Évaluation de performance d'architecture de contrôle-commande en réseau dans un contexte incertain d'avant-vente

Moulaye A.A. Ndiaye

UNIVERSITÉ DE LORRAINE

Ecole Doctorale IAEM Lorraine

Evaluation de performance d'architecture de contrôle-commande en réseau dans un contexte incertain d'avant-vente

THESE

Soutenue le 16/03/2017

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention Génie Informatique, Automatique et Traitement du Signal)

par

Moulaye A.A. NDIAYE

Composition du jury

Président du jury	Marilyne Chetto	Professeur à Université de Nantes
Rapporteurs	Jean-Marc Thiriet Armand Toguyeni	Professeur à l'Université Grenoble Alpes Professeur à l'Ecole Centrale de Lille
Examinateurs	Lars Kristensen Thierry Divoux	Professeur à Bergen University College Professeur à l'Université de Lorraine
Directeur de thèse	Jean-François Pétin	Professeur à l'Université de Lorraine
Encadrants	Jean-Philippe Georges Jacques Camerini	Maître de conférences à l'Université de Lorraine Schneider Electric
Invité	Benoit Jacquemin	Schneider Electric

CRAN

Schneider Electric

Remerciements

Je tiens à adresser mes remerciements et ma sincère gratitude à mes encadrants universitaires : MM. Jean-François Petin et Jean-Philippe Georges. Ils m'ont apporté durant toute la durée de cette thèse leurs conseils, leurs expériences et leur soutien. Ils ont su être très disponibles en me consacrant une part très considérable de leur temps afin que cette thèse puisse être une réussite. Je tiens également à remercier l'ensemble des membres du jury de cette thèse qui ont accepté d'évaluer mon travail : Mme Maryline Chetto pour avoir accepté de présider ce jury, MM. Armand Toguyeni et Jean-Marc Thiriet pour avoir accepté de rapporter sur mon mémoire de thèse, MM. Lars Kristensen et Thierry Divoux pour avoir accepté d'examiner mon travail.

Je tiens particulièrement à remercier Mr Jacques Camerini qui a été mon manager au sein de l'entreprise Schneider-Electric durant toute cette thèse. Il a été d'un très grand soutien et d'une très grande patience. Il a su m'apprendre à travailler de manière plus efficace, m'a livré l'ensemble des connaissances nécessaires à la réussite de ce projet et bien plus encore. Tout cela avec pédagogie et bonne humeur. Il est et restera pour moi une référence.

Je tiens également à remercier l'ensemble de mes collègues de Schneider-Electric et, plus précisément, l'ensemble de l'équipe « Offre et Technologie » qui a su m'accueillir, m'intégrer et me mettre dans d'excellentes conditions afin que je puisse accomplir cette mission. Je tiens notamment à remercier : M. Antonio ORTIZ pour son intéressement et son soutien pour l'ensemble du projet, M. Xavier Moulin pour ses conseils avisés et la motivation nécessaire qu'il m'a souvent insufflée, M. Patrick Jouvenel pour son écoute, sa disponibilité et ses conseils techniques, M. Francisco Garcia Martin qui a été d'un soutien moral et technique considérable et qui a su mettre une très bonne ambiance dans le bureau que nous partagions. Je remercie également l'ensemble de l'équipe du laboratoire TVDA à travers leur responsable M. Christophe Vincent pour nous avoir aidé à effectuer l'ensemble des tests de validation. Je remercie enfin M. Benoit Jacquemin pour son intérêt et son dévouement pour l'ensemble du projet. Il a su, malgré son agenda et ses responsabilités, être présent autant que possible.

Je tiens finalement à remercier les membres de ma famille pour leur soutien et l'ensemble des sacrifices qu'ils ont acceptés de consentir sans rechigner tout au long de mes études. Plus particulièrement mon père qui s'est tué à la tâche pour me voir, pour nous voir mes sœurs et moi, réussir dans les domaines d'études que nous avions choisis. Je lui dédie exclusivement cette thèse.

Je tiens aussi à remercier ma femme pour sa présence son soutien et d'avoir supporté mes complaintes, mes peurs et mon caractère durant cette période.

A vous tous, ayant contribué de près ou de loin à cette thèse, ma gratitude est infinie.

Tables des matières

Chapitre 3 : Génération automatique de modèles pour l'évaluation de performances

Chapitre 4 : Application sur un cas d'étude

Liste des figures

Liste des tableaux

Introduction générale

Cette thèse a été réalisée dans le cadre d'une convention CIFRE (Convention Industrielle de Formation par la Recherche) entre le Centre de Recherche en Automatique de Nancy (CRAN – UMR n°7039 du CNRS et de l'Université de Lorraine) et le département 'Process Automation' de la société Schneider Electric.

En tant que fournisseur d'offres en automatismes industriels, Schneider-Electric est amené à travers les systèmes intégrateurs (SI) en charge des appels d'offres client, à concevoir en phase d'avant-vente, des architectures de contrôle-commande distribuées sur un réseau de communication. Ces architectures sont constituées d'un ensemble d'équipements supportant l'exécution de fonctions de mesure, d'actionnement, de commandes, de surveillance et de conduite d'un procédé industriel. A titre d'exemple, ces architectures sont souvent constituées de cartes électroniques, d'automates programmables industriels (PLC), de systèmes de commande et de supervision (SCADA), de boitiers d'entrées/sorties déportés pouvant embarquer (ou non) du logiciel et pouvant communiquer à l'aide de réseaux informatiques.

Une proposition commerciale d'architecture doit répondre à un ensemble d'exigences parmi lesquelles les performances temporelles des architectures revêtent une importance particulière. La difficulté inhérente à cette validation des performances, en phase d'avant-vente, est principalement due à deux facteurs :

- dans cette phase, la connaissance des architectures reste limitée, par exemple les fonctions de contrôle-commande ne sont connues que par leur entrées/sorties et sont caractérisées par de multiples imprécisions ou incertitudes – par exemple, le volume et la fréquence des flux des communications ne sont que partiellement connus, les caractéristiques temporelles des équipements sont incertaines puisque dépendantes de leur charge et des traitements à réaliser;
- le nombre des architectures, dont les performances doivent être évaluées, peut-être important dans la mesure où plusieurs solutions d'architectures, reposant sur des technologies variées, sont susceptibles de répondre à un même besoin client.

Pour garantir les performances d'une architecture, les pratiques actuelles reposent essentiellement sur du surdimensionnement engendrant des coûts plus élevés que nécessaire et donc un risque de perte du marché ou bien sur des procédures de tests en plateforme très coûteuse et consommatrice de ressources (en temps et en hommes).

L'objectif de Schneider Electric est donc de développer un outil d'aide au dimensionnement des architectures élaborées en phase d'avant-vente afin de fiabiliser et sécuriser l'offre commerciale. Cet outil est destiné aux SI qui constituent la force de vente de Schneider-Electric. En effet, une estimation prévisionnelle des performances constituerait une valeur ajoutée significative vis à vis des pratiques en vigueur en permettant de sécuriser l'offre commerciale la plus adaptée aux exigences des clients.

D'un point de vue scientifique, l'évaluation des performances temporelles des architectures de contrôle-commande en réseau a fait l'objet de nombreux travaux. Parmi ceux-ci, trois grandes familles émergent :

- les approches algébriques permettant notamment le calcul d'une borne maximale des temps de réponse dans le pire des cas (Le Boudec & Thiran, 2001), (Cruz, 1991), (Addad, 2011a), (Witsch, et al., 2006), (Clarke, et al., 1999), (Baier, et al., 2008),
- les approches basées sur des modèles à état temporisés dont les paramètres sont déterministes qui permettent soit de réaliser des analyses formelles comme la vérification formelle de propriétés soit des simulations de Monte-Carlo, (Ben Hedia, et al., 2005), (Ruel, 2009), (Berthomieu, et al., 2004), (Marsal, 2006a), (Brahimi, 2007)
- les approches basées sur des modèles stochastiques, telles que les chaines de Markov, les réseaux de file d'attente ou encore les réseaux de Petri stochastiques qui peuvent, comme pour les approches déterministes, permettre l'évaluation de performances par d'analyse formelle ou simulation de Monte-Carlo. (Royer, 2006), (Heindl & Reinhard, 2001).

Compte tenu du caractère incertain présent en phase d'avant-vente, nos travaux se sont naturellement orientés vers des approches basées sur des simulations de Monte-Carlo. Parmi les formalismes utilisés dans la littérature, les réseaux de Petri colorés et temporisés (CPN – Coloured Petri Nets) définis par (Jensen, 1994) ont démontré à la fois leur pertinence par un pouvoir d'expression parfaitement adapté à la nature de notre problème (modélisation du temps au travers de transitions à durée fixe ou définie par une distribution de probabilité, compacité des modèles grâce à l'utilisation des couleurs, …) et leur efficacité pour l'estimation de performances (calculs de marquage moyen, de fréquences de franchissement, …).

L'originalité de notre proposition réside dans la définition d'un cadre formel de modélisation autorisant la génération automatique des modèles servant de support à l'évaluation des performances à partir d'une description informelle des architectures. Cette proposition est justifiée par le fait qu'en phase d'avant-vente, les performances doivent être évaluées pour un grand nombre d'architectures et dans un temps court, contraintes difficilement compatibles avec une construction manuelle des modèles.

Le premier chapitre introduit, dans un premier, le contexte et les objectifs industriels de ce travail. Nous précisons, en particulier, les différentes performances temporelles qui devront être évaluées ainsi que les spécificités liées au contexte incertain de l'avant-vente. L'objectif scientifique de la thèse est ensuite présenté, notamment à l'aide de modèles UML (Muller & Gaertner, 2000) formalisant les données du problème. Le chapitre se termine par une présentation des critères qui guideront le choix du formalisme de modélisation et des techniques d'évaluation de performances et qui serviront de référence pour l'analyse de l'état de l'art.

Le deuxième chapitre dresse un panorama des différentes méthodes d'évaluation des performances temporelles des architectures de contrôle-commande en réseau. La première partie est consacrée aux méthodes algébriques permettant le calcul de bornes sur les temps de réponse. La seconde section est consacrée aux modèles à états temporisés (automates à états

finis temporisés, réseaux de Petri et ses extensions temporisées, colorées et hiérarchiques). Nous montrons néanmoins les limites de ces approches pour prendre en compte l'ensemble des contraintes relatives au contexte incertain de l'avant-vente. La dernière section porte sur les modèles stochastiques (chaîne de Markov, réseau de files d'attente, réseau de Petri stochastique) qui permettent pour la plupart d'obtenir une estimation statistique des performances moyennes, minimales et maximales. La synthèse de ces éléments bibliographique et l'analyse de leur adéquation à notre problème nous permet de justifier le choix du formalisme des réseaux de Petri temporisés, colorés et hiérarchiques proposés par (Jensen, 1994).

Le troisième chapitre présente notre contribution. Elle porte sur la définition formelle, à l'aide de réseaux de Petri colorés et temporisés, d'un modèle « constructeur » d'architectures embarquant des mécanismes de configuration, d'instanciation et de paramétrage. Plusieurs algorithmes sont proposés pour, d'une part, construire automatiquement le modèle d'une architecture donnée, à partir d'une description informelle de sa topologie et d'une librairie de modèles d'équipements de contrôle-commande, et, d'autre part, pour générer les observateurs requis à partir d'une description informelle des performances à évaluer.

Le quatrième chapitre présente, dans une première partie, un prototype d'outil d'aide au dimensionnement qui implante les différents algorithmes développés au chapitre 3. Ce prototype est interfacé, d'une part avec l'outil utilisé par les ingénieurs en phase d'avant-vente pour décrire les architectures, et, d'autre part avec le simulateur de l'outil CPN Tools (Jensen, et al., 2007) qui permet une estimation des performances via des simulations de Monte-Carlo. Notre contribution est ensuite illustrée, dans une seconde partie, sur quelques exemples d'architectures types fournies par la société Schneider Electric. Nous nous attachons, en particulier, à montrer que l'évaluation de plusieurs offres d'architecture ne nécessite aucun effort de modélisation supplémentaire autre que la représentation graphique des différentes architectures à évaluer.

Le mémoire se termine par une présentation des principales conclusions et perspectives de recherche de ce travail.

Chapitre 1
Contexte et objectif

1 Introduction

Ce chapitre présente le contexte de l'étude, la problématique industrielle ainsi que la problématique scientifique qui en découle. Afin de comprendre la nature du problème soulevé, une définition d'une architecture de contrôle commande est proposée dans la première section de ce chapitre. La section suivante présente les performances attendues d'une architecture de contrôle-commande et les indicateurs utilisés pour leurs évaluations. Nous présentons ensuite les contraintes relatives au contexte industriel de la phase d'avant-vente et nous détaillons les pratiques actuelles en termes d'évaluation compte tenu de ces contraintes. Les limites de ces pratiques justifient l'objectif industriel en termes d'outil d'aide au dimensionnement des architectures. La dernière section de ce chapitre extrait de ce besoin industriel la problématique scientifique qui fera l'objet de notre contribution.

2 Architecture de contrôle-commande

Une architecture de contrôle-commande est un ensemble de composants matériels et logiciels dédiée à la réalisation de services d'automatisation pour un procédé industriel. Ces différents services sont relatifs (Figure 1):

- à la mesure et l'actionnement en interaction avec le procédé (lecture, filtrage et traitement d'un signal, conversion analogique – numérique, gestion de puissance, …),
- à la commande du procédé permettant de maintenir l'état de celui-ci dans une plage de fonctionnement donnée (notamment via l'émission de requêtes à destination des actionneurs),
- à la surveillance et la conduite du procédé (surveillance des grandeurs caractéristiques du procédé, gestion des alarmes, visualisation de l'état du procédé et des chaines d'actionnement de la transmission des requêtes au système de commande.

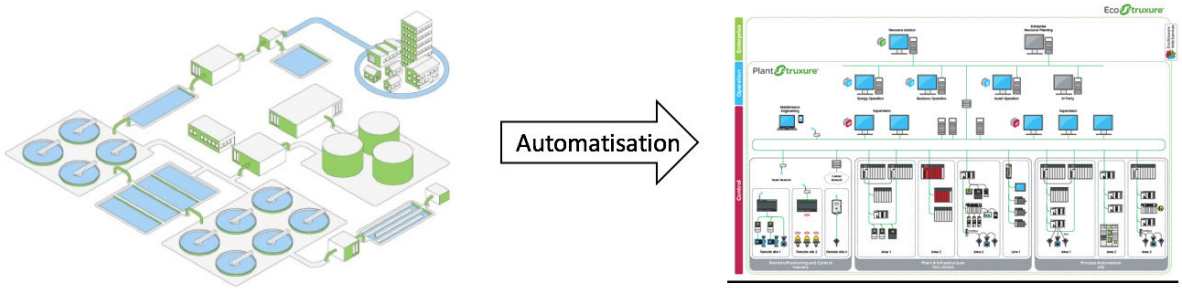


Figure 1: Exemple d'une architecture de C/C pour un procédé industriel

Une architecture de contrôle-commande peut se décrire sous la forme d'architectures fonctionnelles, d'architectures matérielles et d'architectures opérationnelles.

2.1 Architecture fonctionnelle

L'architecture fonctionnelle est définie pour un procédé que l'on suppose connu et a pour objectif d'identifier l'ensemble des services de contrôle-commande nécessaires pour réaliser la

mission de l'installation et sa mise en sécurité et de structurer ces fonctions sous la forme d'un réseau de fonctions hiérarchisées et interconnectées.

La hiérarchisation permet de structurer les fonctions de contrôle-commande selon différents niveaux d'abstraction et d'en proposer une classification efficace. A titre d'exemple, la norme ANSI/ISA-95.00.01-2000 (Scholten, 2007) propose six niveaux fonctionnels hiérarchiques comme indiqué sur la Figure 2. Cette structuration dépend néanmoins fortement des types de procédés automatisés et des pratiques industrielles et peut donc être variable selon les contextes.

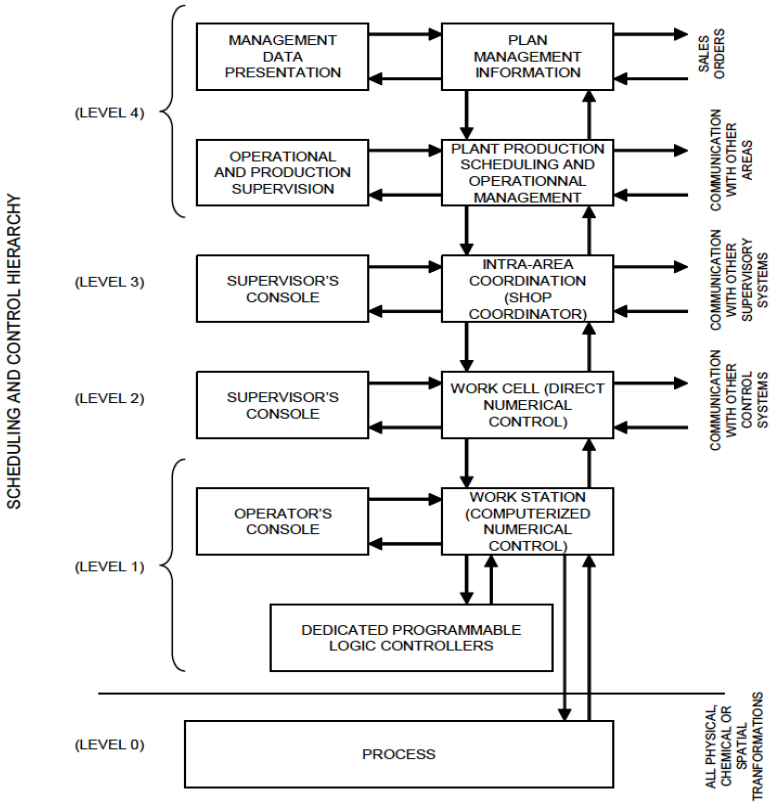


Figure 2: Structure fonctionnelle par ANSI/ISA-95.00.01-2000 (Scholten, 2007)

L'interconnexion des fonctions au sein d'un réseau a pour objectif d'identifier les échanges d'informations entre fonctions, sous la forme de relations entre les entrées consommées par une fonction donnée, les informations produites par d'autres fonctions, entre les fonctions et le procédé, c'est-à-dire les informations émises (ou reçues) à destination (ou en provenance) du procédé.

Ces informations, que nous appellerons par la suite les informations d'entrées/sorties, ont un impact direct sur le dimensionnement de l'architecture de contrôle-commande puisqu'elles induisent, pour chaque fonction, les caractéristiques d'interface avec le procédé (cartes d'entrées/sorties) dont celles-ci auront besoin. Ces informations seront regroupées dans un diagramme appelé dans la littérature P&ID (Piping & Instrumentation Diagram) (Smith, 2016) (Béla, 2013).

2.2 Architecture matérielle

L'ensemble de ces fonctions est exécuté sur des matériels très divers tels que des cartes électroniques, des Automates Programmables Industriels (API ou PLC), des systèmes de supervision (SCADA), pouvant embarquer (ou non) du logiciel et pouvant communiquer à l'aide de réseaux informatiques ou de liaisons filaires électriques. L'architecture matérielle décrit l'organisation structurelle de ces différents équipements. De même que pour les fonctions, l'architecture matérielle est généralement construite de manière hiérarchisée selon les familles d'équipements et l'organisation des réseaux de communication. Les structures les plus classiques font apparaître 4 niveaux représentés sur la Figure 3 (Scholten, 2007), des équipements de pilotage de l'entreprise jusqu'aux équipements de terrain :

- Niveau 0 : regroupant l'ensemble des composants terminaux dédiés à l'exécution et au traitement des différentes tâches du procédé industriel (e.g. variateur de vitesses, départ moteurs, capteurs, actionneurs). L'évolution des technologies d'actionnement et de mesure se traduit par le fait que ces équipements matériels embarquent de plus en plus souvent des logiciels de traitement,
- Niveau 1, regroupant l'ensemble des composants dédiés à la commande du procédé d'automatisation (e.g. les automates programmables industriel PLC). Il sert aussi de lien entre l'unité de production et la salle de supervision du procédé en envoyant périodiquement le statut du procédé industriel.
- Niveau 2, regroupant l'ensemble des composants dédiés à la supervision du procédé d'automatisation (e.g. system de commande et d'acquisition de données SCADA). Ces systèmes sont souvent embarqués sur des serveurs capables de récupérer et d'envoyer des informations au procédé industriel et sont en interaction avec des opérateurs humains.
- Niveau 3, regroupant l'ensemble des composants logiciels dédiés à la planification et à l'optimisation du procédé d'automatisation (par exemple ERP).

