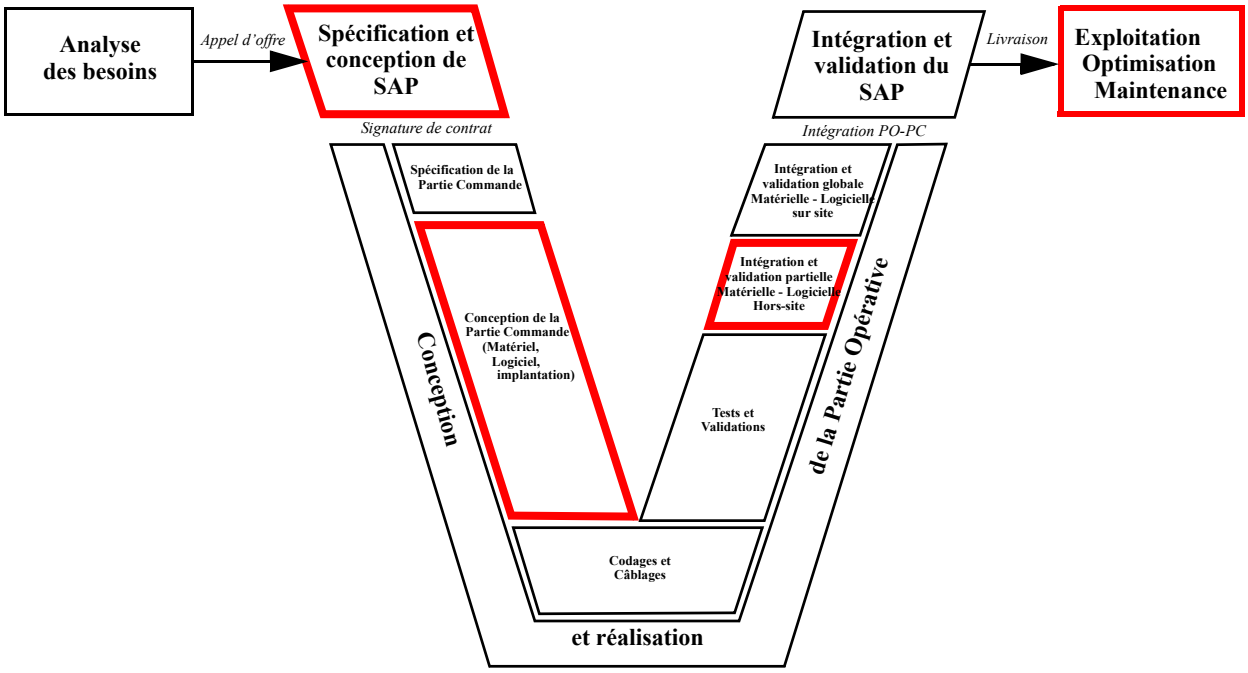


Figure 10 : Besoins en évaluation durant le cycle de vie d'un SAP

Nous pouvons conclure que l'architecte a besoin d'un outil d'évaluation de performance qui pourra l'accompagner durant les différentes phases du cycle de vie de son architecture et non uniquement après la phase de conception. Cela implique d'avoir un outil adaptatif, car les attentes de l'architecte sont différentes suivant les phases du cycle de vie où il se trouvera.

1.1.3 Contraintes à satisfaire par une architecture de contrôle-commande

Afin de déterminer quels sont les contraintes que devra satisfaire une architecture de contrôle-commande, nous nous appuierons sur les travaux de Joseph SIFAKIS [SIFAKIS 2002] qui nous indique que les critères à respecter sont :

- Fonctionnalité ;
- Qualité : Robustesse et Performances ;
- Contraintes matérielles (poids, encombrement, …) ;
- Coût global, consommation électrique ;
- Délai de mise sur le marché.

Il définit son objectif comme *«Construire des systèmes de fonctionnalité, de qualité déterminée et garantie et à coût acceptable, ce qui est un défi technologique et scientifique majeur»*.
Dans le cadre de notre travail, nous retiendrons cet objectif, en portant une attention toute particulière aux critères de qualité :

- performances temporelles : temps de réponses (mini, maxi, moyen, …)
- robustesse en terme de Sûreté de Fonctionnement (SdF) : disponibilité, fiabilité, MTBF, …

1.1.4 Le paradoxe de l'architecte

L'architecte devant concevoir une architecture de commande doit affronter le paradoxe suivant [DENIS 1994] :

- étudier et formaliser *suffisamment en détail* les besoins du client afin de lui proposer une solution *pertinente* (c'est-à-dire pressentir avant la conception une grande partie du travail de conception), avec des coûts calculés au plus juste dans l'espoir d'emporter le marché ;
- étudier *suffisamment sommairement* l'appel d'offre pour ne pas engager trop de frais d'étude sur une affaire dont la société d'ingénierie n'a pas encore la charge contractuelle.

Étant donnée que la position de la conception d'architecture de commande est en début de cycle du vie (fig. 7) et que le client ne fournit qu'une description sommaire et incomplète de son cahier des charges, ce paradoxe est très difficile à résoudre. En règle générale, seule l'expérience de l'architecte permet de contourner ce paradoxe : l'expertise issue des cas d'études déjà traitées lui permet de proposer une solution suffisamment fine à l'issue d'une pré-étude rapide. L'expertise n'est toutefois pas une solution satisfaisante au paradoxe de l'architecte, car elle ne permet pas toujours de faire face :

- à la complexité croissante des systèmes de production qui se traduit par une augmentation importante de leur taille, des fonctions de commande à mettre en oeuvre ;
- aux évolutions rapide du marché de l'offre en équipements de régulation et d'automatisme ;
- au manque de temps alloué pour la phase d'avant projet.

Des méthodes et des outils d'assistance à la conception et à l'évaluation de solutions d'architectures ont été développés pour aider l'architecte. Ces méthodes permettent d'améliorer la *qualité* des architectures conçues, la *réactivité* face à de nouveaux problèmes et la *productivité* dans la phase d'avant projet. Afin de faire le point sur ces méthodes et outils, nous présenterons dans les paragraphes suivants un état de l'art portant sur la formalisation de la conception d'architecture de contrôle-commande ainsi que sur l'évaluation de performances, en s'intéressant aux différentes approches d'évaluations ainsi qu'aux différentes performances étudiées.

1.2 Etat de l'art sur la formalisation de la conception de contrôle-commande

On remarque que les termes employés pour formaliser une architecture de contrôle-commande varient selon les différentes équipes de recherches. Par exemple, dans les travaux du projet AEE (Architecture Électronique Embarquée) [ELLOY & SIMONOT-LION 2002], l'architecture de contrôle-commande est présentée selon cinq niveaux d'abstraction. Chaque niveau modélise un point de vue particulier concernant une architecture embarquée. Les niveaux dit «hauts» représentent les aspects projet ou fonctionnel alors que les niveaux dit «bas» sont associés au matériel (calculateurs, réseaux, …) ou au logiciel. Enfin, un dernier niveau dit «opérationnel» représente l'allocation d'éléments logiciels aux équipements matériels (fig. 11).

On retrouve des éléments similaires dans les travaux de Cavaliere [CAVALIERE 2002] du LORIA groupe TRIO, bien que la terminologie Système d'Information soit utilisée pour le contrôle-commande. En effet, ces travaux ont été réalisés en collaboration avec le groupe PSA. Ils doivent donc prendre en compte les notes techniques internes au groupe PSA. Celles-ci définissent ainsi l'activité de conception

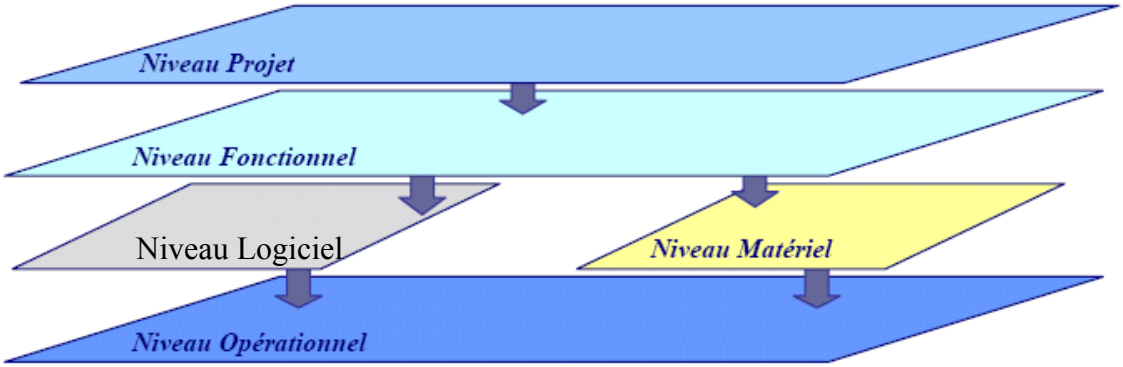


Figure 11 : Niveaux d'abstraction d'une architecture selon [ELLOY & SIMONOT-LION 2002]

de l'architecture du Système d'Information : la conception d'architecture consiste à partitionner puis répartir une spécification fonctionnelle sur une architecture matérielle (automates programmables industriels, calculateurs industriels, réseaux de terrain, …). Cette projection d'une partition de spécifications fonctionnelles sur une architecture matérielle est appelée architecture opérationnelle.

Dans les travaux portant sur la synthèse de communication des systèmes embarqués mené au Departement of Computing and Software Systems de l'University of Washington par Ross B. Ortega en association avec Gaetano Borriello [ORTEGA & BORRIELLO 1998] [ORTEGA & BORRIELLO 1997], on retrouve la phase où l'architecte système projette les spécifications comportementales de processus communicants sur une architecture (qui correspond dans ce cadre à la partie matérielle du système). Les auteurs mettent d'ailleurs en évidence que les communications entre deux processus se retrouvent projetées au travers de plus d'un réseau de communication (fig. 12).

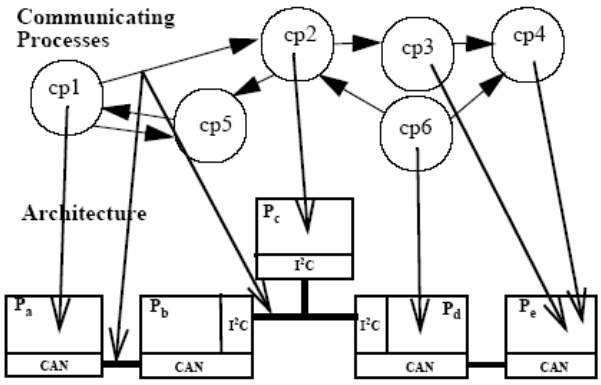


Figure 12 : Affectation de spécifications comportementales à une architecture selon [ORTEGA & BORRIELLO 1998]

Dans les travaux portant sur l'aide à la conception d'architecture de conduite des systèmes de production menés au LURPA par Bruno DENIS [DENIS 1994] [DENIS 1995], on retrouve la même démarche mais avec des termes différents. En effet le modèle d'implantation (spécifications fonctionnelles) est associé au modèle organique (partie matérielle) pour former le modèle d'affectation qui subira plusieurs évaluations afin de déterminer l'architecture matérielle finale, ainsi que l'affectation de chaque modèles des spécifications aux différents équipements (fig. 13).

Nous avons retenu une modélisation de l'architecture de contrôle-commande cohérente avec les diverses approches de la communauté scientifique. Dans la suite de nos travaux : une Architecture Opéra-

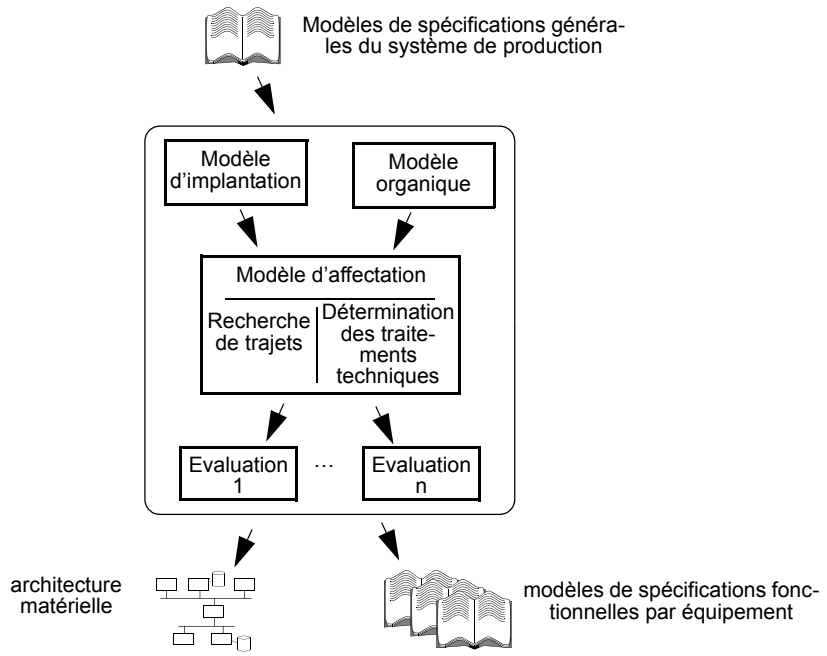


Figure 13 : Démarche de conception d'architecture de contrôle-commande d'après [DENIS 1995]

tionnelle (AO) est le résultat de la composition d'une Architecture Fonctionnelle (AF) et d'une Architecture Matérielle (AM). Dans la suite de ce chapitre, nous décrivons ces trois niveaux d'abstraction.

1.2.1 Architecture Fonctionnelle

Nous utiliserons la définition de Françoise Simonot-Lion [SIMONOT-LION 1999], pour qui une Architecture Fonctionnelle est un modèle de la structure et du comportement des fonctions de l'application. L'Architecture Fonctionnelle est à différencier de l'Architecture Logicielle [BASS & al 1997] [IEEE P1471/D5.3] [LEE 2002] qui s'apparente à la traduction des fonctions à implanter dans l'Architecture Matérielle. Nous définissons pour notre part une Architecture Fonctionnelle (AF) comme un *modèle formel de la structure et du comportement des données et des activités du système automatisé*, précisant ainsi l'importance accordées aux données dans nos travaux.

Nous travaillons sur des activités élémentaires appelées «atomes». Chaque atome est caractérisé par les services qu'il rend au sein du système de commande. La structure représente l'ensemble des activités ainsi que les relations entre les données des atomes (appelé également échange de données).
Un atome est la plus petite entité non décomposable de l'ensemble des traitements du système d'automatisation. Il est caractérisé par trois propriétés : il réalise une *fonction* (vue interne), à laquelle est associé un *service rendu* (vue externe) par une *application biunivoque*; il n'est pas réparti sur plusieurs machines et on ne peut pas en demander une exécution partielle.

L'Architecture Fonctionnelle est unique et ne dépend que du cahier des charges et des contraintes opérationnelles fournies à l'architecte. Pour la description d'architecture fonctionnelle nous avons retenu le formalisme SART de Hatley et Pirbhai [HATLEY & PIRBHAI 1987] qui a été développée à partir des travaux de Ward et Mellor [WARD & MELLOR 1985]. Ce langage fonctionnel, permettant de spécifier un système embarqué temps réel [ALKHODRE & al 2002] convient tout à fait à nos besoins.
Ce choix ne doit cependant pas être vu comme une restriction des méthodes pouvant être prise en

compte dans notre démarche, il doit seulement être interprété comme un choix nécessaire à l'illustration de nos travaux. Toutefois, ce choix se veut représentatif des modèles de spécification couramment utilisés, dans la mesure où il aborde à la fois les aspects fonctionnels, les aspects dynamiques et les aspects de structuration des données du système de production spécifié.

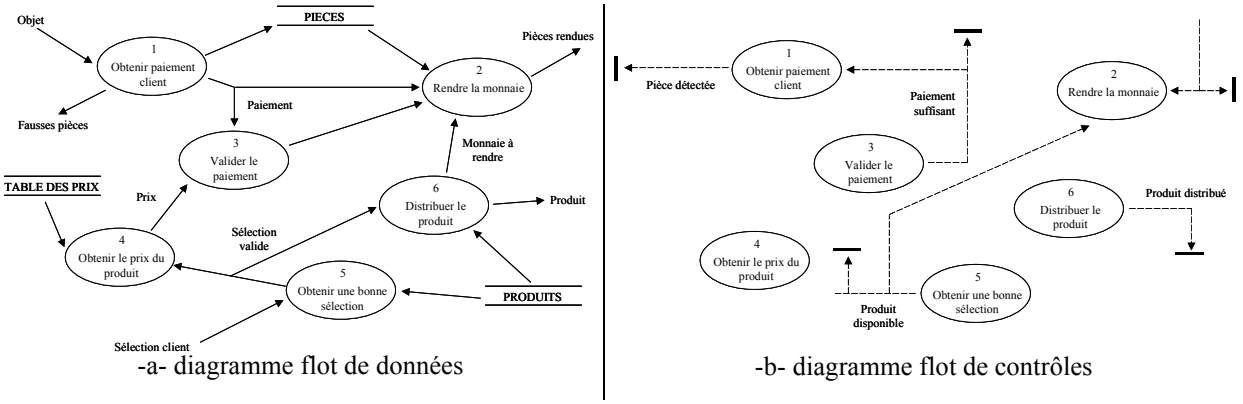


Figure 14 : Exemple de modèle SA-RT d'Architecture Fonctionnelle

La figure 14, tirée d'un Cours d'Alain VAILLY [ANDRE & VAILLY 2001] enseignant à l'IUP MIAGE de l'Université de Nantes, représente les deux points de vue d'un modèle SA-RT : le diagramme flot de données (fig. 14a) qui décrit les données qui sont échangées entre les différentes tâches à réaliser par le système, et le diagramme flot de contrôles (fig. 14b) qui traite les données de contrôles. On peut remarquer que les deux diagrammes partagent les mêmes tâches fonctionnelles. La description arborescente des fonctions permet la description grossière d'une Architecture Fonctionnelle avec peu de niveau au début du processus de conception de l'architecture de contrôle-commande, tandis qu'en phase de réglage de l'architecture, l'architecte peut disposer d'une description plus fine avec davantage de niveaux de description. L'Architecture Fonctionnelle, composée d'atomes qui caractérisent les fonctions attendues du système, est donc une description formelle des fonctions attendues de la commande.

1.2.2 Architecture Matérielle

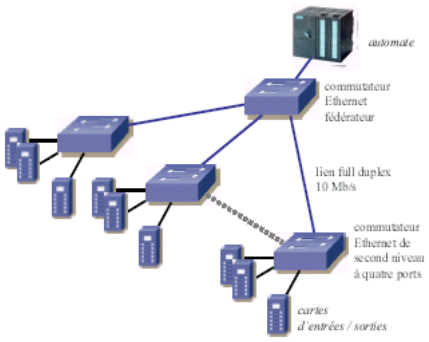
Nous reprendrons la définition de Cavaliere [CAVALIERE 2002] qui définit une Architecture Matérielle (AM) comme un modèle de l'architecture informatique supportant l'Architecture Fonctionnelle (AF). C'est une description de l'ensemble des supports matériels constituant le Système d'Information du Système Automatisé de Production, c'est-à-dire :

- l'ensemble des calculateurs (calculateurs industriels, automates programmables industriels, …) munis de leur propre système d'exploitation et de leurs interfaces d'entrées-sorties ;
- l'ensemble des moyens de communication les connectant munis des protocoles nécessaires (réseaux de terrains, réseaux d'ateliers, liaisons point à point, …) ;
- la définition des connexions des calculateurs sur les moyens de communication (topologie de l'architecture).

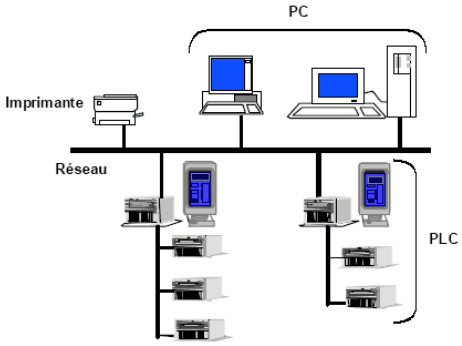
Une Architecture Matérielle est le résultat du choix des équipements réalisé par l'architecte (machines, réseaux, exécutifs et protocoles), du dimensionnement de ces matériels et de la définition des points de connexion.
D'après la position de nos travaux dans le cycle de développement, ces choix ne sont pour la plupart pas

encore finalisés. Des choix partiels peuvent avoir été faits, mais il reste souvent plusieurs choix possibles de matériels, de dimensionnement ou de définitions de points de connexion. On a donc, pour un cahier des charges donné, non-unicité de l'Architecture Matérielle, contrairement à l'Architecture Fonctionnelle.

De plus, on peut remarquer une absence de formalisme «consacré» à la représentation d'une Architecture Matérielle. On peut trouver ainsi des représentations «réalistes» comme sur les figure 15a et figure 15b mais également des représentations «schématiques» comme sur les figure 16a et figure 16b.

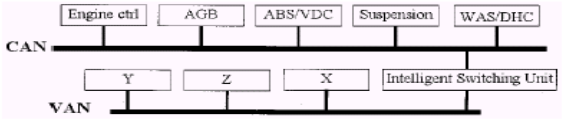


a - Etude de réseau Ethernet commuté [GEORGES & al 2003a]

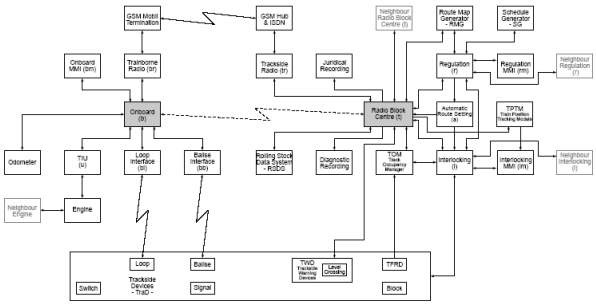


b - Exemple d'architecture de SI [CAVALIERE 2002]

Figure 15 : Exemples de représentations réalistes d'Architectures Matérielles



-a- Architecture Matérielle Multi-réseaux [CASTELPIETRA & al 2002]



-b- Architecture des commandes des systèmes ferroviaires de l'ETCS [JANSEN & al 1998]

Figure 16 : Exemples de représentations schématiques d'Architectures Matérielles

Afin de nous rapprocher des usages de représentation du modèle matériel, nous avons recherché s'il existait une représentation normalisée. Seule la norme française NF E 04-203-3 intitulée «Régulation, mesure et automatisme des processus industriels : représentation symbolique», nous propose pour quelques organes de traitement une représentation graphique (tableau 1).

Tableau 1 : Représentation graphique des équipements selon NF E 04-203-3

Dénomination	Symbole
Instrument individuel	◯
Calculateur de processus	⬡
Calculateur de supervision	⬡
Automate	◇
Système d'affichage ou de commande en temps partagé	▢

En nous appuyant sur cette norme, nous pouvons améliorer la formalisation de la représentation d'une Architecture Matérielle en apposant à coté de chaque composant son symbole (fig. 17).

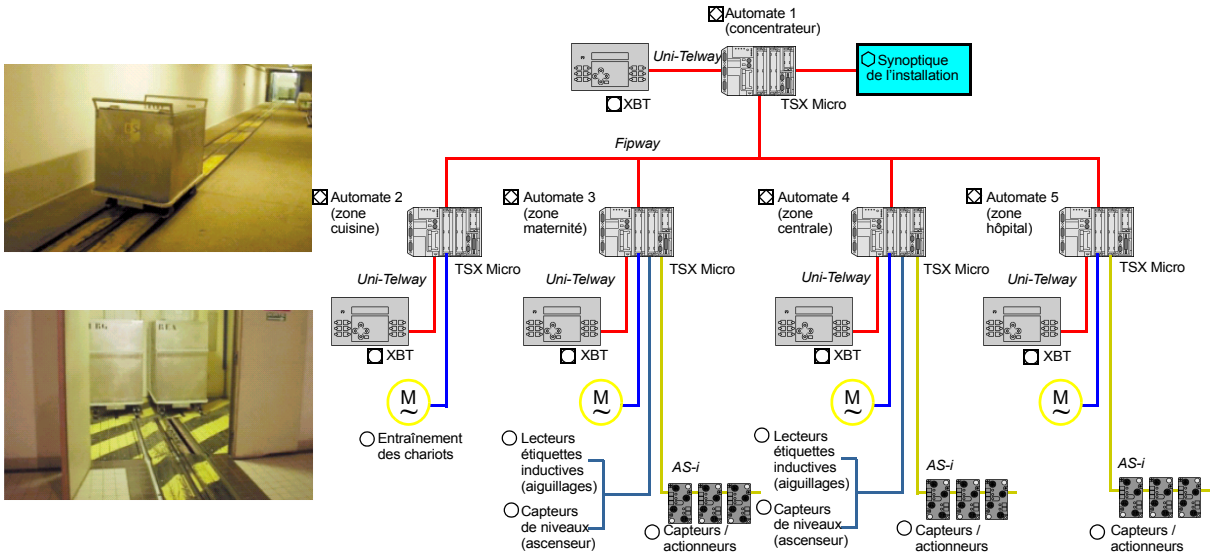


Figure 17 : Architecture Matérielle contrôlant les chariots de l'hôpital de Dreux [SCHNEIDER 2003]

Une Architecture Matérielle est donc le réseau d'interconnexion de l'ensemble des composants matériels de contrôle-commande ainsi que leurs interconnexions.

1.2.3 Architecture Opérationnelle

Nous reprendrons là encore la définition de Cavaliere [CAVALIERE 2002] qui définit une Architecture Opérationnelle (AO) comme le modèle d'une projection (ou «mapping») de l'Architecture Fonctionnelle (AF) sur une Architecture Matérielle (AM).
Cette projection consiste à allouer une ressource dans un équipement de l'Architecture Matérielle à chacun des «atomes» (fonction attendue ou fonction mémoire) de l'Architecture Fonctionnelle. Cette projection est une injection et non une bijection, car dans le cas de systèmes à commandes redondantes une même fonction peut avoir de multiples emplacements. Toutefois, nous poserons l'hypothèse que chaque «atome» ne pourra être placé que dans un unique équipement.
Etant donné que les atomes peuvent avoir plusieurs placements possibles pour un couple AF/AM donné, l'Architecture Opérationnelle n'est donc pas unique. La figure 18 présente un exemple académique d'une possible Architecture Opérationnelle que l'on peut réaliser avec une Architecture Matérielle (en bas de la figure 18) et une Architecture Fonctionnelle (en haut de la figure 18) données.

La création d'une Architecture Opérationnelle, appelée aussi «mappage», est une phase très importante. En effet, les performances d'une Architecture Opérationnelle seront différentes selon la répartition des tâches fonctionnelles sur les équipements matériels. De nombreux travaux ayant trait à l'automatisation ou à l'optimisation de ce «mappage» sont réalisés dans la communauté scientifique. Nous citerons par exemple, les travaux portant sur :

- l'optimisation du «mappage» de l'architecture fonctionnelle sur l'architecture matérielle développé par [CHOU & BORRIELLO 2000] ;
- la synthèse de communication dans les systèmes répartis [ORTEGA & BORRIELLO 1998] [ORTEGA & BORRIELLO 1997] ;

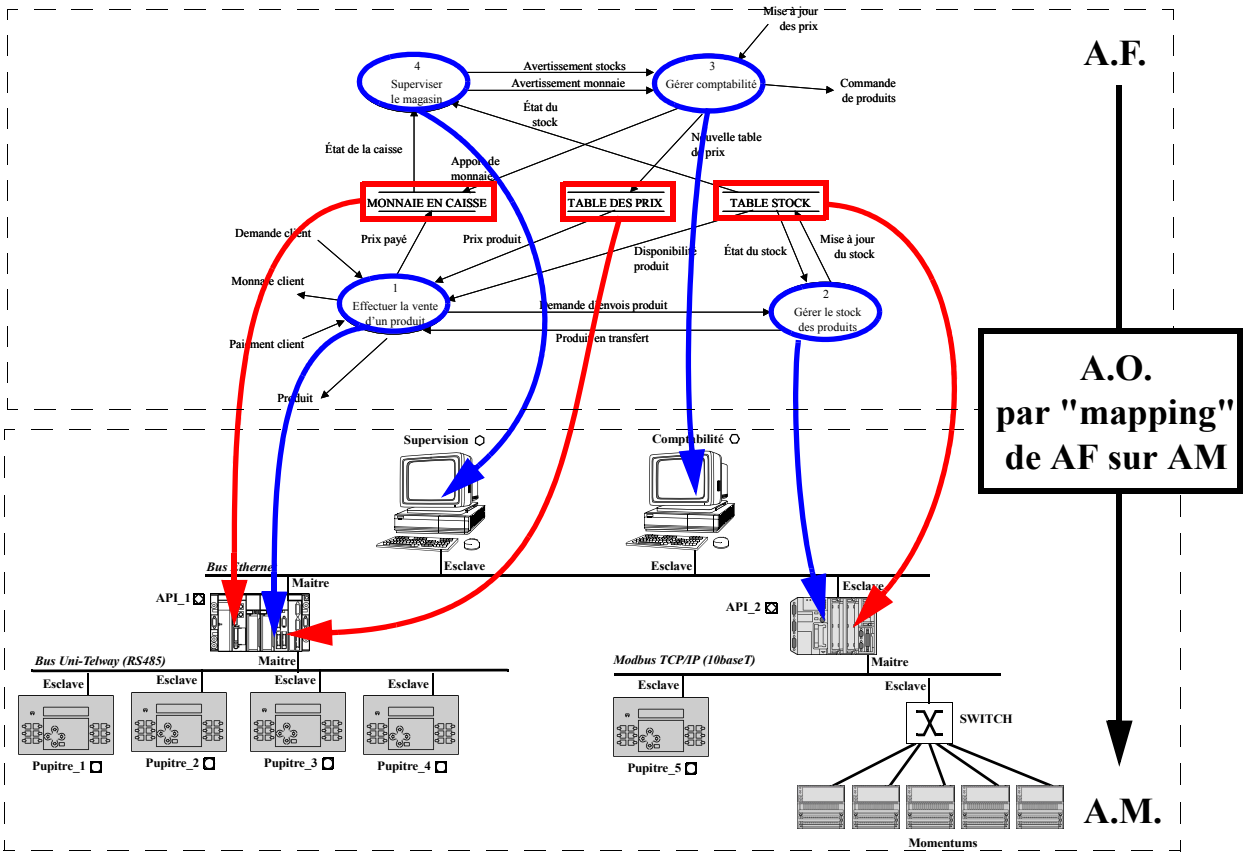


Figure 18 : Architectures Fonctionnelles, Matérielles et Opérationnelles

- le «mappage» de processus de contrôle embarqués en automobile équipée d'un bus de terrain IEC/ISA [CAVALIERI & al 1996] ;
- le placement et la validation dans les systèmes temps réel distribués [RICHARD & al 2003].

Une Architecture Opérationnelle est donc le modèle d'une projection possible de l'unique Architecture Fonctionnelle sur une des Architectures matérielles possibles.

1.3 Etat de l'art sur l'évaluation de performance d'architecture

L'évaluation de performance d'architecture de contrôle-commande s'avère être un vaste sujet. Afin d'orienter au mieux nos recherches, nous allons réaliser un panorama des études réalisées dans ce domaine.

Nous nous intéresserons dans un premier temps aux recherches effectuées en France dans le cadre de l'évaluation de performance d'architectures de contrôle-commande, puis nous élargirons notre panorama aux recherches menées en Europe et enfin dans le reste du monde.
Nous focaliserons notre attention sur ces recherches en nous intéressant :

- au type de performance étudiée (Sûreté de Fonctionnement : disponibilité, fiabilité, … ; temps de réponse ou retard de causalité);

- à la portée des études vis-à-vis de l'architecture (portée locale : une architecture simple ou un unique équipement ; ou portée globale : l'architecture dans son intégralité) ;
- à la manière dont est spécifié le comportement de l'architecture («statique» : définition du comportement par des lois de répartitions ou formules mathématiques ; comportement «dynamique» : définition du comportement de manière à réagir à un ensemble de stimulations).

Nous terminerons ce panorama par une synthèse des recherches que nous aurons présentées au travers de ce paragraphe.

1.3.1 Evaluation de performance d'architectures en France

1.3.1.1 Laboratoire du CRAN - Équipe SURFDIAG

Au sein de l'équipe SURFDIAG (SÛreté de Fonctionnement et DIAGnostic des Systèmes) du CRAN (Centre de Recherche en Automatique de Nancy), sous la direction de Didier MAQUIN, on trouve au cours des dernières années de nombreux travaux de recherche portant sur l'évaluation de fiabilité ou de la disponibilité des systèmes automatisés à commande distribuée.

Ainsi des travaux de Jean-François AUBRY portent dans un premier temps de nombreuses questions ouvertes sur la fiabilité des systèmes automatisés [AUBRY 2002]. Puis il réalise des travaux sur l'évaluation de la fiabilité de systèmes à commande distribuée au travers d'une approche basée sur les informations [JUMEL & al 2003] et participe à la conception de SILKEY, un logiciel pour l'évaluation automatique de la sûreté et la disponibilité d'architectures redondantes multi-niveaux [HAMADI & al 2003]

De plus, au sein de la même équipe, Pavol BARGER a travaillé sur la modélisation et l'analyse de critères de sûreté de protocoles de communications réelles [BARGER & al 2003a] ainsi que sur l'analyse de sûreté et l'estimation de fiabilité de systèmes de contrôle utilisant des réseaux [BARGER & al 2003b]. Il a soutenu une thèse portant sur l'évaluation et la validation de la fiabilité et de la disponibilité des systèmes d'automatisation à intelligence distribuée en phase dynamique [BARGER 2004].

1.3.1.2 Laboratoire du CRAN - Équipe SYMPA

Dans l'équipe SYMPA (SYstèMes de Production Ambiants) du CRAN, dirigée par Thierry DIVOUX, les travaux de recherche menés sur les architectures Ethernet commuté, ont amené un besoin en évaluation de performances. Que cela soit par calcul ou par simulation, les performances évaluées sont des délais maximum de traversée d'un ensemble de commutateurs Ethernet.

On remarque les travaux de Jean-Philippe GEORGES, Eric RONDEAU et Thierry DIVOUX portant sur les architectures Ethernet commutés. On distinguera des évaluations par calculs et simulations [GEORGES & al 2003c] [GEORGES & al 2003b] ainsi que l'évaluation des délais maximums de traversée d'un ensemble de commutateurs Ethernet [GEORGES & al 2003a].

1.3.1.3 Laboratoire LAAS - Équipe TSF

Au sein du LAAS (Laboratoire d'Analyse et d'Architecture des Systèmes), dans l'équipe TSF (Tolérance aux Fautes et Sureté de Fonctionnement Informatique) sous la direction de Malik GHALLAB, de nombreux travaux ont été menés sur l'évaluation de la sûreté de fonctionnement. Ces travaux ont été aidés par la mise en place de SURF-2 [SURF2 1994], un outil d'évaluation de la sûreté de fonctionne-

ment des systèmes matériels et logiciels basée sur la construction rigoureuse, la validation et la résolution numérique de modèles markoviens et réseaux de Petri stochastiques. Plus récemment Karama KANOUN a mené de nombreux travaux afin de modéliser et d'évaluer la disponibilité de systèmes de contrôle-commande.

Ainsi avec Nicolae FOTA [FOTA & al 1999], elle présente une évaluation de la disponibilité d'une partie du réseau de contrôle aérien (CAUTRA) à l'aide de modèles GSPN (Generalized Stochastic Petri Nets) de chaque composant de l'architecture.

Puis avec Cláudia BETOUS [BETOUS & KANOUN 2004a] [BETOUS & KANOUN 2004b], elle présente la modélisation ainsi que la comparaison de différentes architectures de contrôle-commande, toujours selon le point de vue de la disponibilité. Ces architectures comportent de nombreuses interfaces homme-machine, de nombreuses unités de traitement ainsi que de nombreuses interfaces de communications, l'ensemble étant interconnecté par un réseau local.

1.3.1.4 Laboratoire du LAAS - Équipe OLC

En parallèle, dans l'équipe OLC (Outils Logiciels pour la Communication), Guy JUANOLE travaille sur les systèmes distribués temps réel et les applications de contrôle-commande. Dans ce cadre, il s'intéresse à des études sur l'analyse de l'influence de la qualité de service (QoS) au niveau exécutif et réseau sur les performances d'applications de contrôle-commande par l'utilisation de réseaux de Petri stochastiques [JUANOLE 2002]. Ceci l'a amené à travailler sur l'évaluation de performances de protocoles de communications [ABDELLATIF & JUANOLE 2003b] ainsi qu'à l'étude d'algorithmes améliorant le flux d'admission pour les réseaux «rate-controlled packet» [ABDELLATIF & JUANOLE 2003a].

1.3.1.5 Laboratoire du LORIA/INPL - Équipe TRIO

Dans l'équipe TRIO (Temps Réel et InterOpérabilité) du LORIA/INPL (Laboratoire Lorrain de Recherche en Informatique et ses Applications) dirigée par Françoise SIMONOT-LION, de nombreuses études menées par SONG, ont portés sur l'étude de performances de chaînes critiques en évaluant des temps de réponse critiques de différents types de réseaux de communication.

Ainsi avec NAVET [NAVET & SONG 2001], sont étudiés dans un premier temps les performances des réseaux CAN en s'intéressant aux calculs de probabilité de manquer l'observation de la «worst-case deadline». Pour cela deux approches ont été employées : une approche analytique utilisant des formules mathématiques basées sur des chaînes de Markov pour les temps de réponse critiques, et une approche par simulation sur les plateformes OpNet et Qnap afin d'évaluer les temps de réponses moyens ainsi que la charge réseau.

Puis, avec WANG [WANG & al 2002], les temps de réponses de messages apériodiques dans des réseaux utilisant le protocole WorldFIP ont été étudiés. Pour cela, une modélisation par file d'attente des messages périodiques et apériodiques gérés par l'arbitre de bus WorldFIP a été réalisée.

En parallèle avec KOUBAA [KOUBAA & SONG 2002], est réalisée une évaluation de temps de réponse de commutateur Ethernet en se basant sur un exemple consistant à échanger des messages entre deux postes informatiques. L'ensemble étant basé sur une formulation probabiliste définissant les for-

mules mathématiques du nombre moyen de trames dans la file, ainsi que les temps moyens des trames de la file.

Enfin, la thèse de CAVALIERE [CAVALIERE 2002] définit DAMeSI, un profil UML pour l'évaluation de performances des systèmes d'automatisation distribués. Dans ces travaux, on retrouve des évaluation de temps de réponse de chaînes critiques pour de nombreuses architectures de petites tailles (2 PC + 1 réseau + 2 API ou 1 PC + 1 réseau + 1 API) mais aussi de l'atelier PSA AL4 de Valenciennes. Afin de pouvoir réaliser ces évaluations, CAVALIERE a réalisé la modélisation de fonctions de stimuli ainsi que des modèles de comportement des équipements pour gérer la manière dont les signaux sont traités.

1.3.2 Évaluation de performance d'architectures en Europe

1.3.2.1 Université de Padova (ITALIE)

Dans le laboratoire IEIIT (Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni), on retrouve des travaux portant sur l'évaluation des performances temporelle dans les réseaux.

Dans un premier temps, Stefano VITTURI porta son attention sur les réseaux IEC & profibus [VITTURI 2000] sur lesquels il réalisa une évaluation des temps de cycle réseau en faisant varier les protocoles utilisés par le biais de formules mathématiques modélisant le temps de cycle du bus, le temps de transmission des jetons ainsi que le temps global d'indisponibilité des jetons.

Ensuite, StefanoVITTURI s'est intéressé à l'utilisation du réseau Ethernet dans les niveaux bas d'une usine [VITTURI 2001]. Pour cela, il a dans un premier temps réalisé une comparaison entre le Profibus DP et un protocole maître-esclave sur Ethernet ; puis dans un second temps une comparaison entre WorldFIP et un protocole producteur consommateur sur Ethernet. Ses critères de comparaison furent les temps de cycles réseaux ainsi que l'efficience du codage des données. Cette étude portant sur les transmissions au niveau d'un unique réseau, utilisa tout comme son étude des réseaux IEC, des approches analytiques lui permettant de calculer ses critères selon des paramètres choisis.

Plus récemment, StefanoVITTURI s'est intéressé à la différence de traitement entre un Automate Programmable Industriel et un PC [VITTURI 2003]. Afin de pouvoir déterminer quelle solution était la plus intéressante, il a choisi d'évaluer le temps de réaction de la partie commande suivant l'équipement de traitement choisi. Pour cela, dans un premier temps, il a calculé les temps de transmission minimum et maximum attendus à l'aide de formules mathématiques puis dans un deuxième temps, il a réalisé une mesure d'une trentaine d'opérations sur le système réel afin d'évaluer les performances temporelles.

1.3.2.2 Université de Porto (PORTUGAL)

Issus de deux départements différents de la même université, Eduardo TOVAR (Department of Computer Engineering) et Francisco VASQUES (Department of Mechanical Engineering) se sont réunis pour réaliser des recherches communes.

Ils se sont, dans un premier temps, intéressés à l'étude du réseau Profibus avec un protocole «timed-token» afin de supporter les communications temps-réel des systèmes contrôlés par des ordinateurs distribués [TOVAR & VASQUES 1999a]. Ils ont pour cela calculé les temps de réponse critiques pour des messages à haute priorité.

Dans un second temps, ils se sont intéressés à l'analyse des temps de réponse de messages devant transiter par les multiples réseaux P-NET d'une commande distribuée temps réel [TOVAR & VASQUES 1999b]. L'évaluation de ces temps de réponse a été réalisée par calcul.

Dernièrement, ils ont porté leur attention sur l'approche temps-réel utilisant des réseaux WorldFIP [TOVAR & VASQUES 2001]. Leur étude a amené à l'évaluation de temps de réponses critiques pour une table définissant le comportement d'arbitrage de bus WorldFIP.

1.3.2.3 Université de Catania (ITALIE)

Au sein de l'institut d'Informatique et de Télécommunication, on trouve les travaux de CAVELIERI portant sur l'étude de Bus de terrain.

Dans une première approche, Salvatore CAVALIERI s'est intéressé à l'évaluation des performances en termes de fiabilité de réseaux de terrain IEC/ISA [CAVALIERI & al 1999]. L'étude a porté sur les retards d'attente de processus asynchrones en fonction de leur priorité dans le cas où le système de communication par bus de terrain se comporte sans erreur. Pour cela, il a modélisé le comportement avec des signaux continus, puis a réalisé un traitement et une reconstruction de ces signaux afin d'obtenir le signal de sortie. La simulation étant assurée par la méthode de Monte Carlo. Ainsi il pu obtenir les longueurs moyennes des files d'attente pour des messages asynchrones en présence de bruits.

Dans un deuxième temps, Salvatore CAVALIERI a modélisé un réseau Profibus DP ayant plusieurs maîtres dans le but d'évaluer les pires temps de réponse pour la transmission de messages [CAVALIERI & al 2002]. Pour cela, il a défini le comportement de chaque maître au travers de scenarii.

1.3.3 Évaluation de performance d'architectures dans le monde

1.3.3.1 Université de Washington - Seattle (USA)

Au sein du departement of Computer Science & Engineering, Gaetano BORRIELLO réalise des études portant sur les systèmes de commande distribués embarqués.

On retrouve ainsi une étude portant sur la Co-simulation de modèles dynamiques de communication dans les systèmes embarqués [HINES & BORRIELLO 1997] afin d'évaluer des performances de temps de transfert de messages au travers des architectures de commande. L'architecture étudiée est dans un premier temps très simple (1 bus avec 2 composants), puis elle se complexifie (1 bus, 1 mémoire partagée, 1 processeur, 1 interface d'Entrée/Sortie et des liaisons avec la Partie Opérative). Les performances sont étudiées sur le logiciel de cosimulation PIA en réalisant un envoi de 64 pages de données, soit un équivalent de 2^{19} octets.

Gaetano BORRIELLO a aussi étudié le problème de synthèse de communications dans le cas des systèmes distribués embarqués[ORTEGA & BORRIELLO 1998]. Afin de pouvoir valider les résultats de ses synthèses, il a évalué les performances de deux architectures ayant deux structures de communication : la première est composée de 5 Automates Programmables Industriels (API) connectés sur un réseau CAN ; la seconde est composée de 5 API répartis sur deux réseaux CAN et interconnectés par un réseau I²C. Cette évaluation a été réalisée par co-simulation de 7 fichiers de données.

1.3.3.2 Université Nationale de Pusan (Corée du sud)

Au sein de l'école de Mechanical Engineering & RIMT, Kyung Chang Lee et Suk Lee ont étudié l'évaluation de performances de réseaux Ethernet avec switchs dans le cade de communications temps-réel industrielle [LEE & LEE 2002]. L'architecture cible de leur étude a été un hub ou un switch connecté à une station maître et n stations esclaves. Les performances étudiées étant le maximum de temps de communication, elles ont été mesurées sur une plateforme de test en traitant environ 10 000 trames depuis une station esclave vers une station maître.

Ils ont ensuite porté leur études sur le développement de modèles analytiques de calculs afin d'évaluer les performances des délais de communication sur des réseaux Profibus-FMS selon un protocole à passage de jetons [LEE & al 2003]. Leur étude a utilisé des scénarios fixant les générations de messages de haut et bas niveaux ainsi qu'un confrontation avec des mesures sur une plateforme d'expérimentation.

Ils ont ensuite poursuivi par une étude de l'implantation et du réglage d'asservissement PID transitant sur un réseau Profibus DP [LEE & al 2004]. Cette étude met en jeu une modélisation analytique d'un bras robotisé en représentant un contrôleur et un moteur ainsi que deux générateurs de charge.

1.3.3.3 Université du Michigan (USA) - groupe IMPACT

Au sein de l'université du Michigan, le groupe interdisciplinaire IMPACT (Integrated Manufacturing Process Automation and Control Technologies, *www.eecs.umich.edu/~impact*) a mené des travaux de recherche sur le développement, implantation et d'optimisation de contrôle-commande de systèmes industriels. Un des trois domaines explorés dans ces travaux porte sur les bus de capteurs pour le contrôle-commande.
Feng-Li Lian a étudié sous la direction de James R. Moyne et Dawn M. Tilbury, l'évaluation de performances de contrôle-commande distribués en s'intéressant à divers types de réseaux : Ethernet, ControlNet et DeviceNet [LIAN & al 2001]. Ces travaux s'inscrivant dans le cadre de sa thèse portant sur l'analyse, la conception, la modélisation et le contrôle de systèmes distribués [LIAN 2001]. Les travaux suivant portèrent sur l'optimisation des performances de contrôle-commande distribués [OTANEZ & al 2002].
Les études réalisées portent sur un système composé d'une Partie Commande interconnectée avec la Partie Opérative par un unique réseau de communication.

1.3.4 Synthèse bibliographique sur l'évaluation de performances

Afin d'avoir une vue globale des travaux portant sur l'évaluation de performance d'architecture de contrôle-commande, nous reprenons les références que nous avons exposées dans les paragraphes précédents afin de pouvoir mettre en évidence le type de performances étudiées, la portée de l'évaluation réalisée ainsi que la manière dont le comportement de ces architecture a été modélisé (fig. 19).

En analysant la figure 19, on remarque que l'évaluation de performances est un domaine où les travaux de recherche de la communauté scientifique sont très variés. En effet, les travaux se partagent sur le type de performance évaluée (Sûreté de Fonctionnement ou temporelle).
Mais la portée de ces travaux est souvent réduite à une étude locale, portant uniquement sur un équipement de communication ou des architectures très simples comme le montre le nombre croissant de travaux sur l'utilisation des réseaux Ethernet en milieu industriel.
De plus les modèles retenus pour ces études ont souvent un comportement statique, c'est à dire que les

Labo	Equipe	Publications	Performance étudiée		Portée de l'évaluation		Type de modèle retenu	
			SdF	Temps	Locale	Globale	Statique	Dynamique
CRAN	SURFDIAG	[AUBRY 2002]	X					
		[JUMEL & al 2003]	X					
		[HAMADI & al 2003]	X					
		[BARGER 2004]	X					
		[BARGER & al 2003a]	X		X			
		[BARGER & al 2003b]	X		X			X
	SYMPA	[GEORGES & al 2003a]		X	X			
		[GEORGES & al 2003b]		X	X			
		[GEORGES & al 2003c]		X	X			
LAAS	TSF	[BETOUS & KANOUN 2004a]	X			X	X	
		[BETOUS & KANOUN 2004b]	X			X	X	
		[FOTA & al 1999]	X		X	X	X	
	OLC	[JUANOLE 2002]	X		X			X
		[ABDELLATIF & JUANOLE 2003a]	X		X		X	
		[ABDELLATIF & JUANOLE 2003b]	X		X			
LORIA-INPL	TRIO	[NAVET & SONG 2001]	X	X	X		X	X
		[CAVALIERE 2002]		X	X			X
		[WANG & al 2002]		X	X			X
		[KOUBAA & SONG 2002]		X	X		X	
LADSEB CNR University of Padova (Italy)		[VITTURI 2000]		X	X		X	
		[VITTURI 2001]		X	X		X	
		[VITTURI 2003]		X		X	X	
Department Computer Engineering & Department Mechanical Engineering University of Porto (Portugal)		[TOVAR & VASQUES 1999a]		X	X		X	
		[TOVAR & VASQUES 1999b]		X	X		X	
		[TOVAR & VASQUES 2001]		X	X		X	
Faculty of Engineering - Institute of Informatic and Telecommunications University of Catania (Italy)		[CAVALIERI & al 1999]		X	X		X	
		[CAVALIERI & al 2002]		X	X			X
Department of Computer Science & Engineering University of Washington, Seattle (USA)		[HINES & BORRIELLO 1997]		X		X		X
		[ORTEGA & BORRIELLO 1998]		X		X		X
Groupe IMPACT University of Michigan (USA)		[LIAN & al 2001]		X	X			X
		[LIAN 2001]		X	X			X
		[OTANEZ 2002]		X	X			X
School of Mechanical Engineering and RIMT Pusan National University (South Korea)		[LEE & LEE 2002]	X	X	X			X
		[LEE & al 2003]		X	X		X	
		[LEE & al 2004]		X	X		X	

Figure 19 : Synthèse bibliographique sur l'évaluation de performances d'architectures

modèles des architectures ne retranscrivent qu'un comportement généraliste et non pas un comportement dynamique définissant les réactions des équipements de l'architecture soumise à des stimulations.

Notre cadre de travail est l'évaluation de performances temporelles sur une architecture de contrôle-commande considérée dans sa globalité. Parmi l'ensemble des études que nous avons recensées, il s'avère que seuls les travaux de l'équipe de BORRIELLO correspondent à nos attente, mais leur domaine d'application porte sur la commande de systèmes embarqués et non de systèmes manufacturiers distribués qui font l'objet de notre étude.

Toutefois, malgré le fait qu'aucune étude ne corresponde à nos besoins, les divers travaux de recherche précédemment cités contiennent des approches ou des techniques d'évaluation dont nous pourrons tenir compte pour définir notre démarche d'évaluation de performances.

1.3.5 Conclusion

En observant les travaux d'évaluation de performance, nous avons remarqué que la place associée à cette tâche se trouve souvent en fin de conception dans le cycle de vie de l'architecture. Or comme nous l'avons montré, le besoin en évaluation de performance est présent tout au long de ce cycle.
En nous basant sur les travaux de la communauté de recherche portant sur la conception d'architectures de contrôle-commande, nous avons extrait une approche de modélisation commune selon laquelle

l'Architecture Opérationnelle est la distribution de l'Architecture Fonctionnelle sur une Architecture Matérielle.

Sur l'ensemble des travaux portant sur l'évaluation de performance, la plupart des évaluations ont une portée locale (portant sur un réseau seul ou un Automate Programmable seul ou sur des architectures simples). De plus, pour réaliser l'évaluation le comportement modélisé des éléments ou de l'architecture est le plus souvent statique, et ne décrit qu'une charge moyenne associée à chaque équipement. Les travaux d'évaluation des performances temporelles portant sur la globalité de l'architecture de contrôle-commande avec une modélisation de comportement dynamique sont peu nombreux.

1.4 Notre apport scientifique

Notre travail portera sur l'évaluation de performances temporelles en prenant en compte l'ensemble de l'architecture de contrôle-commande en charge dont le comportement sera modélisé de manière dynamique.
Notre travail devra de plus pouvoir accompagner l'architecte tout au long du cycle de conception d'une architecture de contrôle-commande, et ainsi permettre :

- durant un appel d'offre, d'évaluer que les solutions proposées vérifieront bien les performances d'un cahier des charges sommaire ;
- en phase de conception d'évaluer si les performances de l'architecture conçue répondent bien aux attentes du cahier des charges détaillé ;
- en phase d'installation, d'évaluer si les paramètres des différents équipements ne dégraderont pas les performances prévues durant la conception ;
- en phase d'amélioration ou d'optimisation, d'évaluer si les évolutions sur l'architecture permettront d'avoir de meilleures performances.

Dans le chapitre suivant, nous présentons notre méthode d'évaluation de performances temporelles prenant en compte le comportement dynamique de l'ensemble de l'Architecture Opérationnelle permettant d'accompagner l'architecte durant le cycle de vie de son architecture.

Chapitre 2

Démarche retenue pour notre contribution scientifique

« La méthode, c'est le chemin, une fois qu'on l'a parcouru »
Marcel Granet

Dans ce chapitre, nous exposons notre démarche de modélisation d'une architecture de contrôle-commande en vue d'en faire une évaluation de performances.
Ainsi, nous commençons par présenter brièvement notre approche : elle consiste à modéliser les composants des Architectures Matérielles et Fonctionnelles, que nous composons ensuite afin de constituer l'Architecture Opérationnelle.
Ensuite, nous développons le choix de l'outil de modélisation du comportement dynamique de l'Architecture Opérationnelle, ainsi que ses implications sur notre démarche.
Enfin, nous illustrons notre démarche sur un exemple basique afin d'en exposer clairement les différentes phases, qui feront l'objet d'un développement dans le Chapitre 3.

2.1 Démarche générale [MEUNIER & DENIS 1997]

Le travail de l'architecte est de définir une Architecture Opérationnelle à partir d'un cahier des charges composé de spécifications fonctionnelles et de contraintes matérielles. Ce travail amène en règle générale à concevoir plusieurs Architectures Opérationnelles qui pourraient répondre au cahier des charges. L'architecte devra donc choisir l'une d'elle à l'aide de critères (coût de l'architecture, complexité,…), mais dans tous les cas, l'architecture qu'il choisira devra garantir des performances attendues «a priori» par le cahier des charges. Il lui est donc nécessaire d'évaluer les performances des architectures qu'il aura définies (fig. 20).

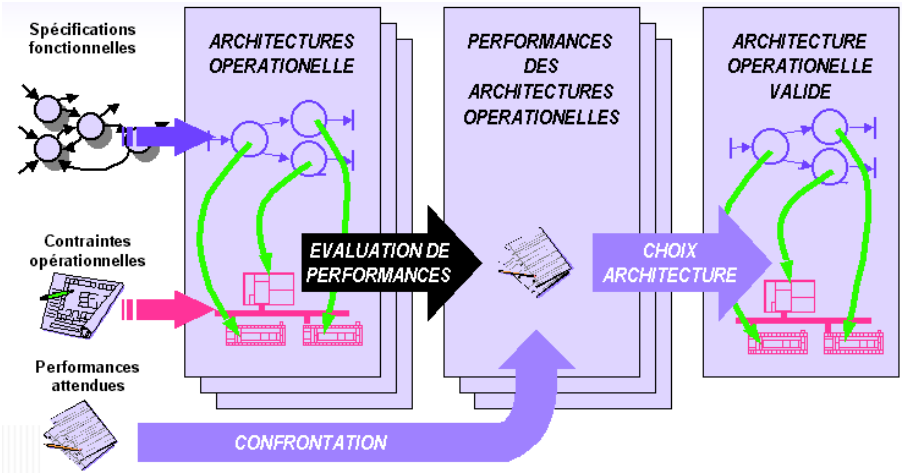


Figure 20 : Travail de l'architecte

Comme notre travail porte sur l'évaluation des performances de ces architectures (flèche noire de la figure 20), les Architectures Opérationnelles définies par l'architecte sont des données. La première étape de notre démarche consiste en une modélisation du comportement dynamique de l'Architecture Opérationnelle, la seconde permet l'évaluation de performance de ce modèle (fig. 21).

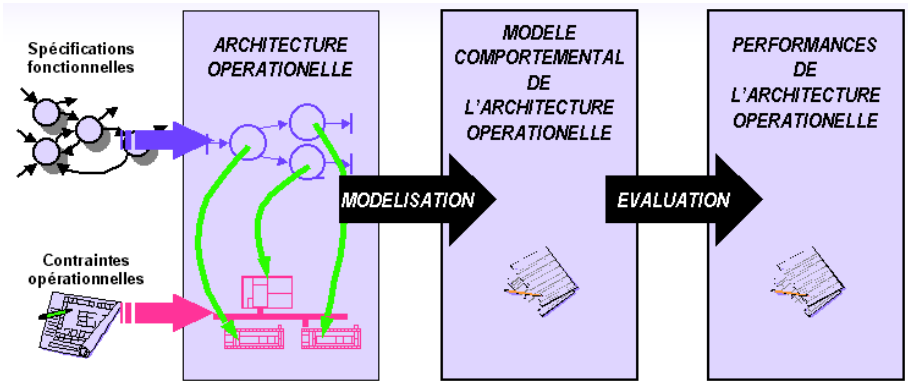


Figure 21 : Notre démarche

Construire directement un modèle définissant le comportement dynamique de l'Architecture Opérationnelle est un travail très complexe et peu reproductible. Nous allons donc privilégier une approche modulaire et générique en modélisant les différentes parties qui la composent. L'Architecture Opérationnelle étant composée d'une Architecture Fonctionnelle, d'une Architecture Matérielle et d'une

association de chaque Tâche Fonctionnelle à un équipement matériel, nous utiliserons cette vision modulaire des architectures afin de modéliser non pas des architectures globales mais un ensemble d'éléments modulaires d'architecture. Une fois que l'on aura modélisé tous les équipements composants ces architectures, nous construirons le modèle comportemental de l'Architecture Opérationnelle en recomposant les modèles des différents éléments tout en respectant les associations Tâches Fonctionnelles - Équipements matériels. On obtiendra alors le modèle du comportement dynamique de chaque Architecture Opérationnelle (fig. 22).

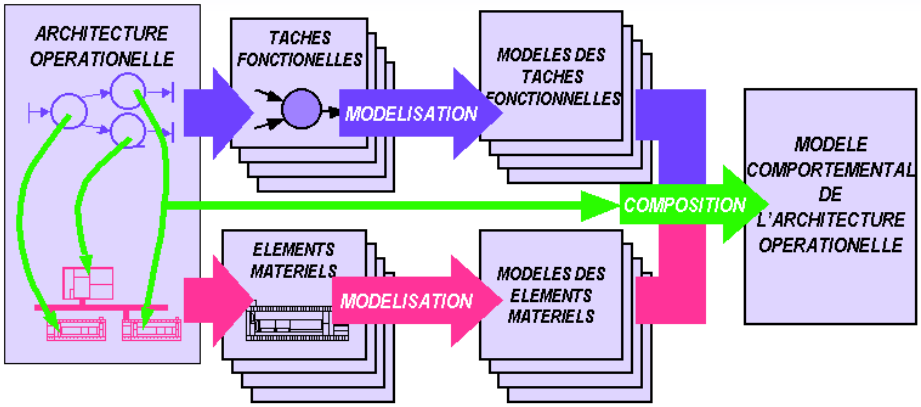


Figure 22 : Démarche de modélisation comportementale

Une fois la phase de modélisation effectuée, reste à effectuer l'évaluation des performances. Cette phase dépend du modèle qui a été utilisé car les outils associés et les évaluations possibles diffèrent d'un modèle à l'autre. Il est donc nécessaire de choisir correctement l'outil de modélisation qui nous servira dans notre démarche. Nous traitons de ce choix dans le paragraphe suivant.

2.2 Outil de modélisation du comportement dynamique

2.2.1 Choix de la méthode d'évaluation

Afin d'évaluer les performances d'une architecture, nous pouvons utiliser une approche analytique ou une méthode de simulation. De nombreux travaux de recherche confrontent ces deux approches qui ont chacune leurs avantages et leurs inconvénients. Sachant que notre démarche devra pouvoir s'appliquer en utilisant des données partielles (voire incomplètes) ou alors très précises et que l'on souhaite réaliser des évaluations temporelles précises du comportement dynamique d'Architectures Opérationnelles, nous avons opté pour une approche d'évaluation par simulation.

2.2.2 Choix d'un formalisme et d'un outil de simulation

Traditionnellement, les langages utilisés pour mettre en oeuvre une technique de simulation sont répartis en trois familles [HEBUTERNE 2000] :

- les langages de programmation universels (C, C++, Fortran, ML, …) : Il faut écrire soit même toutes les procédures de simulation (gestion de l'échéancier, mise en file, tirage de variables aléatoires, prise de mesures, édition)

- les langages de simulation universels : Ce type de langages prend en charge toutes les fonctions énumérées ci-dessus, déchargeant l'usager d'une lourde tâche d'écriture, et augmentant la fiabilité du modèles. Quelques exemples de ces modèles sont :
  - SIMSCRIPT, GPSS, SIMULA. Ces langages de «première génération», travaillent en mode texte.
  - Bones, OP-NET. Ces langages sont plus modernes et offrent des interfaces plus confortables : fonctions graphiques, composition du modèle par menus, gestion de bibliothèques, outils de mise au point et de trace graphique, animation, etc…
- les langages spécifiques à l'application ou à une classe d'application. Tous les degrés de spécialisation sont possibles. Le modèle que construit l'utilisateur se présente, non plus comme une description du système, mais comme le paramétrage d'un modèle préexistant : c'est la force de ce type de langage (gain en sûreté et en rapidité), mais aussi sa faiblesse (impossibilité de l'utiliser dans un autre cadre). Par exemple QNAP permet de simuler très efficacement des réseaux de files d'attentes (systèmes informatiques classiques). Mais tout système doit y être représenté comme un réseau de files.

En amont du langage de simulation, il faut retenir un formalisme de spécifications. Classiquement, les formalismes couramment utilisés sont regroupés en deux familles basés sur :

- les états, on citera par exemple, les Automates à Éléments Finis [BUDKOWSKI & NAJM 1983] ou les Réseaux de Petri [PETRI 1979] [MURATA 1989]. Ces derniers étant un formalisme couramment utilisé par la communauté automatique.
- les files, on citera par exemple les réseaux de files d'attente [KLEINROCK 1975] ou les chaînes de Markov [BREMAUD 1998].

Nous porterons notre choix sur les réseaux de Petri, car c'est un formalisme très courant dans la communauté des automaticiens. De plus, comme nous il est décrit dans l'ouvrage de Michel Diaz [DIAZ & al 2001] les RdP et leurs extensions possèdent un intérêt fondamental indéniable pour :

- la modélisation et la maîtrise des comportements des systèmes parallèles et distribués, synchronisés et communicants ;
- le support graphique qui aide à l'expression et à la compréhension des mécanismes de base de ces comportements ;
- les représentations faciles à comprendre et à manipuler à la fois pour la création des modèles et pour leurs analyses.

Une fois réalisé, le modèle de simulation devra être validé selon plusieurs aspects :

- la validation du modèle (le modèle conceptuel est-il conforme au système réel à étudier) ;
- la validation du modèle écrit, en vérifiant la conformité du modèle exprimé en langage de simulation par rapport au modèle défini par le formalisme choisi par l'utilisateur ;
- la validation des données fournies (paramètres numériques, types de lois, etc…) ;
- la validation des résultats (analyse statistique des résultats, pour fournir un intervalle de confiance).

Toutefois, la validation du modèle écrit peut être contournée en n'utilisant pas un langage de simulation mais un joueur de réseau de Petri qui évitera tout problème de traduction. Étant donné que le joueur n'est pas un logiciel gérant un langage de simulation, il sera donc nécessaire d'adjoindre les générateurs de stimuli, des observateurs ainsi qu'un système de dépouillement. En revanche, il assure la gestion de

l'échéancier, la mise en file, le tirage de variables aléatoires, l'édition, la visualisation dynamique de l'évolution du modèle ainsi que la validation du formalisme RdP du modèle. Nous allons donc nous intéresser au choix de la classe de RdP puis de son joueur (que nous appellerons par la suite outil de simulation) qui répondront à nos besoins.

Les formalismes réseaux de Petri sont nombreux et n'ont pas tous les mêmes potentiels en terme de définitions et de modélisation des concepts, de compréhension et d'études des sémantiques et d'analyse des mécanismes et des comportements. Trois domaines sémantiques structurent les grandes familles de réseaux de Petri (fig. 23):

- une sémantique discrète, pour les comportements qui peuvent se représenter par des graphes finis ou dénombrables d'états ;
- une sémantique sur un domaine continu, pour les comportements qui nécessitent la prise en considération explicite d'un temps dense ;
- une sémantique stochastique, pour les comportements qui incluent des distributions de franchissement et conduisent à des processus stochastiques.

Ces trois domaines correspondent chacun à un formalisme de référence, respectivement les réseaux de Petri places-transitions (RdP PT), les réseaux de Petri temporels (RdP T), les réseaux de Petri stochastiques (RdP S).

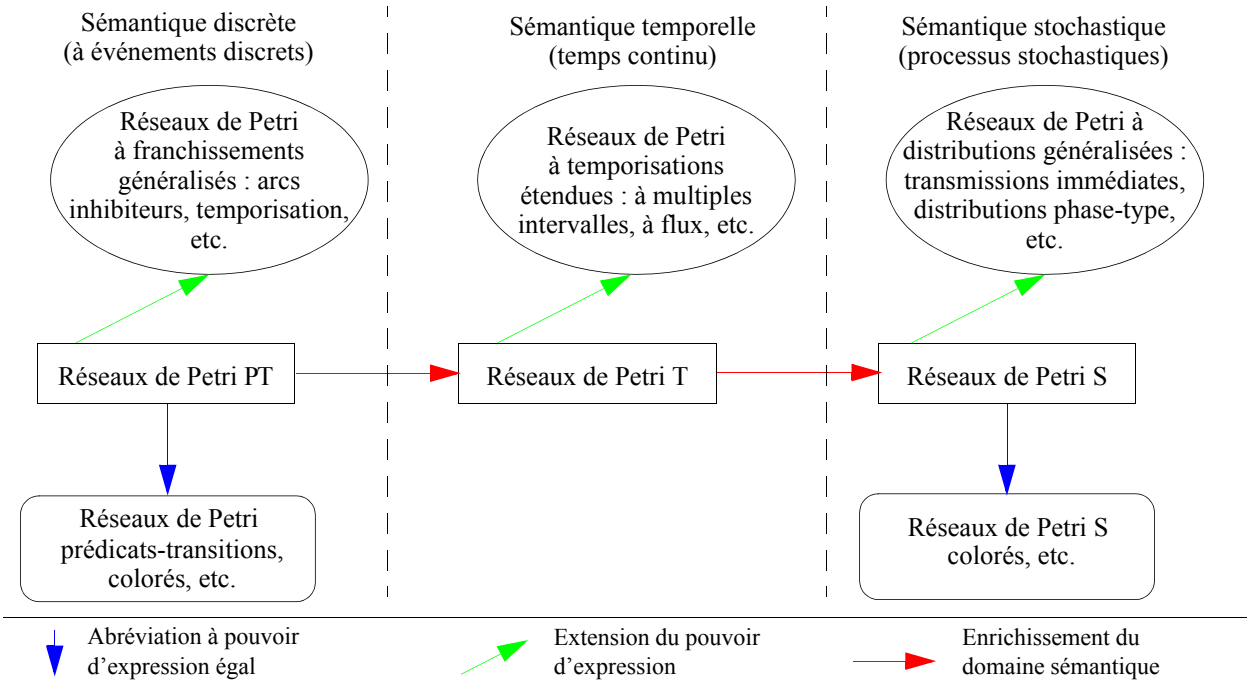


Figure 23 : Les domaines sémantiques des réseaux de Petri

Comme indiqué sur la figure, de nouveaux modèles pourront être dérivés à partir des formalismes de référence, pour proposer des modèles :

- soit plus compacts en termes de description afin d'améliorer la simplicité des modélisations sans augmenter le pouvoir d'expression ;
- soit plus puissants en termes de pouvoir d'expression permettant de décrire des mécanismes qui ne pouvaient pas être décrits par des modèles non-étendus.

Notre démarche ayant pour but d'évaluer les performances de types temps de réponses d'architectures de contrôle-commande, il est donc nécessaire d'avoir une intégration du temps dans nos modèles par le biais de temporisations. Parmi les différents domaines sémantiques précédemment cités, de nombreux modèles réseaux de Petri intègrent les temporisations :

- les réseaux de Petri temporisés, extension des RdP PT, intègrent les temporisations en associant aux transitions des délais de franchissement ;
- les réseaux de Petri Temporels intègrent les temporisation en associant aux transitions des fenêtres de franchissement, ce franchissement se faisant pour un délai nul ;
- les réseaux de Petri Stochastiques intègrent les temporisation en associant aux transitions des durées de franchissement variables déterminées par des tirages aléatoires dans des distributions données.

Dans son livre, Michel DIAZ [DIAZ & al 2001] nous indique que les RdP Temporels sont adaptés pour l'analyse des comportements, les RdP temporisés sont adaptés pour l'analyse de performances et que les RdP Stochastiques sont adaptés aux systèmes stochastiques ou aux étude de systèmes dans des états «transitoires» pour évaluer par exemple la sûreté de fonctionnement. Sachant que nous étudions les performances d'architectures de commande en fonctionnement «normal» ou en état «stationnaire», nous opterons donc pour des RdP temporisés.

De plus, notre démarche de modélisation s'effectue à partir de modèles issus d'une bibliothèque pour les équipements matériels, il nous faut donc avoir des modèles génériques que l'on instanciera et que l'on paramètrera. Il existe une classe de formalismes correspondant à nos besoins, s'appelant les réseaux de Petri de haut niveau qui comprend entre autres les RdP colorés [JENSEN 1986] [DAVID & ALLA 1989] [YANG & al 1995] [XIN & al 2000] [JIANG & al 2001], les RdP à prédicats-transitions [GENRICH & LAUTENBACH 1981], les RdP algébriques [REISIG 1991] ou encore les RdP bien formés [DUTHEILLET 1992].

De nombreuses classes de Réseaux de Petri haut niveau temporisés existent, mais étant donné que nous avons besoin de faire des simulations de nos modèles, il nous est donc nécessaire d'avoir une plateforme de simulation informatique. Parmi l'ensemble des classes de Réseaux de Petri Colorés Temporisés, l'une d'entre elles attire notre attention : les réseaux de Petri Colorés Temporisés (RdPCT) selon Kurt Jensen [JENSEN 1997a] [JENSEN 1997b] [JENSEN 1997c] répond à nos besoins et possède de plus un outil logiciel de modélisation, simulation et vérification : CPNTools [RATZER & al 2003] sous environnement Windows™ remplaçant du logiciel Design/CPN [CHRISTENSEN & al 1997] sous l'environnement linux/unix. La plateforme logiciel Design CPN / CPNTools est un outil utilisé par :

- de nombreuses entreprises nationales (Peugeot-Citroën [MONTCELET & al 1998]), européennes (Deutsche Telekom [CAPELLMANN & al 1998] [CAPELLMANN & al 1999]),
- de nombreuses entreprises internationales (Nokia [XU & KUUSELA 1998] [LORENTSEN & al 2001], Hewlett-Packard [FIGUEIREDO & KRISTENSEN 1999], US Air Force [LINDSTROM & HAIDER 2001], …)
- ainsi que par de nombreux laboratoires de recherche internationaux (University of Las Palmas [MORERA & GONZALES 1999], University of South Australia [ELLIOT & al 2002], University of Aarhus [JORGENSEN 2002], University of Kiel [HIELSCHER & al 1998], University of Calabria [NIGRO & PUPO 1998], University of Newcastle [BURNS & al 1998], …).

Notre but n'est pas de réaliser une étude comparative des différents couples classes/outils portant sur les de RdP de haut niveau temporisés, mais de choisir une d'entre elles correspondant à nos besoins. Aux vues des possibilités du modèle RdPCT, de son utilisation dans le monde de la recherche et de l'existence d'une plateforme logiciel de simulation, nous portons notre choix sur les RdPCT selon Kurt JENSEN et l'outil Design/CPN.

2.2.3 Implications sur notre démarche

Nous venons de choisir notre outil de modélisation du comportement dynamique d'architecture opérationnelle ainsi qu'une méthode d'évaluation par simulation. Ces choix ne se sont pas fait sans conséquence, en effet ils ont des impacts sur les différentes phases de notre démarche : sur la modélisation, et l'évaluation.

2.2.3.1 Impacts sur la phase de modélisation

Une modélisation utilisant les Réseaux de Petri colorés temporisés n'est pas aisée pour tous les ingénieurs automaticiens, et il est fort probable que l'architecte n'aura pas ou peu de connaissance concernant cette classe de Réseaux de Petri. Or, étant donné que l'Architecture Fonctionnelle diffère d'une conception à l'autre, il n'est pas possible d'avoir des modèles types d'architectures fonctionnelles. L'architecte devra donc à chaque fois réaliser les modèles Réseaux de Petri pour l'Architecture Fonctionnelle. Il sera donc nécessaire d'assister cette phase de modélisation en proposant un guide de modélisation à l'architecte.

En ce qui concerne l'Architecture Matérielle, comme le cadre de notre étude se place dans les productions en petite série ou unitaire, les équipements qui composent l'architecture sont standards. Afin d'éviter à l'architecte de refaire des modèles de composants matériels qui sont souvent complexes et long à mettre au point, une bibliothèque comportera les modèles des équipements courants. L'architecte n'a plus qu'à paramétrer le modèle.

Étant donné que l'Architecture Opérationnelle est le fruit de la composition des deux architectures précédemment citées, elle doit être donc construite à chaque conception. Tout comme pour l'Architecture Fonctionnelle, il n'y aura pas de modèles RdP générique. Il conviendra de concevoir un guide d'assistance à la construction du modèle Réseau de Petri de l'Architecture Opérationnelle à partir des modèles des Architectures Fonctionnelle et Matérielle.

2.2.3.2 Impacts sur la phase d'évaluation

Une fois le modèle de l'Architecture Opérationnelle réalisé, on désire évaluer ses performances de type temps de réponse. Cette évaluation est liée au fait que nous réalisons une simulation de modèles réseaux de Petri.

La simulation implique que l'on doit adjoindre en interaction à notre modèle de contrôle-commande son environnement de simulation. En effet, la Partie Commande échange des informations avec :

- les opérateurs via les pupitres de dialogues ;
- la gestion de production via la supervision ;
- les équipements de la Partie Opérative.

Il est donc nécessaire de compléter le modèle de contrôle-commande par un modèle permettant de générer les informations transitant via l'Architecture Opérationnelle, que ce soit les informations liées aux performances évaluées ou les informations liées à la charge de la commande.

Or, la simulation de réseau de Petri ne fourni que des dates de franchissement de transitions, et selon les simulateurs des données statistiques comme le marquage moyen d'une place, le taux de franchissement d'une transition, … Rien ne permet d'atteindre directement le temps de propagation d'une information au travers de l'Architecture Opérationnelle.
Il est donc nécessaire de compléter le modèle de contrôle-commande par des observateurs d'événements pour observer les dates de passage des différents événements (déclencheurs ou effets) afin d'en déduire les temps de propagation et donc les performances à évaluer.

Les deux compléments précédemment cités sont regroupés dans la sous-phase de préparation du modèle avant simulation. Durant cette sous-phase, les modèles devront être enrichis par l'architecte. Afin de lui permettre de réaliser cette sous-phase, il sera nécessaire de lui fournir une assistance à la modélisation de la partie opérative et des générateurs ou observateurs d'événements.
Une fois réalisée la sous-phase de préparation, la sous-phase de simulation est exécutée. À la fin de cette sous-phase, l'architecte obtient les informations récoltées par les observateurs (dates d'occurrence d'événements ou différences entre deux dates) et non les performances qu'il souhaite mesurer (temps moyen, minimum ou maximum de transmission, …). Il est donc nécessaire de faire une sous-phase d'analyse des observations pour en déduire les performances.
La phase d'évaluation du modèle présentée sur la figure 21 se décompose finalement en trois sous-phases (fig. 24) :

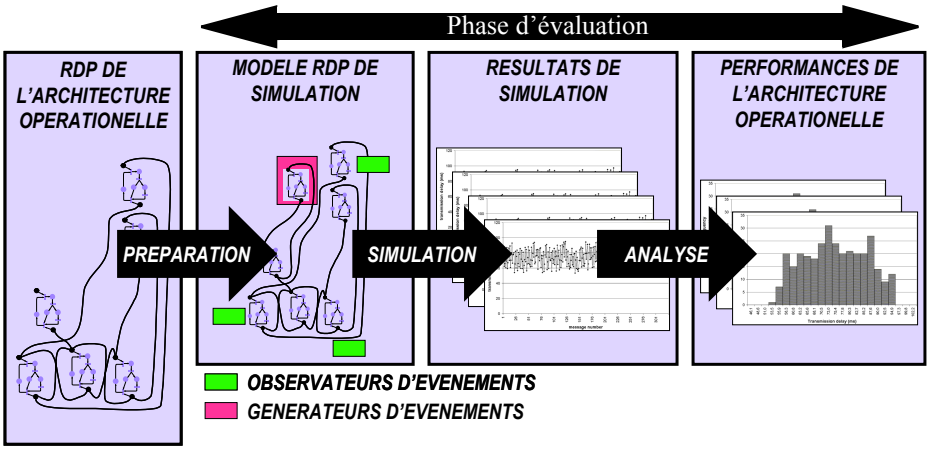


Figure 24 : Démarche d'évaluation du modèle de l'Architecture Opérationnelle

2.3 Illustration de la démarche

2.3.1 Présentation de l'exemple : Sécurité d'une centrale de vapeur

De manière à parfaire la définition de notre démarche, nous allons l'illustrer par le traitement d'un exemple très simple. Nous avons choisi pour ce faire une centrale de vapeur (fig. 25). Cette centrale possède deux énergies de chauffage pour obtenir de la vapeur, une chaudière au gaz et une chaudière électrique.

Sachant que l'on désire illustrer notre méthode et non l'exemple, les contraintes et le cahier des charges sont grandement simplifiés. On étudiera uniquement une fonction de sécurité du système de chauffage : la fermeture des vannes de vapeur. (indiquées sur la figure 25).

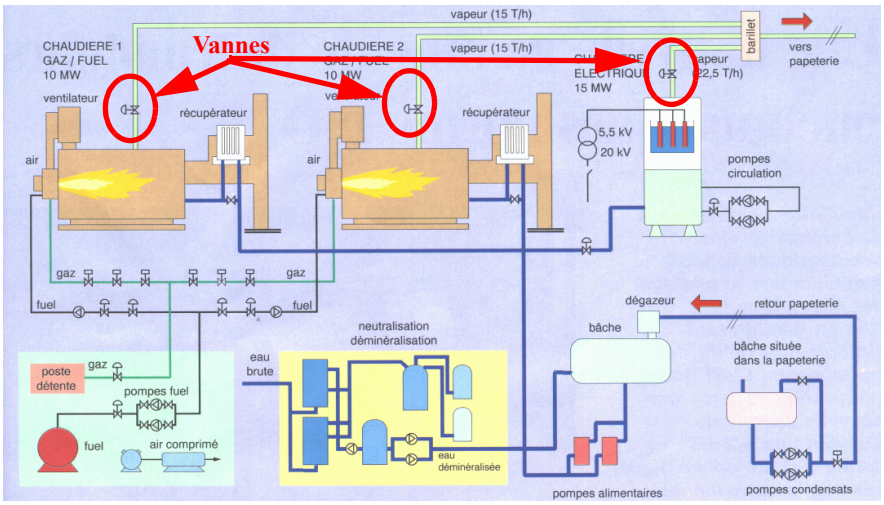


Figure 25 : Centrale de vapeur pour une papeterie

Les contraintes opérationnelles sont définies comme suit :

- en raison d'une proximité géographique (les deux chaudières à gaz sont proches), les électrovannes des chaudières à gaz sont commandées par un unique automate programmable industriel.
- pour une raison d'éloignement géographique (la chaudière électrique est très éloignée des chaudières à gaz), l'électrovanne de la chaudière électrique est commandée par un automate programmable différent de celui des chaudières à gaz.
- le délai de réaction entre l'ordre de fermeture des vannes et la fermeture de celles-ci doit être inférieur à 300 ms pour la chaudière à gaz et doit être inférieur à 400 ms pour la chaudière électrique.
- le retard entre la fermeture des deux types de chaudières (gaz et électrique) ne doit pas excéder 200 ms.