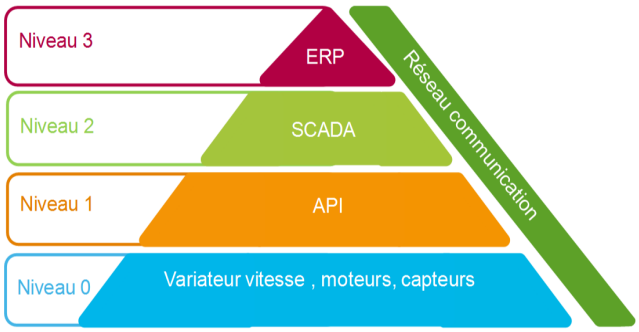


Figure 3: Niveaux d'architecture de contrôle commande (Process automation, 2013)

Afin de réaliser les différentes tâches d'automatisation, les composants échangent des informations via un réseau de communication. La fréquence et la taille de ces informations dépendent de la nature du composant et des services qu'il exécute. Lors de ces échanges, un délai est induit entre l'émission de l'information, sa réception et son traitement par le(s) composant(s) récepteur(s). Ces délais sont liés aux temps de traitements et de transmission ainsi qu'aux ressources disponibles.

2.3 Architecture opérationnelle

L'architecture opérationnelle peut être définie comme la projection de l'architecture fonctionnelle sur l'architecture organique, comme le montre la Figure 4.

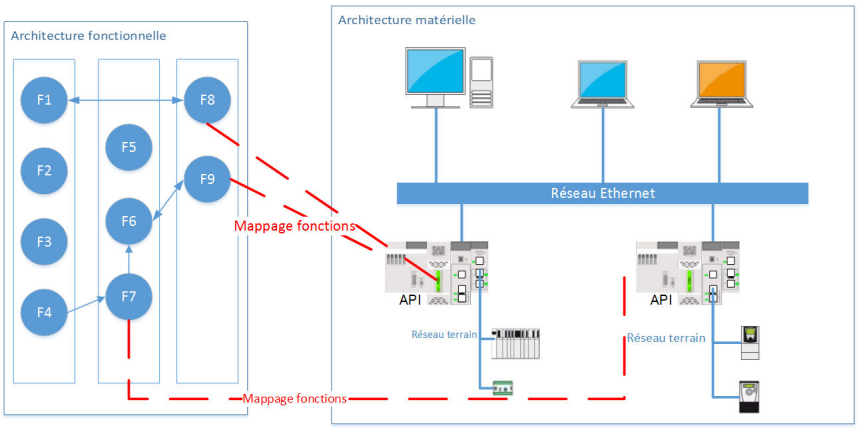


Figure 4: Architecture opérationnelle

Elle contient l'ensemble des informations caractérisant le contexte opérationnel du système de contrôle-commande et en particulier :

- les caractéristiques des fonctions et des équipements identifiées respectivement dans les architectures fonctionnelles et matérielles,
- une matrice d'allocation représentant les relations entre un ensemble de fonctions élémentaires de contrôle-commande et un ensemble d'équipements,
- la caractérisation des flux de communication entre équipements induits par l'allocation des fonctions sur les matériels.

3 Performances des architectures de contrôle - commande

Les performances des architectures de contrôle-commande peuvent être évaluées selon de multiples critères :

- fonctionnels : performance des algorithmes de commande aux regards des exigences et scénarios d'exploitation, performances de fonctionnement en conditions limites, performances comportementales face à des aléas ou stimuli non prévus, …
- non fonctionnels : contraintes de sûreté de fonctionnement, de sécurité, d'ergonomie,

Ce mémoire se focalise sur l'analyse des performances temporelles.

3.1 Performances temporelles

Les performances temporelles seront analysées selon deux critères : temps de réponse et temps de cycle des automates programmables industriels (PLC).

3.1.1 Temps de réponse

Le temps de réponse dans les architectures de contrôle commande se définit comme étant le temps aller et retour entre une requête émise par un équipement client vers un équipement

serveur qui traite la requête et qui renvoie une réponse ou un acquittement vers le client (Jasperneite & Neumann, 2001). Cette communication client-serveur peut se faire entre composants sur un même niveau hiérarchique, ou entre deux composants sur deux niveaux hiérarchiques différents.

Concernant les performances pour les communications entre deux niveaux distincts, elles se résument principalement à la communication entre le SCADA, les PLC et les dispositifs terminaux. Les performances peuvent être identifiées, comme l'indique la Figure 5, par le temps aller et retour de l'application qui correspond aux temps de réponse entre l'émission d'une commande au niveau du SCADA, le traitement de la commande par l'automate et le retour d'exécution de la commande.

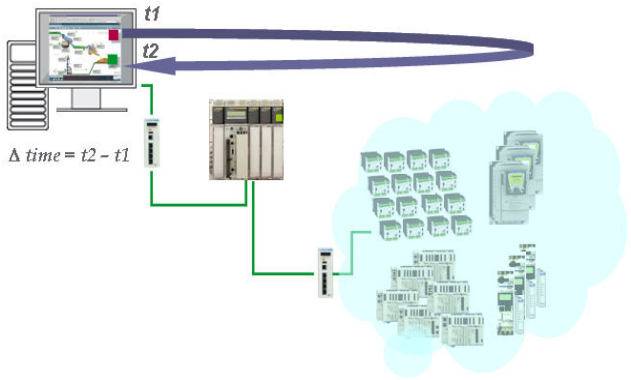


Figure 5: Temps de réponse aller-retour entre station de travail et équipent terminaux

A cela s'ajoute les performances liées aux temps de mise à jour du système. Comme l'indique la Figure 6, il s'agit du temps entre l'activation d'un signal d'entrée au niveau des E/S distant (Figure 6 gauche) ou le changement d'une valeur au niveau de l'automate (Figure 6 droite) et la notification visuelle de ce changement au niveau du SCADA.

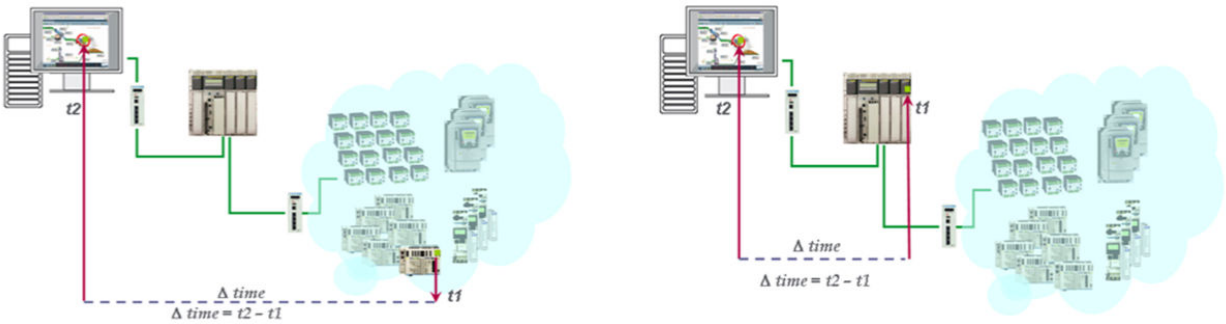


Figure 6: Temps de remontée d'un événement au SCADA

Les PLC, en plus d'être des équipements « serveur », peuvent initier des communications, c'est-à-dire être « client ». Dans ce cas comme l'indique la Figure 7, les performances se définissent, par le délai entre une entrée physique sur un équipement jusqu'à la sortie physique correspondante sur le même équipement ou sur un autre. Cette performance se caractérise par la communication entre les PLC et les entrées et sorties déportées.

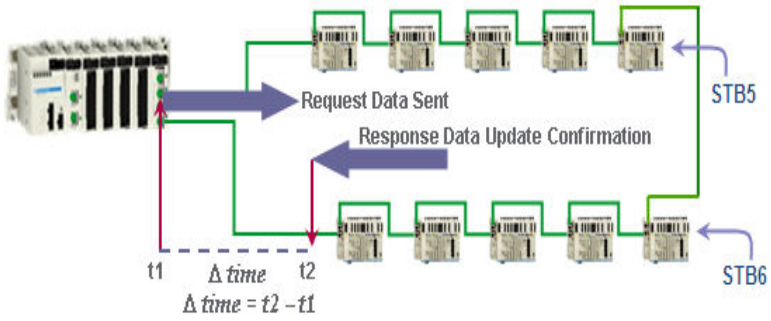


Figure 7: Temps de réponse aller-retour entre PLC et équipements terminaux

Dans le cas particulier des architectures de contrôle-commande basées sur Ethernet, l'évaluation des temps de réponse est confrontée au problème de l'indéterminisme du protocole Ethernet lié à la méthode de contrôle d'accès au médium. En effet, cet indéterminisme implique des performances instables sous un trafic à haute densité et des performances non-bornées (Wheels, 1993).

Face à ce problème, plusieurs évolutions ont été apportées, notamment par l'introduction de commutateurs Ethernet de niveau 2 (Felser, 2005). Ceux-ci séparent le domaine de collisions et permettent de créer des réseaux locaux virtuels. Ils permettent de réduire les collisions grâce à la transmission full-duplex et le recours à des topologies point à point (Lee & Lee, 2002). L'évolution du standard vers les réseaux Ethernet commutés et full duplex ont offert la possibilité d'avoir une communication en temps réel, de réduire le temps de latence, la gigue, les collisions et les pertes de paquets (Zhang, et al., 2001) . Celle-ci a ainsi permis de réduire significativement l'indéterminisme du standard Ethernet. Cette évolution ne permet cependant pas de garantir le déterministe pour les architectures de contrôle-commande. En effet les composants réseaux basés sur le protocole Ethernet, peuvent faire face à une congestion interne due à l'envoi successif de messages vers un seul et même port de sortie pouvant entrainer une perte de paquets. De plus, les architectures d'automatismes à base d'Ethernet facilitent la mise en œuvre de réseaux non isolés qui laissent la porte ouverte à de multiples échanges de type broadcast pouvant aggraver les problèmes de congestion et affecter la performance globale (Walsh, et al., 2001).

Le réseau n'est cependant pas le seul élément impactant les performances de temps de réponse. En effet, l'augmentation des services non directement dédiés aux tâches de commandes (service de messagerie, télémaintenance, service web etc.) dans les équipements d'une architecture de contrôle-commande peuvent conduire à une surcharge de ces équipements dont les ressources sont limitées. Ce problème concerne principalement les automates programmables industriels PLC, ce qui justifie une analyse plus fine de leurs performances qui fait l'objet du paragraphe suivant.

3.1.2 Temps de cycle PLC

Les PLC possèdent des processeurs ayant des ressources limitées en termes de capacité de traitement et de temps de traitement qui peuvent devenir critiques compte tenu de la multiplicité des services qu'ils doivent assurer (gestion des entrées/sorties, messagerie, web, …). Une

information transmise au PLC doit donc attendre la disponibilité d'une ressource « processeur » avant d'être traitée, ce qui engendrer des retards (Jasperneite & Neumann, 2001).

Dans la communication client-serveur, il est important de définir la performance liée au temps de cycle des PLC. En effet dans ces types de communications, la notion de parallélisme entre les émissions de requêtes asynchrones de plusieurs clients vers un même serveur introduit des temps d'attente, des problèmes de disponibilité et de partage de ressources, ceci de manière variable en fonction de l'intensité du flux de communication (Witsch, et al., 2006).

Durant leur cycle, les PLC doivent traiter l'ensemble des requêtes émises par les clients qui lui sont parvenues, lire les données de ses entrées, traiter sa tâche de logique d'automatisation en fonction du procédé, écrire des données sur ses sorties et finalement initier des communications avec d'autres PLC (Figure 8).

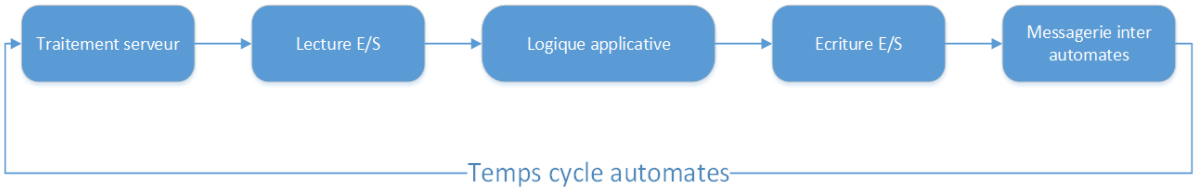


Figure 8: Etapes de traitement des tâches par le processeur d'un PLC

Ce temps de cycle (dans le cas d'un fonctionnement cyclique du PLC) n'est pas borné car dépendant exclusivement du nombre de requêtes à traiter et des capacités techniques de l'automate. Ainsi le temps de cycle n'est plus une valeur statique dans les architectures de contrôle et de commande, mais variable dont l'influence dans les performances globales en termes de temps de réponse est considérable.

4 Evaluation des performances d'une architecture de contrôle-commande

L'activité d'évaluation des performances diffère selon les phases du processus d'ingénierie où elle est effectuée et selon le type d'indicateurs que l'on cherche à mesurer. En effet, les outils/méthodes permettant l'évaluation de performances dépendent fortement de ses objectifs (notamment en termes de précisions), de ses contraintes (niveau de connaissance des architectures) et des ressources qui lui sont allouées (en temps et en hommes).

4.1 Contraintes relatives au processus d'ingénierie : cas particulier de l'avant-vente

Le processus d'automatisation est un ensemble d'activités corrélées qui conduisent à la réalisation d'un système répondant aux finalités qui lui sont assignées.

Il se base généralement, en termes de données d'entrées, sur un cahier des charges défini par le client ayant un procédé à automatiser contenant deux éléments principaux :

- les spécifications du procédé qui décrivent le type, la nature et les caractéristiques du procédé à automatiser ainsi que l'environnement de déploiement de la future architecture ; les spécifications contiennent notamment la liste des entrées/sorties associées aux capteurs et actionneurs ;

- les spécifications fonctionnelles qui décrivent les services de commande, de surveillance, ou de supervision à développer ainsi que les performances fonctionnelles et non fonctionnelles attendues (Auroy, 2000).

En réponse à ces exigences, le traditionnel cycle de développement en V adapté au développement d'un système automatisé (Figure 9) parallélise les processus de conception des architectures fonctionnelle et matérielle, suivi de manière séquentielle par la définition de l'architecture opérationnelle. Néanmoins, cette organisation peut être variable selon les systèmes considérés (manufacturier, énergie, chimie, …).

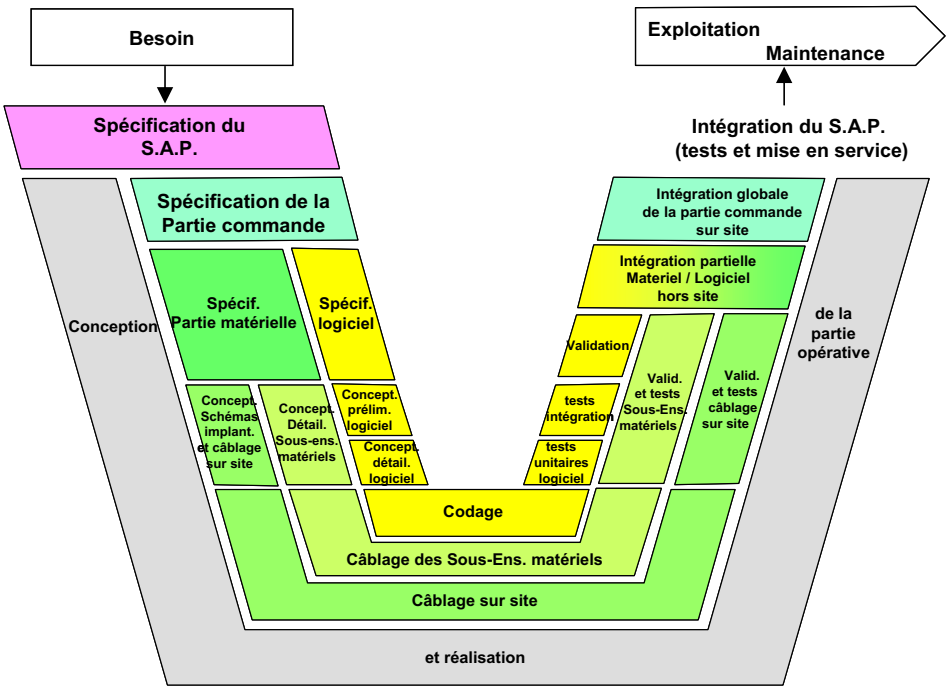


Figure 9: Cycle de vie d'un projet d'automatisation

L'évaluation des performances d'une architecture de contrôle-commande peut être effectuée :

- lors de la phase descendante du cycle en V, il s'agit alors de procéder à des évaluations prévisionnelles des performances sur la base des modèles d'architectures fonctionnelles (modèles comportementaux, (Cottet & Grolleau, 2005), modèles d'informations, modèles structurels (Ward, 1986), (Hatley & Pirbhai, 2013), …), des modèles d'architectures matérielles et des modèles d'architectures opérationnelles. En effet, les performances sont intimement liées à l'architecture fonctionnelle (notamment pour le dimensionnement des fonctions à exécuter par l'architecture matérielle), à l'architecture matérielle (notamment pour les performances intrinsèques des équipements mis en œuvre) et à l'architecture opérationnelle (notamment en termes de communication entre fonctions, dépendante de l'allocation des fonctions sur les équipements).
- lors de la phase ascendante du cycle en V, il s'agit alors de réaliser des campagnes de test d'une architecture opérationnelle réelle

Situé en amont de ce cycle en V classique, la phase d'avant-vente présente des spécificités qui auront un impact direct sur les approches mises en œuvre pour l'évaluation de performances. Cette phase correspond à l'étape d'étude et de définition d'une proposition commerciale pouvant satisfaire aux exigences clients définies dans le cahier des charges.

L'objectif est de proposer aux clients une ou plusieurs solutions techniques pouvant, d'une part, satisfaire ses exigences et, d'autre part, créer de la valeur ajoutée à l'entreprise. Cette solution technique se présente, dans la plupart des cas, sous la forme d'une architecture matérielle dans la mesure où l'étude et la conception de l'architecture fonctionnelle ne peut et n'a pas été réalisée durant cette phase.

En d'autres termes, la phase d'avant-vente se focalise sur la conception d'une architecture matérielle dans un contexte incertain principalement due à l'imprécision des informations relatives à l'architecture fonctionnelle :

- l'architecture fonctionnelle est limitée à une liste des fonctions de contrôle-commande et une évaluation très grossière de la charge qu'elles engendrent (par exemple une taille standard pour les traitements de type logique ou analogique),
- le contexte d'utilisation des fonctions de contrôle-commande n'est que partiellement connu, en particulier le volume et la fréquence des flux de communications et des sollicitations ne peuvent être qu'estimés puisque les échanges entre fonctions de contrôle-commande ne sont pas identifiés de manière précise
- le comportement des équipements (temps de réponse) ne peut qu'être estimé puisque dépendant de l'allocation des fonctions de contrôle-commande et des services qu'ils auront à exécuter (seules les caractéristiques intrinsèques des équipements sont connues).

Néanmoins, le système intégrateur chargé de la proposition d'une offre commerciale doit respecter quatre règles principales qui sont le Coût, la Qualité, les Fonctionnalités et le Délai (CQFD).

Concernant le coût de la solution proposée, la force de vente doit trouver un équilibre entre un prix raisonnable afin d'avoir des chances de gagner le contrat, mais aussi créer le maximum de plus-values pour l'entreprise.

Elle doit aussi s'assurer que la solution proposée répond à un niveau de qualité optimale justifiant le prix proposé. Elle doit garantir que la solution proposée respecte l'ensemble des fonctionnalités et des performances attendues par le client (Rony, et al., 2013). Enfin, elle doit s'assurer que l'équipe d'intégration en charge du déploiement de l'architecture possède les compétences techniques et les ressources matérielles et financières afin d'exécuter le projet dans les délais répartis.

D'autre part, la qualité d'une force de vente étant mesurée par sa capacité à avoir un taux de succès élevé (Anderson & Richard, 1987), elle se doit de répondre à un maximum d'appels d'offre tout en proposant une offre techniquement cohérente et financièrement intéressante. Les délais accordés à l'élaboration d'une proposition s'en trouvent donc extrêmement contraints.

Le contexte d'avant-vente est donc caractérisé par :

- une connaissance partielle des architectures due à des informations imprécises ou non disponibles,
- des ressources en termes de temps et de personnels fortement limitées,
- la nécessité d'élaborer une offre commerciale pertinente constituée de plusieurs solutions d'architectures correspondant, pour le client, à différents compromis coûts / satisfaction des exigences.

4.2 Indicateurs de performances

Deux familles d'indicateurs permettent en général d'évaluer les performances temporelles et de charge mentionnées en section 3 :

- les indicateurs de performances moyennes (temps moyen de réponse, temps de cycle moyen d'un processeur, …) souvent accompagnées d'un indice de confiance (écart-type, demi-longueur d'un intervalle de confiance, …),
- les indicateurs de performances maximales ou minimales (délai maximal de bout en bout, charge maximale d'un processeur) obtenues dans les pires des cas.

La première famille d'indicateur permet d'obtenir un niveau de satisfaction des exigences attendues vis à vis des performances requises. Elle repose souvent sur une estimation statistique des performances issues de différents scénarios de tests ou de simulations. En revanche, elle n'apporte aucune preuve quant au respect des performances. Cette preuve est, en générale, obtenue au travers de la seconde famille d'indicateurs qui envisage les situations les plus pénalisantes d'un point de vue des performances et s'avère très utile dans le cas des applications critiques (aéronautique (Charara, 2007), industrie nucléaire (Marsal, 2006a), système ferroviaire (Mekki, et al., 2010) …).

Compte tenu des objectifs et du contexte incertain de la phase d'avant-vente, la première famille d'indicateurs, relative aux performances moyennes, sera privilégiée dans le cadre de cette étude.

4.3 Pratiques industrielles pour l'évaluation de performance en avant-vente

Afin de garantir les performances de leurs architectures en phase d'avant-vente, les systèmes intégrateurs utilisent principalement 3 méthodes :

- L'évaluation à base d'architectures de références,
- Le surdimensionnement,
- Les tests en laboratoire.

La méthode basée sur les architectures de référence consiste à concevoir l'architecture du client en se basant sur des architectures « type » testées, validées et documentées. Ces architectures de référence sont déployées dans les laboratoires du fabriquant de système où les performances sont mesurées. A la définition de l'offre, le système intégrateur sélectionne dans la librairie des architectures de référence celle se rapprochant le plus du type de procédé décrit dans le cahier des charges. Il conçoit ensuite son architecture en ne faisant que des adaptations

mineures de l'architecture de référence, espérant ainsi avoir des performances similaires. L'avantage principal de cette méthode est, d'une part, le faible temps requis pour établir une offre et, d'autre part, le peu de ressources. Cependant, chaque procédé nécessitant des fonctions de contrôle-commande spécifiques, le système intégrateur s'expose au risque d'avoir des écarts entre les performances espérées lors de la définition de l'architecture et celle constatées durant la phase d'intégration et de validation. Dans certains cas, ces écarts peuvent entrainer des modifications considérables de l'architecture ayant un impact financier important.

La seconde approche consiste à proposer une architecture surdimensionnée sur les composants qui peuvent s'exposer à un trafic à haute densité et ainsi entrainer des goulots d'étranglements, en choisissant des composants dont les spécifications techniques en termes de ressources couvrent très largement les besoins. Grâce à cette méthode, le système intégrateur est assuré d'avoir des performances en adéquation avec les exigences du client, quelles que soient les conditions d'utilisation du système. L'inconvénient principal est le coût final de l'offre commerciale, dans la mesure où celui-ci est corrélé avec la puissance des composants dont elle est composée. Ce qui entraine un risque commercial lié à la perte du contrat.

La dernière méthode consiste à pratiquer des tests en laboratoire. A la définition de l'architecture, le système intégrateur va déployer l'architecture dans ses laboratoires et évaluer ses performances. Si celles-ci correspondent à celles désirées par le client, l'intégrateur système valide l'architecture et fait sa proposition commerciale. Dans le cas contraire, il modifiera l'architecture en fonction des performances obtenues et relancera une nouvelle campagne de test. L'avantage principal de cette méthode est d'offrir une double garantie, d'une part, sur les performances de l'architecture et, d'autre part, sur la pertinence de l'offre commerciale. En effet, l'architecture ne sera ni sur dimensionnée, ni sous dimensionnée. En revanche, l'inconvénient principal de cette méthode est lié aux ressources financières et humaines déployées durant la phase d'avant-vente sans aucune certitude sur l'issue positive de la démarche commerciale.

	Garantie performances	Sécurisation offre	Ressources utilisées
Référence architecture	Faible	Moyen	Très Faible
Surdimensionnement	Très Fort	Très faible	Faible
Test laboratoire	Fort	Très Fort	Très Fort

Tableau 1: Avantages et inconvénients des méthodes actuelles

Le tableau 1 ci-dessus résume les avantages et les inconvénients de chaque méthode. Les limites de ces différentes méthodes ont incité Schneider Electric à développer une méthodologie d'évaluation des performances en phase d'avant-vente basée sur les modèles.

5 Problématique

Cette section présente la problématique industrielle centrée sur la mise à disposition aux ingénieurs en avant-vente d'un outil d'aide à l'évaluation de performances des architectures et de la problématique scientifique qui en découle.

5.1 Besoin industriel

Partant du constat que la validation d'une architecture doit se faire au plus tôt (Denis, 1994) (Goodwin, et al., 2001), l'objectif est de fournir aux systèmes intégrateurs un outil leur permettant de procéder à une estimation, la plus fiable possible, des performances d'une architecture. Cet outil devra se baser sur une modélisation formelle de l'architecture intégrant l'incertain du contexte avant-vente et proposer des mécanismes de calcul des performances.

Afin de fixer les besoins des futurs utilisateurs, une campagne d'interview a été réalisée. Elle permet notamment d'identifier clairement les informations qui devront être fournies en entrée de l'outil ainsi que les principales contraintes et exigences auxquelles celui-ci devra se conformer. Les interviews réalisées auprès d'un panel représentatif des futurs utilisateurs de Schneider Electric sont divisées en deux parties : la première a pour but de recueillir les principaux problèmes auxquels ils sont confrontés pour fournir une offre commerciale, notamment en termes de sécurisation de l'offre, la seconde a pour but de recueillir leurs propositions relatives à une méthode d'évaluation de performances pouvant leur apporter une aide.

L'analyse de la première partie des interviews a montré que la grande majorité des sondés estime que la connaissance des performances de leurs architectures avant de faire une proposition commerciale est primordiale dans la mesure où le fait de ne pas maitriser ces performances peut avoir un impact très critique sur leur business. Le constat qu'ils dressent est qu'ils ne disposent aujourd'hui d'aucun outil efficace leur permettant d'évaluer les performances compte tenu des limites des 3 méthodes utilisées actuellement (voir section 4.3.).

L'analyse de la deuxième partie des interviews a mis en exergue le besoin d'un outil leur permettant d'évaluer les performances de leurs architectures en avant-vente. Dans la vision industrielle, cet outil est considéré comme une aide au dimensionnement des architectures en permettant d'évaluer le comportement de l'architecture sans la déployer, d'anticiper les possibles goulots d'étranglements par rapport aux performances évaluées et de redimensionner si nécessaire cette architecture.

Cette réorientation de l'outil comme une aide au dimensionnement a une conséquence forte en termes de besoins : l'outil développé ne devra pas permettre l'évaluation d'une seule et unique architecture mais d'une multiplicité d'architectures élaborées par un jeu d'essais/erreurs pour répondre aux exigences des clients. En effet, le nombre de solutions d'architectures admissibles est souvent très grand. L'intégrateur système souhaite donc pourvoir en évaluer un grand nombre afin de sélectionner celle ayant le meilleur compromis entre satisfaction des exigences des clients et plus-value pour l'entreprise. Enfin, une exigence incontournable est que la description des architectures en entrée de l'outil devra se faire à partir d'une interface graphique ne nécessitant pas d'expertise particulière sur les formalismes sous-jacents aux modèles d'évaluation de performances.

Sur la base de ces interviews, le cahier des charges de l'outil peut se synthétiser de la manière suivante (Figure 10) :

- *Entrées de l'outil* :
  - la description des architectures matérielles de contrôle-commande étant réalisée de manière graphique par les utilisateurs, son interfaçage avec l'outil d'évaluation nécessitera la définition formelle d'un format d'échange qui fera l'objet de la section suivante ;
  - la description des performances à évaluer sera également fournie de manière graphique ou textuelle, ce qui nécessitera également la définition d'un format d'échange pour son traitement par l'outil d'évaluation ;
- *Fonctionnalités* :
  - l'outil devra permettre l'évaluation quantitative intégrant la marge d'erreur (par exemple sous la forme d'un intervalle de confiance) des performances suivantes : temps de réponse, temps de cycle PLC, performances de charges ;
  - l'outil devra permettre d'effectuer de nombreuses itérations (propositions d'architectures / évaluation des performances) jusqu'à l'obtention d'une solution satisfaisante dans un temps court compatible avec les contraintes d'avant-vente ; si les performances évaluées sont en adéquation avec celle exigées par le client, alors l'architecture sera considérée comme admissible ; dans le cas contraire, le système intégrateur pourra alors, après analyse des résultats, modifier l'architecture et relancer une nouvelle campagne d'évaluation de performances.
- *Sorties de l'outil* :
  - les performances évaluées seront fournies aux utilisateurs sous forme graphique ou textuelle.

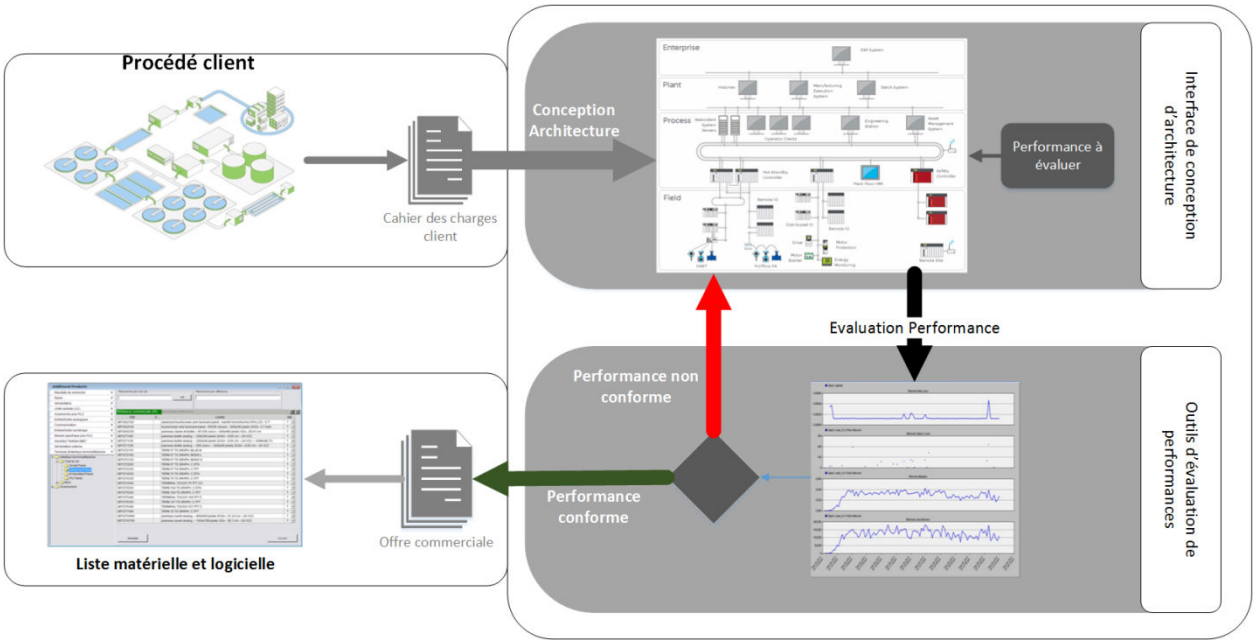


Figure 10: Cahier des charges pour un outil d'évaluation de performances

Pour Schneider Electric, la valeur ajoutée au regard des méthodes utilisées aujourd'hui peut se mesurer selon les trois critères : garantie de performances, sécurisation de l'offre et ressources utilisées. La Figure 11 représente la valeur ajoutée attendue de l'outil d'évaluation en considérant une échelle allant de 1 à 5 pour mesurer le degré croissant de satisfaction de ces trois critères.

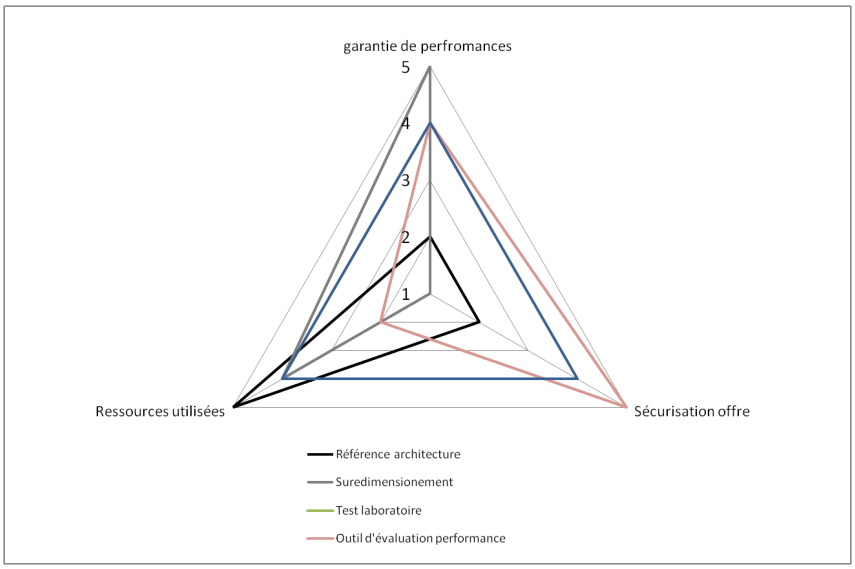


Figure 11: Indicateurs de comparaison pour les méthodes d'évaluation de performances

5.2 Formalisation du problème

Le problème industriel posé porte sur l'évaluation quantitative des performances d'un grand nombre d'architectures matérielles dans un contexte incertain où les informations relatives à l'architecture fonctionnelle ainsi qu'aux scenarios d'exploitation sont imprécises ou absentes. D'un point de vue scientifique, ce problème se traduit comme un problème d'élaboration des modèles formels servant de support aux mécanismes de calcul des performances dans un temps court compatible avec les contraintes de la phase d'avant-vente. Dans l'idéal, les modèles d'évaluation de performances devraient donc pouvoir être obtenus de manière automatique, à partir de la description non formelle des architectures et des performances à mesurer.

5.2.1 Formalisation des données d'entrées

La description non formelle se fait à partir de données d'entrées contenant l'ensemble des informations relatives à l'architecture et ses performances. Ces données d'entrées sont modélisées à travers deux modèles UML dont le premier représente une description de l'architecture et le deuxième représente une description des performances à évaluer.

Ces modèles formalisent donc les deux grandes familles d'informations qui seront supposées connues par l'outil d'aide au dimensionnement des architectures qui fait l'objet de ce travail. En pratique, ces informations devront être saisies par les ingénieurs en charge d'élaborer des propositions commerciales d'architecture sur des outils informatiques qui leur sont propres. Elles seront exportées vers l'outil d'évaluation des performances, développé dans

le cadre de cette thèse et présenté au chapitre 4, à l'aide d'un fichier XML dont le format est basé sur le standard SGML (Goldfarb & Rubinsky, 1990).

5.2.1.1 Données relatives aux architectures

Comme l'indique la Figure 12, il y a 5 grandes familles dans les architectures de contrôle-commande, dont 4 familles regroupant l'ensemble des composants assurant des fonctions de mesure et d'actionnement, de commande, de supervision et de pilotage du procédé et une famille « réseau » représentant l'ensemble des équipements de communications.

Chaque famille est représentée dans le modèle UML par une classe contenant des attributs. Ceux-ci représentent des paramètres d'entrées inhérents à la famille de ces composants. Certaines classes de familles possèdent une relation de composition avec des sous-classes contenant aussi des attributs et représentant un sous-composant matériel dont les attributs constituent également des paramètres d'entrées. L'ensemble des paramètres (attributs pour classe de famille et sous-classes) constituent les données d'entrée de notre problème et sont supposées être fournies par les ingénieurs en charge de l'élaboration d'une architecture.

La classe des PLC est composée de trois sous-classes représentant les trois sous-composants constituant un PLC : le fond de panier, le processeur et le coupleur de communication. La classe « PLC » possède un attribut unique caractérisant le type de PLC. Les sous-classes « processeur » et « coupleurs de communication » ont une relation de composition avec la classe « fond de panier » qui, à son tour, possède une relation de composition avec la classe PLC. La sous-classe « coupleur de communication » possède deux attributs qui sont le type de coupleur et la capacité de traitement des requêtes. La sous-classe « fond de panier » ne possède qu'un seul attribut qui est le nombre de port de connections. Enfin, la sous-classe « processeur » possède trois attributs qui sont la taille de son buffer, le nombre de requêtes traitées par cycle et enfin la durée du programme automate. La valeur de ces paramètres d'entrées dépend du type d'automate dont la valeur est fournie par l'attribut de la classe « PLC ».

La classe « SCADA » est également composée de trois sous-classes représentant les trois types de fonctions assurées un serveur de supervision et de commande : alarmes, affichage, courbes de tendances. Cette classe est composée d'un seul attribut caractérisant le type de serveur. L'ensemble des sous-classes possède 2 attributs qui sont le nombre de requêtes à envoyer et la période d'envoi. Contrairement aux attributs des sous-composants des PLC, les valeurs de ceux du SCADA sont renseignées par l'ingénieur configurant l'architecture.

La classe « Client » possède deux attributs qui sont relatifs au type de client et à la période d'envoi de requêtes. Les valeurs des attributs sont déterminées par l'utilisateur. Ces clients sont constitués d'applications tierces de pilotage et de gestion du procédé pouvant correspondre à des fonctions de Niveau 3 (planification, ERP, …). Comme pour la classe « SCADA », les attributs de la classe « Client » sont renseignés par les ingénieurs élaborant l'architecture.