2.3.2 Présentation des Architectures fournies par l'architecte

Nous considérons pour notre exemple, que l'architecte a défini les éléments suivants à partir du cahier des charges :

- Une architecture Fonctionnelle décrite en modèle SA ;
- Deux Architectures Matérielles ;
- Trois Architectures Opérationnelles.

2.3.2.1 Architecture Fonctionnelle du système

La fonction de sécurité est réalisée par trois tâches *T1*, *T2* et *T3*. La tâche *T1* reçoit l'ordre d'arrêter toutes les vannes des chaudières (*Arret_Vanne* noté AV), elle commande l'arrêt des vannes des chaudières à gaz (*Arret_Vannes_Gaz* noté AVG) et de la vanne de la chaudière électrique (*Arret_Vanne_Electrique* noté AVE). La tâche *T2* reçoit l'ordre de fermeture des vannes des chaudières à gaz et envoie les ordres aux contacteurs correspondant (*KM_Vanne_Gaz_1* et *KM_Vanne_Gaz_2* notés respectivement KMVG1

et KMVG2). De même, la tâche *T3* reçoit l'ordre de fermeture de la vanne de la chaudière électrique et envoie l'ordre au contacteur correspondant (*KM_Vanne_Electrique* noté KMVE) (fig. 26a).

2.3.2.2 Architectures Matérielles proposées

Les Architectures Matérielles que l'architecte a retenues sont bâties autour d'un réseau d'automate et de trois équipements de commande.
Dans la configuration de la figure 26b, un automate programmable traite chacun des signaux *AV*, *KMVG1*, *KMVG2* et *KMVE*. Dans cette configuration *AV* est une entrée de l'automate AP3 qui provient d'un pupitre constitué de boutons poussoirs. Le réseau d'automates (de type Profibus-DP ou UNI-TE) est basé sur un protocole d'accès maître-esclave. L'automate qui dispose de l'information *AV* a été choisi comme maître sur le réseau.
Dans la configuration de la figure 26c, un terminal d'exploitation remplace le pupitre et seulement deux automates sont utilisés. Les terminaux d'exploitation ne pouvant généralement pas assurer le rôle de maître sur le réseau d'automate, l'automate *API1* assure cette fonction dans cette deuxième architecture matérielle.

2.3.2.3 Architectures Opérationnelles retenues

Les figure 26d, figure 26e et figure 26f présentent trois solutions d'Architectures Opérationnelles proposées par l'architecte parmi l'ensemble des projections possibles.

Parmi les Architectures Opérationnelles retenues, il est maintenant nécessaire de faire un choix. Pour cela l'évaluation des performances de chacune de ces architectures permettra de guider l'architecte dans son choix final. Afin de réaliser ces évaluations, notre méthode nous amène maintenant à générer les architectures qui une fois simulées, permettront d'évaluer les performances des Architectures Opérationnelles.

Afin de ne pas surcharger notre discours, nous ne présenterons que les études portant sur la projection de l'architecture fonctionnelle sur la configuration à trois API (fig. 26d) et sur la projection sur la configuration à deux automates et un terminal avec les tâches T1 et T2 dans l'*API1* (fig. 26e).
Dans la suite du chapitre, nous décrirons les différentes étapes de notre méthode d'évaluation de performances.

2.3.3 Modélisation comportementale de l'Architecture Fonctionnelle

2.3.3.1 Modélisation des tâches fonctionnelles

La modélisation du comportement dynamique de l'Architecture Fonctionnelle résulte de la modélisation des atomes qui la composent. Chaque atome est modélisé par un réseau de Petri dont le comportement correspond à celui de l'atome. Ces modèles sont en général simples car le comportement dynamique des atomes l'est aussi. Par exemple, on trouve souvent un comportement du type : «une ou plusieurs entrées provoquent une ou plusieurs sorties».

Après examen de l'Architecture Fonctionnelle de notre exemple support, nous pouvons la découper en trois atomes correspondant aux trois tâches T1, T2 et T3 (fig. 27).

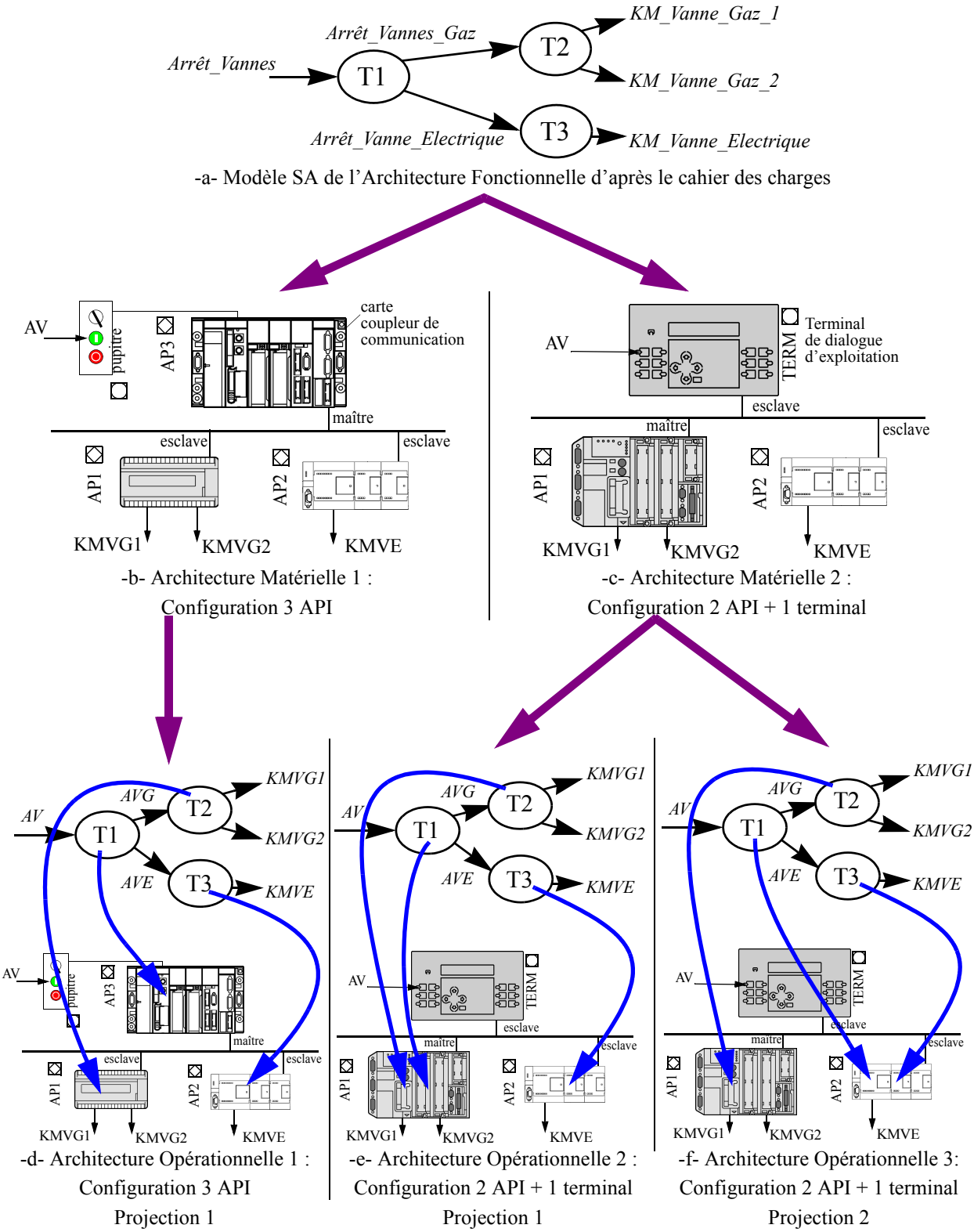


Figure 26 : Architectures Fonctionnelle, Matérielles et Opérationnelles

Le modèle réseau de Petri d'un tel atome est composé de deux places et une transition. Le poids de l'arc entrant correspond à la somme des informations ou ordres entrants ; le poids de l'arc sortant étant défini de manière similaire. Nous obtenons donc pour notre exemple support, trois modèles des différentes tâches de l'Architecture Fonctionnelles (fig. 28).

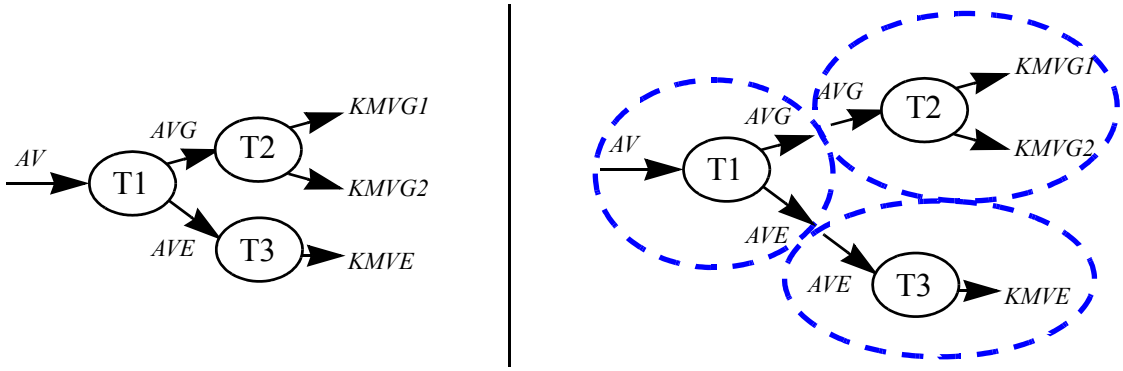


Figure 27 : Décomposition de l'Architecture Fonctionnelle en atomes

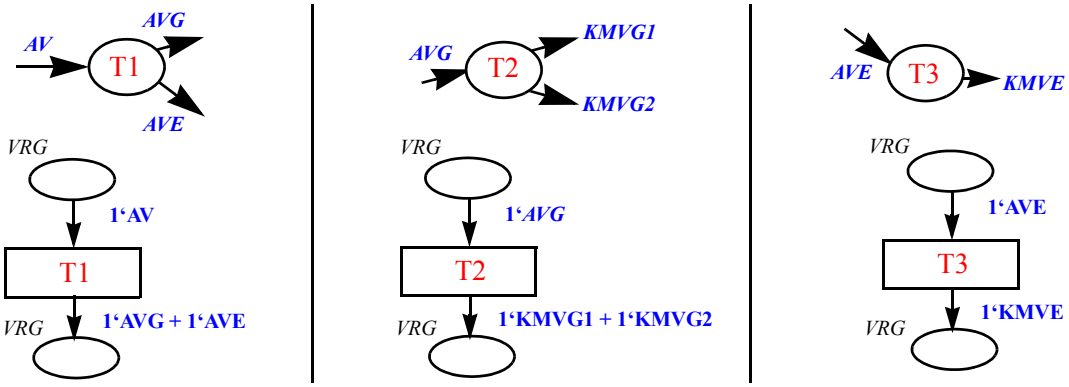


Figure 28 : Modèles de comportement dynamique des tâches de l'Architecture Fonctionnelle

Dans le formalisme RdP selon Jensen, il faut associer à chaque place le type des données qui seront contenues dans celle-ci. Nous avons nommé «VRG» le type énuméré représentant l'ensemble des messages ou informations qui circulent dans l'architecture. Nous avons donc pour notre exemple l'ensemble «VRG» :

$$VRG = \{AV, AVE, AVG, KMVE, KMVG1, KMVG2\} \quad (1)$$

Une fois les modèles comportementaux de l'Architecture Fonctionnelle réalisés, il nous faut faire de même pour l'Architecture Matérielle.

2.3.4 Modélisation comportementale d'Architectures Matérielles

Tout comme pour la modélisation de l'Architecture Fonctionnelle, la modélisation d'une Architecture Matérielle se fait par la modélisation des divers équipements qui composent cette architecture. Dans notre cas, nous avons à modéliser trois types d'équipements matériels : Automates Programmables Industriels, terminal de dialogue et réseau de terrain.

Contrairement aux atomes de l'Architecture Fonctionnelle, les équipements de l'Architecture Matérielle n'ont pas de comportement aussi simple à modéliser. Toutefois, le modèle comportemental d'un équipement est globalement le même pour une gamme de matériel donné. Par exemple, une famille d'Automate Programmable Industriel (API) peut être modélisée de la même manière si on peut définir les caractéristiques qui lui sont spécifiques comme sa capacité d'exécution, ses retards de cartes d'entrées ou de sortie …

Des modèles génériques et paramétrables peuvent donc éviter de recréer des modèles pour chaque Architecture Matérielle à modéliser. Afin d'éviter ce travail de modélisation à l'architecte, nous avons créé divers modèles d'équipements matériels génériques et paramétrables. Ces équipements forment une bibliothèque que l'architecte peut utiliser pour construire le modèle du comportement de son Architecture Matérielle. Ici des réseaux de Petri hiérarchiques sont utilisés, permettant la représentation d'un modèle de la bibliothèque sous la forme d'une transition de niveau hiérarchique supérieur et d'un réseau de Petri détaillé de niveau inférieur (sous DesignCPN, un cadre avec le nom du réseau de niveau inférieur est associé à la transition de haut niveau). Enfin, pour concevoir le modèle de comportement dynamique d'une Architecture Matérielle, nous utiliserons les équipements d'une bibliothèque de modèles de comportement dynamique.

2.3.4.1 Modèle d'un Automate Programmable Industriel

D'une manière globale, un Automate Programmable Industriel est un équipement composé d'un processeur qui exécute le programme utilisateur, de cartes d'entrées/sorties et de cartes de communication. Nous considérerons pour notre cas des API mono processeurs avec un traitement des informations cyclique dont la structure est représentée sur la figure 29a.

Le modèle du comportement dynamique construit pour la bibliothèque, a donc une structure en trois blocs : le premier gère le retard dû aux cartes d'entrées ainsi que le retard induit par le cycle API, le second exécute le programme et le troisième gère les retards des cartes de sorties ainsi que la destination des informations (figure 29b).

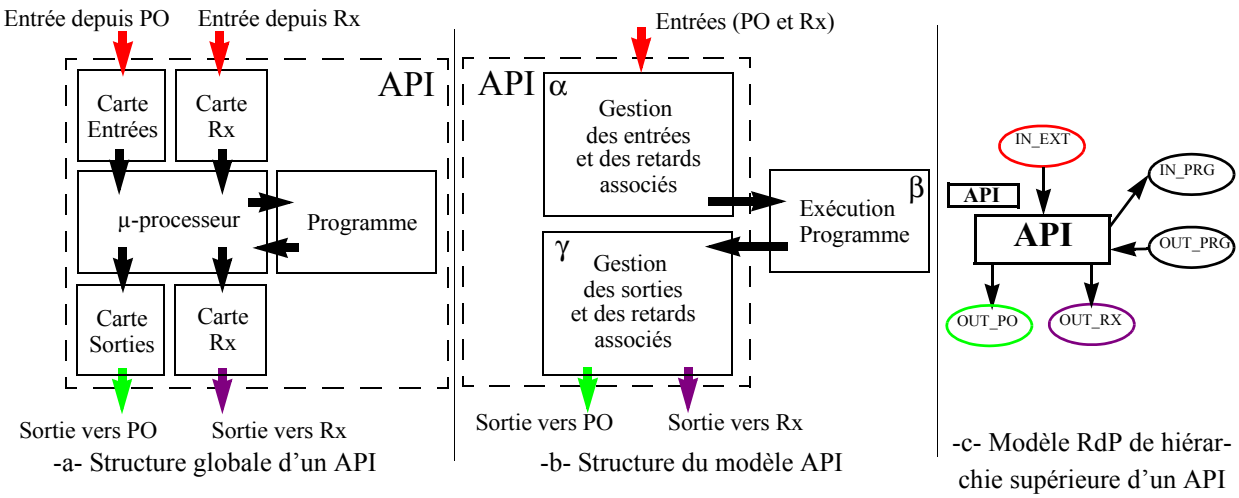
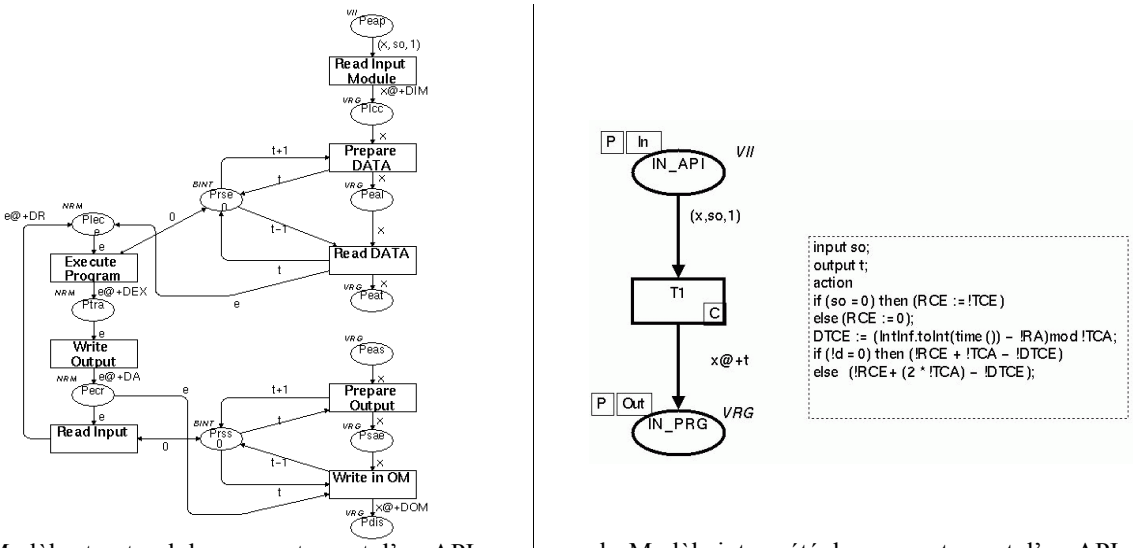


Figure 29 : De la structure globale de l'API au modèle RdP de l'API

Les blocs α et γ correspondent à des éléments physiques alors que le bloc β correspond à des atomes fonctionnels qui ont été modélisés dans le paragraphe 2.3.3. La figure 30c représente le modèle RdP de hiérarchie supérieure pour un API. Les modèles des blocs a et g étant complexes, nous allons les expliciter plus en avant.

Le bloc α correspondant à la prise en compte des entrées et à la gestion du cycle, la bibliothèque des modèles comportementaux comporte divers modèles dont ceux présentés par les figure 30a et figure 30b.



-a- Modèle structurel du comportement d'un API

-b- Modèle interprété du comportement d'un API

Figure 30 : Exemples de modèles dynamiques du traitement cyclique d'un API

Le fonctionnement ainsi que la création de ces modèles seront explicités dans le Chapitre 4. De par sa compacité et sa facilité de représentation, nous optons pour le modèle de la figure 30b.
Il traduit un comportement générique et peut être instancié pour représenter plusieurs équipements matériels de type API. Il est alors nécessaire de paramètrer chacune des instances en quantifiant leurs paramètres (retard des cartes d'entrées, durée du cycle automate, durée de la phase de lecture des entrées dans le cycle automate, durée de la phase d'émission des sorties dans le cycle automate).

Le bloc γ, qui gère les sorties et leurs retards associés, correspond aux problèmes de transmission des informations via les cartes de sorties ou les cartes réseaux. La figure 31 présente un modèle provenant de notre bibliothèque correspondant à une carte de sortie et une carte réseau.

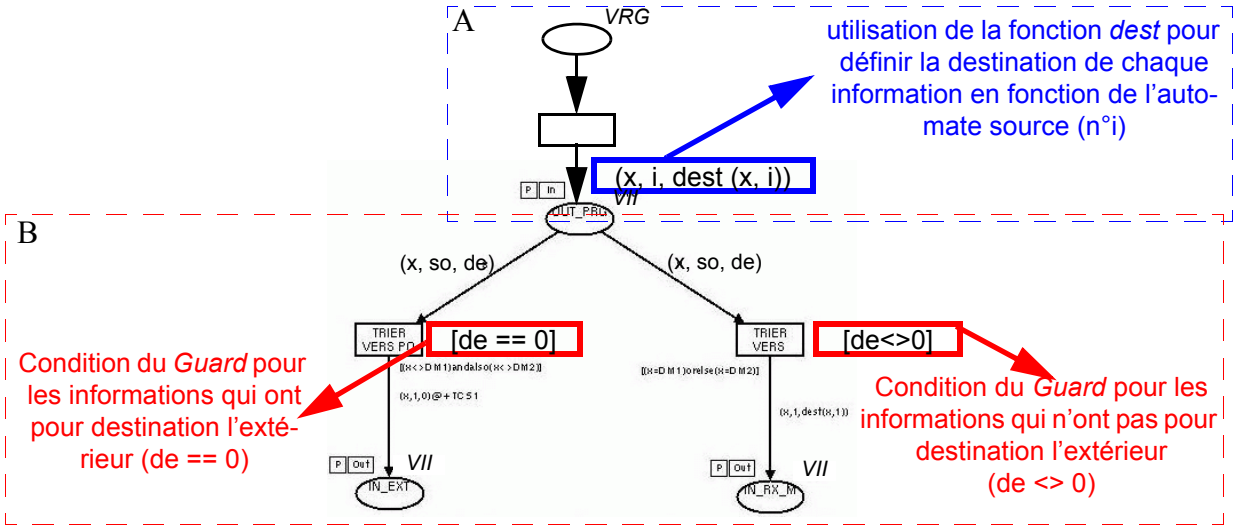
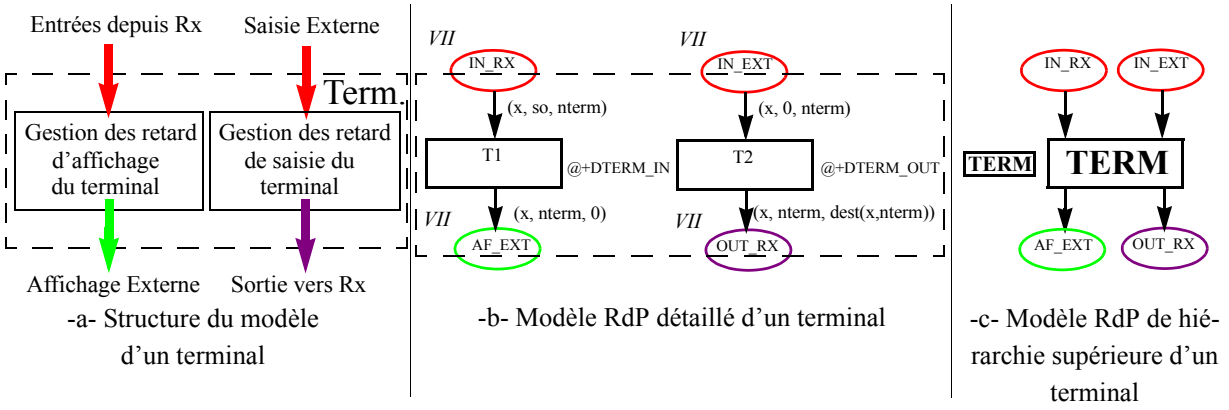


Figure 31 : Exemple de modèle de gestion des cartes de sorties et réseau d'un API

Ce modèle comprend deux grands cadres : le cadre A définit la destination des informations transitant entre les atomes fonctionnels et le cadre B oriente ces informations vers l'équipement cible (ce qui correspond aux cartes de sorties ou cartes interfaces réseaux).

2.3.4.2 Modèle d'un terminal de dialogue

Un terminal de dialogue n'est pas un équipement de traitement car il n'exécute pas de programme utilisateur. La structure du modèle issu de notre bibliothèque est décrit par la figure 32a.



-a- Structure du modèle d'un terminal

-b- Modèle RdP détaillé d'un terminal

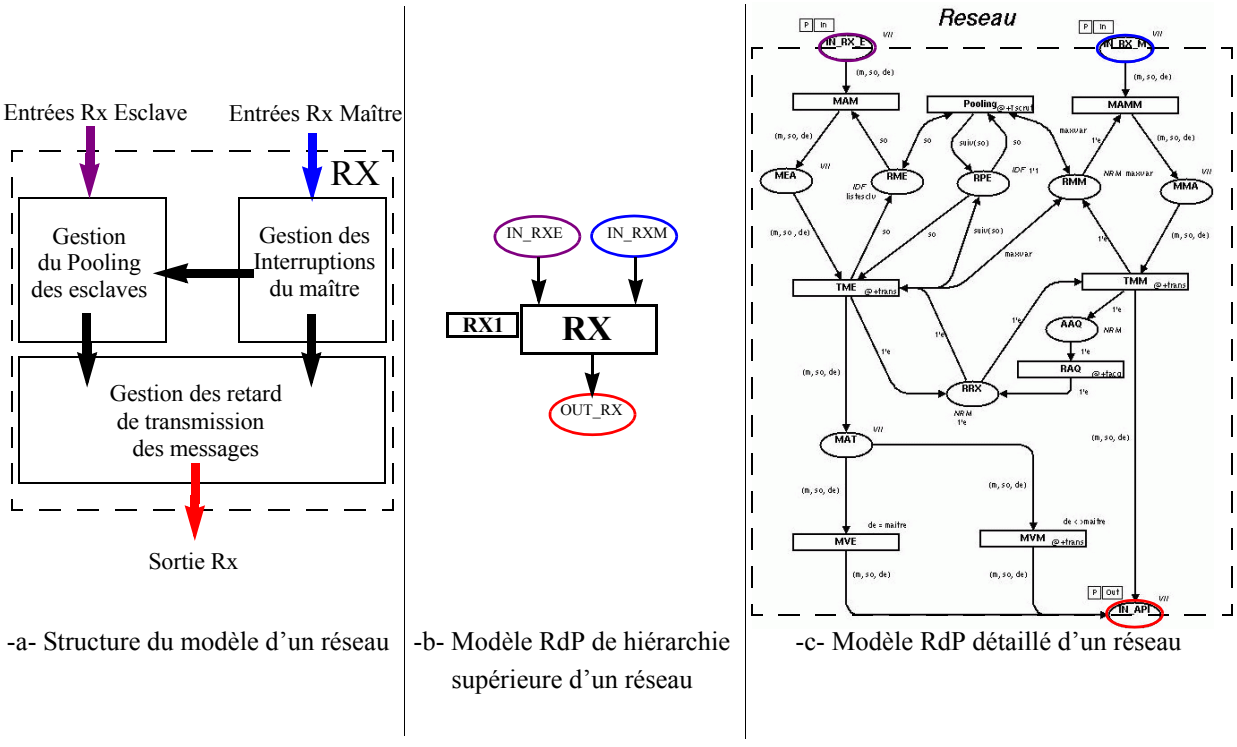
-c- Modèle RdP de hiérarchie supérieure d'un terminal

Figure 32 : structure et modèle d'un terminal

Le modèle comportemental (figure 32b) est assez simple et est générique et peut donc être instancié. Les paramètres qui sont associés à ce modèle sont les durées de prise en compte d'informations lors de la saisie ou lors de l'affichage.

2.3.4.3 Modèle d'un réseau de communication

Notre exemple est construit autour d'un réseau de type maître-esclave intégrant une scrutation cyclique. La structure du modèle que nous avons construit est décrit par la figure 33a.



-a- Structure du modèle d'un réseau

-b- Modèle RdP de hiérarchie supérieure d'un réseau

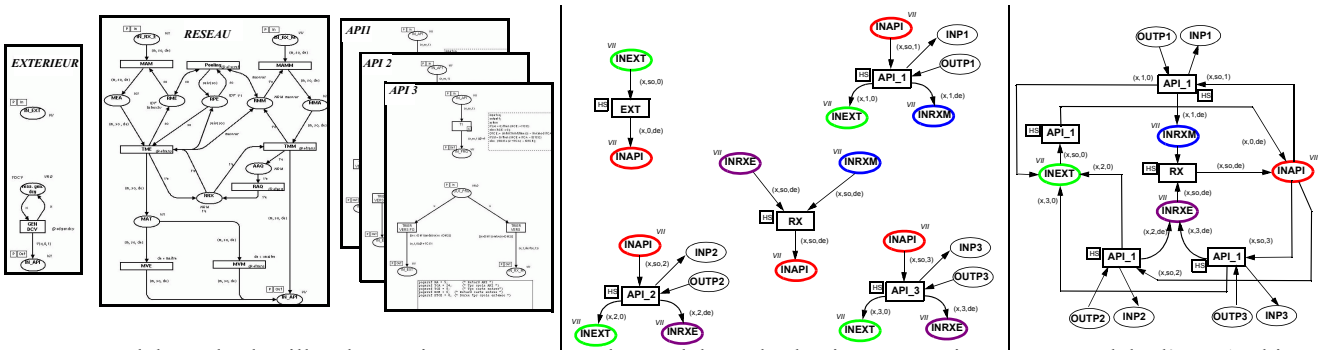
-c- Modèle RdP détaillé d'un réseau

Figure 33 : Structure et modèle RdP d'un réseau

Le modèle du comportement dynamique (figure 33c) est très élaboré, du fait des interruptions du maître sur le pooling des esclaves (les messages provenant du maître sont prioritaires sur les temps de parole des esclaves). Ce modèle est également générique. En effet, on peut instancier ce modèle si plusieurs réseaux du même type sont nécessaires ; il reste alors à définir les paramètres de chacune des instances (temps de transmission, nombre d'esclaves, durée d'une scrutation pour un esclave).

2.3.4.4 Modèle de l'Architecture Matérielle

Le modèle du comportement dynamique d'une Architecture Matérielle est obtenu par assemblage des modèles des équipements. L'opération s'effectue en fusionnant les places communes des différents équipements. Ainsi par exemple, les places d'entrée des API sont fusionnées. Toutes les informations pour les API sont regroupées dans une même place, ces informations étant codées sous la forme d'un triplet *(x, so, de)* composé d'une information *x*, de la provenance de l'information *so* et de sa destination *de*. Ainsi chaque modèle ne prendra que les informations qui lui sont destinées.
Il faut donc, en premier lieu, réaliser le choix et l'instanciation des modèles des équipements matériels dans la bibliothèque (figure 34a) ; puis construire les transitions de niveau supérieur (figure 34b) et enfin réaliser la réunion des places communes (figure 34c).



-a- Modèles RdP détaillés des équipements matériels

-b- Modèles RdP de niveau supérieur des équipements matériels

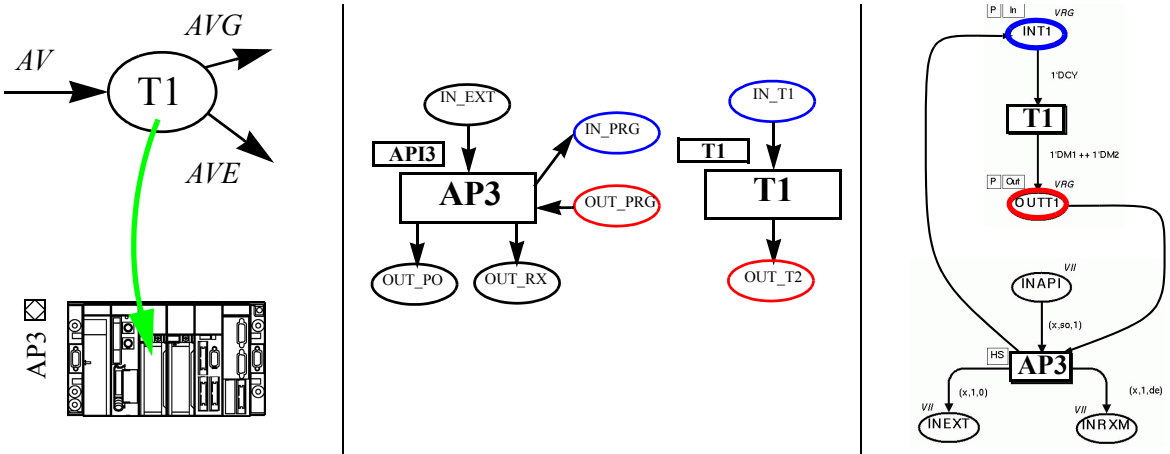
-c- Modèle d'une Architecture Matérielle

Figure 34 : Construction d'une AM à partir de modèles d'équipements

Le formalisme RdP selon Jensen offre la possibilité de déclarer un ensemble de places comme fusionnées sans avoir à réaliser la connexion dans le modèle réseau de Petri. Cela ne permet de représenter les liens entre les différents modèles, mais cela permet de simplifier la représentation lorsque le nombre de modèle est important en supprimant les liens qui le rendaient illisible.

2.3.5 Modélisation d'Architectures Opérationnelles

Une Architecture Opérationnelle est une projection de l'Architecture Fonctionnelle sur une Architecture Matérielle. Pour la conception du modèle dynamique de l'Architecture Opérationnelle, on réalise l'intégration des deux modèles de comportement dynamique. Il faut dans un premier temps avoir une représentation de chaque tâche fonctionnelle au niveau hiérarchique supérieur afin de pouvoir associer le modèle RdP d'une tâche avec celui d'un équipement matériel. Prenons par exemple, l'affectation de l'atome fonctionnel T1 sur l'API n°3 (figure 35a). Au niveau hiérarchique supérieur, le modèle de l'équipement matériel possède deux places qui correspondent aux places du modèle de l'atome fonctionnel (figure 35b). Dans le modèle de l'Architecture Opérationnelle, les places qui ont un même rôle sont fusionnées ; ce qui est le cas de l'association entre l'atome T1 et l'API n°3 (figure 35c).



-a- Affectation d'un atome à un équipement matériel actif

-b- Correspondance entre les modèles matériel et fonctionnel

-c- Modèle de l'association matériel-fonctionnel

Figure 35 : Projection d'une tâche dans un équipement matériel

En réalisant la modélisation de toutes les projections des atomes fonctionnels avec leurs équipements matériels associés, on construit le modèle de l'Architecture Opérationnelle. Dans le cas de la première Architecture Opérationnelle proposée, on obtient le modèle RdP de niveau hiérarchique supérieur représenté par la figure 36.

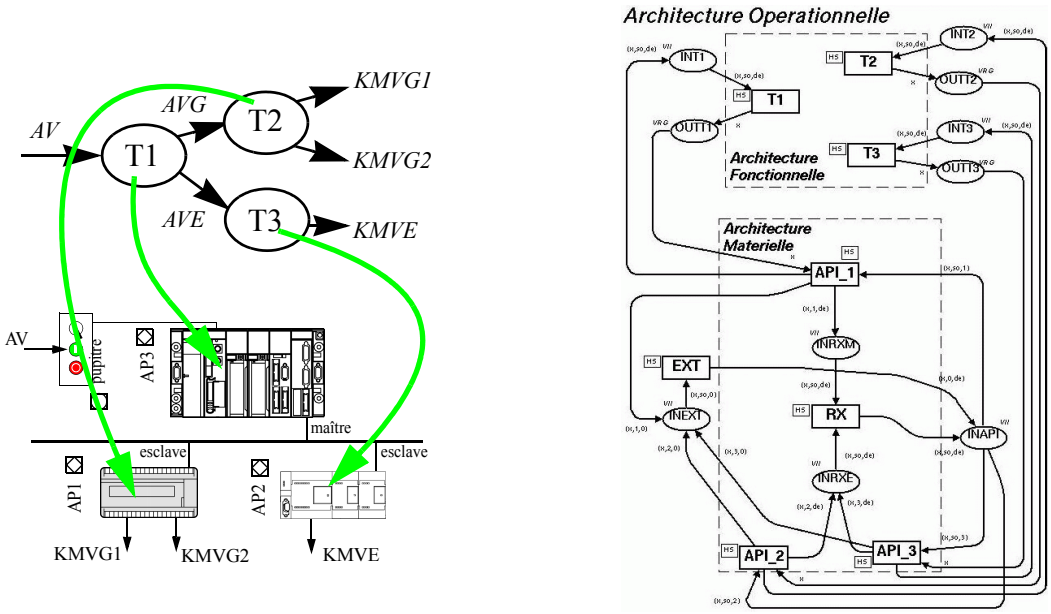


Figure 36 : Modélisation dynamique de l'Architecture Opérationnelle

Le placement des différentes tâches composant une Architecture Fonctionnelle ne suffit pas pour modéliser une Architecture Opérationnelle. En effet, il convient de plus d'indiquer le chemin des échanges de données entre les différentes tâches. Ces chemins doivent être parfaitement définis afin d'indiquer à chaque information circulant dans le modèle quel est l'équipement destination. Par exemple (fig. 37), l'information *AV* est émise par la tâche *T1* à destination de la tâche *T2*; ces deux tâches ont été respectivement placées dans les automates programmables *AP3* et *AP1*. Il faut définir que l'information *AV* en

sortie de l'automate *AP3* doit donc aller vers l'automate *AP1*, de plus ce trajet doit se faire via le réseau *RX*.

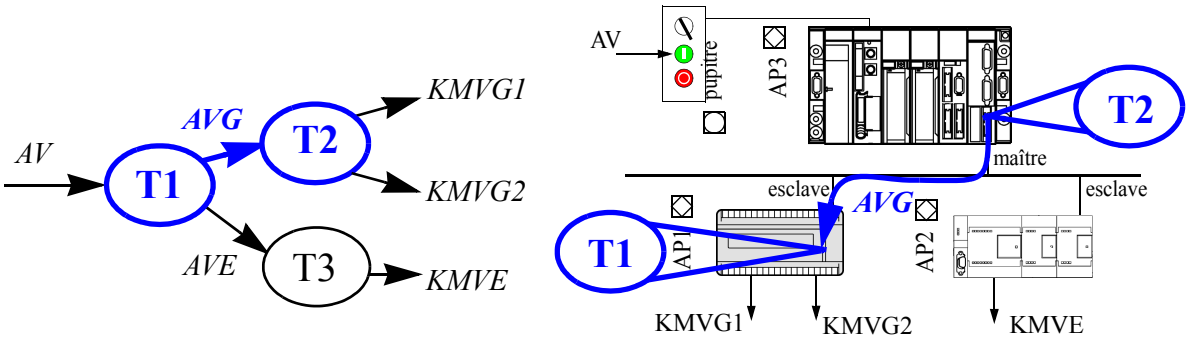


Figure 37 : Exemple de routage d'une information

Suivant l'information (notée *x*) et l'équipement matériel source (noté *so*), on peut déduire quel sera l'équipement matériel destinataire. D'après le modèle de l'API que nous avons choisi, les informations qui ont été traitées doivent être orientées. C'est le rôle de la fonction *dest* qui se place avant les aiguillages vers les cartes de sortie ou de coupleur réseau.
La fonction *dest* (dont la localisation est décrite sur la figure 31) doit être définie pour permettre le routage des informations d'un équipement matériel vers un autre (la figure 38 représente le cas de l'Architecture Opérationnelle avec trois API).

```
function dest (x, so) {case so of
          1 => (case x of
                  KMVG1 => 0 |
                  KMVG2 => 0) |
          2 => (case x of
                  KMVE => 0) |
          3 => (case x of
                  AVG => 1 |
                  AVE => 2);
```

Définitions du routage des informations

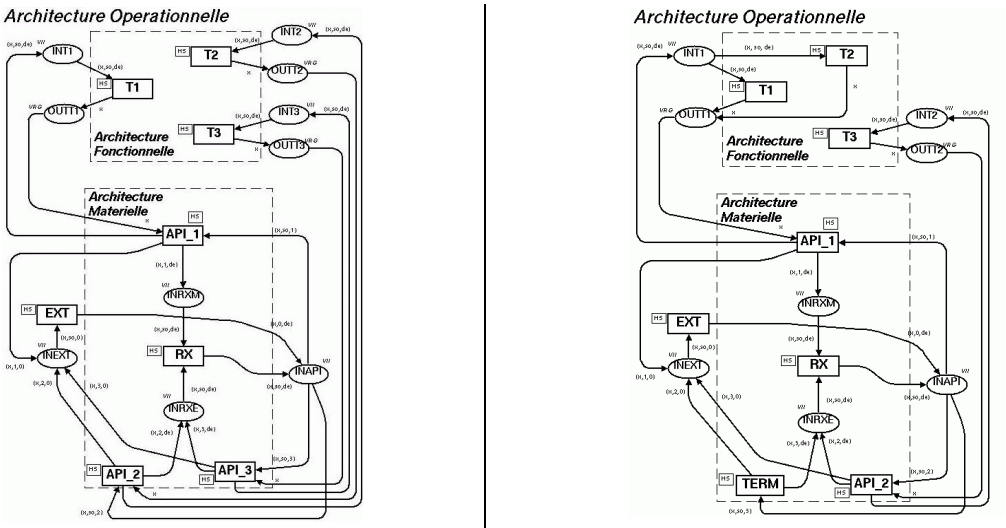
Figure 38 : Définition des routages d'informations dans la page de déclarations globales

Si nous reprenons l'exemple de l'information *AV* qui transite entre les atomes *T1* et *T2* qui sont respectivement projetés sur les *AP3* et *AP1* ; nous retrouvons dans la figure 38, que le résultat de *dest(AVG, 3)* est égal à 1 (les lignes correspondant dans le code sont indiquées en gras-italique sur la figure 38).

Le modèle de l'Architecture Opérationnelle est donc composé du modèle de comportement d'une projection de l'Architecture Fonctionnelle sur une Architecture Matérielle (fig. 39) et de la définition des trajets d'informations (fig. 40).

2.3.6 Préparation de la simulation

Un système de commande en «fonctionnement» est couplé à un système contrôlé et de plus avec son process il n'est pas «isolé» mais est soumis à des stimuli extérieurs tels que des lancements de production, des demandes de suivi, … Afin d'avoir une simulation réaliste du comportement du modèle précédemment obtenu, il est donc nécessaire de le compléter avec un modèle du système contrôlé ainsi qu'un



a - Configuration 3 API

b - Configuration 2 API + 1 terminal

Figure 39 : Modèles des Architectures Opérationnelles

```
function dest (x, so) {case so of
          1 => (case x of
                  KMVG1 => 0|
                  KMVG2 => 0) |
          2 => (case x of
                  KMVE => 0) |
          3 => (case x of
                  AVG => 1|
                  AVE => 2);
```

a - Configuration 3 API

```
function dest (x, so) {case so of
          1 => (case x of
                  AVG => 1 |
                  AVE => 2 |
                  KMVG1 => 0|
                  KMVG2 => 0) |
          2 => (case x of
                  KMVE => 0);
```

b - Configuration 2 API + 1 terminal

Figure 40 : Routages des informations

modèle des interactions avec l'extérieur. Nous appellerons «environnement de simulation» ces modèles complémentaires.

De plus, il nous faut évaluer les performances de type temps de réponse de notre modèle. Or le logiciel que nous avons choisi n'est pas un logiciel exécutant un langage de simulation mais un joueur de réseau de Petri. Cela implique que la gestion de l'échéancier, la mise en file ou le tirage de variables aléatoires sont gérés par Design/CPN mais que l'observation des performances nécessite de compléter le modèle avec des observateurs.

L'Architecture Opérationnelle ainsi enrichie d'un environnement de simulation et d'observateurs sera appelée par la suite Architecture de Simulation.

2.3.6.1 Modélisation de l'environnement de simulation

L'environnement de simulation se compose du modèle du système à contrôler et du modèles des stimuli extérieurs.

De nombreux travaux de recherche portent sur la modélisation de la Partie Opérative (PO) comme ceux de José Machado [MACHADO & al 2003], Véronique Carré Ménétrier [PHILIPPOT & al 2004] ou David Gouyon [GOUYON & al 2004]. Mais les performances que nous souhaitons évaluer portent sur le contrôle-commande et non sur le système dans son intégralité. Ainsi, les délais mesurés correspondent à des informations qui ne parcourent que l'Architecture Opérationnelle et non la PO.

Toutefois, la charge de l'Architecture Opérationnelle est elle fortement dépendante des interactions qu'elle aura avec la PO. La finesse du modèle de cette dernière correspondra donc à la finesse de la charge que l'on voudra associer aux différents équipements du contrôle-commande. Si l'équipement matériel est parcouru par une des informations relatives aux performances étudiées, il est nécessaire d'avoir une description réaliste de la charge de cet équipement (l'Annexe C propose une approche de modélisation d'éléments de la Partie Opérative). Dans le cas contraire, une description synthétique suffit amplement (délais constants ou tirés à partir d'une loi de répartition (fig. 41)).
Dans la suite du document, nous modéliserons de manière synthétique le comportement de la PO afin de nous concentrer la modélisation de l'architecture de contrôle-commande.

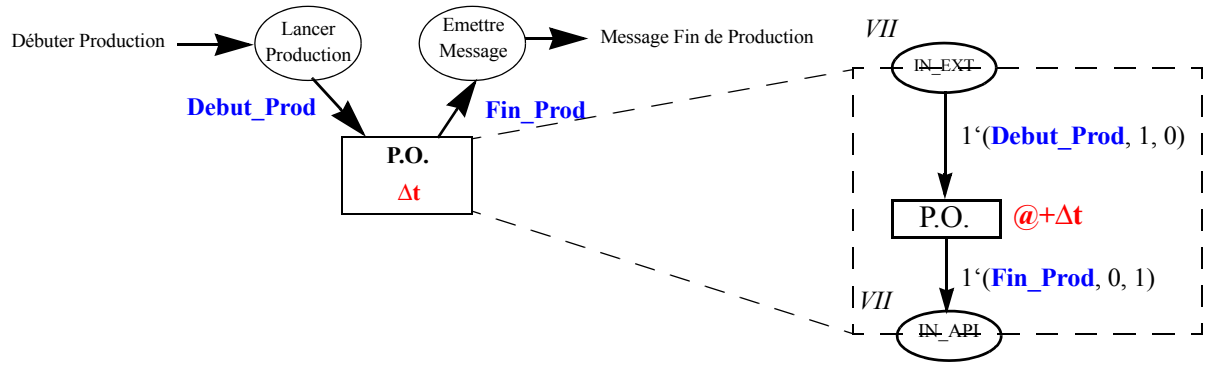


Figure 41 : Exemple de modèle de Partie Opérative

En ce qui concerne les stimuli, seuls les informations observées pour les performances ou les informations participant à la charge des équipements matériels doivent être modélisés (par exemple, un réseau peut avoir un comportement différent si on étudie le temps de transmission d'un événement seul ou d'un événement parmi un ensemble d'informations transitant par ce media).
Le principe de fonctionnement du modèle d'un générateur d'événement stimulus, consiste à placer régulièrement des jetons événement dans une place d'entrée du modèle de l'Architecture Opérationnelle. Ce générateur peut être périodique, aléatoire (la période de génération des jetons est choisie à l'aide d'une fonction de génération de nombres aléatoires) ou selon une progression mathématique (suite arithmétique, géométrique, …). Mais la structure de ce générateur reste globalement identique (fig. 42).

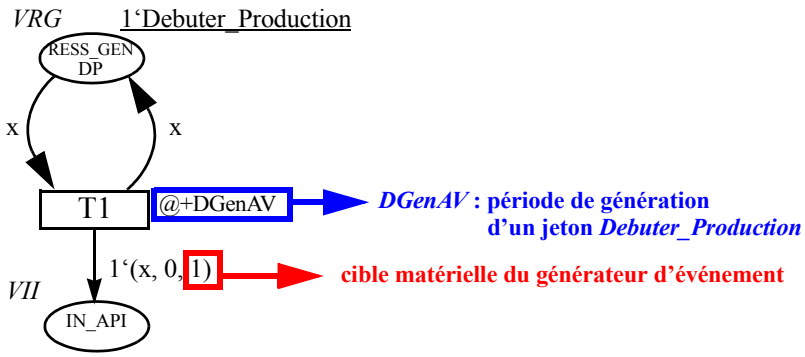


Figure 42 : Exemple de générateur d'événements

2.3.6.2 Modélisation des observateurs

Une fois les modèles des générateurs d'événements et du processus ajouté au modèle de l'Architecture Opérationnelle, la simulation peut être réalisée dans un contexte cohérent avec le futur fonctionnement du contrôle-commande. Mais pour juger des performances de l'architecture, il est nécessaire d'être capable d'observer l'évolution des paramètres devant être évalués durant cette simulation.
Le logiciel de simulation propose des fichiers traces (log files) de la succession des évolutions du modèle simulé. Cette fonctionnalité ne discernant pas les événements pertinents des événements non pertinents, elle génère des fichiers de taille démesurés (plusieurs Giga-octets) et pénalise le temps de simulation. Dans un soucis d'améliorer les performances de simulation, il est préférable de particulariser l'observation du modèle simulé.
Étant donné que les performances que nous étudions dans ce chapitre sont temporelles, nous devrons donc mesurer des temps de réaction ou des temps de transmissions d'informations. Nous utiliserons pour cela des observateurs en deux parties :

- la partie observant le début d'une transmission à évaluer pour en mémoriser la date ;
- la partie observant les effets d'une transmission pour en calculer la durée de transmission.

Nous utiliserons une possibilité du logiciel Design CPN, qui permet d'associer à une transition un code ML (de l'anglais *Meta Langage*, c'est un langage de programmation généraliste fonctionnel). Ce code est exécuté à chaque tirage de la transition associée. Les codes observateurs permettent d'écrire des données dans des fichiers, il n'y a donc pas surcharge de la mémoire avec les données observées au cours de la simulation. Le code observateur doit donc permettre de mesurer le temps Δt entre un événement *Ev* et sa réaction *Re*.

Dans le cas de notre exemple, il nous faut mesurer les délais entre la demande d'arrêt et les actions effectives sur les vannes correspondant pour vérifier si ces délais sont conformes par rapport au cahier des charges. Il est possible de faire une unique simulation avec tous les observateurs ou de faire diverses simulations avec à chaque fois une performance à observer. Sachant que la simulation est la partie qui est la plus consommatrice de temps, nous choisirons de placer les observateurs de toutes les performances à évaluer sur un même modèle pour faire une unique simulation.
On doit déterminer les informations qu'il faut observer dans le modèle pour étudier les performances attendues. L'événement à observer étant l'apparition de l'information *AV*, les réactions attendues sont les sorties *KMVG1*, *KMVG2* et *KMVE*. Sachant que *KMVG1* et *KMVG2* sont deux sorties d'un même automate programmable et qu'elles sont émises par une même tâche, le décalage temporel entre les deux sera négligeable (une simulation pourrait le mettre en évidence), donc nous n'observerons qu'une seule des deux (*KMVG1*).
L'observateur d'événements ne fera que noter la date d'occurrence de l'événement et les observateurs des réactions noteront les dates des réactions correspondantes. Le dernier arrivé écrira dans le fichier les trois données : date de début, date de fin gaz et date de fin électricité. Pour savoir lequel est le dernier arrivé, on utilisera la variable booléenne *PremierArrive*.
On a donc trois observateurs : le premier (début de mesure) doit être placé pour permettre la détection de l'apparition de l'événement *AV*, il doit donc être placé dans la transition du générateur de jetons *AV* (fig. 43).

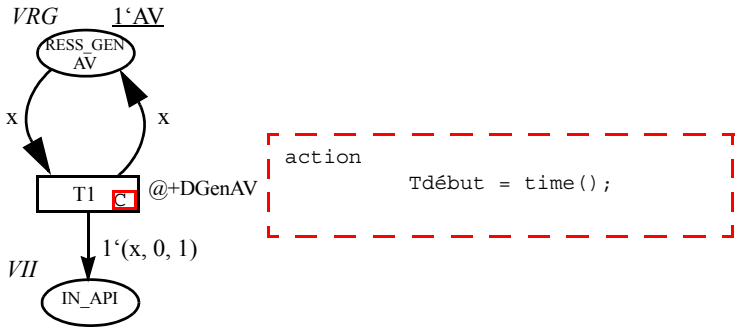


Figure 43 : Localisation du code d’observation d’événement AV

Les deux observateurs de réactions se placent dans les transitions correspondant aux cartes de sorties des Automates programmables (fig. 44).

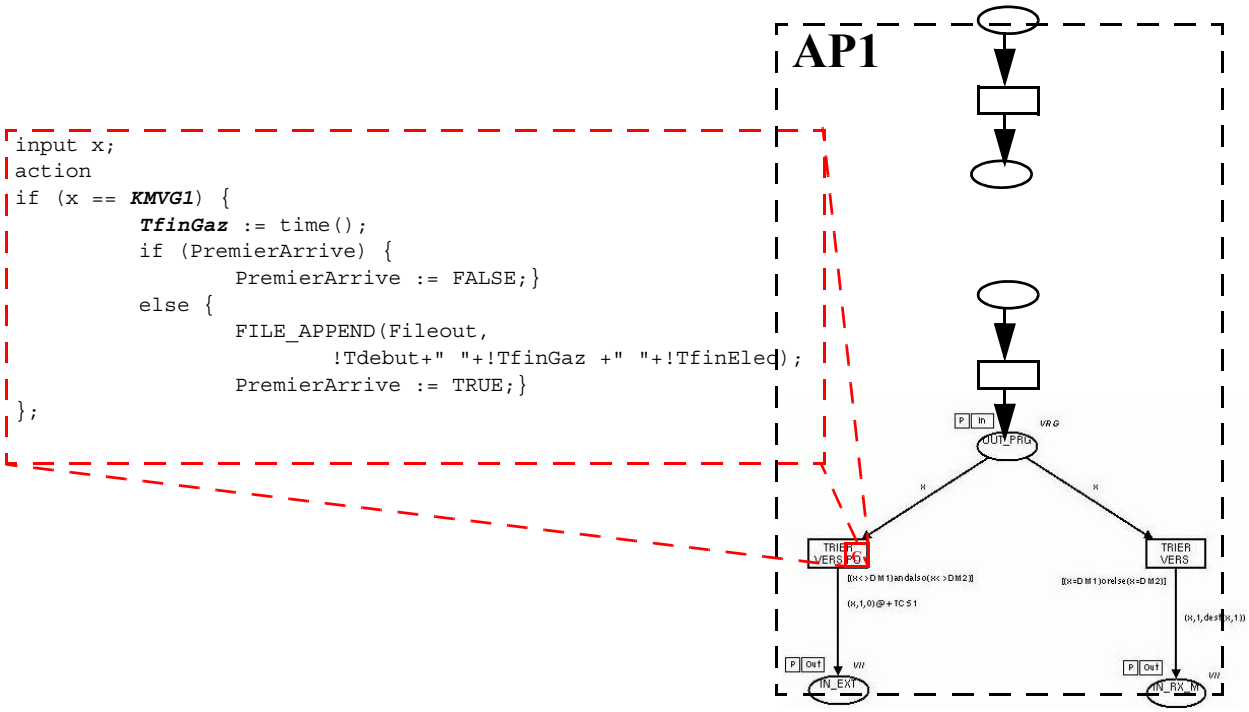


Figure 44 : Localisation des codes d’observation de la réaction KMVG1

Le code de l’observation de la réaction *KMVE* est similaire à celui de la figure 44, sauf qu’il faut remplacer le code indiqué en italique (*KMVG1* par *KMVE* et *TfinGaz* par *TfinElec*).
Nous disposons ainsi d’une technique d’observation des événements pertinents qui maintien un bon niveau de performance à la simulation tout en produisant un fichier trace très simple à exploiter.

2.3.7 Simulation et présentation des résultats obtenus [MEUNIER & al 2000b]

La simulation dépend, comme on le verra dans le chapitre suivant, des modèles utilisés pour concevoir le modèle du comportement dynamique de l’Architecture Opérationnelle à évaluer. De plus, ce qui nous intéresse dans la simulation, ce n’est pas le temps simulé sur le système, mais les stimulations qui ont été effectuées afin de voir leurs réactions. On donnera donc un rapport entre le nombre de stimulations

effectuées (on pourra indiquer la période de stimulation dans la simulation) et le temps de simulation (on indiquera le type d'ordinateur utilisé pour exécuter le logiciel de simulation).
Au cours de la simulation, un fichier qui collecte les données que l'on souhaite observer est créé. Ce fichier est généré par le code des observateurs, il contient des données brutes, c'est-à-dire que les données sont les temps de réaction ou les délais de transmission successifs qui ont été mesurés au cours de la simulation. Si l'on faisait un graphe à partir de ces données, on obtiendrait la figure 45.

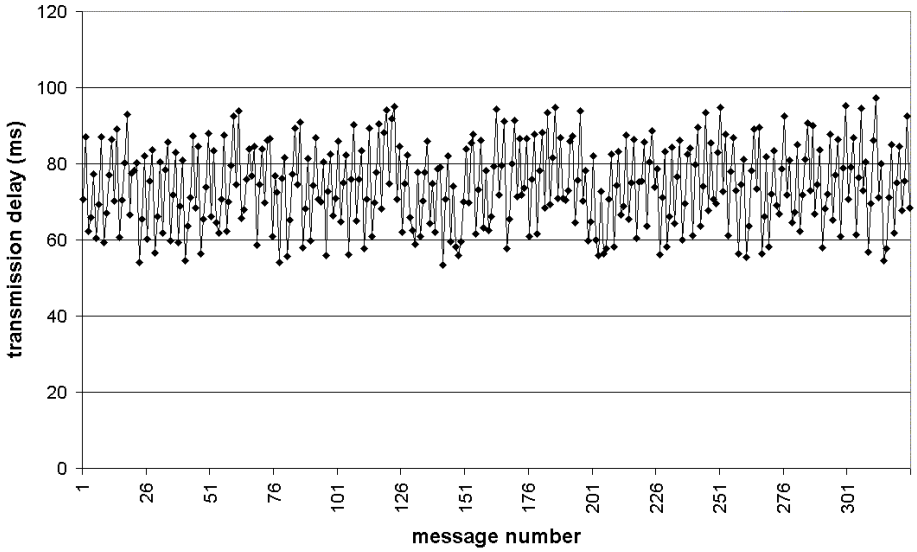


Figure 45 : Exemple de données obtenues par simulation

Ces données n'étant pas exploitables directement pour analyser les performances du système, on doit faire des traitements statistiques afin de pouvoir juger des performances correspondant aux mesures. A partir des données, on génère donc un histogramme pour observer la répartition des résultats; on pourra aussi en déduire les minimum et maximum des délais, ainsi que la valeur moyenne (fig. 46).

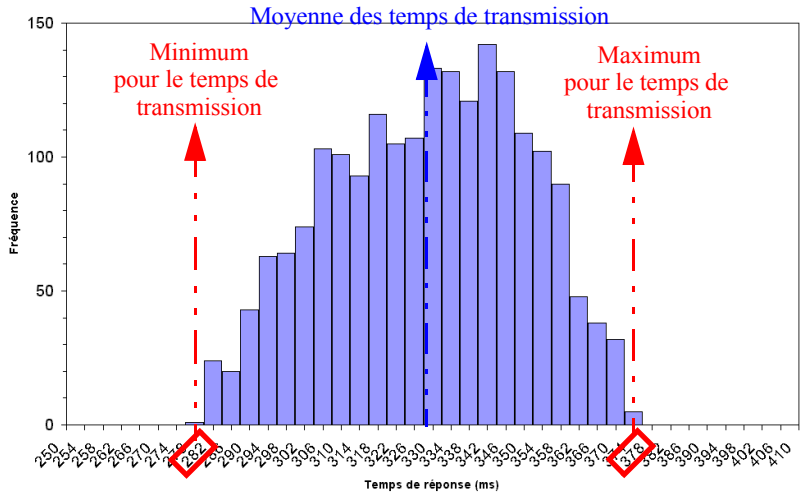


Figure 46 : Traitement des données de la simulation

La simulation est réalisée pour 360 événements *AV* avec une période *DGenAV* de 1 seconde soit un horizon temporel de simulation de 1 heure pour une durée de simulation de 1 minute et 30 secondes sur un ordinateur dont le processeur est un Pentium IV à 3,2 GHz avec 1Go de mémoire RAM avec un sys-

tème d'exploitation Linux. Les données brutes du fichier de données sont : la date de l'événement *AV*, la date de la réaction *KMVG1* et la date de la réaction *KMVE*. Pour évaluer les performances attendues du cahier des charges, il faut maintenant calculer :

- Le retard *ΔG* entre l'événement *AV* et la réaction *KMVG1* (ΔG = KMVG1 - AV);
- Le retard *ΔE* entre l'événement *AV* et la réaction *KMVE* (ΔE = KMVE - AV);
- Le retard *ΔGE* entre *KMVG1* et la réaction *KMVE* (ΔG = | KMVG1 - KMVE |);

On génère ensuite les histogrammes pour ces trois résultats de mesure (fig. 47).

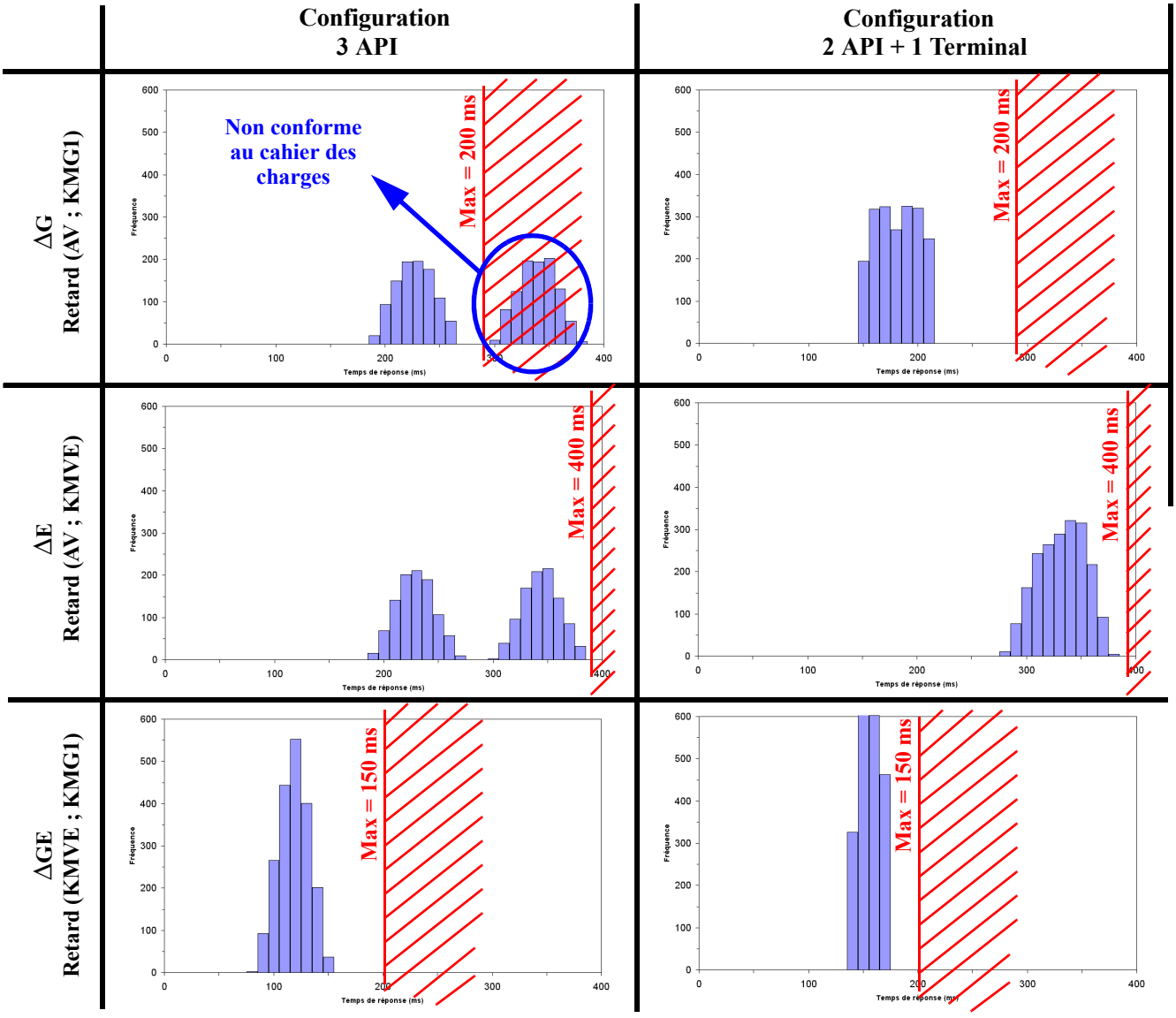


Figure 47 : Histogrammes des résultats de mesures

Grâce aux résultats obtenus, l'architecte peut placer sur ces histogrammes les contraintes opérationnelles définies par le cahier des charges (indiquées en rouge sur la figure 47). Il peut ainsi vérifier si les Architectures Opérationnelles qu'il a sélectionnées répondent aux contraintes opérationnelles attendues. Dans notre exemple, on remarque que tous les résultats sont conformes aux attentes sauf concernant *ΔG* (le retard entre *AV* et *KMVE*) pour l'architecture à trois automates programmables. L'architecte peut donc exclure cette solution car elle ne répond pas au cahier des charges. Dans le cas où il y aurait

plusieurs solutions acceptables, d'autres critères de choix seraient à prendre en compte comme par exemple le coût de l'architecture. Par contre, dans le cas où aucune solution ne serait conforme aux cahier des charges, l'architecte devrait étudier les sources de non-conformité afin de pouvoir obtenir au moins une solution correcte.

2.4 Conclusion

Dans ce second chapitre, nous avons présenté à partir d'un exemple simple notre démarche permettant à l'architecte d'évaluer les performances temporelles des Architectures Opérationnelles qu'il a conçues. Cette démarche a amené au choix d'une technique de modélisation RdP Colorés et Temporisés et d'un outil support pour éditer et simuler Design CPN. Cette méthode comporte certaines étapes spécifiques à chaque cas d'étude (modélisation de l'AF, de l'AO et de l'AS) et d'autres plus génériques qui s'appuient sur l'utilisation d'une bibliothèque de modèles (modélisation de l'AM grâce à l'utilisation de modèles de composants pré-définis).

Nous allons maintenant développer ces deux aspects (spécifiques et génériques) en proposant dans le Chapitre 3 des guides méthodologiques pour faciliter la modélisation des architectures Fonctionnelles, Opérationnelles et de Simulation. Le Chapitre 4 présentera une étude détaillée de la création et du choix des modèles des équipements de commande.

Chapitre 3

Obtention des modèles spécifiques

« Si un aveugle guide un aveugle, tous les deux tomberont dans un trou »

Saint Luc - Les évangiles

Dans ce chapitre, nous détaillons chacune des phases de modélisation comportementale des architectures de contrôle-commande.
Ainsi, nous commençons par présenter la modélisation des éléments de l'Architecture Fonctionnelle, point d'entrée de notre démarche.
Puis, nous exposons le processus de modélisation de l'Architecture Matérielle en partant des modèles des équipements élémentaires la composant (dont la conception et la validation sont l'objet du Chapitre 4).
Ensuite, nous explicitons comment est construit le modèle de comportement de l'Architecture Opérationnelle à partir des modèles définis dans les phases précédentes.
Enfin, nous présentons la phase de préparation à la simulation, qui consiste pour l'essentiel à adjoindre aux modèles précédents des générateurs de scénarii et des observateurs de performance.

3.1 Introduction

Notre démarche d'évaluation de performances est basée sur la simulation de modèles réseaux de Petri. Certains de ces modèles (des équipements de l'Architecture Matérielle) sont génériques et peuvent être réutilisés d'une étude à l'autre. Mais les autres modélisations sont spécifiques aux architectures étudiées. Ainsi, il est nécessaire de réaliser un travail de modélisation de l'Architecture Fonctionnelle, de l'Architecture Matérielle (à partir de ses composants génériques), de l'Architecture Opérationnelle, et de l'Architecture de Simulation.

Cette partie a pour but de formaliser ces démarches de création de modèles en proposant des lignes directrices qui faciliteront leur obtention. Nous nous intéresserons donc dans un premier temps à modélisation des atomes composant l'Architecture Fonctionnelle, puis nous décrirons brièvement le processus d'obtention de l'Architecture Matérielle à partir de ses composants génériques. Nous développerons alors le processus de modélisation de l'Architecture Fonctionnelle et enfin, nous enchaînerons sur la préparation du modèles de l'Architecture de Simulation.

3.2 De l'Architecture Fonctionnelle aux modèles des tâches fonctionnelles

3.2.1 Introduction

Comme nous l'avons décrit auparavant dans le paragraphe 1.2.1, p. 26, l'Architecture Fonctionnelle est unique pour un cahier des charges donné. Ce qui implique qu'elle doit être complètement définie à chaque fois que l'on traite une nouvelle application.
Nous nous proposons donc d'expliciter ce processus en décrivant au cours des deux paragraphes suivants, la modélisation des tâches fonctionnelles, et de la description des flux d'informations entre eux.

3.2.2 Modélisation du comportement dynamique d'une tâche fonctionnelle

L'Architecture Fonctionnelle est décrite par ses tâches fonctionnelles et par les liens qui les relient. Il s'avère que la complexité des tâches fonctionnelles est variable, elle peut avoir un comportement simple ou complexe. Dans le cas des comportements simples, nous proposons une représentation quasi-générique alors que pour les comportements complexes, un travail de modélisation de l'architecte est nécessaire.

L'étude de temps de transmission dans une architecture de contrôle-commande correspond à l'observation du temps de propagation d'une information dans une chaîne fonctionnelle dite «critique». On considérera qu'une tâche fonctionnelle appartiendra à une chaîne critique si elle intervient dans la propagation d'un événement qui fait l'objet de performances temporelles spécifiée dans le cahier des charges de l'application. Les tâches fonctionnelles composant cette chaîne critique doivent donc avoir leurs comportements dynamiques correctement définis alors que les tâches fonctionnelles qui n'en font pas partie peuvent avoir une description plus approximative. Le niveau de granularité de la description est proportionnel à l'implication de la tâche dans la chaîne critique.

Nous décrirons dans les paragraphes suivants les modélisations des informations circulant dans une Architecture Fonctionnelle, puis nous présenterons la modélisation de flux informationnels. Alors, nous présenterons la modélisation des tâches fonctionnelles appartenant à la chaîne critique et enfin nous décrirons brièvement la modélisation des tâches fonctionnelles hors chaîne critique.

3.2.2.1 Modélisation des informations d'une Architecture Fonctionnelle

L'architecture Fonctionnelle est composée de tâches qui sont interconnectées entre elles. Chacune de ces tâches possède des entrées et des sorties (fig. 48).

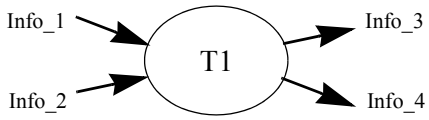


Figure 48 : Exemple de tâche fonctionnelle

Pour notre exemple, la tâche *T1*, reçoit les informations *Info_1* et *Info_2*, et elle renvoie les informations *Info_3* et *Info_4*. On peut donc définir les ensembles des entrées et des sorties de cet atome :

$$\begin{aligned} IN &= \{\text{Info\_1}, \text{Info\_2}\} \\ OUT &= \{\text{Info\_3}, \text{Info\_4}\} \end{aligned} \tag{2}$$

Les informations de sorties d'une tâche étant les informations d'entrée d'une autre tâche, il est nécessaire de définir l'ensemble *VRG (VaRiables Globales)* des informations pouvant être consommées ou produites par l'atome à modéliser.

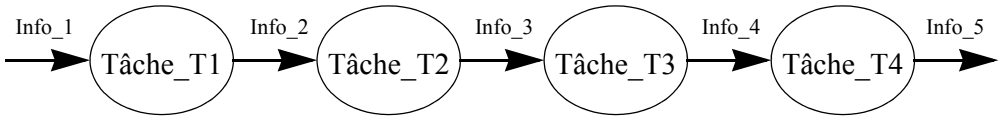


Figure 49 : Exemple d'un ensemble de tâches fonctionnelles

Dans l'exemple décrit sur la figure 49, on a l'ensemble *VRG* suivant :

$$VRG = \{\text{Info\_1}, \text{Info\_2}, \text{Info\_3}, \text{Info\_4}, \text{Info\_5}\} \tag{3}$$

Pour faciliter l'écriture des variables, nous conseillons d'utiliser des notations abrégées :

$$VRG = \{\text{I1}, \text{I2}, \text{I3}, \text{I4}, \text{I5}\} \tag{4}$$

La modélisation en réseau de Petri Colorés Temporisés selon le formalisme RdP selon Jensen, amène à décrire cet ensemble d'informations sous la forme d'une «couleur» (fig. 50).

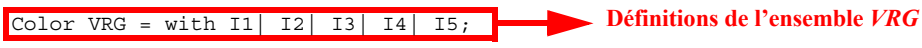


Figure 50 : Extrait de déclarations globales, définition de l'ensemble VRG

3.2.2.2 Modélisation des flux informationnels d'une Architecture Fonctionnelle

Les flux informationnels entre les tâches fonctionnelles ne peuvent pas être modélisés au moment du processus de modélisation de l'Architecture Fonctionnelle, car ils dépendent de la distribution de ces tâches dans les équipements de l'Architecture Matérielle. Ce sera donc lors de la modélisation de l'Architecture Opérationnelle que l'on réalisera la modélisation des flux informationnels. Toutefois, il est possible à ce stade de modélisation de réaliser un travail préparatoire permettant de simplifier la future modélisation des flux informationnels. Ce travail préparatoire consiste à faire une description de chacun des flux. Nous considérerons que :

- Un flux d'une information a toujours au moins une origine,
- Un flux d'une information a toujours au moins une destination,
- Pour une origine donnée, un flux d'une information a une et une seule destination

Pour chaque flux, on précisera son ou ses origines ainsi que sa destination pour chaque origine.

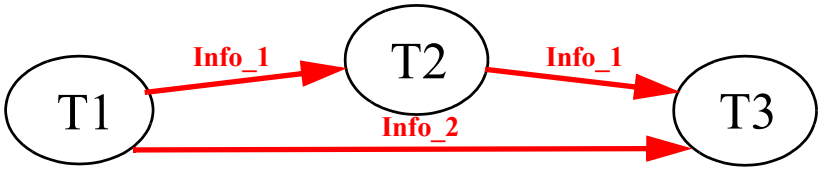


Figure 51 : Exemple de flux informationnels

Par exemple, si l'on considère la figure 51, nous avons les descriptions de flux suivantes :

Tableau 2 : Description des Flux informationnels

Nom du Flux d'information	Origine	Destination
Info_1	T1	T2
Info_1	T2	T3
Info_2	T1	T3

3.2.2.3 Modélisation des tâches fonctionnelles de la chaîne critique

Les tâches fonctionnelles de la chaîne critique (que nous appellerons «tâches critiques») peuvent être assimilées à des tâches consommant des informations pour en produire d'autres. Leur comportement dynamique correspondant à la conversion d'un ensemble d'informations en d'autres se modélise simplement en consommant les jetons information (de type *VRG*) qui doivent être convertis et de générer des jetons correspondant aux informations (de type *VRG*) résultat de la conversion (fig. 52).

Une tâche fonctionnelle «critique» est modélisée par une transition et par deux places. La place amont correspond aux informations d'entrée tandis que la place aval contient les informations de sortie. L'arc amont a pour poids l'union (modélisée avec ++ en formalisme RdP selon Jensen) des informations nécessaires à la réalisation de la tâche ; et l'arc aval a pour poids l'union des messages émis par la réalisation de cette tâche.
Le modèle d'une tâche «critique» est donc un réseau de Petri simple composé de deux places et d'une transition. La coloration permet de différencier les différents messages ou informations qui transitent dans des mêmes places.

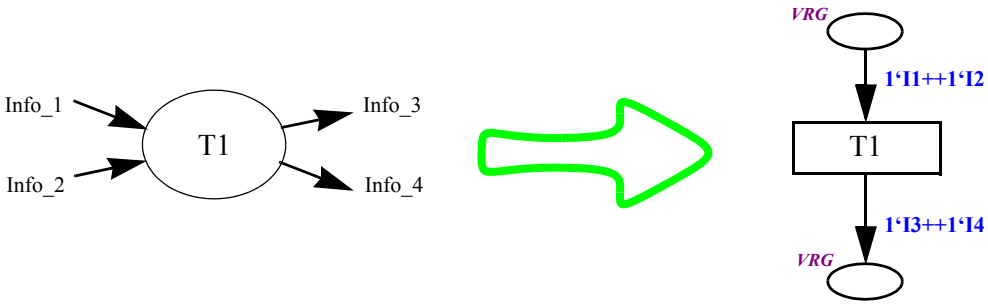


Figure 52 : Modélisation du comportement dynamique d’une tâche fonctionnelle «critique»

3.2.2.4 Modélisation des tâches fonctionnelles hors chaîne critique

Les tâches fonctionnelles hors chaîne critique ne participent pas directement au temps de transmission d’une information. Toutefois, ces tâches définissent un comportement dynamique global de l’ensemble du système, il est donc nécessaire de les modéliser sous peine de ne pouvoir étudier que des performances hors-charge.
Le comportement dynamique de ces tâches doivent être défini suivant le niveau d’implication dans la charge de l’architecture. Ainsi, un ensemble de tâches fonctionnelles qui ne participent qu’à la charge d’un unique équipement de traitement sera modélisé comme une charge unique de cet équipement car l’influence de l’ensemble ne sera que locale. La modélisation de la charge locale s’effectuera par le biais d’un temps de calcul dans des équipements en fonction des tâches qui lui seront associées.
Par contre, toute tâche fonctionnelle qui apportera une charge sur un équipement de communication devra être totalement modélisée car son influence sera globale et donc aura un impact sur les performances de la chaîne critique. Étant donné que le comportement attendu de ces tâches varie d’une Architecture Fonctionnelle à l’autre, il n’y a pas modèle générique permettant de représenter le comportement dynamique des tâches «non-critiques» chargeant l’architecture.
Toutefois, il est possible de proposer une approche générique permettant la représentation de ces ensembles de tâches :

- Dans un premier temps, on identifie les informations dites «de charge» entre les tâches fonctionnelles «non-critiques» qui circuleront sur les moyens de communication, et on regroupe les tâches fonctionnelles «non-critiques» associées à un même équipement (fig. 53).

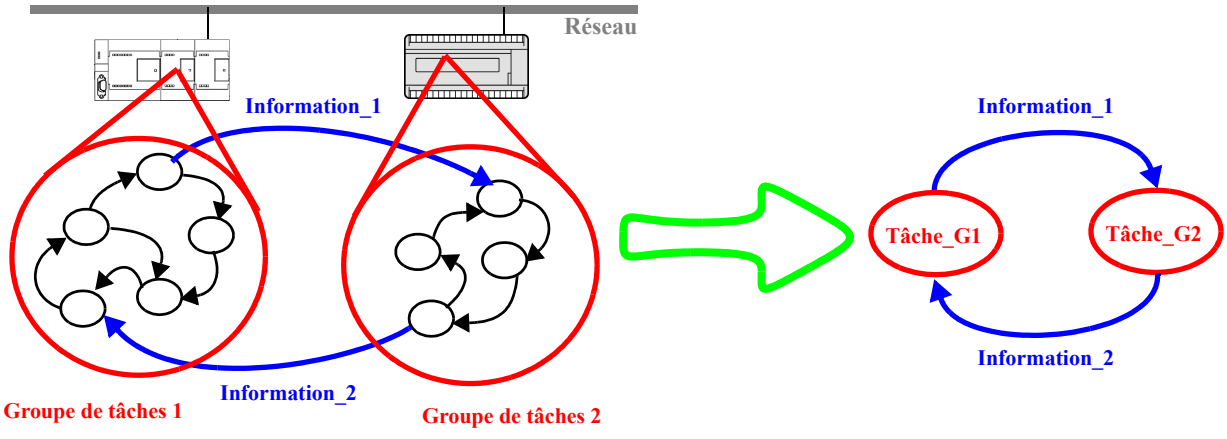


Figure 53 : Exemple de regroupement de tâches fonctionnelles «non-critiques»

- Dans un deuxième temps, on doit définir le comportement dynamique de ces groupes de tâches fonctionnelles. Pour cela, on peut réaliser des graphes d'état que l'on traduira en réseau de Petri. Les modèles comportementaux associés à ces groupes devront essentiellement définir la consommation-production des informations «de charge».