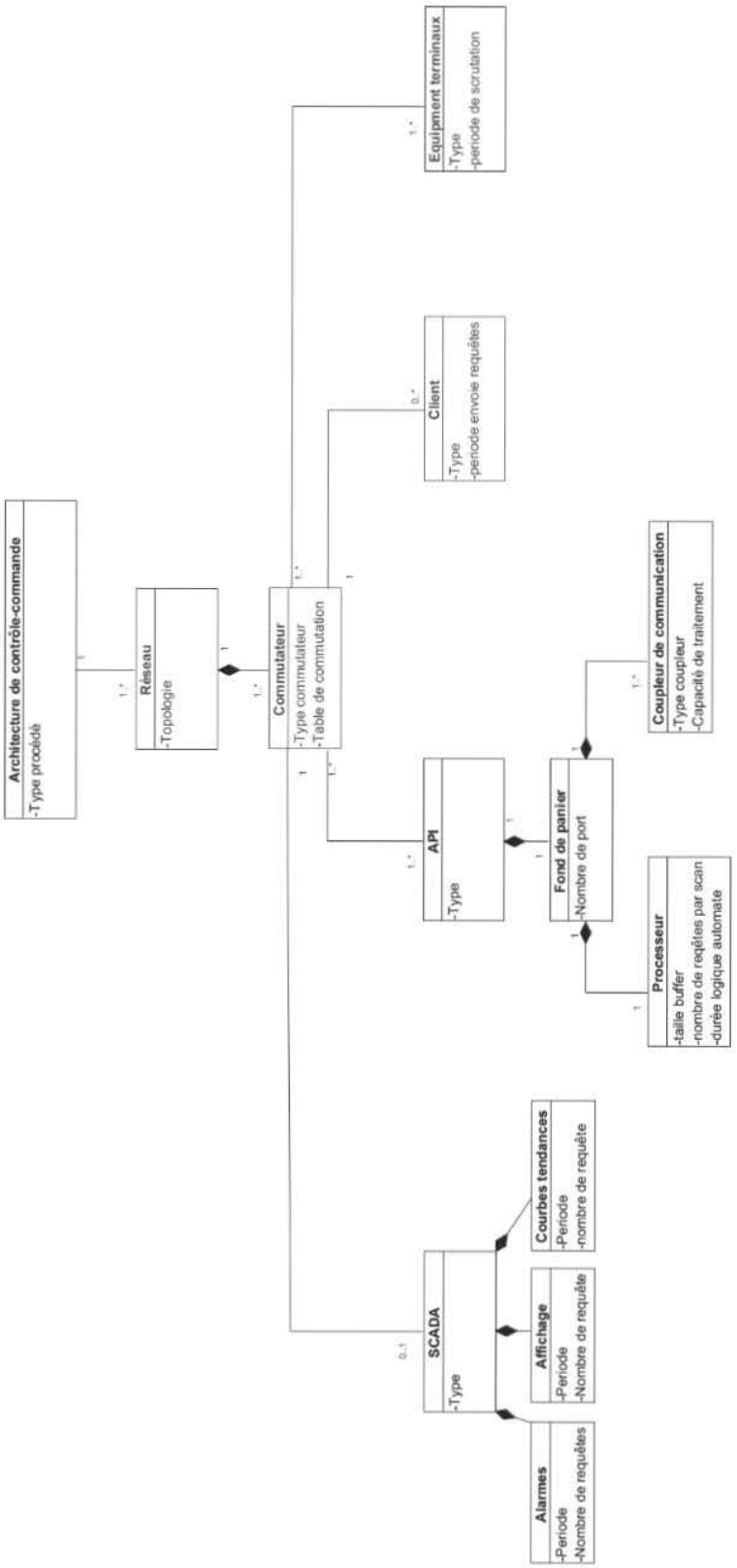


Figure 12: Spécification semi-formelle des composants d'une architecture de C/C

La classe des équipements terminaux possède également deux attributs qui sont le type et la période de scrutation. Les valeurs de ces attributs à l'instar de la classe « SCADA » et « client » sont renseignées par le système ingénieur en charge de l'élaboration de l'architecture.

Enfin, la classe « réseau » est une classe centrale possédant un lien d'association avec l'ensemble des classes de composants. Cette classe possède un attribut relatif à la topologie de l'architecture dont la valeur découle de la configuration de l'architecture. Dans le contexte industriel de cette étude, nous nous limiterons volontairement à un sous-composant unique représenté par la sous-classe « commutateur ». Cette sous-classe possède un attribut qui est la table de commutation. La valeur de celui-ci découle de l'attribut topologie.

5.2.1.2 Données relatives aux performances

Les données d'entrées représentant les performances temporelles de l'architecture, comme l'indique la Figure 13, se divisent en deux grandes sous-classes : sous-classe des performances temporelles liées au temps de réponse du système et sous-classe des performances du temps de cycle des équipements.

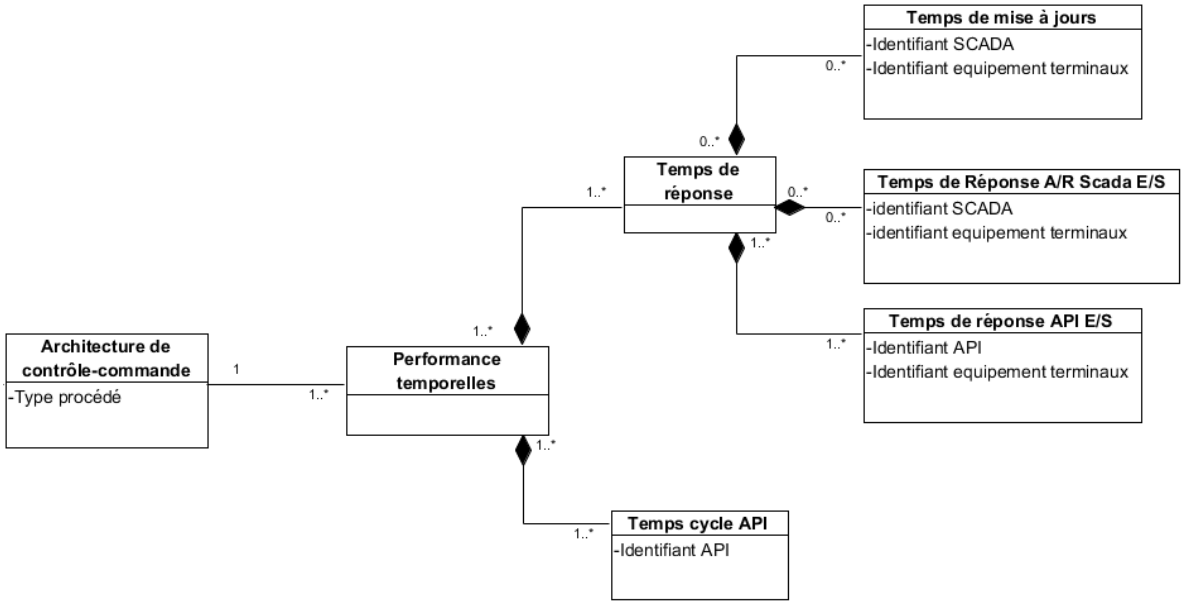


Figure 13 Spécification semi-formelle des performances à évaluer

La première sous-classe est composée de trois sous-classes représentant les trois grandes familles de performances temporelles liées aux temps de réponses et présentées en section 3.1.1. Chacune de ces sous-classes possède deux attributs qui sont l'identifiant de l'équipement émetteur de la requête et l'identifiant de l'équipement destinataire de la requête.

La deuxième sous-classe correspond aux performances liées au temps de cycle des automates et présentées en section 3.1.2. Elle est composée d'un attribut unique qui est l'identifiant du PLC.

L'ensemble des attributs de ces différentes classes et sous-classes est supposé connu et renseigné lors de l'élaboration d'une architecture.

De même, la description des performances à évaluer devra expliciter les types d'indicateurs à produire. Dans le cas des performances évaluées en phase d'avant-vente, les indicateurs à produire seront de type « valeur moyenne ». Dans ce cas, ils devront être accompagnés d'une indication relative au niveau de précision attendu. Celui-ci pourra revêtir différentes formes : intervalle de confiance, nombre de scénarios et durée dans le cas de tests ou de simulations.

5.2.2 Exigences sur les formalismes de modélisation

Compte tenu des besoins industriels exprimés, le formalisme utilisé devra répondre à différentes exigences permettant d'une part, de couvrir la structure et la hiérarchisation des architectures à évaluer, et, d'autre part, de modéliser le comportement dynamique des architectures dans un contexte incertain.

5.2.2.1 Modularité et hiérarchisation

Comme nous l'avons montré en section 2, les architectures de contrôle-commande sont la plupart du temps hiérarchiques. Chaque niveau regroupe des familles d'équipements définies en fonction des services offerts ou du type, de la taille et de la fréquence des informations échangées : par exemple, les SCADA pour le niveau 2, les PLC pour le niveau 1 ou les cartes d'entrées/sorties déportées ou cartes intelligentes pour le niveau 0. A ces équipements, il convient d'ajouter les familles d'équipements assurant la communication entre matériels de même niveau ou de niveaux différents.

Par ailleurs, les travaux dans le domaine de la modélisation de partie opérative ou d'équipement de contrôle-commande montrent qu'une certaine similitude se dégage entre les comportements des équipements d'une même famille. Dans ce contexte, il est tout à fait envisageable d'obtenir des modèles d'équipements génériques et paramétrables pour couvrir certaines différences comportementales mineures.

L'ensemble de ces éléments conduit à poser deux contraintes fortes relatives au formalisme servant de support à l'évaluation des performances des architectures de contrôle-commande :

- il devra supporter une modélisation hiérarchique permettant de structurer les modèles selon le niveau d'architecture
- il devra supporter une modélisation modulaire, réutilisable et paramétrable pour tirer parti des similitudes comportementales entre certains équipements composant l'architecture de contrôle-commande.

5.2.2.2 Pouvoir d'expression

Le formalisme étant destiné à supporter des mécanismes d'évaluation des performances temporelles, il devra bien entendu permettre la modélisation d'un comportement temporisé sous une forme continue ou discrète. Ce comportement temporisé permettra notamment de représenter les temps de traitements au sein des équipements et les délais de communication induits par les réseaux.

Par ailleurs, les principales contraintes portant sur le pouvoir d'expression du formalisme essentiellement sont relatives à la prise en compte du contexte incertain de la phase d'avant-vente qui peut revêtir plusieurs aspects. La connaissance extrêmement réduite de l'architecture fonctionnelle entraine en effet des incertitudes quant aux caractéristiques comportementales des équipements, notamment celles des PLC et des équipements de communication. Ces caractéristiques sont en effet sensibles à la charge que ces équipements auront à supporter, charge qui ne peut qu'être estimée en phase d'avant-vente compte tenu du manque d'informations sur les architectures fonctionnelles et donc sur les flux échangés entre fonctions

ou encore sur les types d'information à traiter (taille message, type de service etc..). A titre d'exemple, le temps de cycle d'un PLC peut dépendre des fonctions qu'il supporte, des incertitudes quant aux sollicitations que subira l'architecture de contrôle-commande. Ces incertitudes peuvent être :

- temporelles, dans la mesure où la fréquence des flux apériodiques (stimuli capteurs ou requêtes des opérateurs humains) n'est pas connue,
- spatiales, puisque l'absence d'information relatives à l'allocation des fonctions sur les matériels de l'architecture engendre une incertitude sur les matériels qui devront effectivement communiquer via le réseau.

Pour couvrir le premier point, le formalisme devra permettre la modélisation des incertitudes comportementales des équipements sous la forme d'intervalles de temps ou encore de distributions de probabilités uniformes (Seno, et al., 2009).

Pour couvrir le second point, le formalisme devra supporter la modélisation des incertitudes relatives aux flux apériodiques sous la forme de distributions de probabilités quelconques, potentiellement non exponentielles (Lois uniformes, Erlang, Weibull, …). En effet, les sollicitations humaines (actions de supervision des opérateurs humains) ou en provenance des capteurs (déclenchement d'un seuil, génération d'une alarme, occurrence d'une défaillance, …) génèrent des flux aléatoires de nature imprédictible (Klir, 2005) (Gertler, 1988) (Zwick & Walisten, 1989).

6 Conclusion

Ce chapitre a permis de fixer le contexte industriel de notre étude. L'objectif est de proposer un outil d'aide au dimensionnement des architectures de contrôle-commande en réseau dans le contexte de la phase d'avant-vente. Celui-ci induit deux grandes contraintes fondamentales, l'une relative au nombre important d'architectures qui devront être évaluées au gré des différentes offres élaborées, l'autre relative au contexte incertain de la phase-vente.

L'évaluation des performances d'architectures de contrôle-commande en réseau ayant fait l'objet de nombreux travaux, le chapitre suivant dresse un état de l'art des approches développées. L'analyse de leurs apports mais aussi de leurs limites, au regard des critères mentionnés en section 5.2.2 permettra d'une part, de définir la problématique scientifique de cette thèse, et, d'autre part, de procéder au choix d'un formalisme de modélisation.

Chapitre 2
Positionnement scientifique

1 Introduction

Ce chapitre a pour objectif de dresser un état de l'art des approches d'évaluation de performances qui pourraient répondre aux contraintes industrielles de notre étude comme explicité dans le précédent chapitre. L'analyse des apports et des limites de chaque approche nous conduira à sélectionner une approche de modélisation qui sera utilisée par la suite pour générer automatiquement, en avant-vente, les modèles et pour supporter l'évaluation de performances. Compte tenu des contraintes, une attention particulière sera apportée quant à la capacité de chaque approche à conduire efficacement une analyse d'un grand nombre d'architectures et de supporter le contexte incertain, aussi bien en termes de données d'entrée que de pertinence dans les résultats. Nous rappelons enfin que l'approche devra pouvoir modéliser l'architecture de bout en bout, aussi bien les composants terminaux (client, E/S déportées, API, etc.) que le réseau de communication (ici de type Ethernet commuté).

De manière macroscopique, l'évaluation de performances de tels systèmes peut s'apparenter à une modélisation en files d'attente. Chaque composant, chaque ressource (mémoire, processeur, bande passante), chaque fonction d'un composant va être associée à une file et l'étude de la concurrence d'accès à la ressource partagée va permettre d'estimer la dégradation du service engendrée par la traversée du composant. Afin de recouvrer une certaine forme d'exhaustivité, ce chapitre aborde les différentes solutions d'évaluation de performances de systèmes discrets selon une classification en trois grandes approches : modèles déterministes, modèles à états à paramètres déterministes et modèles stochastiques. Tout en ayant conscience que cette taxonomie est discutable et potentiellement arbitraire, le but sera d'établir s'il existe des propriétés structurelles propres à chaque approche voire des paramètres d'entrée qui ne seraient pas adaptés à notre cadre de modélisation d'avant-vente. Aussi, nous présenterons, pour chacun de ces modèles, un bref aperçu de leur fondement théorique ainsi qu'une discussion autour de leur utilisation dans le domaine de l'évaluation de performances temporelles (attendu qu'il s'agit du besoin attendu comme spécifié précédemment et même si d'autres propriétés pourront également être parallèlement estimées). Pour chaque approche candidate, une synthèse permettra de mettre en avant son adéquation à notre étude, et ces différentes synthèses seront discutées dans la dernière section en vue d'exprimer notre choix de formalisme de modélisation.

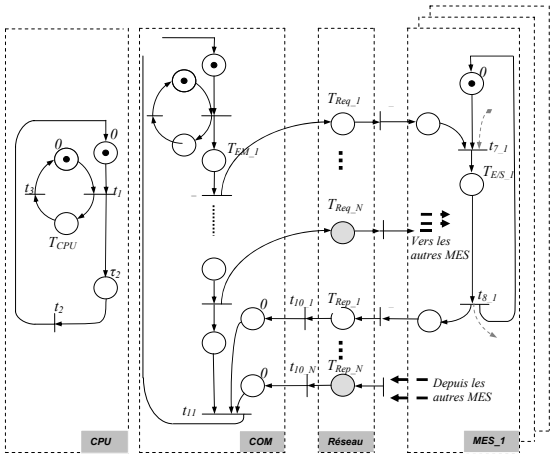
2 Evaluation de performances à l'aide d'approches algébriques

Dans la littérature, les modèles dits déterministes sont généralement construits suivant une géométrie tropicale, et plus spécifiquement suivant l'algèbre (max,+) (voire (min,+)). Ils s'expliquent par la volonté d'exprimer des performances déterministes (de type enveloppe), et pour cela visent à conduire une analyse elle-même déterministe. La modélisation du système consiste ici à l'expression d'un système d'équations dont la résolution permettra par la suite d'obtenir notamment des propriétés temporelles. Ces équations se singularisent par leur écriture où l'addition et la multiplication sont respectivement redéfinies par le maximum (ou le minimum) et l'addition. On parle ainsi de dioïde (max,+) ou (min,+). Cette technique a été développée par de nombreux auteurs, néanmoins nous focaliserons dans cette section sur son emploi pour des systèmes de contrôle/commande en réseau.

2.1 L'algèbre Max,+

L'algèbre (max,+) (Cohen, et al., 1996), (Davey & Priestley, 2002), (Hardouin, 2004), (Cohen, 1993) et (Lotito, et al., 2005) est utilisé pour la modélisation et l'évaluation de performance de systèmes à évènements discrets (réseaux de transport ou de télécom, systèmes de production). Elle peut également être utilisée pour déterminer les temps de marquage dans les réseaux de Petri. Elle a notamment été développée pour l'évaluation de performances avec les travaux de (Baccelli, et al., 1992) puis (Amari, et al., 2012). L'évaluation des performances passe par la résolution d'un système d'équations (max,+) linéaires. Ces équations peuvent être directement posées ou dérivées d'un autre formalisme (comme les réseaux de Petri) comme dans le cas suivant.

(Addad, 2011a) propose une modélisation générique d'un système contrôlé en réseau (SCR) à base d'équations (max,+). La modélisation s'attache à décrire l'ensemble des composants d'un tel système, à savoir les modules d'entrées/sorties déportées, d'un contrôleur (CPU et carte de communication), d'un MES et d'un réseau à base d'Ethernet commuté. Cette modélisation est accomplie ici à l'aide de réseaux de Graphes d'Évènements Temporisés (GET). Un graphe d'évènements temporisés (GET) est un réseau de Petri (voir section 3.2) ordinaire temporisé où chaque place n'admet qu'une transition d'entrée et une transition de sortie au maximum. C'est ensuite la dynamique de ce GET qui est traduite en un système d'équations (max,+) linéaires. On associe ainsi à chaque transition t_i une variable x(k), appelée dateur, qui représente la date de son franchissement pour la $k^{ème}$ fois. Le dateur associé à une transition d'entrée t_{uj} est noté $u_j(k)$. La Figure 14 décrit ainsi le système d'équations proposé par (Addad, et al., 2010) et (Addad, et al., 2011b) pour un système contrôlé en réseau ainsi les délais obtenus.



$$\begin{cases} \theta_1(k) = e \otimes \theta_2(k-1) \oplus e \otimes \theta_3(k-1) \\ \theta_2(k) = T_{CAL} \otimes \theta_1(k) \\ \theta_3(k) = T_{CPU} \otimes \theta_1(k) \end{cases}$$

$$\begin{cases} \theta_4(l) = (\theta_{11}(l-1) \otimes e) \oplus (\theta_{12}(l-1) \otimes e) \\ Pour\ i = 1, \cdots, N \\ \quad \theta_{5\_i}(l) = \theta_{5\_(i-1)}(l) \otimes T_{EM\_i} \quad // \theta_{5\_0}(l) = \theta_4(l)\ pour\ l'initialisation \\ \quad \theta_{6\_i}(l) = \theta_{5\_i}(l) \otimes T_{Req\_i} \\ \quad \theta_{7\_i}(l) = (\theta_{6\_i}(l) \otimes e) \oplus (\theta_{8\_i}(l-1) \otimes e) = \\ \quad \theta_{8\_i}(l) = \theta_{7\_i}(l) \otimes T_{E/S\_i} \\ \quad \theta_{9\_i}(l) = \theta_{8\_i}(l) \otimes e \\ \quad \theta_{10\_i}(l) = \theta_{9\_i}(l) \otimes T_{Rep\_i} \quad Fin \\ \theta_{11}(l) = (\theta_{5\_N}(l) \otimes e) \oplus \left[\bigoplus_{1 \le j \le N} (\theta_{10\_j}(l) \otimes e) \right] \\ \theta_{12}(l) = \theta_4(l) \otimes T_{COM} \end{cases}$$

Figure 14: Modèle d'un SCR mono client – multi serveur (Addad, et al., 2011b)

Les paramètres de ces équations étant tous connus, il est ainsi possible d'évaluer analytiquement des dates d'occurrences d'évènements remarquables dans le réseau. La résolution revient alors à traquer les messages et à partir des dates calculées, de développer une analyse déterministe pour le calcul de bornes (maximales et/ou minimales) du temps de réponse. (Ammour & Amari, 2015) développent plus en détail l'approche algébrique pour la résolution de ce système.