Nous ne développerons pas dans ce chapitre de méthode permettant la construction de modèles comportementaux complexes. Toutefois, afin d'aider l'architecte durant la phase de modélisation, nous proposons en Annexe A :

- un exemple de modélisation d'une tâche fonctionnelle temporisée ;
- un exemple de modélisation d'un comportement complexe spécifié à l'aide d'un graphe d'état ;
- un exemple de modélisation de comportement spécifié en Grafcet.

3.3 Des modèles d'équipements génériques à la modélisation de l'Architecture Matérielle

Bien que les modèles comportementaux des équipements qui composent l'Architecture Matérielle soient génériques, l'Architecture Matérielle est dépendant de chaque installation et donc son modèle est spécifique. Il est donc nécessaire d'expliciter la modélisation de l'Architecture Matérielle.

Rappelons que la représentation hiérarchique du formalisme RdP selon Jensen permet de visualiser les modèles des équipements matériels sous la forme d'une unique transition avec toutes les places d'entrée et de sortie du modèles. On dit que cette transition sert de substitution à un réseau de Petri. Cette transition de substitution est repérée par HS dans les représentations de réseaux de Petri sous Design CPN. Prenons par exemple l'Architecture Matérielle représentée par la figure 54, elle est composé de cinq Automates Programmables Industriels communiquant via deux réseaux.

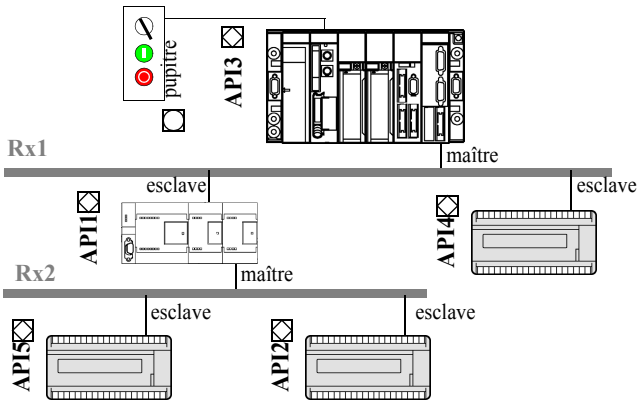


Figure 54 : Exemple d'Architecture Matérielle

On peut représenter chacun de ces équipements matériels par une unique transition. Un automate Programmable Industriel aura :

- une place *INAPI* pour les événements d'entrées correspondant aux cartes d'entrées ;
- deux places *INTT* et *OUTTT* pour connecter les modèles des atomes fonctionnels ;
- une place *INEXT* correspondant aux cartes de sorties ;
- une éventuelle place *INMRX* pour l'envoi des messages sur un réseau dont l'API est maître ;
- une éventuelle place *INERX* pour l'envoi des messages sur un réseau dont l'API est esclave.

Chacun des équipements de l'Architecture Matérielle de la figure 54 se traduit donc par une unique transition de subtitution comme sur la figure 55.

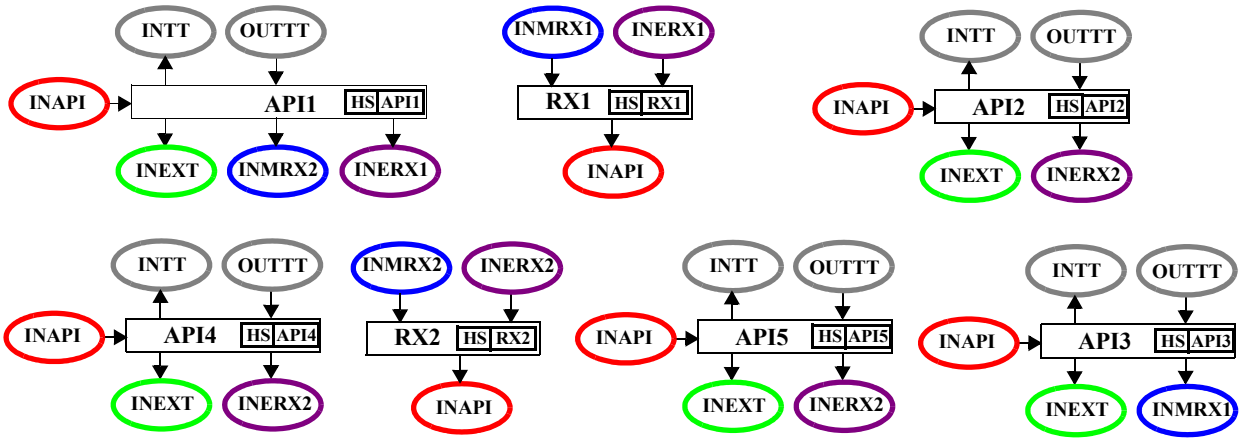


Figure 55 : Correspondance des entrées et des sorties des différents modèles

Chaque transition de substitution est ensuite associée à un réseau de Petri qui traduit le comportement de l'équipement (fig. 56).

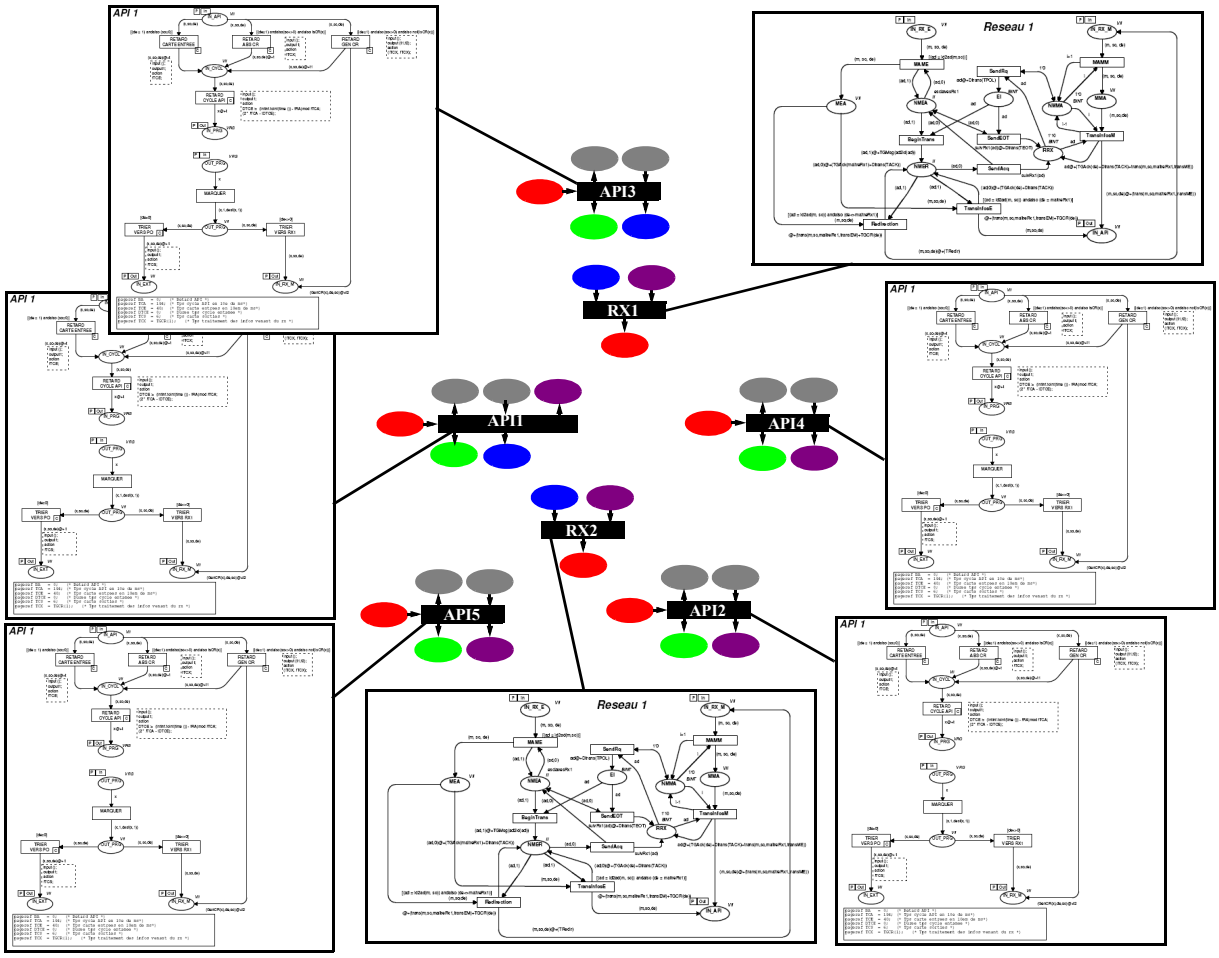


Figure 56 : Association Transition de substitution - Modèle réseau de Petri

Pour assembler l'Architecture Matérielle, il faut regrouper toutes les places qui ont la même sémantique. Pour réaliser ce regroupement, deux solutions sont possibles :

La première consiste à réaliser des connexions en utilisant la structure du réseau de Petri, en reliant les transitions à une même place. Par exemple, toutes les places *INAPI* sont confondues car les modèles des équipements matériels sélectionnent parmi l'ensemble des messages, ceux qui leur sont destinés. Par contre on ne réunit pas les places *INTT* et *OUTTT* correspondant aux connexions des modèles des tâches fonctionnelles, car chaque tâche est affectée à un unique équipement matériel actif. Dans le cadre de notre exemple, l'assemblage des équipements matériels est décrit par la figure 57.

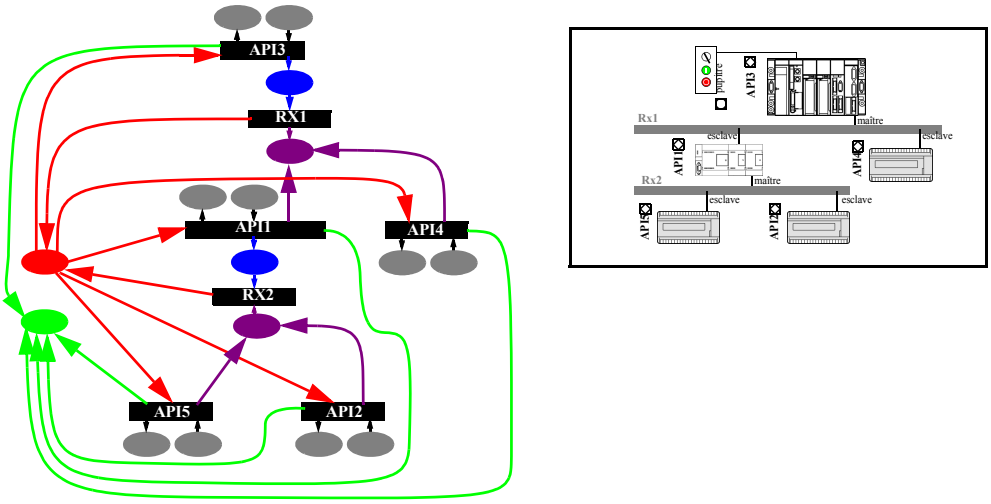


Figure 57 : Assemblage de l'Architecture Matérielle

Une fois les transitions de substitution des équipements matériels disposées (fig. 58a), il faut supprimer toutes les places surnuméraires (fig. 58b), et enfin on recrée les arcs qui ont été supprimés durant l'étape précédente (fig. 58b).

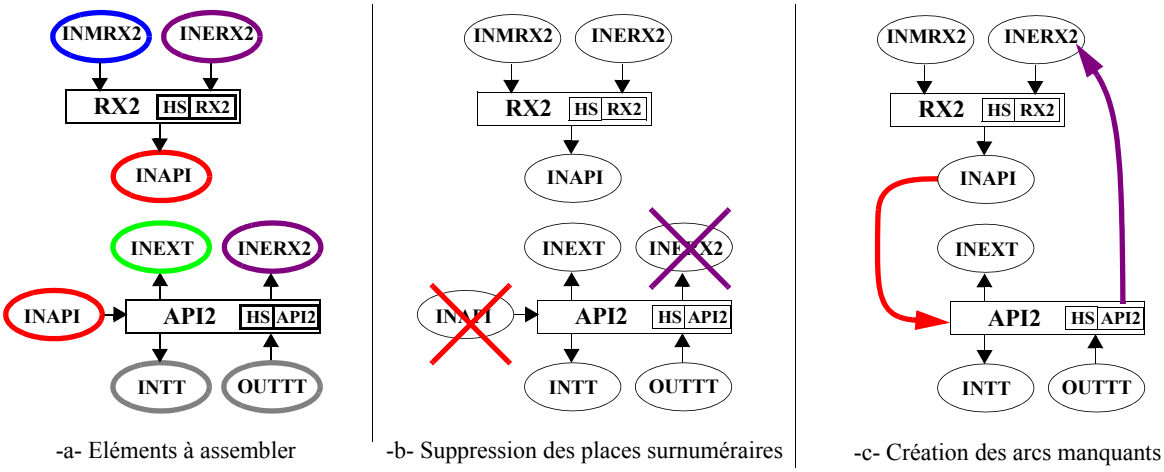


Figure 58 : Méthode d'assemblage par connexion de deux équipements matériels

Le poids des arcs qui doivent être recréés est *(x, so, de)*, la forme générique des informations circulant entre les équipements matériels.

- La seconde méthode consiste à utiliser une possibilité du formalisme RdP selon Jensen qui permet de déclarer un ensemble de places comme fusionnées, c'est à dire qu'elles sont considérées comme étant plusieurs «images» d'une même place. Dans le logiciel Design CPN, à coté de chaque place fusionnée apparaît alors un marqueur FG accompagné du nom de fusion.

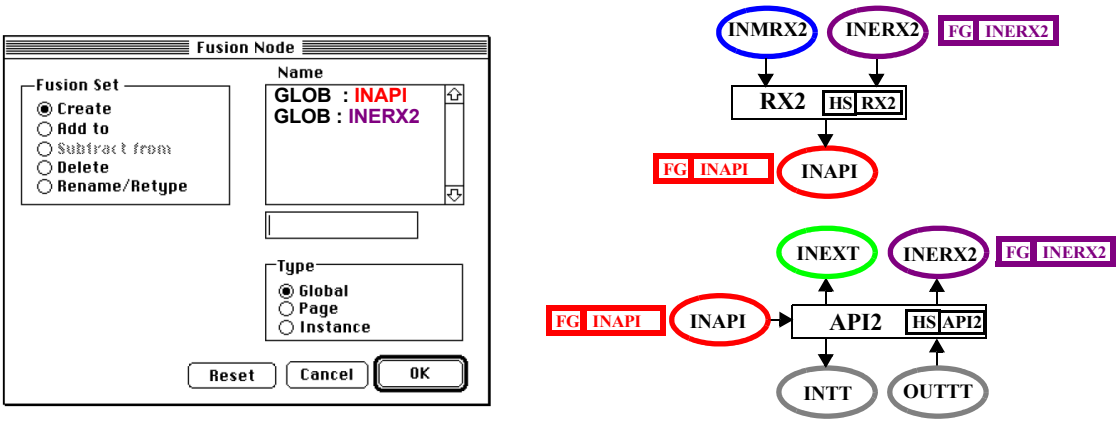


Figure 59 : Méthode d'assemblage de deux équipements matériels par fusion de places

Les deux solutions confèrent au RdP exactement le même comportement. Toutefois, la première méthode permet de représenter structurellement les liens entre les différents matériels, mais surcharge la représentation dans le cas de d'architectures composées de nombreux équipements. La seconde méthode ne permet pas de représenter la structure de l'architecture, mais est nettement plus lisible.

3.4 Modélisation de l'Architecture Opérationnelle

Dans ce chapitre nous nous intéresserons à la phase de modélisation de l'Architecture Opérationnelle. Enfin, nous étudierons l'impact de cette modélisation sur le processus d'évaluation de performances.
La répartition de l'Architecture Fonctionnelle sur une Architecture Matérielle est définie par l'Architecture Opérationnelle (comme nous l'avons vu dans les parties précédentes). Il est donc nécessaire de modéliser l'association des tâches fonctionnelles avec des équipements matériels, mais également de modéliser les flux informationnels entre les tâches fonctionnelles.

3.4.1 Modélisation de l'association d'une tâche avec un équipement matériel

Comme nous l'avons présenté sous paragraphe 2.3.4.4, p. 55, le formalisme RdP selon Jensen permet de représenter les réseaux de Petri sous une forme hiérarchique en les faisant apparaître sous la forme d'une unique transition dont le contenu est décrit dans une autre page. Nous utiliserons ce principe pour simplifier nos modèles.
En règle générale, le modèle d'un équipement matériel comporte une place d'entrée des tâches fonctionnelles et une place pour les sorties des tâches fonctionnelles. Associer une tâche fonctionnelle à équipement matériel de traitement consistera donc à :

- associer la place d'entrée des tâches fonctionnelles aux différentes places d'entrée de chaque tâche fonctionnelle
- et de manière similaire, associer la place de sortie des tâches fonctionnelles aux différentes places de sorties de chaque tâche fonctionnelle.

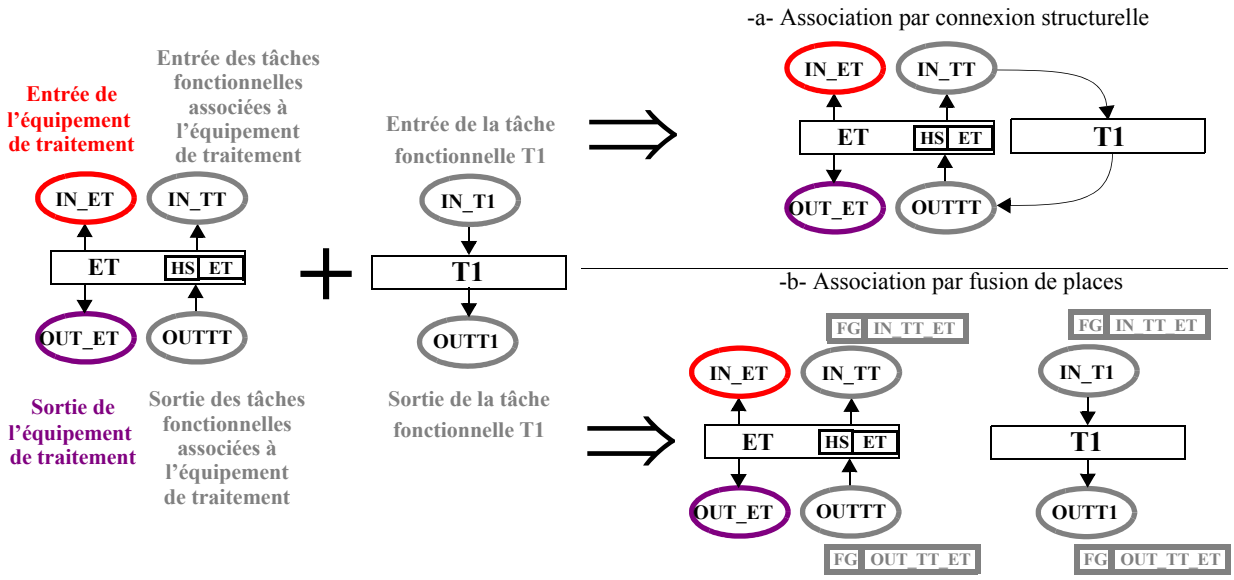


Figure 60 : Association d'une tâche fonctionnelle à un équipement matériel

Cette opération se déroule de la même manière que pour la construction de l'Architecture Matérielle que nous avons décrite dans le paragraphe précédent. On aura donc le choix de réaliser cette connexion de manière structurelle en reliant les différents modèles (fig. 60a) ou en utilisant la possibilité de fusion de places que nous offre le formalisme RdP selon Jensen (fig. 60b).
Dans le cas où plusieurs tâches fonctionnelles seraient associées à un même équipement de traitement, la méthode reste identique. Cependant, lorsque le nombre de connexions est important, la lisibilité de la représentation est réduite. Dans ce cas on privilégiera la fusion de places (fig. 61).

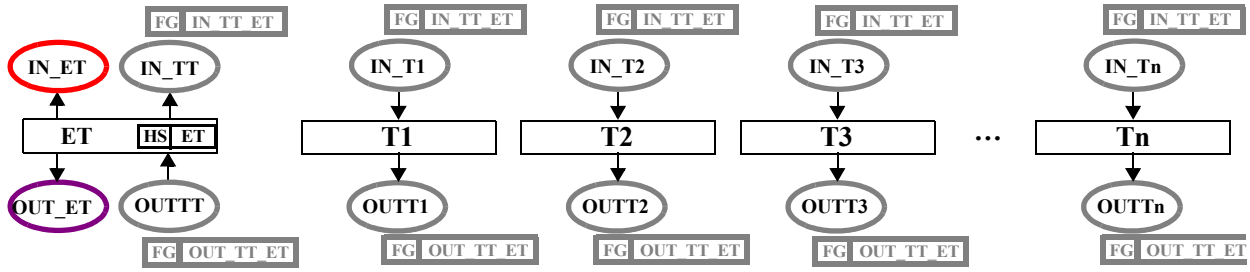


Figure 61 : Association de plusieurs tâches fonctionnelles à un équipement de traitement

3.4.2 Modélisation des flux informationnels

Comme nous l'avons décrit dans le paragraphe 3.2.2.2, p. 69, chaque information transitant dans l'Architecture Fonctionnelle relie deux tâches fonctionnelles ou une tâche fonctionnelle et le milieu extérieur. Étant donné que l'Architecture Opérationnelle définit la projection des tâches fonctionnelles sur des équipements matériels, ces informations doivent pouvoir trouver leur chemin dans l'ensemble des équipements matériels.
Il est donc nécessaire de définir le chemin emprunté par une information pour se rendre d'une tâche à l'autre, c'est-à-dire l'ensemble des équipements tranversés par la dite information. Nous appellerons cette opération «*Routage de l'information*». Un tableau des différentes informations transitant a été réalisé durant la phase de modélisation de l'Architecture Fonctionnelle (Tableau 2, à la page 69) afin de préparer cette phase de définition du routage.

Prenons par exemple le cas de la figure 62. On obtient un tableau avec chaque flux d'information ainsi que ses tenant et aboutissant. On notera que l'on retrouve toutes les tâches (*T1*, *T2* et *T3*) mais également *Ext* qui correspond à l'extérieur du système, à la Partie Opérative ou à une Interface Homme-Machine.

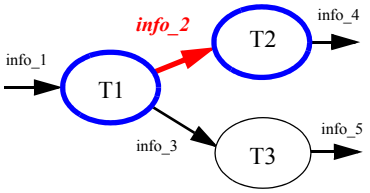


Tableau 3 : Description des Flux informationnels

Nom du Flux	Origine	Destination
info_1	Ext	T1
info_2	***T1***	***T2***
info_3	T1	T3
info_4	T2	Ext
info_5	T3	Ext

Figure 62 : Description des flux informationnel

L'information *info_2* doit être transmise entre *T1* et *T2*. Ces deux tâches étant respectivement associées aux automates *API3* et *API2* (à gauche de la figure 63). La première étape du routage de l'information consiste à déterminer le ou les chemins possibles entre ces deux équipements matériels. Pour cela, un graphe équivalent est construit à partir de l'Architecture Matérielle. Chaque point de ce graphe correspondant à un équipement matériel. La recherche de trajet dans l'architecture Matérielle se traduit donc par la recherche de chemin dans un graphe.
Pour notre exemple, la figure 63 représente le graphe équivalent à l'Architecture Matérielle. Elle comporte des noeuds ● et ■ pour représenter respectivement les équipements matériels hôtes de tâches fonctionnelles et les équipements matériels de communication.

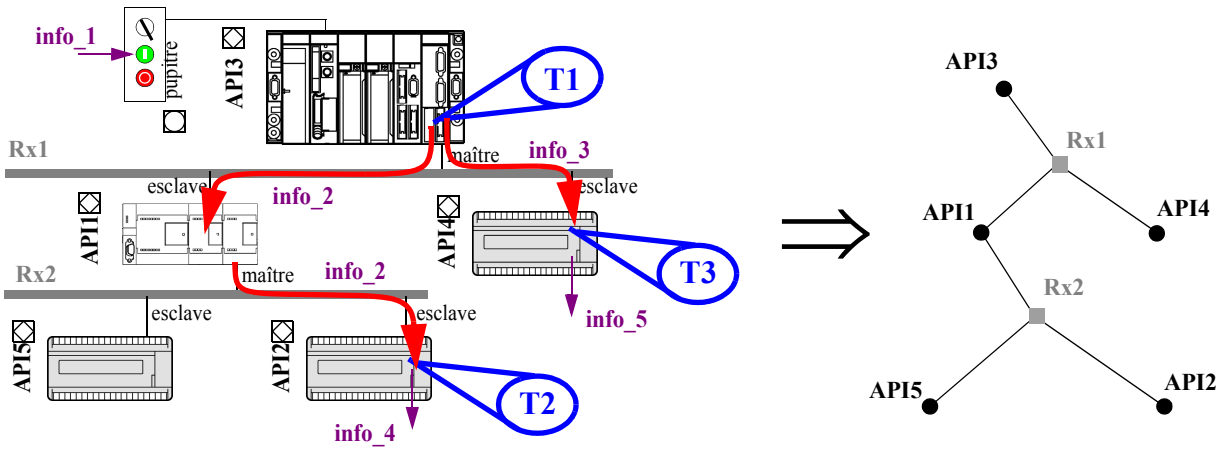


Figure 63 : Exemple de routage d'une information

Lors du parcours du graphe, trois cas peuvent se présenter :

- Aucun chemin n'est possible, dans ce cas l'Architecture Opérationnelle est invalide, cette solution est donc à rejeter,
- Un chemin unique existe, alors tous les équipements traversés par l'information doivent être référencés,

- Plusieurs chemins sont possibles, alors l'architecte doit sélectionner le chemin que l'information suivra. Dans le cas où l'architecte désire avoir une redondance de communication, on considère que chaque chemin sera parcouru par une information différente, on retombe alors sur le second cas.

Dans le cas de la figure 63, le chemin de l'information *info_2* est :

$$API3 \Rightarrow RX1 \Rightarrow API1 \Rightarrow RX2 \Rightarrow API2 \quad (5)$$

On va découper ce chemin en chemins élémentaires. Un chemin élémentaire étant le parcours minimal d'une information entre deux éléments pouvant accueillir des tâches fonctionnelles. On aura alors deux types de chemins élémentaires :

- transmission directe entre deux équipements matériels hôtes ;
- transmission par réseau entre deux équipements matériels hôtes.

Dans le cadre de l'exemple de la figure 63, les chemins élémentaires sont :

- *API3* vers *API1* par le réseau *RX1 ;*
- *API1* vers *API2* par le réseau *RX2.*

Il convient de réaliser cette opération pour chaque flux d'information de l'Architecture Fonctionnelle. On obtient pour notre exemple le tableau de routage matériel suivant :

Tableau 4 : Description des Flux informationnels

Nom du Flux	Elément matériel d'origine	Elément matériel destination
info_1	Ext	API3
info_2	API3	AP1
	API1	API2
info_3	API3	API4
info_4	API2	Ext
info_5	API4	Ext

Une fois ce travail réalisé, il reste à effectuer la modélisation du routage. Nous avons présenté sur la figure 31, la partie des équipements matériels actifs qui s'occupe des sorties. Une portion de ce modèle, appelée «marqueur», définit au travers de la fonction *dest*, l'orientation des informations.
On notera que les informations qui sont à l'intérieur d'un équipement matériel ou d'une tâche fonctionnelle sont uniquement représentées par leur nom alors que ce n'est pas le cas des informations qui transitent à l'extérieur de ces équipements. En effet, il est nécessaire pour le modèle d'avoir la destination associée à l'information, et afin de faciliter le transit, on associe aussi la source de provenance de l'information.
On définit pour cela deux nouvelles «couleurs» : la première *BINT* définit les numéros des équipements actifs, alors que la seconde *VII* correspond à la définition du triplet *(x, so, de)* composé de l'information, de sa source et de sa destination (fig. 64).

On réalise à partir du tableau de routage matériel défini précédemment, la fonction *DEST* qui à partir d'une information *x* et de la référence de l'équipement matériel source *so* calcule sa destination. Toutefois, seuls les modèles hôtes de tâches fonctionnelles seront suivi d'aiguillages, le tableau de routage

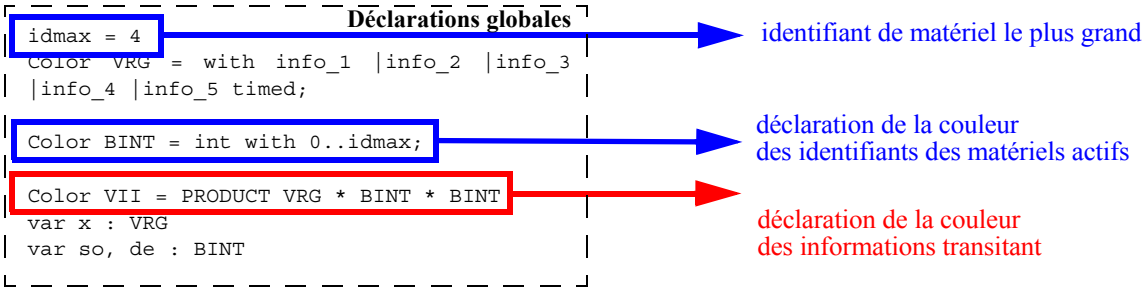


Figure 64 : Extrait des déclarations globales, déclaration des couleurs BINT et VII

sera donc trié en fonction de l'équipement matériel d'origine puis en fonction du flux d'information. le tableau devient donc :

Tableau 5 : Description des Flux informationnels

Elément matériel d'origine	Nom du Flux	Elément matériel destination
Ext	info_1	API3
API1	info_2	API2
API2	info_4	Ext
API3	info_2	AP1
	info_3	API4
API4	info_5	Ext

Les informations venant de l'extérieur auront déjà la forme du triplet *(x, so, de)* comme on le définira dans le paragraphe suivant (modélisation de la Partie Opérative), donc il n'est pas nécessaire de surcharger la fonction *DEST* avec le traitement des informations venant de l'extérieur. On obtient donc dans le cadre de notre exemple, la fonction *DEST* définie par la figure 65.

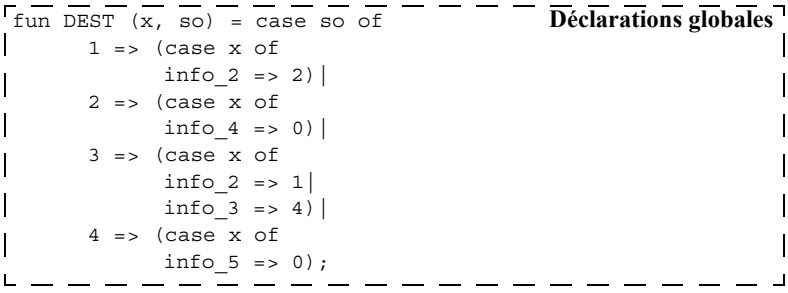


Figure 65 : Définition des routages d'informations dans la page de déclarations globales

A chaque équipement matériel correspond un identifiant (*API1* = 1; *API2* = 2; *API3* = 3; *API4* = 4; *API5* = 5), la zone extérieure a pour identifiant zéro. La fonction *DEST* ne traite que les identifiants de matériel hôtes de tâches fonctionnelles ; ainsi dans notre exemple (comme l'indique la figure 65), l'identifiant 5 n'est pas utilisé.

Maintenant que l'on a défini la fonction *DEST*, il reste à modéliser les «marqueurs» qui se placeront à la suite des modèles d'équipements matériels. Ce modèle reçoit une information *x* et renvoie un triplet

(x, so, de) où *so* est l'identifiant de l'équipement matériel source et *de* l'identifiant de l'équipement matériel destination. La figure 66 représente le modèle d'un marqueur.

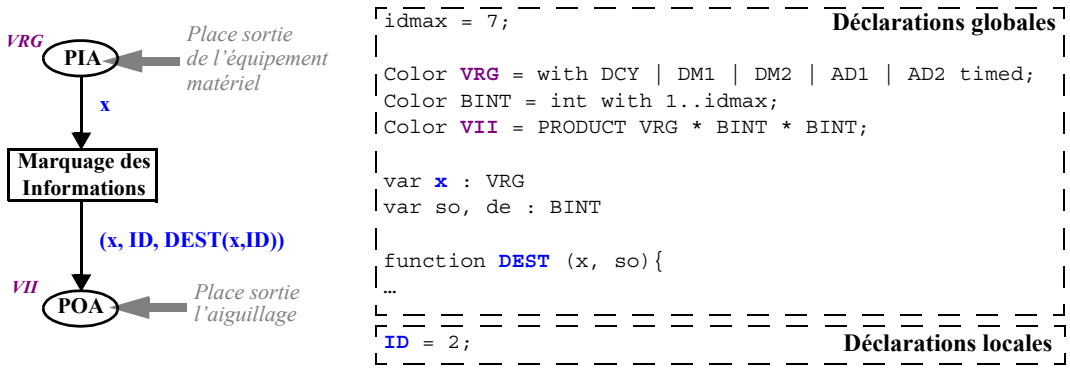


Figure 66 : Modèle réseau de Petri d'un marqueur

Le modèle du «marqueur» est générique, la fonction *DEST* est utilisée pour tous les équipements matériels, seul un équipement doit spécifié à chaque implantation : la valeur de la variable *ID* dans les déclarations locales. Cette variable définit l'identifiant de l'équipement matériel source des informations. Le modèle de «l'aiguilleur» qui suit le «marqueur» (connecté à la place *POA*) est défini par le modèle matériel (fig. 31), il ne sera donc pas exposé dans ce chapitre.

3.5 Préparation de l'Architecture de Simulation

L'architecture de Simulation (définie en paragraphe 2.3.6, p. 57) nécessite d'adjoindre au modèle de l'Architecture Opérationnelle son environnement de simulation ainsi que des observateurs. Dans un premier temps, nous présenterons donc dans ce paragraphe, la modélisation de l'environnement de simulation de l'architecture puis dans un second temps, nous présenterons le processus de modélisation des observateurs.

3.5.1 Modélisation de l'environnement de simulation de l'architecture

Dans ce paragraphe, nous ne préciserons que la modélisation des stimuli, la modélisation comportementale de la Partie Opérative étant volontairement simplifiée, nous ne la développerons pas dans ce paragraphe ; toutefois, un exemple de modélisation d'élément de Partie Opérative est proposé en Annexe C.

Les stimuli de l'architecture de simulation sont en fait des jetons qui permettent la simulation du modèle RdP. Il nous faut donc définir de tels générateurs de jetons.
Le modèle associé est composé d'une transition une place ressource contenant le jeton à générer, d'une place cible et d'une transition temporisée qui consomme un jeton de la place ressource et produit un jeton information dans la place cible ainsi qu'un jeton replacé dans la place ressource.
Le jeton information doit avoir la forme du triplet (*x*, *so*, *de*) avec *x* correspondant à l'information devant être générée, *so* valant zéro car la source est le milieu extérieur, et *de* qui représente le numéro de l'équipement matériel actif cible. Pour notre exemple, le triplet à générer est *(info_1, 0, 1)*. Le modèle du générateur de jetons est donc donné par la figure 67.

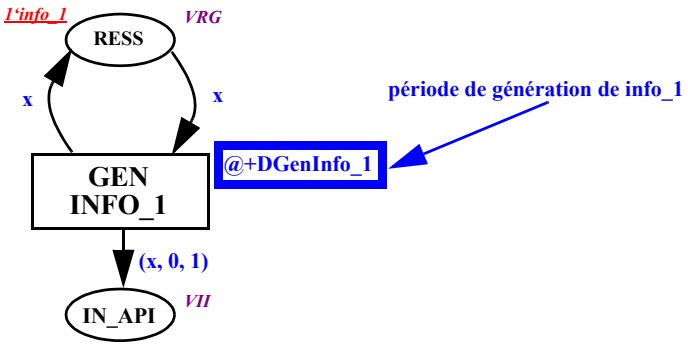


Figure 67 : Forme générale d'un générateur de jetons

La temporisation associée à la transition correspond à la périodicité d'émission des jetons et donc des stimuli. Cette période peut être fixe, ou variable. Dans le premier cas, elle devra être choisie afin ne pas correspondre à un multiple du temps de cycle d'un équipement matériel pour éviter toute corrélation et afin de ne pas fausser les réactions du système. La période est alors définie comme une valeur dans les déclarations globales (fig. 68).

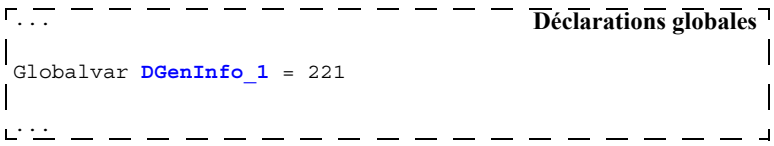


Figure 68 : Déclaration de la période fixe

Dans le second cas, on associe à la transition un code qui permettra de définir la valeur de la période à chaque génération (fig. 69).

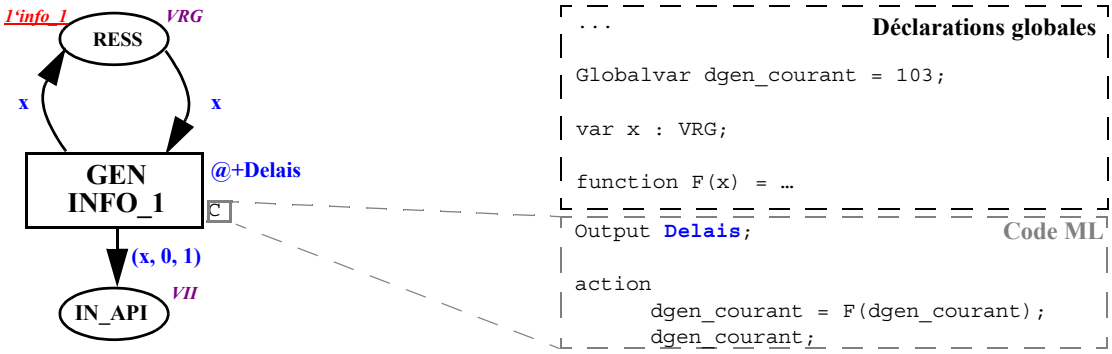


Figure 69 : générateur de jetons à Période non fixe

Le code rend une valeur qui dépend de la fonction *F(x)* et de la valeur courante de la période *dgen_courant*. Cette fonction *F* peut être :

- aléatoire (utilisant la fonction *CPN'randint* de DesignCPN) dans un intervalle donné (fig. 70)
- définie par une suite arithmétique (fig. 71)
- ou définie par l'utilisateur dans les déclarations globales.

```
...                                          Déclarations globales
Globalvar mini = 100;
Globalvar maxi = 150;

function F(x) (
      CPN'randint(mini, maxi);
)
```

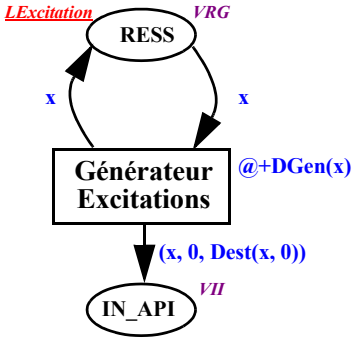
Figure 70 : Déclaration d'une fonction de génération aléatoire bornée

```
...                                          Déclarations globales
Globalvar increment = 7;

function F(valeur) (
      valeur+increment;
)
```

Figure 71 : Déclaration d'une fonction basée sur une suite arithmétique

Dans le cas où l'on désirera avoir de multiples stimuli, on peut réaliser autant de générateurs que nécessaires ou afin d'avoir un modèle compact, on peut opter pour un modèle de générateur de jetons générique qui sera paramétré pour chacun des jetons à émettre. Pour cela, nous devons définir :

- la liste *LExcitations* des jetons à générer ;
- la fonction *DGen* calculant les périodes de génération de chaque jeton.




```
…                                            Déclarations globales
(* Declarations des generateurs d'evenements*)
val LExcitations = 1`I1 ++ 1`I2 ++ 1`I3;

fun DGen(x)= case x of
       I1 => CPN'randint(100000,200000)
      |I2 => CPN'randint( 20000,220000)
      |I3 => CPN'randint( 50000,110000)
      |_ => 0;

fun Dest(x,so) = …
```

Figure 72 : Générateur de jetons générique

En règle générale, on évitera de choisir une période constante car l'ensemble des stimuli pourrait être corrélé par rapport à une des autres durées dans le système, ce qui ne permettrait pas une exploration correcte de l'espace des stimuli.

3.5.2 Modélisations des observateurs

L'évaluation de performances de type «temps de réponse» correspond à évaluer la différence entre la date d'apparition d'un événement t_A et la date de réaction à ce stimulus t_R. Pour cela, il faut soit :

- définir des observateurs de calculs de performances rendant directement les valeurs des performances en sortie du logiciel de simulation ;
- définir des observateurs de trace retranscrivant les dates t_A et t_R en sortie du logiciel de simulation. La performance sera alors calculée à partir de ces données par un programme ou logiciel annexe au logiciel de simulation.

3.5.2.1 Observateurs de performance

Le calcul de performance de type temps de réponse a priori est très simple car il faut soustraire t_A à t_R pour obtenir la durée de transmission D_T. Dans le modèle RdP, les dates t_A et t_R correspondent respectivement aux dates de franchissement de transitions *T1* et *T2* par les jetons *EV* et *RE*.
Afin de prendre en compte les dates t_A et t_R au moment du franchissement de *T1* et *T2*, on associe aux transitions des codes ML (fig. 73) :

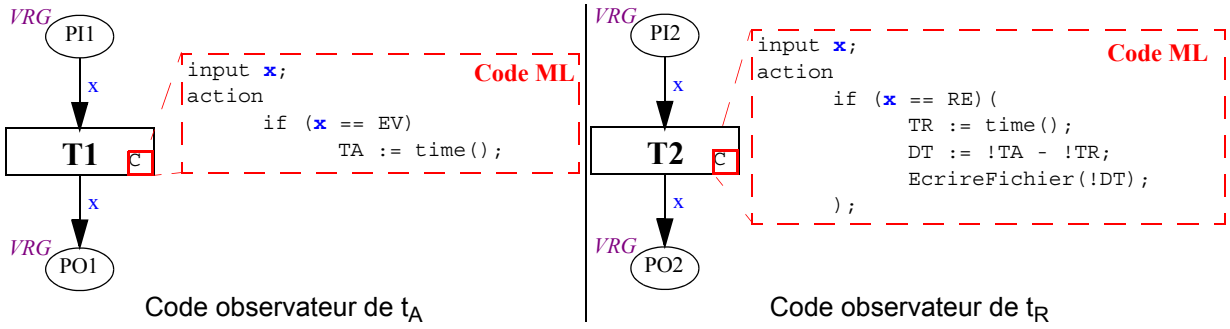


Figure 73 : Codes observateurs de performance

Dans les deux codes, la mémorisation des dates t_A et t_R s'effectue par un stockage de la valeur de la fonction *time()* dans des variables *TA* et *TR*. De plus, la durée de transmission calculée à partir de *TA* et *TR* est stockée dans la variable *DT* puis est enregistrée dans un fichier par la procédure *EcrireFichier()*. Ces variables sont globales afin qu'elles puissent être lues ou écrites à partir de n'importe quel code du modèle de simulation. Leur définition est réalisée dans les déclarations globales (fig. 74):

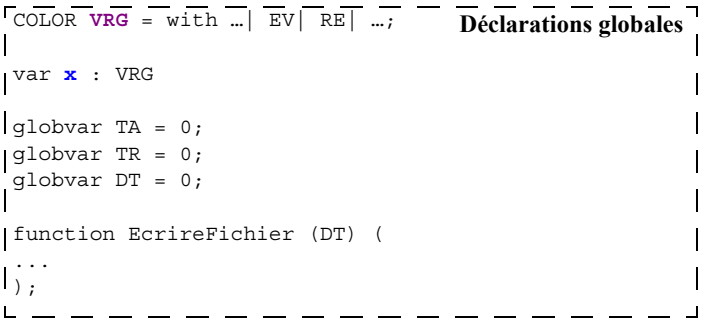


Figure 74 : Extrait des déclarations globales pour les codes observateurs de performance

Toutefois cette méthode d'observation n'est valable que si la période d'apparition des jetons *EV* est supérieure à la durée de transmission *DT*. En effet, si cela n'est pas le cas alors un ou plusieurs jetons *EV* pourraient franchir la transition *T1* avant que la transition *T2* ne soit franchie par le premier jeton *RE* ! La durée DT calculée n'aurait alors aucun sens.
Pour pallier ce problème, on se trouve devant un dilemme : observer tous les événements ou en observer une partie. Dans le premier cas, il est nécessaire d'avoir plusieurs mémoires pour stocker les dates de franchissement des transitions. Dans le second cas, il suffit d'attendre la réaction de l'événement observé pour commencer une nouvelle observation.
Etant donné que la première méthode nécessiterait de déterminer le nombre de mémoires utiles en fonction de la période d'apparition des événements et que celle-ci n'est pas forcément connue; nous avons donc porté notre choix sur la seconde méthode de mesure (fig. 75).

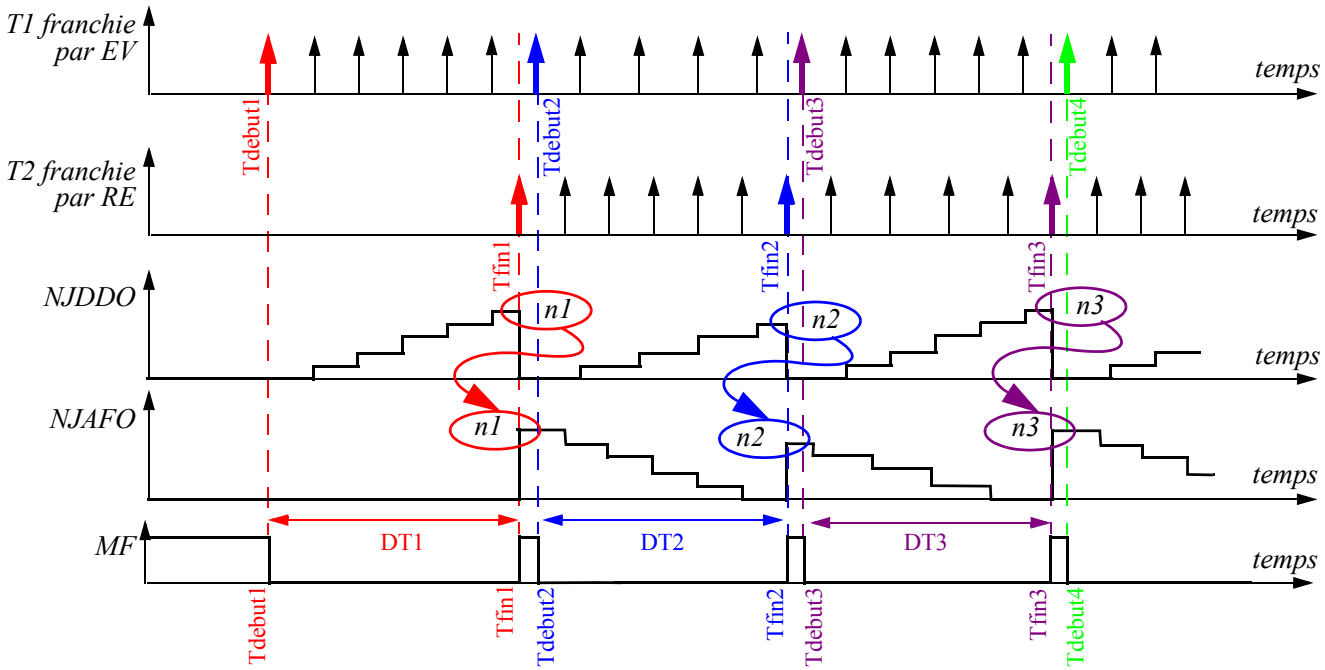


Figure 75 : Méthode d'observation d'une partie des temps de transmission

Il est nécessaire de savoir si l'observation en cours est terminée pour commencer une nouvelle observation. On définit donc une variable booléenne *MF* (Mesure Finie) qui est vraie lorsque l'observation est terminée, MF signale donc qu'une autre observation est possible. Or lorsque l'on commence une nouvelle observation, il peut y avoir *n* jetons qui sont en cours de propagation. Afin de savoir quand l'observation est terminée, il convient de compter ces jetons intermédiaires; on utilise à cette fin la variable *NJDDO* qui compte le Nombre de Jetons apparus Depuis le Début de l'Observation et la variable *NJAFO* qui compte le Nombre de Jetons restants Avant la Fin de l'Observation.

Soit l'opérateur d'affectation noté : «⇒». Une observation commence si l'observation précédente est terminée (si *MF* = 1 alors 0 ⇒ *MF*). Lorsqu'il n'y a plus de jeton avant la fin de l'observation (si *NJAFO* = 0) alors la prochaine réaction correspond à la date T_R; l'observation est donc terminée (1 ⇒ *MF*) et on prend en compte les jetons intermédiaires (*NJDDO* ⇒ *NJAFO* puis 0 ⇒ *NJDDO*). Dans les autres cas, si l'on a un jeton en *T1* alors on incrémente le compteur jetons intermédiaires (*NJDDO* + 1 ⇒ *NJDDO*) et si une réaction arrive en *T2* alors on décrémente le compteur des jetons avant fin d'observation (*NJAFO* -1 ⇒ *NJAFO*). On obtient donc les codes définis sur la figure 77 dont les variables sont définies dans les déclarations globales (fig. 76).

Déclarations globales

```
COLOR VRG = with ...| EV| RE| ...;

var x : VRG

globvar TA = 0;
globvar TR = 0;
globvar DT = 0;
globvar MF = 1;
globvar NJDDO = 0;
globvar NJAFO = 0;

function EcrireFichier (DT) ( ... );
```

Figure 76 : Déclarations globales pour les codes observateurs d'une partie des performances

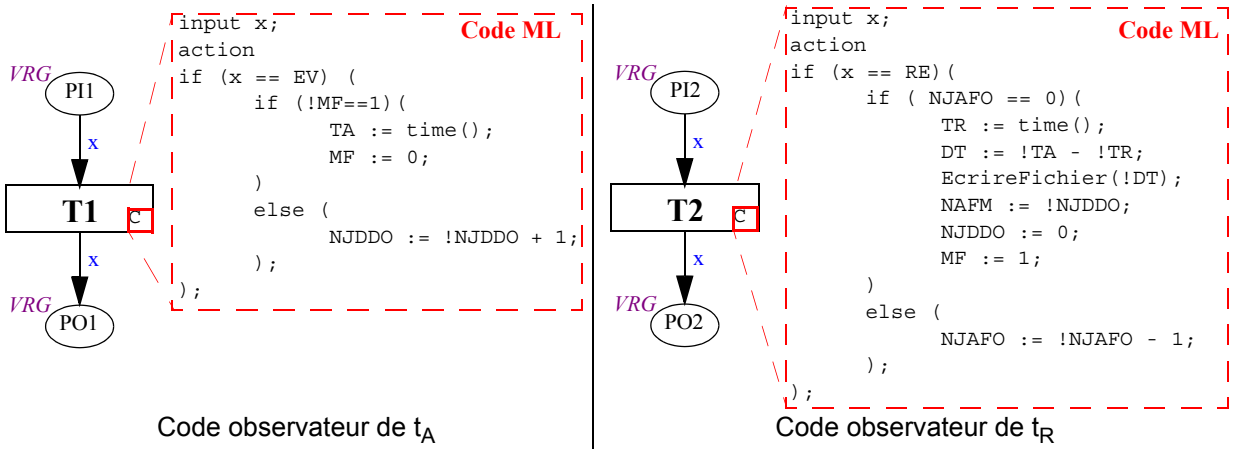


Figure 77 : Codes observateurs d'une partie des performances

3.5.2.2 Observateurs de trace

Dans certains cas, l'observation d'une partie des performances pendant une simulation implique une augmentation de la durée de celle-ci. De plus, lorsque l'on désire observer de nombreuses performances durant une même simulation, la mise en oeuvre d'observateurs de performances peut s'avérer complexe et lourde à gérer car il faut définir pour chacune des performances un ensemble de variables.
Afin de pallier à ce problème, une solution consiste à utiliser des observateurs de traces permettant d'enregistrer l'ensemble des dates t_A et t_R qui surviennent durant la simulation. Le calcul de la performance de type temps de transmission DT devra être réalisé en dehors du logiciel de simulation avec par exemple un logiciel ou un programme de traitement de données.
Les codes d'observations ne contiennent plus le calcul de la performance, mais uniquement la mémorisation et l'écriture des dates dans des fichiers afin de permettre un calcul de performance hors-simulation (fig. 78) :

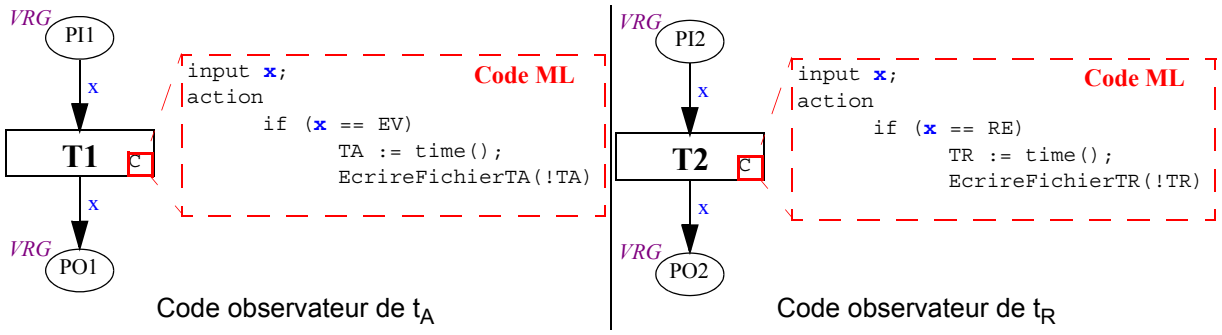


Figure 78 : Codes observateurs de trace

Étant donné que chacune des informations est sauvegardée dans un fichier différent, il est nécessaire d'avoir pour chacune d'entres elles une procédure d'écriture dans un fichier.

Les observateurs de trace permettent d'observer l'ensemble des événements et cela même si plusieurs observations sont réalisées en parallèle ; leur mise en oeuvre est très simple mais elle nécessite un traitement post-simulation afin d'extraire des fichiers de trace les performances à évaluer.

3.5.3 Implantation des éléments de simulation

L'environnement de simulation de l'Architecture Opérationnelle est composé de deux parties : les générateurs de jetons (ou générateur d'événements) et les modèles associés à la Partie Opérative. Ces deux éléments sont externes au contrôle-commande, ils sont donc placés en dehors du reste du modèle. L'ensemble de ces modèles est placé dans un réseaux de Petri qui sera représenté par une transition de substitution connectée par deux places au reste du modèle. Ces deux places *INAPI* et *INEXT* sont respectivement la place d'envoi de messages à l'ensemble des API, et la place d'envoi de messages vers l'extérieur.(fig. 79).

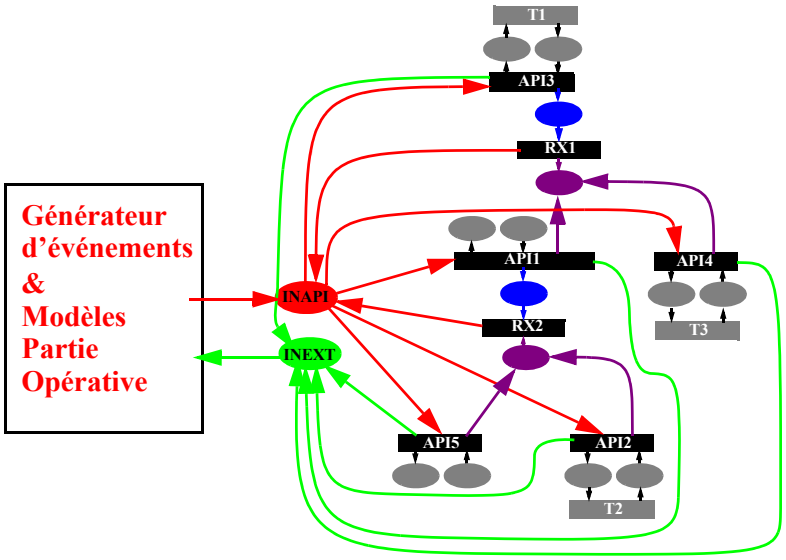


Figure 79 : Implantation des générateur d'événements et des modèles PO

En ce qui concerne les observateurs, étant donné qu'ils doivent toujours être placés au plus près des informations qui devront être surveillées, ils seront toujours associés, par l'intermédiaire des codes ML, à des transitions dans les modèles des équipements matériels.

3.5.4 Impact de l'Architecture de Simulation

Nous allons étudier quelles perturbations peuvent être engendrées par l'ajout des éléments de l'environnement de simulation au modèle de l'Architecture Opérationnelle. En effet, il serait fort dommageable d'avoir modélisé correctement l'Architecture Opérationnelle et que ces ajouts perturbent la simulation ou les performances observées. Ces perturbations peuvent affecter l'évaluation de performance au travers de la simulation elle même ou en polluant les indicateurs de performance mesurées. Nous étudierons dans un premier temps, l'impact de l'introduction des générateurs de jetons et dans un deuxième temps, l'impact des observateurs.

3.5.4.1 Impact des générateur de jetons

Les générateurs sont-ils non-invasifs ?
Les modèles des générateurs étant externes au modèle de l'architecture Opérationnelle, il n'y a pas de modifications de comportement par l'adjonction des générateurs. Toutefois, les générateur de jetons peuvent avoir un effet néfaste si les périodes de génération de jetons sont corrélées aux temps internes

des équipements matériels. Le choix de ces périodes ou de la méthode de stimulation est donc très importante.

Les générateurs ont-ils un impact sur le temps de simulation ?
Les générateurs de jetons introduisent de nombreux franchissements de transition, et donc de jetons, dans le modèle. Le modèle va donc voir son nombre de jetons croître au cours de la simulation, ce qui a pour effet de faire croître le temps de simulation de façon très importante. Cette augmentation du temps de simulation a une allure exponentielle due à l'augmentation de jetons à gérer à chaque pas de l'horloge temporelle de simulation. Cette croissance dure jusqu'à ce que la mémoire vive (RAM) soit pleine. Au delà de cette limite, c'est la mémoire SWAP qui prend le relais, ce qui implique des temps d'accès mémoire beaucoup plus important donc une «explosion» du temps de simulation. Il est de plus nécessaire de contrer l'augmentation du nombre de jetons dans le modèle, car les jetons qui permettent d'observer une réponse après stimulation finissent dans une place et ne sont plus jamais utilisés. Nous avons donc créés des «absorbeurs» de jetons dont l'intérêt est de maintenir un nombre fini de jetons dans les modèles réseaux de Petri. Effectuons une comparaison entre un modèle muni d'un seul générateur de jetons et un modèle qui génère et absorbe les jetons (fig. 80).

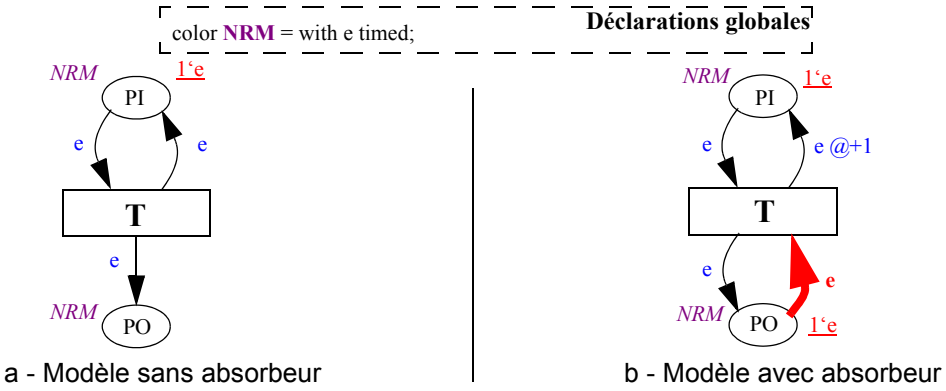


Figure 80 : Modèles pour l'étude de l'impact des absorbeurs

Le nombre de jetons augmente au fur et à mesure que la simulation se déroule pour le modèle figure 80a alors que le nombre de jetons reste constant dans le modèle figure 80b. Les temps de simulation résultants sont représenté sur la figure 81 et montrent la pertinence des absorbeurs.

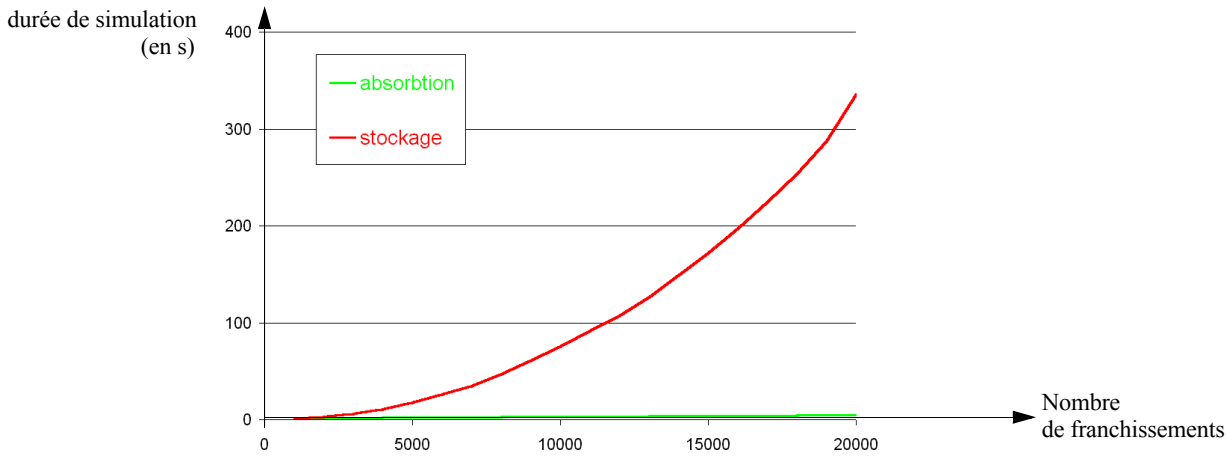


Figure 81 : Temps de simulation avec ou sans absorption de jetons

3.5.4.2 Impact des observateurs

Les observateurs sont-ils non-invasifs ?
Étant donné que les observateurs sont des codes ML associés à des transitions du modèle RdP de l'Architecture Opérationnelle, il n'y a donc pas de modification de la structure RdP du modèle. De plus, nous nous sommes interdit toute action sur la production de jetons lors du tir des transitions dans les codes associés aux observateurs ce qui nous garanti une véritable indépendance entre les observateurs et le comportement modélisé. Il n'y a donc aucune modification du comportement par l'adjonction des générateurs.

Les observateurs ont-ils un impact sur le temps de simulation ?
Il convient d'étudier l'impact des codes ML sur le temps de simulation. Pour ce faire, nous étudions dans un premier temps l'influence des codes de calculs puis du code qui va inscrire dans un fichier la mesure effectuée.

Les codes associés aux transitions permettent de réaliser des calcul. Nous les avons utilisés afin de remplacer des structures places/transitions qui peuvent parfois être très lourdes. Mais quel est leur impact au niveau du temps de simulation ? Pour répondre à cette question, nous avons comparé un modèle sans code avec absorbeur (figure 80b) avec un modèle similaire auquel on associe un code à la transition *T*. Plusieurs opérations ont été utilisées dans ces codes :

- opérations sur entiers : addition, multiplication, modulo; on examinera aussi le temps induit par la répétition d'une même opération (dans notre cas, l'addition);
- opérations sur fonction interne : *time*( ) ; on utilise la fonction *InfInt.toInt*( ) pour convertir le résultat de la fonction *time*( ) en un entier.

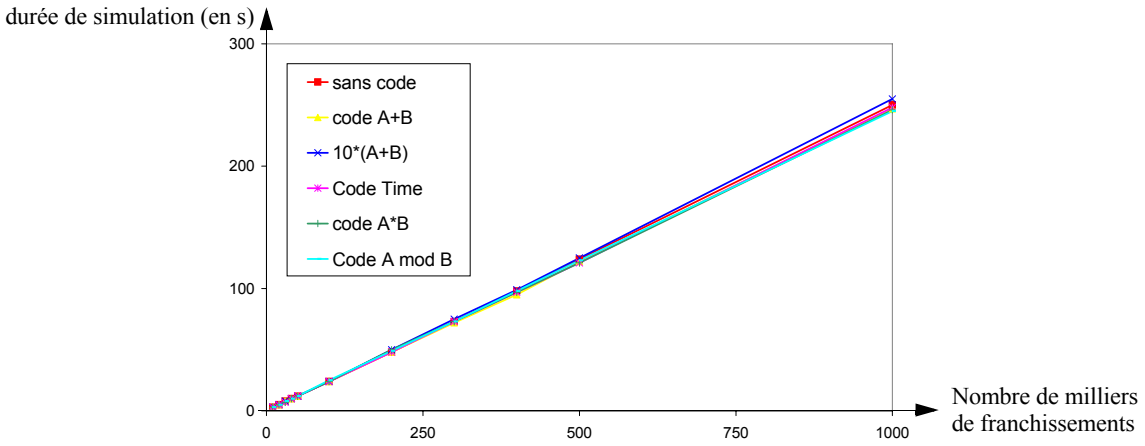


Figure 82 : Temps de simulation pour divers codes de transition

Comme l'indique la figure 82, les durées d'exécution sont quasiment identiques, nous pouvons donc conclure que la durée d'un calcul est négligeable par rapport à la durée de franchissement d'une transition.

Maintenant que nous avons évalué l'impact des codes de calculs au niveau du temps de simulation, il reste à évaluer l'impact du code permettant l'écriture dans un fichier; une fois cette étude effectuée, nous pourrons en conclure quant à l'impact des observateurs sur le temps de simulation. Afin de déterminer cet impact, nous avons simulé un modèle qui, à chaque franchissement de sa transition, exécute le

code qui écrit (ouverture, écriture et fermeture) dans un fichier un entier qui est incrémenté entre chaque franchissement. On obtient les mesures suivantes :

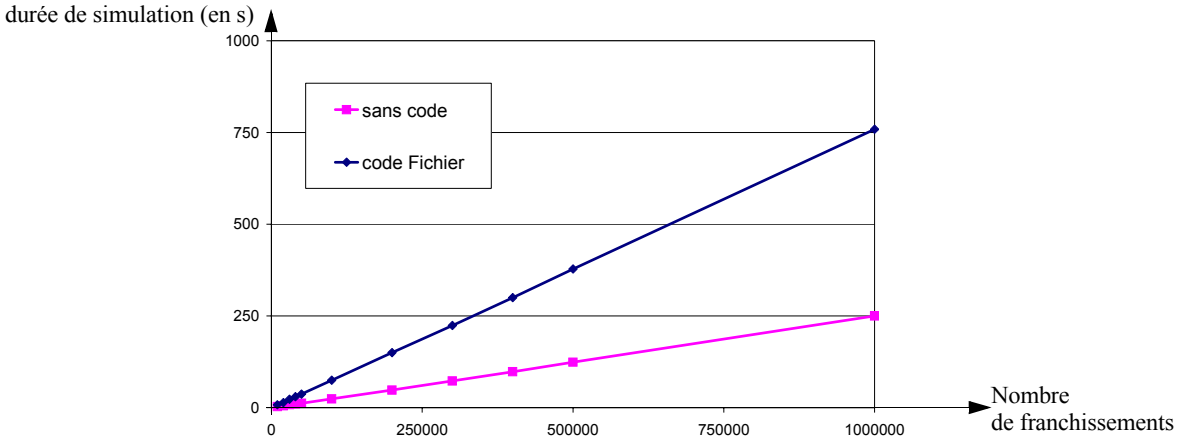


Figure 83 : Temps de simulation avec un code d'écriture dans un fichier

En observant la figure 83, on remarque que le temps nécessaire à l'écriture de données dans un fichier est d'environ deux fois le temps de franchissement d'une transition. Toutefois, étant donné que l'écriture de données n'est réalisée que sur le tirage d'un petit nombre de transitions et qu'en plus ces transitions ne sont franchies que rarement, on peut conclure que l'impact au niveau du temps de simulation est négligeable.

3.6 Conclusion

Dans ce chapitre, nous avons montré que :

- Les tâches composant l'Architecture Fonctionnelle sont modélisées en grande partie de manière générique et nous avons proposé des approches de modélisation pour les tâches ayant un comportement spécifique ;
- Le modèle de l'Architecture Matérielle se construit à l'aide de composants existant que l'on connecte aisément suivant deux méthodes proposées (fusion de places ou connexion structurelle) ;
- Le modèle de l'Architecture Opérationnelle est réalisée par le biais de l'association des tâches fonctionnelles aux équipements de traitement par la même méthodologie que pour la construction de l'Architecture Matérielle et par la fonction de routage de l'information dont la conception est facilitée par la définition des tables de routages ;
- L'Architecture de Simulation est obtenue en complétant le modèle de l'Architecture Opérationnelle par des générateurs de stimuli et des observateurs. Pour chacun de ces modèles, nous proposons plusieurs variantes afin que l'architecte puisse choisir selon ses besoins.

L'ensemble de ces aides permet à l'architecte d'être accompagné durant chacune des phases composant la modélisation du comportement dynamique de son architecture de contrôle-commande en vue de l'évaluation de ses performances. Cependant, l'architecte peut ne pas disposer dans sa bibliothèque de modèle correspondant à l'équipement qu'il souhaite inclure dans son architecture. C'est pourquoi nous décrivons dans le chapitre suivant un ensemble de méthodologies de conception et de validation de modèles d'équipements matériels.

Chapitre 4

Élaboration des modèles des équipements matériels

« Beaucoup de personnes pensent être myopes alors que, en réalité, les choses ne sont pas nettes »

Alexakis, Vassilis

Dans ce chapitre, nous présentons plusieurs démarches de modélisation d'équipements de l'Architecture Matérielle ainsi qu'une étude comparative de la pertinence de ces modèles pour atteindre nos objectifs.
Ainsi, dans un premier temps, nous exposons une formalisation de la méthodologie de conception de modèles matériels en Réseaux de Petri (RdP). Puis, nous mettons en oeuvre cette méthodologie en modélisant un Automate Programmable Industriel.
Enfin, nous effectuons une étude comparative de 6 modèles obtenus par des approches différentes, afin de déterminer lequel d'entre eux est le mieux adapté aux besoins de l'architecte en terme de simulation.

4.1 Introduction

Lors de la réalisation du modèle de l'Architecture Matérielle, l'architecte ne trouvant pas le modèle d'un équipement matériel souhaité doit donc réaliser un travail de modélisation.
Dans un premier temps, il est nécessaire de définir le comportement devant être modélisé soit par une connaissance de l'équipement et soit grâce à une identification expérimentale du comportement.
Dans un deuxième temps, il faut réaliser la modélisation. Pour cela, nous proposons trois approches : une modélisation basée sur le comportement de la structure des RdP, une modélisation basée sur l'interprétation du comportement défini, et une modélisation basée sur un comportement probabiliste.
Afin d'illustrer ce travail de modélisation, nous appliquons les différentes méthodes de modélisation en étudiant un Automate Programmable Industriel (API).

Enfin, étant donné que l'on obtient divers modèles suivant la méthode utilisée, il est nécessaire de valider les modèles les plus pertinents afin de les intégrer dans notre bibliothèque. Pour cela, nous proposons une étude comparative des différents modèles obtenus en étudiant les six modèles d'API obtenus précédemment.

4.2 Formalisation de la méthodologie de modélisation [MEUNIER & al 2000a]

Les modèles de la bibliothèque d'équipements matériels seront utilisés sans que l'architecte n'ait autre chose à faire qu'un paramétrage et un assemblage de modèles génériques. La qualité de l'évaluation de performance des architectures de contrôle-commande repose sur la pertinence de ces modèles d'équipements. Nous allons examiner dans un premier temps les moyens qui nous sont offerts pour modéliser ces équipements.

4.2.1 Quel comportement modéliser ?

Pour déterminer le comportement à modéliser, l'architecte peut se baser sur la connaissance du comportement qu'il a de l'équipement matériel ou en réaliser une identification par un moyen expérimental.
Ainsi, pour définir le comportement d'un équipement matériel par connaissance, il convient de prendre en compte les documentations des constructeurs liées à l'équipement matériel considéré. Toutefois, étant donné que les informations contenues dans les documentations techniques sont en général partielles et elles ne décrivent que le fonctionnement général de l'équipement afin de permettre à l'utilisateur de le mettre en oeuvre. La définition du comportement est donc globale mais assez peu précise.
Alors que pour définir le comportement d'un équipement matériel par identification, il est nécessaire d'identifier le comportement de ce matériel. Pour cela, il est nécessaire d'avoir l'équipement matériel à disposition, ainsi qu'un procédé permettant de solliciter et d'observer cet équipement. L'identification permet d'obtenir un comportement réel de l'équipement matériel. Mais le comportement obtenu est souvent incomplet car les observations ne couvrent en général pas le spectre complet des excitations possibles. On obtient donc un modèle précis mais souvent partiel.

4.2.2 Comment Modéliser ?

Étant donné que nous utilisons des réseaux de Petri Colorés Temporisés selon Kurt Jensen, nous pouvons envisager trois approches de modélisation :

- une approche privilégiant les capacités de modélisation structurelle des réseaux de Petri
- une approche privilégiant les capacités de modélisation par interprétation, en utilisant les possibilités des codes ML associé aux transitions
- une approche privilégiant les capacités de modélisation probabiliste, en utilisant des fonctions statistiques dans des codes ML

Dans les paragraphes suivants, nous allons présenter ces trois approches de modélisation. Nous prendrons dans un premier temps, un exemple académique simple pour expliquer brièvement ces trois approches : le comportement à modéliser est celui d'un marquage de pièces : on associe à une pièce *p* un décimal compris entre 0 et 9. Ce chiffre sera le successeur du chiffre précédemment associé (Remarque : le successeur du chiffre 9 sera 0). On considérera que le nombre de pièces est très important.

4.2.2.1 Approche privilégiant la structure

Dans cette approche, chaque aspect du comportement du système modélisé est transcrit en places et en transitions. Chaque changement d'état du système est représenté par le franchissement de transitions du modèle. Dans l'exemple du marquage de pièces, un jeton pièce *p* est marqué d'une valeur *i*. Cette valeur *i* est incrémentée pour le prochain jeton pièce; elle est remise à zéro s'il y a dépassement de 9. Le modèle obtenu est décrit figure 84.

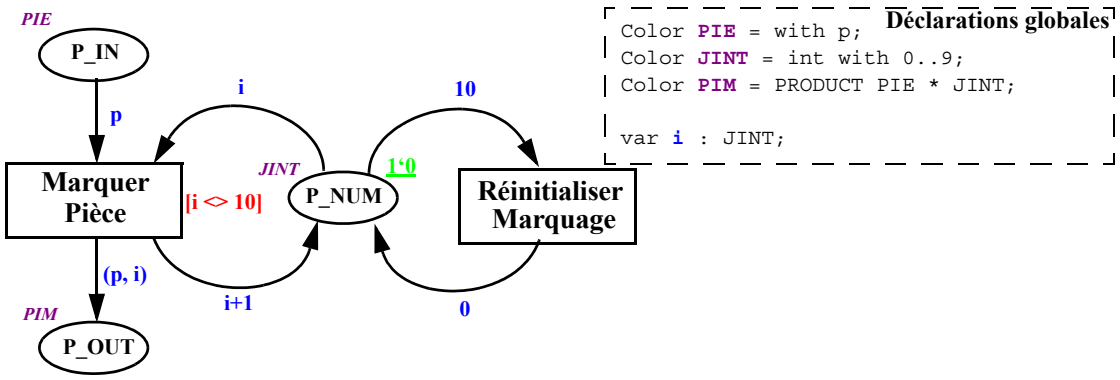


Figure 84 : Modélisation structurelle du marquage de pièce

Ce type de modèle est facile à construire mais durant la simulation on a en général un grand nombre de transitions qui doivent être franchies. Dans cet exemple, pour n pièces, on a $\left(1+\frac{1}{10}\right)\times n$ franchissements de transitions. Ce type de modélisation implique donc en général une simulation longue et coûteuse en temps CPU.

4.2.2.2 Approche privilégiant l'interprétation

Dans cette approche, certains comportements du système sont transcrits en instructions associées à des transitions. Durant la simulation, les codes ML sont exécutés quand leurs transitions associées sont franchies. Pour l'exemple du marquage de pièces, des lignes de codes associés au franchissement de la

transition *Marquer Pièce* permettent la détermination de la valeur *i* associée à la pièce *p*. Cette valeur est déterminée à partir d'une variable globale *VAL*. Le modèle obtenu est décrit par la figure 85.

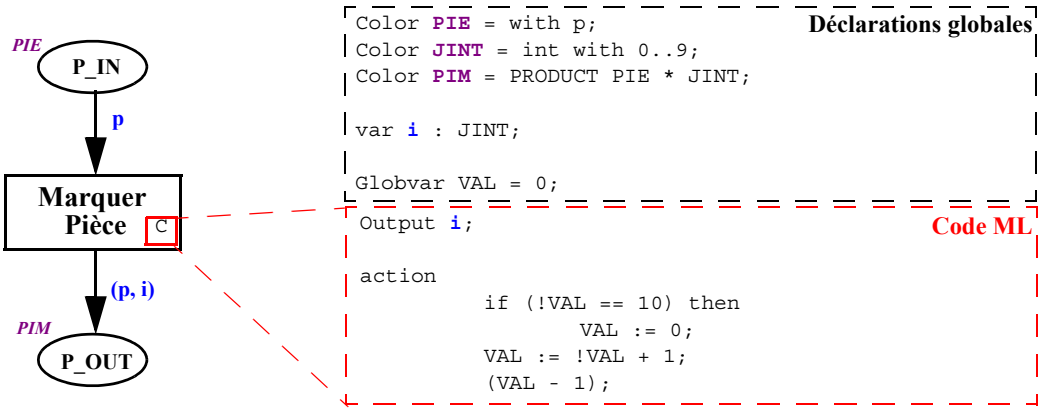


Figure 85 : Modélisation privilégiant l'interprétation de l'exemple du marquage de pièces

Cette approche permet de répartir la complexité de la description du comportement dans des structures plus simples et dans des algorithmes associés aux transitions. Le temps de simulation est réduit : pour n informations, on a n franchissements de transitions (le temps d'exécution des codes ML est négligeable, comme nous l'avons montré sur la figure 82).

4.2.2.3 Approche probabiliste

Dans cette approche, des transitions utilisent des codes ML contenant des fonctions probabilistes qui influent sur les valeurs de temporisation ou le marquage de jetons. Durant la simulation, les valeur courantes des paramètres observés ne seront pas forcément significatives car ce ne sont que les résultats finaux qui ont une valeur significative. Dans le cas de l'exemple de marquage de pièces, la répartition des marquages entre 0 et 9 est uniforme, donc pour déterminer la valeur de *i* associée à une pièce *p*, on ne cherche pas à déterminer la valeur *i* de chaque pièce *p*, mais on réalise un tirage aléatoire (avec une répartition uniforme) entre 0 et 9 avec la fonction *RAND(min, max)*. Le modèle obtenu est décrit par la figure 86.