En synthèse, l'algèbre (max,+) s'avère intéressante à l'évaluation de performances d'architectures de contrôle/commande en réseau puisqu'elle permet d'obtenir de manière déterministe des bornes (minimales et maximales) sur les temps de réponse. Cette caractéristique de temps de réponse est d'autant plus intéressante puisqu'elle permet, à contrario d'une analyse des seuls délais de traversée du réseau, de prendre en compte à la fois la performance de la commande et du réseau de communication. Comme le montre (Addad, 2011a) cette algèbre peut également être couplée à une approche stochastique dans le but de déterminer la distribution (fonction de densité de probabilité) du temps de réponse. Les temps de franchissement sont exprimés par méthode combinatoire pour de petits réseaux ou par heuristique (algorithmes génétiques) pour gérer l'explosion combinatoire.

Néanmoins, il s'agit là de performances extrêmes qui nécessitent une connaissance sur le(s) scénario(s) qui n'est pas forcément disponible dans le contexte d'avant-vente. La complétude de la modélisation et la recherche des cas critiques restent également limitées. Enfin, le système d'équation de départ peut s'avérer difficile à exprimer (même en passant par une étape préliminaire de modélisation par graphe).

2.2 Évaluation de bornes déterministes par calcul réseau

2.2.1 Principe

La théorie du calcul réseau (ou *network calculus* dans la littérature anglophone) regroupe un ensemble de résultats obtenus dans le dioïde (min,+) permettant d'exprimer des bornes sur les délais et les arriérés de traitement (*backlog*). Cette théorie, formalisée par les travaux de (Le Boudec & Thiran, 2001), (Chang, 2000) et (Cruz, 1991) se présente comme une analyse déterministe des systèmes à files d'attente et a principalement été utilisée pour l'évaluation de performances dans les réseaux. L'obtention de bornes maximales concourt ainsi à démontrer la validité d'un système en réseau dans le pire des cas, que ce soit pour de grands réseaux ouverts comme l'Internet (spécification de la réservation de ressources *via* RSVP (Le Boudec & Thiran, 2001) ou encore pour des systèmes à contraintes temps-réel comme le contrôle-commande en avionique (qualification du réseau Ethernet embarqué dans un A380 (Grieu, 2004)

Les courbes d'arrivée et de service correspondent aux hypothèses d'entrée du calcul réseau. La courbe d'arrivée est une fonction croissante au sens large qui maximise à tout instant l'arrivée réelle du trafic (ou travail). Cette fonction peut s'exprimer à partir des limites capacitaires physiques et/ou logicielles (par exemple, (Grieu, 2004), (Charara, et al., 2006) et (Charara, 2007) s'appuient sur les réservations de liens virtuels sur un réseau AFDX) ou sur une connaissance de la génération du trafic (Georges, 2005) repose sur la connaissance de l'algorithme de contrôle/commande notamment pour des flux périodiques). La courbe de service est une fonction croissante au sens large qui minimise le service offert par le système étudié. La caractérisation de cette fonction pour l'étude d'un système réel n'est pas simple ; elle peut se faire par identification en approche boîte noire (Grieu, 2004) ou par composition de fonctions élémentaires Pour un système unitaire (une file d'attente, un flux), l'évaluation repose en calcul réseau sur la recherche pour le délai de traversée de la plus grande distance horizontale entre la courbe d'arrivée et la courbe de service. Pour l'arriéré, on majorera la

performance par la plus grande distance verticale entre ces deux mêmes courbes. Chacune des expressions est exprimée dans l'algèbre (min,+) et peut se faire en continu (Le Boudec & Thiran, 2001) ou en discret (Chang, 2000). On notera que par définition, un conservatisme sur l'identification des courbes entraîne un pessimisme des majorants obtenus par la suite.

Ces travaux ont ensuite été étendu pour le support de systèmes plus complexes. Pour l'analyse de la traversée d'un ensemble de files (correspondant par exemple à un chemin traversant plusieurs commutateurs), (Schmitt, et al., 2006b), (Le Boudec & Thiran, 2001) ont ainsi mis en évidence le principe du « *pay bursts only once* » qui consiste à calculer une courbe de service « du chemin » obtenue par convolution (min,+) des courbes de service de chaque file traversée. Pour la prise en compte de la concurrence de plusieurs flux quant à l'accès à une même ressource, il s'agit de prendre en compte l'ordonnancement opéré au niveau de la file. (Le Boudec & Thiran, 2001) et (Schmitt, et al., 2008) définissent ainsi une courbe de service propre à un flux donné et appelée service résiduel (ou *blind multiplexing*). Cette courbe revient pour une politique FIFO à supposer que le flux considéré est traité comme dans le cas d'un ordonnancement strict avec la priorité la plus basse. Cela peut sembler très conservateur, mais correspond en fait à la seule hypothèse possible attendu que la date d'arrivée des paquets est ici inconnue. Enfin pour des réseaux à multiples sauts, chemins et flux, (Lenzini, et al., 2008), (Bouillard, et al., 2008) introduisent la notion de « *pay multiplexing only once* » qui permet d'optimiser le recours à la composition de serveurs et au service résiduel.

2.2.2 Synthèse

L'obtention de bornes temporelles présente un intérêt indéniable pour l'étude des systèmes contrôlés en réseau. En vérifiant que ces bornes soient inférieures à l'exigence du contrôle/commande, il est ainsi possible de valider l'architecture. Le calcul réseau a ainsi permis de démontrer que les délais calculés (là où il n'est pas possible de reproduire expérimentalement l'ensemble des situations) resteraient inférieurs à ceux exigés pour l'avionique d'un A380 (Grieu, et al., 2003) et qualifier ainsi l'architecture de contrôle/commande. Il en va de même dans (Robert, 2012) concernant la qualification d'un lanceur spatial basé sur un réseau Ethernet commuté. Dans (Georges, 2005) le calcul réseau est employé pour déterminer à priori les limites de stabilité d'un système contrôlé en en réseau. Outre la performance temporelle, le calcul réseau peut également être utilisé pour dimensionner correctement une architecture pour qu'aucune information ne soit écartée du fait d'une saturation des mémoires, pour spécifier les priorités/ordonnancements entre les différents flux ou encore pour caractériser les possibilités d'évolutions capacitaires d'une architecture en termes de passage à l'échelle. Moult outils de calcul sont aujourd'hui disponibles : on peut ainsi citer RTAW-Pegase (Boyer, et al., 2010) ou Disco (Schmitt, et al., 2006a).

Toutefois, son utilisation nécessite une connaissance minimale sur la génération du trafic qui ne peut tolérer un contexte incertain comme celui de l'avant-vente. Les majorants obtenus peuvent également souffrir dans certains cas d'un conservatisme (pessimisme qui se réduit avec les derniers résultats disponibles dans la littérature) qui pourrait engendrer un « sur chiffrage » de la solution par un ingénieur en avant-vente. De plus, dans le cas de réseaux fortement cyclique et pour réduire le pessimisme, les derniers travaux font appel à des heuristiques (type

programmation linéaire) pour la résolution des différentes équations pour des ordonnancements de type FIFO, ce qui augmente de nouveau le temps d'analyse.

2.3 Évolution : approche par trajectoire

Plus récemment, une nouvelle approche par trajectoire a été développée. Il s'agit d'une méthode analytique qui permet de calculer un majorant de délai de bout-en-bout en identifiant les paquets d'autres flux que le paquet en question rencontre sur sa trajectoire lors de la traversée du réseau (Migge, 1999), (Martin & Minet, 2006a) et (Martin & Minet, 2006b). Cette approche a été notamment appliquée pour les réseaux Ethernet AFDX (Bauer, et al., 2009), (Bauer, et al., 2010) et (Li, et al., 2014) comme le montre la Figure 15.

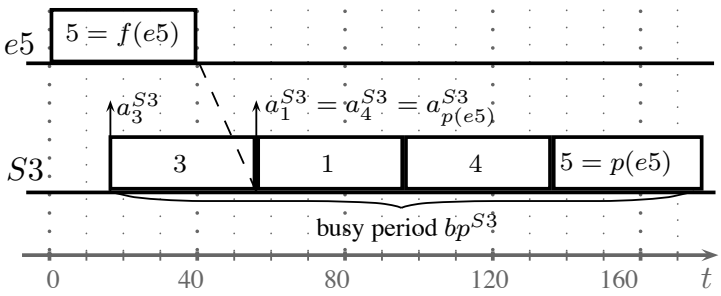


Figure 15: Exemple d'une trajectoire extraite de (Bauer, et al., 2009)

À l'instar du calcul réseau, l'approche par trajectoire est une méthode analytique qui permet de calculer un majorant de délai de bout en bout. Pour un flux donné, il s'agit d'identifier les paquets d'autres flux que le paquet en question rencontre sur sa trajectoire lors de la traversée du réseau. Elle se base sur le concept de période active qui représente le délai d'attente correspondant au traitement des paquets préalablement arrivés. Avec un calcul récursif et en recherchant les différentes trajectoires sur chaque composant depuis la destination vers la source, il est alors possible d'exprimer un majorant du délai de bout en bout.

Si cette approche permet de réduire le pessimisme du calcul réseau, elle présente les mêmes autres inconvénients dans le cadre d'une utilisation en avant-vente. La recherche poussée des différentes trajectoires peut même ici s'avérer plus compliquée. Enfin, on notera que toutes ces approches algébriques déterministes ne vont pas permettre de prendre en compte les différentes sources d'incertitudes, notamment sur la définition du trafic (inter arrivée, chemin, etc.).

3 Evaluation de performances à l'aide de modèles à états et à paramètres déterministes

Les modèles de type états-transitions (automates à états finis, réseaux de Petri, …) constituent une voie très largement explorée pour évaluer les performances d'un système. Lorsque l'on traite de l'évaluation de performances temporelles, ces modèles intègrent une représentation du temps, par exemple en associant une durée (temporisation ou intervalles de temps) aux états ou aux transitions du modèle. Dans cette section, nous nous focalisons sur les modèles dont les durées sont déterminées de manière déterministe. Dans ce contexte, l'évaluation repose sur une exploration totale de l'espace d'état à l'aide de techniques de vérification formelle ou bien sur une exploration partielle à l'aide de la simulation.

3.1 Vérification formelle des performances par model-checking

3.1.1 Principes et applications à la vérification de performances temporelles

Le model-checking est un ensemble de techniques de vérifications automatiques de propriétés temporelles, basées sur l'exploration de l'espace d'état d'un modèle de Système à Evénements Discrets. Cette théorie, développée principalement par (Witsch, et al., 2006) et (Clarke, et al., 1999) est principalement utilisée dans les domaines comme le développement logiciel (Havelund & Thomas, 2000) ou dans les systèmes à temps réel, comme les architectures de contrôle-commande (Bozga, et al., 1998) (Rossi, 2003) pour démontrer la conformité d'un programme vis à vis de ses spécifications.

Le model-checking temporisé est une famille de techniques de model-checking qui repose sur l'exploration de l'espace d'état de modèles SED temporisés. Plus précisément, un algorithme de model-checking prend en entrée un modèle du comportement du système à vérifier et une formule en logique temporelle qui décrit la propriété que l'on souhaite prouver. Si le modèle satisfait la formule temporelle, la vérification est un succès, dans le cas contraire, un contre-exemple est retourné.

Le model-checking temporisé est donc parfaitement adapté à la vérification formelle du respect de performances temporelles. Le principe consiste à modéliser le comportement d'un système à l'aide de modèles SED temporisés, des automates à états finis temporisés dans la plupart des applications et de décrire la performance à atteindre soit :

- sous la forme d'une propriété exprimée en logique temporelle que l'on cherchera à vérifier,
- sous la forme d'un état « défaut » intégré dans le modèle du système qui traduit le fait qu'une performance de temps n'est pas satisfaite, la vérification se limitant, dans ce cas, à prouver que les états « défaut » ne sont pas atteignables depuis l'état initial.

La première méthode a été appliquée sur les systèmes de contrôle commande par (Ben Hedia, et al., 2005) distribués où les paramètres temporels de synchronisations des procédés ont été modélisés (Figure 16). Nous pouvons citer une autre application de cette approche par (Krakora & Zdenek, 2004) pour vérifier les propriétés temporelles et logiques d'un système de contrôle-commande dont le protocole réseau est de type CAN (Tindell, et al., 1994). Dans cette étude, les auteurs considèrent que les systèmes de contrôle-commande possèdent deux types de ressources partagées qui sont les processeurs des automates et le bus de communication. Le traitement réalisé par les processeurs est complétement désynchronisé avec l'accès au bus de communication pour la transmission de données. Dès lors, un arbitrage est réalisé afin de déterminer le processeur pouvant accéder au bus de communication. Ainsi les temps impactant les performances du système se situe au niveau de l'attente pour l'accès au bus de communication, et aussi sur le temps de traitement des processeurs. Pour cela les auteurs ont réalisé deux modèles représentés sur la Figure 17 afin de vérifier les propriétés du système et ainsi avoir les performances de celui-ci.

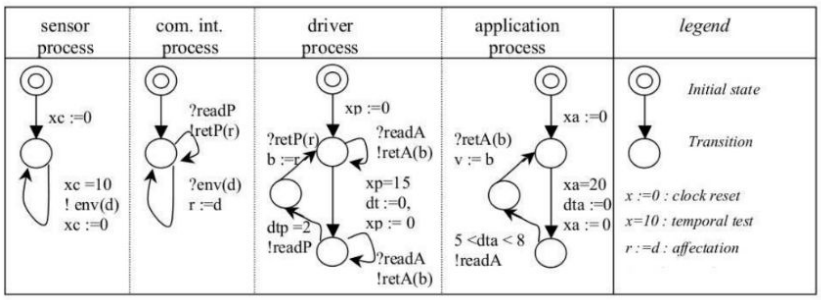


Figure 16: Modèle d'un système de contrôle et d'acquisition (Ben Hedia, et al., 2005)

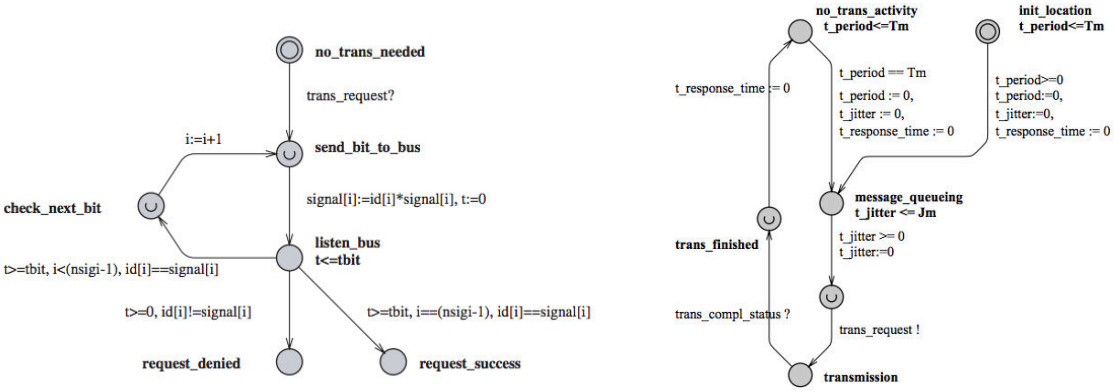


Figure 17: Modèle d'arbitrage (gauche), traitement (droite) par (Krakora & Zdenek, 2004).

La seconde approche a été mise en œuvre par (Bordbar & Anane, 2005) pour la gestion automatique de la qualité de service pour des réseaux sans fil. Un état « défaut » est atteint si le débit du réseau, ainsi que la latence et la gigue que subissent les trames sortent d'une plage de valeur prédéfinie. De manière similaire, (Rodriguez-Navas, et al., 2006) traite du problème de la synchronisation d'horloge entre maître et esclave dans un réseau CAN en se ramenant à un problème de recherche d'atteignabilité d'un état « défaut » d'un automate observateur (Figure 18). Enfin, cette technique a également été utilisée par (Limal, 2009) et (Mekki, 2012) pour la validation d'architectures de contrôle-commande redondantes à base d'Ethernet industriel.

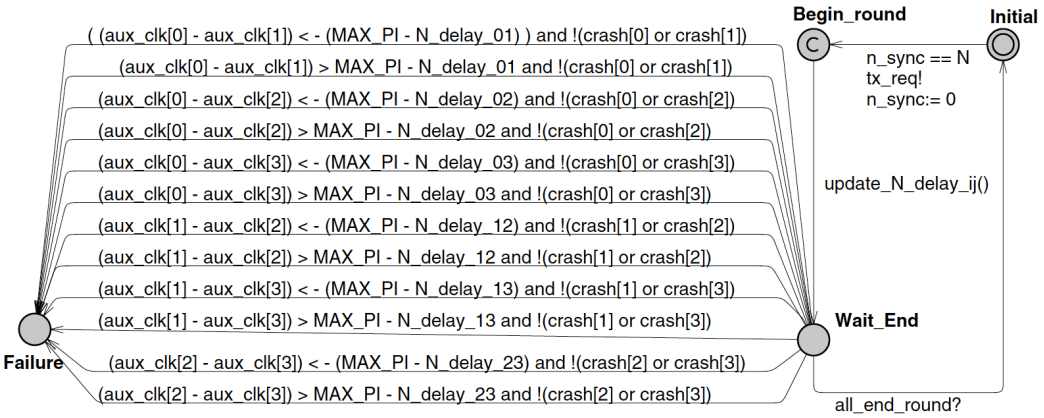


Figure 18: : Automate observateur (Rodriguez-Navas, et al., 2006)

Dans toutes ces approches, la performance temporelle à vérifier est supposée connue pour être intégrée soit dans une propriété soit dans le modèle de système sous la forme d'un état « défaut ». Les travaux développés au LURPA par (Ruel, 2009) étendent ces approches à l'évaluation de performances (par nature inconnues) en appliquant une démarche itérative qui consiste à partir d'une performance très surévaluée mais vérifiée sur le model-checker (lorsque l'on mesure un temps de réponse par exemple) puis à réduire progressivement celle-ci jusqu'à obtention d'un échec de preuve (Figure 19). Dans ce contexte, la dernière valeur de la performance qui aura pu être vérifiée constitue une borne supérieure de la performance que l'on cherche à évaluer.

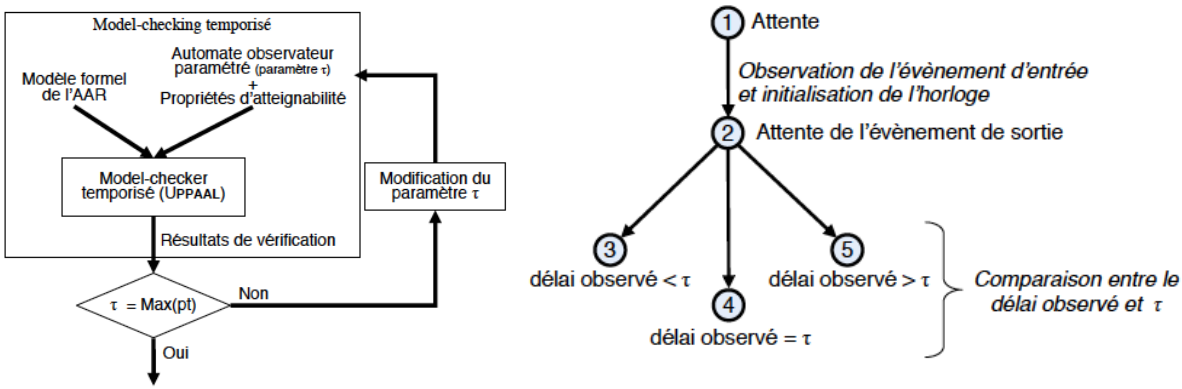


Figure 19: Evaluation de performances par vérification itérative de propriétés (Ruel, 2009)

3.1.2 Synthèse

Même si ces approches ont donné d'excellents résultats, elles restent limitées par la taille des systèmes que l'on peut évaluer. En effet, elles souffrent généralement d'un problème classique d'explosion combinatoire qui peut conduire à un échec du processus de vérification lorsque le nombre d'états à explorer est trop important. Malgré le fait que le model-checking fournit des résultats précis, celui-ci souffre de l'explosion combinatoire liée à la taille des modèles. Ainsi pour les systèmes d'automatisation où l'architecture peut être complexe avec une nomenclature très importante une explosion combinatoire est fortement envisageable. Malgré les travaux visant à réduire ce phénomène, comme par exemple par des techniques d'abstraction ou de raisonnement compositionnel (Clarke, et al., 2001), il reste néanmoins un frein pour l'évaluation de systèmes complexes de grande taille.