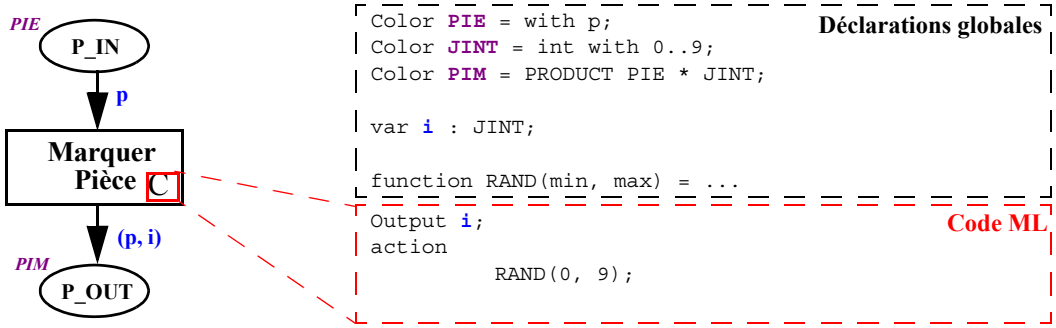


Figure 86 : Modélisation probabiliste de l'exemple du marquage de pièces

Cette approche permet d'avoir des résultats globalement équivalents à ceux des deux approches précédentes tout en ayant une structure de réseau de Petri très simple et un code associé très simple. Par contre, étant donné que l'approche est probabiliste, il ne sera pas possible d'avoir une étude individuelle de chaque marquage de pièce.

Avec deux approches possibles pour la modélisation du comportement (connaissance et identification) et trois manières de traduire ces comportements en réseau de Petri (structure privilégiée, interprétation privilégiée, aspect probabiliste), on peut imaginer pouvoir construire un RdP de six manières différentes.
Nous allons maintenant explorer ces six approches en les appliquant à la construction du modèle de comportement d'un même équipement de commande : un Automate Programmable Industriel (API).

4.3 Construction du modèle de comportement d'un API

4.3.1 Description de l'équipement

Un Automate Programmable Industriel (API) est un équipement matériel qui est qualifié d'actif car il peut se voir affecté un ou plusieurs atomes fonctionnels. Nous avons choisi d'appliquer nos six approches sur ce type de support car c'est un équipement très courant dans les architectures de contrôle-commande. Nous poserons l'hypothèse que l'API étudié sera mono-processeur sans tâches préemptives. Sa structure est composée d'un micro-processeur qui exécute le programme implanté et d'un ensemble de cartes qui gèrent les entrées, les sorties ou les accès réseaux (fig. 87).

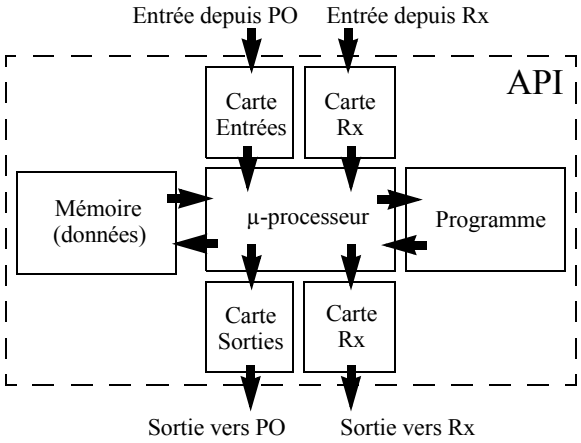


Figure 87 : Structure d'un Automate Programmable Industriel

Le comportement du micro-processeur (CPU) d'un API se décompose en trois phases :

- une phase de mise en mémoire des données disponibles sur les cartes d'entrées ou réseau, on appelera par la suite cette phase *Lecture des entrées* (*L*),
- d'une phase de traitement du programme utilisant les valeurs des données en mémoire et dont les résultats seront stockés en mémoire, on appelera par la suite cette phase *EXecution des programmes* (*EX*),
- d'une phase d'émission des données à emettre issues de la mémoire vers les cartes des sortie ou carte réseau, on appelera par la suite cette phase *emission des Sorties* (*S*).

Toutefois, bien que l'on retrouve couramment ces trois phases dans le comportement du CPU, la relation entre ces phases peut varier d'un API à l'autre et de plus certains d'entre eux ont la possibilité de modifier leur comportement en agissant sur certains réglages ou paramètres. Ainsi, les comportements les plus couramment observés sont cyclique (fig. 88a) ou périodique avec ou sans synchronisation de la lecture des entrées et de l'affectation des sorties (fig. 88b et fig. 88c).

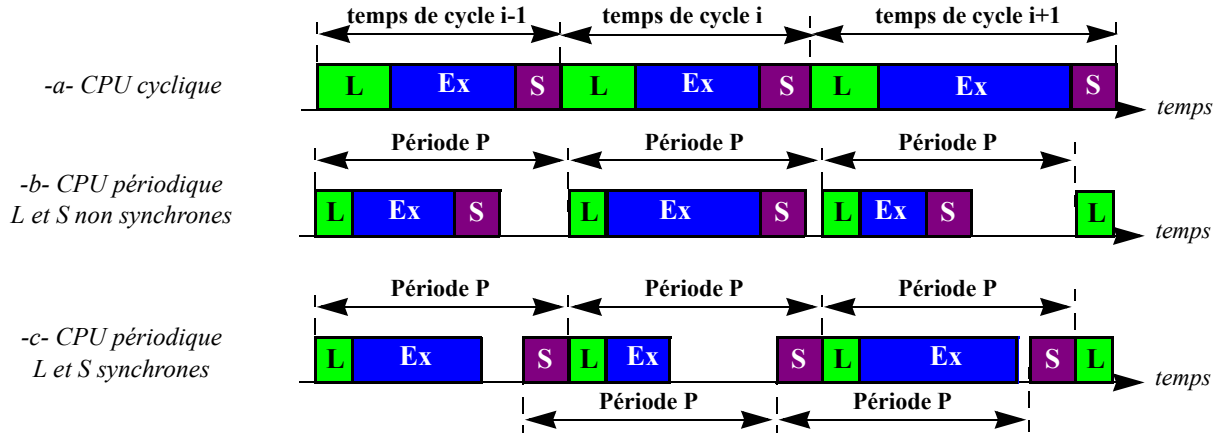


Figure 88 : Exemple de comportement cyclique de l'activité CPU.

Ainsi il serait nécessaire d'avoir un modèle particulier pour chaque API. Cela impliquerait d'avoir des connaissances expertes sur le comportement de chacun d'entre eux. Heureusement, bien qu'ils soient tous différents, leurs comportements sont globalement semblables et ils se conforment aux trois classes de comportements énoncés précédement. Étant donné que les modèles de deux API auront globlalement un comportement très proche à quelques nuances près, les méthodes présentées s'appliqueront tout aussi bien à la modélisation d'un autre équipement ayant un comportement différent de celui que nous allons étudier.
Donc nous choisissons pour illustrer nos démarches de modélisation de les appliquer à une catégorie d'API :

- le comportement du micro-processeur (*CPU*) est périodique avec L et S synchrones (fig. 88c), non préemptible (sans interruption);
- le programme exécuté sera identique à chaque cycle de l'API ;
- les comportements des cartes de lecture des entrées (*LE*) et d'émission des sorties (*ES*) seront considérées comme des retards purs (respectivement D_{LE} et D_{ES}) ;
- les cartes réseaux n'auront aucun effet sur le comportement de l'API, les retards associés seront gérés par le modèle de comportement du réseau.

Toutefois, étant donné que c'est la modélisation du comportement du CPU dans un API qui est la plus problématique, nous ne nous intéresserons pas à la gestion des cartes de sorties et réseaux ainsi que l'aiguillage des informations vers les équipements destinataires. La modélisation de cette partie de l'équipement API que nous avons appelé «aiguilleur» sera décrite en Annexe B.
L'objet de notre étude est donc sur la modélisation du comportement du CPU vis-à-vis des entrées, des sorties et des programmes qui lui sont associés.

4.3.2 Description du comportement basé sur la connaissance de l'équipement

En cours de fonctionnement, le nombre d'instructions exécutées varient selon chaque cycle de fonctionnement. Toutefois, pour les CPU périodiques on peut considérer que la durée du cycle est constante. Cette valeur peut être obtenue :

- en estimant le maximum du temps d'exécution du programme ;
- en réalisant le calcul à partir de la vitesse d'exécution de chaque instruction (données constructeur).

Puisque que l'on a considéré que le programme exécuté à chaque cycle est de taille constante, on aura donc une durée D_{EX} constante. De plus, nous considérons que les phases de lecture ou d'écriture des données ont des durées constantes (respectivement D_L et D_S), ce qui revient à considérer que l'API aura des cycles dont les durées (D_{cycle}) seront constantes (fig. 89).

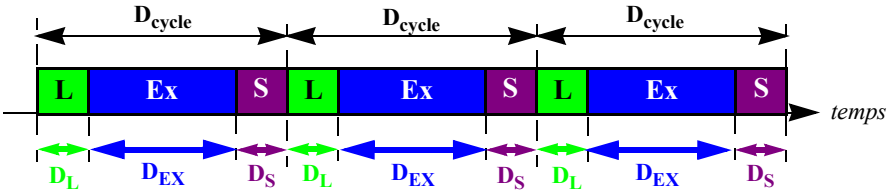


Figure 89 : Activité CPU d'un API cyclique périodique d'après la documentation constructeur.

4.3.3 Application des méthodes de modélisation basée sur la connaissance

Dans ce paragraphe, nous appliquerons les trois approches de modélisation au comportement décrit au paragraphe précédent

4.3.3.1 Modélisation structurelle basée sur la connaissance du comportement

Le modèle est construit d'après deux points de vue :

- le point de vue interne, correspondant au concept de cycle *CPU* (Lecture, Exécution des programmes, Ecriture) ;
- le point de vue données, correspondant aux états des données au cours de leurs transmission à travers l'API (à l'entrée du *CPU*, en attente de Lecture, en attente de Traitement, données traitées, en attente d'Ecriture, à la sortie de *CPU*).

Le modèle obtenu, décrit figure 90, se divise en trois parties :

- la partie *a* pour le point de vue matériel avec le cycle *CPU* de l'API ;
- la partie *c* pour le point de vue données avec les états des données dans l'API ;
- et la partie *b* pour les éléments nécessaires à la synchronisation des deux points de vue précédents.

La partie *a* (encadrée à gauche en rouge sur la figure 90) est la forme structurelle du cycle *CPU*, elle est obtenue simplement en recréant le cycle par trois transitions correspondant aux trois tâches du cycle (Lecture, Exécution, Ecriture).

La partie *c* (encadrée à droite en bleu sur la figure 90) correspond aux différentes opérations qui peuvent affecter une donnée, à chaque phase correspond une place. Il n'y a pas de transitions entre les places *attente traitement* et *données traitées*, car cela correspond au traitement effectué sur les données; ce qui n'est pas décrit dans l'équipement matériel mais dans la tâche fonctionnelle qui sera associée à cet équipement matériel.

La partie *b* (encadrée en vert au centre de la figure 90) permet de synchroniser les parties *a* et *c*. En effet, la phase d'*Exécution* commence uniquement lorsqu'il n'y a plus de données en *Attente Lecture*; et de même, la phase de *Lecture* ne commence que lorsqu'il n'y a plus de données en *Attente Ecriture*. Etant donné que le formalisme réseau de Petri que nous utilisons n'accepte pas les arcs inhibiteurs, nous

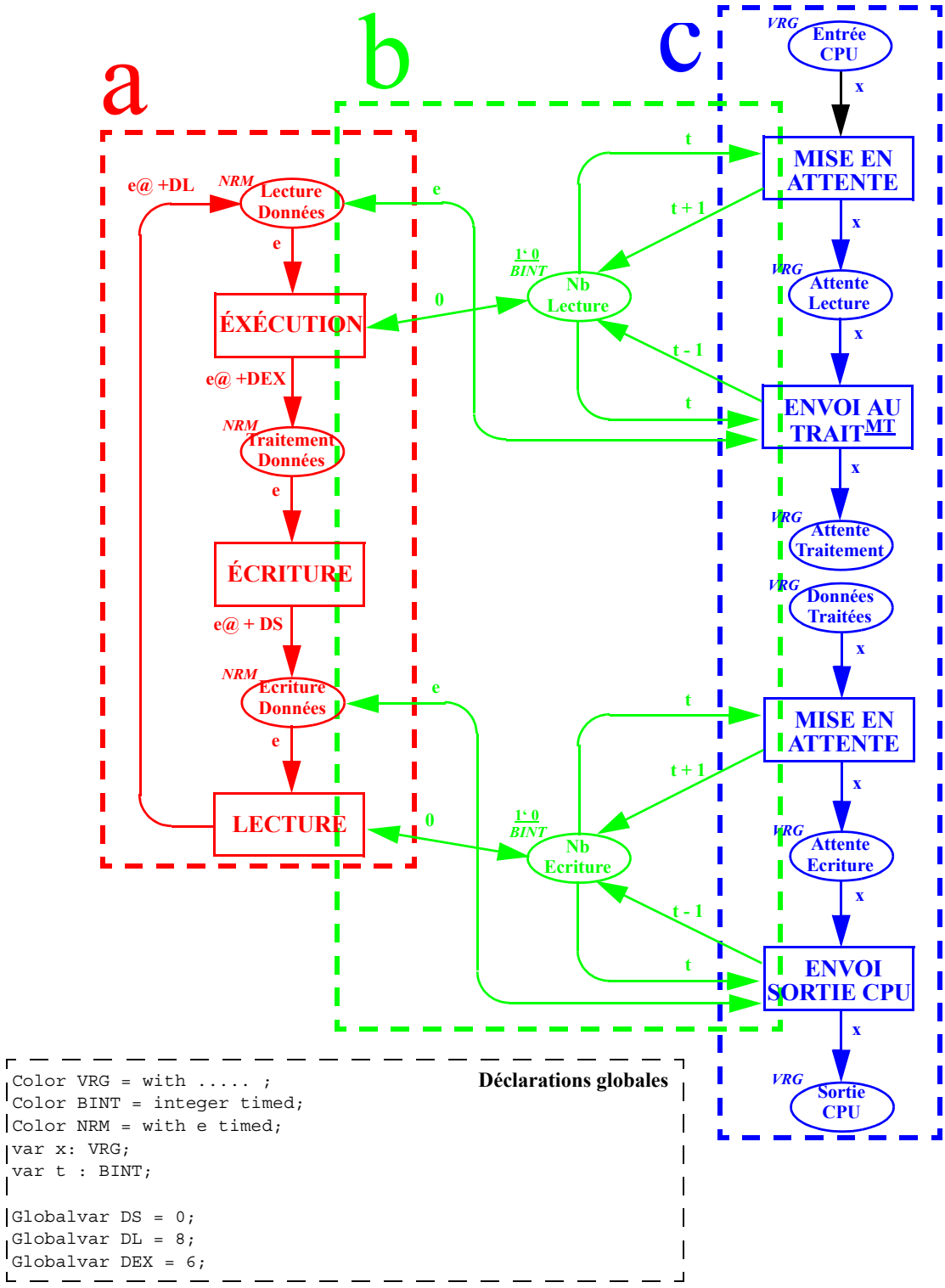


Figure 90 : Modèle RdPCT structurel d'après une synthèse d'informations

sommes obligés de compter les jetons qui se trouvent dans les places *Attente Lecture* et *Attente Ecriture*. Chaque compteur est modélisé par une place qui contient un jeton de type entier dont la valeur correspond aux nombres de jetons dans les places sus-nommées.

Le modèle obtenu est composé de 12 places et de 8 transitions. Pour *n* messages entrants, le nombre de transitions franchies est de $4 \times n + \left(\frac{\text{temps simulé}}{\text{temps de cycle}}\right) \times 3$.

4.3.3.2 Modélisation par interprétation basée sur la connaissance du comportement

La figure 91 illustre le comportement dynamique définit dans le paragraphe 4.3.2 à la page 95. Nous pouvons remarquer que le comportement global est un simple retard entre l'arrivée d'un événement et de sa conséquence en sortie de l'API. Le modèle par interprétation va calculer ce retard, dans un code associée à une transition, en fonction de la date d'arrivée dans l'API.

On défini les données suivantes :

- D_L, la durée de la phase de Lecture ;
- D_{EX}, la durée de la phase d'ÉXécution des programmes ;
- D_S, la durée de la phase d'écriture des Sorties ;
- D_{LE} et D_{ES}, les valeurs des retards des modules de Lecture des Entrées et d'Écriture des Sorties.

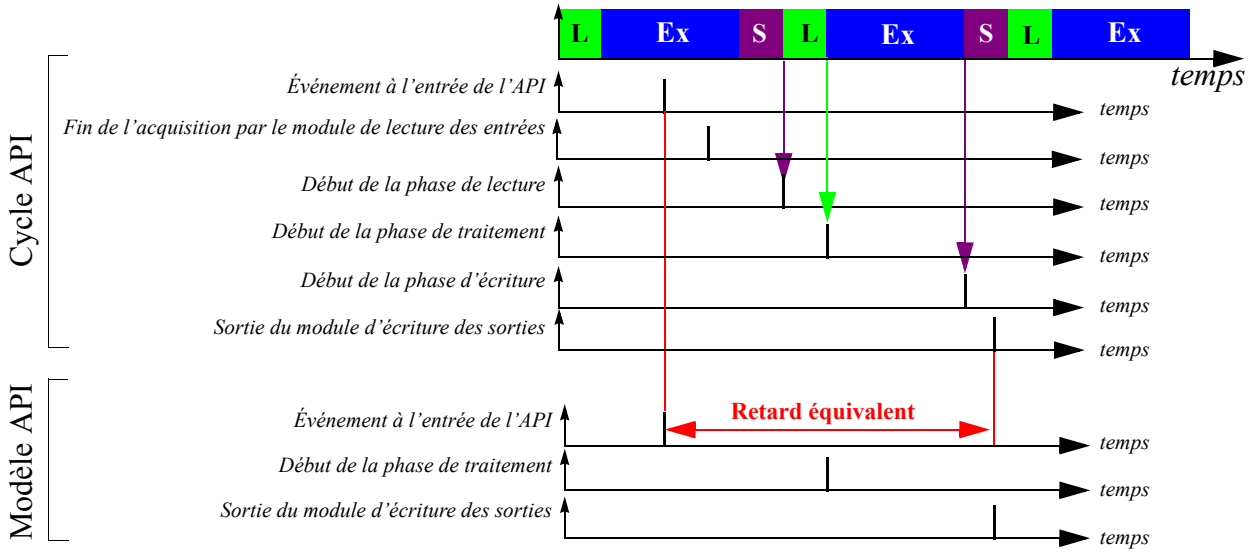


Figure 91 : Activité détaillée du CPU (d'après les documents constructeurs)

Donc, quelque soit l'événement arrivé à la date t, nous calculons sa position relative dans le cycle *CPU* (notée $P_{CY}(t)$) après acquisition par le *Module de Lecture des Entrées* (décalage dans le temps de D_{LE}). On définit cette position par :

$$P_{CY}(t) = (t + D_{LE}) Mod\, D_{CY} \text{ avec } D_{CY} = D_L + D_{EX} + D_E \text{ et où } Mod \text{ est l'opérateur modulo} \quad (6)$$

En observant précisément l'activité *CPU* entre l'acquisition par le *Module de Lecture des Entrées*, et le traitement par le *Module d'Ecriture des Sorties*, on peut distinguer deux cas :

- Le cycle *CPU* se situe dans la phase de *Lecture des Entrées* (fig. 92). Donc, il faut attendre la prochaine phase d'*Ecriture des données* qui est après la fin de la phase de *Lecture des entrées* puis après la Phase d'*Exécution des programmes*. Le retard est donc :

$$\text{Si } P_{CY}(t) \le D_L \Rightarrow retard(t) = D_{LE} + (D_L - P_{CY}(t)) - D_{EX} + D_{LE} + D_{ES} \quad (7)$$

- Le cycle *CPU* se situe hors de la phase de *Lecture des Entrées* (fig. 93). Donc, il faut attendre la fin du cycle *CPU*, puis la fin de la phase de *Lecture des Entrée*, puis la phase d'Exécution *des programmes*. Le retard est donc :

$$\text{Si } P_{CY}(t) > D_L \Rightarrow retard(t) = D_{LE} + (D_{CY} - P_{CY}(t)) + D_L + D_{EX} + D_{ES} \quad (8)$$

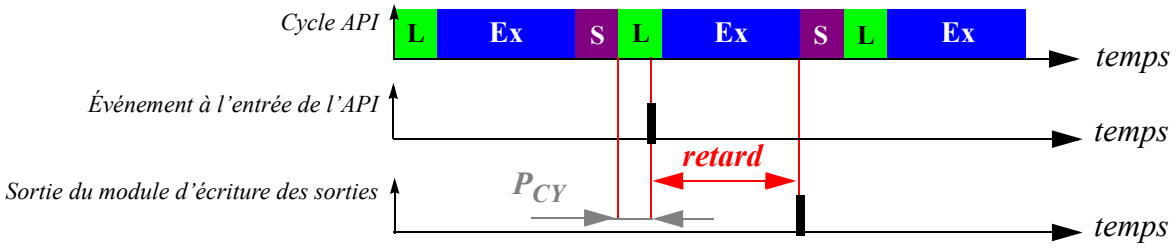


Figure 92 : Premier cas, arrivée d'un événement pendant la phase de lecture

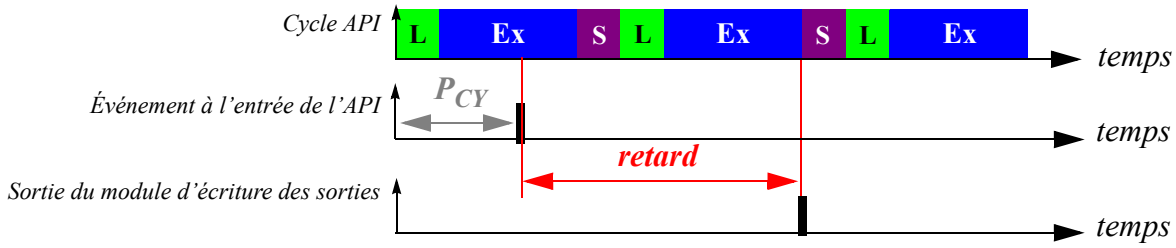


Figure 93 : Second cas, arrivée d'un événement hors de la phase de lecture

La figure 94 représente le modèle réseaux de Petri par interprétation du comportement d'un API. Les déclarations dans la zone de code associée à la transition *RETARD_API* correspondent au calcul de la fonction retard(t).

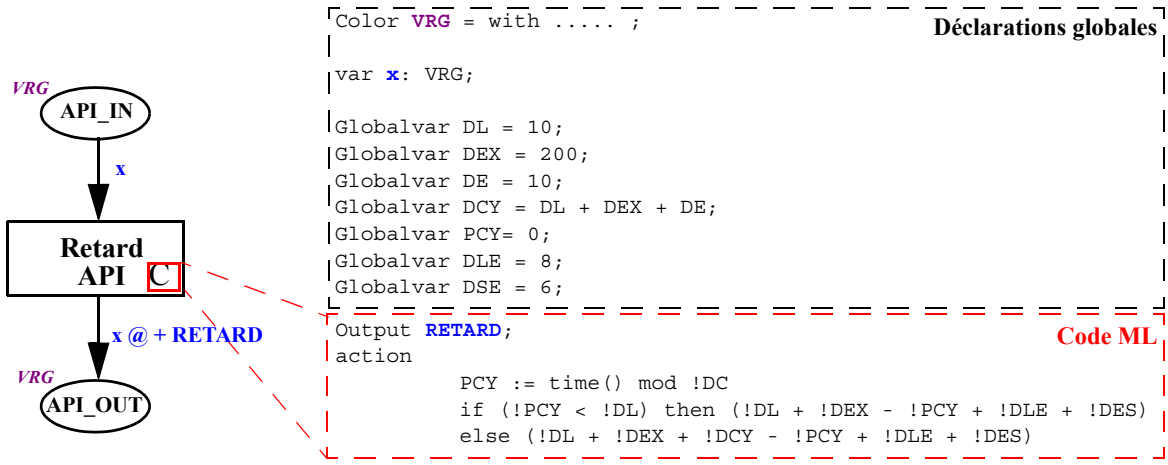


Figure 94 : Modèle RdPCT interprété basé sur la connaissance du comportement

Le modèle obtenu est composé de 2 places et 1 transition. Pour *n* messages entrants, le nombre de transitions franchies est *n*.

4.3.3.3 Modélisation probabiliste basée sur la connaissance du comportement

Comme nous l'avons expliqué précédemment, le comportement global d'un API est un retard (fig. 91). En observant les équations précédemment définies, on peut identifier les limites que peut prendre ce retard pour chacun des deux cas. Pour l'équation (7) :

$$P_{CY}(t) \in [0, D_L] \Rightarrow Retard \in [D_{LE} + D_{EX} + D_{ES}\,,\; D_{LE} + D_L + D_{EX} + D_{ES}] \qquad (9)$$

Pour l'équation (8):

$$P_{CY}(t) \in (D_L, D_{CY}) \Rightarrow Retard \in [D_{LE} + D_L + D_{EX} + D_{ES}\,,\, D_{LE} + D_{CY} + D_{EX} + D_{ES}] \quad (10)$$

donc les bornes du retard sont :

$$min = D_{LE} + D_{EX} + D_{ES} \quad (11)$$

$$max = D_{LE} + D_{CY} + D_{EX} + D_{ES} \quad (12)$$

On suppose que les événements ne sont pas synchrones avec le cycle de l'API ; ainsi on considère que le retard aura une valeur comprise entre la valeur *min* et la valeur *max* qui pourra être prise en respectant une loi de répartition uniforme. La figure 95 représente le modèle probabiliste du comportement dynamique d'un API. La fonction *Rint(Min, Max)* rend un nombre entier aléatoire entre *Min* et *Max*.

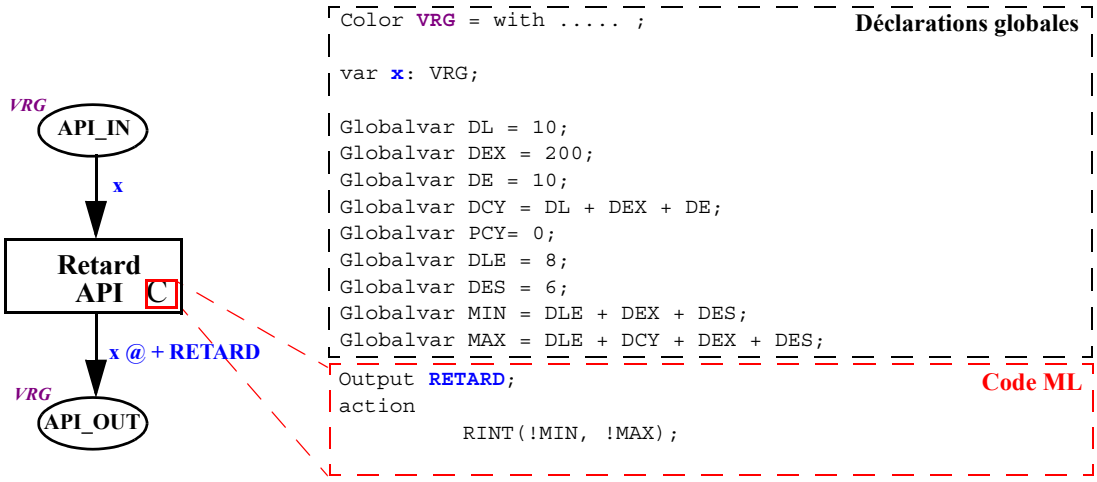


Figure 95 : Modèle RdPCT probabiliste d'après connaissance du comportement

Ce modèle comprend 2 places et 1 transition. Pour *n* messages entrant, le nombre de transitions franchies est *n*.

4.3.4 Description du comportement basé sur l'identification

l'approche basée sur l'identification consiste à construire un modèle du comportement dynamique d'un équipement par observation d'un équipement réel. Les entrées de cet équipement sont sollicitées par des stimuli bien choisis, et grâce à l'observation de ses sorties on établit une description du comportement dynamique de l'équipement.
Pour étudier les signaux logiques d'un équipement industriel, nous avons utilisé la plate-forme expérimentale PRISME (fig. 96) (développé et brevetée au LURPA [PRISME 2001]), composée de générateurs de signaux et d'un analyseur logique commandé depuis une station de travail via un réseau GPIB. Ainsi, il nous est possible d'effectuer un très grand nombre de mesure avec une grandes variété des types de sollicitations.

Dans le cadre de notre exemple, nous désirons mesurer le temps de cycle de l'automate programmable industriel et non pas son temps de réponse. Pour se faire, le choix du programme de test est important. En effet, si on place dans l'automate un programme *Ladder Diagram* (LD) [ISO/IEC 1131-3] comme

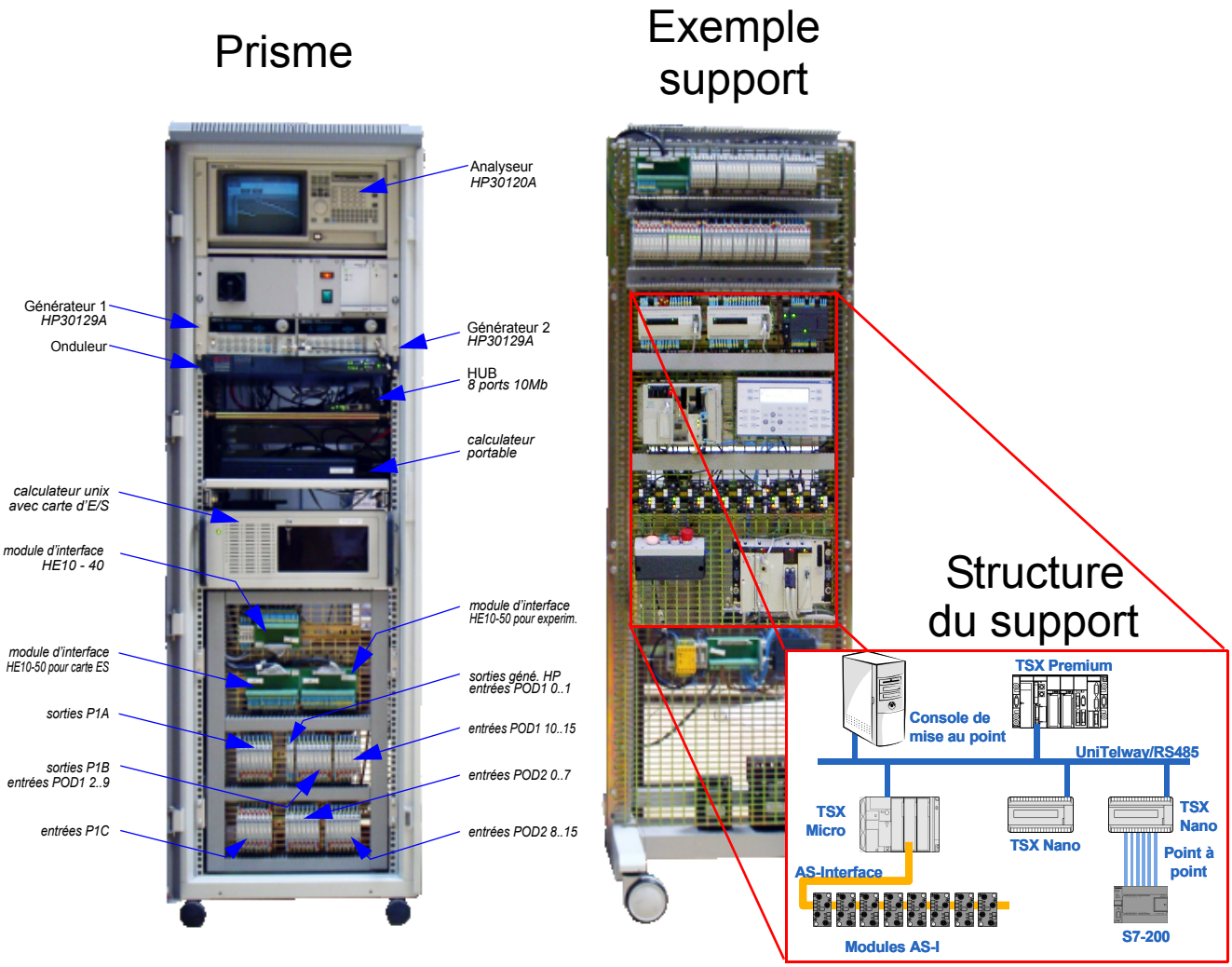


Figure 96 : Plate-forme d'expérimentation PRISME [HEVIN 1997]

indiqué par la figure 97, le résultat sera que pour un événement *a* donné, une réaction *S* aura lieu après un retard *D*.

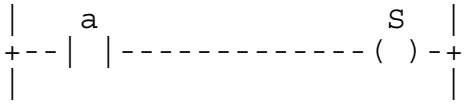


Figure 97 : Programme LD pour observer la réaction d'un API

Ce retard prendra donc en compte le temps de traversée :

- du module de Lecture des Entrées ;
- du module *CPU ;*
- du module d'Emission des Sorties.

On n'aura alors pas de moyen de différencier les retards associés à chacun de ces modules et en particulier le retard D_{CPU} que l'on souhaite identifier (fig. 98).

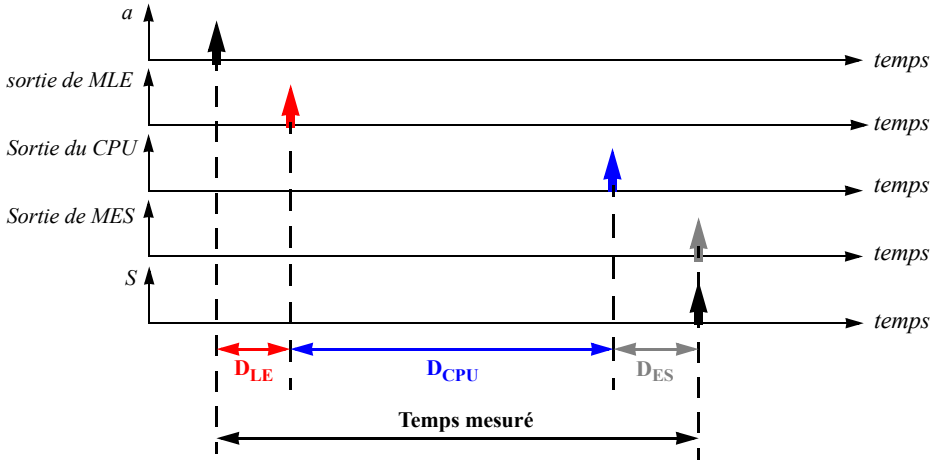


Figure 98 : Comportement du programme de la figure 97

Un second programme LD (fig. 99), permet de résoudre ce problème. Suite à l'occurrence d'un événement d'entrée *a*, le programme va générer la mise à un d'une sortie *S1* et une deuxième sortie *S2* sera émise au cycle suivant.

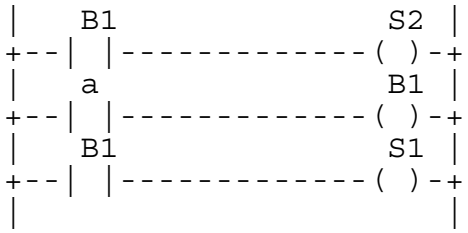


Figure 99 : Programme LD pour observer le temps de cycle d'un API

Le comportement de ce programme peut être traduit par le chronogramme figure 100.

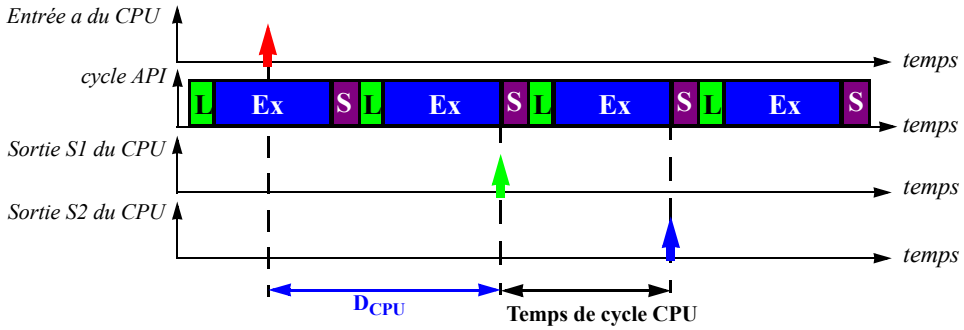


Figure 100 : Comportement du programme de la figure 99

Les mesures ont été effectuées avec ce programme sur un API TSX 17-20 de marque Schneider Electrique. La plateforme PRISME a permis de récolter 12000 valeurs que l'on a représentées sous la forme d'un histogramme (fig. 101).

Les travaux de Mikaël LAVANDIER [LAVANDIER 1998] portant sur l'observation et l'analyse de temps de cycle API nous permettent de conclure que l'activité correspondant à l'ÉXécution du pro-

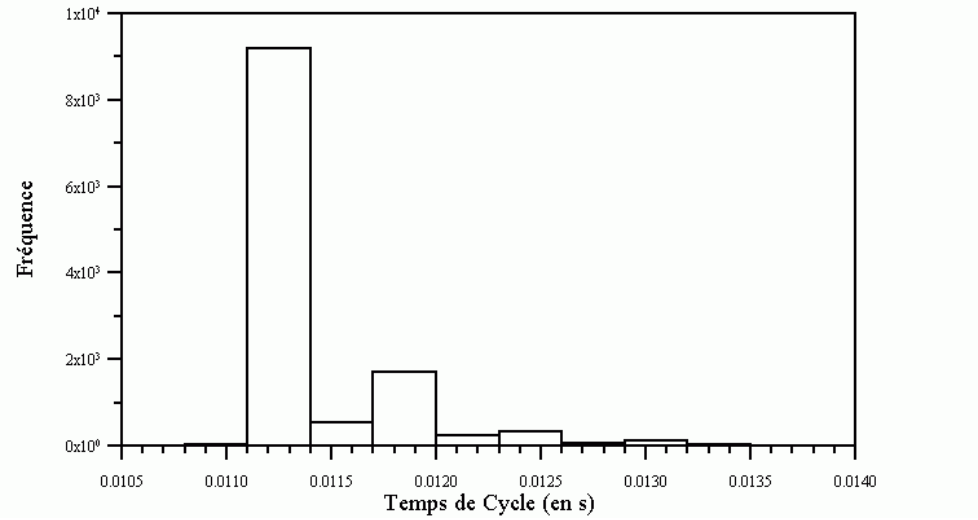


Figure 101 : Histogramme des mesures de temps de cycle d'un API

gramme est interrompue par une tâche de gestion *TG*. Cette tâche a pour durée D_{TG} égale à 0,6 ms et possède une période P_{TG} égale à 2,1 ms. Cette tâche étant périodique, elle pourra donc interrompre la phase d'ÉXécution n ou n+1 fois. Nous obtenons ainsi une nouvelle description du comportement du cycle d'un automate (fig. 102).

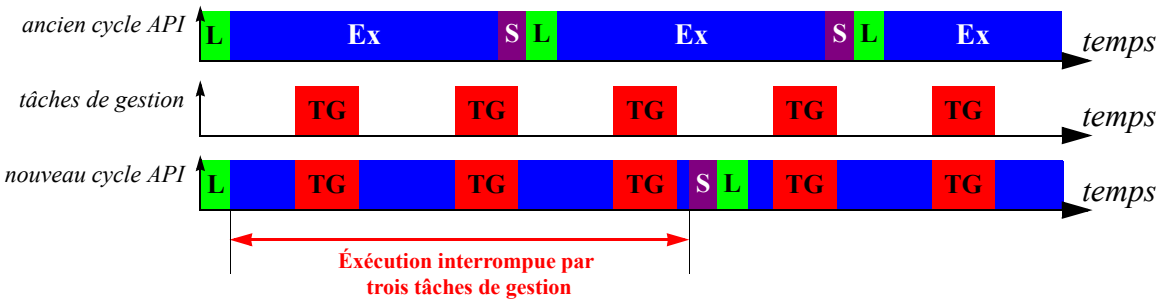


Figure 102 : Activité du CPU d'après l'identification du comportement

4.3.5 Application des méthodes de modélisation basée sur l'identification

Dans ce paragraphe, nous appliquerons les trois approches de modélisation au comportement décrit au paragraphe précédent

4.3.5.1 Modélisation structurelle basée sur une identification du comportement

La Modélisation a été réalisée de la même manière que dans le paragraphe 4.3.3.1, à ceci près qu'un nouveau point de vue est pris en compte : la tâche de gestion interne de l'API. On a donc un découpage du modèle en quatre parties :

- d'un coté, on retrouve les trois points de vues définissant le cycle représentés par les trois parties a, b et c sur la figure 103 ;
- de l'autre la partie du modèle gérant les tâches de gestion interne représentées par la partie d sur la figure 103.

On considère que seule la phase d'*EXecution des programmes* est interrompue régulièrement par des tâches de gestion. Les tâches de gestions apparaissent avec la période *PTG*. Le comportement est donc équivalent à attendre en fin de phase d'*EXecution des programmes* autant de fois la durée d'une tâche de gestion *DTG* qu'il y en a eu de générées. La modélisation de la partie *d* du modèle consiste à faire un générateur de tâches (transition *Générer TG*) et d'exécuter ces tâches (transition *Exécuter TG*) dès que la phase d'*EXecution des programmes* est terminée (double flèche avec poids e). La transition permettant de passer à la phase d'Ecriture des Sorties n'étant franchie que s'il n'y a plus de tâches de gestion à exécuter (double flèche avec poids 0). La figure 103 représente le modèle de comportement dynamique d'un API avec tâches de gestion cycliques périodiques.

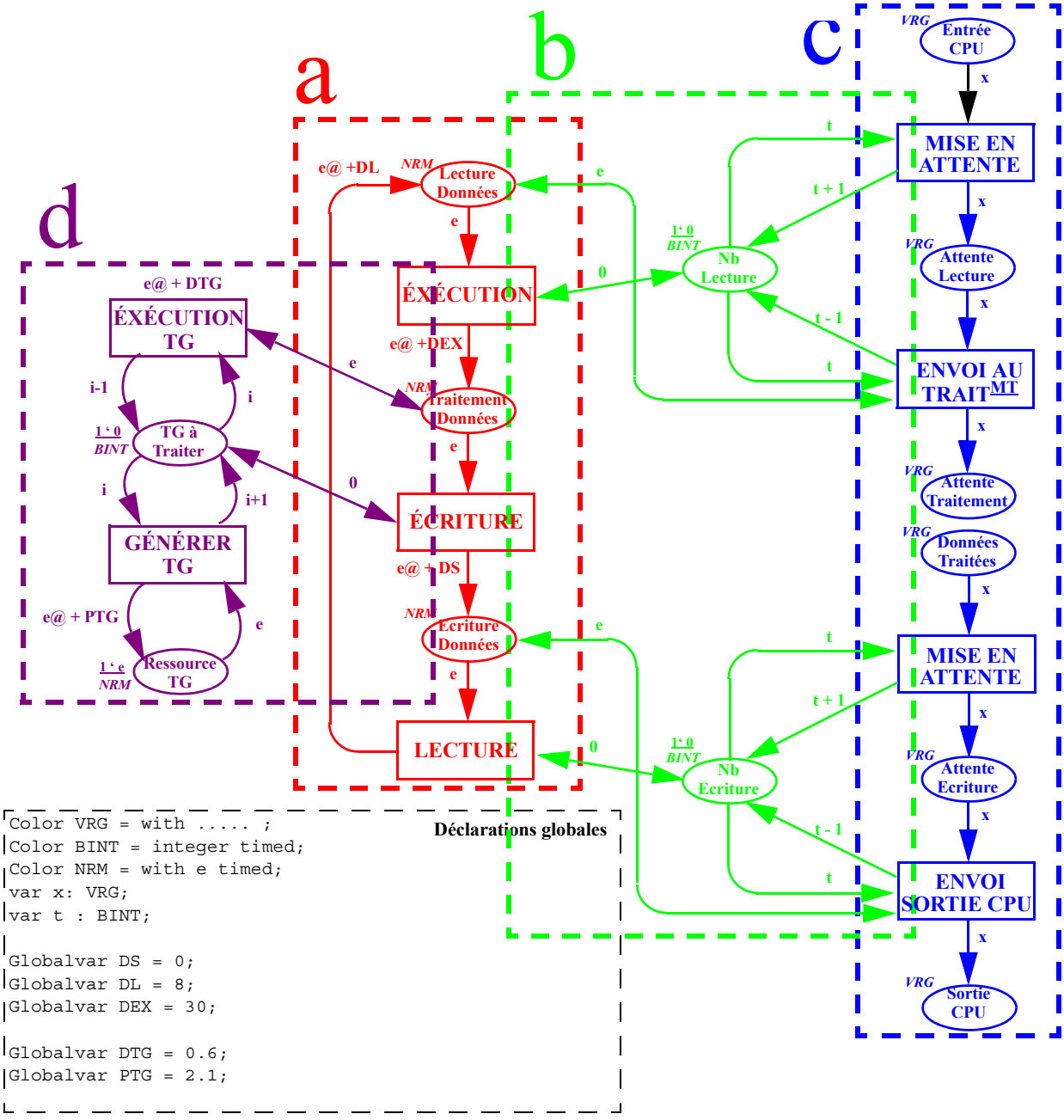


Figure 103 : Modèle RdPCT structurel d'après identification du comportement

Le modèle obtenu contient 15 places et 10 transitions. Pour *n* messages entrants, le nombre de transitions franchies est de $4 \times n + \left(\frac{\text{temps simulé}}{\text{temps de cycle}}\right) \times 3 + \left(\frac{\text{temps simulé}}{P_{TG}}\right) \times 2$.

4.3.5.2 Modélisation par interprétation basée sur identification du comportement

Tout comme pour le cas du paragraphe 4.3.3.2, le comportement dynamique d'un API peut être considéré comme un retard. Mais contrairement au cas précédemment défini, le calcul de ce retard n'est pas simple. Toutefois, il est possible d'estimer la date de la prochaine fin de cycle en connaissant la précédente date de fin de cycle. Étant donné l'ordre de grandeur entre les durées des phases composant le cycle CPU, on considère que les phases de *Lecture des Entrées* et d'*Ecriture des Sorties* sont négligeables face à la phase d'*EXecution des sorties*. Nous considérons donc les variables suivantes (fig. 104):

- FC_i, la dernière date de fin de cycle connue ;
- AT_i, la durée écoulée depuis la dernière tâche de gestion n°i ;
- BT_{i+1}, la durée qui s'écoulera avant la prochaine tâche de gestion n°i+1.

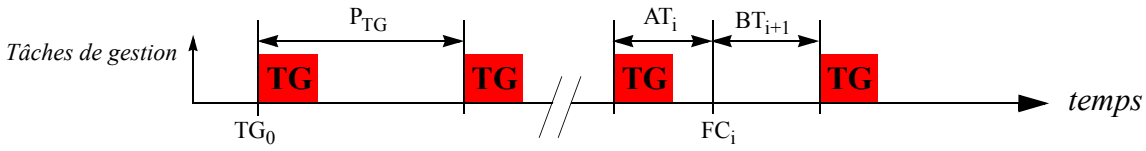


Figure 104 : Positionnement d'une date de fin de cycle FC_i par rapport aux tâches de gestion

On peut effectuer les calculs suivants :

$$AT_i = (FC_i - TG_0 - D_{TG}) \text{ Mod } P_{TG} \text{ où } Mod \text{ est l'opérateur modulo} \quad (13)$$

$$BT_{i+1} = P_{TG} - D_{TG} - AT_i \quad (14)$$

On peut alors distinguer deux cas : la durée BT_{i+1} est supérieure ou non à la durée de la phase d'*EXecution des programmes* (fig. 105 et fig. 106). Dans le premier cas, le cycle n°i a le temps de se terminer avant l'apparition de la prochaine tâche de gestion.

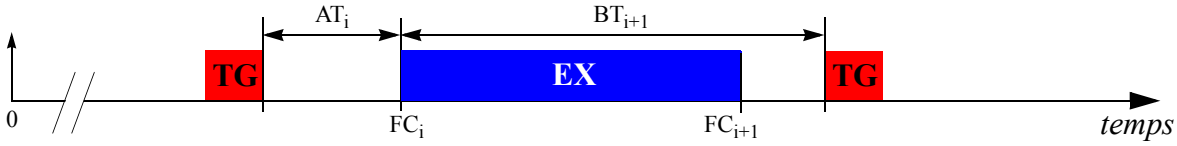


Figure 105 : Cas $BT_i > D_{EX}$

Le calcul de la prochaine date de fin de cycle est donc :

$$EC_{i+1}(EC_i) = EC_i + D_{CY} \quad (15)$$

Dans le second cas, le cycle n'a pas le temps de se terminer avant l'apparition de tâches de gestion. Une ou plusieurs tâches de gestion viennent donc s'intercaler durant la phase d'*EXecution des programmes*.

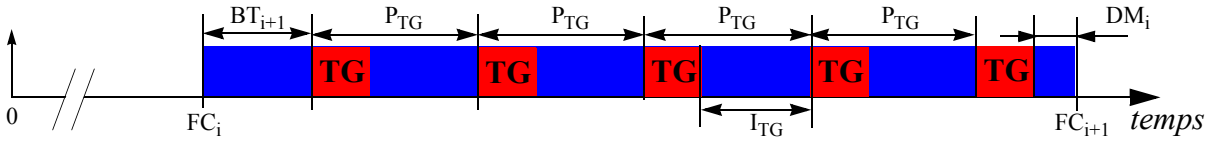


Figure 106 : Cas $BT_i \leq D_{EX}$

Nous définissons de nouvelles variables :

- I_{TG}, l'intervalle entre deux tâches de gestion ;
- np_i, le nombre de morceaux du cycle CY_i de taille I_{TG} ;
- DM_i, la taille du dernier morceau du cycle CY_i.

nous pouvons donc définir la prochaine date de fin de cycle FC_{i+1} :

$$FC_{i+1}(FC_i) = FC_i + BT_{i+1} + np_i \times P_{TG} + DM_i \quad (16)$$

$$I_{TG} = P_{TG} - D_{TG} \quad (17)$$

$$np_i = (D_{EX} - BT_i) \text{ Div } I_{TG} \text{ où } Div \text{ est la division entière} \quad (18)$$

$$DM_i = (D_{EX} - BT_i) \text{ Mod } I_{MT} \text{ où } Mod \text{ est la fonction modulo} \quad (19)$$

Nous avons déterminé la prochaine date de fin de cycle par rapport à l'ancienne date de fin de cycle. Mais ce qui nous intéresse c'est d'obtenir la date de fin de cycle qui suivra une date quelconque t. Il nous faut donc avoir la date de fin de cycle précédant la date t. Nous définissons pour cela deux nouvelles dates :

- D_{FC}, la dernière date de fin de cycle mémorisée ;
- DP_{FC}, la date de fin de cycle précédant D_{FC}.

On peut distinguer deux cas : la date t se trouve ou non entre les dates DP_{FC} et D_{FC}. On calcule alors la date de fin de cycle :

- Si la date est comprise dans l'intervalle alors la date de fin de cycle suivant la date t est D_{FC}.
- Dans le cas contraire il faut calculer les successeurs de D_{FC} (à l'aide de l'équation (14)) et ce jusqu'à que l'on ait $DP_{FC} < t$; D_{FC} sera alors la date de fin de cycle suivant la date t. L'algorithme de la figure 107 décrit le processus de calcul de la date de fin de cycle suivant une date t.

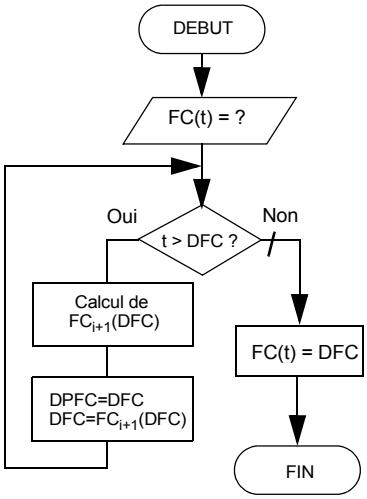


Figure 107 : Algorithme de calcul de FC(t)

La figure 108 représente le modèle du comportement dynamique d'un API avec tâches de gestion. Trois codes ont été associés à des transitions :

- le *code 1* mémorise la date d'arrivée (notée T) d'un événement ;

- le *code 2* sert au calcul de l'algorithme décrit par la figure figure 107 ;
- le *code 3* calcule le retard (noté *RETARD*) de transmission de l'événement traversant l'API.

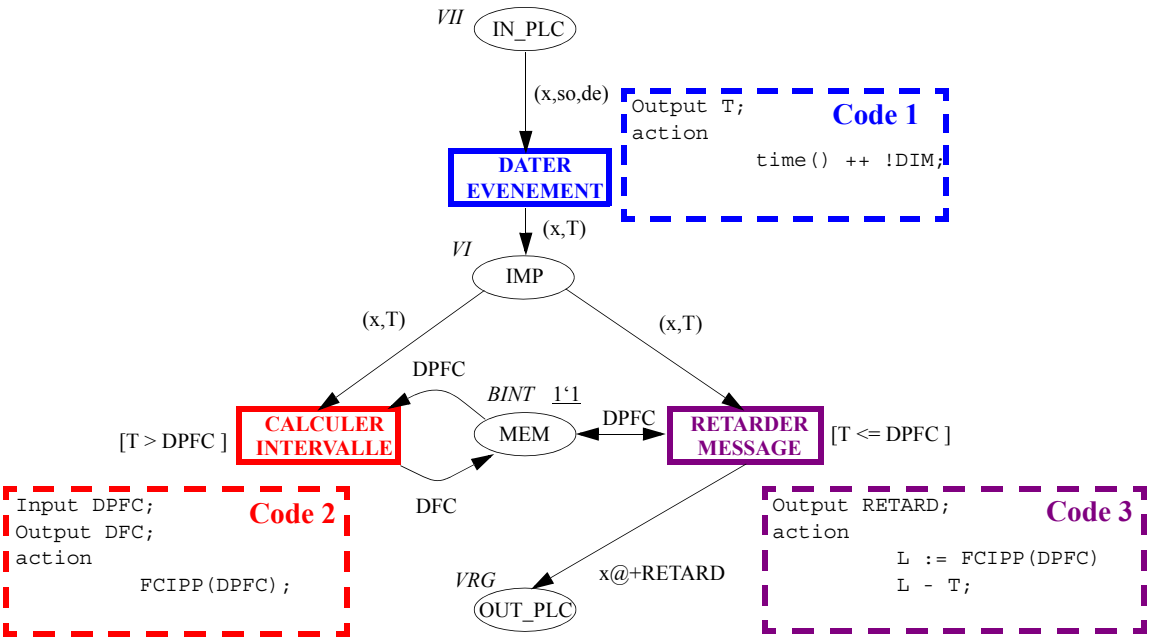


Figure 108 : Modèle RdPCT avec transitions interprétées d'après identification du comportement

Le modèle obtenu comprend 4 places et 3 transitions. Pour n messages entrants, le nombre de transitions franchies est approximativement $2 \times n + \left(\frac{\text{temps simulé}}{P_{TG}}\right)$.

4.3.5.3 Modélisation probabiliste basée sur une identification du comportement

D'après l'identification réalisée sur la plateforme expérimentale PRISME, on connait la répartition des temps de cycle de l'API. Afin de modéliser le comportement dynamique, nous allons définir une loi probabiliste dont la répartition correspond à celle observée.
On réalise une discrétisation de la fonction de probabilité, pour cela on regroupe la population en classes dont on identifie le pourcentage de la population totale et les bornes associées :

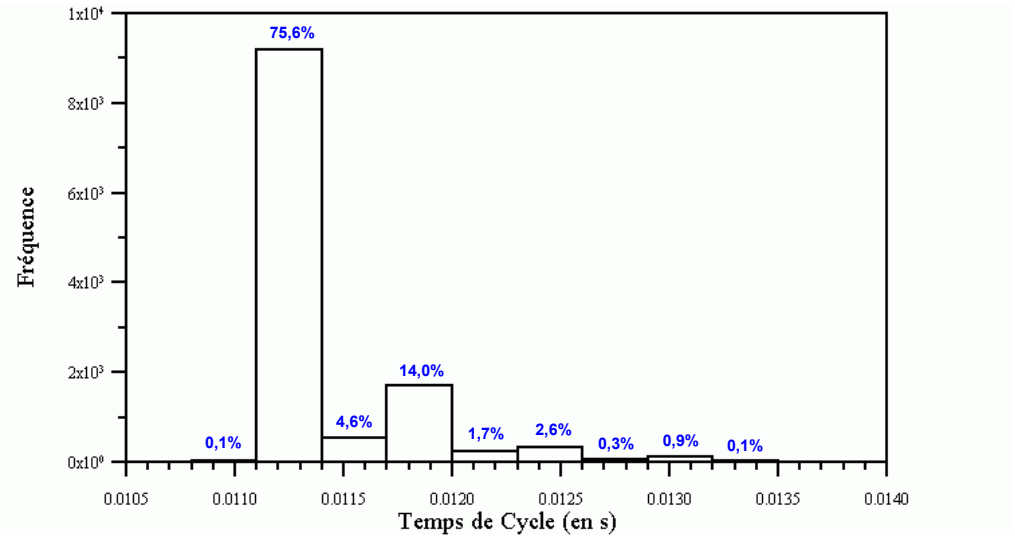


Figure 109 : Définition de la fonction de répartition correspondant au temps de cycle d'un API

On en déduit le tableau suivant :

Tableau 6 : Données de la répartition probabiliste correspondant au temps de cycle d'un API

N° de classe	Borne mini (en ms)	Borne maxi (en ms)	population
1	10,8	11,1	0,1%
2	11,1	11,4	75,6%
3	11,4	11,7	4,6%
4	11,7	12,0	14,0%
5	12,0	12,3	1,7%
6	12,3	12,6	2,6%
7	12,6	12,9	0,3%
8	12,9	13,2	0,9%
9	13,2	13,5	0,1%

On définit à partir de ce tableau le code ML effectuant dans un premier temps un tirage aléatoire entre 1 et 1000 (la taille des classes ayant une précision d'un 1/10ième de pourcent) afin de déterminer dans quelle classe on se trouve puis un second tirage aléatoire entre les deux bornes de la classe concernée.

Figure 110 : Modèle RdPCT probabiliste basé sur l'identification du comportement

L'unité de temps utilisée pour la simulation étant le 1/10ième de milliseconde, les bornes seront donc multipliées par 10 par rapport aux valeurs du tableau.

Le modèle obtenu (fig. 110) comprend 2 places et 1 transition. Pour *n* messages entrants, le nombre de transitions franchies est de 1.

4.4 Étude comparative des modèles obtenus

Dans le paragraphe 4.3.3, nous avons conçu six modèles réseau de Petri en nous basant sur les différentes méthodes de modélisation et sur les différentes approches d'obtention du comportement à modéliser. Afin de déterminer quel modèle est le plus performant pour notre objectif de simulation, nous avons besoin de comparer de ces six modèles. Dans ce paragraphe, nous allons décrire les critères que nous utiliserons pour comparer les modèles; puis nous présenterons la démarche d'obtention de la référence de mesure; alors nous expliciterons la mise en oeuvre de la simulation des modèles obtenus et enfin, nous procéderons à leurs comparaison afin d'aider l'architecte à choisir le modèle qui lui convient le mieux.

4.4.1 Critères de comparaison

Puisque nous disposons de l'environnement expérimental nécessaire, nous effectuerons une comparaison des performances de ces différents modèles avec le comportement réel mesuré de l'API. Les critères que l'on utilisera pour juger de la qualité de ces modèles sont :

- La justesse : correspondant à l'aptitude du modèle à produire un comportement proche du réel ;
- L'efficience en simulation: correspondant au temps pris par la simulation du modèle.

Pour quantifier la justesse d'un modèle, nous observerons la répartitions des résultats de simulation par rapport à une référence donnée. Pour comparer des répartitions, on utilisera les informations statistiques classiques que sont le minimum, le maximum, la moyenne et l'écart type. Pour effectuer des comparaisons, nous calculerons les écarts entre les données statistiques de mesure et celles de référence. On définit ainsi :

$$\Delta_{min} = \frac{Min(reference) - Min(simulation)}{Min(reference)} \tag{20}$$

$$\Delta_{max} = \frac{Max(reference) - Max(simulation)}{Max(reference)} \tag{21}$$

$$\Delta_{moyenne} = \frac{Moyenne(reference) - Moyenne(simulation)}{Moyenne(reference)} \tag{22}$$

$$\Delta_{ecarttype} = \frac{Ecarttype(reference) - Ecarttype(simulation)}{Ecarttype(reference)} \tag{23}$$

Ces informations ne permettant pas de juger de la «proximité» entre deux répartitions, nous utiliserons donc de plus le taux de couverture du modèle par rapport à la référence (comportement réel). Par exemple, à la suite d'une simulation, si nous avons la répartition des résultats indiquée sur la figure 111, cette répartition couvre une partie de la répartition de référence (indiquée par les barres pleines sur la figure 111).

La partie couverte correspond aux barres rayées sur la figure 111. Le taux de couverture correspond alors à :

$$\text{taux de couverture} = \frac{\text{nombre de résultats couverts}}{\text{nombre de résultats}} \tag{24}$$

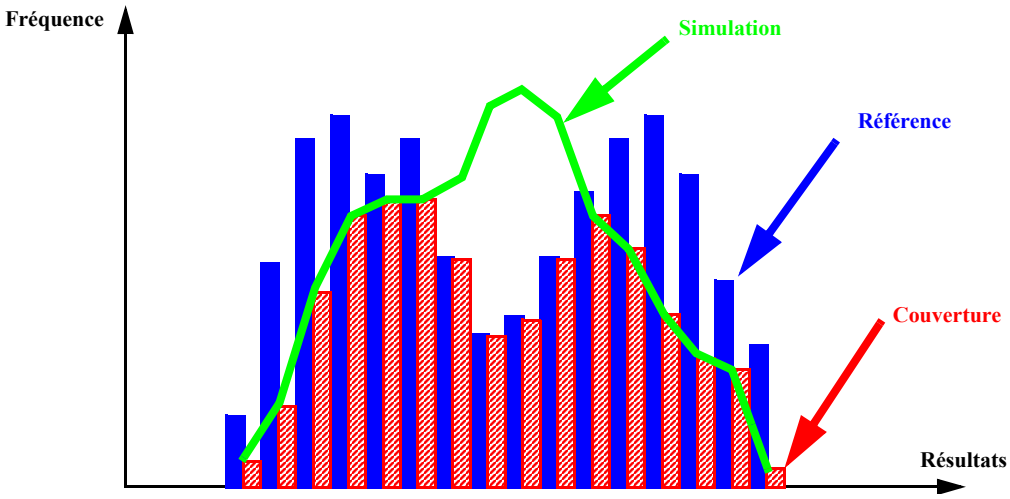


Figure 111 : Exemples de taux de couverture

4.4.2 Obtention de la référence de mesure

Nous utilisons pour cela la plate-forme PRISME, afin de mesurer le temps de propagation d'une information au travers d'un API, grâce au programme décrit par la figure 112.

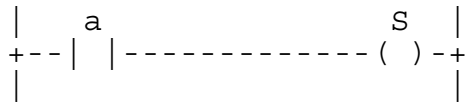


Figure 112 : programme de mesure de temps de propagation au travers d'un API

L'entrée *a* de l'API est stimulée par un signal carré de période *P* à l'aide d'un des deux générateurs de signaux de PRISME. On mesure, grâce à l'analyseur logique de PRISME, le temps de réaction, c'est-à-dire le temps entre un front montant du signal *a* et le premier front montant de la sortie *S* suivant le front montant de *a* (fig. 113).

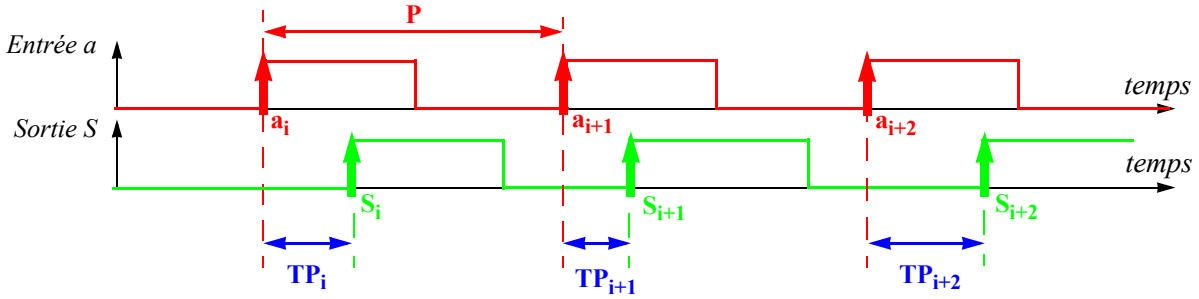


Figure 113 : Mesure de temps de propagation à travers un API

Nous avons réalisés 60000 mesures afin de pouvoir par la suite effectuer des études statistiques. La figure 114 représente l'histogramme des résultats de mesure sur un API cohérent avec les hypothèse que nous avons précédement formulées au début de ce chapitre : mono-processeur, cyclique, avec le même programme éxécuté à chaque cycle

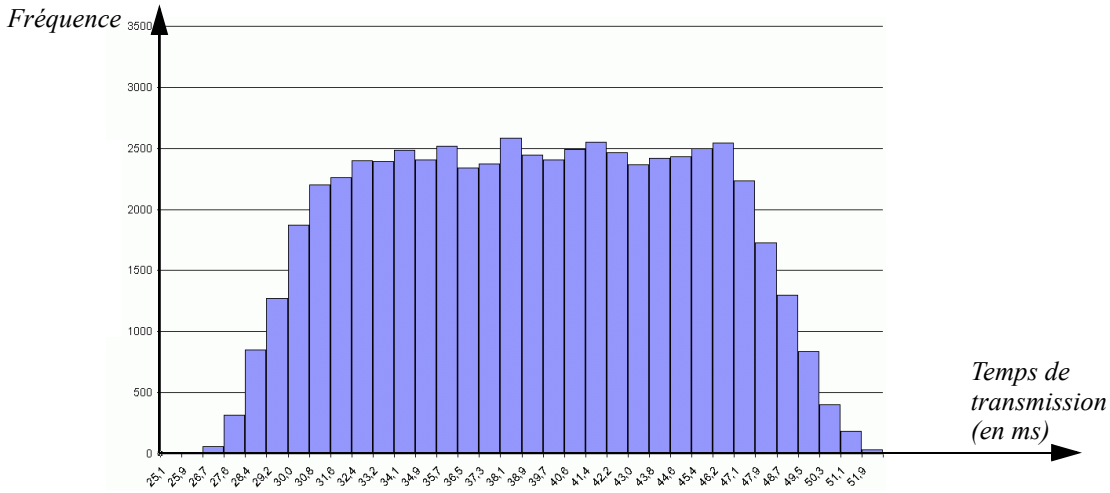


Figure 114 : Histogramme des résultats de mesure

4.4.3 Mise en oeuvre de la simulation des modèles

Pour chacun des modèles obtenus, nous avons réalisé une simulation. Pour chacune d'elles, les excitations des entrées du modèles sont identiques; nous évaluons les temps de transmissions d'une information au travers de l'API. Afin de pouvoir réaliser des études statistiques, nous stimulons les modèles de façon à avoir 60000 mesures. La figure 115 représente le modèle réseau de Petri correspondant à un générateur de signal à l'entrée *a* de l'API.

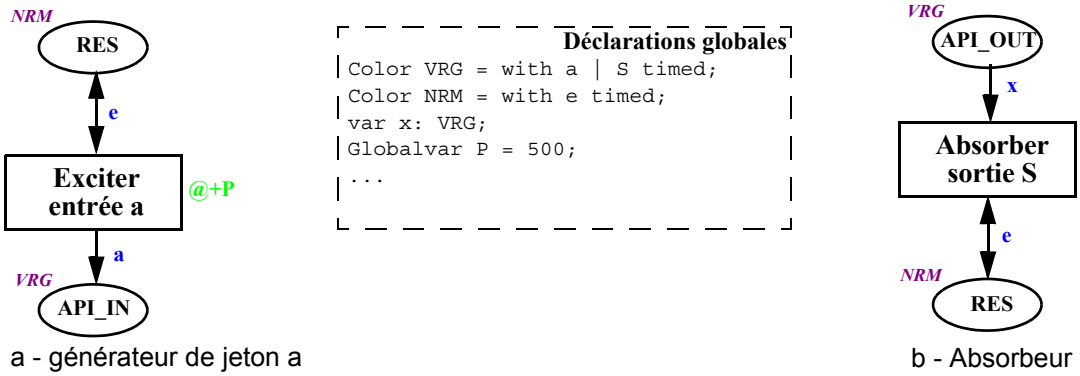


Figure 115 : Modèle d'excitation et d'absorption de jetons

Afin de ne pas surcharger le modèle de simulation avec un nombre croissant de jetons (provenant de l'exitateur décrit figure 115a), nous plaçons un absorbeur de jeton en sortie de l'API. Le modèle réseau de Petri correspondant à l'absorbeur est décrit par la figure 115b. Au centre de la figure 115 se trouve la partie des déclarations globales nécessaires à l'exécution des ces deux modèles.

La simulation que l'on souhaite réaliser prend en compte l'aspect matériel, mais aussi l'aspect fonctionnel; il faut donc modéliser le programme qui est implanté au sein de l'API. La figure 116 représente le modèle réseau de Petri du programme décrit figure 112.

Pour finaliser le modèle de simulation, il est nécessaire d'ajouter au modèle des observateurs, qui permettront de mesurer les performances du modèle. Dans notre cas, il nous faut mesurer le temps de trans-

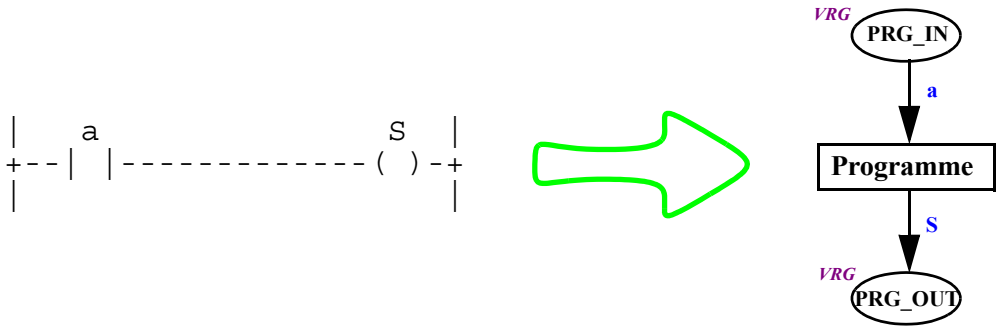


Figure 116 : Modèle du programme implanté

mission au travers d'un API, soit le retard entre l'apparition d'un événement en entrée et sa réaction à la sortie de l'API. On place donc deux observateurs, le premier pour observer l'apparition de l'événement à l'entrée *a* de l'API; le second pour observer la réaction à la sortie *S* de l'API (fig. 117).

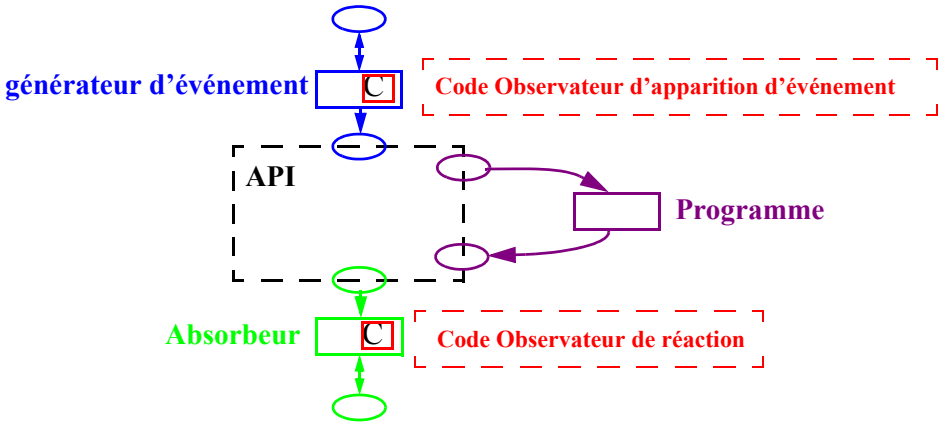


Figure 117 : Emplacement des observateurs

On réalise les simulations pour chacun des modèles obtenus dans le paragraphe 4.3.3. Afin de pouvoir réaliser la comparaison de ces modèles, les résultats de simulation, sont transcrits sous la forme d'histogrammes (fig. 118). Pour permettre une comparaison, tous ces histogrammes sont représentés à la même échelle sur le même intervalle.

4.4.4 Première analyse des résultats obtenus

Les modèles interprétés ont un code qui est normalement conçu pour remplacer la lourde structure réseau de Petri des modèles structurels. Toutefois, on remarque que les résultats de simulation sont légèrement différents comme indiqués sur la figure 119.

Ces modèles étant tous deux excités de la même manière, il est possible de comparer leurs temps de transmission résultat par résultat. On réalise la différence entre chaque résultat de simulation, on trouve que 99,44% des différences sont nulles. Il se trouve donc 0,54% de résultats qui diffèrent entre les deux modèles. Cette proportion est certes minime, mais montre qu'il y a bien une différence de comportement entre les deux modèles. Étant donné que la conception du modèle interprété est basée sur le modèle structurel, on devrait avoir les mêmes résultats. Nous allons donc analyser les causes de ces différences.

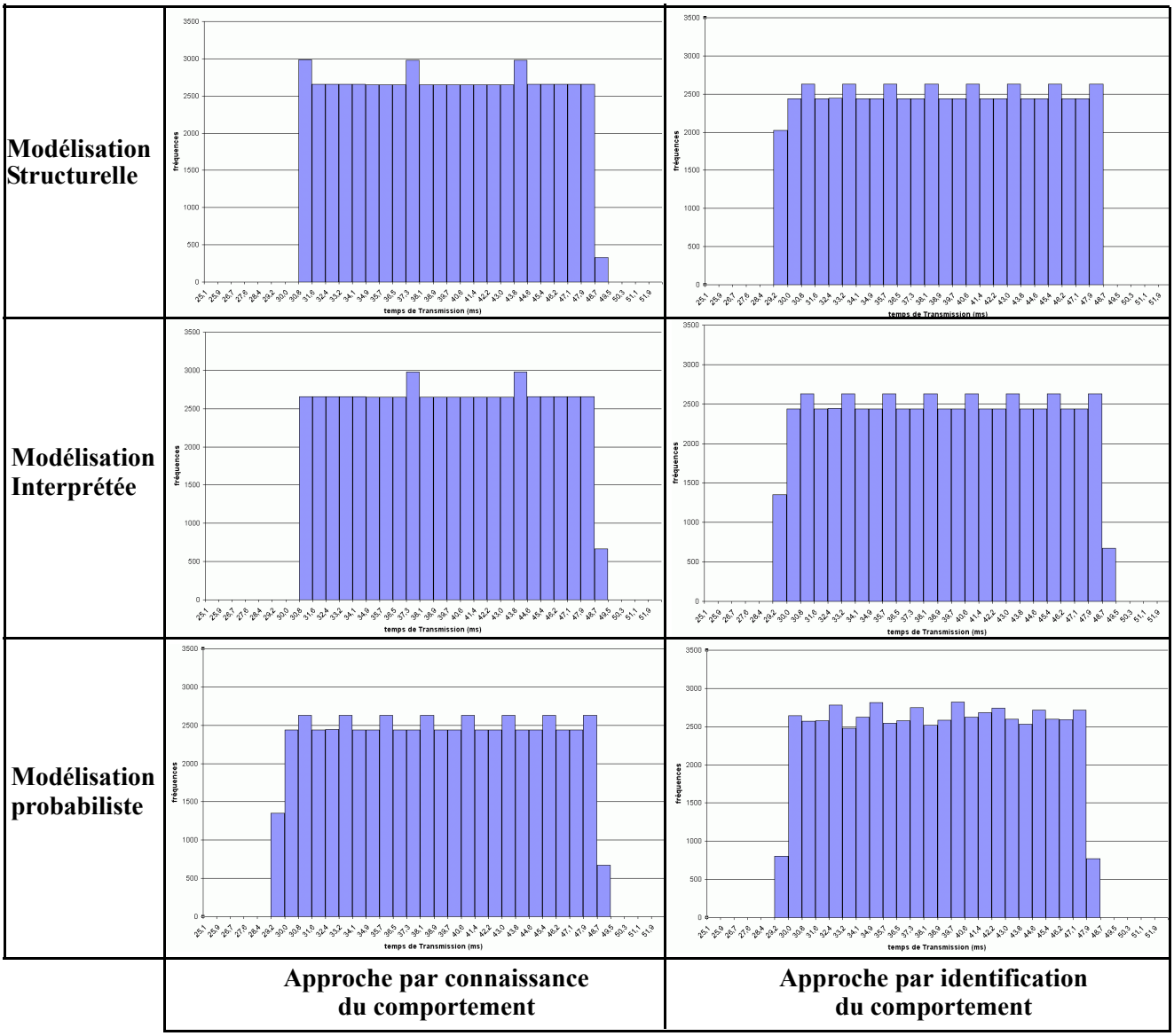


Figure 118 : Histogrammes résultats des simulations

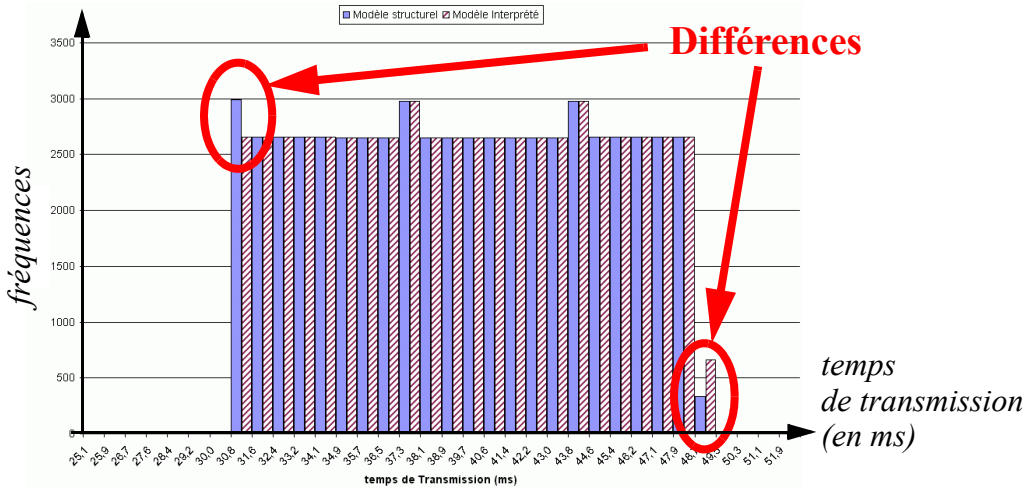


Figure 119 : Histogramme des modèles structurels et interprétés d'après la synthèse d'information

Il existe des effets de bord lorsqu'un événement arrive à l'entrée de l'automate à la fin de la phase de lecture des entrées. L'événement peut alors être pris en compte immédiatement ou attendre la prochaine phase de lecture des entrées. Afin d'être cohérent, il convient de choisir l'une des deux hypothèses et de ne plus en changer.
Le modèle interprété prend en compte tous les événements qui interviennent dans la période de la phase de lecture, incluant les événements qui interviennent à la date de fin de cette phase comme l'indique le signe inférieur ou égal de l'équation (7) de la page 98. Le modèle interprété répond à cette cohérence.
Pour le modèle structurel, on commence la phase de traitement lorsque la phase de lecture des entrées est terminée, donc que la temporisation gérant la durée de cette phase est terminée et que toutes les entrées à prendre en compte sont lues.