Enfin, ces techniques ne sont pas adaptées au contexte particulier de l'avant-vente où il rester très difficile de caractériser le comportement de l'architecture de contrôle-commande de manière déterministe avec des durées ou des plages de durée fixes. Des extensions, basées sur le model-checking probabiliste (outil PRISM par exemple (Kwiatkowska, et al., 2011) permettent d'appréhender la vérification de propriétés de systèmes non-déterministes (Greifeneder & Frey, 2007) mais reposent souvent sur l'introduction de probabilités discrètes dans la modélisation du système ou sur les propriétés à vérifier. Elles ne couvrent donc que très partiellement le problème de la modélisation des incertitudes présentes en avant-vente, notamment sur les durées de traitement ou de tâches, sur les flux de communications ou encore sur les sollicitations de l'environnement.

3.2 Evaluation de performances à l'aide de réseaux de Petri

L'utilisation des réseaux de Petri (RdP) et de plusieurs de ses extensions (temporisé, temporel, coloré, hiérarchique, …) dans le domaine de l'évaluation de performance a fait l'objet de nombreux travaux. Dans cette section, nous considérons les extensions :

- non autonomes (RdP temporisé, RdP temporel) permettant d'introduire la modélisation du temps dans les modèles de RdP, indispensable pour procéder à des évaluations de performances temporelles,
- structurelles (RdP coloré, RdP hiérarchique) qui contribuent à faciliter la modélisation de systèmes complexes ou de grande taille à l'aide de représentations plus compactes ou modulaires.

Nous soulignerons l'intérêt de ces différentes extensions dans le cadre de notre étude au travers d'exemples d'application à l'évaluation de performances, tout en mettant en évidence, en fin de section, quelques limites pour prendre en compte le contexte de l'avant-vente.

3.2.1 Les RdP et quelques une de leurs extensions

Un Réseau de Petri généralisé (RdPG) (Murata, 1989) est un graphe orienté biparti, alternant deux types de nœuds : les places et les transitions. Chaque place contient un nombre entier (positif ou nul) de marques (ou jetons). Le marquage d'un réseau est défini par le vecteur des marquages de ses places. Les arcs reliant une place P_i à une transition T_j (ou une transition à une place) sont associés à un poids représenté par un entier N_{PiTi} (ou N_{TjPi}) qui, par défaut, est égal à un. Une transition T_j est franchissable si pour chaque place d'entrée P_i de cette transition, son marquage est supérieur ou égal au poids de l'arc les reliant. Le franchissement de la transition consiste à retirer, dans chaque places d'entrée, un nombre de jetons égal au poids des arcs reliant les places d'entrée à la transition et à déposer, dans chaque place de sortie, un nombre de jetons égal au poids des arcs reliant la transition aux places de sortie.

L'analyse d'un réseau de Petri peut s'effectuer de manière analytique à l'aide de sa formalisation dans une algèbre linéaire ou bien à l'aide du graphe de marquage. L'analyse de ce graphe, dont les sommets sont les vecteurs de marquage et les arcs sont les transitions, permet d'exhiber des propriétés classiques telles que la bornitude, la vivacité, la réinitialisabilité ou encore le blocage. De plus, la détermination de P semi-flots ou de T semi-flots permet d'identifier respectivement des invariants de marquage ou de transitions. Si ces propriétés permettent, d'une certaine manière, d'évaluer qualitativement les performances d'un système modélisé sous la forme d'un RdP, elles demeurent insuffisantes pour procéder à une analyse quantitative des performances temporelles. Celles-ci imposent de prendre en compte le temps dans les modèles, ce qui fera l'objet des extensions temporisées et temporelles des RdP.

3.2.1.1 Le temps dans les modèles RdP

Les **réseaux de Petri temporisés** (Timed Petri Nets en anglais), introduits par (Ramchandani, 1973) associent soit une durée de séjour aux places pour les réseaux P-temporisés, soit une durée de franchissement aux transitions pour les réseaux T-temporisés (Wang, 1998).

Un RdP P-temporisé est un doublet {R, τ} tel que R est un RdP marqué et τ est une application $P \rightarrow \mathbb{Q}^+/\tau(P_i) = d_i$ où $\mathbb{Q}^+$ est l'ensemble des rationnels positifs ou nuls et d_i la temporisation appliquée à la place P_i. Un RdP T-temporisé est un doublet {R, τ} tel que R est un RdP marqué et τ est une application $T \rightarrow \mathbb{Q}^+/\tau(T_i) = d_i$ où $\mathbb{Q}^+$ est l'ensemble des rationnels positifs ou nuls et d_i la temporisation appliquée à la transition T_i.

Dans un RdP P-temporisé, lorsqu'une marque est déposée dans une place, elle reste indisponible dans celle-ci pour la durée associée à la place (i.e. durant ce laps de temps, la marque ne pourra servir à sensibiliser une transition située en aval de la place). Dans les RdP T-temporisé, le franchissement d'une transition conduit à réserver le nombre de marques nécessaires dans les places situées en amont pendant la durée associée à la transition (i.e. les jetons sont maintenus dans les places amont mais ne peuvent plus être utilisés pour d'autres tirs) ; lorsque la durée est écoulée, les jetons déposés dans les places en aval sont immédiatement disponibles.

Les durées associées aux places et aux transitions définissent donc des valeurs minimales de temporisation. Ils constituent donc un formalisme parfaitement adapté pour l'évaluation de performances des processus comportant des opérations dont l'exécution nécessite un certain temps. L'analyse de ces réseaux, qu'elle soit basée sur le graphe de marquage ou bien effectuée à l'aide de simulations, fait très souvent l'hypothèse qu'ils fonctionnent à vitesse maximale, c'est à dire qu'une transition est franchie dès qu'elle devient franchissable. Dans ce cas, et sous réserve qu'ils soient bornés, ces réseaux atteignent un régime stationnaire caractérisé par un cycle sur le graphe de marquage qui permet assez aisément de déterminer des fréquences de franchissement de transitions, représentatives des performances temporelles (Figure 20).

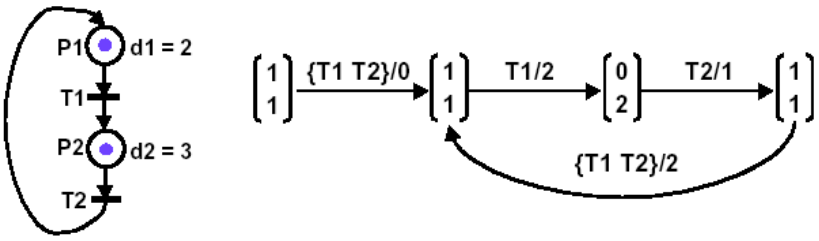
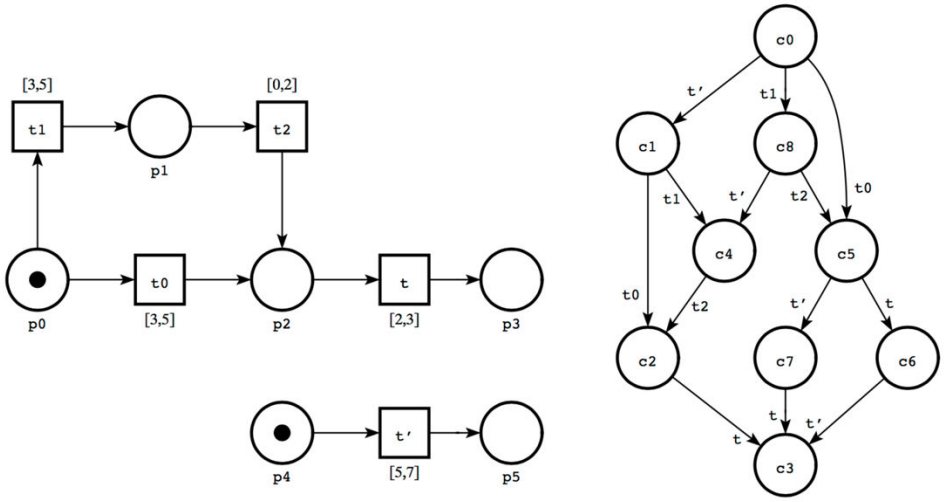


Figure 20: RdP P-temporisé et son graphe de marquage pour M_0 = (1,1)

Les **réseaux de Petri temporels** (Time Petri Nets en anglais) (Merlin, 1974) sont obtenus depuis les réseaux de Petri en associant deux dates *min* et *max* à chaque transition (pour les RdP T-temporels, il existe également des RdP P-temporels (Khansa, 1997). Supposons que la transition T soit devenue sensibilisée pour la dernière fois à la date *d*, alors elle ne peut être tirée avant la date *d – min* et doit l'être au plus tard à la date *d + max*, sauf si le tir d'une autre transition a désensibilisé T avant que celle-ci ne soit tirée. Le tir des transitions est de durée nulle.

Par analogie au graphe de marquage des RdP généralisés ou temporisés, il est possible de définir un graphe de classes d'états caractérisées par un marquage des places et des domaines temporels associés aux tirs des transitions (Figure 21).



classe marquage domaine de tir	$c0$ $p0,\ p4$ $5 \leq t' \leq 7$ $3 \leq t0 \leq 5$ $3 \leq t1 \leq 5$	$c1$ $p0,\ p5$ $0 \leq t0 \leq 0$ $0 \leq t1 \leq 0$	$c2$ $p2,\ p5$ $2 \leq t \leq 3$	$c3$ $p3,\ p5$	$c4$ $p1,\ p5$ $0 \leq t2 \leq 2$
classe marquage domaine de tir	$c5$ $p2,\ p4$ $2 \leq t \leq 3$ $0 \leq t' \leq 4$	$c6$ $p3\ , p4$ $0 \leq t' \leq 2$	$c7$ $p2\ , p5$ $0 \leq t \leq 3$	$c8$ $p1\ , p4$ $0 \leq t' \leq 4$ $0 \leq t2 \leq 2$	

Figure 21: RdP T-temporel et son graphe de classes d'états (Berthomieu & Vernadat, 2006)

Ces réseaux sont donc particulièrement adaptés pour la modélisation de temps de séjour exprimés sous la forme d'intervalles, comme par exemple pour des processus caractérisés par des dates de début au plus tôt et fin au plus tard d'opérations. Même si leur cible d'application d'origine concerne les systèmes de communication munis de *time-out* (Berthomieu & Diaz, 1991), les RdP temporels peuvent être utilisés dans des domaines très larges pour l'évaluation de performance ou l'ordonnancement de tâches. Notons également que, même si d'autres formalismes sont plus adaptés à la modélisation des incertitudes comme nous le verrons par la suite, la modélisation du temps sous la forme d'intervalles dans les RdP t-temporels peut traduire une incertitude sur la durée.

3.2.1.2 Coloration et hiérarchisation des RdP

En compléments des extensions de RdP ayant pour objectif la modélisation du temps nécessaire pour l'évaluation des performances temporelles, deux extensions doivent être mentionnées dans la mesure où elles contribuent à la maîtrise de modèles de grande taille caractérisant nos applications : les RdP colorés et les RdP hiérarchique.

Les **réseaux de Petri colorés**, introduits par (Jensen, 1981), sont une classe de RdP dans laquelle les jetons sont porteurs d'une information, appelée couleur. Cette couleur appartient à un ensemble de couleurs d'un type simple (booléen, entier, réel, chaine de caractères) ou complexe construit comme le produit cartésien d'ensembles de couleurs élémentaires. Chaque

place est associée à un ensemble de couleurs (simple ou complexe) et ne peut contenir que des jetons dont les couleurs appartiennent à cet ensemble. De la même manière, chaque transition est associée à un ensemble de couleurs, chacune de ces couleurs indiquant une possibilité distincte de franchissement. Des fonctions d'incidence avant et arrière sont associés aux arcs respectivement entrant et sortant des transitions et permettent de :

- définir les conditions de franchissement en termes de nombre de jetons colorés devant être présents dans les places d'entrée de la transition (incidence avant ou PRE),
- définir le nombre et les couleurs des jetons déposés dans les places en sortie de la transition (incidence arrière ou POST).

Soit $C(T_j)$ l'ensemble des couleurs associées à la transition T_j. Cette transition peut être franchie par une quelconque de ces couleurs. Soit C_k une couleur de $C(T_j)$ et M un marquage courant du RdP coloré. La transition T_j est **validée** par rapport à C_k pour le marquage M si et seulement si : $\forall P_i \in {}^\circ T_i, \quad M(P_i) \geq PRE(P_i, T_j/C_k)$. Sur la Figure 22, la transition T1 sera validée pour la couleur $\langle b \rangle$ si $M(P_1) \geq PRE(P_1, T_1/\langle b \rangle) = f(\langle b \rangle) = \langle c \rangle$. Une transition T_j validée pour une couleur C_k peut être franchie. Son franchissement, noté T_j / C_k consiste à retirer à toute place P_i en amont de T_j une quantité de marque égale à Pré(P_i, T_j/C_k) et à déposer dans toute place P_i en aval de T_j une quantité de marque égale à Post(P_i, T_j/C_k). Sur la Figure 22, le franchissement de la transition T1 par rapport à la couleur $\langle b \rangle$ conduira à retirer de la place P1 un jeton de couleur $\langle c \rangle$ et à déposer dans la place P2 un jeton de couleur $\langle b \rangle$ et un jeton de couleur $\langle a \rangle$.

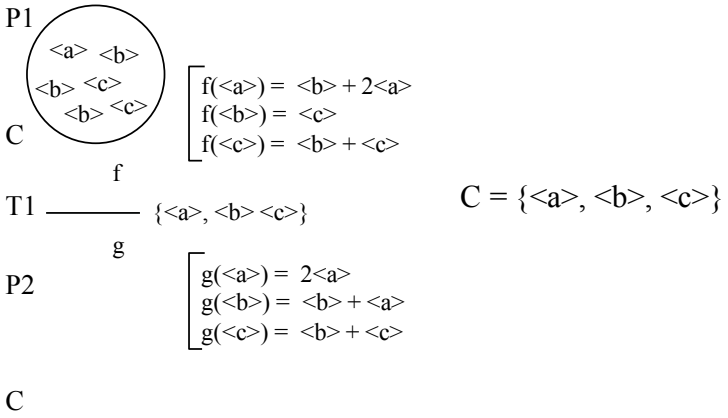


Figure 22: Exemple de RdP coloré

Le principal intérêt de la coloration réside dans les opérations de pliage (et inversement de dépliage) définissant une équivalence entre un RdP généralisé et un RdP coloré. En effet, un choix judicieux de couleur doit permettre la représentation de plusieurs composants, décrits chacun par un RdP identique, sous la forme d'un RdP coloré unique dans lequel les couleurs identifieront les composants comme le montre l'exemple très simple de la Figure 23. Cette caractéristique des RdP colorés présente un intérêt très significatif pour la modélisation de systèmes impliquant plusieurs composants dont le comportement est similaire, ce qui est évidemment le cas des architectures de contrôle-commande dont nous cherchons à évaluer les performances.

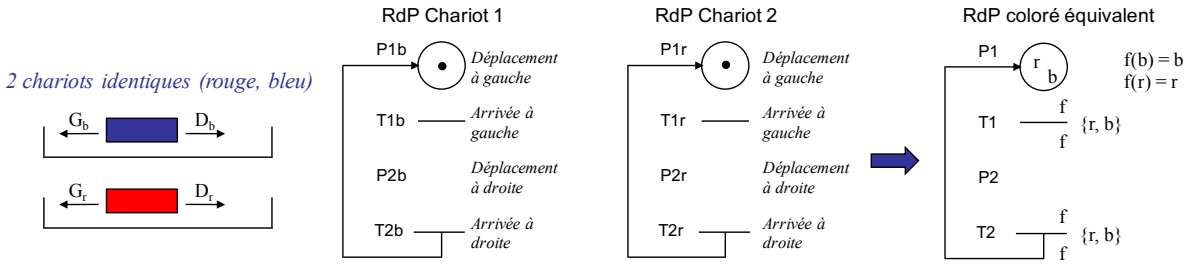


Figure 23: Exemple de pliage d'un RdPG vers un RdP coloré

Les **réseaux de Petri hiérarchiques**, introduits par (Jensen, 1994), propose une structuration modulaire des RdP dans laquelle des éléments du réseau, appelés transition de substitution, sont eux-mêmes composés d'un réseau de Petri. Le modèle RdP associé à une transition de substitution est appelé module RdP hiérarchique ; il peut contenir, à son tour, d'autres transitions de substitution.

Un module RdP hiérarchique est un 9-uplet N = $(P, T, E, S, M_0\, T_{sub,}\, P_{port,}\, PT)$ où :

- (P, T, E, S, M_0) définissent un RdP
- T_{sub} est un ensemble de transitions de substitutions appartenant à l'ensemble fini des transitions T d'un RdP. $T_{sub} \subseteq T$.
- P_{port} définit une place représentant l'ensemble des ports de connections d'un module. Celles-ci appartenant à l'ensemble fini des places P. $P_{port} \subseteq P$.
- PT défini un attribut caractérisant le type de port de la place $P_{port.}$ Cet attribut peut être de type entrée (IN), sortie (OUT), ou d'entrée et de sortie (I/O). tel que PT : $P_{port} \rightarrow$ {IN,OUT,I/O}.

Un RdP hiérarchique est un n-uplet N = $(P, T, C, E, S, M_0, T_{sub,}\, P_{port,}\, PT, S, SM, PS, FS)$ où :

- (P, T, C, E, S, M_0) définissent un RdP coloré
- $(T_{sub,}\, P_{port,}\, PT)$ définissent un module RdP hiérarchique.
- S définit un ensemble non vide de modules RdP hiérarchique
- SM : $T_{sub} \rightarrow S$ est la fonction qui assigne à chaque transition de substitution T_{sub} un ensemble de module S.
- PS définit la fonction d'assignation des places P_{port} à une transition de substitution $T_{sub.}$
- FS est un ensemble non vide de P où tous les éléments on la même couleur et les mêmes poids (nombre de jeton).

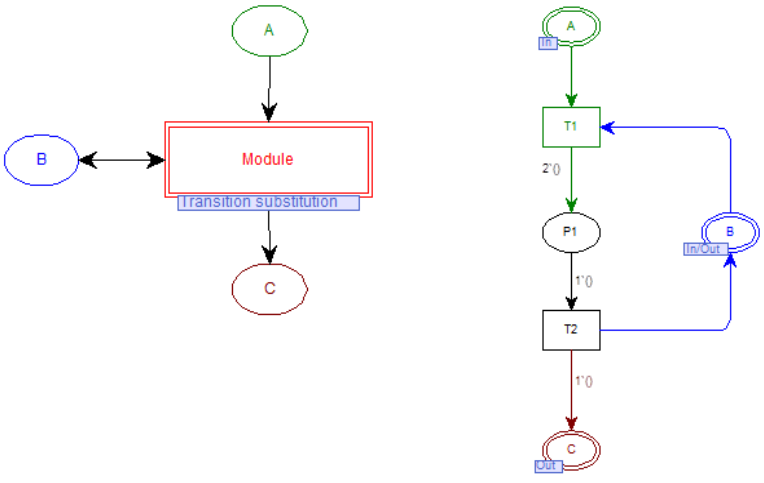


Figure 24: RdP hiérarchique (Gauche) et module hiérarchique (Droite)

La Figure 24 représente un RdP hiérarchique comportant 3 places A, B et C ainsi qu'une transition de substitution *Module*, associée réseau de droite sur la figure dont les places A, B et C constituent respectivement les ports d'entrée(IN), d'entrées/sorties (I/O) et de sortie (OUT).

L'intérêt de cette extension est de permettre une description hiérarchique des systèmes mais aussi modulaire dans la mesure où deux transitions de substitution du réseau hiérarchique peuvent être associées à deux instances d'un même modèle RdP de module hiérarchique. Elle offre donc à la fois une visualisation simplifiée par niveau d'un modèle RdP complexe mais aussi la possibilité de mettre en œuvre une modélisation, sinon orientée objet au sens strict du terme, tout au moins orientée « composants » au sens des approches Component-Based Automation prônées par la norme IEC 61499 (Maffezzoni, et al., 1999), (Thramboulidis, 2004), (Sünder, et al., 2006), (Estévez, et al., 2007).

Les extensions colorées et hiérarchiques sont donc tout à fait pertinentes pour la modélisation d'architectures de contrôle-commande caractérisées par une structuration hiérarchique et par la présence de composants dont les comportements sont similaires.

3.2.1.3 Les réseaux de Petri colorés, hiérarchiques et P-temporisés

Le formalisme proposé par (Jensen, 1994) regroupe trois des extensions présentées précédemment – RdP P-temporisé, RdP coloré et RdP hiérarchique – sous l'abréviation CPN (pour Coloured Petri Nets) quelque peu trompeuse car réduite à la seule extension colorée.

Ce formalisme a servi de base au développement de l'outil CPN Tools[1] (Jensen, et al., 2007) qui complète les fondements théoriques de ces trois extensions par :

- le couplage des réseaux CPN à un langage de modélisation fonctionnelle (ML) qui permet une manipulation plus aisée des trois extensions précédentes et la définition de structures complexes telles que les listes par exemple,
- l'introduction de variables aléatoires pour caractériser la durée associée aux temporisations des places.

Ces deux notions complémentaires contribuent, de manière indiscutable, à accroître le pouvoir d'expression des réseaux CPN de Jensen mais peuvent réduire, en contrepartie, les capacités de raisonnement formel associés aux RdP et ses extensions temporisés, colorés et hiérarchiques. En effet, la manipulation au travers du langage ML de structures de données complexes telles que les listes ou l'introduction de variables aléatoires définies par une distribution de probabilité quelconque peuvent rendre difficile, voire impossible dans certains cas, la construction et l'analyse formelle d'un graphe de marquage par exemple. Le recours à la simulation de Monte-Carlo est donc d'un usage très fréquent dans les applications développées à l'aide de CPN Tools, notamment pour l'évaluation de performances.

Nous reviendrons, en particulier, dans la section 4.3, sur la présence de variables aléatoires dans les temporisations qui ne permet cependant pas de considérer les réseaux CPN comme un modèle stochastique (c'est à dire caractérisant un processus stochastique), sauf sous certaines hypothèses que nous détaillerons ultérieurement.

[1] CPN Tools a été développé par le CPN Group à l'Université de Aarhus entre 2000 et 2010. Ses principaux contributeurs sont K. Jensen, S. Christensen, L.M. Kristensen et M. Westergaard. Depuis 2010, CPN Tools est maintenu par le AIS group à l'Université de Technologie. Site web : *cpntools.org*

3.2.2 Evaluation de performances à l'aide des RdP

L'évaluation de performances à l'aide des réseaux de Petri et de ses extensions a fait l'objet de nombreux travaux qui font apparaître deux grandes familles d'approches :

- la vérification formelle d'une propriété, par exemple exprimée dans une logique temporelle, qui représente le respect d'une performance (ou inversement sa violation),
- l'analyse du réseau temporisé au travers de trois indicateurs : les fréquences de franchissement de transitions, les marquages des places ou encore les temps de séjour dans les places. Ces trois indicateurs peuvent être mesurés par :
    - des approches formelles basées sur une analyse des graphes d'accessibilité par exemple,
    - les approches basées sur la simulation de Monte-Carlo pour évaluer, de manière statistique, les performances moyennes, minimum et maximum ainsi que les indicateurs de confiance associés (écart type, intervalle de confiance, …).

La vérification formelle repose sur des techniques analogues à celles mise en œuvre pour le model-checking temporisé. Une performance temporelle est traduite sous la forme d'une formule en logique temporelle, par exemple CTL ou LTL, qui exprime soit une propriété d'accessibilité (on cherche dans ce cas à vérifier que la performance est atteignable), soit une propriété de sûreté (on cherche dans ce cas à vérifier qu'une performance non acceptable ne peut jamais être constatée). Dans cette catégorie d'approches, nous pouvons citer notamment les travaux développés autour des réseaux de Petri temporels et de l'outil TINA (Berthomieu, et al., 2004) . Cette approche permet donc de vérifier qu'une performance donnée est respectée (ou non) ; en d'autres termes, elle est inefficace pour mesurer une performance supposée non connue, excepté à l'aide d'une approche par vérification itérative de propriété qui permet par ajustements successifs de s'approcher de la performance (Ruel, 2009).

L'utilisation la plus courante des RdP temporisés pour l'évaluation de performances repose sur la détermination des indicateurs : marquage, fréquence de franchissement et temps de séjour ainsi que toute autre combinaison de ces trois indicateurs de base. Ces indicateurs (marquage moyen, temps de séjour, temps de traversée du réseau d'une place à une autre, fréquences de franchissement, …) peuvent être évalués de manière formelle, en exploitant le comportement cyclique du graphe de marquage en régime stationnaire, des systèmes d'inéquations extraites des P et T semi-flots ou encore à l'aide de calculs symboliques (Sifakis, 1980).

Cependant, face à la complexité et la taille des modèles, l'approche par simulation est très largement privilégiée dans la littérature, et notamment la simulation de Monte-Carlo. Cette dernière a pour essence l'utilisation d'expériences répétées pour évaluer une quantité : la répétition des expériences permet, en autres, de s'affranchir du déterminisme intrinsèque à l'algorithme de simulation (par exemple dans le cas de transitions simultanément franchissables ou en conflits) mais aussi d'intégrer les aspects aléatoires pouvant être présents dans les modèles (par exemple, une durée de temporisation aléatoire dans les réseaux CPN définis par Jensen).

A titre d'illustration, nous pouvons citer les travaux de (De Figueiredo & Kristensen, 1999) pour l'évaluation de performances des protocoles TCP à l'aide de RdP colorés, hiérarchiques et temporisés ou encore les travaux de pour l'évaluation des performances de service web à l'aide des RdP temporels (Abdelli, et al., 2015). Ces approches utilisent une représentation du temps déterministe, sous la forme de temporisation ou d'intervalles. D'autres approches, basées sur les réseaux CPN définis par Jensen, exploite la modélisation du temps sous forme de variables aléatoires pour représenter les sollicitations (par exemple, l'arrivée des requêtes décrite par un processus de Poisson de taux λ). Toujours à titre d'illustration et dans le domaine de l'évaluation de performances d'architectures de contrôle-commande, nous pouvons citer :

- des approches centrées sur la modélisation des dispositifs de commande, tels que des automates programmables pour permettre l'évaluation de temps de réponse (Meunier, 2006), (Marsal, 2006a), (Marsal, et al., 2006b),

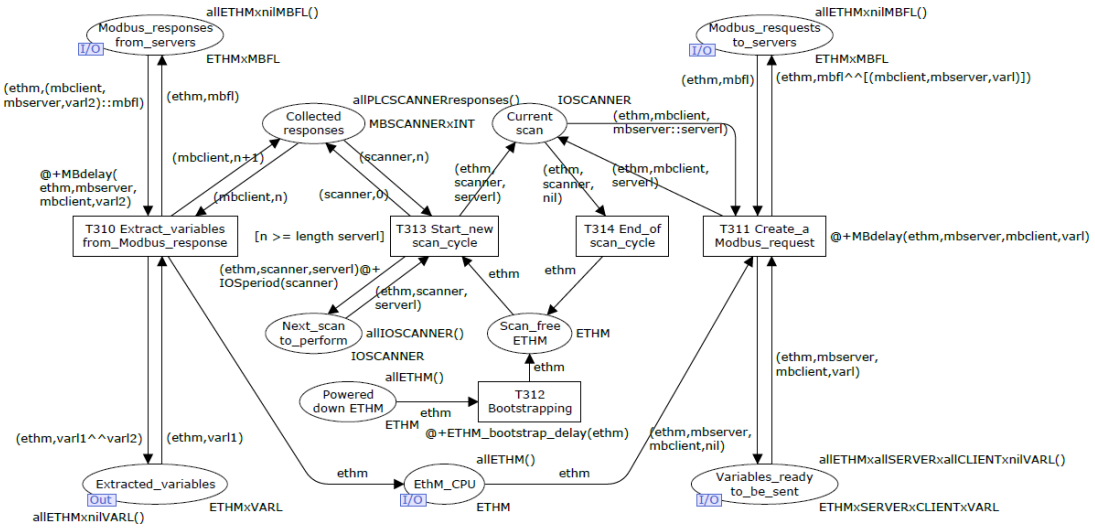


Figure 25: Modèle d'un scan des E/S par un automate selon (Marsal, 2006a)

- des approches centrées sur la modélisation des dispositifs de communication pour déterminer une distribution de la charge et des délais ; la (Figure 26) présente le modèle de communication proposé par (Brahimi, 2007) pour une architecture à base d'Ethernet Commuté.

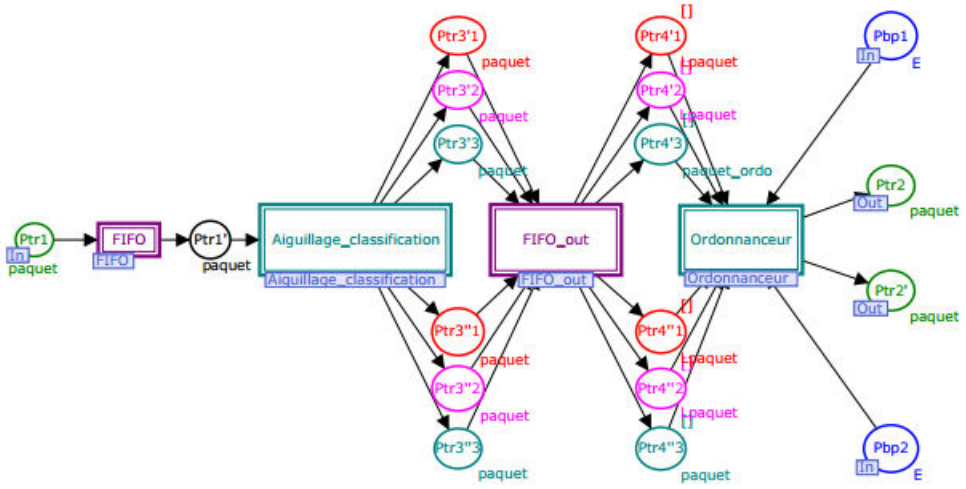


Figure 26: Modèle RdPC d'un switch selon (Brahimi, 2007)

3.2.3 Synthèse

Ces quelques illustrations se sont pas exhaustives dans la mesure où de très nombreux travaux reposent sur les réseaux de Petri colorés et temporisés pour l'évaluation de performances. Ce foisonnement, tant au niveau académique, qu'industriel (dans des entreprises telles que la SNCF (Lalouette, et al., 2010), Ericsson (Vanit-Anunchai, et al., 2006), Helwett-Packard (De Figueiredo & Kristensen, 1999) ou encore PSA (Moncelet, et al., 1998) dénote de la richesse de ce formalisme et de son adaptation au contexte de l'évaluation de performances.

Le champ d'application des réseaux CPN n'est, en outre, pas limité à l'évaluation de performances temporelles puisqu'il est largement utilisé dans le domaine de la sûreté de fonctionnement (Barger, 2003), (Pinna, et al., 2013), (Brinzei, et al., 2014), (Barger, et al., 2009). En effet, ces modèles exploitent l'extension offerte par les réseaux CPN définis par Jensen pour modéliser les durées sous une forme probabiliste et ainsi traduire l'occurrence de défaillances. Même si une telle utilisation, comme nous le verrons à la section 4.3, doit se faire avec quelques précautions d'usage pour conserver le cadre théorique des processus stochastiques, elle offre néanmoins un réel potentiel pour la modélisation de comportements ne pouvant être estimés qu'à l'aide de distributions de probabilités comme cela peut être le cas en phase d'avant-vente.

En revanche, l'ensemble de ces approches laissent apparaître la complexité de la modélisation au travers de modèles comprenant plusieurs centaines d'états et de transitions. On ne peut donc qu'en déduire que la phase de modélisation est coûteuse en ressources humaines et en temps. Ce constat est incompatible avec les besoins exprimés pour l'évaluation de performance en phase d'avant-vente durant laquelle de multiples solutions d'architectures doivent être évaluées avec le support de ressources limitées. Dans ce contexte, la définition de modèles génériques (en exploitant les transitions de substitutions), voire l'automatisation de l'élaboration des modèles constituera une étape indispensable pour lever cet écueil.

4 Evaluation de performances à l'aide de modèles à états et à paramètres stochastiques

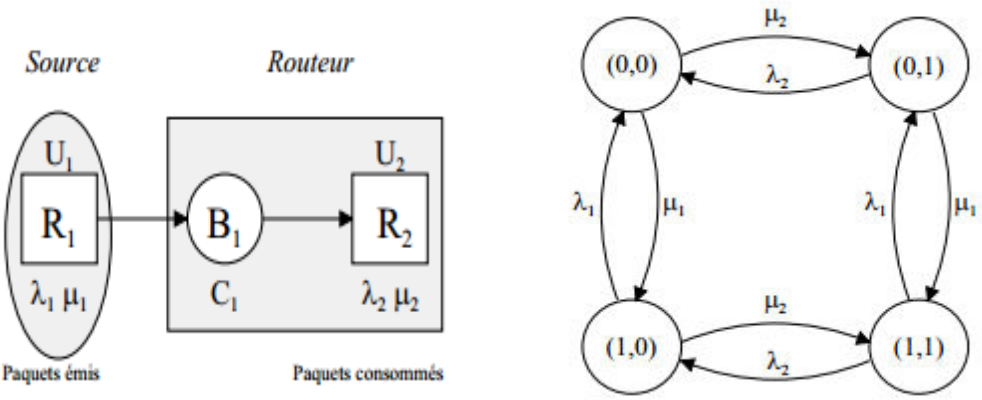
Comme nous l'avons évoqué précédemment, la modélisation de phénomènes aléatoires (tels que des sollicitations externes, des durées d'exécution dépendant de paramètres non déterministes, …) peut s'avérer indispensable dans le cadre d'un processus d'évaluation de performances. Nous présentons dans cette section, trois modèles stochastiques largement utilisés dans ce contexte : les chaînes de Markov, les réseaux de file d'attente et les réseaux de Petri stochastiques. Comme lors de la section précédente, nous présenterons, pour chacun de ces modèles, un bref aperçu de leur fondement théorique ainsi qu'une discussion autour de leur utilisation dans le domaine de l'évaluation de performance.

4.1 Les chaines de Markov

Les chaînes de Markov (CdM) correspondent à une première approche mathématique pour modéliser des systèmes de contrôle/commande en réseau via l'étude des transitions entre différents états à partir de probabilités bien définies. Une chaîne de Markov repose ainsi sur une suite de variables aléatoires (X_n, $n \in N$) qui permet de modéliser l'évolution dynamique

d'un système aléatoire : X_n représente l'état discret du système à l'instant n. La propriété fondamentale des chaînes de Markov, dite propriété de Markov, est qu'il y a plusieurs évolutions possibles à partir d'une situation, chacune d'entre elles ayant une probabilité de se réaliser – sans considérer l'histoire passée $P(X_{t+1}=x_j / X_t=x_i, X_{t-1}=x_k, X_{t-2}=x_l, ...)=P(X_{t+1}=x_j / X_t=x_i)$. C'est cette probabilité $P(X_{n+1}=b \mid X_n=a)$ qui détermine la chance de passer d'un état X_n à un nouvel état X_{n+1} à l'instant suivant. C'est cette incertitude qui introduit la dynamique aléatoire ou stochastique. Cela conduit également à un processus de décisions multiples, usuellement représenté sous la forme de graphe. Les applications des chaînes de Markov sont très nombreuses (réseaux, génétique des populations, mathématiques financières, gestion de stock, algorithmes stochastiques d'optimisation, simulation, . . .).

Le recours aux chaînes de Markov a donné lieu à de nombreux travaux dans la littérature et différentes classes : chaînes de Markov à temps discret, continu, hybrides, interactives, etc. Dans le cadre des réseaux informatiques, les chaînes de Markov sont notamment utilisées pour modéliser les pertes de propagation, notamment dans le cas inhérent des réseaux sans fil. En ce qui concerne l'évaluation de performances temporelles, on mentionnera (Royer, 2006), (Ghanaim, et al., 2009), et (Ishak, et al., 2016) parmi les différents travaux disponibles la littérature. Le but est ici de déterminer des temps moyen, probabilité moyenne de séjour dans un état. Dans (Royer, 2006), une chaîne de Markov est utilisée pour décrire le fonctionnement d'un routeur qui est perçu comme un serveur ON/OFF comme le montre la Figure 27.



(a) Modèle mono buffer d'un routeur (b) Modèle chaine de Markov du buffer

Figure 27: Modélisation d'un routeur avec des CdM (Royer, 2006)

Ici les périodes disponibilités (et d'indisponibilités) suivent des lois exponentielles de moyennes $1/\lambda_i$ ($1/\mu_i$). Comme dans la plupart des utilisations. Dans (Ghanaim, et al., 2009) les états et les probabilités associées correspondent à des valeurs discrètes de délais obtenus par simulations CPN d'un système commandé via un réseau Ethernet. (Ishak, et al., 2016) explicitent l'usage des chaînes de Markov quant à l'évaluation du temps moyen de tirage du *backoff* des réseaux Ethernet, et ainsi des délais d'émission. L'arbre de recherche proposé est décrit à la Figure 28. Comme le montre cette figure, le graphe devient rapidement étendu alors qu'il ne s'attache qu'au plus petit des réseaux : deux émetteurs. Il est à noter aussi que les probabilités nécessaires à l'étude sont ici le fruit d'une distribution uniforme, de par la nature de l'algorithme.

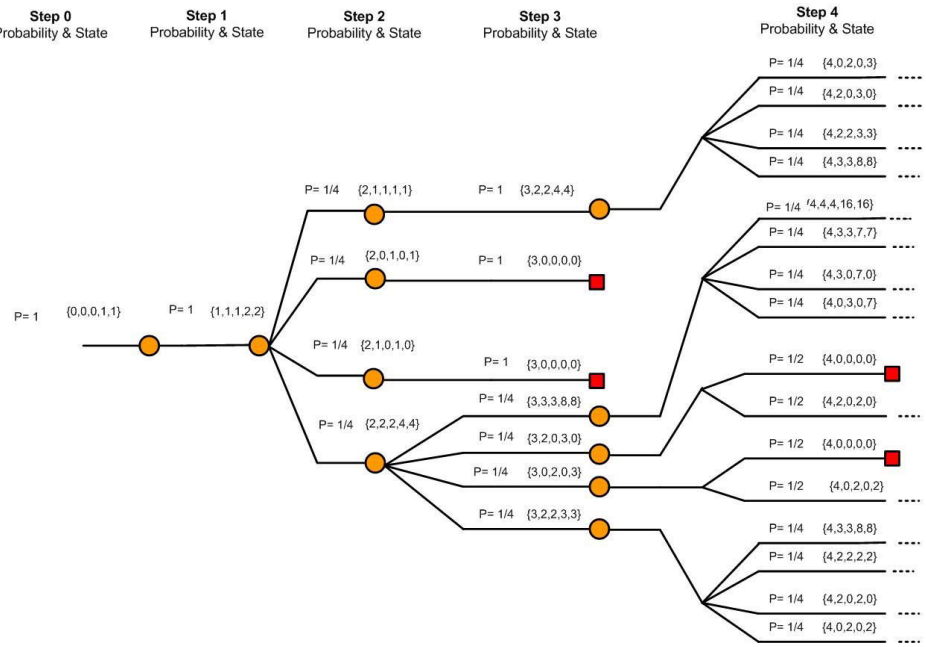


Figure 28: CdM modélisant l'algorithme BEB pour un réseau à deux nœuds (Ishak, et al., 2016)

Pour conclure, les chaînes de Markov sont un outil intéressant pour appréhender les délais moyens de bout en bout sur une architecture en réseau. Il reste néanmoins nécessaire de connaître les probabilités de transition d'un état à un autre, ce qui n'est pas toujours évident, plus spécifiquement dans le cadre de l'avant-vente. Il n'est pas possible de prendre en compte directement une loi de distribution, les valeurs des probabilités nécessitant d'être connue et invariantes. De même, le support de multiples lois de distribution n'est pas directement pris en compte. Enfin, nous noterons qu'un large nombre de décisions (correspondant chacune à un état différent) peut conduire à un très large arbre de recherche.