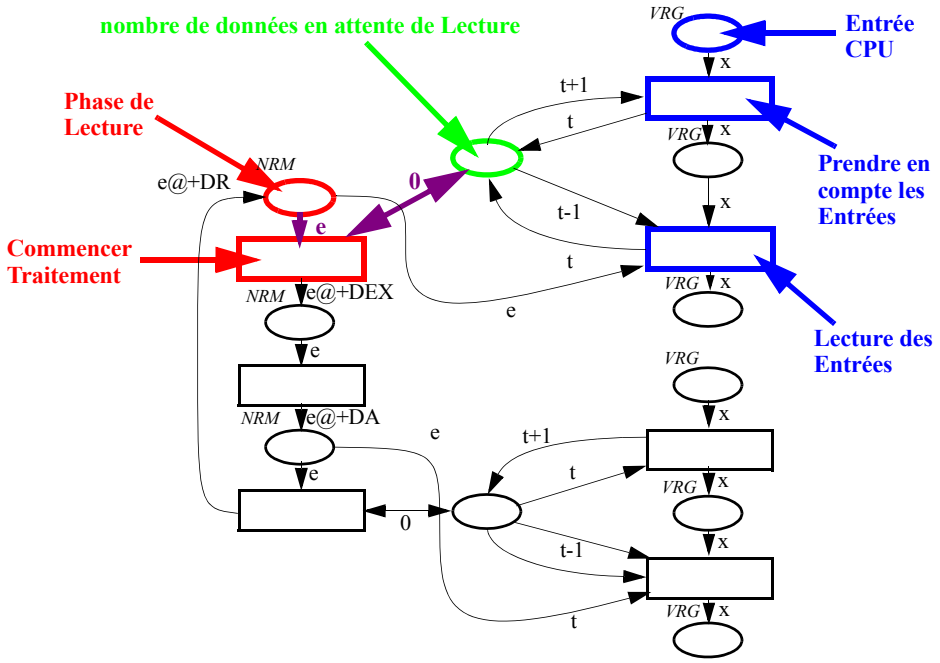


Figure 120 : Modèle RdPCT structurel d'après connaissance du comportement

La figure 120 représente le modèle structurel d'un API obtenu par synthèse du comportement. Sur cette figure, on retrouve la transition qui permet de commencer la phase *d'EXécution des programmes* dont le franchissement nécessite qu'un jeton soit disponible dans la place de phase de lecture (fin de temporisation de phase de lecture) et qu'un jeton de valeur zéro soit disponible dans la place indiquant le nombre de messages en attente de lecture (toutes les entrées ont été lues).
Si l'on se place dans l'hypothèse d'arrivée d'un message à la date de fin de la phase de lecture, on se retrouve avec deux transitions franchissables : celle qui permet de prendre en compte les entrées et celle qui permet de commencer l'Exécution des programmes.
Si c'est la transition de prise en compte des entrées qui est franchie, alors le nombre de données en attente de lecture devient égal à 1 et donc la transition «Commencer *traitement*» n'est plus franchissable jusqu'à que l'on ait lecture des entrées et remise à zéro du nombre de données en attente de lecture. Si par contre c'est la transition «Commencer *traitement*» qui est la première à être franchie alors, l'entrée sera prise en compte durant la prochaine phase de lecture.
Sachant que si l'on a deux transitions simultanément franchissables durant une simulation, le choix de la transition à franchir est déterminé aléatoirement par le logiciel de simulation. Le modèle structurel n'est donc pas déterministe à cause d'un effet de bord à la limite de la phase de lecture des entrées.

4.4.5 Comparaison des modèles construits

La figure 121 représente les différents histogrammes obtenus après simulations des différents modèles et identification du comportement réel de l'API. Afin de faciliter la lecture du graphique, les histogrammes barres on été remplacés par des courbes lignes.

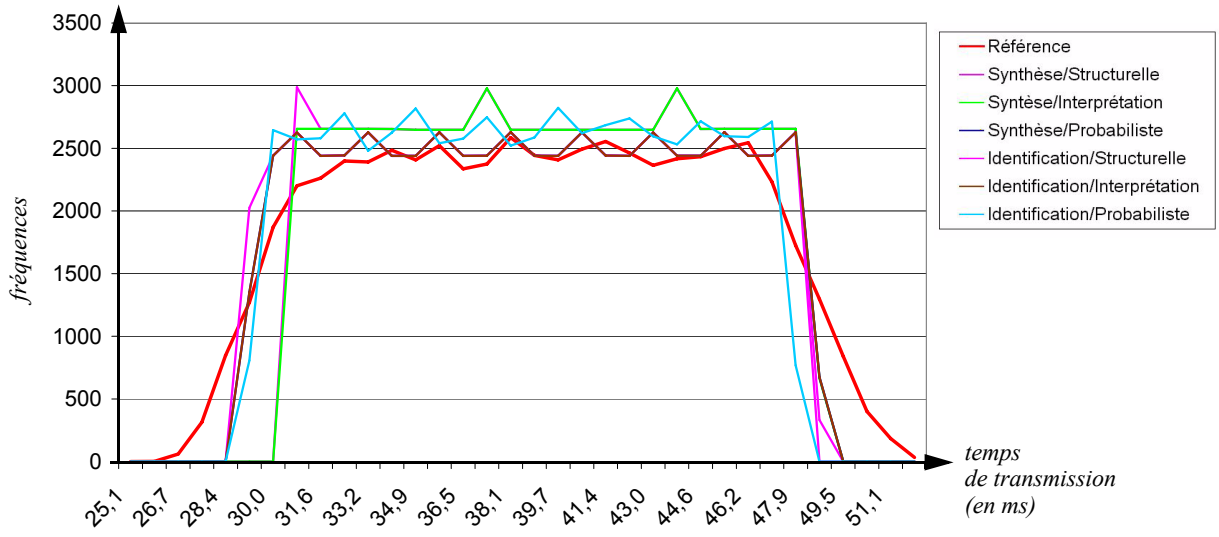


Figure 121 : Histogramme de comparaison des modèles

Le tableau 7 contient l'analyse des résultats de simulation. Pour chaque ligne, les meilleures valeurs sont indiquées en gras de couleur bleu, les deuxièmes meilleures valeurs (deltas minimums, couvertures maximales et durées minimales) sont soulignées et les pires valeurs sont en italique de couleur rouge.

Tableau 7 : Analyse des résultats

	Approche par connaissance			***Approche par Identification***		
	Struct.	***Interp.***	***Proba.***	***Struct.***	***Interp.***	***Proba.***
$\Delta_{minimum}$	-6,4%	-9,1%	**-0,8%**	-10,7%	-10,7%	*-11,8%*
$\Delta_{maximum}$	7,8%	6,4%	**-2,6%**	6,9%	6,9%	*8,9%*
$\Delta_{moyenne}$	-0,9%	-1,1%	0,6%	**0,5%**	**0,5%**	*1,2%*
$\Delta_{écart\_type}$	*36,3%*	34,4%	**18,7%**	21,2%	**18,7%**	29,3%
Taux de couverture	90,2%	*89,2%*	**94,0%**	92,9%	**94,0%**	90,8%
Durée simulation	*13h15'*	**49'25"**	**49'25"**	*13h55'*	4h22'	**49'25"**

En analysant le tableau 7, on remarque que le modèle probabiliste construit par connaissance est celui qui possède quasiment l'ensemble des meilleurs résultats :

- son temps de simulation est le plus petit (ce qui s'explique par le fait que ce modèle ne contient qu'une unique transition) ;
- ses résultats statistiques sont très proches de ceux du modèle réel ;
- son taux de couverture est le plus important.

Toutefois, du fait de sa nature probabiliste, ce modèle ne pourra pas être utilisé dans le cas où le modèle API interviendrait dans l'étude de performances. En effet, si l'on a deux événements corrélés en entrée de l'API, ils sortirons corrélés avec un modèle structurel ou par interprétation, ce qui n'est pas le cas pour un modèle probabiliste. Cela est dû au fait qu'un tirage probabiliste est réalisé pour chacune des informations, et donc la correlation sera perdue. Si un tel modèle interviennait dans le trajet d'une information servant pour l'étude de performances, les données seraient alors faussées et n'auraient plus aucun sens.
Par contre, dans le cas où l'équipement ne participerais qu'à la modélisation de la charge de l'architecture et non à l'analyse des performances, l'architecte peut choisir un modèle probabiliste.

Pour l'étude de performances, l'architecte a donc besoin d'un autre modèle d'API. Parmi les modèles non-probabilistes, on retient le modèle interprété construit à partir de l'identification du comportement car il a de bons résultats statistiques et le taux de couverture le plus important. Toutefois deux éléments peuvent s'avérer genant pour l'architecte :

- le temps de simulation de ce modèle est près de cinq fois plus important que le modèle précédent ;
- le paramétrage du modèle nécessite d'avoir réalisé une identification de l'équipement.

Dans le cas où l'architecte n'a pas la possibilité expérimentale de réaliser une identification de son équipement, il est donc nécessaire d'avoir à disposition un modèle non-probabiliste obtenu par connaissance. Au final, trois modèles intégreront donc la bibliothèque de modèle pour leur fidélité par rapport au comportement réel :

- le modèle de comportement obtenu par connaissance et conçu de manière probabiliste ;
- le modèle de comportement obtenu par identification et conçu par interprétation ;
- le modèle de comportement obtenu par connaissance et conçu par interprétation.

L'architecte pourra ainsi, suivant ses besoins avoir a disposition le modèle qui conviendra le mieux à ses besoins.

4.5 Conclusion

Dans cette partie, nous avons décrit notre méthodologie de conception de modèles réseau de Petri pour les équipements matériels. L'application de cette méthode sur l'exemple d'un API, nous a amené à la construction de six modèles distincts. Pour valider ces modèles, nous avons réalisé une étude comparative qui nous a amené au choix de trois modèles pour la bibliothèque.

Notre méthodologie propose six approches de modélisation d'un équipement matériel (réseaux, API avec cycles non périodiques ou interrompus par des tâches rapides, …), mais toutes ces approches ne sont pas toujours réalisables ou réalistes.

C'est par exemple le cas pour un réseau de type Profibus avec un protocole maître-esclave qui a un comportement complexe : les messages du maître sont prioritaires par rapport à ceux des esclaves qui sont interrogés de manière cyclique («polling»). Une modélisation structurelle permet de modéliser aisément ce comportement en utilisant des places ressources qui permettent de gérer les interruptions et le «polling». Par contre une modélisation par interprétation ne semble pas réalisable car les retards associés aux messages provenant d'esclaves peuvent évoluer si un message du maître apparaît. Quant à une

modélisation probabiliste, le résultat serait que le réseau n'aurait pas un comportement réaliste car le cycle d'interrogation des esclaves et les priorités entre les messages provenant du maître ou des esclaves ne seraient pas assurées. Dans ce cas, il est plus judicieux de n'utiliser que l'approche par modélisation structurelle.

Notre méthodologie présente donc diverses voies qui sont mises à la disposition des utilisateurs qui désirent réaliser des modèles d'équipements matériels. Dans le cas où plusieurs modèles seraient conçus, nous proposons une technique de comparaison des modèles pour vérifier leur validité et pour permettre leur sélection pour la bibliothèque.

Chapitre 5

Validation de la démarche

« Evaluer, c'est créer : écoutez donc, vous qui êtes créateurs !
C'est l'évaluation qui fait trésors et joyaux de toutes choses évaluées. »
Friedrich Nietzsche

Dans ce chapitre, nous validons notre démarche en l'appliquant sur un exemple complet et en étudiant les résultats de notre démarche afin de déterminer si elle peut répondre aux attentes de l'architecte.
Ainsi, nous présentons dans un premier temps l'Architecture Opérationnelle support de la validation. Elle a été conçue afin d'être représentative des architectures industrielles.
Ensuite, nous présentons les résultats des performances observées à la fois par simulation mais aussi par mesure expérimentale sur notre plateforme.
Enfin, nous évaluons la qualité de notre approche, en réalisant une étude de convergence des performances en simulation, une étude de sensibilité des performances aux erreurs de paramétrage et une étude de la précision des performances obtenues par simulation par rapport à celles mesurées sur l'architecture réelle.

5.1 Introduction

Dans les chapitres précédents, nous avons présenté notre démarche permettant d'évaluer les performances d'Architectures Opérationnelles. Dans ce chapitre, nous vérifierons la praticabilité de notre méthode en traitant un exemple complet dont nous expliciterons le choix, puis nous nous intéresserons à la qualité de notre méthode en étudiant dans un premier temps la convergence de nos simulations (erreur fonction de la durée de simulation), puis la sensibilité de nos modèles de simulation aux erreurs de paramétrage. Enfin nous évaluerons la précision de nos résultats en les comparant à des résultats de mesure sur l'architecture réelle.

5.2 Choix de l'architecture support de la validation

5.2.1 Problématique

En milieu industriel, les performances couramment étudiées sont de type temps de *transmission d'informations* au travers d'une architecture, souvent multi-niveaux. Ces architectures étant très variées, il nous faut pour valider notre démarche étudier une architecture suffisamment complexe et représentative que ce soit au niveau matériel ou au niveau fonctionnel.
En se basant sur les architectures industrielles rencontrées, nous définissons l'architecture étudiée comme devant comporter :

- plusieurs niveaux de réseaux interconnectés,
- des équipements de traitements de type Automate Programmable Industriel,
- des modules d'entrées-sorties déportées,
- plusieurs fonctions échangeant régulièrement des informations au travers de l'architecture afin d'étudier un comportement réaliste de l'architecture en charge.

5.2.2 Architecture Matérielle

L'Architecture Matérielle doit être représentative des contraintes définie ci-dessus (i.e. suffisament complexe). De plus, elle doit aussi être une architecture existante afin de pouvoir faire des mesures sur un système réel et ainsi comparer le comportement réel avec celui du modèle.

Afin de répondre à notre besoin d'une architecture à plusieurs niveaux de communication, nous utiliserons à la fois un réseau d'automates Uni-Telway, un réseau AS-i d'entrées-sorties déportées et une communication point-à-point entre deux API. Le réseau Uni-Telway est un réseau à protocole maître-esclaves avec ici un maître et quatre esclaves. Le réseau AS-i comporte 8 modules esclaves d'entrées/sorties. La liaison point-à-point entre deux automates programmables représente la liaison entre l'architecture de contrôle-commande et une commande pré-définie pour un équipement livré «clé en main».

Au niveau des équipements de traitement, nous disposons d'un ensemble de différents automates programmables : un TSX 57 Premium, un TSX 37 Micro et deux TSX Nano de marque Schneider Electric ainsi qu'un S7-200 de marque Siemens et une console avec un atelier de développement pour assurer la maintenance des programmes API. Cela nous permet finalement d'obtenir l'architecture matérielle décrite par la figure 122.

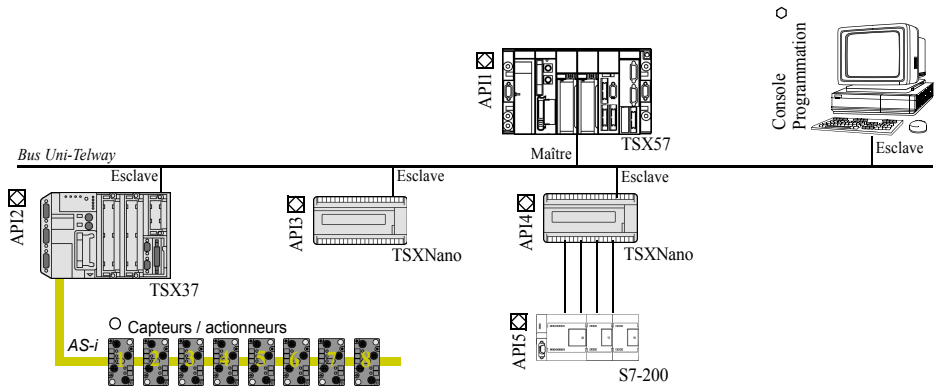


Figure 122 : Architecture Matérielle

Pour chacun des équipements de traitements, on associe un rôle sur une Partie Opérative supposée :

- le TSX 57 (API1) aura un rôle de supervision, soit l'émission et la réception des messages ainsi que l'échange de tables de données avec les différents autres organes de traitement.
- les TSX 37 (API2), TSX Nano (API3 et API4) et S7-200 (API5) assurent la gestion d'un sous-ensemble (Poste) de la Partie Opérative ainsi que leurs coordinations entre eux.

5.2.3 Architecture Fonctionnelle

Nous considérerons l'Architecture Fonctionnelle décrite par la figure 123.

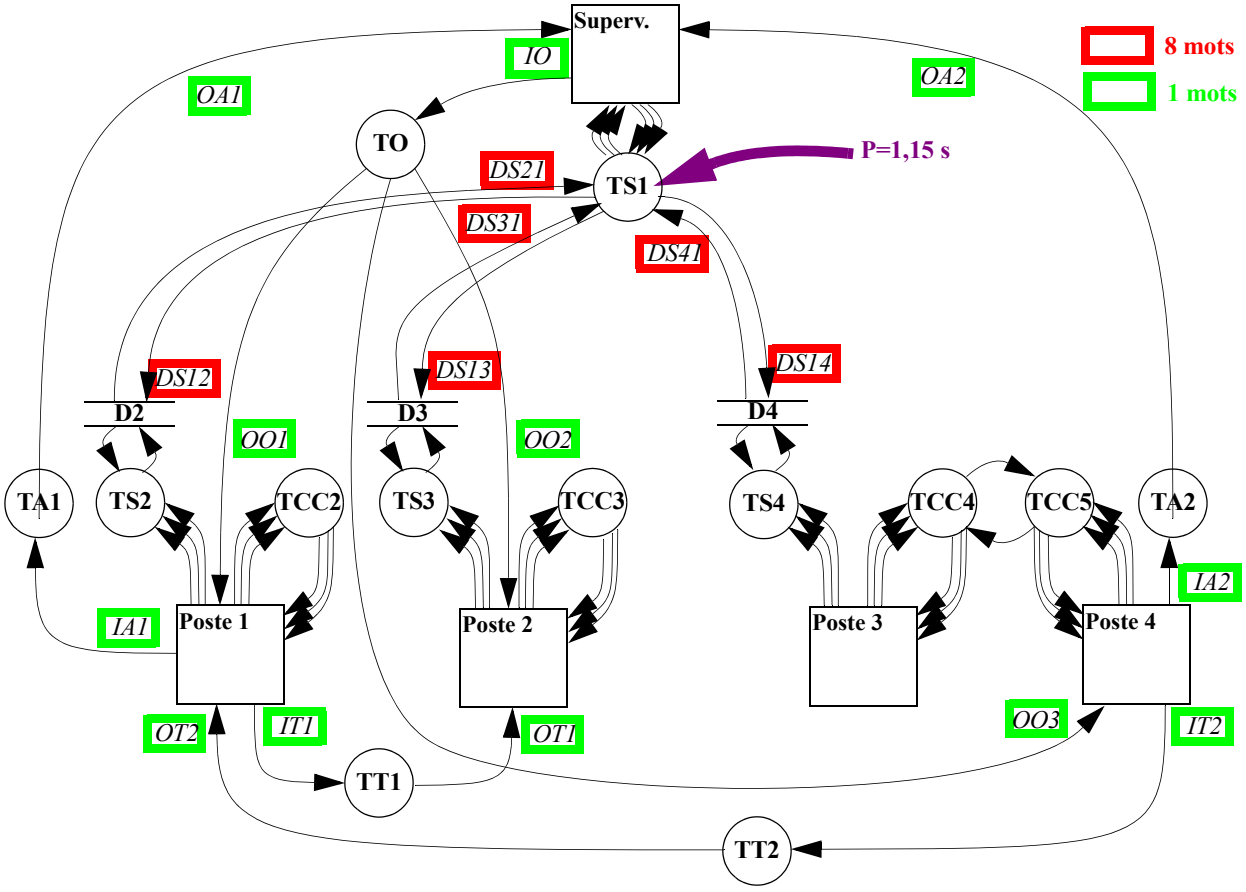


Figure 123 : Architecture Fonctionnelle

Les tâches fonctionnelles composant cette Architecture Fonctionnelle sont :

- TS1, la Tâche qui centralise et distribue les informations liées à la supervision ;
- TSi, avec $i \in [2,4]$, les tâches qui localement produisent les informations pour la supervision des postes n°*i-1* ;
- Di, avec $i \in [2,4]$, les Données échangées pour la supervision du poste n°*i-1* ;
- TCCi, avec $i \in [2,4]$, les Tâches de contrôle-commande des poste n°*i-1I* ;
- TAi, les Tâches des gestion des Alarmes provenant des postes ;
- TOi, les Tâches de gestion des Ordres à transmettre aux différents postes ;
- TTi, les Tâches de Coordination entre deux postes.

La tâche TS1, de supervision, réalise toute les 1150 millisecondes un échange d'une table de 8 mots avec les tables de données D2, D3 et D4.

5.2.4 Architecture Opérationnelle

La réalisation d'une projection "optimale" de l'Architecture Fonctionnelle sur une Architecture Matérielle est un travail complexe qui fait l'objet de nombreux travaux de recherche dans la communauté scientifique, comme nous l'avons présenté au paragraphe 1.2.3 page 29. Notre objectif n'est pas ici de réaliser la meilleure projection, mais d'avoir une Architecture Opérationnelle dont la projection comporte à la fois des traitements placés au plus proches des interfaces et aussi des traitements volontairement éloignés des interfaces. Ce qui nous permettra d'étudier à la fois des configurations «classiques» mais aussi des configurations plus «élaborées».
Ainsi, nous plaçons les tâches associées aux ordres ou aux alarmes au plus près des interfaces dans des équipements de traitement qui correspondent à une entrée ou une sortie physique de ces tâches.
Pour le cas des transmissions transversales, nous plaçons leurs tâches associées éloignées des interfaces dans des équipements de traitement qui ne sont pas connectés physiquement aux entrées ou aux sorties de ces tâches.
La figure 124 représente l'Architecture Opérationnelle dont nous évaluerons les performances.

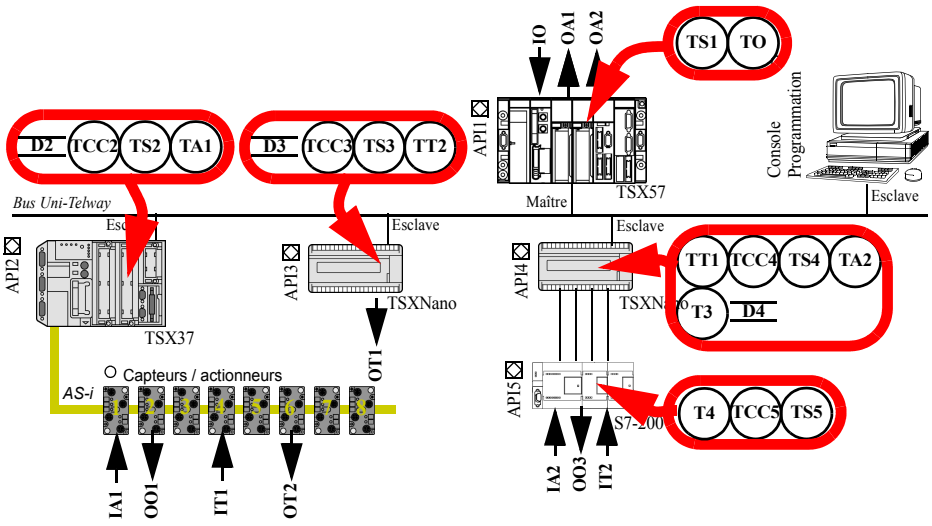


Figure 124 : Projection de l'Architecture Fonctionnelle sur l'Architecture Matérielle

5.2.5 Performances à évaluer

On souhaite évaluer des performances de type temps de réponse classiques pour une architecture de contrôle-commande. Les observations que l'on réalisera seront des temps de transmissions d'informations qu'elles soient ascendantes (d'un poste vers la supervision), descendantes (de la supervision vers un poste), ou transversales (d'un poste à un poste). La figure 125 présente les trajets des informations de l'architecture que nous observerons.

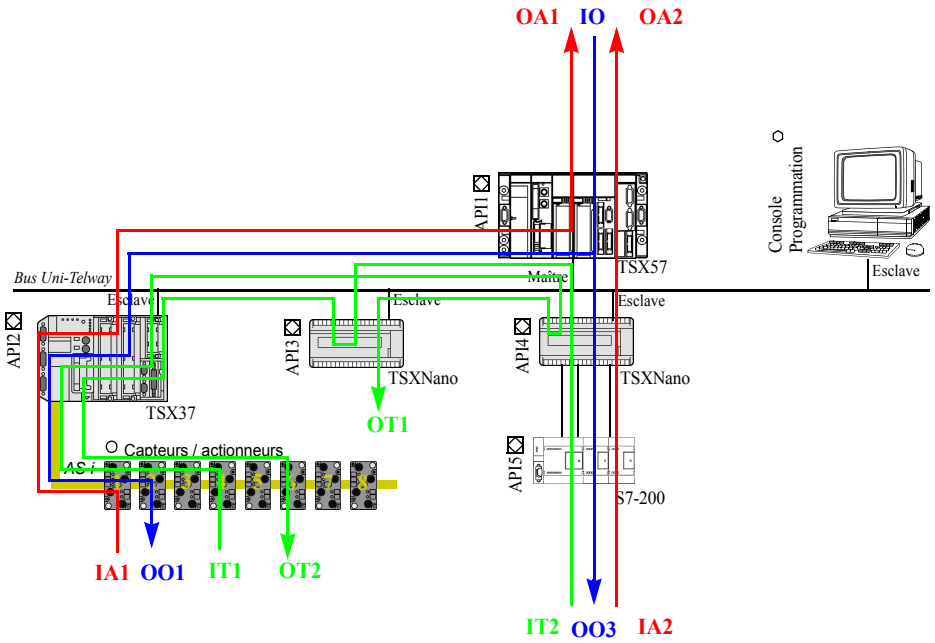


Figure 125 : Performances évaluées sur l'Architecture Opérationnelle

Les temps de transmission d'informations ascendantes correspondent à des remontées de signaux de types «alarmes». Elles remontent couramment d'un esclave vers un maître. Dans le cas de l'Architecture Fonctionnelle support, nous nous intéresserons à l'évaluation de deux alarmes qui correspondent aux plus grands trajets de remontée :

- Une alarme émise sur une entrée déportée sur AS-i qui remonte jusqu'à l'automate dédié à la supervision. Le temps de transmission associé correspond au temps entre l'occurrence d'un événement IA1 et son effet OA1.
- Une alarme émise par un commande indépendante (API5) interconnectée au reste de l'architecture par entrées/sorties croisées (avec l'API4). Cette alarme remonte jusqu'à l'automate dédié à la supervision. Le temps de transmission associé correspond au temps entre l'occurrence d'un événement IA2 et son effet OA2.

Les temps de transmission d'informations descendantes correspondent à des transmissions de type «ordres». Couramment, ces transmissions sont émises par un maître vers un esclave. Dans le cas de l'Architecture Fonctionnelle support, il y a un ordre IO qui provoque à trois effets OO1, OO2 et OO3. Nous chercherons à évaluer les ordres ayant les trajets les plus complexes, soit OO1 et OO3, ils correspondent aux inverses des trajets des alarmes OA1 et OA2 précédemment décrites. De plus nous évaluerons le temps de décalage entre les deux ordres OO1 et OO3.

Les temps de transmissions transversales correspondent à des trajets d'informations émises par un esclave et ayant un autre esclave comme destination. Dans notre Architecture Fonctionnelle support, nous nous intéresserons aux transmissions IT1 et IT2 qui ont pour effet OT1 et OT2.

5.3 Évaluation de performance de l'exemple support

Dans ce paragraphe, nous définirons dans un premier temps, les scénarios d'excitation que l'on soumettra à l'Architecture Opérationnelle pour son évaluation de performances. Dans un second temps, nous estimerons quel devra être la durée des observations par simulation ou des mesures expérimentales pour avoir des résultats acceptables. Ensuite, nous évaluerons les performances du modèle de l'Architecture Opérationnelle par simulation. Enfin, nous évaluerons les performances de l'Architecture Opérationnelle réelle par mesure expérimentale.

5.3.1 Définition du scénario d'excitation de l'architecture

Afin de ne pas étudier l'architecture que dans des cas particuliers, il est nécessaire que les périodes des scénarios d'excitation ne soient pas en corrélation avec les temps intrinsèques des équipements matériels ni synchrones entre elles.
Une solution consiste à définir les valeurs des périodes d'excitation comme des valeurs qui ne sont pas multiples entres elles ni multiples des temps intrinsèques aux équipements. Étant donné que les paramètres des équipements matériels peuvent évoluer selon les réglages de l'architecte, cela implique qu'il faudrait redéfinir les valeurs des périodes à chaque modification pour éviter une quelconque corrélation.
Une autre solution consiste à ne pas fixer de valeur pour les périodes d'excitation mais d'avoir des valeurs pouvant varier dans un intervalle donné selon une répartition uniforme. Cela permet d'avoir l'indépendance des signaux d'excitation vis-à-vis du comportement de l'Architecture. De plus, cette solution permet aussi de s'assurer de l'asynchronisme de deux simulations ou deux mesures expérimentales. Ainsi, dans le cas où les paramètres seront identiques, il sera possible de cumuler les résultats de plusieurs évaluations de performances. Nous optons donc pour cette solution.
De plus, si l'on veut avoir un maximum de relevés d'évaluations de performances dans un temps donné, il est nécessaire que les périodes d'excitation soient réduites, mais pour garder une cohérence avec un comportement réaliste sur une architecture il faut que la relation d'ordre de grandeur entre les différentes excitations soient respectées : sachant que les échanges de données de supervision sont des événements courants ($P_{supervision}$ = 1,15s), nous définissons donc les périodes suivantes :

- les signaux de type «alarme» sont rares : $P_{IA1} \in [10s, 20s]$ et $P_{IA2} \in [2s, 22s]$
- les signaux de type «ordre» sont peu courants : $P_{IO} \in [5s, 11s]$
- les signaux de type «transversaux» sont courant : $P_{IT1} \in [1s, 2s]$ et $P_{IT2} \in [1.2s, 3.2s]$

5.3.2 Estimation de la durée de simulation ou d'expérimentation

Afin de pouvoir estimer le temps de simulation ou de mesure que nous allons réaliser, nous avons étudié la probabilité d'avoir une mesure du délais mini de l'alarme A1 à *n* ms près. Pour cela nous nous sommes placé dans l'hypothèse selon laquelle l'architecture n'est pas en charge, que les temps de cycle sont constants et que les occurrences des informations A1 sont suffisamment éloignées pour ne pas interférer entre elles.

Chacun des équipements intervenant dans la chaîne de transmission voit son temps de "traversée" varier. On fera l'hypothèse pour cette estimation que :

- les automates auront un temps de «traversée» allant d'un à deux temps de cycle ;
- les réseaux ont leur temps de traversée allant d'un temps de transmission simple à un temps de transmission auquel on ajoutera le temps de scrutation de leurs esclaves ;
- chacune des variations de temps de «traversée» correspond à une répartition uniforme.

Sachant que l'alarme A1 traverse le réseau AS-i puis, le TSX37 puis le réseau UniTelway et enfin le TSX57, on peut estimer que la variation de A1 hors charge (que nous noterons Δ_{A1}) est égale à :

$$\Delta_{A1} = \Delta_1 + \Delta_2 + \Delta_3 + \Delta_4 \tag{25}$$

Avec Δ_1, Δ_2, Δ_3 et Δ_4 qui sont respectivement les étendues du temps de traversée du réseau AS-i, de l'API TSX Micro, du réseau Unitelway et de l'API TSX Premium. Nous cherchons à déterminer le nombre de mesures nécessaires pour obtenir une simulation avec une erreur de moins de n ms sur la valeur minimale du temps de traversée ; ce qui est équivalent à l'inverse de la probabilité d'avoir Δ_{A1} inférieur ou égal à n ms que nous noterons $P(\Delta_{A1} \le n)$. Nous ferons l'estimation que :

$$P(\Delta_{A1} \le n) = \prod_{i=1}^{4} P(\Delta_i \le n_i) \tag{26}$$

De plus comme nous avons fait l'hypothèse que les répartitions des Δ_i sont uniformes, alors on a :

$$P(\Delta_i \le n_i) = \frac{n_i}{\Delta_i} \tag{27}$$

Donc trouver le nombre de mesures à réaliser correspond à maximiser $\prod \frac{n_i}{\Delta_i}$ tel que $\sum n_i = n$

Pour résoudre ce problème nous avons utilisé le solveur disponible sous le tableur Microsoft Excel. En réalisant plusieurs résolutions avec des valeurs d'erreur différentes, nous obtenons la figure 126 :

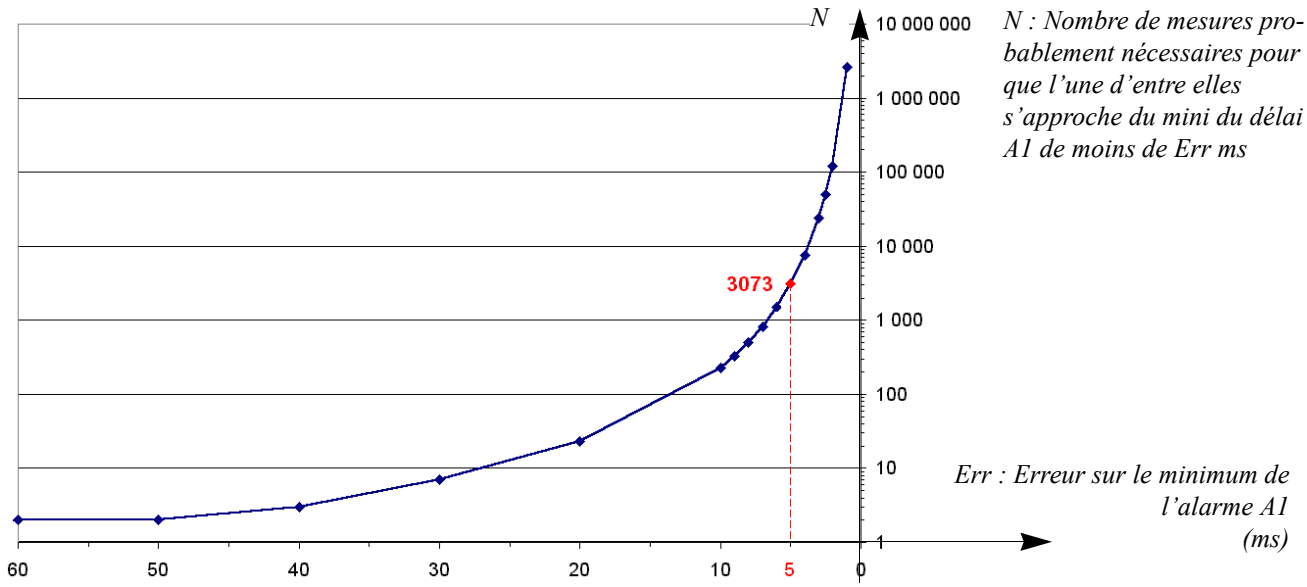


Figure 126 : Estimation du nombre de mesures en fonction de l'erreur sur le minimum de A1

Une lecture de la figure 126 nous montre que si l'on désire avoir une erreur probable de 5 ms sur la valeur minimale de A1, on peut estimer qu'il est nécessaire de réaliser 3073 mesures.
Les hypothèses qui ont permis de calculer ces valeurs ne correspondent pas exactement aux conditions de nos expérimentations. De ce fait, il nous faut compenser ces approximations par un facteur de sécurité. Aussi, nous décidons de réaliser cinq fois plus de mesures que ce que notre estimation nous avait indiqué ; soit environ 16000 mesures de l'alarme A1.

5.3.3 Évaluation des performances par simulation

Pour évaluer les performances de notre architecture support, nous avons appliqué notre démarche de modélisation de l'Architecture Opérationnelle. Afin de porter toute notre attention sur la qualité des évaluations de performances réalisées, nous ne développerons pas dans ce chapitre l'intégralité des modélisations réalisées. Ce développement est disponible en Annexe C.

A partir du modèle de simulation de notre architecture de contrôle-commande, nous effectuons une première simulation ayant un horizon H_0 de 100 000 000 unités de temps Design/CPN, (chacune d'entre elle correspondant à 0,1 milliseconde dans notre modèle), soit 10 000 secondes simulées. Cet horizon de simulation ne peut être dépassé car à partir de cette valeur certains codes associés aux transitions qui manipulent des entiers codés sur 32 bits, génèrent un dépassement de capacité. Toutefois, étant donné que les données obtenues sont différentes d'une simulation à l'autre du fait que les générateurs de jetons utilisés n'ont pas une période fixe mais une période de durée probabiliste, pour atteindre un horizon H dépassant les 100 000 000 unités de temps, il suffit de réaliser plusieurs simulations d'horizon H_0.
La simulation de notre architecture support pour un horizon H_0 a nécessité 45 minutes avec une utilisation de 100% du CPU sur un Pentium IV à 3,2 GHz avec 1Go de RAM sous système d'exploitation linux ; soit 2 700 secondes. La simulation est donc environ 3,7 fois plus rapide que le fonctionnement du système réel. Durant cette simulation, nous avons généré les excitations suivantes :

- 662 signaux d'alarme IA1 ;
- 845 signaux d'alarme IA2 ;
- 1 252 signaux d'ordre IO ;
- 6 645 signaux transversaux IT1 ;
- 4 526 signaux transversaux IT2.

Pour atteindre les 16 000 valeurs de IA1 nécessaires à une simulation ayant une erreur de l'ordre de 5 ms sur les résultats de mesure de la valeur minimum de l'alarme A1, il a donc été réalisé 18 heures et 45 minutes de simulation, ou 25 simulations d'horizon H_0.

Un programme en langage Python a été développé, pour calculer les différents temps de transmission de chacun des signaux à partir des fichiers générés par les observateurs. Le programme génère un fichier pour chacun des temps de réponse simulés.
Toutefois, les données ainsi obtenues ne permettent pas encore une analyse des performances simulées. En effet, ce ne sont que des listes de temps de transmissions. Nous avons donc réalisé une procédure automatisée permettant de récupérer les données et de représenter les répartitions pour chacune des transmissions de signaux. Cette procédure a été réalisée par le biais de programme en VBA (Visual Basic pour Application) sous l'environement Excel, ce qui nous permet d'obtenir les répartitions présentées par la figure 127.

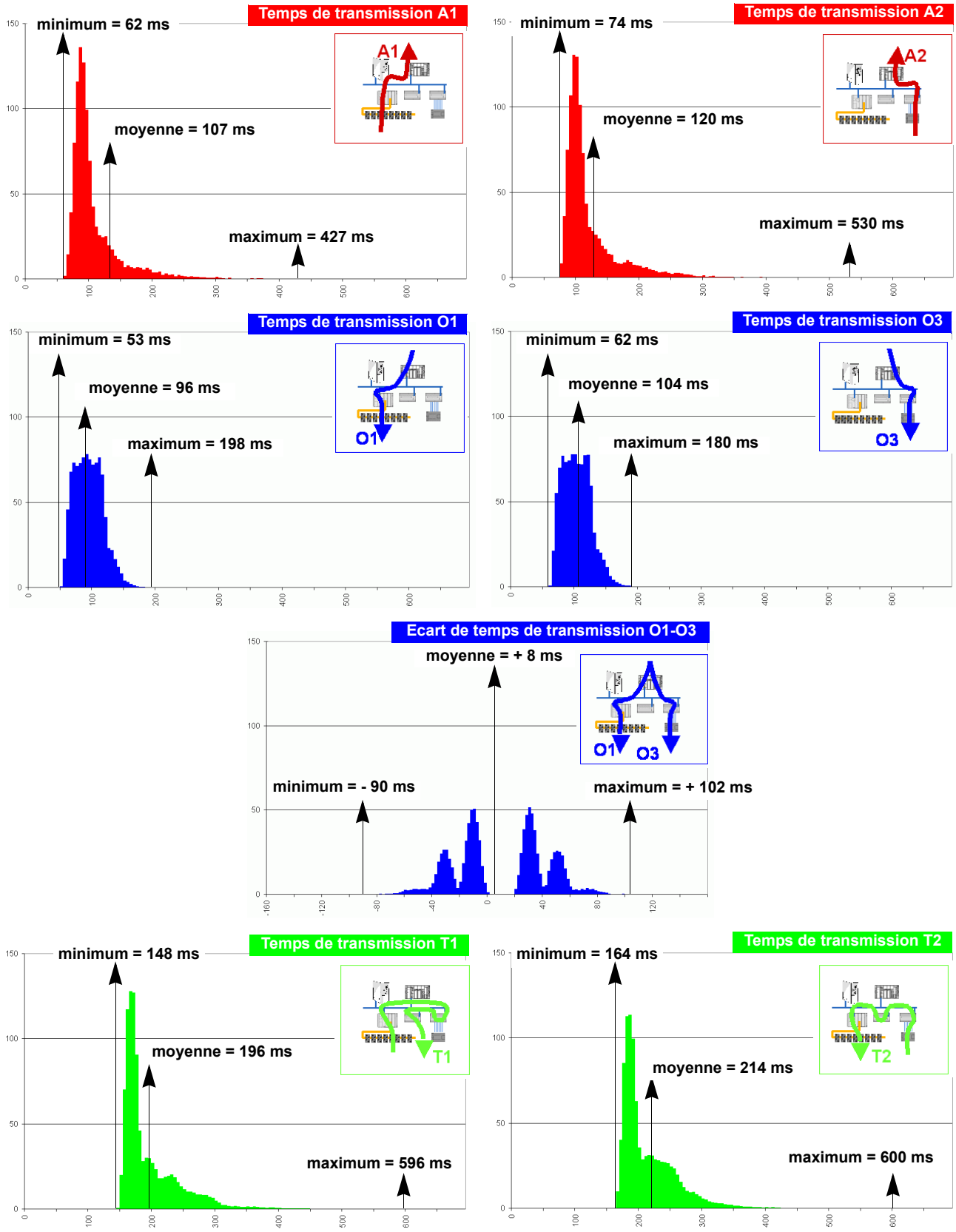


Figure 127 : Répartitions de temps de réponses obtenus par simulation

Etant donné que les données de l'analyse de performance sont issues de multiples simulations d'horizon H_0, les programmes Python ont été conçus de manière à prendre en compte les 25 lots de fichiers de données pour une même information (traitement batch).

5.3.4 Évaluation de performances par mesure expérimentale

Afin de pouvoir confronter nos résultats d'évaluation de performance obtenus par simulation, nous allons réaliser une mesure expériementale des performances sur l'architecture opérationnelle réelle à l'aide de la plateforme expérimentale PRISME. Nous expliciterons dans ce paragraphe les différentes phases qui nous ont permit d'obtenir ces mesures de performances.

5.3.4.1 Configuration de l'Architecture Opérationnelle

Dans un premier temps, il nous est nécessaire de construire et configurer l'architecture de la même manière que notre modèle de simulation, sinon toute comparaison serait vaine. Ensuite, il faut implanter dans les différents automates programmables industriels l'ensemble des tâches de notre Architecture Fonctionnelle. Pour cela nous avons défini pour l'ensemble des cinq automates :

- les codes des tâches des chaînes critiques ;
- les codes des tâches non-critiques réalisant des charges réseaux ;
- les codes des tâches non-critiques réalisant des charges CPU

Le codage a dû s'effectuer par le biais de trois plateformes de programmation différentes : STEP7 microwin pour S7-200, PL7pro pour les TSX Premium et les TSX Micro et PL707 pour le TSX Nano. De plus, il a été nécessaire de réaliser des procédures de communication variées (Maître => Esclave, Esclave => Maître, Esclave => Esclave) sur les différents automates (TSX Premium, Micro et Nano).

5.3.4.2 Configuration de la plateforme expérimentale PRISME

Une fois que l'Architecture Opérationnelle est correctement assemblée, configurée et programmée, nous devons tout comme pour nos modèles de simulation lui ajouter des générateurs de jetons et des observateurs afin de pour pouvoir mesurer ses performances.
Nous utiliserons la plateforme expérimentale PRISME qui comprend deux générateurs de signaux logiques, un analyseur logique (comportant 32 entrées) ainsi qu'un calculateur industriel (comportant 16 entrées et 8 sorties logiques) sous environement Linux. L'ensemble étant relié par une interface à une plateforme comportant l'architecture à étudier (fig. 128).

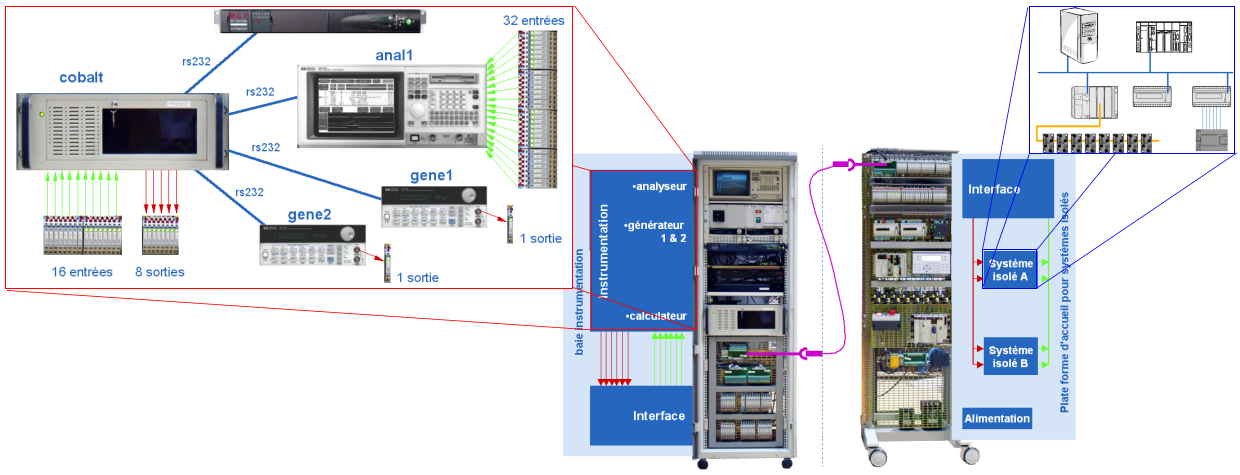


Figure 128 : Structure générale de la plateforme PRISME interfacée à l'architecture étudiée

Le calculateur assure la génération du scénario (A1, A2, O1, O2, O3, T1, T2). Nous avons dû pour cela concevoir un programme générateur de scénarii selon la stratégie d'occurrences aléatoires à distribution uniforme correspond à celle des générateurs de jetons de notre modèle de l'Architecture de Simulation. La difficulté de conception d'un tel programme a été de maîtriser le temps au dixième de milliseconde avec un OS non temps réel (GNU/Linux). Nous avons opté pour une solution où la maîtrise du temps porte sur la distribution d'une population et non pas sur chaque individu. Le programme générateur écrit en langage Python à été validé par l'analyseur de signaux logiques de la plateforme PRISME.

L'analyseur logique permet d'observer à la fois les signaux du scénario d'excitation et les effets de ce scénario. Puisque que la mémoire de l'analyseur est remplie en environ une minute d'observations, il nous est nécessaire de décharger les données régulièrement ; nous utiliserons pour cela le calculateur industriel. Un script d'enchaînement automatique des commandes a été réalisé de sorte que tant que le nombre d'acquisition est insuffisant, les opérations suivantes sont répétées :

- lancer l'acquisition sur l'analyseur logique ;
- lancer le scénario d'excitation ;
- décharger les données de l'analyseur logique ;
- pré-traiter et sauver les données sur le disque dur du calculateur.

5.3.4.3 Mise en oeuvre des expérimentations

L'analyseur de signaux logiques pose une contrainte technologique : sa capacité mémoire est remplie en environ 60 secondes de mesure. Cela se traduit par un temps moyen d'un cycle de mesure (acquisition, déchargement, traitement et sauvegarde) de 180 secondes.

Sachant que la durée moyenne entre deux occurences de l'alarme A1 est de 15 secondes, il y aura donc trois à quatre excitations IA1 observées pendant les 60 secondes de mesure. On considèrera le cas le plus défavorable c'est-à-dire 3 observations par cycle de mesure de 180 secondes.
Étant donné que notre objectif est d'atteindre les 16 000 occurences de l'alarme A1, on calcule la durée prévisionnelle qui est égale à 960 000 secondes, c'est à dire 11 jours 2 heures et 40 minutes.

Lors de la mise en service de l'architecture, il est apparu que l'un des API (le TSX37) subissait un stress de communication sensiblement trop important pour son coupleur de communication intégré à sa CPU. Cela a engendré des erreurs de communications qui nous ont obligées à réaliser un programme de détection et d'élimination automatique des mesures polluées.
Au final, la durée effective de la campagne de mesure aura été de 3 semaines. Durant ce laps de temps, auront été observées les excitations suivantes :

- 16 654 signaux d'alarme IA1,
- 21 200 signaux d'alarme IA2,
- 31 383 signaux d'ordre IO,
- 166 549 signaux transversaux IT1,
- 113 945 signaux transversaux IT2.

5.3.4.4 Traitement et mise en forme des données mesurées

Les données brutes obtenues par le biais de l'analyseur sont des chronogrammes sous la forme de deux listes de valeurs :

- dates = [1.23, 1.75, 1.92, 2.09, 2.31, 2.62]
- values = [2, 3, 1, 5, 4, 0]

Ces données brutes peuvent être représentées graphiquement comme l'indique la figure 129:

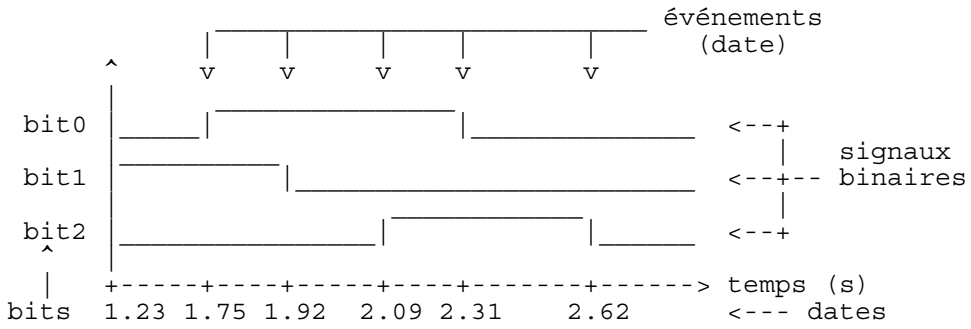


Figure 129 : Représentation graphique des mesures brutes

Afin, de déterminer les performances que l'on souhaite évaluer, il nous faut extraire les valeurs des temps de transmission à partir de ces histogrammes. Par exemple, pour l'alarme A1, il est nécessaire d'observer l'effet (bit 30 de l'analyseur logique) de la cause (bit 6 de l'analyseur logique) (fig. 130).

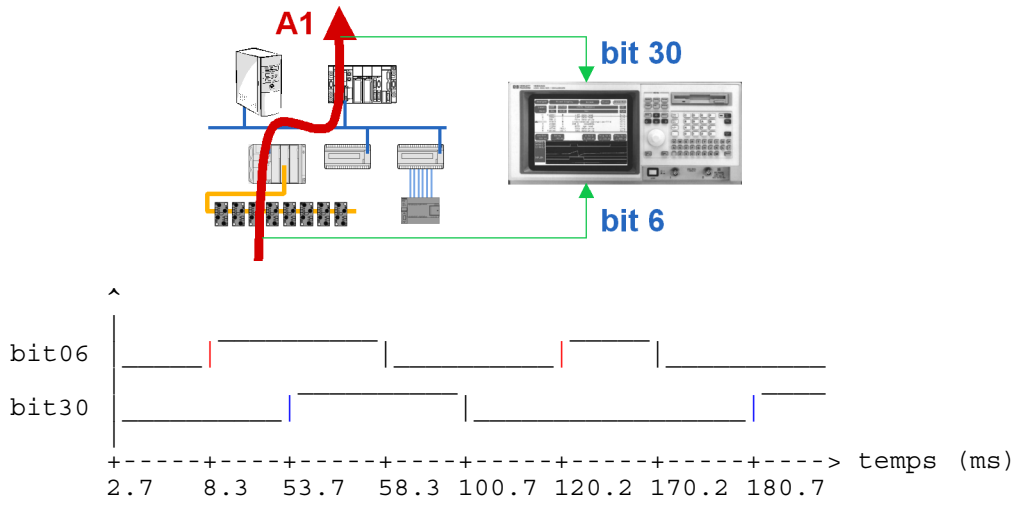


Figure 130 : Observation de A1 sur l'analyseur logique

Il faut donc, pour tous les fronts montants du bit 6 (en rouge sur la figure 130), identifier le délai au bout duquel un front montant se produit sur le bit 30 (en bleu sur la figure 130). Un programme en langage Python permet d'extraire chaque délais sous la forme d'une liste : delai_A1 = [45.4, 68.5, …]. A partir de ces données, il est nécessaire, tout comme pour les données obtenues par simulation, de représenter les répartition des performances mesurées.

Les mesures nous ont donc permis d'obtenir les répartitions suivantes :

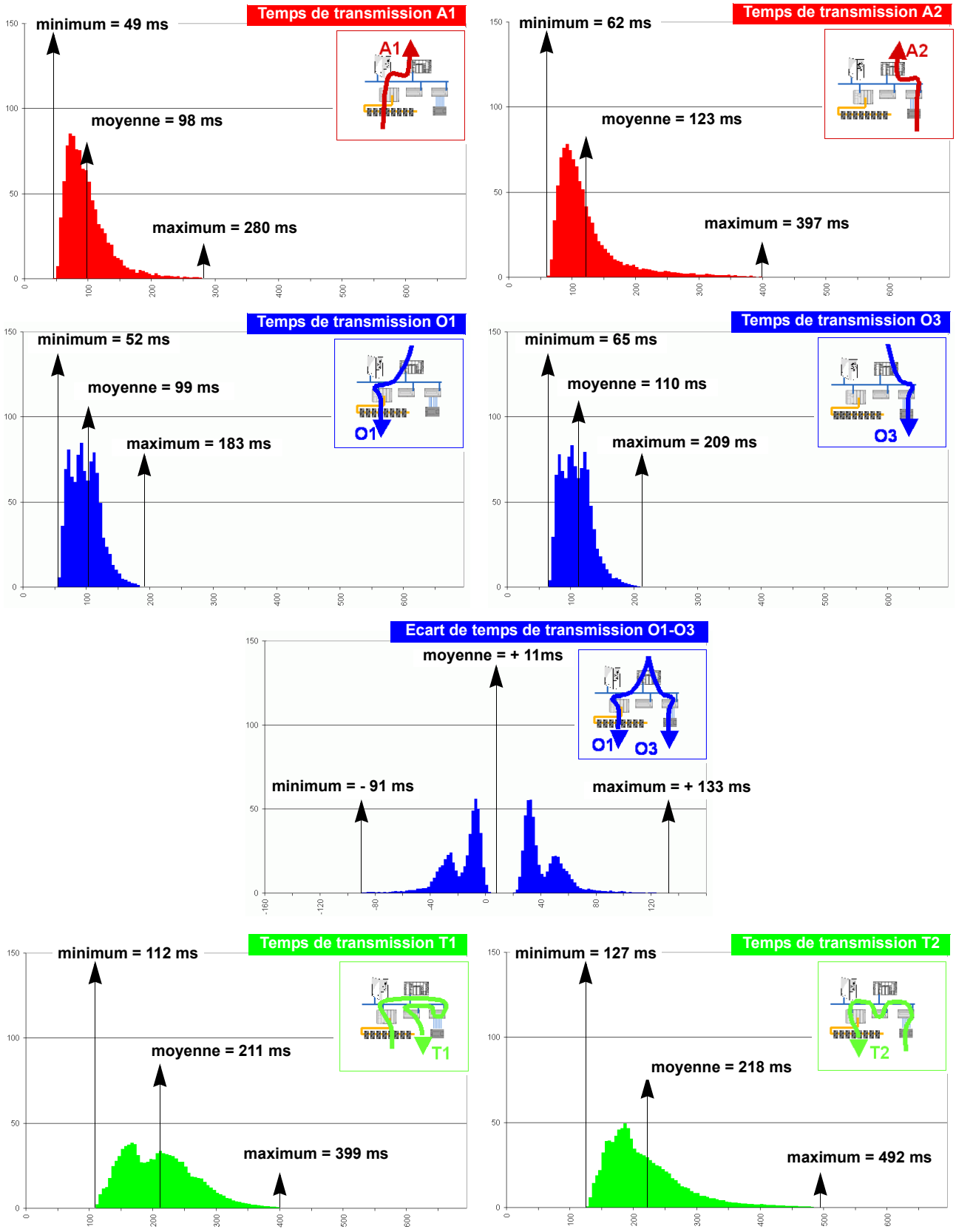


Figure 131 : Répartitions de temps de réponses obtenus par mesure expérimentale

5.4 Étude de la qualité des résultats de simulation

Comme nous venons de le montrer dans le paragraphe 5.3, notre démarche est donc praticable dans le cadre d'étude de performances sur une architecture complexe. De plus comme nous venons de le notifier, le processus de simulation est 3,7 fois plus rapide que le fonctionnement de l'architecture réelle équivalente. Nous allons maintenant étudier dans les paragraphes suivants, la qualité de nos résultats de simulations au travers de :

- la convergence de l'erreur sur les performances par rapport au nombre d'observations,
- la sensibilité de l'erreur des performances en fonction des erreurs associées aux paramètres de nos modèles,
- la précision de nos performances simulées (pour un nombre suffisant d'observations) par rapport aux performances mesurées sur la plateforme expérimentale.

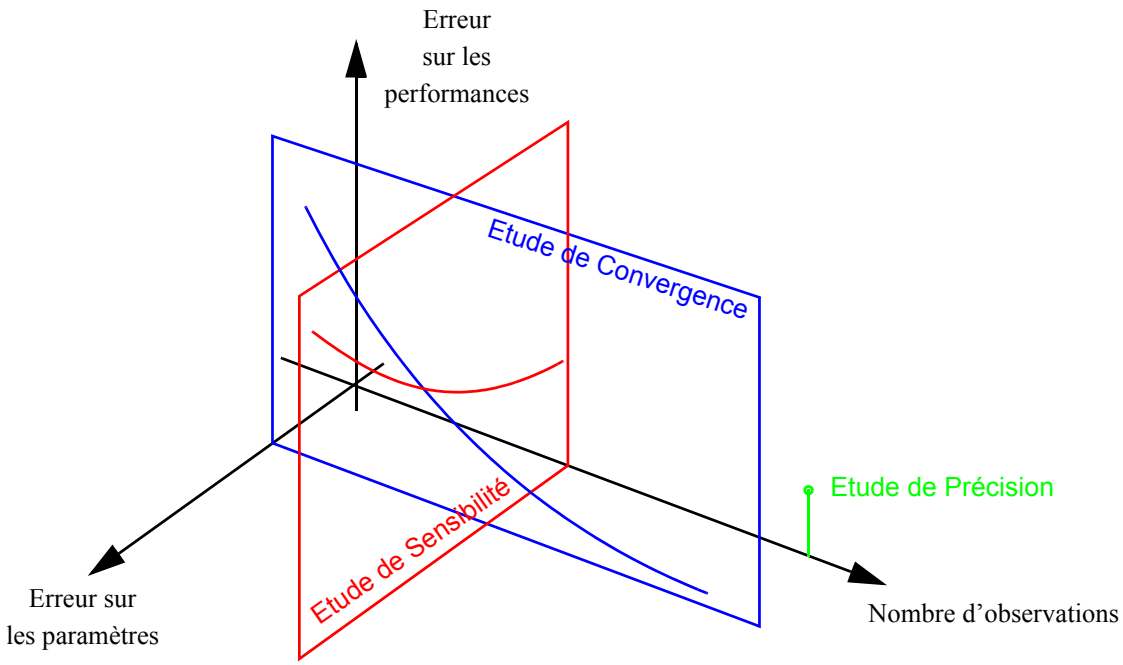


Figure 132 : Etude de la qualité de notre démarche

5.4.1 Étude de la convergence du modèle de simulation

Suivant la phase du cycle de vie de l'architecture dans laquelle on se trouve, l'architecte a plus ou moins de temps à accorder à la simulation :

- durant la phase de pré-conception, les résultats devront être obtenus très rapidement mais pourront n'être qu'approximatifs ;
- pendant la phase de conception détaillée, le temps à accorder aux simulations pourra être plus important mais en contre-partie, des résultats précis sont attendus ;
- pendant la phase de rétro-fit ou de reconfiguration de l'installation, l'erreur devra être la plus réduite possible mais en échange la durée de simulation pourra être importante.

Il est donc nécessaire d'estimer l'erreur des performances évaluées en fonction de la durée des simulations.

5.4.1.1 Estimations préliminaires

Les performances sur lesquelles nous allons porter notre étude, seront les valeurs moyennes, minimales et maximales pour chacun des types de temps de transmission : un ordre (O1), une alarme (A1), une transmission d'information (T1).

La valeur moyenne d'un échantillon de valeurs étant la tendance générale de celles-ci, cette valeur devrait converger assez rapidement. Par contre, en réalisant une rapide analyse d'une courbe de répartition (par exemple T1), on peut remarquer que pour une erreur donnée : la population équivalent à la zone autour de la valeur minimale est faible alors que celle autour de la valeur maximale est extrememement faible.

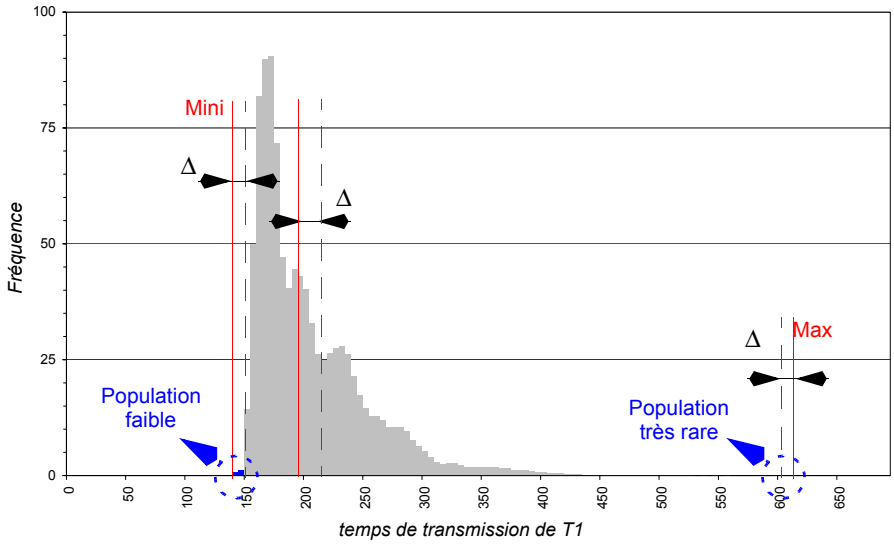


Figure 133 : répartition de la population autour des extrémums

On s'attend donc à avoir une convergence lente pour l'obtention de la valeur minimale et très lente pour l'obtention de la valeur maximale. En effet, il faudra attendre des conjonctions particulières d'événements au niveau des processus concourants de l'architecture pour avoir des valeurs proches des extrémums.

5.4.1.2 Méthodologie d'étude de la convergence

Etant donné que nous devons évaluer la convergence de nos modèles, il va être nécessaire de réaliser une campagne de simulations où l'on étudiera la répartition de la moyenne et des extrema en faisant varier le nombre d'observations de données durant la simulation. Pour cela, on définit :

- $SIM_i(x)$ comme étant une simulation dont l'horizon permet d'obtenir 10^i observations de x ; une simulation permet d'obtenir une valeur moyenne et une valeur pour chaque extremum ;
- $SERIE_i(x)$ comme étant une série de 10 000 $SIM_i(x)$, une série permet d'obtenir la répartion des valeurs moyennes et des extremums pour un nombre d'observations donné de x ;
- $CAMPAGNE(x)$ comme étant l'ensemble des $SERIE_i(x)$ avec i variant de 1,6 à 5 (de 1,6 à 6 pour l'étude de IT1, car il y a environ 10 fois plus d'occurences que IA1 dans une même période) par incréments de 0,2 ; elle permet d'obtenir les évolutions de la variation des valeurs moyennes et des extremums en fonction du nombre d'observations de x.

D'après les résultats de simulation du paragraphe 5.3.3 page 127, on observe 662 IA1 dans une période de 45 minutes ; soit environ 4 secondes pour l'observation d'une alarme A1. on a donc :

$$duree(SIM_i(\mathrm{A1})) = 10^i \times 4 \text{ secondes} \quad (28)$$

$$duree(SERIE_i(\mathrm{A1})) = 10000 \times duree(SIM_i(\mathrm{A1})) \quad (29)$$

$$duree(CAMPAGNE(\mathrm{A1})) = \sum_{\substack{i=1,6 \\ inc = 0,2}}^{i=5} duree(SERIE_i(\mathrm{A1})) \quad (30)$$

Cela nous amène à un problème de taille : la durée nécessaire pour réaliser une CAMPAGNE est de 1 083 613 286 secondes, soit plus de 34 années ! Même si l'on ne réalise que la dernière série, on aura alors une valeur de la *duree(SERIE$_5$(A1))* dépassant les 12 années. Cela est dû au fait que l'on veut avoir une répartition basée sur une population de 10 000 valeurs de moyenne ou d'extrema.

Toutefois, en réalisant une simulation SIM_n avec $n > 5$, on obtiendra une population de taille 10^n. Si on réalise des échantillonages de taille 10^k dans la population de taille 10^n on obtiendra alors l'équivalent de simulations SIM_k avec $k < n$. Ainsi, pour toute valeur de i, il n'est plus nécessaire de réaliser 10 000 fois une simulation SIM_i mais de réaliser 10 000 échantillonages de taille 10^i sur la population de taille 10^n avec $i<n$.
Afin de construire la population de taille 10^n, nous devons calculer son horizon de simulation. Étant donné que l'on désire avoir plus de 10^5 occurences de IA1 et qu'il y en a eu 662 pour un horizon de simulation de 100 000 000 unités de temps DesignCPN, il nous en faudra $\frac{10^5}{662}$ fois plus soit environ 153 fois plus.
La simulation SIM_n, dont l'horizon de simulation équivaut à 15 300 000 000 unités de temps Design CPN soit l'équivalent de 17 jours et 17 heures de fonctionnement du système, a utilisé l'équivalent de 4 jours et 16 heures de charge CPU d'un Pentium IV à 3,2 GHz avec 1Go de mémoire RAM. Cette simulation a permi d'observer l'ensemble des signaux suivants :

- 101 810 signaux d'alarme IA1,
- 127 722 signaux d'alarme IA2,
- 190 938 signaux d'ordre IO,
- 1 018 821 signaux transversaux IT1,
- 694 323 signaux transversaux IT2.

À partir de cette ensemble important de données, nous devons générer les *SERIE$_i$(x)*. Pour cela, nous avons dû réaliser un programme en langage Python réalisant 10 000 échantillonages de taille 10^i dans la population de taille 10^n. Puis, pour chaque valeur de i, le programme calcule la moyenne et les extremums de chacun des 10 000 échantillons *SIM$_i$(x)*, que l'on notera :

- *Min$_i$(x)* = valeur minimale de la population de taille 10^i pour les informations x,
- *Moy$_i$(x)* = valeur moyenne de la population de taille 10^i pour les informations x,
- *Max$_i$(x)* = valeur maximale de la population de taille 10^i pour les informations x.

Enfin, le programme calcule pour chaque valeur de i, les répartions des 10 000 valeurs de *Min$_i$(x)*, *Moy$_i$(x)* et *Max$_i$(x)*. La durée globale d'exécution de ce programme a été d'environ 72h sur un Pentium III 500 MHz avec 500 Mo de RAM.

5.4.1.3 Analyse de la convergence des valeurs moyennes des temps de transmission

D'après les théories statistiques, la répartition des valeurs moyennes d'un ensemble d'échantillon est une gaussienne centrée autour de la valeur moyenne de la population globale. Si on observe la répartition des valeurs expérimentales moyennes de T1, on observe effectivement une telle répartition (fig. 134).

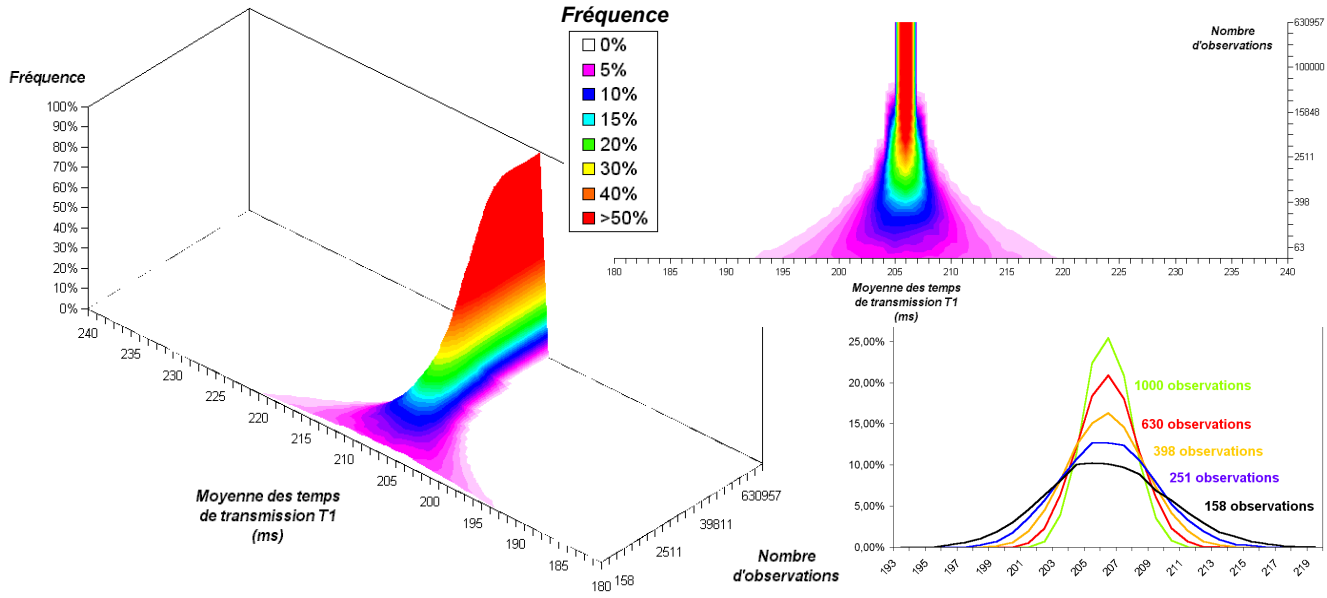


Figure 134 : Répartitions de valeurs moyennes de T1 en fonction du nombre d'observations

Les répartitions des valeurs moyennes de O1 et de A1 étant similaires à celle de T1, nous ne présenteront pas ces courbes. Par contre lorsque l'on réalise la mesure de l'écart absolu (égal à $|Moy_n(x) - Moy_i(x)|$ avec *Nombre d'observation* = 10^i), on remarque que la convergence des trois mesures n'est pas équivalente (fig. 135).

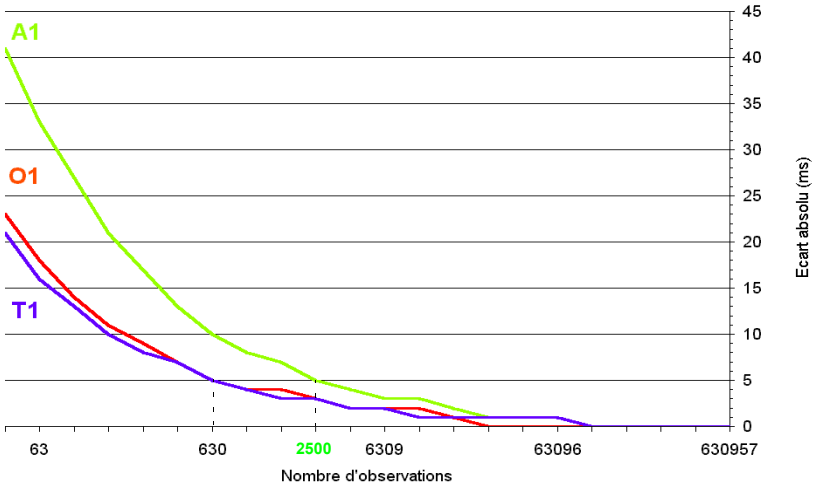


Figure 135 : Ecarts absolus sur la valeur moyenne des signaux O1, A1 et T1

Si l'on accepte une erreur de l'ordre de 5 ms, la valeur moyenne de O1 et T1 converge très rapidement (environ 630 observations), alors qu'il en faut presque 4 fois plus (environ 2500 observations) pour que

la valeur moyenne de A1 converge. Il faudrait donc réaliser une simulation de 45 minutes pour avoir une convergence correcte de O1 et de T1 alors qu'il en faudrait près de 180 minutes de simulation pour avoir une bonne convergence de A1 (à 5 ms près).

5.4.1.4 Analyse de la convergence des valeurs minimum des temps de transmission

Pour le cas des valeurs minimales, nous n'avons pas de répartition théorique connue. En observant la répartition des valeurs minimales du signal de transmission T1 en fonction de la taille des échantillons, on a (fig. 136):

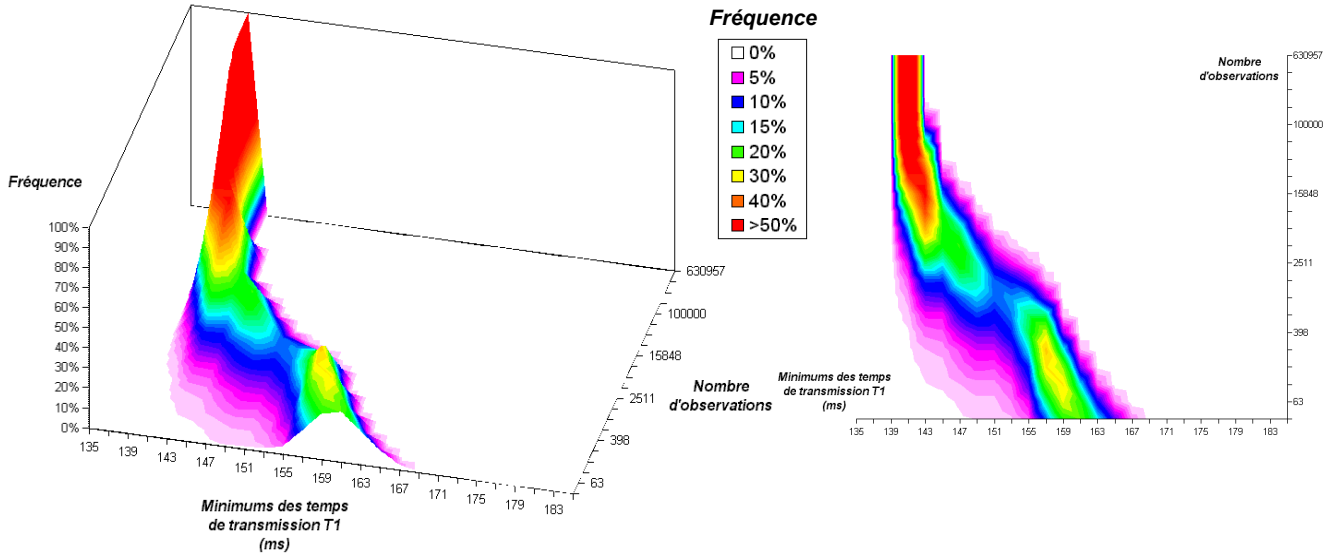


Figure 136 : Répartition de valeurs minimales de T1 en fonction du nombre d'observations

La répartition n'a pas d'allure gaussienne et ne ressemble pas à une allure connue. Toutefois elle converge autour de la valeur minimale. Afin d'avoir une analyse plus pertinente des répartitions des valeurs moyennes de O1, A1 et T1 on réalise la mesure de l'écart absolu (égal à $|Min_n(x) - Min_i(x)|$ avec *Nombre d'observation* = 10^i) (fig. 135).

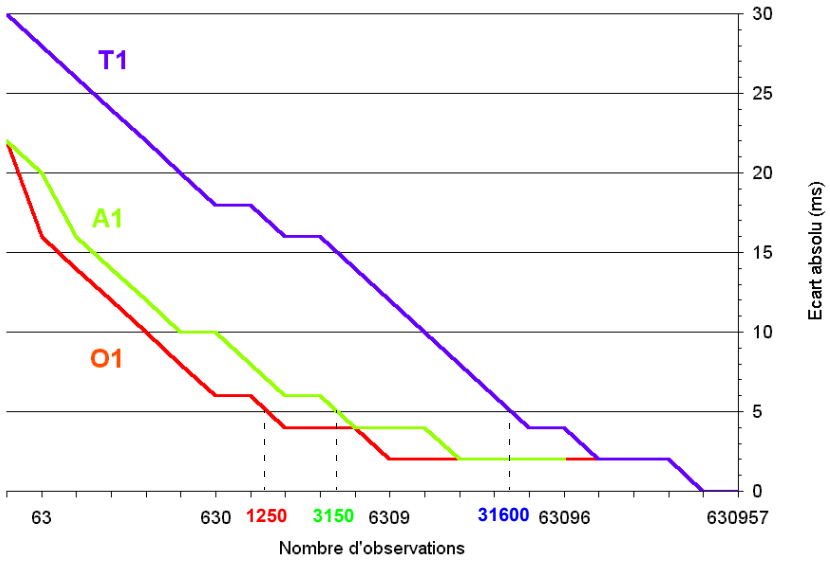


Figure 137 : Écarts absolus sur la valeur minimale des signaux O1, A1 et T1

Si l'on considère une erreur de 5 ms, il s'avère que la convergence la plus rapide est celle de la valeur minimale de O1 (environ 1250 mesures) suivit celle de A1 (3150 mesures) et enfin celle de T1 (31600 mesures). Cela correspondrait à un temps de simulation de 45 minutes pour O1 et 3 heures, 45 minutes pour A1 et T1.

5.4.1.5 Analyse de la convergence des valeurs maximums des temps de transmission

Tout comme pour le cas des valeurs minimales, les valeurs maximales n'ont pas de répartition théorique connue. En observant la répartition des valeurs maximales du signal de transmission T1 en fonction de la taille des échantillons, on a (fig. 136):

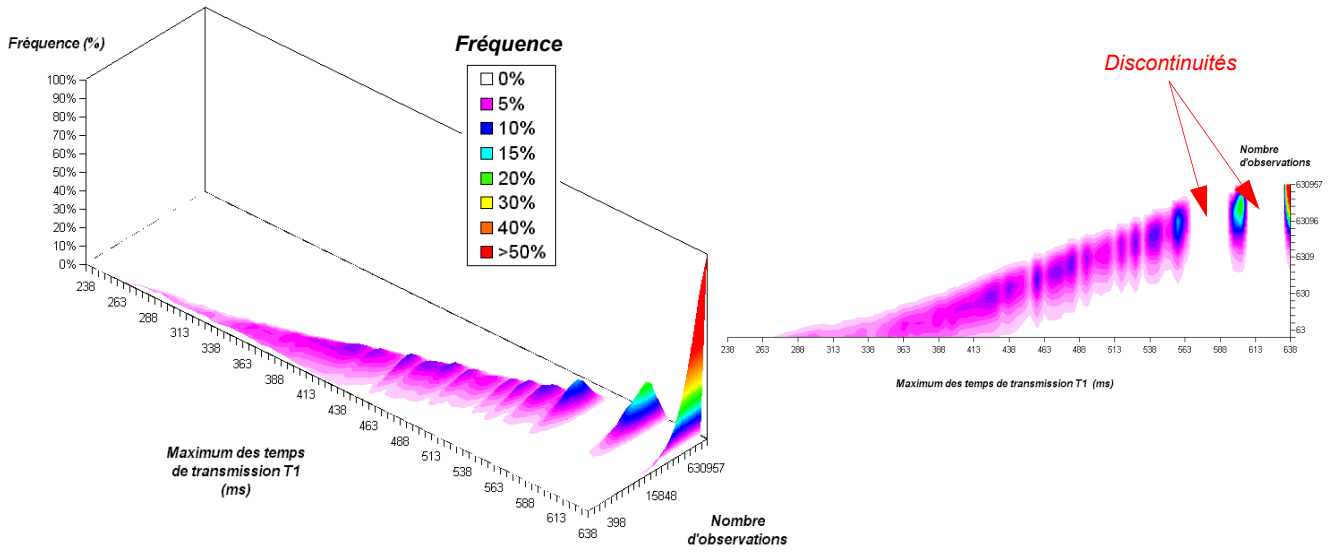


Figure 138 : Répartition de valeurs maximale de T1 en fonction du nombre de mesures

La répartition n'a pas d'allure gaussienne et se trouve être discontinue. On a le même type de répartition non-continue pour les signaux O1 et A1.

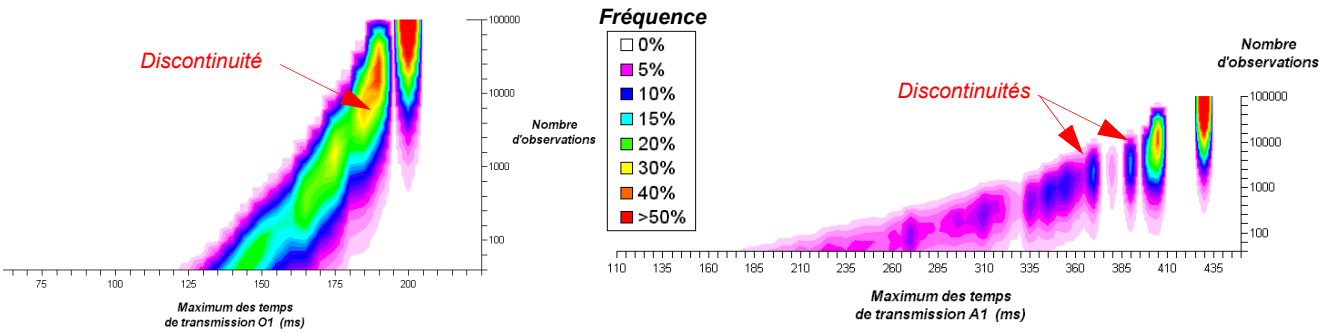


Figure 139 : Répartition de valeurs maximales de O1 et A1 en fonction du nombre d'observations

Étant donné que l'étude réalisée porte sur les performances d'architectures "en charge", cela implique que la borne maximale n'est pas connue et qu'elle correspond à la conjonction de plusieurs phénomènes rares qui ne suivent pas des lois probabilistes standard.
Aussi pour observer la borne maximale, il sera nécessaire de réaliser de nombreuses observations. Toutefois, avant d'observer une occurence de la valeur maximale, correspondant à la conjonction de l'ensemble des phénomènes, on observera dans un premier temps les conjonctions d'une partie de ces

phénomènes. Or plus le nombre de phénomènes intervenants dans une conjonction est important plus la conjonction est rare. Donc plus grand sera le nombre d'observations plus les chances de relever une conjonction sont importantes. Les valeurs des conjonctions les plus courantes étant proches, les répartitions autour de ces valeurs se chevauchent et donc donnent des répartitions continues. Alors que dans le cas des conjonctions rares, les valeurs sont éloignées, ce qui donne des discontinuités sur les figure 138 et figure 139.

On remarque en calculant la probabilité d'observation des conjonctions en fonction du nombre de d'observations (fig. 140) que pour avoir la certitude d'avoir relevé une occurence de la valeur maximale, il faut un très grand nombre d'observations. Sinon, on a une très grande probabilité d'observer des conjonctions "partielles" à la place de la borne maximale.

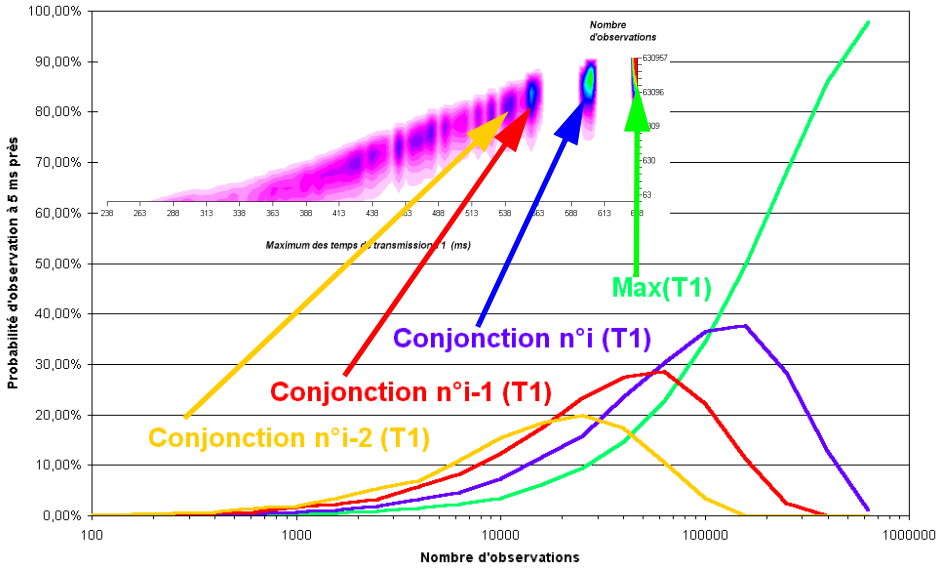
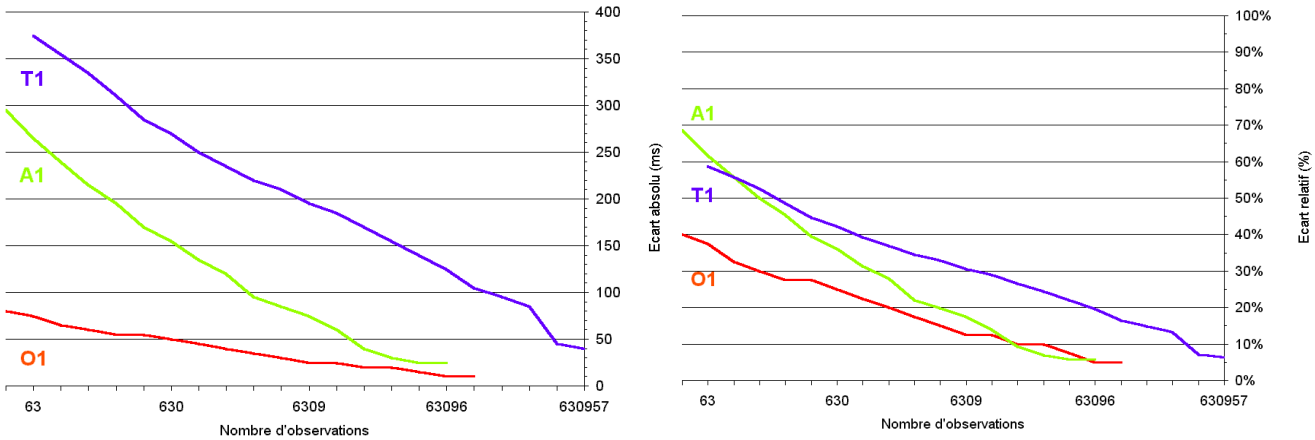


Figure 140 : Probabilité d'observation de conjonctions pour la valeur maximale de T1

Afin d'avoir une analyse pertinente de la répartition des valeurs maximales de O1, A1 et T1, on réalise le calcul des écarts absolus (égal à $|Max_n(x) - Max_i(x)|$ avec *Nombre d'observation* = 10^i) (fig. 135a) et relatif (égal à $\frac{|Max_n(x) - Max_i(x)|}{Max_n(x)}$ avec *Nombre d'observation* = 10^i) (fig. 135b).



-a- Ecart absolu sur la valeur maximale

-b- Ecart relatif sur la valeur maximale

Figure 141 : Écarts sur la valeur maximale des signaux O1, A1 et T1

Si on analyse ces figures, on remarque que d'après les écarts absolus, la valeur maximale de O1 converge plus rapidement que celle de A1 (qui elle même est plus rapide que celle de T1). Toutefois, contrairement aux convergences des valeurs moyennes et minimales, il est nécessaire d'avoir un très grand nombre d'observations pour obtenir une valeur à 10% près : O1 nécessiterait environ 9 heures de simulation, A1 en consommerait 18 heures, alors que T1 aurait besoin de 36 heures.

La valeur maximale est, comme nous l'avons définis précédement, observée comme étant la conjonction de plusieurs phénomènes rares qui sont liés aux équipements matériels intervenants dans la chaîne critique. Donc plus le nombre d'équipements de cette chaîne sera important, plus rare sera la conjonction permettant de relever la borne maximale. Cela explique pourquoi la valeur maximale de T1 converve moins rapidement que O1 et A1. De plus sur le réseaux UniTelway, O1 est un ordre envoyé par l'API maître alors que A1 est une alarme envoyé par un API esclave ; le maître prenant la priorité sur les messages en attentes, sa valeur maximale est soumise à moins que perturbations que celle de A1. Ce qui explique que la convergence de la valeur maximale de O1 est plus rapide que celle de A1.

5.4.1.6 Conclusion sur la convergence des modèles de simulation

Nous avons montré que la convergence de la valeur moyenne est très rapide et qu'en l'espace d'environ une heure de simulation cette valeur était atteinte à 5 ms près.

Pour ce qui est de la convergence des valeurs minimales, elle s'avère assez rapide pour les transmissions de signaux "simples" que sont les ordres (O1, O2, O3) et les alarmes (A1, A2) et est un peu plus lente pour les signaux complexes que sont les transmissions d'informations (T1, T2). Le temps à accorder à la simulation pour atteindre une valeurs satisfaisante à 5 ms près serait de quelques heures.

Par contre la convergence de la valeur maximale est très lente. Ainsi pour une erreur de 25 ms sur la valeur maximale, il serait nécessaire de réaliser 10 000 mesures pour O1, 63 000 pour A1 alors qu'en 630 000 mesures T1 aurait encore une erreur de 40 ms … Donc si on espère avoir la valeur maximale à 5 ms près, le nombre de simulation devra être très important.
Étant donné que l'on ne connait pas la valeur de la borne maximale, on ne peut dire si la valeur relevée correspond à une conjonction "partielle" ou à la valeur maximale. Et alors que l'on peut penser que la valeur maximale a convergé, l'apparition d'une nouvelle conjonction peut remettre tout en cause …
Toutefois, la valeur obtenue correspondra à la valeur maximale que l'on devrait observer dans le même intervalle de temps de fonctionnement du système réel. Ainsi si l'on observe par exemple 100 000 valeurs d'un signal, on pourra considérer qu'il y aura moins d'une chance sur 100 000 que l'on ait une valeur plus grande que la valeur maximale observée.

Une simulation de 45 minutes nous permet donc d'obtenir les erreurs sur les valeurs des performances :

Tableau 8 : Erreurs pour une simulation de 45 minutes

Signal	Erreur sur la valeur moyenne	Erreur sur la valeur minimale	Erreur sur la valeur maximale
O1	4 ms	5 ms	40 ms (22%)
A1	10 ms	10 ms	155 ms (36%)
T1	2 ms	12 ms	195 ms (30%)

Par ailleurs avec une simulation de 112 heures, on obtient les erreurs suivantes sur les valeurs des performances :

Tableau 9 : Erreurs pour 150 simulations de 45 minutes

Signal	Erreur sur la valeur moyenne	Erreur sur la valeur minimale	Erreur sur la valeur maximale
O1	< 1 ms	2 ms	10 ms (5%)
A1	1 ms	2 ms	25 ms (6%)
T1	< 1 ms	< 1 ms	40 ms (6,3%)

La convergence des moyennes, minimums et maximums des performances temporelles d'une architecture de contrôle-commande répond aux besoins d'un architecte pendant le cycle de vie d'une architecture : elle permet d'avoir des résultats avec un ordre de grandeur acceptable d'erreur avec une simulation assez courte, et elle permet d'avoir des résultats avec une erreur faible lorsque l'on réalise une simulation sur une longue période.

5.4.2 Étude de la sensibilité du modèle aux paramètres de l'architecture

L'architecte désirant évaluer les performances de ses architectures tout au long du cycle de vie de celle-ci, est nécessairement confronté aux incertitudes dans la valeur des paramétres des équipements. Ces imprécisions seront importantes en début de cycle de vie et se réduiront dans les phases d'évolution de l'architecture.

Il est donc nécessaire de connaître la sensibilité des performances observées par rapport aux erreurs portant sur les paramètres des équipements de l'architecture. En phase de conception de l'architecture, les paramètres que l'architecte doit estimer sont :

- les temps de cycle des équipements de traitement qui ne seront réellement connus qu'après implantation des programmes ;
- les tailles des informations devant être échangées sur les différents médias ;
- le nombre d'esclaves pour certains types de réseaux, le nombre d'entrées/sorties déportées, qui ne sera connu qu'après la définition totale des équipements.

Pour notre exemple support, nous avons choisi de faire varier :

- le temps de cycle de l'ensemble des 5 API ;
- la taille des informations échangées par la tâche de supervision ;
- le nombre d'esclaves AS-i connectés à l'automate TSX Micro.

Puisqu'il nous faut effectuer un grand nombre de simulations pour évaluer les influences des différentes combinaisons, nous allons porter notre attention sur les variations de la valeur moyenne et de la valeur minimale qui d'après l'étude du paragraphe précédent ont une convergence assez rapide. Nous observerons bien sûr, chacun des types de signaux : un ordre O3, une alarme A1, et un transmission d'information T1.

Notre première approche a consisté à faire varier l'ensemble des paramètres de la même façon en les sous-évaluant ou les sur-évaluant dans la même proportion. On notera :

- *Min(x)* et *Moy(x)* respectivement la valeur minimale et la valeur moyenne de *x* sans variation sur les paramètres ;
- $Min_p(x)$ et $Moy_p(x)$ respectivement la valeur minimale et la valeur moyenne de *x* avec une variation de *p*% sur les l'ensemble des paramètres ;
- $\Delta Min_p(x)$ et $\Delta Moy_p(x)$ respectivement les variations relatives des valeurs minimale et moyenne de *x* avec une variation de *p*% sur l'ensemble des paramètres par rapport aux valeurs sans variations sur les paramètres :

$$\Delta Min_p(x) = \frac{|Min(x) - Min_p(x)|}{p \times Min(x)} \quad (31)$$

$$\Delta Moy_p(x) = \frac{|Moy(x) - Moy_p(x)|}{p \times Moy(x)} \quad (32)$$

On obtient les variations relatives sur les performances suivantes :

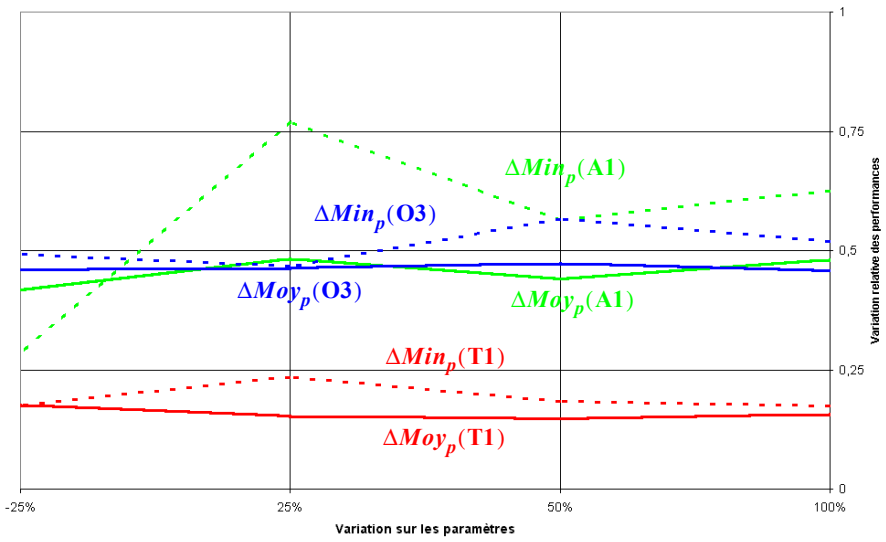


Figure 142 : Variations des performances suivant une variation de l'ensembre des paramètres

En analysant la figure 142, on remarque que l'ensemble des variations relatives sont inférieures à 1, ce qui signifie que l'ordre de grandeur des variations sur les performances sera toujours inférieur à l'ordre de grandeur des variations sur l'ensemble des paramètres.

Toutefois les paramètres sont rarement toujours tous sujets aux mêmes erreurs (sous-évalués ou sur-évalués du même pourcentage) en même temps, nous avons donc réalisé quelques essais en faisant varier les paramètres de +25% ou -25%. Nous avons réalisé les essais suivants :

Tableau 10 : Essais de sensibilités avec variations différentes selon les équiepements

Essai	Varation du temps de cycle					variation du nombre d'esclaves AS-i	varation du nombre de mots échangé
	API1	API2	API3	API4	API5		
1	+25%	-25%	+25%	+25%	+25%	+25%	-25%
2	+25%	+25%	-25%	-25%	+25%	-25%	+25%
3	+25%	+25%	+25%	+25%	+25%	-25%	+25%
4	+25%	+25%	25%	-25%	-25%	-25%	+25%
5	-25%	+25%	+25%	+25%	-25%	+25%	+25%
6	-25%	-25%	-25%	+25%	-25%	-25%	-25%
7	-25%	-25%	-25%	-25%	-25%	-25%	+25%
8	-25%	-25%	-25%	-25%	+25%	-25%	+25%

Nous obtenons alors :

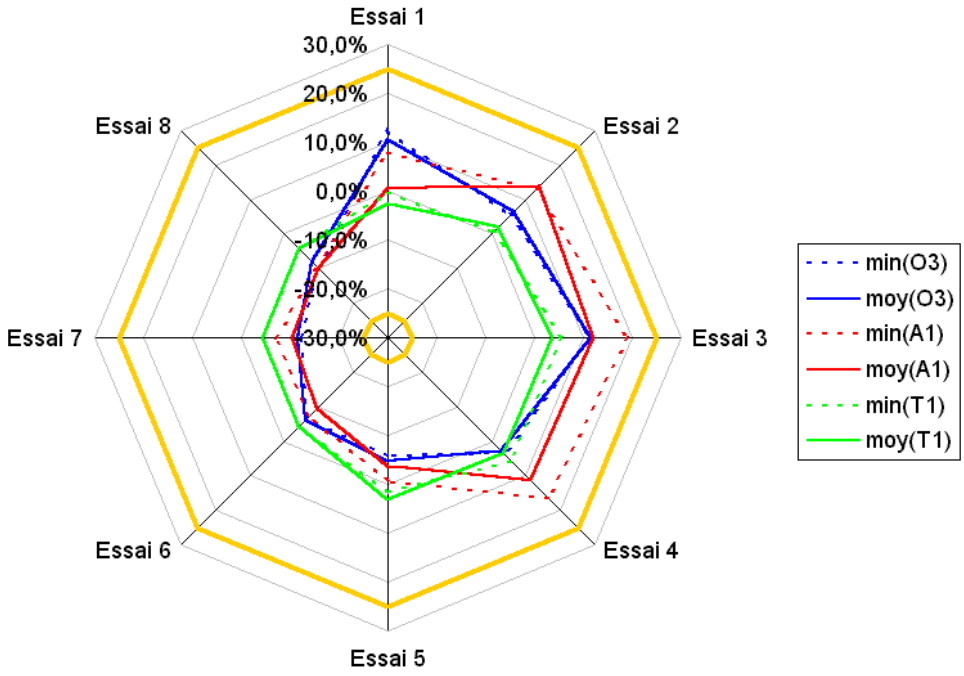


Figure 143 : Variations des performances suivant une variation associé à chacun des paramètres

On a placé en traits forts de couleur jaune les limites correspondant à des variations de -25% et + 25%. On remarque que l'ensemble des performances dont les paramètres ont été entachées d'erreurs à -25% ou +25% sont dans cet intervalle.

On peut donc en conclure que la sensibilité de notre modèle aux variations de paramètres est faible car l'erreur sur les performances observées serait inférieure ou égale à l'erreur que l'architecte ferait sur la valeur des paramètres.

5.4.3 Étude de la précision du modèle

Nos modèles ont une bonne convergence de leurs performances et une faible sensibilité aux variations de paramètres, mais qu'en est-il de leur précision ? En effet, l'architecte a besoin d'avoir des résultats réalistes pour ses évaluations.

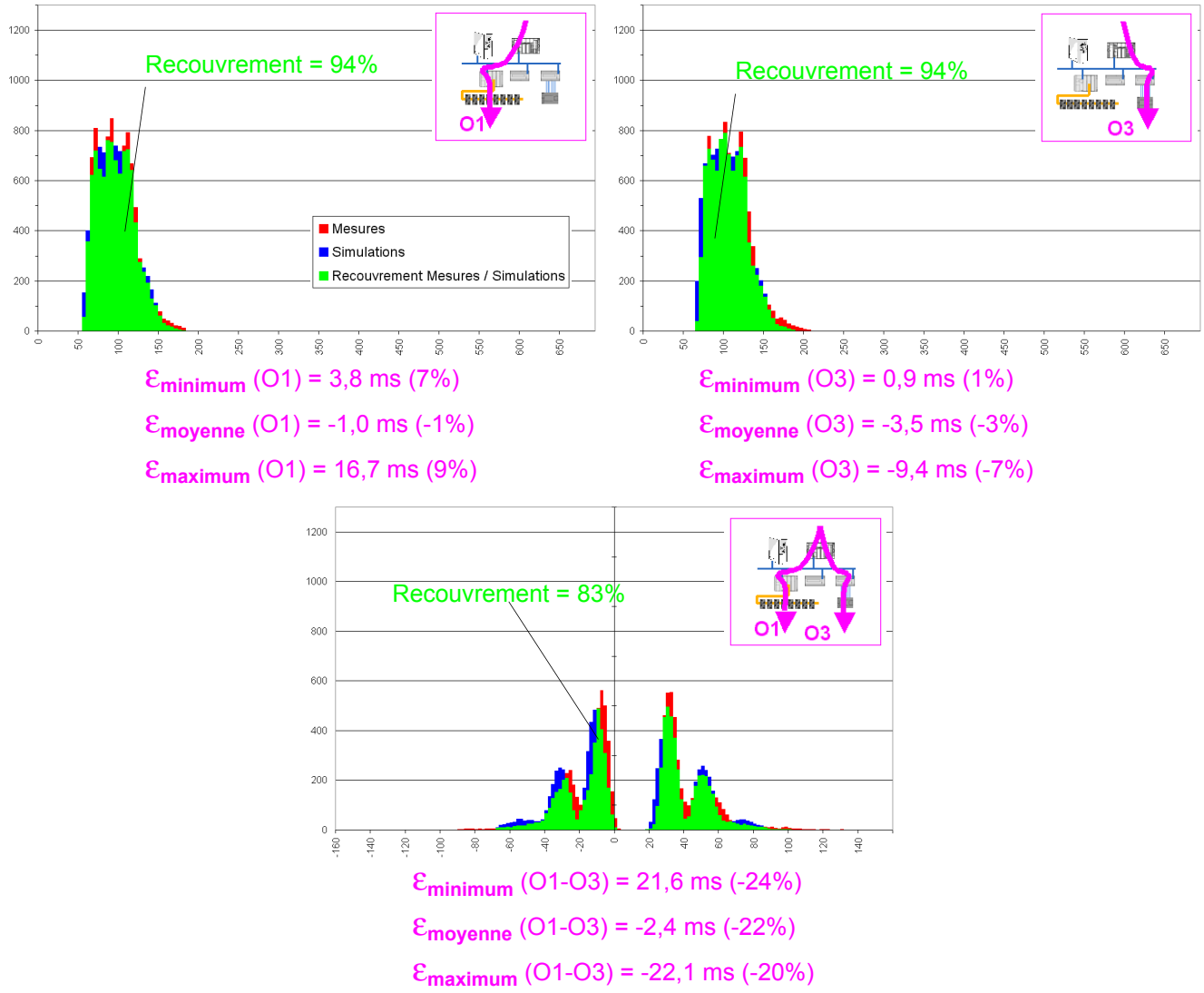
En comparant les résultats obtenus par la mesure expérimentale (fig. 131) sur la plateforme PRISME et ceux obtenus par simulations, nous allons estimer l'erreur de précision de nos modèles par rapport au comportement réel. Pour les simulations, nous réutiliserons les 153 simulations que nous avons effectuées pour l'étude de la convergence, en effet, nous avions paramètré nos modèles pour qu'ils soient conformes aux paramètres de l'architecture de contrôle-commande réelle.

5.4.3.1 Comparaison des résultats de simulation avec ceux des expérimentations

Afin d'étudier la précision, nous évaluerons le taux de recouvrement entre les histogrammes des simulations et des mesures, ainsi que l'erreur relative et absolue entre les valeurs moyennes, minimales, et maximales de chacunes des performances.

Afin de faciliter la lecture et de permettre une comparaison aisée des différents histogrammes, nous les avons représentés avec la même échelle et en les contraignant à apparaitre comme un histogramme d'une population de 10 000 occurences.

Dans un premier temps, nous nous interresserons aux signaux de type ordre (O1 et O3) et au retard de causalité entre ces deux signaux (O1-O3).



$\varepsilon_{minimum}$ (O1) = 3,8 ms (7%)

$\varepsilon_{moyenne}$ (O1) = -1,0 ms (-1%)

$\varepsilon_{maximum}$ (O1) = 16,7 ms (9%)

$\varepsilon_{minimum}$ (O3) = 0,9 ms (1%)

$\varepsilon_{moyenne}$ (O3) = -3,5 ms (-3%)

$\varepsilon_{maximum}$ (O3) = -9,4 ms (-7%)

$\varepsilon_{minimum}$ (O1-O3) = 21,6 ms (-24%)

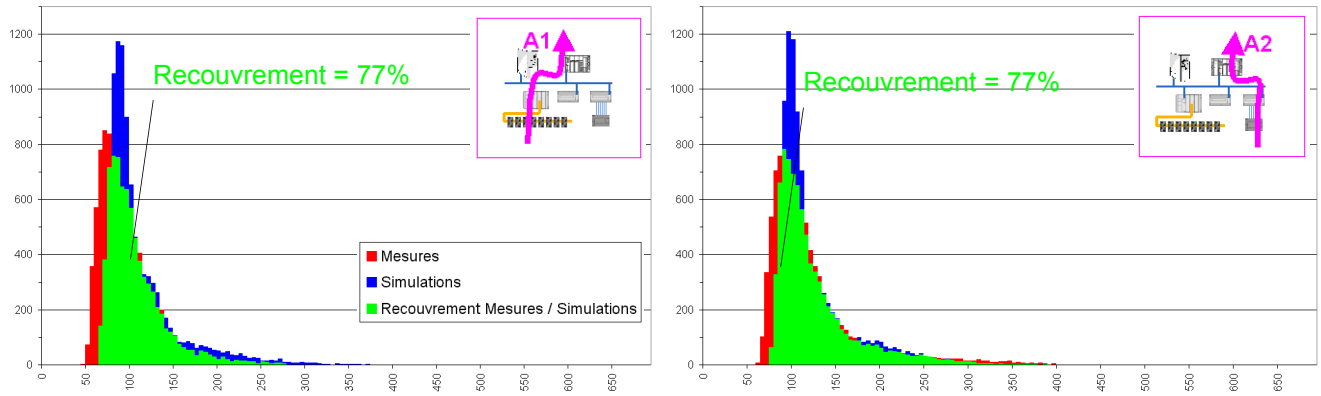
$\varepsilon_{moyenne}$ (O1-O3) = -2,4 ms (-22%)

$\varepsilon_{maximum}$ (O1-O3) = -22,1 ms (-20%)

Figure 144 : Comparaison Simulation/Mesure des performances de type Ordre

Les histogrammes des ordres (O1 et O3) et du retard de causalité (fig. 144) nous montrent que le taux de recouvrement est très important, que l'erreur sur la valeur moyenne est très faible (inférieure à 5%) alors que l'erreur sur les extremums est faible (inférieure à 10%) pour les ordres mais double pour le retard de causalité (les erreurs des ordres se cumulant sur les extremums). On a donc une précision des résultats de simulation très importante pour les signaux de type ordre.

Dans un second temps, nous nous intéresserons aux signaux de type alarmes (A1 et A2).



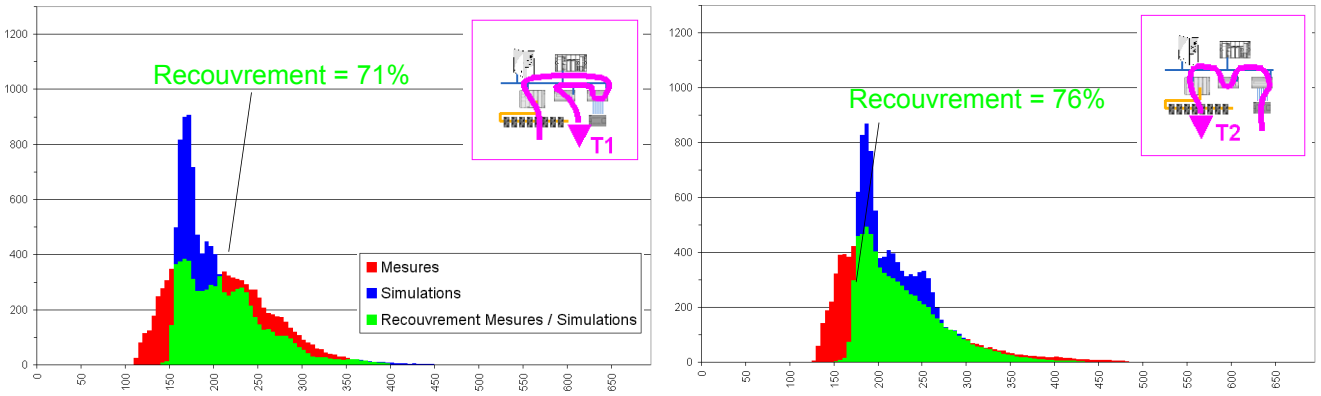
$\varepsilon_{minimum}$ (A1) = 27,7 ms (25%)
$\varepsilon_{moyenne}$ (A1) = -5,1 ms (2%)
$\varepsilon_{maximum}$ (A1) = 239 ms (60%)

$\varepsilon_{minimum}$ (A2) = 26,4 ms (21%)
$\varepsilon_{moyenne}$ (A2) = 6,3 ms (3%)
$\varepsilon_{maximum}$ (A2) = 53,9 ms (11%)

Figure 145 : Comparaison Simulation/Mesure des performances de type Alarmes

On remarque sur les histogrammes (fig. 145) que les taux de recouvrement sont plus petits que ceux des signaux de type ordre mais ils restent très importants. L'erreur sur la valeur moyenne s'avère être très faible pour A2 alors qu'elle est de l'ordre de 10% pour A1. Pour ce qui est de la valeur des extremums, l'erreur se trouve être d'environ 30%.

Enfin, dans un dernier temps, nous nous interresserons aux signaux de type temps de transmission d'informations (T1 et T2).



$\varepsilon_{minimum}$ (T1) = 16,1 ms (33%)
$\varepsilon_{moyenne}$ (T1) = 13,3 ms (14%)
$\varepsilon_{maximum}$ (T1) = 150,9 ms (58%)

$\varepsilon_{minimum}$ (T2) = 15,7 ms (25%)
$\varepsilon_{moyenne}$ (T2) = 1,3 ms (1%)
$\varepsilon_{maximum}$ (T2) = 190,3 ms (52%)

Figure 146 : Comparaison Simulation/Mesure des performances de type transmission

Cette fois, on remarque que le taux de couverture est légèrement plus faible que celui des signaux de type alarme, mais reste très important. L'erreur sur la valeur moyenne s'avère très faible alors que les extremums peuvent avoir des erreurs qui peuvent être importantes (plus de 50%).

5.4.3.2 Analyse de la précision du Modèle

Ces résultats de comparaison sont dans leur ensemble très corrects, mais sachant que l'on a réalisé des simulations et des mesures avec des paramétrages similaires, la question que l'on se pose est sur l'origine de ces écarts.

Les écarts ayant en fait des origines multiples, nous allons nous pencher sur chacun d'entre-eux afin de les expliciter :

- Les résultats de simulation sur la valeur de la moyenne de A1 sont entachés d'une erreur très importante, alors que celle de T1 a une erreur très faible : cela s'explique par le fait que l'on a accumulé durant les 153 simulations près de 1 000 000 de valeurs de T1, alors qu'il n'y a eu que "seulement" 100 000 valeurs de A1 et A2, ces performances ont donc convergées moins efficacement que celles de T1. Etant donné que la période d'excitation de A1 est environ 10 fois plus petite que celle de T1, il est normal d'avoir un tel décalage.
- On observe 30% d'écart entre simulations et mesures concernant la valeur minimale des alarmes et des transmissions : l'explication vient du fait que les temps de cycle des automates programmables utilisés dans l'architecture de contrôle-commande n'ont pas une durée fixe et peuvent varier jusqu'à 5 ms. Étant donné que les modèles réseaux de Petri des API que nous avons utilisés possèdent eux une durée de temps de cycle fixe, nous avons pris le parti de les paramètrer à partir de la valeur maximale des temps de cycle. Cela implique un décalage des extrema qui s'amplifie lorsque le nombre d'API qui intervient dans la chaîne critique de l'information augmente. Le fait d'avoir une erreur due à une vision pessimiste du temps de cycle, est un atout sécuritif car la simulation sur-estime les temps minimum.
- Les répartitions des performances obtenues par simulation sont plus tranchées que celles obtenues par mesure : étant donné que l'ensemble des modèles utilisés en simulation ont des durées fixes dont la précision est le dixième de milliseconde, les petites variations de comportement des équipements de l'architecture réelle ne sont pas prises en compte et donc malgré le fait que la forme générale des répartitions est souvent équivalente entre la simulation et les mesure, l'absence des petites variations implique des répartitions plus "tranchées".
- Les valeurs maximales de la simulation sont beaucoup plus importantes que celles des mesures : le nombre de simulations a été nettement plus important que celles des observations sur la plateforme expérimentale, cela implique que les mesures n'ont sans doute pas eu d'occurrence pour des valeurs dont la probabilité est très faible alors que les simulations ont eu "la chance" d'en obtenir.

La précision des performances obtenues par simulation, est donc dans son ensemble très bonne et permet d'avoir une très bonne estimation des performances que l'on obtiendrait sur une architecture réelle.

5.5 Conclusion

Au cours de ce chapitre, nous avons prouvé dans un premier temps que notre approche basée sur des modèles modulaires des Architectures Matérielle, Fonctionnelle et Opérationnelle complétées par des codes observateurs et des générateurs de jetons est praticable et nous avons ainsi obtenu des résultats de simulations nous permettant d'évaluer les performances d'une architecture.

Dans un second temps, notre ambition est que notre méthode puisse être un compagnon de l'architecte tout au long du cycle de vie de l'architecture. Pour cela, il est nécessaire que ce compagnon puisse donner des résultats satisfaisants pour une pré-conception dans un faible laps de temps mais aussi pouvoir

donner des résultats plus fiables en accordant plus de temps à la simulation.
Nous avons donc vérifié au travers de l'étude de la convergence que notre méthode donne des résultats acceptables dans un laps de temps très court et qu'ils s'améliorent en augmentant le temps de simulation.
De plus l'étude de la sensibilité nous a permis de montrer que l'ordre de grandeur de l'erreur d'une performance étudiée lors de la pré-conception sera inférieure ou égale à l'ordre de grandeur des erreurs sur les paramétrages de l'architecture.

Enfin, nous avons montré au travers de l'étude de la précision que les modèles RdP de la bibliothèque sont valides et qu'ils permettent d'obtenir des résultats justes avec des paramétrages corrects des éléments de l'architecture.

Dans ce chapitre nous avons donc validé notre démarche, et nous avons prouvé qu'elle était un compagnon tout au long du cycle de vie de l'architecture.

Conclusions et perspectives

Notre travail a débuté sur l'évidence selon laquelle l'architecte devait réaliser des évaluations de performances tout au long du cycle de vie de l'architecture de contrôle-commande d'un système automatisé. Ce besoin multiple d'évaluation impose d'utiliser plusieurs méthodes d'évaluations ou d'avoir une unique méthode qui serait contrainte par des paramètres pouvant être totalement maîtrisés ou très approximatifs et par des temps variables accordés à l'évaluation de performances. De plus, les performances que l'architecte doit évaluer doivent tenir compte de l'ensemble du comportement dynamique de l'architecture.

Étant donné que les travaux de recherche de la communauté scientifique ne répondent pas à ce besoin, nous nous sommes fixé l'objectif de développer une démarche unique consistant à réaliser un modèle de comportement dynamique de l'Architecture Opérationnelle complète à partir des modèles modèles de l'Architecture Matérielle et de l'Architecture Fonctionnelle.
Nous avons enrichit le modèle en lui ajoutant un environnement de simulation permettant de réaliser des excitations et d'observer leurs réactions pour en déduire les performances.
Nous avons opté pour une évaluation par simulation en utilisant des modèles réseaux de Petri Colorés et Temporisés selon le formalisme de Jensen avec la plateforme logicielle Design CPN.

Notre démarche devant accompagner l'architecte tout au long du cycle de vie de l'architecture, nous avons développé des guides méthodologiques précisant les différentes phases de la modélisation de l'Architecture Opérationnelle, ainsi que la modélisation et le choix des équipements d'Architecture Matérielle qui devraient êtres développés spécifiquement.

Enfin, nous avons validé notre démarche en traitant un exemple d'architecture comportant les problématiques courantes des architectures complexe, le résultat de ce traitement étant l'obtention des performances temporelles de notre exemple support. Nous avons alors montré la qualité de notre démarche en étudiant la convergence des modèles de simulations, la sensibilité des performances aux erreurs de paramétrages et la précision de nos résultats par rapport à des mesures effectuées sur une plateforme expérimentale PRISME.

Les résultats de ces études ont montrés que la qualité de notre démarche satisfait entièrement les contraintes qu'imposait une méthodologie unique d'évaluation de performances sur l'ensemble du cycle de vie de l'architecture et que l'objectif que nous nous étions fixé est entièrement atteint. Toutefois à partir de ces travaux de nombreuses perspectives s'ouvrent à nous.

La première perspective serait de chercher à compenser les erreurs que l'on a relevées dans la comparaison des résultats de simulation et les résultats de mesure. Résoudre ces problèmes serait une amélioration certaine, mais ces résolutions ne sont pas forcément aisées à mettre en oeuvre. Afin de compenser les erreurs, il serait nécessaire de :

- réaliser autant de mesures expérimentales sur la plateforme PRISME, ce qui nécessite un temps beaucoup plus important car d'après les expériences menées, une série de mesures sur la plateforme PRISME aurait besoin d'environ 30 fois plus de temps qu'une simulation pour atteindre le même nombre de valeurs observées.
- utiliser des modèles d'automates programmables plus fins, qui tiendraient en compte une durée de temps de cycle variant selon les informations traitées. L'important dans le développement de ces nouveaux modèles API, serait que leurs paramètres soient aisément identifiable afin que l'architecte puisse les utiliser.
- rajouter aux modèles des variations qui permettraient de se rapprocher du comportement réel de l'architecture de contrôle-commande. Toutefois, cela risque de perturber les valeurs en rajoutant ou soustrayant des perturbations alors qu'elles ne devraient pas participer au temps de transmission et donc fausser les performances de l'architecture.
- réduire l'unité de temps de simulation (qui est actuellement de 100 µs) pour éviter les problèmes d'arrondis dans les modèles (le modèle réseau UniTelway calcule la durée de transmission en fonction de la taille des informations) afin de réduire les écarts entre la simulation et les mesures. Toutefois, DesignCPN ne nous permettant pas de réaliser une simulation avec un horizon de plus de 100 000 000 unités de temps, si on passe à une résolution de 10 µs, il sera nécessaire de réaliser 10 fois plus de simulations.

La seconde perspective irait dans le sens de l'amélioration de la praticabilité de la méthode en développant une interface logiciel où l'architecte n'aurait plus à modéliser des modèles réseaux de Petri mais à représenter uniquement une représentation de son architecture de contrôle-commande. Cela permettrait à l'architecte de ne pas avoir à développer de compétences en modélisations réseau de Petri pour effectuer une évaluation de performances.

La troisième perspective va dans le sens de l'extension des performances étudiées en s'intéressant aux taux de charges ainsi qu'à la disponibilité des équipements composant l'architecture de contrôle-commande. Il serait donc nécessaire de développer des nouveaux types d'observateurs qui permettraient d'observer la charge des différents équipements. Mais aussi des nouveaux modèles matériels qui tiendraient compte des pannes associées aux équipements. Cela permettrait à l'architecte de pouvoir prendre en compte les problèmes de sûreté de fonctionnement ainsi que la gestion des modes dégradés des installations industrielles.

Références bibliographiques

[ABDELLATIF & JUANOLE 2003a]
Improved flow admission algorithm for a class of rate-controlled packet networks, *S. Abdellatif , G. Juanole*, 24th IEEE Real-Time Systems Symposium. Work-in-Progress Session, Cancun (Mexique), 3-5 Décembre 2003, 7p.

[ABDELLATIF & JUANOLE 2003b]
Evaluation de performances de protocoles de communications, *S. Abdellatif , G. Juanole*, Ouvrage Verification et mise en oeuvre des reseaux de Petri, Hermes Science, Traite IC2 Information-Commande-Communication, N°ISBN 2-7462-0445-2, 2003, Chapitre 11, pp.357-384

[ALKHODRE & al 2002]
Méthodologie de développement des systèmes embarqués temps réel basé sur le langage SDL, *A. Alkhodre, J.-P. Babau, J.-J. Schwarz*. 3ieme Colloque de CAO de circuits et systèmes intégrés. Paris, 15-17 Mai 2002

[ANDRE & VAILLY 2001]
Conception des systèmes d'informations; panorama des méthodes et techniques, *P. Andre, A. Vailly*, Edition Ellipse, janvier 2001, ISBN 2-7298-0479-X)

[AUBRY 2002]
Automated systems : from the control to the dependability. Few answers, some oponed questions, *AUBRY Jean-François*, in 8th Internationnal Conference on Quality, reliability and maintainability, CCF02 Sinaia, Roumanie 18-20 septembre 2002

[BARANA & al 2002]
Design choices for control and data acquisition in RFX, *Oliviero Barana, Adriano Luchetta, Gabriele Manduchi and Cesare Taliercio*, Fusion Engineering and Design Volume 60, Issue 3 , June 2002, Pages 361-366

[BARGER & al 2003a]
Modelling and analyzing safety criteria of real communication protocol, *BARGER Pavol, THIRIET Jean-marc et ROBERT Michel*, dans le 5ième congrès international pluridisciplinaire Qualité et Sureté de fonctionnement, QUALITA'2003

[BARGER & al 2003b]
Safety Analysis and Reliability Estimation of a Networked Control System, *P. Barger, J.-M. Thiriet, M. Robert*, 5th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes, SAFEPROCESS'2003, Washington D.C. - USA, June 9-11 2003, p.1047-1052

[BARGER 2004]
Evaluation et validation de la fiabilité et de la disponibilité des systèmes d'automatisation à intelligence distribuée en phase dynamique, *BARGER Pavol*, Doctorat de l'Université Henri Poincaré - Nancy 1, 15 décembre 2003.

[BASS & al 1997]
Software Architecture in Practice, *Bass, Clements and Kazman*, Published by Addison-Wesley 1997 in the SEI Series.

[BETOUS & KANOUN 2004a]
Construction and stepwise refinement of dependability models,*Cláudia Betous-Almeida and Karama Kanoun*, ELSEVIER Performance Evaluation, Volume 56, Issues 1-4, March 2004, Pages 277-306

[BETOUS & KANOUN 2004b]
Dependability modelling of instrumentation and control systems: A comparison of competing architectures, *Cláudia Betous-Almeida and Karama Kanoun*, ELSEVIER Safety Science, Volume 42, Issue 5, June 2004, Pages 457-480

[BHOWAL & al 1999]
Estimating micro-PLC execution time for time critical system design, *Prodip Bhowal, Rajib Mall and Anupam Basu*, Journal of Systems Architecture, Volume 45, Issue 14, July 1999, Pages 1245-1248

[BREMAUD 1998]
Markov chains, Gibbs fields, Monte Carlo simulation and queues, *Pierre Bremaud*, Springer-Verlag, 1998.

[BUDKOWSKI & NAJM 1983]
Structured Finite State Automata - A new approach for modelling distributed communications systems, *Stanislaw Budkowski, Elie Najm*. Protocol Specification, Testing, and Verification 1983: 95-110

[BURNS & al 1998]
Analysing Superscalar Processor Architectures with Coloured Petri Nets, *F. Burns, A. Koelmans, A. Yakovlev.* International Journal on Software Tools for Technology Transfer, 2 (1998), Springer-Verlag, 182-191

[CAPELLMANN & al 1998]
Visualising the Behaviour of Intelligent Networks, *C. Capellmann, S. Christensen, U. Herzog* In: **Services and Visualization, Towards User-Friendly Design,** *T. Margaria, B. Steffen, R. Rückert, J. Posegga* (Eds.), Lecture Notes in Computer Science, vol. 1385, Springer-Verlag 1998, 174-189.

[CAPELLMANN & al 1999]
Using High-Level Petri Nets in the Field of Intelligent Networks, *C. Capellmann, H. Dibold, U. Herzog* In: **Application of Petri Nets to Communication Networks,** *J. Billington, M. Diaz, G. Rozenberg* (eds.), Lecture Notes in Computer Science Vol. 1605, Springer-Verlag 1999, 1-36.

[CASTELPIETRA & al 2002]
Analysis and Simulation Methods for Performance Evaluation of a Multiple Networked Embedded Architecture, *Paolo Castelpietra, Ye-Qiong Song, Françoise Simonot-Lion, and Mondher Attia*, IEEE Transactions on Industrial Electronics , Vol. 49, N°. 6, December 2002

[CAVALIERE 2002]
DAMeSI : un profil UML pour l'évaluation de performances des systèmes d'automatisation distribués, *Cavaliere Domenico*, Thèse d'université, INPL, June 2002

[CAVALIERI & al 1996]
Mapping automotive process control on IEC/ISA Fieldbus functionalities, *S. Cavalieri, A. Di Stefano, O. Mirabella*, Computers in Industriy 28 (1996) 233-250

[CAVALIERI & al 1999]
Enhancing reliability in IEC/ISA FieldBus, *S. Cavalieri , O. Mirabella, S. Monforte*, Computer Standards & Interfaces 21 1999 217–240

[CAVALIERI & al 2002]
Multi-master Profibus Dp Modelling And Worst Case Analysis-based Evaluation, *Salvatore Cavalieri, Salvatore Monforte, Eduardo Tovar, Francisco Vasques*, 15th Triennial World Congress of the International Federation of Automatic Control, Barcelona, 21–26 July 2002

[CHOU & BORRIELLO 2000]
Synthesis and Optimization of Coordination Controllers for Distributed Embedded Systems, *Pai H. Chou, Gaetano Borriello*, Proceeding of 37th Design Automation Conference, Los Angeles, CA, USA, June 5 - 9, 2000 [p. 410]

[CHRISTENSEN & al 1997]
Design/CPN - A Computer Tool for Coloured Petri Nets, *Søren Christensen, Jens Bæk Jørgensen, Lars Michael Kristensen*. Proceedings in Tools and Algorithms for Construction and Analysis of Systems, Third International Workshop, TACAS '97, Enschede, The Netherlands, April 2-4, 1997, p209-223

[DAVID & ALLA 1989]
Du Grafcet aux réseaux de Petri, *David R., Alla H.*, In Traité des Nouvelles Technologies - série Automatique. 2nde édition, Hermès (Eds), ISBN 2-86601-325-5

[DENIS & al 1993]
Cahier des charges au format guide d'Analyse des Besoins et Modélisation SADT, MERISE, GEMMA et OBJETS, *Bruno Denis, Jean-Jacques Lesage, Jean-Marc Roussel, Guy Timon*, Rapport de fin d'étude EXERA : RAFALE-X54, Mai 1993

[DENIS 1994]
Assistance à la conception et à l'évaluation de l'architecture de conduite des systèmes de production complexes, *Bruno Denis*, Thèse de doctorat de l'Université Nancy I., 17 février 1994

[DENIS 1995]
Formalisation de la conception d'architecture de conduite des systèmes de production, *Bruno Denis, Jean-Jacques Lesage et Guy Timon*, Association Internationale pour l'Automatisation Industrielle, 2ème Conférence Internationnal sur L'Automatisation Industrielle, Nancy, France, 7-9 Juin 1995

[DIAZ & al 2001]
Les réseaux de Petri - Modèles fondamentaux, *B. Berthomieu, M. Boye, M. Diaz, C. Girault, S. Haddad, P. Moreaux, J-F. Pradat-Peyre, P. Sénac, F. Vernadat*, Edition Hermès science publications 2001 ; série Informatique et Systèmes d'Informations, traité IC2 Informations - Commande - Communication, ISBN : 2-7462-0250-6

[DUTHEILLET 1992]
Symétries dans les réseaux colorés. Définition, analyse et applications à l'évaluation de performance, *C. Dutheillet*, Thèse de doctorat, Université Pierre et Marie Curie, Paris, 1992

[ELLIOT & al 2002]
Using Design/CPN to Design a Visualisation Extension for Design/CPN, *M. Elliot, J. Billington, L.M. Kristensen*. In Proceedings of the Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, *K. Jensen* (ed.). August 2002, Department of Computer Science, University of Aarhus, PB-560, p21-37

[ELLOY & SIMONOT-LION 2002]
An Architecture Description Language For In-vehicle Embedded System Development, *Jean-Pierre Elloy, Françoise Simonot-Lion*, 15th Triennial World Congress of the IFAC, Barcelona, 21–26 July 2002

[ERNST 1998]
Codesign of embedded systems: Status and trends, *R. Ernst*, IEEE Design & Test of Computers, pages 45-- 54, April 1998.

[FIGUEIREDO & KRISTENSEN 1999]
Using Coloured Petri Nets to Investigate Behavioural and Performance Issues of TCP Protocols, *J.C.A. de Figueiredo, L.M. Kristensen*. In **Proceedings of the 2nd Workshop on Practical Use of Coloured Petri Nets and Design/CPN**, *K.Jensen (ed.)*, Aarhus 1999, Department of Computer Science, University of Aarhus, PB-541, 21-40.

[FOTA & al 1999]
Dependability evaluation of an air traffic control computing system, *Nicolae Fota, Mohamed Kaâniche, Karama Kanoun*, in Performance Evaluation, Volume 35, Issues 3-4, May 1999, Pages 253-273.

[FRAPPIER & HABRIAS 2001]
Software Specification Methods - An Overview Using a Case Study, *Marc Frappier and Henri Habrias* (Eds), Edition Springer 2001, ISBN : 1-85233-353-7

[GENRICH & LAUTENBACH 1981]
System modelling with high-level Petri nets; *H.J. Genrich et K. Lautenbach*, Theorical Computer Science, vol 13, p. 103-136, 1981

[GEORGES & al 2003a]
Evaluation des performances de réseaux Ethernet industriels : Etude et comparaisons de différents modèles, *Jean-Philippe Georges, Eric Rondeau et Thierry Divoux*, JDA'03, Journées Doctorales d'Automatique, Valenciennes, 25-27 juin 2003.

[GEORGES & al 2003b]
Evaluation de performance d'architectures Ethernet commuté, *GEORGES Jean-Philippe, RONDEAU Eric et DIVOUX Thierry*, TS Technique et science informatiques, , vol. 22, n°5, pp. 621-649, ISSN 0752-4072, juin 2003

[GEORGES & al 2003c]
Evaluation d'architectures Ethernet commuté par calcul et simulation, *GEORGES Jean-Philippe, RONDEAU Eric et DIVOUX Thierry*, dans le 4ième Colloque Francophone sur la Modélisation des Systèmes Réactifs, MSR'03, Metz, du 6 au 8 octobre 2003

[GOUYON & al 2004]
Product-driven Automation Issues For B2m-control Systems Integration, *David GOUYON, Jean Marcelo SIMÃO, Khaled ALKASSEM, Gerard MOREL*, INCOM'2004, 11th Symposium on Information Control Problem in Manufacturing, Salvador - Bahia - Brazil, 5-7 April, 2004

[HAMADI & al 2003]
SILKEY : a tool for the automatic evaluation of safety and avaibility of multi-level redundancies architectures, *HAMADI Karim, AUBRY Jean-François et MALASSE Olaf*, in 21st ISSC : Internationnal System Safety Conference, Ottawa, 4-9 août 2003

[HATLEY & PIRBHAI 1987]
Strategies for Real-Time System Specification, *Hatley, D. J., Pirbhai, I. A.*, Dorset House, New York, 1987

[HEBUTERNE 2000]
Introduction à la simulation par événements, *Gérard Hébuterne*, cours à l'Institut National des Télécommunications, 2000

[HEVIN 1997]
Identification des systèmes séquentiels par la plate-forme PRISME, *N. Hevin*, Mémoire, DEA de Production Automatisée, LURPA/ENS Cachan, Juil. 1997

[HIELSCHER & al 1998]
On Modelling Train Traffic in a Model Train System, *W. Hielscher, L. Urbszat, C. Reinke, W. Kluge*. In Proceedings of the Workshop on Practical Use of Coloured Petri Nets and Design/CPN, *K.Jensen* (ed.). Aarhus 1998, Department of Computer Science, University of Aarhus, PB-532, 83-101.

[HINES & BORRIELLO 1997]
Dynamic Communication Models in Embedded System Co-Simulation, *Ken Hines, Gaetano Borriello*, in Proc. of the 34th Design Automation Conference, June 1997. pp. 395-400

[HUMBERT & al 1997]
Embedded Applications in the Paris Buses, *L. Humbert, E. Rondeau, T. Divoux, L. Roulet*, SICICA'97, 3rd IFAC Symposium on Intelligent Components and Instruments for Control Applications, Annecy, France, 9-11 juin 1997. Edité par L. Foulloy, pages 101-105

[HUMBERT & al 1998]
Simulation de trafic sur le réseau embarqué CAN : Application aux bus parisiens, *L. Humbert, E. Rondeau, T. Divoux, L. Roulet*, JDIR'98, 2ème Congrès Journées Doctorales Informatique et Réseaux, Paris, France, 27-29 avril

[JANSEN & al 1998]
Technical issues in modelling the European train control system, *L. Jansen and M. zu and H. Schnieder*, Technical issues in modelling the European train control system. In Proc. 1st CPN Workshop, DAIMI PB 532, pages 103--115. Aarhus University, 1998.

[JENSEN 1986]
Coloured Petri Nets, *Kurt Jensen*. Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, **Proceedings of an Advanced Course**, Bad Honnef, 8.-19. September 1986. Lecture Notes in Computer Science, *Wilfried Brauer, Wolfgang Reisig, Grzegorz Rozenberg (Eds.)* 255 Springer 1987 , ISBN 3-540-17906-2

[JENSEN 1997a]
Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts. *K. Jensen*, Monographs in Theoretical Computer Science Springer-Verlag 2nd corrected printing 1997. ISBN: 3-540-60943-1

[JENSEN 1997b]
Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 2, Analysis Methods. *K. Jensen*, Monographs in Theoretical Computer Science Springer-Verlag 2nd corrected printing 1997. ISBN: 3-540-58276-2

[JENSEN 1997c]
Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 3, Practical Use. *K. Jensen*, Monographs in Theoretical Computer Science Springer-Verlag 2nd corrected printing 1997. ISBN: 3-540-62867-3

[JIANG & al 2001]
Colored Petri Nets with changeable structures (CPN-CS) and their applications in modeling one-of-a-kind production (OKP) systems, *Zhibin Jiang, Ming J. Zuo, Richard Y. K. Fung and Paul Y. L. Tu*, Computers & Industrial Engineering, Volume 41, Issue 3, December 2001, Pages 279-308

[JORGENSEN 2002]
Coloured Petri Nets in UML-Based Software Development. Designing Middleware for Pervasive Healthcare, *J.B. Jørgensen*. In Proceedings of the Fourth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, *K. Jensen (ed.)*. August 2002, Department of Computer Science, University of Aarhus, PB-560, 61-80.

[JUANOLE 2002]
Quality Of Service Of Communication Networks And Distributed Automation: Models And Performances, *Guy Juanole*, 15th Triennial IFAC World Congress, Barcelona, Spain, 2002

[JUMEL & al 2003]
Towards an information-based approach for the dependability evaluation of distributed control systems, JUMEL *Fabrice et THIRIET Jean-Marc et AUBRY Jean-François et MALASSE Olaf*, 20th IEEE Instrumentation and Measurement Technology Conference (IEEE/IMTC2003), Vail (Colorado, United States), 20-22nd May 2003

[KLEINROCK 1975]
Queueing Systems, *Leonard Kleinrock*, Volume 1: Theory, 1975, ISBN 0-471-49110-1. Volume 2: Computer Applications, 1976, ISBN 0-471-49111-X. John Wiley and Sons.

[KOUBAA & SONG 2002]
Evaluation de performances d'Ethernet commuté pour des applications temps réel., *Anis Koubaa and YeQiong Song.*, In 10th International Conference on Real Time and Embedded Systems - RTS'2002, Paris, France, March 2002

[KOWALEWSKI & al 1999]
An open tool architecture for the formal verification of logic controllers in processing systems, *S. Kowalewski, N. Bauer, J. PreuBig, O. Stursberg, H. Treseler*, Preprin IFAC World Congress, Bejing, P.R. China 1999

[LAVANDIER 1998]
Méthode de détermination expérimentale du comportement réactif d'un équipement industriel de commande, *M. Lavandier*, Mémoire, DEA de Production Automatisée, LURPA-ENS Cachan, Juil. 1998

[LEE 2002]
Evaluating real-time software specification languages, *Dong-Tsan Lee*, Computer Standards & Interfaces Volume 24, Issue 5 , November 2002, Pages 395-409

[LEE & LEE 2002]
Performance evaluation of switched Ethernet for real-time industrial communications, *Kyung Chang Lee and Suk Lee*, Computer Standards & Interfaces Volume 24, Issue 5 , November 2002, Pages 411-423

[LEE & al 2003]
Development of performance model for calculation of communication delay in Profibus token passing protocol, *Kyung Chang Lee, Hyun Hee Kim and Suk Lee*, Computer Standards & Interfaces Volume 25 (2003), Pages 539–552

[LEE & al 2004]
Implementation and PID tuning of network-based control systems via Profibus polling network, *Kyung Chang Lee, Suk Lee and Hong Hee Lee*, Computer Standards & Interfaces Volume 26, (2004), Pages 229–240

[LIAN & al 2001]
Performance Evaluation of Control Networks : Ethernet, ControlNet and DeviceNet, *Feng-Li Lian, James R. Moyne and Dawn M. Tilbury*, IEEE Control Systems Magazine, Vol. 21, n° 1, page(s) : 66-83, February 2001

[LIAN 2001]
Analysis, Design, Modelling and Control of Networked Control Systems, *Feng-Li Lian*, thèse de doctorat de l'université du Michigan (USA), May 2001

[LINDSTROM & HAIDER 2001]
Equivalent Coloured Petri Nets Models of a Class of Timed Influence Nets with Logic, *B. Lindstrøm, S. Haider.* In: **Proceedings of the Third Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools,** *K. Jensen (ed.)*, August 2001, Department of Computer Science, University of Aarhus, PB-554, 35-54.

[LORENTSEN & al 2001]
Modelling Feature Interaction Patterns in Nokia Mobile Phones using Coloured Petri Nets and Design/CPN, *L. Lorentsen, A.-P. Touvinene, J. Xu* In: **Proceedings of the Third Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools**, *K. Jensen (ed.)*, August 2001, Department of Computer Science, University of Aarhus, PB-554, 1-14.

[MACHADO & al 2003]
Increasing the efficiency of PLC Program Verification using a plant model, *José M. Machado, Bruno Denis, Jean-jacques Lesage, Jean-marc Faure, Jaime C. L. Ferreira Da Silva*, 6th Industrial Engineering and Production Management IEMP'03, CDRom proceedings, Paper ormhm-4, 10 pages, Porto (Portugal), May 2003

[MARCA & McGOWAN 1988]
SADT: Structured Analysis and Design Technique, *D. A. Marca and C. L. McGowan* . New York, NY. McGraw-Hill Book Co., Inc. 1988

[MATEOS & al 2001]
Design and development of an automatic small-scale house for teaching domotics, *Felipe Mateos, Víctor M. González, Reyes Poo, Marta García and Rosana Olaiz*, 31st ASEE/ IEEE Frontiers in Education Conference, October 10 - 13, 2001 Reno, NV

[MEUNIER & DENIS 1997]
Validation du Comportement Dynamique des Architectures de Conduite des Systèmes de Production par Simulation, *Pascal Meunier, Bruno Denis.* MOSIM'97.

[MEUNIER & al 2000a]
Comparison of differents modelling aproaches in simulation of Programmable Logic Controller, *Pascal Meunier, Bruno Denis, Jean-jacques Lesages*. CSD'00, june 2000, Bratislava (Slovak Republic).

[MEUNIER & al 2000b]
Safety Analysis During the Control Architecture Design of Automated Systems, *Pascal Meunier, Bruno Denis, Jean-jacques Lesages*. SAFEPROCESS'00, june 2000, Budapest (Hungary).

[MONTCELET & al 1998]
Analysing a Mechatronic System with Coloured Petri Nets, *G. Moncelet, S. Christensen, H. Demmou, M. Paludetto, J. Porras*. International Journal on Software Tools for Technology Transfer, 2 (1998), Springer-Verlag, 160-167

[MORERA & GONZALES 1999]
A CPN Model of the MAC Layer, *P.H. Morera, T.M.P. Gonzalez*. In Proceedings of the 2nd Workshop on Practical Use of Coloured Petri Nets and Design/CPN, *K.Jensen* (ed.), Aarhus 1999, Department of Computer Science, University of Aarhus, PB-541, 153-172.

[MURATA 1989]
Petri Nets: Properties, Analysis and Applications, *T. Murata*. Proceedings of the IEEE, vol. 77, n.4, April 1989, p541-580.

[NAVET & SONG 2001]
Validation of in-vehicle real-time applications, *Nicolas Navet and Ye-Qiong Song*, Computers in Industry, Volume 46, Issue 2, September 2001, Pages 107-122

[NIGRO & PUPO 1998]
Using Design/CPN for the Schedulability Analysis of Actor Systems with Timing Constraints, *L. Nigro, F. Pupo*. In Proceedings of the Workshop on Practical Use of Coloured Petri Nets and Design/CPN, *K. Jensen* (ed.). Aarhus 1998, Department of Computer Science, University of Aarhus, PB-532, 271-285.

[ORTEGA & BORRIELLO 1997]
Communication Synthesis for Embedded Systems with Global Considerations, *Ross Ortega, Gaetano Borriello*, in Proc. Codes/CACHE, Braunschweig, Germany, March 24-26, 1997, pp. 69--73.

[ORTEGA & BORRIELLO 1998]
Communication Synthesis for Distributed Embedded Systems, *Ross B. Ortega and Gaetano Borriello*, Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, November 1998

[OTANEZ & al 2002]
Performance Optimization of Networked Control Systems, *Paul G. Otanez*, Master of Science Thesis de l'université du Michigan (USA), April 2002

[PETRI 1979]
Introduction to General Net Theory. in Net Theory and Applications, *C. A. Petri* . In Proceedings of the Advanced Course on General Net Theory of Processes and Systems, Hamburg, October 8-19, 1979, p1-19

[PHILIPPOT & al 2004]
Méthodologie de modélisation dans le cadre de la synthèse formelle des SED, *Alexandre Philippot, Abdelouahed Tajer, François Gellot et Véronique Carré-ménétrier*, Conférence Internationnale Francophone d'Automatique, pages 160-165, les 22-24 novembre 2004, Douz (Tunisie)

[RATZER & al 2003]
CPN Tools for Editing, Simulating, and Analysing Coloured Petri Nets, *Anne V. Ratzer, Lisa Wells, Henry Michael Lassen, Mads Laursen, Jacob Frank Qvortrup, Martin Stig Stissing, Michael Westergaard, Søren Christensen, Kurt Jensen*. ICATPN 2003: 450-462

[REISIG 1991]
Petri nets and algebraic specifications, *W. Reisig*, Theorical Computer Science, Vol. 80, p. 1-34, 1991

[RICHARD & al 2003]
Placement et Validation dans les Systèmes Temps Réel Distribués, *Michael Richard, Pascal Richard and Francis Cottet*, RTS'03, edited by Teknea, 2003, pp. 387-410

[SIFAKIS 2002]
Systèmes embarqués, défis et perspectives, *J. Sifakis*. Journées du Réseau Thématique Prioritaire SECC, « Systèmes Embarqués Complexes ou Contraints », 14 - 15 Novembre 2002 à l'ENS Cachan

[SIMONOT-LION 1999]
Une contribution à la modélisation et à la validation d'architectures temps réel, *Simonot-Lion F.*, INPL, Habilitation à Diriger des Recherches, décembre 1999.

[SOUMBAEV & al 2002]
Plans for a control system for the LUE200 LINAC in JINR, *A.P. Soumbaev, A.S. Kayukov, V. A. Shvets, O. V. Strekalovsky, M.M. Korjovkina*, Proceedings of LINAC 2002, 21th International Linear Accelerator Conference, Gyeongju, Korea, 19-23 August 2002

[SURF2 1994]
SURF-2 : outil d'évaluation de la sûreté de fonctionnement par chaînes de Markov et réseaux de Petri stochastiques, Rapport LAAS No94088 : 9ème Colloque International de Fiabilité et de Maintenabilité (ESREL'94) (Short Paper), La Baule (France), 30 Mai - 3 Juin 1994, pp.19-22

[TATUM & al 1995]
Control System for the Holifield Radioactive Ion Beam Facility, *B.A. Tatum, R.C. Juras, and M.J. Meigs*, presented at the 16th Biennal Particle Accelerator Conference - IEEE PAC'95, Dallas, TX, USA, 1-5 May 1995

[TOVAR & VASQUES 1999a]
Cycle Time Properties of the PROFIBUS Timed Token Protocol, *Eduardo Tovar, Francisco Vasques*, Published in Computer Communications, Vol. 22, No. 13, pp. 1206-1216, Elsevier Science, August 1999.

[TOVAR & VASQUES 1999b]
Supporting real-time distributed computer-controlled systems with multi-hop P-NET networks, *Eduardo Tovar, Francisco Vasques, Alan Burns*, Published in Control Engineering Practice, N° 7, pp. 1015-1025, Elsevier Science, 1999.

[TOVAR & VASQUES 2001]
Distributed computing for the factory-floor: a real-time approach using WorldFIP networks, *Eduardo Tovar and Francisco Vasques*, Computers in Industry, Volume 44, Issue 1, January 2001, Pages 11-31

[VITTURI 2000]
Some features of two fieldbuses of the IEC 61158 standard, *S. Vitturi*, Computer Standards & Interfaces, Volume 22, Issue 3, 1 August 2000, Pages 203-215

[VITTURI 2001]
On the use of Ethernet at low level of factory communication systems, *S. Vitturi*, Computer Standards & Interfaces, Volume 23, Issue 4, September 2001, Pages 267-277

[VITTURI 2003]
PC-based automation systems: an example of application for the real-time control of blowing machines, *S. Vitturi*, Computer Standards & Interfaces, In Press, Uncorrected Proof, Available online

[WANG 1998]
Timed Petri Nets: Theory and Application, *J. Wang*. Kluwer Academic Publishers, Boston, 1998.

[WANG & al 2002]
Worst-case Response Time Of Aperiodic Message In Worldfip, *Zhi WANG, Ye_qiong SONG, Hai-bin YU and Youxian SUN*, 15th Triennial World Congress of the International Federation of Automatic Control, Barcelona, 21–26 July 2002

[WARD & MELLOR 1985]
Structured Development for Real-Time Systems, *Ward, P., Mellor, St*. Vol. 1-3, Yourdon Press, Englewood Cliffs, 1985.

[XIN & al 2000]
Non-autonomous coloured Petri net-based methodology for the dispatching process of urban fire-fighting, *Han Xin, Li Jie and Shen Zuyan*, Fire Safety Journal, Volume 35, Issue 4, November 2000, Pages 299-325

[XU & KUUSELA 1998]
Analyzing the Execution Architecture of Mobile Phone Software with Coloured Petri Nets, *J. Xu, J. Kuusela*, International Journal on Software Tools for Technology Transfer, 2 (1998), Springer-Verlag, 133-143.

[YANG & al 1995]
Modelling and simulation of a soaking pit/rolling mill process based on extended coloured petri nets, *Y. Y. Yang, D. A. Linkens and N. Mort*, Control Engineering Practice, Volume 3, Issue 10, October 1995, Pages 1359-1371

Références techniques

[IEEE P1471/D5.3]
IEEE Standard for Architectural Description of Software-Intensive Systems, IEEE P1471/D5.3

[ISO 11898:1993]
Road vehicles -- Interchange of digital information -- Controller area network (CAN) for high-speed communication published in 1999-06-10

[ISO/IEC 1131-3]
Norme IEC 1131-3 Langages de programmation des Automates Programmables, publiée en mars 1993

[ISO/IEC 15909-1]
ISO/IEC 15909-1 - Software and Systems Engineering - High-level Petri Nets - Concepts, Denitions and Graphical Notation - *Committee Draft ISO/IEC 15909*, October 2, 1997, Version 3.4

[ISO/IEC 9899]
Norme ISO/IEC 9899:1999 - Langages de programmation -- C, published 1999-12-01.

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

[SCHNEIDER 2000]
Le meunier veille au grain, article dans InterSection, le magazine Schneider Electrique de l'enseignement technique et professionnel du 26 Juin 2000,
http://www.intersections.schneider-electric.fr/stock_images/telec/1/n3/Minoterie.pdf

[SCHNEIDER CA 2000]
"Scrutinizing well safety Control system for gas product and processing facility." article paru dans Manufacturing Automation du March/April 2000
http://www.schneider-electric.ca/www/en/crns/SHELL_CAROLINE.PDF

[SCHNEIDER 2002]
Un soutient efficace pour les moteurs HDI, article dans InterSection, le magazine Schneider Electrique de l'enseignement technique et professionnel du 26 Novembre 2002,
http://www.intersections.schneider-electric.fr/stock_images/telec/1/n3/moteurs_hdi.pdf

[SCHNEIDER 2003]
Les bus et les réseaux de terrain en automatisme industriel, Guide Technique dans InterSection, le magazine Schneider Electrique de l'enseignement technique et professionnel du 20 Novembre 2002,
http://www.intersections.schneider-electric.fr/stock_images/telec/1/n3/GT_RESEAUX.pdf

Annexe A

Exemples de modélisation du comportement dynamique de tâches fonctionnelles spécifiques

A.1 Exemple de modélisation d'une tâche fonctionnelle temporisée

Une tâche fonctionnelle temporisée correspond à une tâche dont le comportement est dépendant du temps. Nous considérerons que cette dépendance se limite à un retard entre un événement d'entrée et la sortie associée.

La modélisation du comportement dynamique d'un atome temporisé s'effectue de manière similaire à celle d'une tâche de consommation-production. On adjoint cependant à la transition une temporisation qui permet de modéliser le temps de réalisation de la tâche fonctionnelle associée à cet atome (dans notre exemple figure 147 : un retard noté Δt).
La temporisation associée à la transition se note @+Δt (fig. 147).

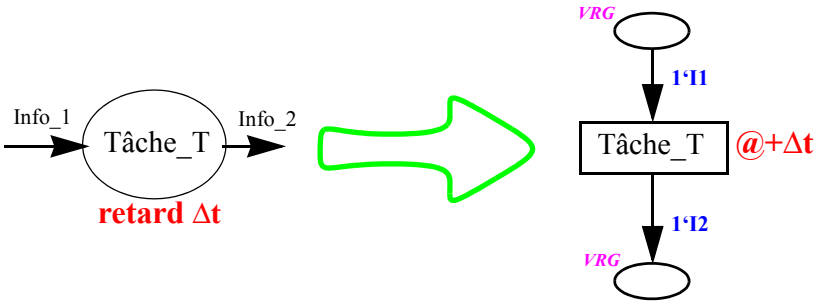


Figure 147 : Modélisation d'un atome temporisé

Dans le logiciel Design CPN, le symbole «@» correspond à la date à laquelle la transition est franchie. Aussi, lorsque l'on associe à une transition la temporisation @+Δt, tout jeton qui la franchira se verra marqué par la date actuelle augmentée de la durée Δt.

Le comportement de la temporisation vis-à-vis du jeton est différent du comportement des réseaux de Petri T-Temporisés ou du P-Temporisés. Nous appellerons ce type de comportement «J-Temporisé» (comme Jeton-Temporisé). Chaque jeton qui devra prendre en compte le temps, est appelé jeton temporisé. La date de disponibilité d'un jeton temporisé est notée entre crochet à coté du jeton. Si un jeton temporisé n'a pas franchi de transition temporisée, alors sa disponibilité est à zéro (fig. 148b).

Lorsqu'un jeton correspondant à la condition associée à l'arc amont de la transition est présent, alors on a le franchissement de la transition temporisée, le jeton est marqué par sa date de disponibilité future, soit la date actuelle incrémentée de la durée de temporisation associée à la transition (fig. 148c).

Ce jeton se retrouve dans la place aval à la transition, mais il est indisponible tant que l'horloge interne n'atteindra pas la date de marquage du jeton (fig. 148d).

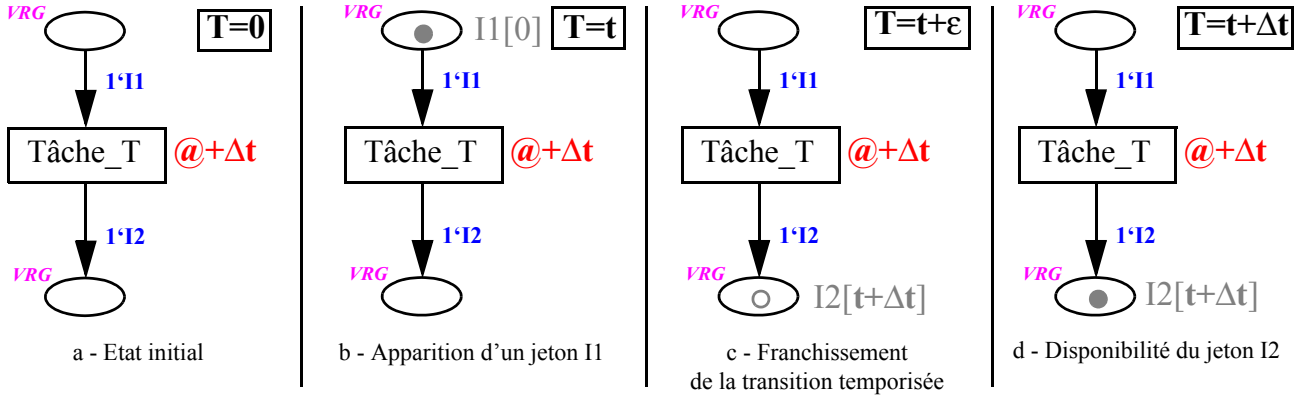


Figure 148 : Comportement de réseau de Petri J-Temporisé

La déclaration des informations temporisées se fait selon le format ci-dessous (fig. 149) :

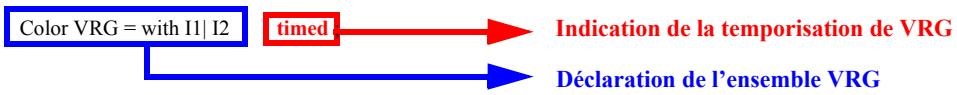


Figure 149 : Extrait des déclarations globales, exemple de déclaration d'informations temporisées

A.2 Exemple de modélisation d'une tâche fonctionnelle complexe

On appelle tâche fonctionnelle complexe, une tâche ayant un comportement qui ne dépend pas uniquement des événements d'entrées. On associe à cette tâche, des données qui peuvent varier en fonction des événements d'entrées. La ou les sorties de cette tâche dépendent également de ces données (fig. 150).

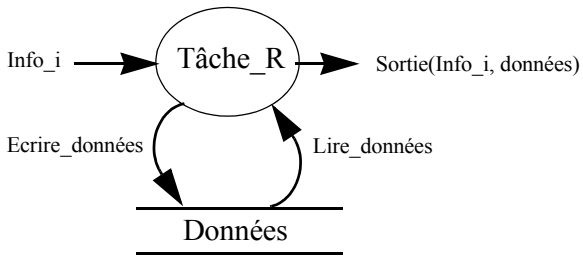


Figure 150 : exemple de tâche fonctionnelle liée à des données

Par exemple, on considère que les données servant à déterminer la sortie sont définies par le graphe d'état (fig. 151).

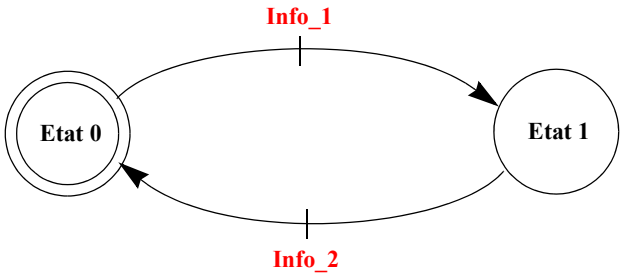


Figure 151 : Graphe des états des données

Les opérations «*lire_données*» et «*écrire_données*» deviennent donc, «*lire_état_courant*» et «*écrire_nouvel_état*».

Nous devons donc définir un état afin de pouvoir facilement le modéliser en réseau de Petri. Soit *ETAT*, l'ensemble des états possibles et *EC* l'état courant. Pour notre exemple, on a :

$$ET = \{ 0; 1 \} \tag{33}$$

On ajoute dans la déclaration globale les lignes suivantes définies par la figure 152.

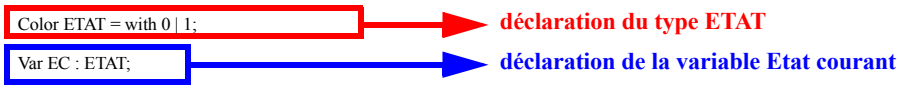


Figure 152 : Extrait des déclarations globales, déclaration de l'état de la production

Afin de réaliser la modélisation d'une tâche complexe, il est nécessaire de réaliser des fonctions permettant d'obtenir :

- un état en fonction d'un événement d'entrée et d'un état
- une sortie en fonction d'un événement d'entrée et d'un état

Ces fonctions dépendent d'événements d'entrée et d'un état, il est donc nécessaire de définir des variables d'entrée des fonctions ci-dessus. Dans notre cas, l'entrée qui provoque l'évolution du graphe d'état est toujours la même, mais ce n'est pas toujours le cas ; aussi nous considérerons le cas général où les

événements déclencheurs sont différents. On définit donc une variable IN correspondant à un événement d'entrée (fig. 153).

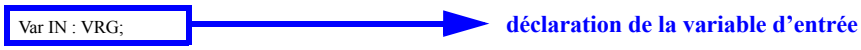


Figure 153 : Extrait des déclarations globales, déclaration de l'état de la production

Nous pouvons maintenant définir les deux fonctions nécessaires au fonctionnement de cet atome réactif. On considère que la sortie dépend de l'état des données et de l'entrée qui sont définis par les règles suivantes :

Tableau 11 : Définition de la sortie en fonction de l'entrée et de l'état

	Entrée = Info_1	Entrée = Info_2
Données = Etat_0	Info_3	Info_4
Données = Etat_1	Info_4	Info_3

Soit l'expression suivante définie avec des opérateurs ternaires :

$$Sortie(IN, EC) = (IN = \text{Info\_1})?((EC = 0)?\text{Info\_3}:\text{Info41}):((EC = 0)?\text{Info\_4}:\text{Info\_3}) \quad (34)$$

D'après le graphe d'état des données (fig. 151), on peut définir l'expression suivante :

$$NouvelEtat(IN, EC) = ((IN = \text{Info\_1})\&(EC = 0))?1:(((IN = \text{Info\_2})\&(EC = 1))?0) \quad (35)$$

Nous réalisons les déclarations de ces fonctions sur la figure 154:

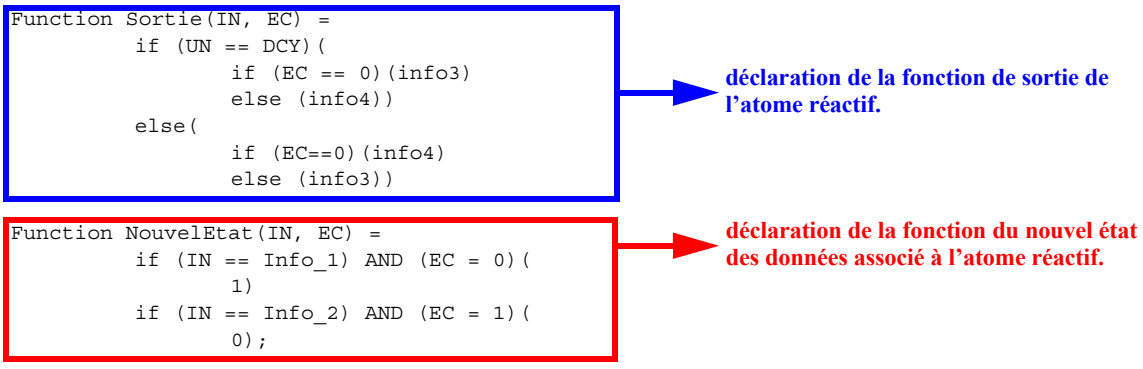


Figure 154 : Extrait des déclarations globales, déclaration de l'état de la production

Ce qui nous permet de construire la figure 155.

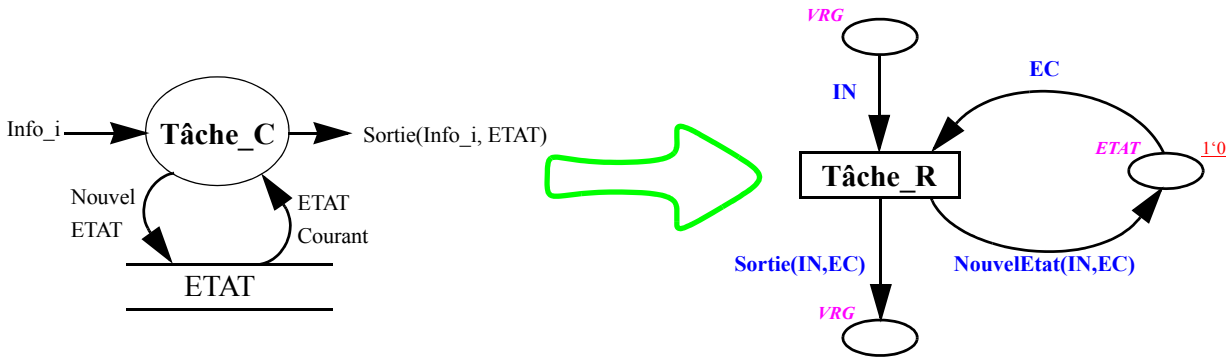


Figure 155 : Modélisation d'une tâche complexe

Dans le cas où plusieurs tâches complexes utilisent les mêmes données, alors chacune des transitions associées à ces tâches devra être connectée à la même place correspondant aux données partagées (fig. 156).

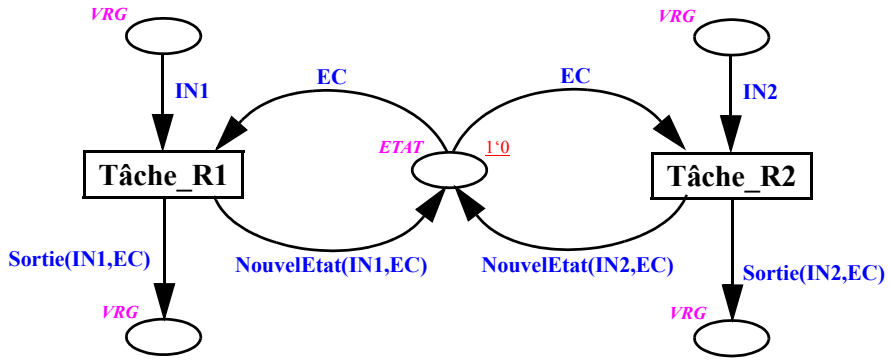


Figure 156 : Tâches réactives partageant des données

La difficulté de modélisation du réseau de Petri d'une tâche complexe passe par l'écriture des fonctions gérant les sorties et le nouvel état. Cette difficulté est atténuées par le fait d'avoir défini le graphe d'état comportemental associé à la tâche fonctionnelle.

Annexe B

Modélisation des «aiguilleurs» pour les équipements du type Automates Programmables Industriels

Tout équipement hôte d'une tâche fonctionnelle doit être suivi d'un aiguillage permettant de marquer le flux d'information afin de permettre son routage, puis d'un triage qui permet d'orienter les informations vers leurs cibles.

La partie aiguillage est réalisée par la fonction *dest* (voir figure 38, page 57), mais cette fonction est exécutée «à temps nul» et ne prend pas en compte les retards induits par le routage au travers des différentes cartes de sorties. Afin de respecter le comportement réel de notre système, il est nécessaire de modéliser ces retards et le triage de ces informations.

Suivant l'orientation que prendra le flux d'information, le matériel mis en oeuvre est différent :

- pour les informations émises vers le système opérant, c'est une carte de sortie qui prend en charge le flux d'information
- tandis que pour les informations émises vers un réseau de communication, c'est la carte coupleur réseau qui doit s'en occuper.

On réalise donc le modèle des cartes traitant les informations (fig. 157).

Les informations ayant pour destination le système opérant auront pour la troisième composante du triplet *(x, so, de)* la valeur zéro (identifiant de l'extérieur). Afin de ne prendre en compte que les informations dont la destination est zéro, nous utilisons un «Garde». C'est à dire que la transition *Orienter vers l'extérieur* ne sera franchissable que si la condition de «Garde» est vraie (dans le cas de l'exemple de la figure 157, lorsque la condition *de=0*). Ces informations doivent traverser une carte de sorties dont le comportement est modélisée par un retard pur *TCS*.

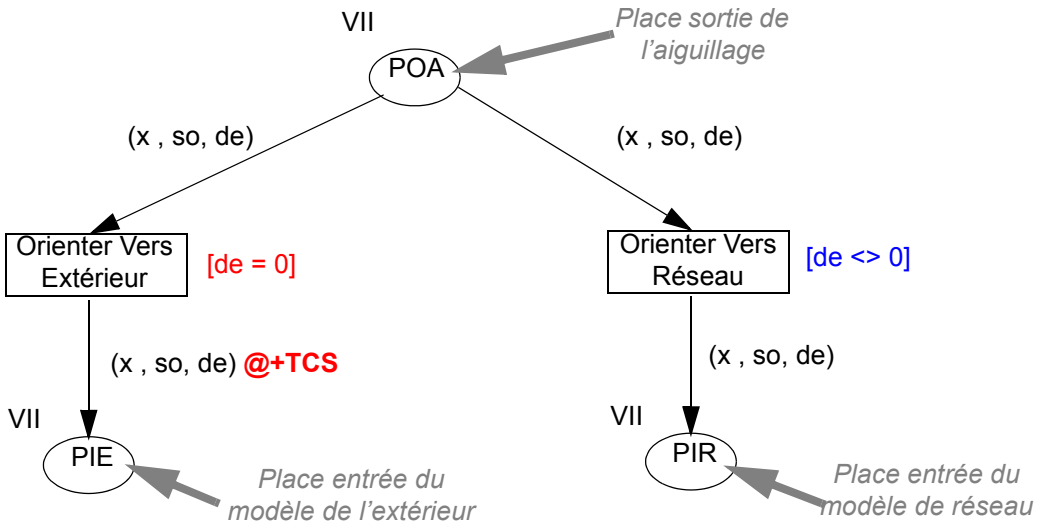


Figure 157 : Modèle réseau de Petri des cartes (sorties et réseau)

Pour le cas des informations destinées à être envoyées vers un réseau, c'est-à-dire l'ensemble des informations n'ayant pas pour destination le milieu extérieur, nous utilisons également un «Garde» dont la condition est le complément de la condition de sélection des informations vers l'extérieur : $\overline{(de = 0)} = (de \neq 0) = (de < \ > 0)$. Aucun temps de retard n'est spécifié pour la traversée d'une carte réseau, le modèle du réseau prenant en compte ce délais de manière intrinséque.

Dans le cas où l'on a plusieurs cartes réseau, il est nécessaire de réaliser autant d'embranchement qu'il y a de réseaux cibles (fig. 158). Il faut alors déterminer quel est le réseau cible pour une information donnée. L'utilisation des gardes permet de résoudre ce problèmes mais les conditions des «gardes» de transition deviennent alors plus complexes si l'on désire garder le coté générique.

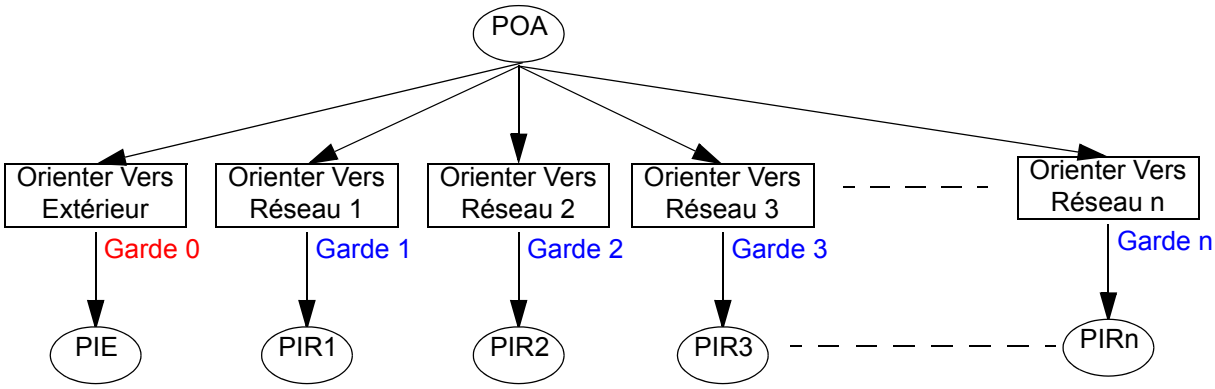


Figure 158 : Structure d'un aiguilleur avec plusieurs cartes réseau

Si on prend l'exemple d'un API ayant deux cartes coupleur réseau dont une est maître sur une des deux réseaux et l'autres est esclave sur le second réseau, on obtient un modèle ayant 3 transitions.

Les «Gardes» devront orienter les différentes informations selon leurs destinations. Ainsi dans le cas décrit par la figure 159 :

- les informations pour lesquelles *de=0* sont dirigées vers l'extérieur (transition de gauche) ;
- les informations vérifiant *de<>0 ET so=2* sont dirigées vers le réseau *RX1* (place *PIER1*, Place "In" Esclave Réseau n°1) ;
- les informations vérifiant *de<>0 ET so<>2* sont dirigées vers le réseau *RX2* (place *PIMR2*, Place "In" Maître Réseau n°2)
- dans le cas où une information ne correspond pas aux conditions ci-dessus, son jeton reste dans la place *POA*. Il est conseillé de vérifier que la somme logique des conditions de «Garde» précédents soit égale à 1 pour éviter d'avoir des informations «bloquées».

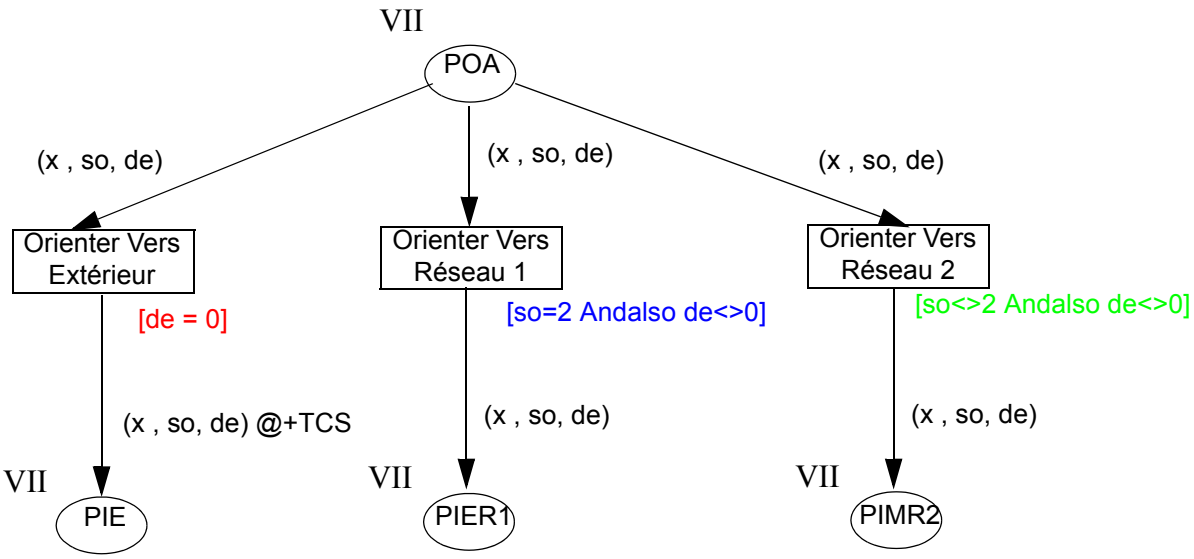


Figure 159 : Modèle réseau de Petri des cartes (sorties et réseau)

Annexe C

Modélisation détaillée d'une Architecture Opérationnelle

Cette annexe développe de manière détaillée le traitement de l'exemple support à la validation de notre démarche, présentée dans le Chapitre 5.

Avant la phase de construction du modèle de l'Architecture Opérationnelle, quelques travaux préparatoires peuvent être réalisés afin de faciliter les phases de conception du modèles réseaux de Petri. Nous présenterons donc dans ce paragraphe, les travaux préparatoires correspondants aux architectures matérielle, fonctionnelle et opérationnelle.

C.1 Architecture Matérielle

Afin de différencier les différents équipements matériels, on leur associe un identifiant. Tout équipement de traitement doit en recevoir un, même s'il ne traite aucune tâche observée. Tout équipement de communication qui a une connection avec l'extérieur doit en avoir un aussi. La (fig. 160) indique les différents identifiants de l'Architecture Matérielle.

Les autres équipements de communication n'ont pas d'identifiant car ils ne servent que de «passerelle» entre deux équipements identifiés. L'environnement extérieur à l'architecture de commande est identifié par le nombre 0.

Pour le réseau Uni-Telway, chaque équipement connecté possède un à trois ports de liaison (nommés Ad0, Ad1 et Ad2). Ces ports de liaison correspondent aux différentes parties de l'équipement (respectivement système, traitement, gestion). Lorsque le maître scrute ses esclaves, il scrute en fait tous les ports de liaison de chacun des équipements esclaves. Il est donc nécessaire de différencier ces différents ports de liaison. La figure 160 présente les identifiants des différents ports de liaison des équipements connectés au réseau Uni-Telway. Le maître du réseau a pour identifiant de port de liaison 0.

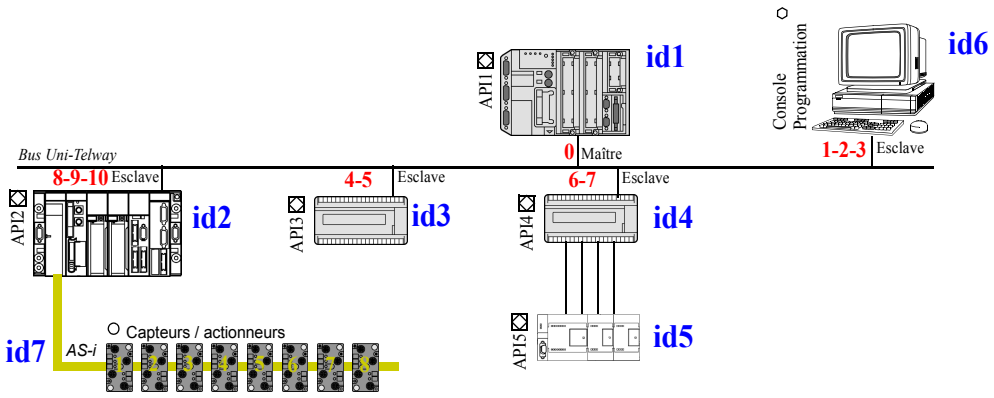


Figure 160 : Numérotation des API et des ports de liaisons avec le réseau UniTELWAY

La table des flux informationnels qui est nécessaire pour définir le routage des informations au travers de l'architecture matérielle nécessite de connaitre le graphe structurel correspondant à l'architecture matérielle. Ce graphe est construit en reliant les différents équipements de traitements ou de communication (fig. 161).

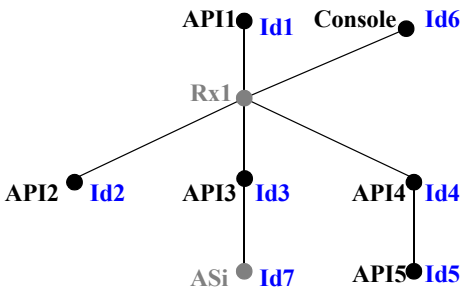


Figure 161 : Graphe structurel de l'architecture matérielle

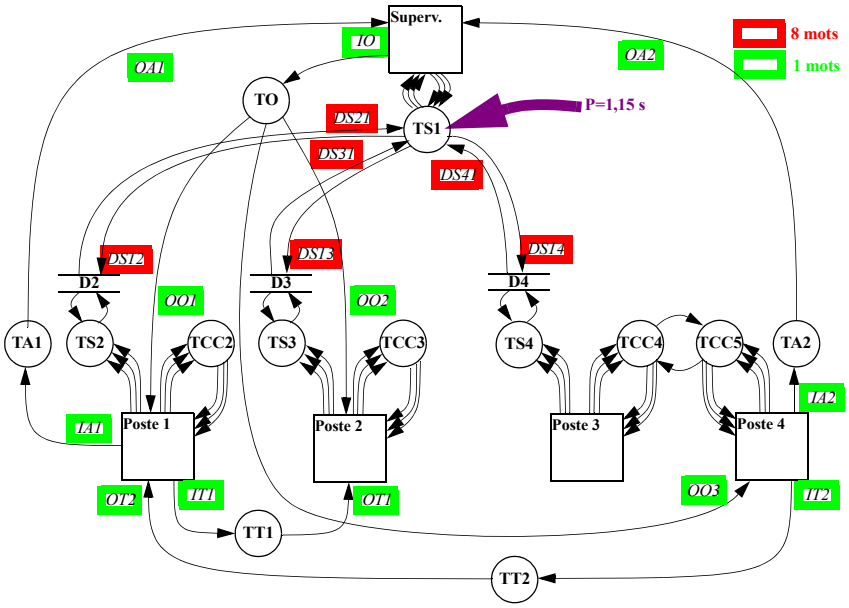
C.2 Architecture Fonctionnelle

On peut distinguer deux types de tâches fonctionnelles : les tâches qui correspondent aux informations que l'on souhaite observer et celles qui correspondent aux informations qui «chargent» l'Architecture Opérationnelle.

Les tâches fonctionnelles correspondant aux informations observées ont un comportement de consomation-génération. Leur traduction en modèle réseau de Petri ne posera pas de problème. c'est le cas des tâches fonctionnelles TO, TA1, TA2, TT1 et TT2.

Pour les tâches fonctionnelles qui chargent l'Architecture Opérationnelle, on peut en distinguer deux sortes : celles qui chargent uniquement un équipement de traitement ou celles qui chargent tout ou partie de l'architecture.

Les tâches qui chargent uniquement un équipement de traitement ne seront pas modélisées par un réseau de Petri mais seront prise en compte pour le paramétrage du temps de traitement associé à l'équipement. c'est le cas des tâches fonctionnelles TCC2, TCC3, TCC4, TCC5, T3, T4, TS2, TS3, TS4 et TS5.



Rappel de la figure 123 : Architecture Fonctionnelle

Les tâches qui chargent tout ou partie de l'Architecture Opérationnelle doivent être complètement modélisées. Ces modèles étant spécifiques, il est nécessaire de réaliser des nouveaux modèles. c'est le cas de la tâche fonctionnelle TS1 et des données D1, D2 et D3.

Durant cette phase préparatoire, il convient de définir clairement le comportement de ces tâches afin faciliter la réalisation des modèles réseau de Petri.

La tâche fonctionnelle TS1 correspond aux envois successifs des demandes d'écriture et de lecture de tables de données de supervision. Le compte-rendu d'un échange de table est attendu avant de commencer le suivant. La tâche fonctionnelle TS1 est périodique avec une période *PPolling* définie de 1,15 secondes. Toutefois, une nouvelle série d'envoi d'échange de tables ne commencera que si la précédente est terminée. On définit le comportement de TS1 par le réseau de Petri défini par la figure 162 :

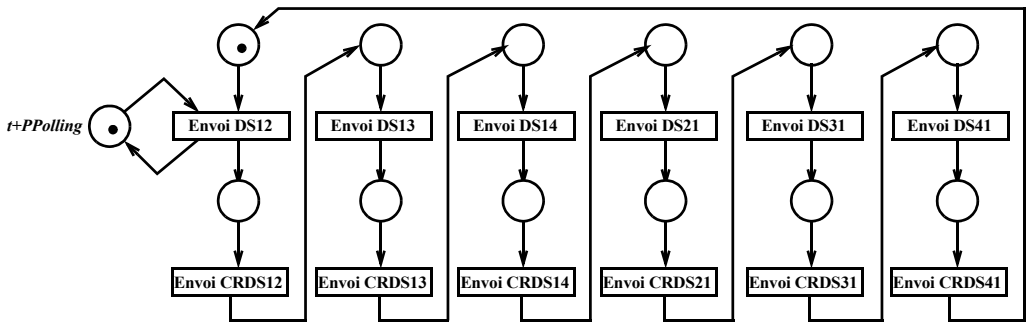
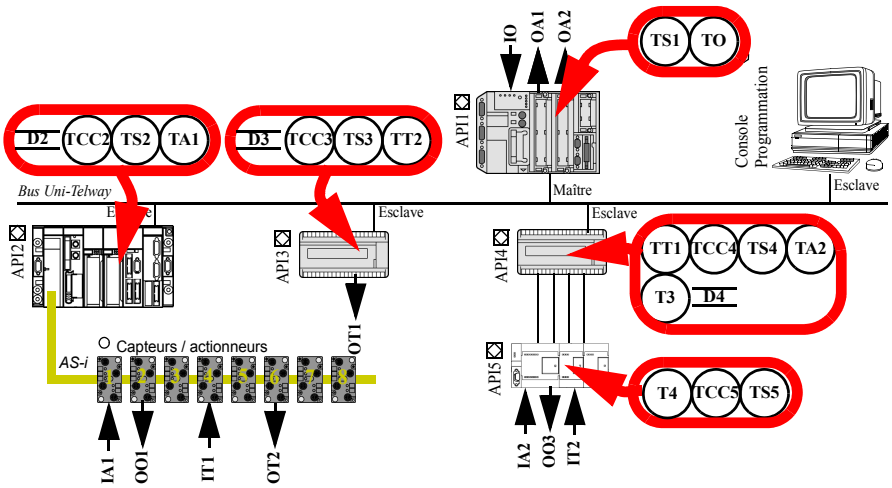


Figure 162 : Réseau de Petri de la tâche fonctionnelle TS1

Les données Di correspondent aux zones d'informations correspondant aux données de supervision des différents équipements. Etant donné que les informations observées n'en tiennent pas compte, leur comportement modélisé sera réduit au minimum : absorber chaque demande d'échange de table, car le compte-rendu est automatiquement réalisé par le modèle réseau.

C.3 Architecture Opérationnelle

Etant donné que l'Architecture Opérationnelle définit les projections des tâches fonctionnelles sur les équipements matériels, certaines tâches fonctionnelles ont leurs entrées ou sorties qui ne sont physiquement pas présentes dans l'équipement matériel sur lequel elles sont implantées. Par exemple : la tâche fonctionnelle TT2 est associée à l'API3 alors que son entrée IT2 est physiquement sur l'API5 et que sa sortie est physiquement sur le réseau AS-i connecté à l'API2.



Rappel de la figure 124 : Projection de l'Architecture Fonctionnelle sur l'Architecture Matérielle

Comme l'a définit dans sa thèse B. DENIS [DENIS 1994], il est nécessaire d'ajouter des tâches techniques qui ont pour but de transmettre les informations depuis leurs entrées physiques vers le lieu d'implantation de la tâche fonctionnelle qui les traite puis de transmettre les informations résultantes vers leurs sorties physiques. Dans notre cas celà donne (fig. 163):

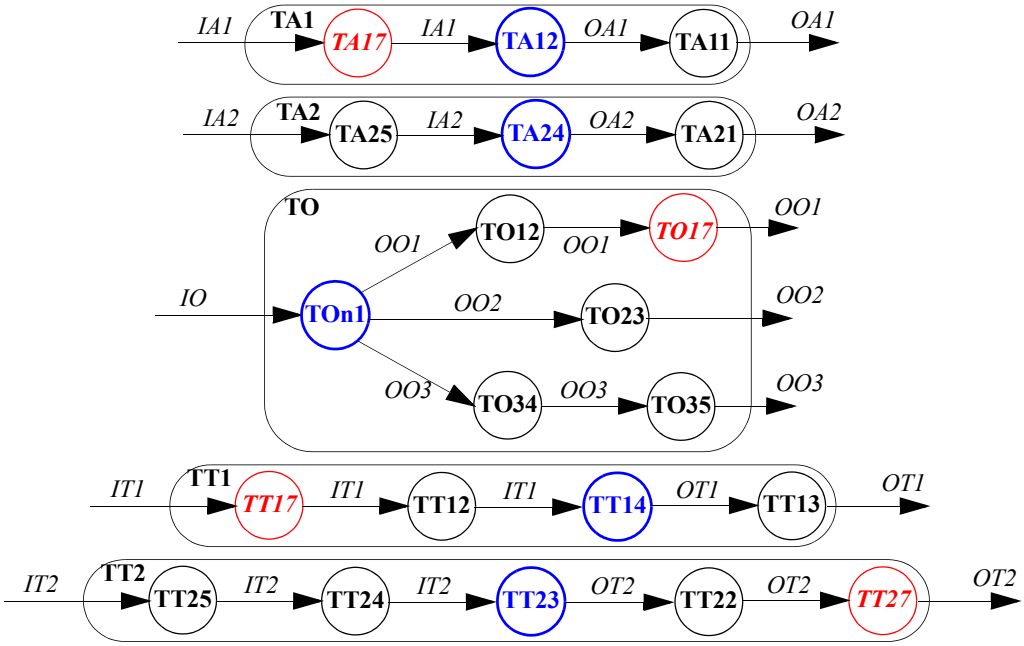


Figure 163 : Ajouts des tâches techniques

Les tâches en bleu correspondent aux tâches fonctionnelles, les autres tâches sont des tâches techniques. Celles en rouge correspondent aux tâches techniques du réseau AS-i, elles n'ont pas à être décrites car elles sont déja réalisées par le modèle du réseau AS-i.

L'architecture Opérationnelle complétées par les tâches techniques est donc représentée par la figure 164.

Les tâches représentées en magenta sont celles qui correspondent à une charge d'un équipement de traitement et qui ne seront pas modélisées, les tâches en vert sont celles qui chargent l'Architecture Opérationnelle et qui nécessitent une modélisation spécifique.

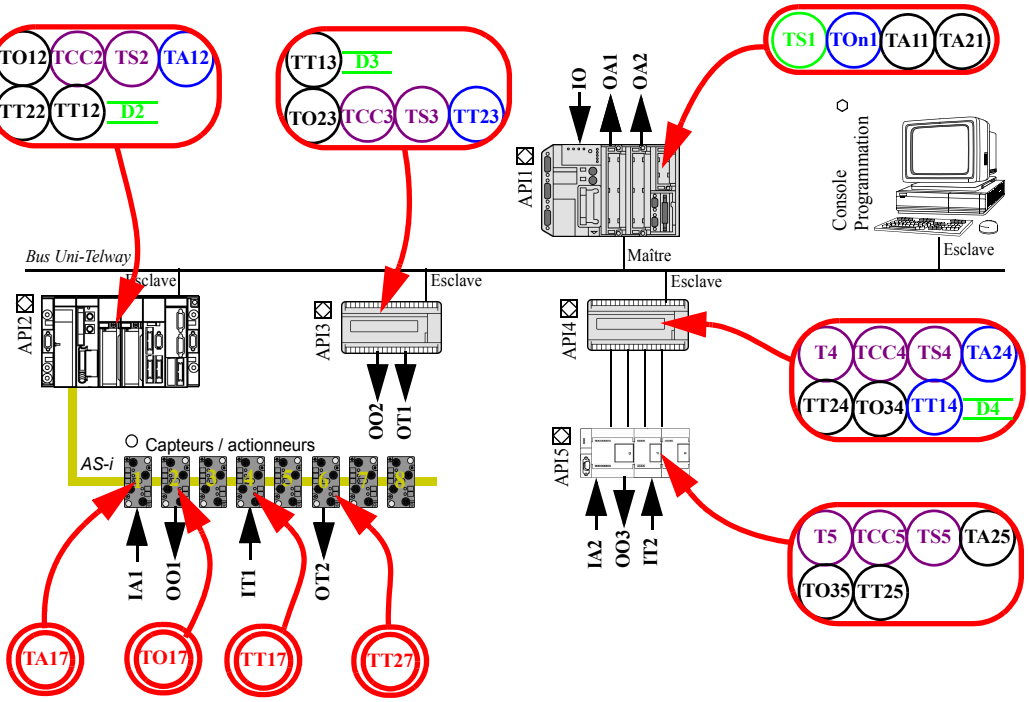


Figure 164 : Architecture Opérationnelle avec tâches techniques

A partir de cette projection, on peut réaliser la table des projections des tâches fonctionnelles sur les différents équipements de traitement. On ne tiendra compte que des tâches qui participent aux chaînes critiques correspondant aux performances évaluées.

Tableau 12 : Table des projections des tâches fonctionnelles

identifiant matériel	tâche fonctionelle
1	TS1
	TOn1
	TA11
	TA21
2	TO12
	TA12
	TT22
	TT12
	D2

identifiant matériel	tâche fonctionelle
3	TT13
	TO23
	TT23
	D3
4	TA24
	TT24
	TO34
	TT14
	D4
5	TA25
	TO35
	TT25

Maintenant que les tâches techniques sont définies et que le graphe structurel de l'architecture matérielle a été réalisé, nous pouvons concevoir la table de flux informationnels.

Cette table contient pour chaque information provenant d'une tâche source donnée, sa tâche destination (tableau 13).

Tableau 13 : Table des Flux informationnels

Information	tâche source	tâche destination
DS12	TS1	D2
DS13	TS1	D3
DS14	TS1	D4
DS21	D2	TS1
DS31	D3	TS1
DS41	D4	TS1
IA1	EXT	TA17
	TA17	TA14
OA1	TA12	TA11
	TA11	EXT
IA2	EXT	TA25
	TA25	TA24
OA2	TA24	TA21
	TA21	EXT
IO	EXT	TOn1
OO1	TOn1	TO12
	TO12	TO17
	TO17	EXT

Information	tâche source	tâche destination
OO2	TOn1	TO23
	TO23	EXT
OO3	TOn1	TO34
	TO34	TO35
	TO35	EXT
IT1	EXT	TT17
	TT17	TT12
	TT12	TT14
OT1	TT14	TT13
	TT13	EXT
IT2	EXT	TT25
	TT25	TT24
	TT24	TT23
OT2	TT23	TT22
	TT22	TT27
	TT27	EXT

Cette table permettra par la suite de définir la fonction de routage des informations au travers du modèle de l'architecture.

C.4 Construction du modèle

Dans ce paragraphe nous passerons en revue toutes les grandes étapes de la construction du modèle réseau de Petri de l'Architecture de Simulation.

C.4.1 Déclarations globales de base

Dans un premier temps, on définit deux couleurs très courantes dans les modèles : la couleur NRM correspondant à un jeton simple "e" temporisé ainsi que la couleur BINT correspondant à un nombre entier temporisé.

```
color NRM  = with e timed;
color BINT = int timed;
```
Déclarations globales

Figure 165 : Extrait de déclarations globales, définition des couleurs NRM et BINT

De plus, il est nécessaire de définir l'ensemble des informations circulant dans l'architecture dans les déclarations globales du RdP au travers d'une couleur VRG. La couleur VRG comprend l'ensemble des informations circulant dans l'architecture, donc tous les messages ainsi que tous leurs comptes-rendus associés :

```
color VRG  = with      IA1| OA1| IA2| OA2|                    Déclarations globales
                       IT1| OT1| IT2| OT2|
                       IO | OO1| OO2| OO3|
                       DS12| DS13| DS14| DS21| DS31| DS41|
                       CRIA1| CROA1| CRIA2| CROA2|
                       CRIT1| CROT1| CRIT2| CROT2|
                       CRIO | CROO1| CRIO2| CROO3|
                       CRDS12| CRDS13| CRDS14| CRDS21| CRDS31| CRDS41 timed;
```

Figure 166 : Extrait de déclarations globales, définition de la couleur VRG

C.4.2 Modèle de l'Architecture Matérielle

Etant donné que l'Architecture Matérielle est composée d'équipements matériels, il est nécessaire de créer une page contenant chaque modèle RdP de ces équipements matériels. De plus afin d'avoir une vision globale sur une page que nous nommerons "Architecture", nous placerons l'ensemble des équipements matériels par l'intermédiaire de transitions de substitution. On obtient donc la figure 167.

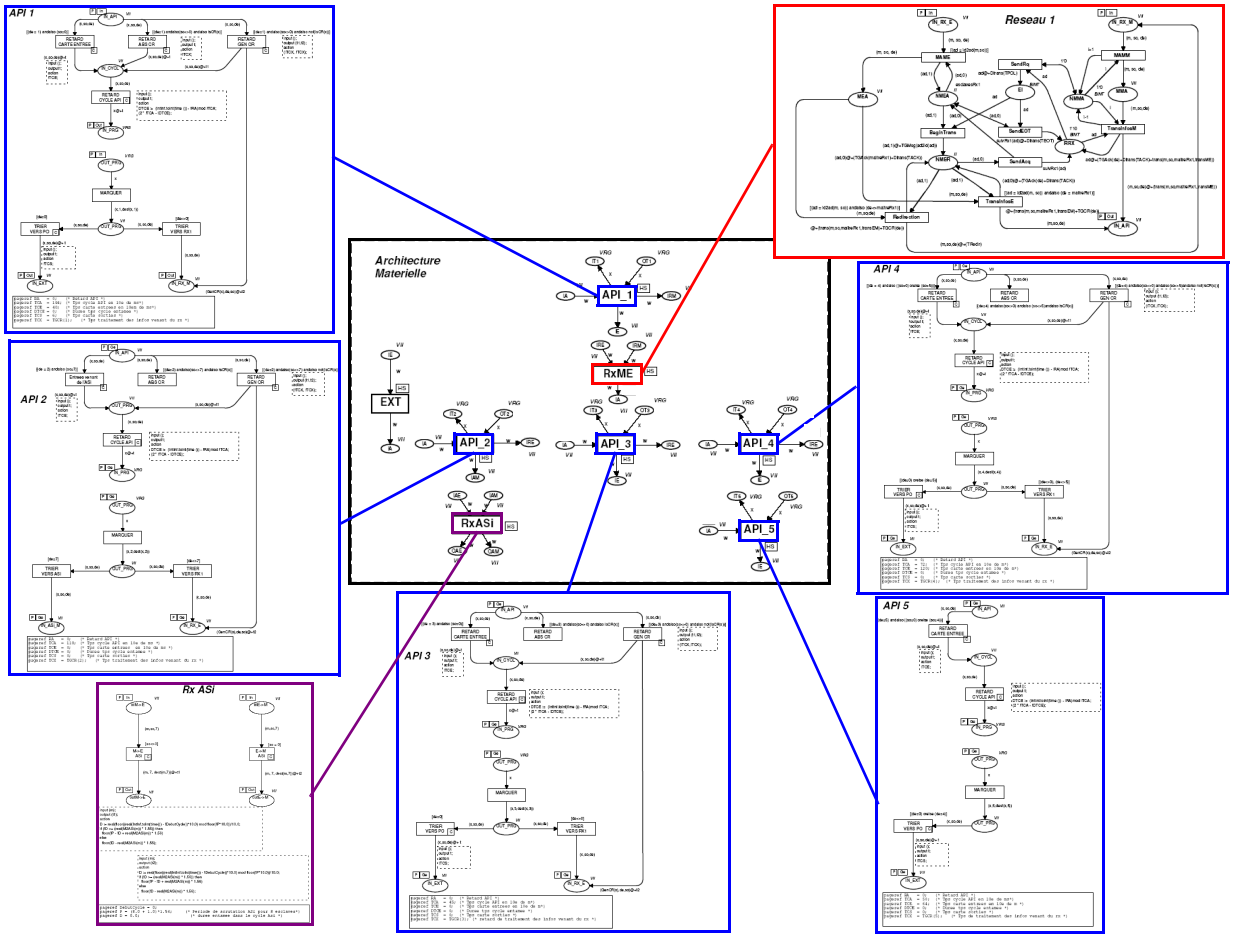


Figure 167 : Construction du modèle de l'Architecture Matérielle

De plus, les modèles d'équipements matériels étant reliés entre eux, on fusionne les places qui sont communes aux différents modèles sur la page "Architecture". Les places restantes correspondent aux connections des modèles des tâches fonctionnelles (fig. 168).

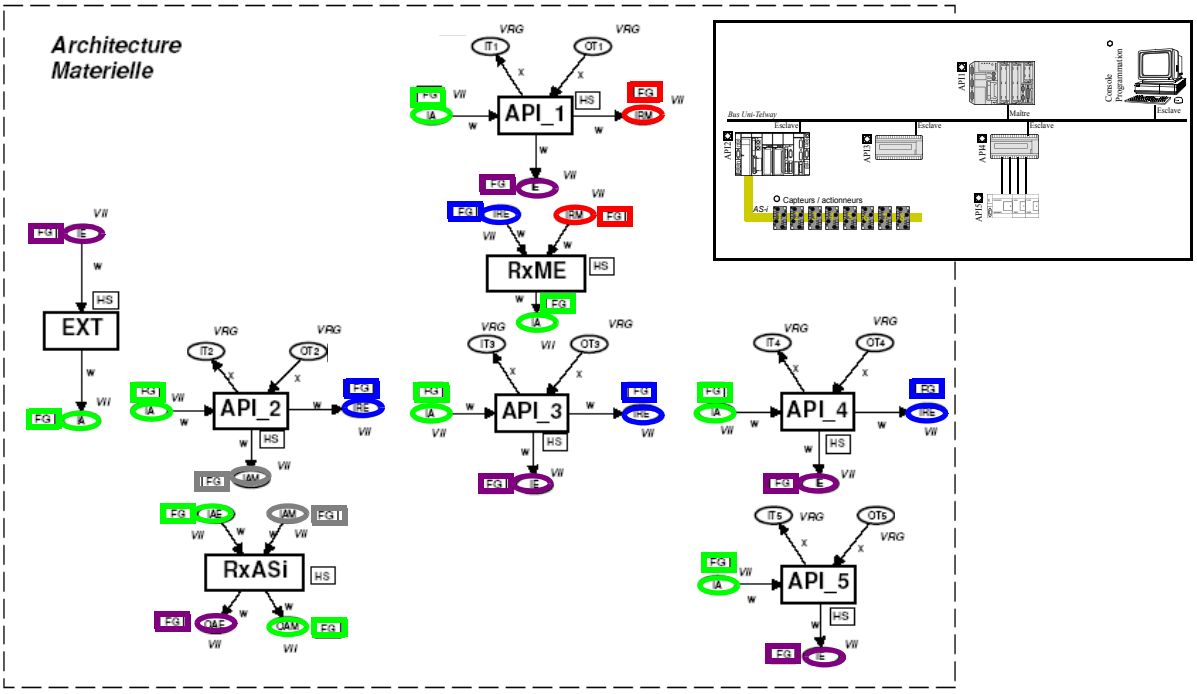


Figure 168 : Fusion des places communes pour l'Architecture Matérielle

Il est maintenant nécessaire de paramétrer les différents modèles des équipements matériels.

Pour les équipements de traitement de type API, il convient de paramétrer la période du cycle API, les temps de retards des cartes d'entrées/sorties, les temps de retournement pour les réponses réseaux. Chaque modèle API possède une zone de déclaration locale qu'il convient de renseigner avec les paramêtres de l'équipement (fig. 169).

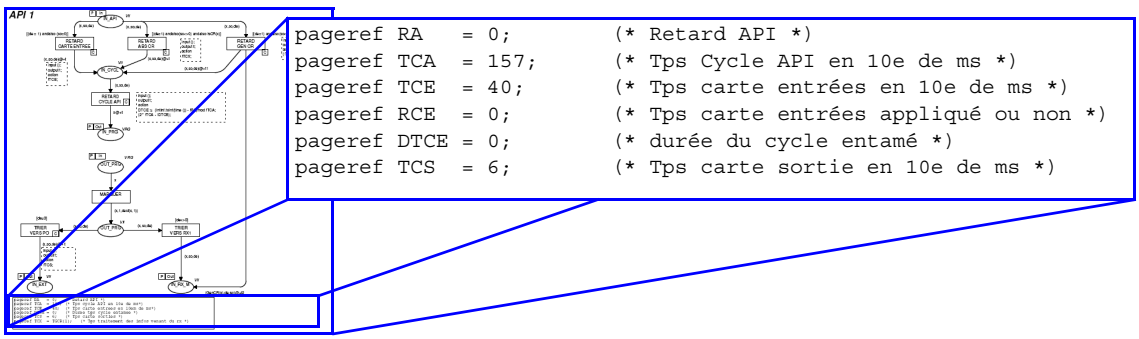


Figure 169 : Paramétrage d'un modèle de l'API 1

Certaines modifications structurelles sur les aiguillages pré ou post traitement peuvent être nécessaires suivant la configuration matérielle de l'API (nombre de cartes d'entrée, coupleurs réseau, …). Dans le cas du modèle de base d'un API, la carte réseau, transmets les comptes-rendus de communications dans le cas où le traitement aurait un besoin de ces informations. Or dans le cas des API2, API3 et API4, cette fonctionnalité n'est pas nécessaire ; il convient donc de supprimer l'arc transmettant le compte-rendu ainsi que le code associé à cette transition (fig. 170).

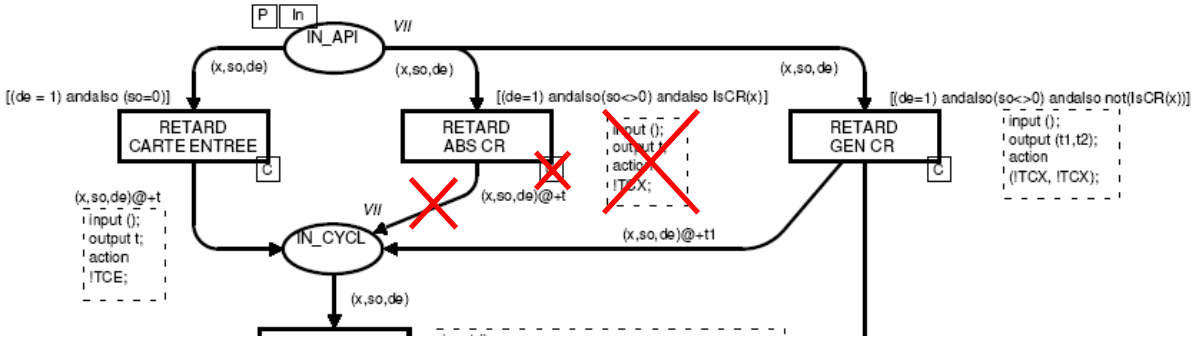


Figure 170 : Exemple de modification structurelle de l'API2

Dans le cas de l'API5, il n'y a pas de carte réseau, le modèle générique doit donc être modifié (fig. 171).

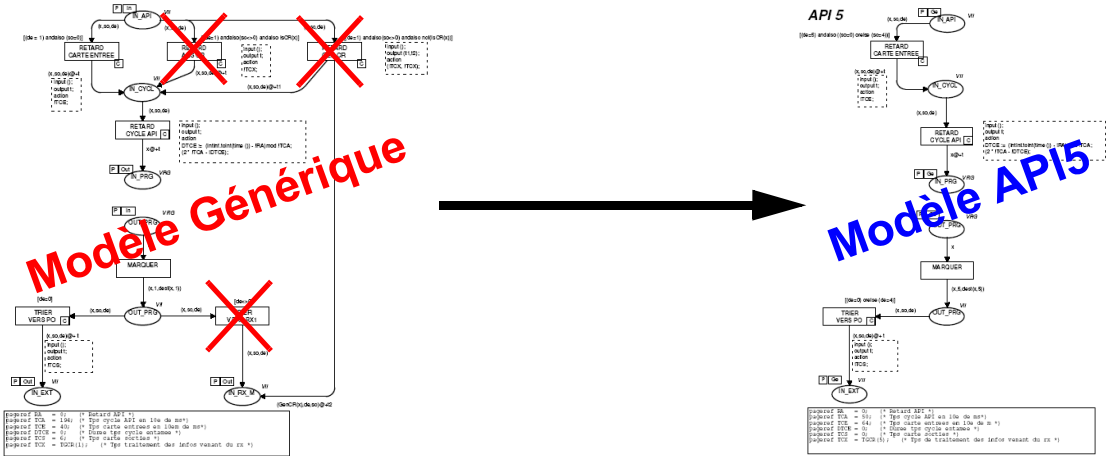


Figure 171 : Exemple de modification structurelle pour l'API5

De plus dans le cas de l'interconnection entre l'API4 et l'API5, il est nécessaire de modifier les conditions associées aux «Gardes» des transitions qui gèrent les informations issues des cartes d'entrées ou de sorties. Par exemple, la carte de sortie de l'API4 traite les informations qui sont destinées à l'extérieur (identifiant 0) et l'API5 (identifiant 5), le reste est traité par la carte réseau (fig. 172).

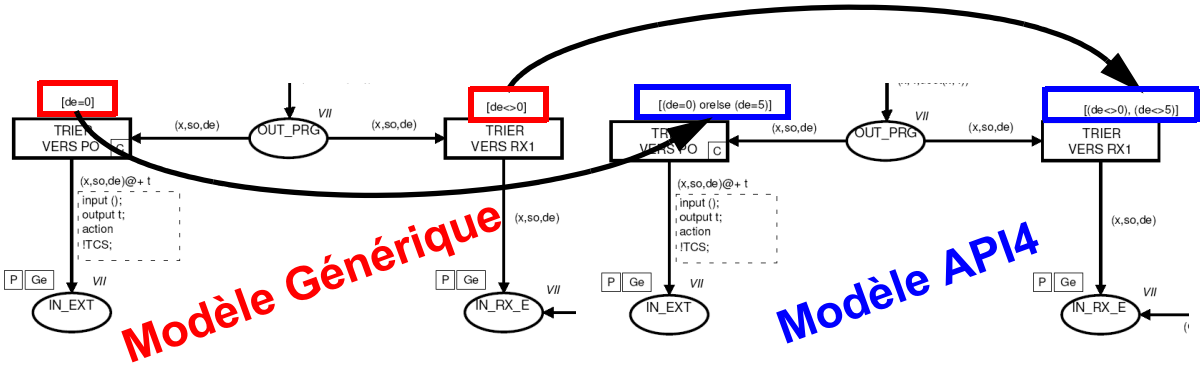


Figure 172 : Exemple de modification structurelle pour l'API4

pour le cas des modèles API avec une carte réseau, il est nécessaire de définir la fonction *GenCR()* qui génère le compte rendu d'une information. Par exemple, l'information *IT1* arrivant sur l'API4 renverra le compte-rendu *CRIT1* à l'API2 qui en était la source (fig. 173).

Déclarations globales

```
fun GenCR(m) = case m of
          IA1 => CRIA1 |OA1 => CROA1 |IA2 => CRIA2 |OA2 => CROA2
         |IT1 => CRIT1 |OT1 => CROT1 |IT2 => CRIT2 |OT2 => CROT2
         |IO  => CRIO  |OO1 => CROO1 |OO2 => CROO2 |OO3 => CROO3
         |DS12 => CRDS12 |DS13 => CRDS13 |DS14 => CRDS14
         |DS21 => CRDS21 |DS31 => CRDS31 |DS41 => CRDS41
         |_ => BUG;
```

Figure 173 : Fonction GenCR(x) de génération de compte-rendus

Pour les équipements matériels de communication, il est nécessaire de paramétrer le Bus AS-i et le réseau Uni-Telway.

Dans le cas du Bus AS-i, pour permettre son utilisation, il est nécessaire de déterminer sur quel esclave AS-i les informations seront écrites ou lues. C'est le rôle de la fonction *M2ASi( )* qui renvoi le numéro d'esclave AS-i en fonction de l'information (fig. 174).

Déclarations globales

```
fun M2ASi(x) = case x of
          IA1 => 1
         |OO1 => 2
         |IT1 => 4
         |OT2 => 6;
```

Figure 174 : Fonction M2ASi(x) de correspondance Message-n°esclave ASi

Il faut de plus paramétrer le nombre d'esclaves AS-i connectés qui permet de définir le temps de cycle de scrutation de l'ensemble des périphériques connectés, dans notre exemple nous avons huit esclaves AS-i (fig. 175).

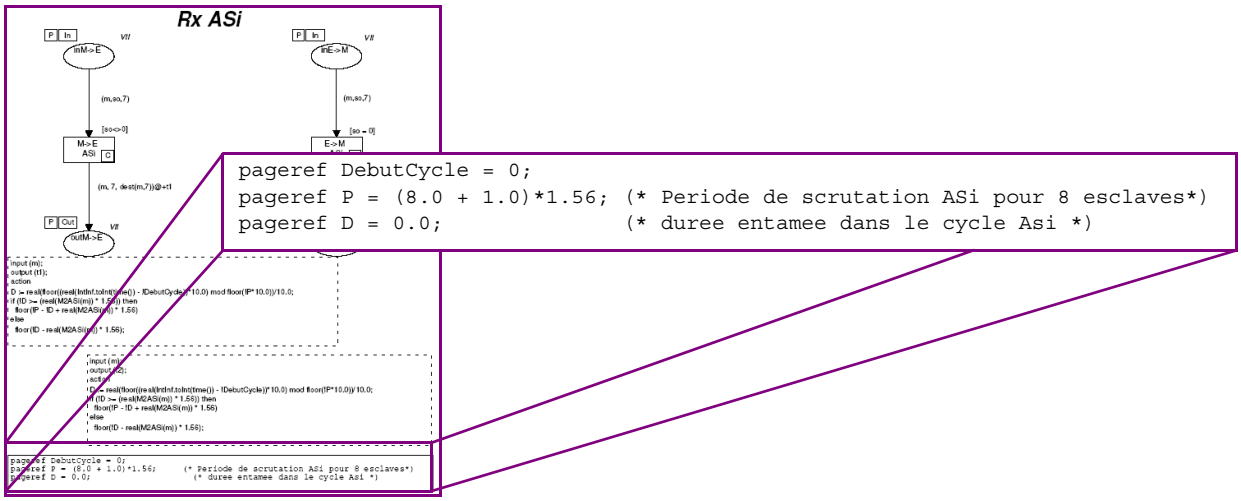


Figure 175 : Paramétrage d'un modèle AS-i

Pour le réseau Uni-telway, le modèle réseau de Petri est générique le paramétrage s'effectue par le biais de fonctions et constantes. Ce paramétrage se découpe en deux parties : les paramètres de scrutation des esclaves et les paramètres de transmission des messages.

La scrutation des esclaves est définie par (fig. 176):

- la liste *esclavesRx1* des ports de liaison des esclaves à scruter,
- l'identifiant *maitreRx1* du maître sur le réseau,
- la fonction *SuivRx1* permettant de déterminer le port suivant à scruter
- des fonctions *id2ad* et *ad2id* permettant de passer respectivement d'un identifiant matériel à un numéro de port de liaison et inversement.

Déclarations globales

```
(* Declarations pour le Rx1 *)
val maitreRx1 = 1;       (* identifiant du maitre du Rx1 *)
val esclavesRx1 = 1`(1,0) ++ 1`(2,0) ++ 1`(3,0) ++ 1`(4,0) ++ 1`(5,0) ++ 1`(6,0)
                  ++ 1`(7,0) ++ 1`(8,0) ++ 1`(9,0) ++ 1`(10,0);
fun suivRx1(ad) =        (* fonction pour déterminer l'esclave suivant à scruter *)
          if (ad = 10) then 1`1 else 1`(ad+1);
fun id2ad(m, so) =       (* fonction permettant de savoir quel port est envoyé le message *)
          case so of
                   2 => if IsCR(m) then 8 else 9
                  |3 => if IsCR(m) then 4 else 5
                  |4 => if IsCR(m) then 6 else 7
                  |_ => 0;
fun ad2id(ad) =          (* fonction permettant de savoir d'identifiant en fonction du port *)
          case ad of
                   0 => 1
                  |1 => 6 |2 => 6 |3 => 6
                  |4 => 3 |5 => 3
                  |6 => 4 |7 => 4
                  |8 => 2 |9 => 2 |10 => 2
                  |_ => 0;
```

Figure 176 : Paramètres et fonctions servant à la scrutation des esclaves

Pour le paramétrage des transmissions des informations, il faut définir (fig. 177):

- la valeur *VRx1* de la vitesse de transmission des données en Bauds,
- la valeur *TailleTrameNiv1* de la taille d'une trame niveau 1,
- la fonction *Dtrans* calculant le temps de transmission pour une taille donnée,
- la fonction *trans* calculant le temps de transmission pour une information suivant la source et sa destination,
- les fonctions *TGMsg*, *TGAck* et *TGCR* indiquant le temps de retournement des différents équipements connectés au réseau Uni-Telway pour respectivement la préparation d'un message à envoyer, la préparation d'un acquitement après reception de message et la préparation d'un compte-rendu suite à la reception d'un message.
- la valeur *TRedir* correspondant au temps de traitement par le maître d'une transmission d'esclave à esclave.

Toutes ces fonctions ont une structure générique, il n'est donc pas nécessaire de les recréer, il suffit de les adapter aux contraintes de l'Architecture Matérielle.

Le modèle de l'Architecture Matérielle est donc maintenant réalisé. La phase suivante est la modélisation de l'Architecture Fonctionnelle.

Déclarations globales

```
val VRx1 = 19200          (* vitesse de transmission en bauds *)
val TailleTrameNiv1 = 11 (* taille d'une trame simple au niveau 1 *)
(* fonction determinant le temps de transmission des messages au travers du reseau Rx1 *)
fun Dtrans(t) = floor(real(t)*10000.0/(real(VRx1)/real(TailleTrameNiv1)));
fun trans(m, so, ma, senstrans) =
          if ((so <> ma) andalso (senstrans = transME)) then
                  Dtrans(taille(m)+1)
          else
                  Dtrans(taille(m));
(* Temps de generation d'un message suite a une scrutation du maitre *)
fun TGMsg(id) = case id of
          2 => 50 |3 => 5 |4 => 5 |_ => 0;
(* Temps de generation d'un acquitement suite a la reception d'un message *)
fun TGAck(id) = case id of
          1 => 20 |2 => 80 |3 => 80 |4 => 80 |_ => 0;
(* Temps de generation d'un compte-renduaprès envoi de l'acquitement de reception*)
fun TGCR(id) = case id of
          1 => 60 |2 => 60 |3 => 80 |4 => 80 |_ => 0;
(* durée du traitement d'une redirection dans le maître avant envoi vers la destination *)
val TRedir = Dtrans(TACK)+1;
```

Figure 177 : Paramètres et fonctions servant à la transmission des messages

C.4.3 Modèle de l'Architecture Fonctionnelle

Comme nous l'avions indiqué dans le paragraphe C.2, les tâches composants l'Architecture Fonctionnelle seront traitées de différentes manières suivant qu'elles sont observées, ou qu'elles ne servent qu'à la charge d'un unique équipement ou d'une partie de l'Architecture Opérationnelle.

Les tâches fonctionnelles correspondants aux informations que nous observons, seront modélisées par des structures simples composées de deux places et d'une transition. L'arc amont consommant les informations d'entrées de la tâche et l'arc aval produisant les informations de sortie de la tâche. L'ensemble des tâches fonctionnelles observées (TA12, TA24, TOn1, TT14, TT23) ainsi que toutes les tâches techniques de routage de l'informations (TA11, TA25, TA21, TO12, TO23, TO34, TO35, TT12, TT13, TT25, TT24, TT22) sont modélisées de cette manière. La figure 178 présente les modèles des tâches TA12 et TOn1.

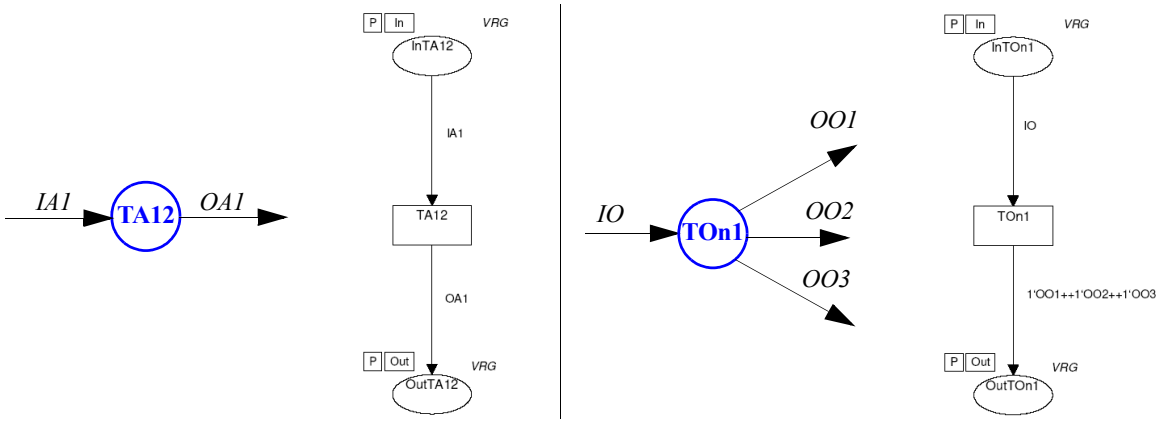


Figure 178 : Modèles des tâches fonctionnelles TA12 et TOn1

Les tâches fonctionnelles qui ne chargent qu'un unique équipement de traitement (TCC2, TCC3, TCC4, TCC5, T3, T4, TS2, TS3, TS4, TS5), n'auront pas de modèle réseau de Petri mais seront prises en

compte dans le paramétrage des temps de cycle de ces équipements (voir le paragraphe précédent pour le réglage du temps de cycle d'un API).

Les tâches fonctionnelles chargeant tout ou partie de l'architecture doivent êtres modélisées complètement. C'est le cas de la tâche fonctionnelle TS1.

La figure 162 présentait le modèle comportemental de la tâche TS1 que nous avions défini. Toutefois ce modèle est général et ne peut s'insérer directement dans le modèle de l'Architecture. En effet, les envois de comptes rendus sont gérés par les modèles des cartes réseaux des API. Il est donc nécessaire de modifier ce modèle. De plus nous profiterons des possibilités de coloration des jetons et des possibilités de codage de fonctions en langage ML qui nous permettent ainsi d'obtenir un modèle beaucoup plus compact (fig. 179).

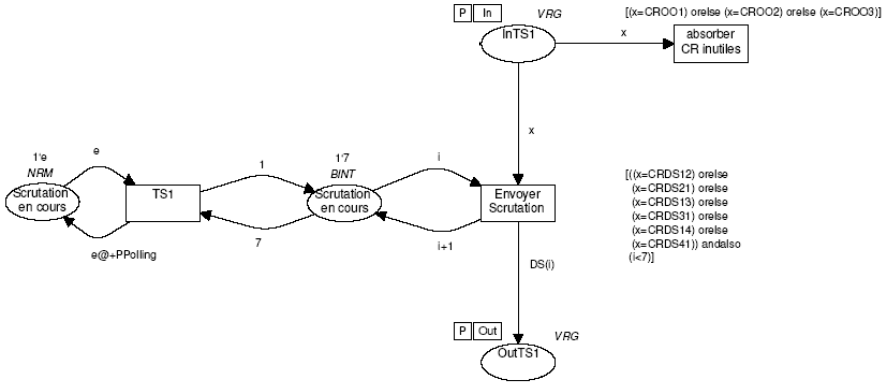


Figure 179 : modèle RdP de la tâche fonctionnelle TS1

Pour les données Di, le modèle consiste en fait à absorber les demandes d'échange de table DSij pour ne pas avoir des jetons inutiles dans les modèles (fig. 180).

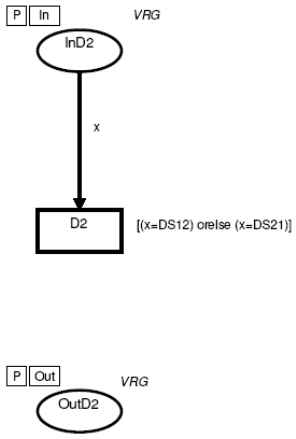


Figure 180 : Modélisation de la donnée D2

Les modèles des tâches fonctionnelles sont maintenant définis indépendament les uns des autres. Il faut que ces tâches soient présentes sur la page de l'Architecture. Pour cela, tous les modèles des tâches

fonctionnelles sont réalisés sur des pages indépendantes. Puis sur la page de l'Architecture on représente toutes ces tâches par l'intermédiaire de tâches de substitution (fig. 181).

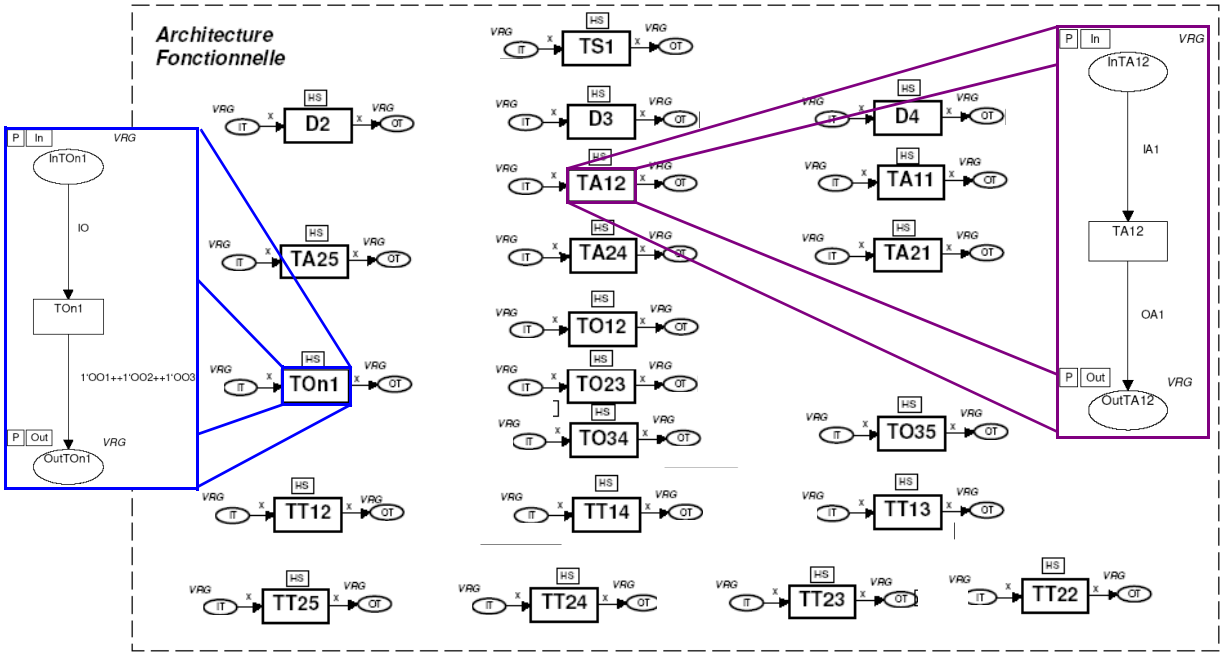


Figure 181 : Modélisation de l'Architecture Fonctionnelle par transitions de substitution

Il ne reste donc maintenant qu'à définir la fonction *taille* qui donne la taille en octets des messages transmis au travers du réseau. Ces informations sont dépendantes de la manière dont sont réalisés les codages des demandes d'envois d'informations.

En se basant sur la documentation Uni-Telway, il est possible de déterminer la taille en octets de l'information transmise au niveau données (DATA). Sachant que pour 1 octet niveau donnée, le protocole de la RS485 définit qu'il faudra transmettre 11 bits au niveau physique.

Prenons le cas d'une requête *WRITE_INTERNAL_WORD*, sa taille au niveau 1 des couches OSI est de 13 octets à transmettre (fig. 182).

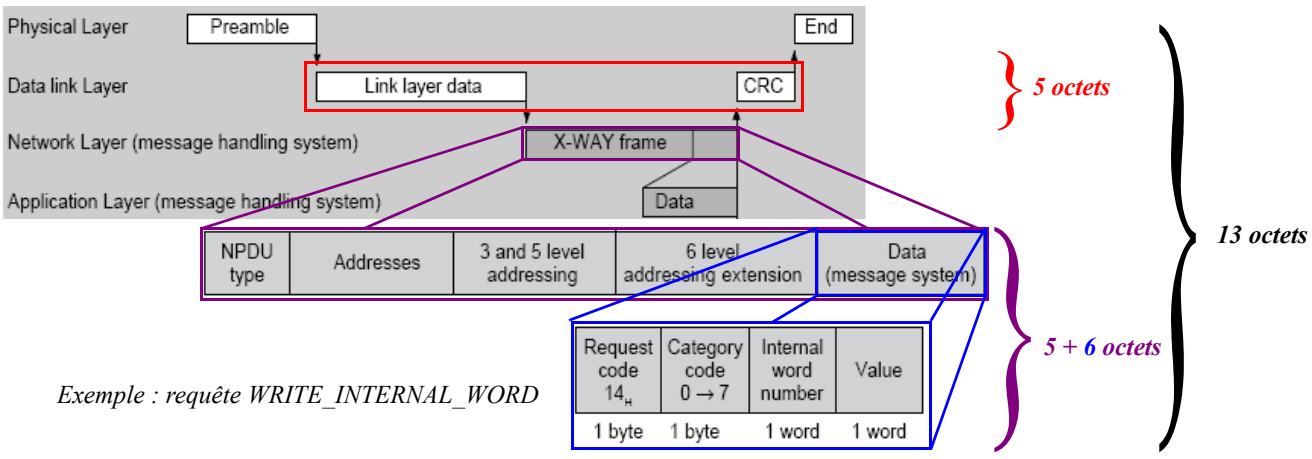


Figure 182 : Taille d'une requête WRITE_INTERNAL_WORD au niveau physique

Une fois, l'ensemble des informations traitées, on peut ainsi définir la fonction taille comme indiqué par la figure 183.

Déclarations globales

```
fun taille(m) =
          case m of
                   OA1    => 20 (* Requete WRITE_VAR 1 word *)
                  |CROA1  => 10 (* Compte tendu WRITE_VAR 1 word *)
                  |OA2    => 15 (* Requete WRITE_INTERNAL_WORD 1 word *)
                  |CROA2  => 10 (* Compte tendu WRITE_INTERNAL_WORD 1 word *)
                  |IT1    => 20 (* Requete WRITE_OBJECT 1 word *)
                  |CRIT1  => 10 (* Compte tendu WRITE_OBJECT 1 word *)
                  |OT1    => 15 (* Requete WRITE_INTERNAL_WORD 1 word *)
                  |CROT1  => 10 (* Compte tendu WRITE_INTERNAL_WORD 1 word *)
                  |IT2    => 15 (* Requete WRITE_INTERNAL_WORD 1 word *)
                  |CRIT2  => 10 (* Compte tendu WRITE_INTERNAL_WORD 1 word *)
                  |OT2    => 15 (* Requete WRITE_INTERNAL_WORD 1 word *)
                  |CROT2  => 10 (* Compte tendu WRITE_INTERNAL_WORD 1 word *)
                  |OO1    => 20 (* Requete WRITE_VAR 1 word *)
                  |CROO1  => 10 (* Compte tendu WRITE_VAR 1 word *)
                  |OO2    => 20 (* Requete WRITE_VAR 1 word *)
                  |CROO2  => 10 (* Compte tendu WRITE_VAR 1 word *)
                  |OO3    => 20 (* Requete WRITE_VAR 1 word *)
                  |CROO3  => 10 (* Compte tendu WRITE_VAR 1 word *)
                  |DS12   => 34 (* Requete WRITE_VAR 8 words *)
                  |CRDS12 => 10 (* Compte tendu WRITE_VAR 8 words *)
                  |DS13   => 34 (* Requete WRITE_VAR 8 words *)
                  |CRDS13 => 10 (* Compte tendu WRITE_VAR 8 words *)
                  |DS14   => 34 (* Requete WRITE_VAR 8 words *)
                  |CRDS14 => 10 (* Compte tendu WRITE_VAR 8 words *)
                  |DS21   => 18 (* Requete READ_VAR 8 words *)
                  |CRDS21 => 27 (* Compte tendu READ_VAR 8 words *)
                  |DS31   => 18 (* Requete READ_VAR 8 words *)
                  |CRDS31 => 27 (* Compte tendu READ_VAR 8 words *)
                  |DS41   => 18 (* Requete READ_VAR 8 words *)
                  |CRDS41 => 27 (* Compte tendu READ_VAR 8 words *)
```

Figure 183 : Fonction définissant la taille des informations transmises via Uni-Telway

La modélisation de l'Architecture Fonctionnelle est maintenant achevée. L'étape suivante consiste à modéliser l'Architecture Opérationnelle.

C.4.4 Modèle de l'Architecture Opérationnelle

L'Architecture Opérationnelle étant le fruit de la projection de l'Architecture Fonctionnelle sur une Architecture Matérielle, il est nécessaire de relier les tâches avec leurs différents équipements de traitements. Etant donné qu'il a été réalisé une représentation des tâches fonctionnelles et des équipements sur la page "Architecture" par le biais de transitions de substitutions, il est maintenant nécessaire de connecter ces différentes transitions. Pour cela, il convient de fusionner les places communes (comme pour les connections inter-équipements dans le paragraphe C.4.2).

Maintenant que les liens entre l'Architecture Fonctionnelle et l'Architecture Matérielle sont réalisés, il est nécessaire de réaliser la fonction *dest* qui a pour rôle de définir les destinations de chaque information en fonction de leur provenance. Pour celà, on s'aide du Tableau 12, Table des projections des tâches fonctionnelles, à la page 177 et du Tableau 13, Table des Flux informationnels, à la page 178 .
La figure 185 ci-dessous représente la traduction des informations de ce tableau.

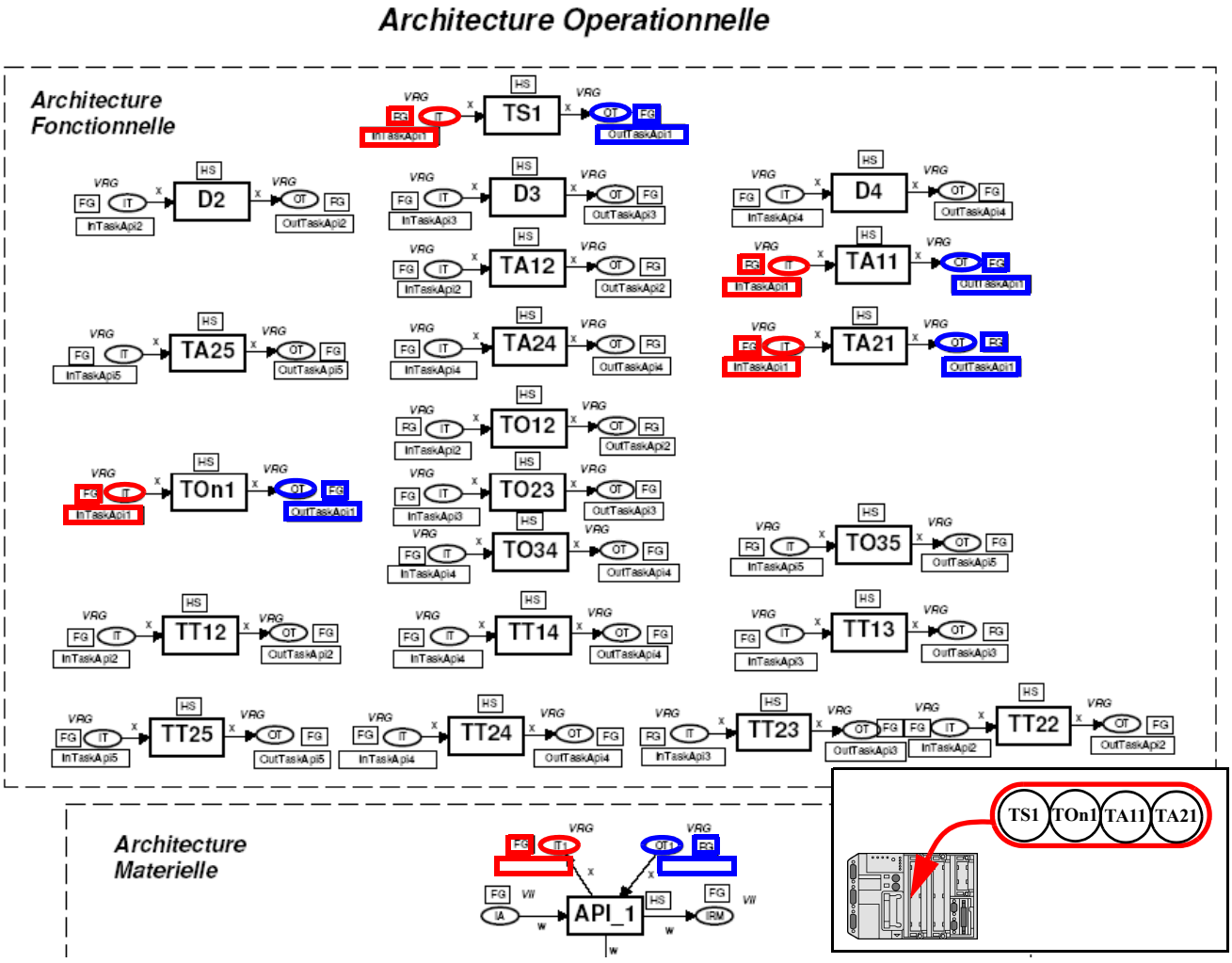


Figure 184 : Fusion des tâches fonctionnelles associées à l'API1

Déclarations globales

```
(* fonction definissant le routage des informations *)
fun dest(x, so) = case so of
          0 => (case x of
                  IA1 => 7| IA2 => 5| IO => 1| IT1 => 7| IT2 => 5| _ => 0)|
          1 => (case x of
                  DS12 => 2| DS13 => 3| DS14 => 4| DS21 => 2| DS31 => 3| DS41 => 4|
                  OA1 => 0| OA2 => 0| OO1 => 2| OO2 => 3| OO3 => 4| _ => 0)|
          2 => (case x of
                  OO1 => 7| OT2 => 7| OA1 => 1| IT1 => 4| _ => 0)|
          3 => (case x of
                  OO2 => 0| OT1 => 0| OT2 => 2| _ => 0)|
          4 => (case x of
                  OA2 => 1| OO3 => 5| OT1 => 3| IT2 => 3| _ => 0)|
          5 => (case x of
                  IA2 => 4| IT2 => 4| OO3 => 0| _ => 0)|
          7 => (case x of
                  IA1 => 2| OO1 => 0| IT1 => 2| OT2 => 0| _ => 0)|
          _ => 0;
```

Figure 185 : Fonction dest servant au routage des informations

C.4.5 Modèle de l'Architecture de Simulation

Le modèle de simulation correspond au modèle de l'Architecture Opérationnelle auquel sera adjoint son environnement de simulation (composé des scénarios d'excitation, des modèles de Partie Opératives utiles) et des observateurs de performances.

Dans un premier temps, nous modélisons les scénarios d'excitations. Nous avons opté pour des excitations qui n'ont pas une période figée mais une période dont la durée est variable, sa valeur est déterminée par une fonction probabiliste *CPN'randint(min, max)* qui a une répartion uniforme comprise entre deux bornes *min* et *max*.

Afin d'avoir un modéle compact, nous avons opté pour un modèle de générateur de stimuli générique qui sera paramétré pour chacunes des informations à émettre. Pour celà, nous définissons :

- la liste *LExcitations* des signaux à exciter,
- la fonction *DGen* calculant les périodes d'excitation de chaque signal
- la date *DebutExcitation* à partir de laquelle débutera le générateur d'excitations

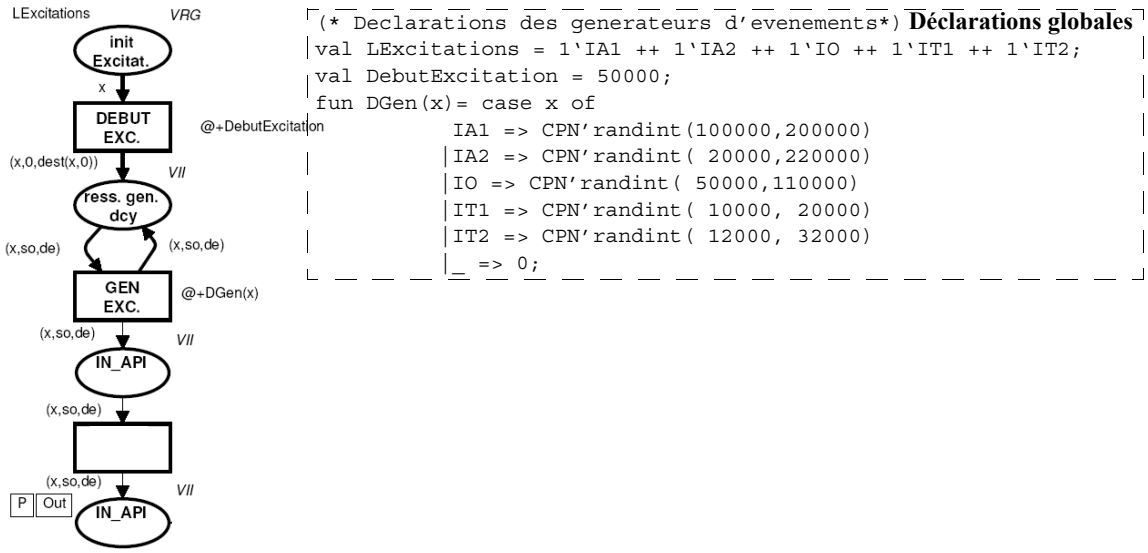


Figure 186 : Générateur de jetons générique

Toutefois, afin d'obtenir un réseau de Petri n-borné, il est nécessaire de ne pas accumuler les jetons correspondant aux informations émises. Il nous faut donc ajouter un absorbeur de jetons lorsqu'un jeton est émis vers l'extérieur (fig. 187).

En second, nous modélisons le reste de l'environement de simulation. Dans notre cas, la Partie Opérative n'a pas d'influence sur les performances étudiées, nous ne la modélisons donc pas. Toutefois, l'interconnection entre les automates API4 et API5 sont externes à la structure de l'Architecture, il est donc nécessaire de réaliser cette connection comme un équipement de Partie Opérative. Cette connection consiste à renvoyer toute information émise vers l'extérieur (ce qui correspond à la place IN_EXT dans le modèle) qui n'est pas à destination de l'extérieur (identifiant 0) vers les entrées des équipements de traitement (ce qui correspond à la place *IN_API* dans le modèle) (fig. 187).

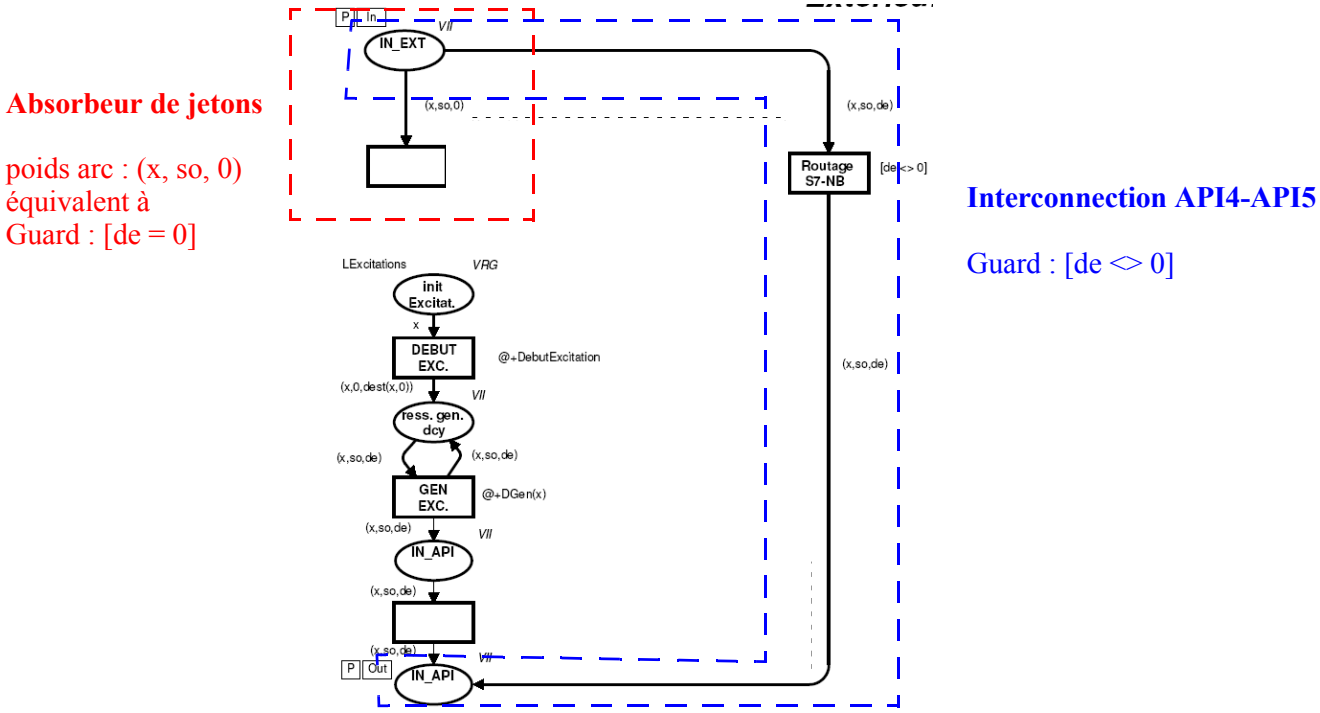


Figure 187 : Modélisation de l'absorbeur et de l'interconnection entre l'API4 et l'API5

Reste à intégrer dans le modèle de simulation les observateurs des signaux pour réaliser les mesures de performances. Etant donné que les signaux à observer sont nombreux et que l'on souhaite aussi étudier la différence entre deux temps de transmission, nous avons décidé de choisir de faire des observations de type «trace». Il faut donc définir pour chacun des signaux, le nom du fichier trace. Nous ajoutons donc pour cela la fonction *M2File* ainsi qu'une référence globale *outfile* qui nous permettra d'écrire dans un fichier (fig. 188).

Déclarations globales

```
fun M2File(m)= case m of
           IA1 => "Log_IA1.txt"
          |IA2 => "Log_IA2.txt"
          |IO  => "Log_IO.txt"
          |IT1 => "Log_IT1.txt"
          |IT2 => "Log_IT2.txt"
          |OA1 => "Log_OA1.txt"
          |OA2 => "Log_OA2.txt"
          |OO1 => "Log_OO1.txt"
          |OO2 => "Log_OO2.txt"
          |OO3 => "Log_OO3.txt"
          |OT1 => "Log_OT1.txt"
          |OT2 => "Log_OT2.txt"
          |_ => "Erreur.txt";
globref outfile = std_out; (* fichier log courant *)
```

Figure 188 : Fonction M2File déterminant le nom du fichier de trace

Nous optons pour l'observation de la génèse des signaux ainsi que leurs aboutissements. Il faut donc greffer des observateurs sur le générateur d'excitation et sur l'absorbeur de jetons (fig. 189).

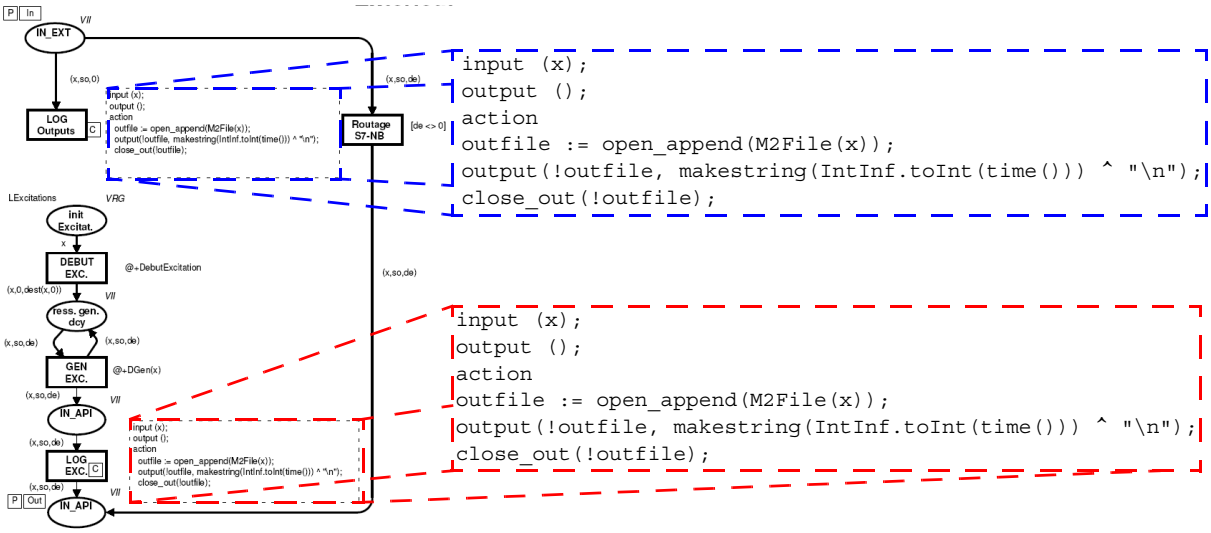


Figure 189 : Modélisation des observateurs

Il est également nécessaire d'initialiser les fichiers trace, nous ajoutons un code permettant l'initialisation associé à une transition qui ne sera tirée qu'au début de la simulation.

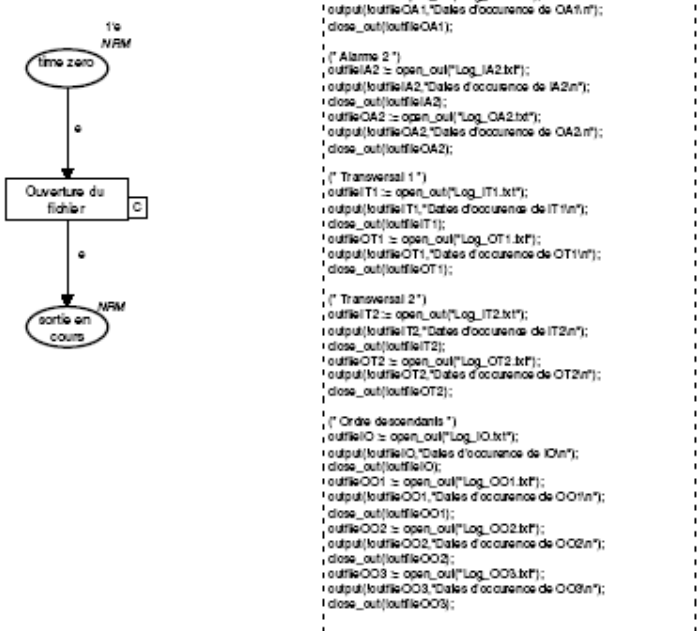


Figure 190 : Réseau de Petri pour l'initialisation des fichiers trace

Le modèle de Simulation étant à présent terminé, il ne reste qu'à executer une simulation et analyser les fichiers trace pour évaluer les performances.

C.5 Modèles réseau de Petri

Les pages suivantes contiennent l'ensemble des modèles RdP correspondant au modèle de simulation

C.5.1 déclarations globales (sur page «Global»)

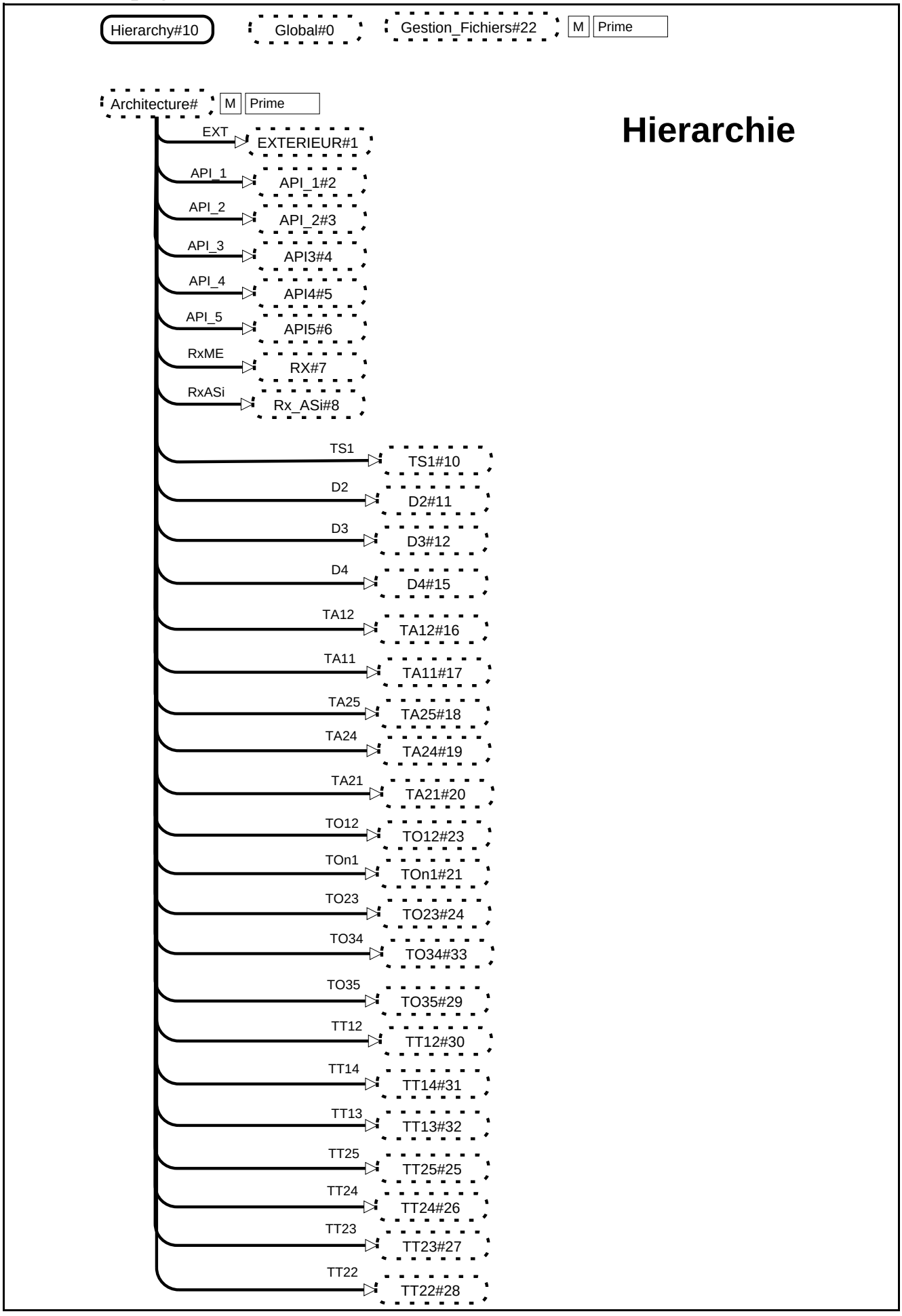
```
(* Declaration des couleurs *)
color NRM = with e timed;                                    (* jeton de base *)
color BINT = int timed;                                      (* entiers *)
color II = product BINT*BINT timed;                          (* Messages en attente par source *)
color VRG = with IA1| OA1| IA2| OA2| CRIA1| CROA1| CRIA2| CROA2|
                 IT1| OT1| IT2| OT2| CRIT1| CROT1| CRIT2| CROT2|
                 IO | OO1| OO2| OO3| CRIO | CROO1| CROO2| CROO3|
                 DS12| DS13| DS14| DS21| DS31| DS41|
                 CRDS12| CRDS13| CRDS14| CRDS21| CRDS31| CRDS41|
                 BUG timed; (* variables globales *)
color VII = product VRG*BINT*BINT timed;                     (* format des informations *)

(* Declarations des variables *)
var x, m : VRG;             (* messages *)
var so, de : BINT;          (* so=source, de=destination *)
var i,j,t,t1,t2 : BINT;     (* entiers *)
var w : VII;                (* donnee transitant *)

(* Declarations pour le Rx 1 *)
var ad : BINT;                                  (* numero esclave scrute sur Rx1 *)
val maitreRx1 = 1;                              (* identifiant du maitre du Rx1 *)
                                                (* marquage initial des Messages en attente *)
val esclavesRx1 = 1`(1,0) ++ 1`(2,0) ++ 1`(3,0) ++ 1`(4,0) ++ 1`(5,0) ++ 1`(6,0) ++ 1`(7,0) ++ 1`(8,0)
++ 1`(9,0) ++ 1`(10,0);

val TPOL = 2;               (* taille d'une requete de polling M->E en octets *)
val TACK = 2;               (* taille d'un acquitement en fin de tranmission d'un message *)
val TEOT = 2;               (* taille d'une reponse EOT de polling E->M en octets *)

val TQF   = 0;              (* duree en 1/10e de ms de la phase entre deux operations du maitre rx *)
val TGEOT = 0;              (* duree en 1/10e de ms de preparation d'un EOT *)

fun suivRx1(ad) =
  if (ad = 10) then 1`1 else 1`(ad+1); (* fonction pour determiner l'esclave suivant devant parler *)

fun IsCR(m) =
 ((m = CRDS12) orelse (m = CRDS13) orelse (m = CRDS14) orelse
  (m = CRDS21) orelse (m = CRDS31) orelse (m = CRDS41) orelse
  (m = CRIA1)  orelse (m = CROA1)  orelse (m = CRIA2)  orelse (m = CROA2) orelse
  (m = CRIT1)  orelse (m = CROT1)  orelse (m = CRIT2)  orelse (m = CROT2) orelse
  (m = CRIO)   orelse (m = CROO1)  orelse (m = CROO2)  orelse (m = CROO3));

(* fonction pour determiner sur quel port de liaison le message passe *)
fun id2ad(m, so) =
case so of
  2 => if IsCR(m) then 8 else 9
 |3 => if IsCR(m) then 4 else 5
 |4 => if IsCR(m) then 6 else 7
 |_ => 0;

fun ad2id(ad) =
case ad of
  0  => 1
 |1  => 6
 |2  => 6
 |3  => 6
 |4  => 3
 |5  => 3
 |6  => 4
 |7  => 4
 |8  => 2
 |9  => 2
 |10 => 2
```

```
  |_  => 0;

fun taille(m) =
case m of
   OA1    => 20   (* Requete WRITE_VAR 1 word                *)
  |CROA1  => 10   (* Compte rendu WRITE_VAR 1 word           *)
  |OA2    => 15   (* Requete WRITE_INTERNAL_WORD 1 word      *)
  |CROA2  => 10   (* Compte rendu WRITE_INTERNAL_WORD 1 word *)
  |IT1    => 20   (* Requete WRITE_OBJECT 1 word             *)
  |CRIT1  => 10   (* Compte rendu WRITE_OBJECT 1 word        *)
  |OT1    => 15   (* Requete WRITE_INTERNAL_WORD 1 word      *)
  |CROT1  => 10   (* Compte rendu WRITE_INTERNAL_WORD 1 word *)
  |IT2    => 15   (* Requete WRITE_INTERNAL_WORD 1 word      *)
  |CRIT2  => 10   (* Compte rendu WRITE_INTERNAL_WORD 1 word *)
  |OT2    => 15   (* Requete WRITE_INTERNAL_WORD 1 word      *)
  |CROT2  => 10   (* Compte rendu WRITE_INTERNAL_WORD 1 word *)
  |OO1    => 20   (* Requete WRITE_VAR 1 word                *)
  |CROO1  => 10   (* Compte rendu WRITE_VAR 1 word           *)
  |OO2    => 20   (* Requete WRITE_VAR 1 word                *)
  |CROO2  => 10   (* Compte rendu WRITE_VAR 1 word           *)
  |OO3    => 20   (* Requete WRITE_VAR 1 word                *)
  |CROO3  => 10   (* Compte rendu WRITE_VAR 1 word           *)
  |DS12   => 34   (* Requete WRITE_VAR 8 words               *)
  |CRDS12 => 10   (* Compte rendu WRITE_VAR 8 words          *)
  |DS13   => 34   (* Requete WRITE_VAR 8 words               *)
  |CRDS13 => 10   (* Compte rendu WRITE_VAR 8 words          *)
  |DS14   => 34   (* Requete WRITE_VAR 8 words               *)
  |CRDS14 => 10   (* Compte rendu WRITE_VAR 8 words          *)
  |DS21   => 18   (* Requete READ_VAR 8 words                *)
  |CRDS21 => 27   (* Compte rendu READ_VAR 8 words           *)
  |DS31   => 18   (* Requete READ_VAR 8 words                *)
  |CRDS31 => 27   (* Compte rendu READ_VAR 8 words           *)
  |DS41   => 18   (* Requete READ_VAR 8 words                *)
  |CRDS41 => 27   (* Compte rendu READ_VAR 8 words           *)
  |_      => 0;

(* Definition de la vitesse du Rx1 *)
val VRx1 = 19200;          (* vitesse de transmission en bauds *)
val TailleTrameNiv1 = 11;  (* taille d'une trame simple au niveau 1 *)
val transME = 1;
val transEM = 0;

(* fonction determinant le temps de transmission des messages a travers le reseau *)
fun Dtrans(t) =
  floor(real(t)*10000.0/(real(VRx1)/real(TailleTrameNiv1)));

fun trans(m, so, ma, senstrans) =
  if ((so <> ma) andalso (senstrans = transME)) then
    Dtrans(taille(m)+1)
  else
    Dtrans(taille(m));

val DCRX = 1; (* duree de retournement de la carte reseau du maitre, ne peut pas etre nul *)

(* fonction gerant les envois des Data Supervision *)
fun DS(i) =
case i of
  1 => DS12
 |2 => DS21
 |3 => DS13
 |4 => DS31
 |5 => DS14
 |6 => DS41
 |_ => BUG;

(* Definition des branchements sur le reseau ASi de l'API2 *)
fun M2ASi(m)=
case m of
  IA1 => 1
 |OO1 => 2
 |IT1 => 4
 |OT2 => 6
 |_   => 0;
```

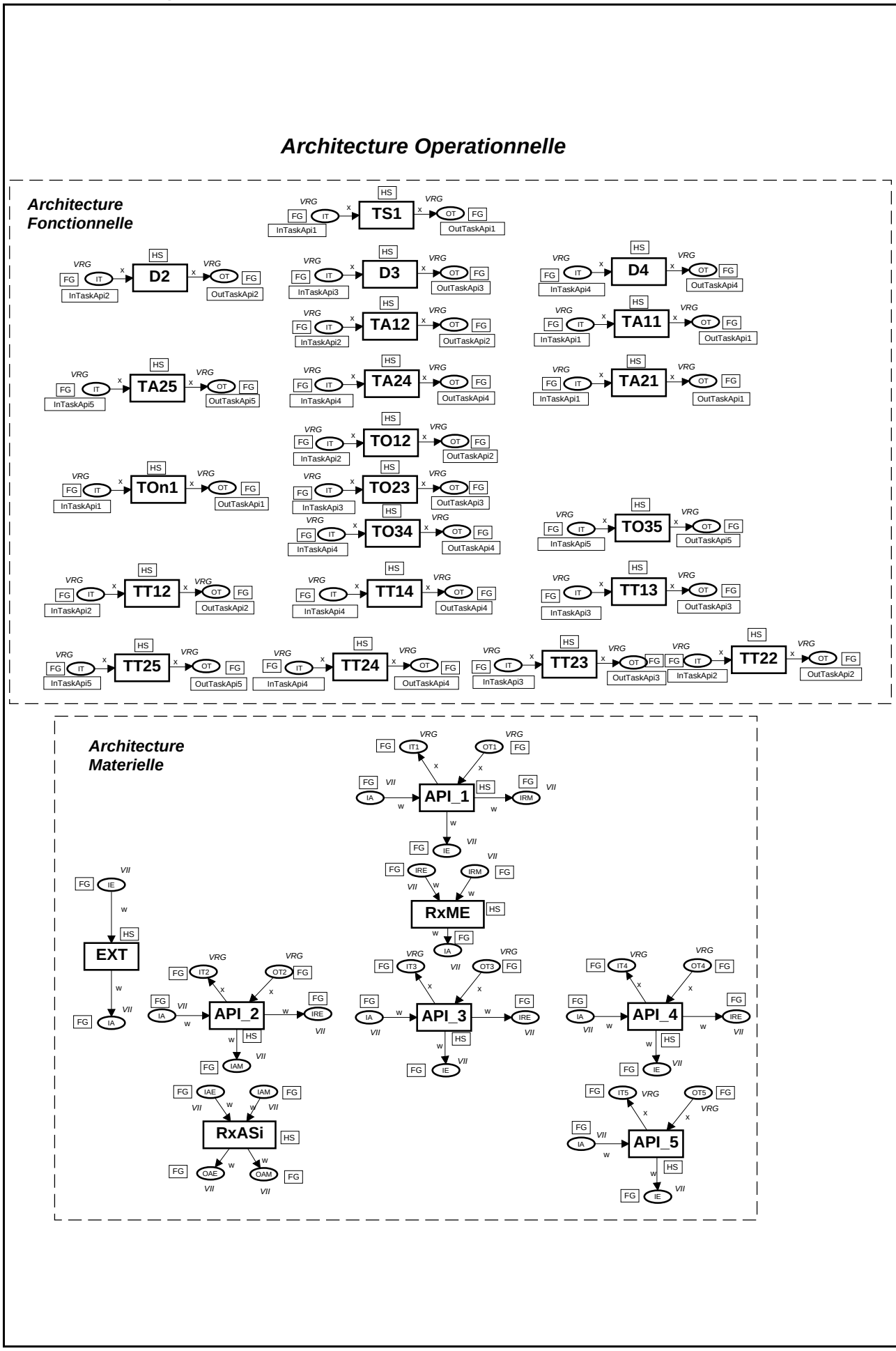
```
(* fonction definissant le routage des informations *)
fun dest(x, so) = case so of
  0 => (case x of
    IA1 => 7| IA2 => 5| IO => 1| IT1 => 7| IT2 => 5| _ => 0)|
  1 => (case x of
    DS12 => 2| DS13 => 3| DS14 => 4| DS21 => 2| DS31 => 3| DS41 => 4| OA1 => 0| OA2 => 0| OO1 => 2| OO2
=> 3| OO3 => 4| _ => 0)|
  2 => (case x of
    OO1 => 7| OT2 => 7| OA1 => 1| IT1 => 4| _ => 0)|
  3 => (case x of
    OO2 => 0| OT1 => 0| OT2 => 2| _ => 0)|
  4 => (case x of
    OA2 => 1| OO3 => 5| OT1 => 3| IT2 => 3| _ => 0)|
  5 => (case x of
    IA2 => 4| IT2 => 4| OO3 => 0| _ => 0)|
  7 => (case x of
    IA1 => 2| OO1 => 0| IT1 => 2| OT2 => 0| _ => 0)|
  _ => 0;

(* Declarations des generateurs d'evenements*)
val LExcitations = 1`IA1 ++ 1`IA2 ++ 1`IO ++ 1`IT1 ++ 1`IT2;
val DebutExcitation = 50000;

fun DGen(x)=
case x of
  IA1 => CPN'randint(100000,200000)
 |IA2 => CPN'randint( 20000,220000)
 |IO  => CPN'randint( 50000,110000)
 |IT1 => CPN'randint( 10000, 20000)
 |IT2 => CPN'randint( 12000, 32000)
 |_   => 0;


(* Declarations pour les fichiers d'observation*)
globref outfile    = std_out; (* log courant *)
globref outfileIA1 = std_out; (* log de IA1  *)
globref outfileOA1 = std_out; (* log de OA1  *)
globref outfileIA2 = std_out; (* log de IA2  *)
globref outfileOA2 = std_out; (* log de OA2  *)
globref outfileIT1 = std_out; (* log de IT1  *)
globref outfileOT1 = std_out; (* log de OT1  *)
globref outfileIT2 = std_out; (* log de IT2  *)
globref outfileOT2 = std_out; (* log de OT2  *)
globref outfileIO  = std_out; (* log de IO   *)
globref outfileOO1 = std_out; (* log de OO1  *)
globref outfileOO2 = std_out; (* log de OO2  *)
globref outfileOO3 = std_out; (* log de OO3  *)

fun M2File(m)=
case m of
  IA1 => "Log_IA1.txt"
 |IA2 => "Log_IA2.txt"
 |IO  => "Log_IO.txt"
 |IT1 => "Log_IT1.txt"
 |IT2 => "Log_IT2.txt"
 |OA1 => "Log_OA1.txt"
 |OA2 => "Log_OA2.txt"
 |OO1 => "Log_OO1.txt"
 |OO2 => "Log_OO2.txt"
 |OO3 => "Log_OO3.txt"
 |OT1 => "Log_OT1.txt"
 |OT2 => "Log_OT2.txt"
 |_   => "Erreur.txt";

val PPolling = 11500;

fun GenCR(m)=
case m of
  IA1  => CRIA1
 |OA1  => CROA1
 |IA2  => CRIA2
 |OA2  => CROA2
```

```
 |IO   => CRIO
 |OO1  => CROO1
 |OO2  => CROO2
 |OO3  => CROO3
 |IT1  => CRIT1
 |OT1  => CROT1
 |IT2  => CRIT2
 |OT2  => CROT2
 |DS12 => CRDS12
 |DS13 => CRDS13
 |DS14 => CRDS14
 |DS21 => CRDS21
 |DS31 => CRDS31
 |DS41 => CRDS41
 |_    => BUG;

fun TRetId(id) =
case id of
  1 => 10
 |2 => 20
 |3 => 40
 |4 => 40
 |_ => 0;

(* Temps de generation d'un message suite a un polling du maitre *)
fun TGMsg(id) =
case id of
  2 => 50
 |3 => 5
 |4 => 5
 |_ => 0;

(* Temps de generation d'un acquitement suite a la reception d'un message *)
fun TGAck(id) =
case id of
  1 => 20
 |2 => 80
 |3 => 80
 |4 => 80
 |_ => 0;

(* Temps de generation d'un Compte rendu apres envoi de l'acquitement de reception *)
fun TGCR(id) =
case id of
  1 => 60
 |2 => 60
 |3 => 80
 |4 => 80
 |_ => 0;

val TRedir = Dtrans(TACK)+1;
```

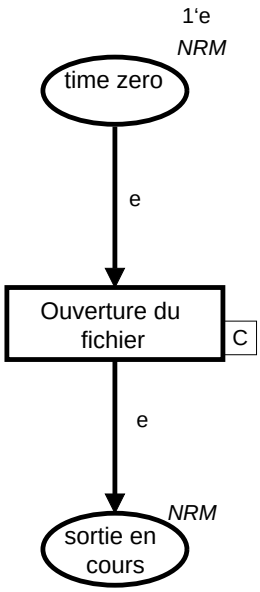
C.5.2 page «Hiérarchie»



C.5.3 Page «Architecture»

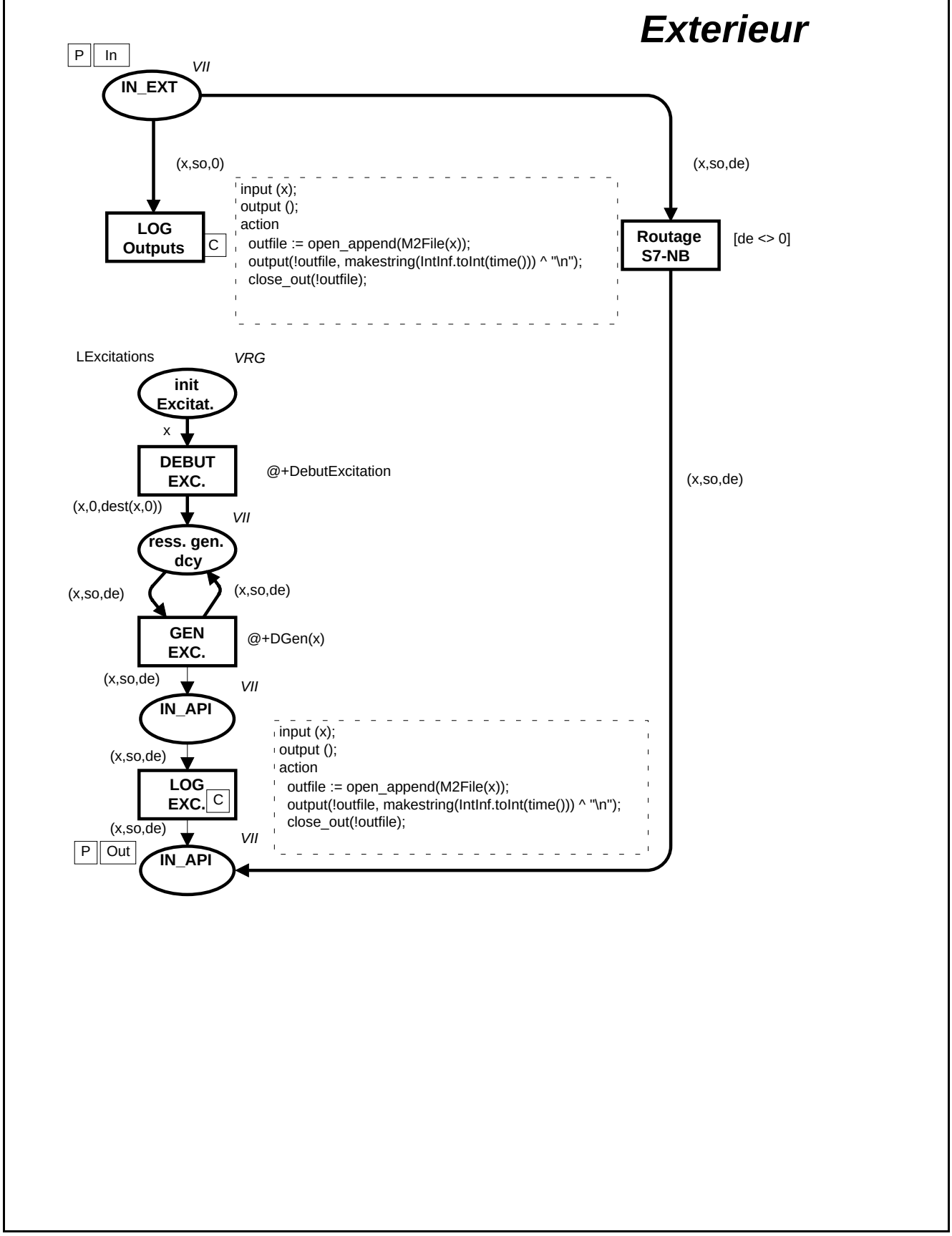


C.5.4 Page «Gestion_Fichiers»

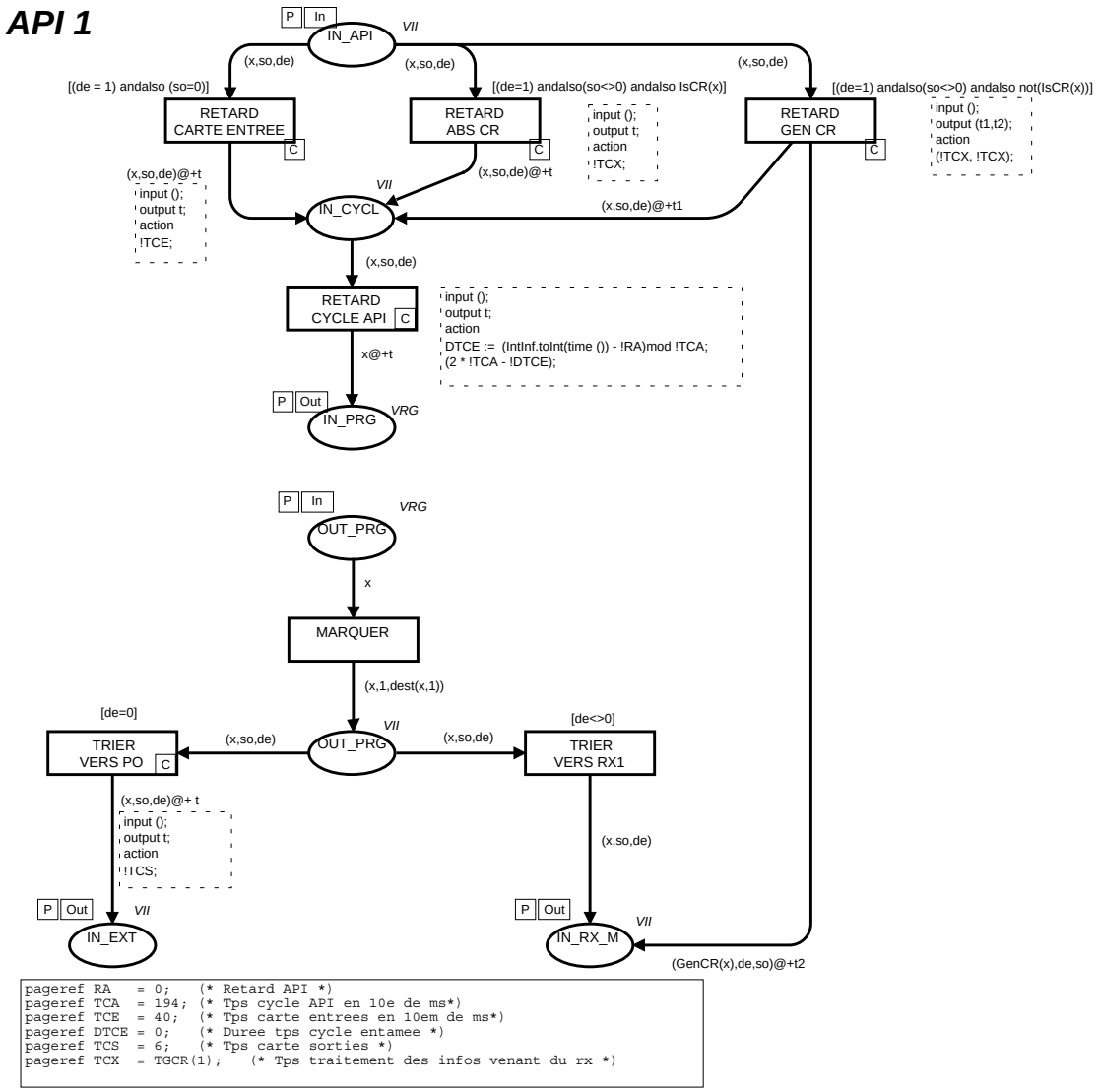



```
input ();
output ();
action
(* ouverture des fichiers logs  *)
(* Alarme 1 *)

outfileIA1 := open_out(M2File(IA1));
output(!outfileIA1,"Dates d’occurence de IA1\n");
close_out(!outfileIA1);
outfileOA1 := open_out("Log_OA1.txt");
output(!outfileOA1,"Dates d’occurence de OA1\n");
close_out(!outfileOA1);

(* Alarme 2 *)
outfileIA2 := open_out("Log_IA2.txt");
output(!outfileIA2,"Dates d’occurence de IA2\n");
close_out(!outfileIA2);
outfileOA2 := open_out("Log_OA2.txt");
output(!outfileOA2,"Dates d’occurence de OA2\n");
close_out(!outfileOA2);

(* Transversal 1 *)
outfileIT1 := open_out("Log_IT1.txt");
output(!outfileIT1,"Dates d’occurence de IT1\n");
close_out(!outfileIT1);
outfileOT1 := open_out("Log_OT1.txt");
output(!outfileOT1,"Dates d’occurence de OT1\n");
close_out(!outfileOT1);

(* Transversal 2 *)
outfileIT2 := open_out("Log_IT2.txt");
output(!outfileIT2,"Dates d’occurence de IT2\n");
close_out(!outfileIT2);
outfileOT2 := open_out("Log_OT2.txt");
output(!outfileOT2,"Dates d’occurence de OT2\n");
close_out(!outfileOT2);

(* Ordre descendants *)
outfileIO := open_out("Log_IO.txt");
output(!outfileIO,"Dates d’occurence de IO\n");
close_out(!outfileIO);
outfileOO1 := open_out("Log_OO1.txt");
output(!outfileOO1,"Dates d’occurence de OO1\n");
close_out(!outfileOO1);
outfileOO2 := open_out("Log_OO2.txt");
output(!outfileOO2,"Dates d’occurence de OO2\n");
close_out(!outfileOO2);
outfileOO3 := open_out("Log_OO3.txt");
output(!outfileOO3,"Dates d’occurence de OO3\n");
close_out(!outfileOO3);
```

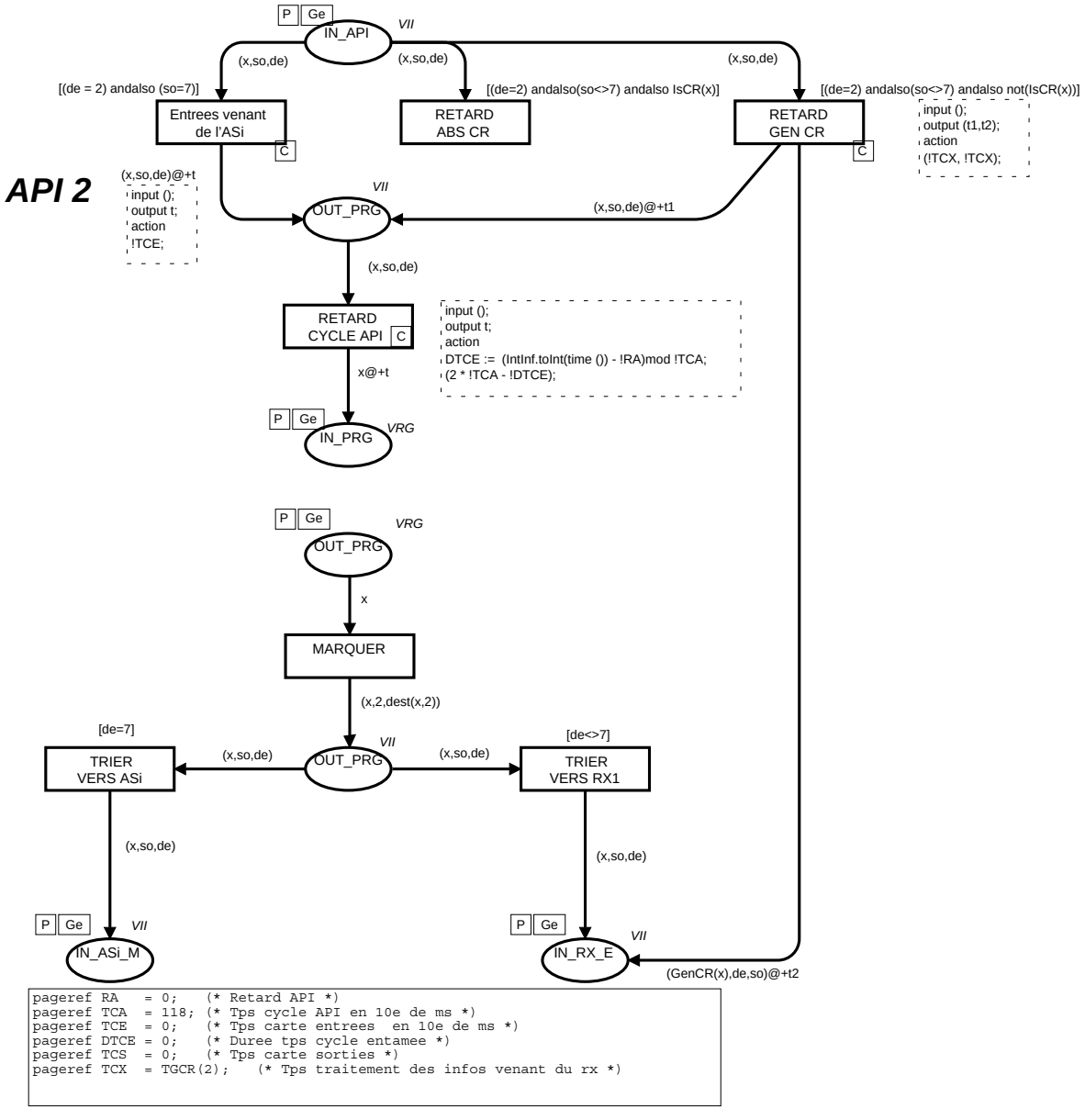
C.5.5 Page «Exterieur»



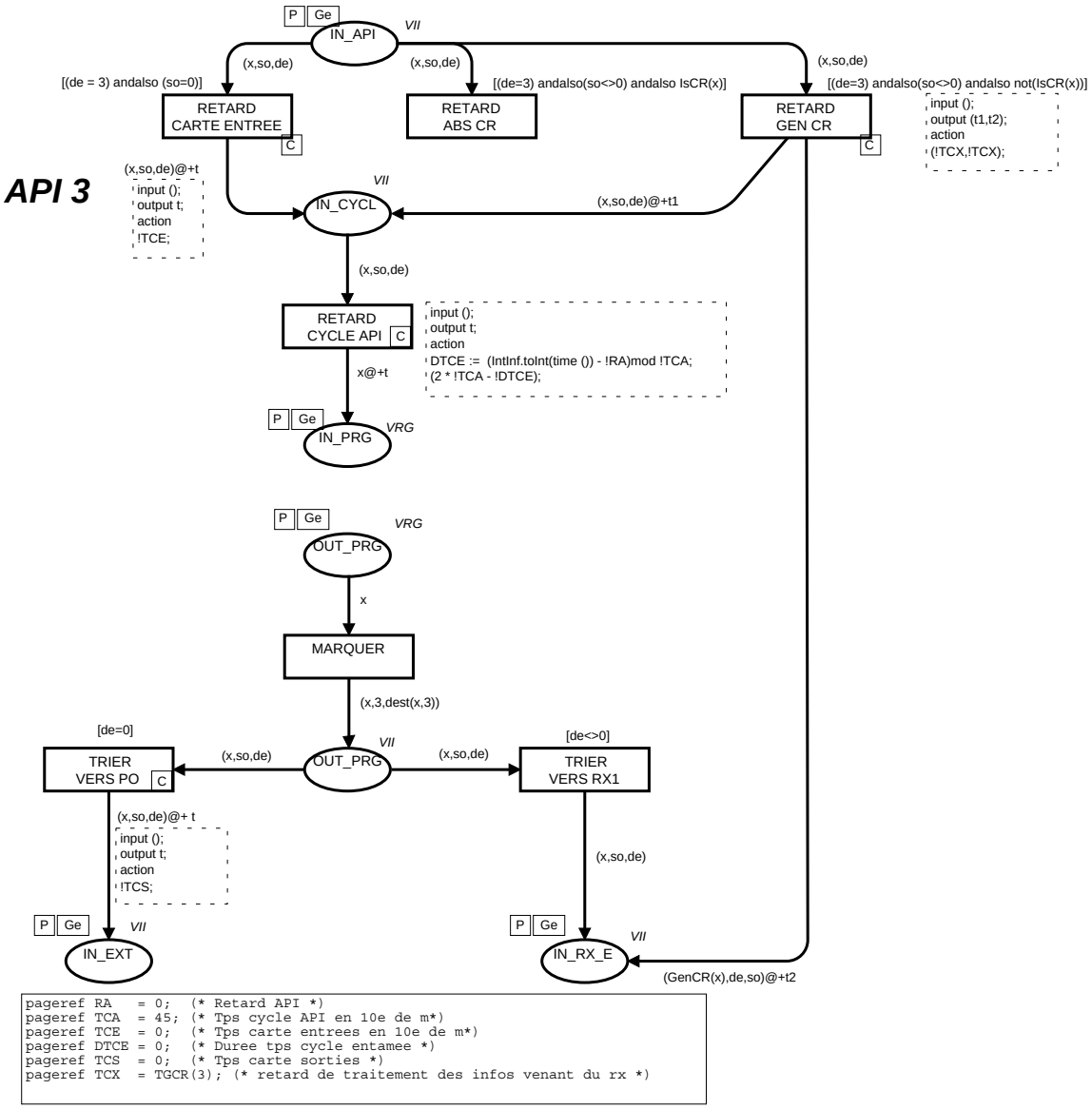
C.5.6 Page «API_1»



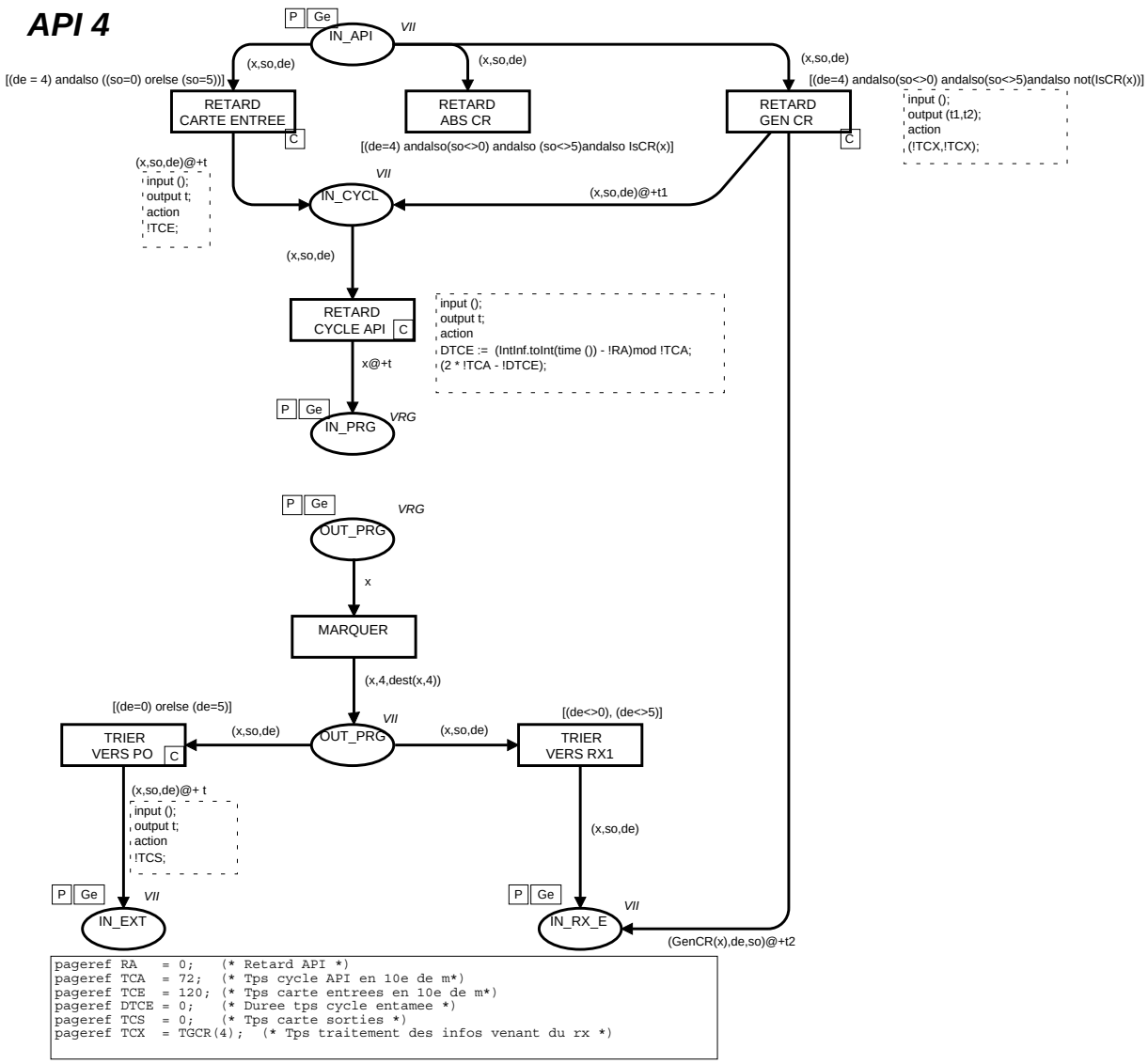
C.5.7 Page «API_2»



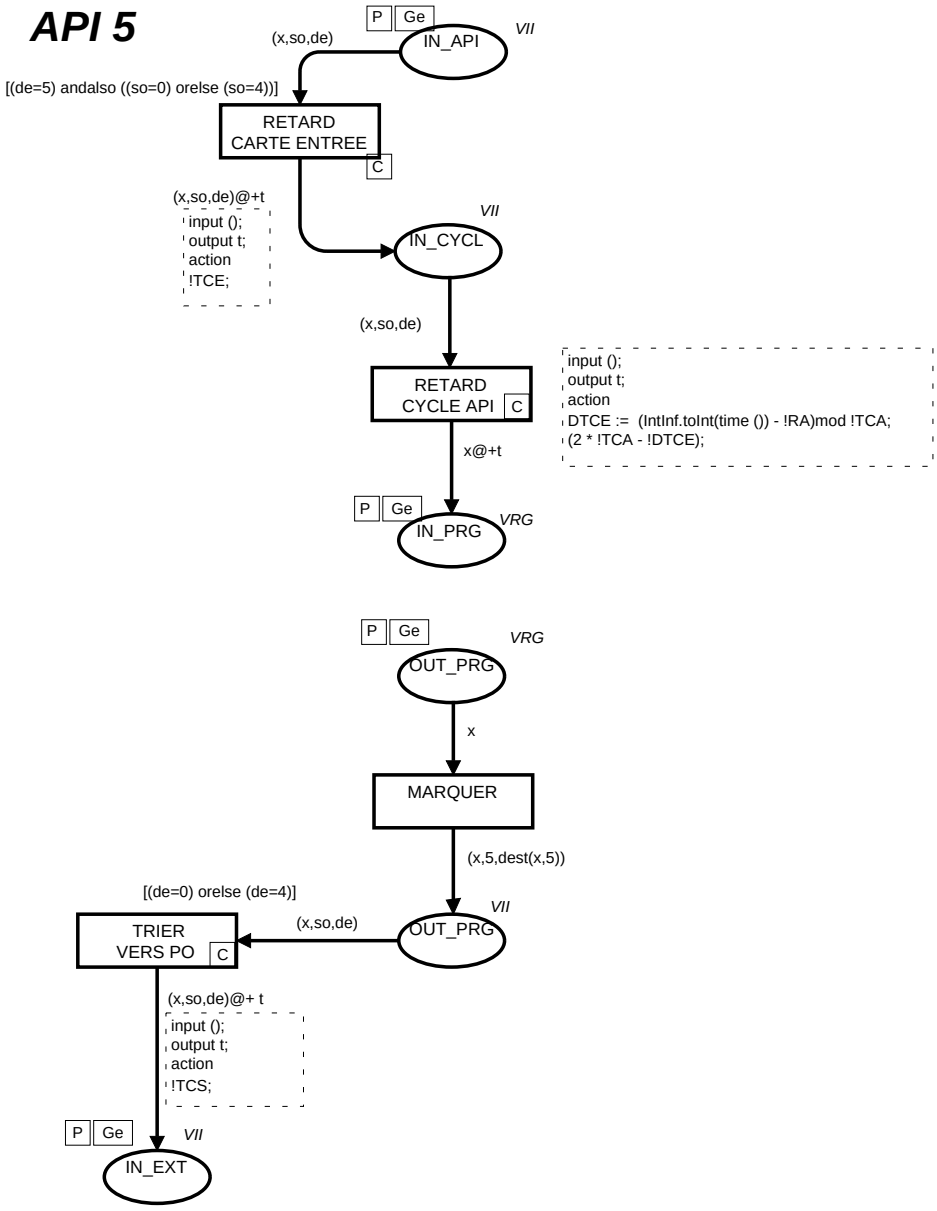
C.5.8 Page «API_3»



C.5.9 Page «API_4»

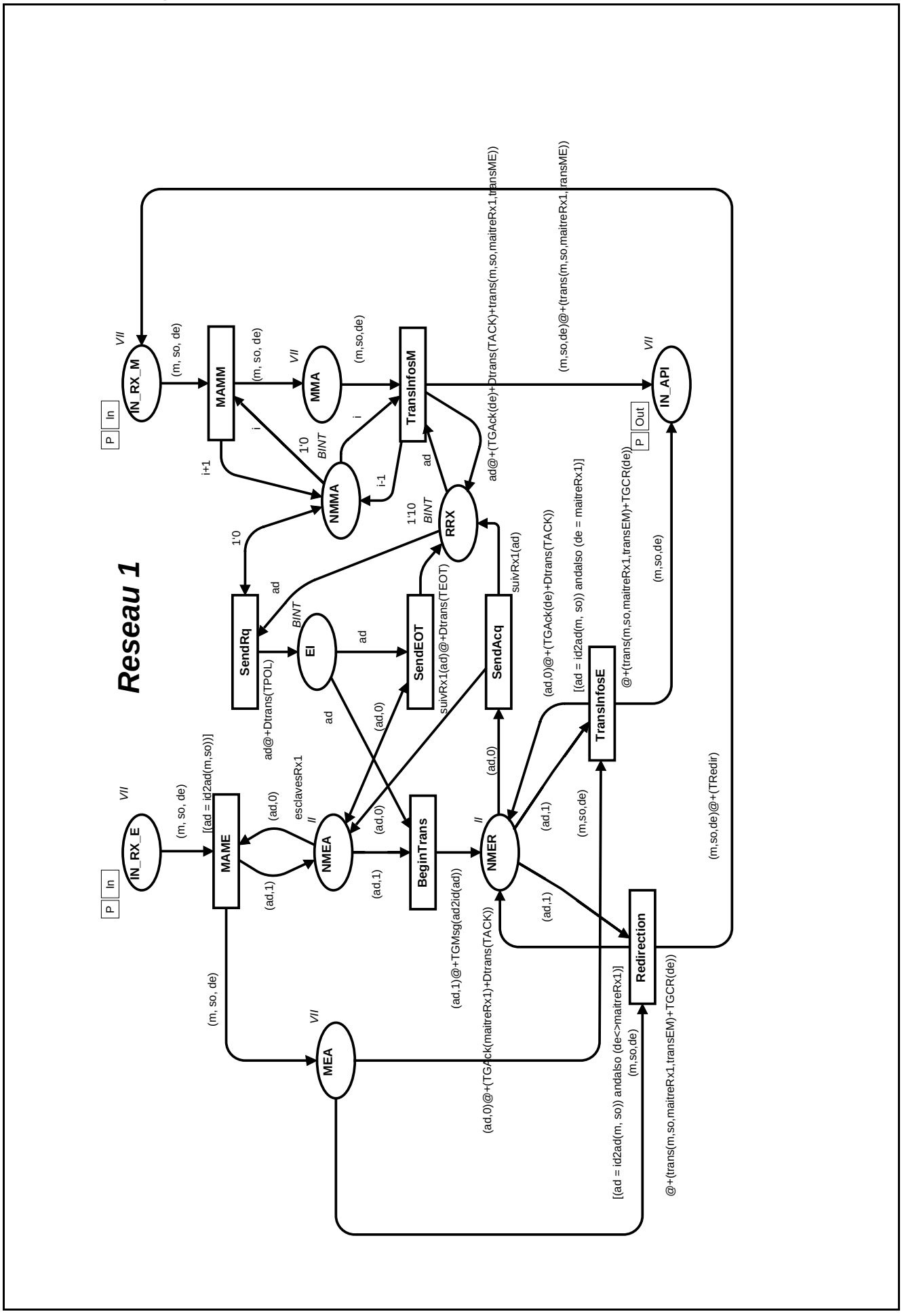


C.5.10 Page «API_5»

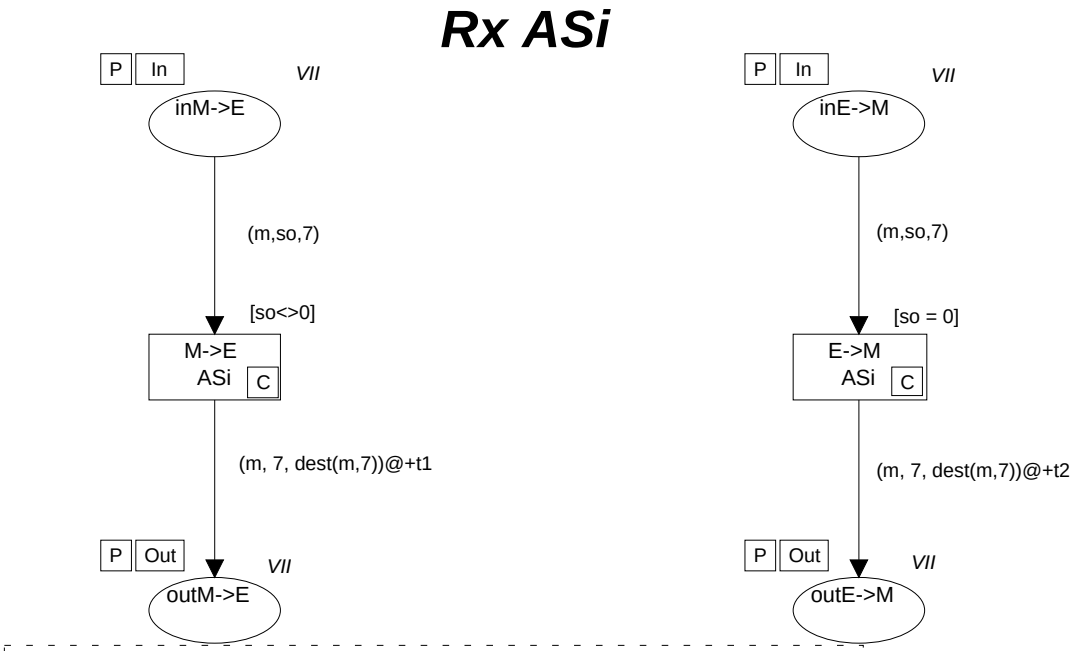



```
pageref RA   = 0;   (* Retard API *)
pageref TCA  = 50;  (* Tps cycle API en 10e de ms*)
pageref TCE  = 64;  (* Tps carte entrees en 10e de m *)
pageref DTCE = 0;   (* Duree tps cycle entamee *)
pageref TCS  = 0;   (* Tps carte sorties *)
pageref TCX  = TGCR(5);   (* Tps de traitement des infos venant du rx *)
```

C.5.11 Page «RX»


Reseau 1
IN_RX_M
IN_RX_E
MAMM
MAME
MMA
NMMA
NMEA
NMER
MEA
TransInfosM
TransInfosE
RRX
SendRq
SendEOT
SendAcq
EI
BeginTrans
Redirection
IN_API
P
In
Out
VII
II
BINT
1'0
1'10
i
i-1
i+1
ad
(m, so, de)
(m,so,de)
(ad,0)
(ad,1)
esclavesRx1
ad@+Dtrans(TPOL)
suivRx1(ad)@+Dtrans(TEOT)
suivRx1(ad)
(ad,0)@+(TGAck(de)+Dtrans(TACK)
(ad,1)@+TGMsg(ad2id(ad))
(ad,0)@+(TGAck(maitreRx1)+Dtrans(TACK))
ad@+(TGAck(de)+Dtrans(TACK)+trans(m,so,maitreRx1,transME))
(m,so,de)@+(trans(m,so,maitreRx1,transME))
@+(trans(m,so,maitreRx1,transEM)+TGCR(de))
[(ad = id2ad(m, so)) andalso (de = maitreRx1)]
[(ad = id2ad(m, so)) andalso (de<>maitreRx1)]
(m,so,de)@+(TRedir)
@+(trans(m,so,maitreRx1,transEM)+TGCR(de))

C.5.12 Page «RX_ASI»

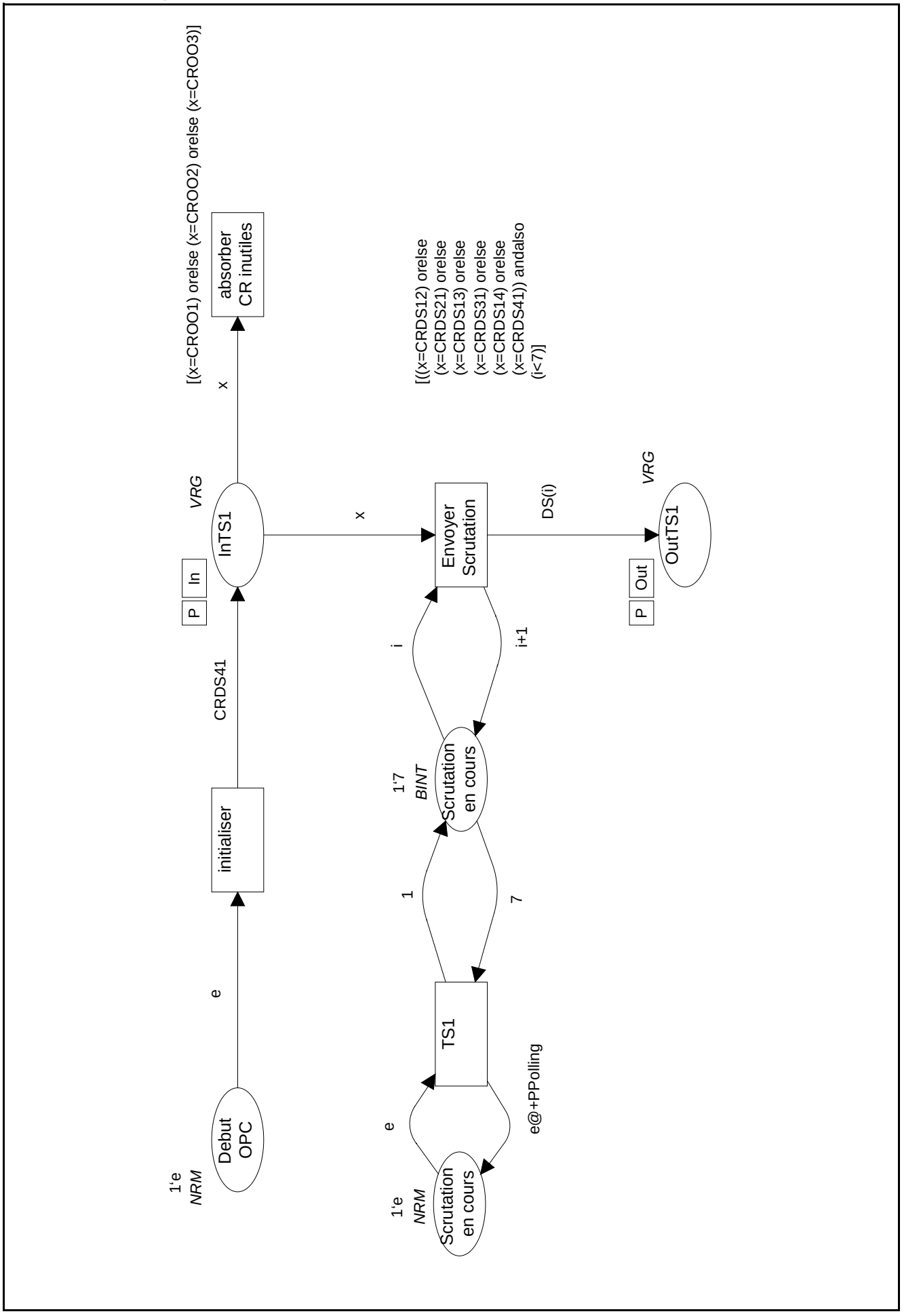



```
input (m);
output (t1);
action
D := real(floor((real(IntInf.toInt(time()) - !DebutCycle))*10.0) mod floor(!P*10.0))/10.0;
if (!D >= (real(M2ASi(m)) * 1.56)) then
  floor(!P - !D + real(M2ASi(m)) * 1.56)
else
  floor(!D - real(M2ASi(m)) * 1.56);
```

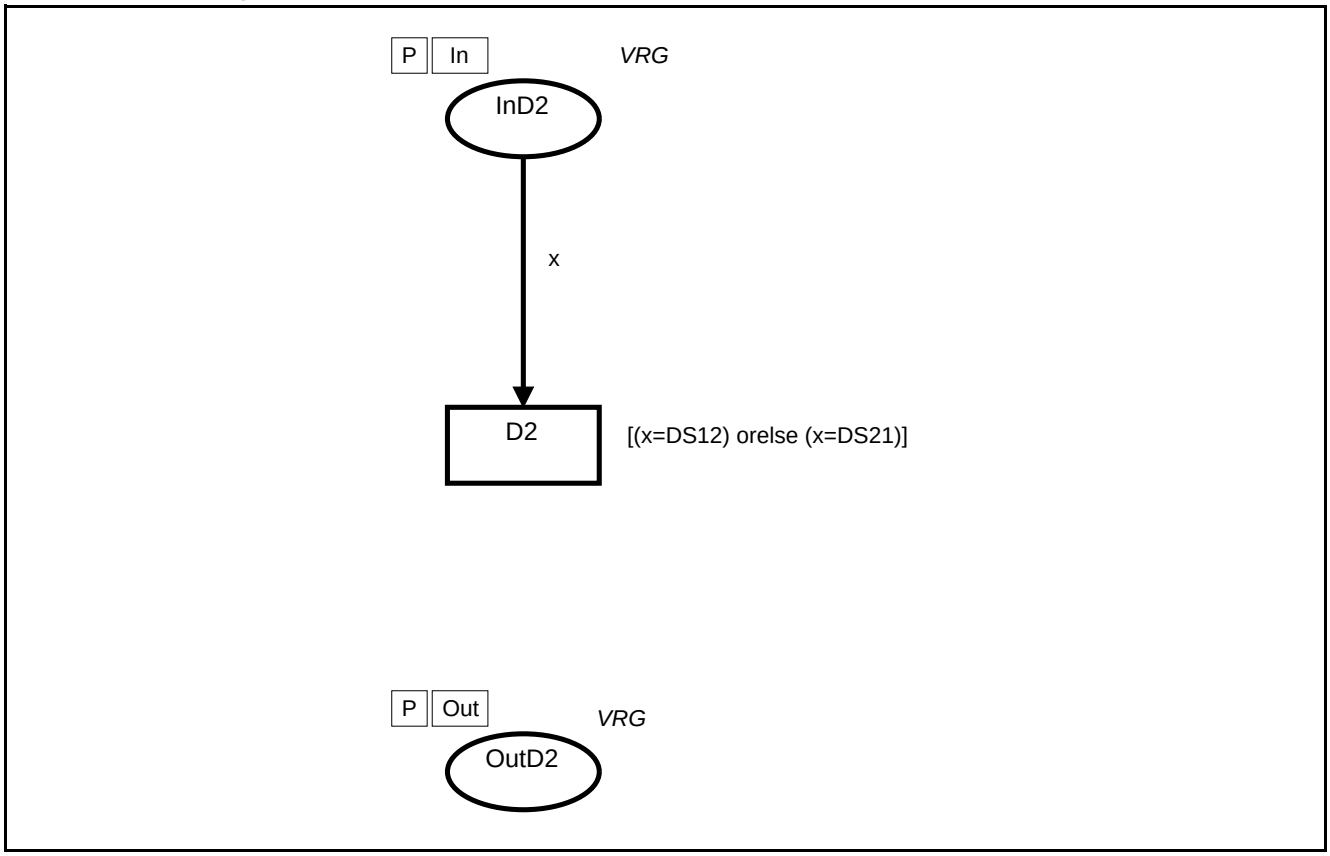
```
input (m);
output (t2);
action
D := real(floor((real(IntInf.toInt(time()) - !DebutCycle))*10.0) mod floor(!P*10.0))/10.0;
if (!D >= (real(M2ASi(m)) * 1.56)) then
  floor(!P - !D + real(M2ASi(m)) * 1.56)
else
  floor(!D - real(M2ASi(m)) * 1.56);
```

```
pageref DebutCycle = 0;
pageref P = (8.0 + 1.0)*1.56;     (* Periode de scrutation ASi pour 8 esclaves*)
pageref D = 0.0;                    (* duree entamee dans le cycle Asi *)
```

C.5.13 Page «TS1»

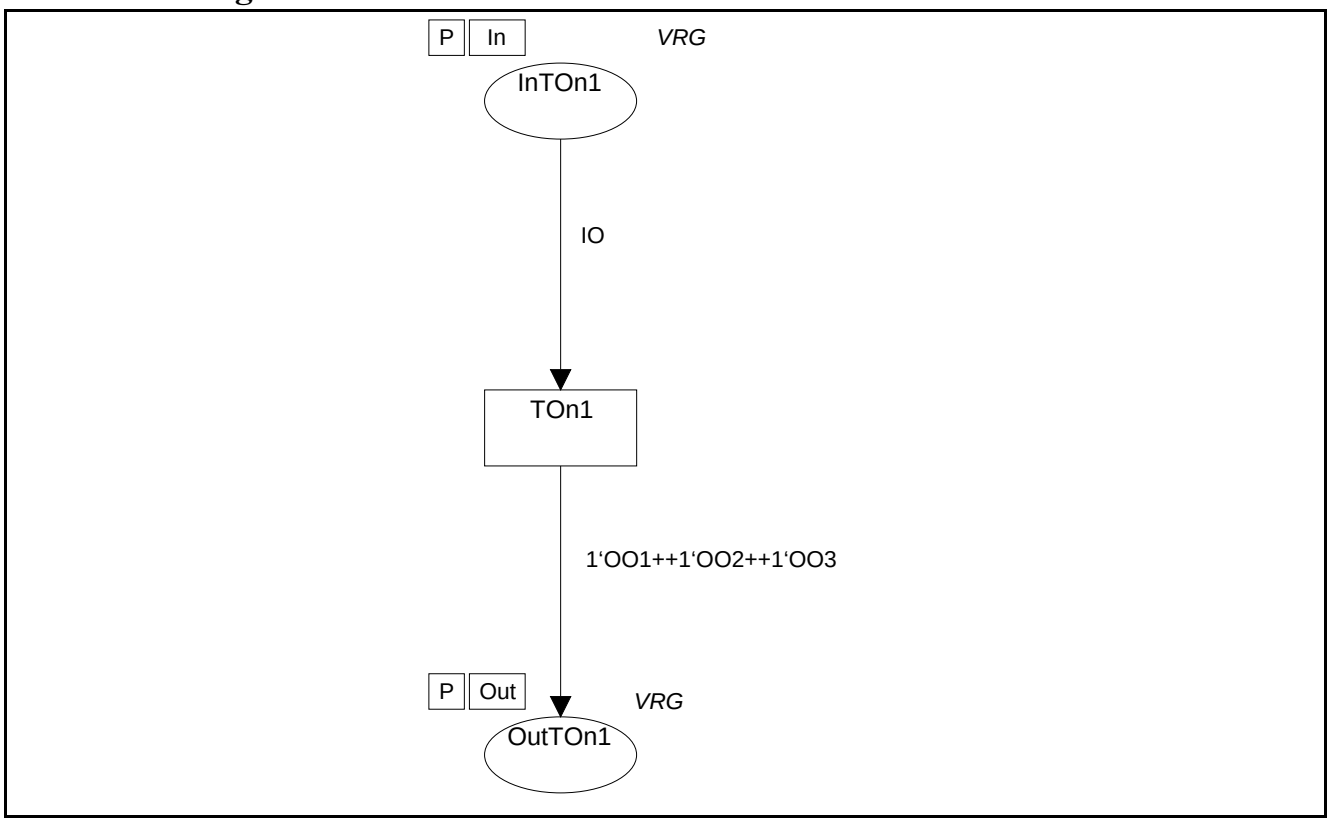


C.5.14 Page «D2»

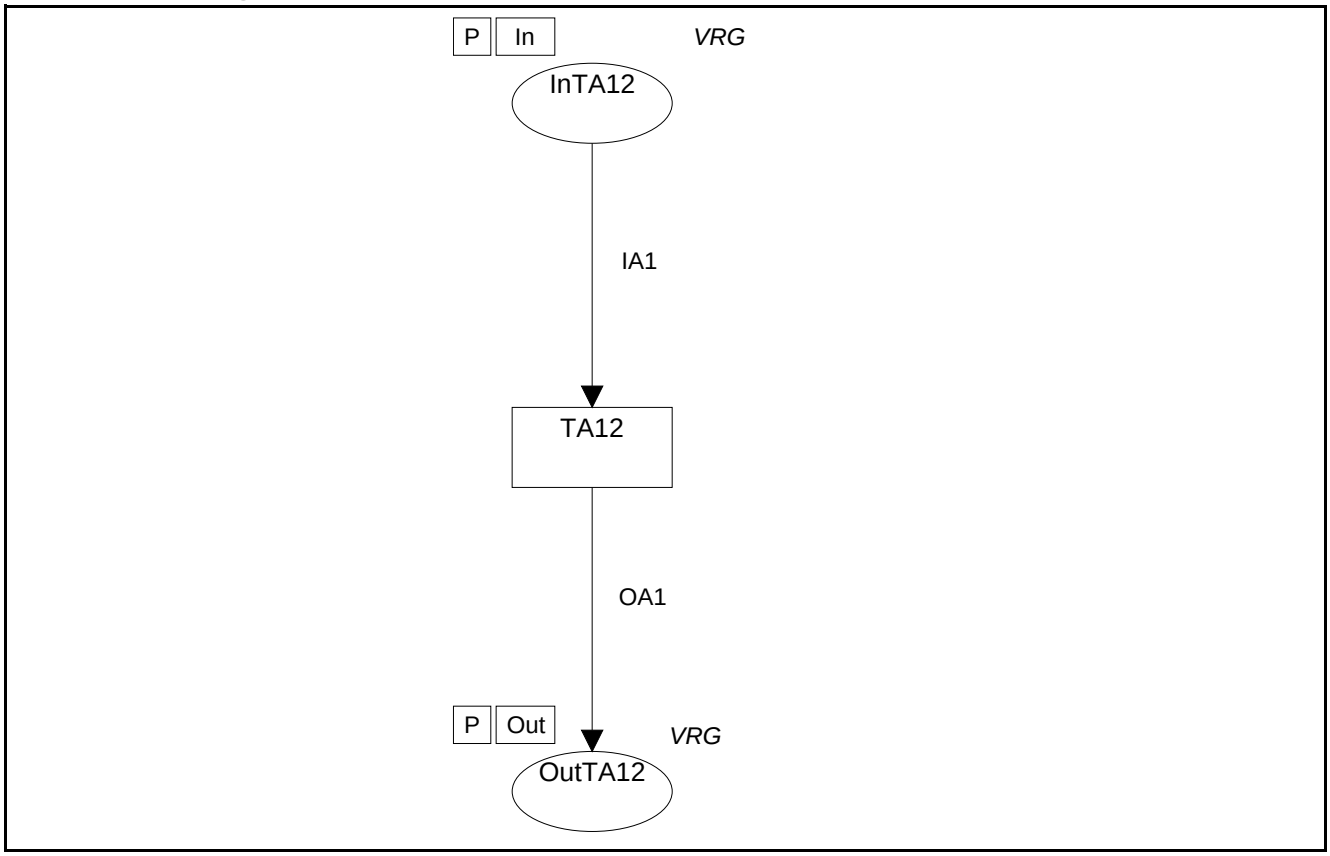


Les pages correspondant aux tâches D3 et D4 ont exactement la même structure que la page «D2», leurs représentations ne seront donc pas présentes dans cette annexe.

C.5.15 Page «TOn1»



C.5.16 Page «TA12»



Les pages correspondant aux tâches TA11, TA25, TA24, TO12, TO23, TO34, TO35, TT12, TT14, TT13, TT25, TT24, TT23, TT22 ont exactement la même structure que la page «TA12», leurs représentations ne seront donc pas présentes dans cette annexe.

Résumé :

Les performances temporelles d'une architecture de contrôle-commande conditionnent fortement celles du système automatisé de production commandé. Pour maîtriser ces performances temporelles, l'architecte automaticien doit pouvoir les évaluer à chaque phase du cycle de développement : de l'étude d'avant-projet à la conception détaillée ainsi que lors de la mise au point. Il n'a cependant pas les mêmes attentes concernant les performances estimées. Des résultats approximatifs, obtenus rapidement à partir de données encore imprécises, lui suffisent en début de cycle de développement, alors que des prévisions fiables, même si elles sont plus difficiles à obtenir, lui sont nécessaires en conception détaillée.

L'approche que nous présentons prend en compte ces différents besoins et contraintes. Elle consiste en une méthode d'évaluation des performances temporelles d'architectures de commande complexes distribuées en réseaux, destinée à accompagner l'architecte tout au long du cycle de développement. Cette méthode est basée sur une modélisation modulaire du comportement temporel de la commande par réseaux de Petri Colorés Temporisés. Pour ce faire, trois temps sont nécessaires dans la modélisation. La modélisation de l'*architecture fonctionnelle* consiste à représenter les fonctions de commande, et leurs interactions. La modélisation de l'*architecture matérielle* permet de traduire la topologie et les connexions entre équipements de commandes (automates programmables, réseaux de communication, ...). L'affectation des fonctions de commande aux équipements ainsi que la prise en compte des communications entre ces fonctions via des réseaux de communication constitue le modèle de l'*architecture opérationnelle*. Une fois le modèle de comportement de l'ensemble de l'architecture opérationnelle constitué, l'évaluation des performances temporelles est réalisée par simulation du réseau de Petri obtenu.

Pour valider notre approche, nous traitons un exemple significatif à l'aide de la plate forme logicielle Design CPN. Cette étude de cas nous permet de présenter une série d'études portant sur la convergence de nos modèles, sur la sensibilité des résultats de simulation aux erreurs de paramétrages et sur la précision des performances obtenues par simulation en les confrontant à celles mesurées sur le système réel.

Mots clés :

Architecture de contrôle-commande, système de production, évaluation de performances temporelles, modélisation, simulation, réseaux de Petri colorés et temporisés, mesures expérimentales.

Abstract :

The performances of controlled system are strongly linked to performances of the control architecture timed. To manage these performances, the control engineer has to evaluate them at each stage of the life cycle: from the project requirements to the detailed design and at the setup stage. However, the engineer expectations levels concerning the estimated performances are variables. Approximated results quickly obtained with rough data are enough at the beginning of the life cycle. But he needs reliable results at the detailed design stage even if the performances are longer and more difficult to obtain.

Our approach takes into account theses various engineer needs. The proposed method evaluates timed performances of networked control architecture and guides the engineer throughout the architecture development. A modular design has been retained to model the dynamic behavior of the control architecture using Timed Colored Petri Nets. Three points of view are considered: The dynamic behavior model of functional architecture is obtained by modeling the control tasks and their interactions; The dynamic behavior model of hardware architecture is obtained by modeling the topology and the connections between hardware components (PLC, communication networks …); The dynamic behavior model of operational architecture is obtained by modeling the mapping of the control tasks and data flows to the hardware components. Then the timed performances evaluation is obtained by simulation of the Petri net.

To validate our approach, we treat a relevant example using the software platform Design/CPN. With this case study, we present different studies concerning the dynamic behavior models convergence, the simulation results sensitivity according to the parameters settings error and the simulation performances accuracy by confrontation with the performances measured on the real system.

Key words :

control command architecture, production system, timed performance evaluation, modelisation, simulation, coloured timed Petri nets, experimental mesurements