La section suivante s'intéresse ainsi aux théories des files d'attente développant une approche stochastique qui permettront le support direct des lois de distribution. Notons d'ores et déjà qu'il sera possible de modéliser en théorie des files d'attente des processus markoviens.

4.2 Modélisation en files d'attentes

4.2.1 Algèbre stochastique

La modélisation par file d'attente est une méthode analytique qui permet de représenter le comportement d'un système sous forme d'équations mathématiques comme introduit par (Kleinrock, 1976). La théorie des files d'attente définit ainsi deux types d'entités : les clients (qui modélisent les éléments circulant dans le système réel) et les stations (avec un ou plusieurs serveurs modélisant la ressource et avec une file d'attente caractérisant l'accumulation et l'attente des clients). L'arrivée dans la file est régie selon une loi d'arrivée qui spécifie l'intervalle de temps moyen entre deux arrivées. Une file d'attente est généralement représentée en utilisant la notation de Kendall : A/S/C (L, DS) avec A, le processus d'arrivée, S, le processus de sortie, C le nombre de serveurs, L la capacité maximale de la file et DS la discipline de service (FIFO, LIFO, etc.). Les processus les plus utilisés sont Markovien (loi

exponentielle, poisson), Déterministe (loi constante) ou Générique (loi quelconque). Parmi les plus classiques, la file M/M/1 représente un système avec un processus d'arrivée et de sortie de type Markovien et avec un seul serveur. Une telle file peut d'ailleurs se représenter selon une chaîne de Markov. Pour une file d'attente et à partir de résultats comme la formule de Little, on va alors chercher à évaluer un ensemble de caractéristiques dont le temps de réponse moyen, le temps moyen d'attente des clients, le nombre moyen de clients en attente de service ou le taux d'utilisation du serveur.

Dans le cas où l'architecture est constituée de plusieurs stations interconnectées, on s'intéresse alors à un réseau de file d'attentes. Ce réseau peut être fermé dans le cas d'un nombre fini de clients circulant indéfiniment dans le réseau ou ouvert lorsqu'un client peut quitter le système ou rejoindre une autre station avec une certaine probabilité. En présence d'une seule classe de clients, d'un seul serveur à chaque station, d'une capacité illimitée et de routages probabilistes, on parlera de réseau de Jackson ouverts. Différents résultats permettent d'obtenir aujourd'hui des indicateurs similaires à ceux pour une seule file d'attente. Il existe également d'autres travaux pour des réseaux multi classes (de clients) et à capacité limitée (qui peuvent générer des pertes dues à une saturation, donc plus réalistes mais également plus complexes).

Les files d'attente sont utilisés pour l'étude d'un ensemble varié de domaines (la production, la finance, etc.) et les télécommunications. Parmi les différents travaux d'application de la théorie des files d'attente aux architectures de commande sur un réseau Ethernet, on évoquera les travaux de (Song, et al., 2002) et (Song, 2004.). Dans leurs travaux, les auteurs cherchent à établir le temps de mémorisation dans un commutateur Ethernet. Ces travaux sont établis pour un ensemble de sources de trames périodiques et apériodiques. Plusieurs processus d'arrivées sont étudiés : périodique, loi Binomiale (processus de Bernoulli) et loi de Poisson. Le service est déterministe et ne repose ici que sur un serveur. La théorie des files d'attente leur permet alors d'exprimer la distribution du délai d'attente liée à la traversée d'un commutateur.

Ces travaux mettent en évidence l'intérêt et les limites de cette approche. En effet, l'estimation des délais moyens est aisée dans le cas d'un réseau de faible taille (ici, un seul commutateur est modélisé) et la complexité du modèle devient très vite élevée pour des réseaux de plus grande taille (et notamment multi classes et à capacité limitée), rendant son utilisation difficilement envisageable pour des architectures de contrôle commande de grands systèmes. Ceci serait d'autant plus avéré en incluant la modélisation des différents éléments de commandes (carte d'E/S, PLC, SCADA, etc.).

Pour faire face à cette complexité tout en conservant le support de multiples lois de distribution, une solution consiste à se tourner vers les outils de simulation.

4.2.2 Par simulation

La simulation a comme principal avantage de masquer les théories mathématiques en proposant des interfaces de modélisation explicite. En ce qui concerne la simulation des architectures Ethernet commutées, l'outil principalement utilisé est le simulateur de réseau

Riverbed (ex Opnet) Modeler. D'une manière plus générale, on peut également citer l'outil NS-2(/3) (Network Simulator) ou OmNet+.

Riverbed Modeler permet de décrire une architecture réseau à partir de modèles de composants généralement fournis directement par les différents constructeurs (Cisco, 3Com, etc.). Ces modèles sont décrits à partir d'un pseudo code C lui-même structuré selon des machines à états finis. De multiples lois de distribution sont également disponibles, ce qui permet de recouvrer des résultats similaires à une approche algébrique stochastique. Ce logiciel a notamment été utilisé dans les travaux de (Brahimi, 2007). (Habib, 2010) a, quant à lui, montrer les conditions de modélisation au sein de Modeler d'équipements « non réseaux » de commande. (Haffar, 2011) a quant à lui utilisé Modeler pour la co-simulation d'architectures de distributions électriques, le simulateur gérant la partie communications Modbus TCP/IP.

NS est un simulateur à évènements discrets qui sert à modéliser des réseaux de communication en influant sur un grand nombre de paramètres (débit, taux d'erreur bit, etc). Il est basé sur deux langages de programmation à savoir C++ et OTcl. Il concerne toutefois principalement les cœurs de réseau, et ne dispose pas de modèles complets d'équipements de commande.

Des travaux de simulation conjointe de la commande et du réseau Ethernet commuté ont été mené *via* la librairie TrueTime pour Matlab (Cervin, et al., 2003). Cette librairie a été développée par l'Université de Lund en Suède. Elle permet de co-simuler l'exécution des tâches temps-réels implantées dans le contrôleur et les transmissions via le réseau. L'utilisation de TrueTime privilégie l'utilisation du logiciel Matlab/Simulink. Les blocs « réseau » ajoutés par la librairie ne correspondent toutefois qu'à une modélisation macroscopique des protocoles de communication. De plus, (Habib, 2010) a pu montrer les autres limites de modélisation, notamment par rapport à Modeler.

Comme mentionné ci-avant, d'autres simulateurs existent. Néanmoins, ils partagent de la même manière les limites des logiciels précédemment présentés, notamment la capacité à modéliser dans un même environnement les équipements de commande et d'interconnexion réseau. De plus, pour obtenir des résultats corrects, il est nécessaire d'exécuter un grand nombre de simulations qui nécessitent chacune des ressources (CPU, mémoire). Enfin, pour chaque nouvelle architecture, il est nécessaire de redéfinir un nouveau fichier modèle et relancer l'ensemble des expériences.

4.3 Réseaux de Petri stochastiques

4.3.1 Principes

Les réseaux de Petri stochastiques sont une extension des réseaux de Petri temporisés pour laquelle l'ensemble des trajectoires temporisées est muni d'une mesure de probabilité de telle sorte à ce que le couple (marquage, date d'entrée dans le marquage) soit un processus stochastique. D'un point de vue du formalisme, cela se traduit d'une part, par l'association, à toute transition t_k, d'une variable temporelle aléatoire λ_k caractérisée par une distribution de probabilité, et d'autre part, par la définition d'une politique d'exécution permettant de définir le processus stochastique.

La politique d'exécution comprend deux éléments principaux :

- la politique de sélection qui permet de définir la transition devant être tirée en cas de transitions simultanément sensibilisées depuis un marquage (Figure 29) ; deux techniques sont distinguées :
    - la présélection qui consiste d'abord à choisir une transition à l'aide de probabilités de franchissement puis à effectuer le tirage aléatoire correspondant à la transition sélectionnée,
    - la compétition qui consiste à effectuer tous les tirages aléatoires puis de sélectionner la transition ayant la variable temporelle la plus faible ;
- la politique de mémorisation qui définit la prise en compte du passé, lors de l'entrée dans un marquage, de trois manières :
    - avec réinitialisation qui, lors du franchissement d'une transition t_k consiste à ne pas conserver les valeurs temporelles des tirages effectués sur toutes les transitions sensibilisées simultanément à t_k ; un nouveau choix de franchissement conduira donc à un nouveau tirage ;
    - avec mémoire de la dernière période (*enabling memory*) qui considère que la mémoire des tirages se maintient tant que les transitions restent sensibilisées (par exemple, sur la Figure 29, un nouveau tirage sera effectué lorsque t2 sera à nouveau franchissable tandis que le tirage sera conservé et décrémenté pour t3) ;
    - avec mémoire de toutes les périodes (*age memory*) même si les transitions sont désensibilisées.

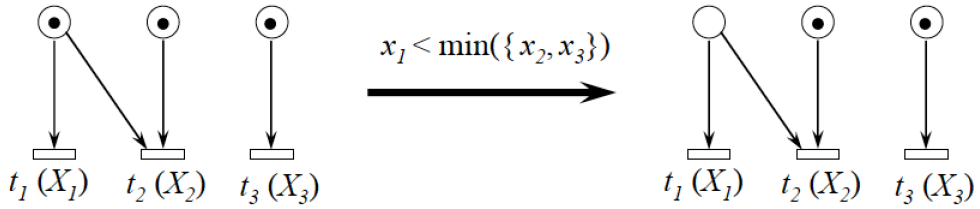


Figure 29: Politique de sélection par compétition

La nature du processus stochastique dépendra donc de la politique d'exécution choisie. Si l'on considère une politique de réinitialisation, le processus sera markovien ou semi-markovien quel que soit la distribution de probabilités associée aux variables temporelles aléatoires λ_k. En revanche, si l'on considère une politique avec mémorisation de la dernière période ou de toutes les périodes (ce qui implique le choix d'un mode de sélection par compétition), cela dépendra des types de distributions. Dans ce contexte, plusieurs types de réseaux de Petri stochastiques devront être considérés :

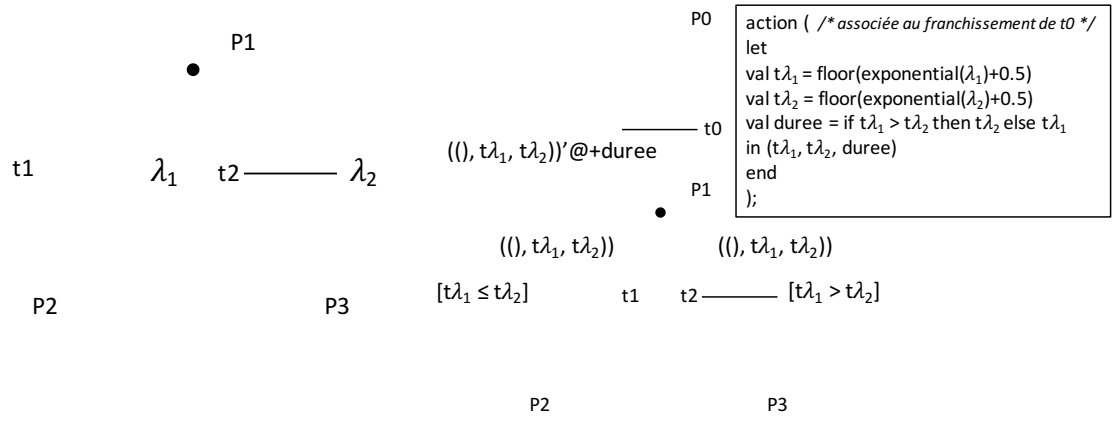
- les réseaux SPN, pour Stochastic Petri Nets (Natkin, 1980), ne considèrent que des distributions de probabilités exponentielles ; avec une politique de compétition, le graphe de marquage de ce type de réseau est isomorphe avec une chaine de Markov à temps continu sur laquelle des calculs de performances peuvent être réalisés ;
- les réseaux GSPN, pour Generalized Stochastic Petri Nets (Marsan, et al., 1984) introduisent en complément des transitions à durée distribuée exponentiellement, des transitions immédiates à durée nulle ; la technique de compétition est utilisée entre transitions temporisées et entre transitions temporisées et immédiates (avec une

priorité pour les transitions immédiates) tandis que la présélection est utilisée entre transitions immédiates ; le processus stochastique associé aux états de durée non nulle est encore un processus markovien qui peut être analysé à l'aide de techniques associées aux chaines de Markov immergée ;

- les réseaux ESPN, pour Extended Stochastic Petri Nets (Dugan, et al., 1984) considèrent des distributions quelconques, mis à part pour les transitions en parallèle pour lesquelles les distributions sont exponentielles ; le graphe de marquage des ESPN est semi-markovien ;
- les réseaux DSPN, pour Deterministic Stochastic Petri Nets (Marsan & Chiola, 1986), sont les plus complexes puisqu'ils considèrent à la fois des transitions immédiates, des transitions à durée exponentielle et des transitions à durée déterministe ; le processus stochastique est semi-markovien régénératif.

Le modèle à la base des réseaux de Petri colorés, hiérarchiques et temporisés définis par Jensen n'est donc conforme à aucun de ces modèles. En effet, il autorise des transitions à durée aléatoire sans restriction sur les distributions de probabilités (qui peuvent de type exponentielle, Weibull, Erlang, uniforme ou autre) à l'instar des ESPN mais également des transitions immédiates ou à durée fixe (comme pour les DSPN). La conséquence immédiate est que, sauf à émettre des restrictions importantes sur les modèles élaborés, l'analyse de ces réseaux imposera, dans la plupart des cas, le recours à la simulation.

Ce constat est renforcé par une autre différence majeure relative à la gestion du temps. Dans les réseaux de Petri stochastiques, les durées sont affectées aux transitions alors qu'elles le sont aux places sur les réseaux CPN de Jensen. Prenons, par exemple, le cas du réseau représenté par la Figure 30-a. Si on le considère comme un GSPN, la politique de compétition entre transitions temporisées (transitions *t1* et *t2*) conduira à sélectionner la transition pour laquelle la variable temporelle aléatoire sera la plus faible ; dans tous les cas, le jeton sera maintenu dans la place P conformément à la durée sélectionnée.



(a) Transitions temporisées en conflit (b) Modélisation CPN équivalente à un GSPN

Figure 30: GSPN vs CPN

Si on considère le réseau de la Figure 30-a comme un réseau CPN à la Jensen, la signification des distributions λ_i sera différente. En effet, dans un réseau CPN, le temps est représenté par une horloge globale. En plus de leur couleur, les jetons d'un marquage contiennent une valeur

temporelle, également appelée *time stamp*, qui caractérise la valeur globale de l'horloge à partir de laquelle le jeton est disponible pour sensibiliser une transition. Lorsqu'une transition est sensibilisée, elle est immédiatement tirée et dépose dans ses places de sortie un jeton dont le *time stamp* prend la valeur de la date de franchissement augmentée de la durée aléatoire associée à la transition. Le jeton ainsi déposé reste donc indisponible dans les places de sortie jusqu'à jusqu'à ce que le temps courant de l'horloge globale soit supérieur ou égal à son *time stamp*. Ce comportement correspond donc bien à une P-temporisation des places de sortie fonctionnant à sa vitesse maximale. Pour les transitions immédiates, cette différence d'interprétation ne conduit à aucun écart de comportement entre GSPN et CPN mais ce n'est malheureusement pas le cas pour les transitions stochastiques. En effet, si le réseau de Petri de la Figure 30-a est un CPN, dès que le jeton arrive dans la place *P1*, il est affecté à l'une des deux transitions *t1* ou *t2* (de manière aléatoire) et cette transition est immédiatement tirée. Le jeton déposé dans la place P2 ou P3 est alors affecté d'une durée de séjour égale à la durée correspondant au taux de la transition *t1* ou *t2* et il séjourne dans la place P2 ou P3 aussi longtemps que requis par son *time stamp*.

Pour résoudre ce problème et assurer un comportement équivalent à un GSPN avec mémorisation de la dernière période (sous réserve de n'utiliser que des distributions exponentielles ou des transitions immédiates), deux approches sont envisageables (Pinna, et al., 2013) :

- interdire dans les modèles CPN la compétition entre deux transitions à durée aléatoire,
- introduire une place supplémentaire *P0* (Figure 30-b) permettant de traiter le choix stochastique de la concurrence avant d'autoriser les transitions t1 et t2. Les tirages aléatoires associés aux transitions t1 et t2 sont tous deux effectués lors du franchissement de la transition d'entrée de cette place intermédiaire, le jeton déposé dans la place intermédiaire séjourne dans cette place pendant une durée égale à la plus petite valeur des variables aléatoires tirées et seule la transition correspondant à cette valeur peut être tirée en lui ajoutant une garde ; ces différents traitements peuvent être réalisés à l'aide des fonctions ML disponibles dans les modèles CPN de Jensen.

4.3.2 Applications à l'évaluation de performances

Les réseaux de Petri stochastiques peuvent être appliqués à l'évaluation de performances à l'aide d'approches :

- formelles au travers d'analyses qualitatives (bornitude, vivacité) et quantitatives (probabilités des états, temps de séjour, temps d'absorption, temps de premier passage, …) d'un graphe de marquage probabilisé. Des outils tels que GreatSPN[2] (Baarir, et al., 2009) ou encore SPNP (Hirel, et al., 2000) fournissent un éditeur et des algorithmes d'analyse. A titre d'illustration et sans être exhaustif, (Juanole & Gallo, 1995) modélise un protocole de réseaux industriels de terrain à l'aide d'une classe spécifique de réseaux de Petri stochastiques (les STPN pour Stochastic Timed Petri Nets) introduisant la modélisation du temps aléatoire sous la forme d'intervalles pour exhiber les performances en termes de temps de réponse dans un contexte critique à l'aide d'un

[2] http://www.di.unito.it/~greatspn/index.html

graphe d'états probabilisé (Figure 31) ; (Labadi, 2005) utilise l'analyse de graphes pour déterminer les performances d'un système logistique modélisé avec un réseau de Petri stochastique enrichi par la notion de lots ; dans le domaine de la gestion de production, l'analyse de GSPN permet d'évaluer des taux moyens de production ou de stocks (Al-Jaar & Desrochers, 1990), (Itsuo, et al., 1991);

- basées sur la simulation de Monte-Carlo, notamment pour les réseaux comportant des distributions de probabilités non exponentielles et combinant des transitions temporisées aléatoires, à durée fixe et/ou immédiates ; des outils de simulation, tels que TimeNET[3], (Zimmermann, 2012) et dans une certaine mesure CPN Tools (avec les précautions d'usage mentionnées à la section précédente) facilitent la mise en œuvre de ces approches basées sur la simulation. A titre d'illustration, (Heindl & Reinhard, 2001) utilise la simulation de réseaux de Petri stochastiques pour évaluer la procédure de backoff et l'espace inter-trames sur des réseaux WiFi.

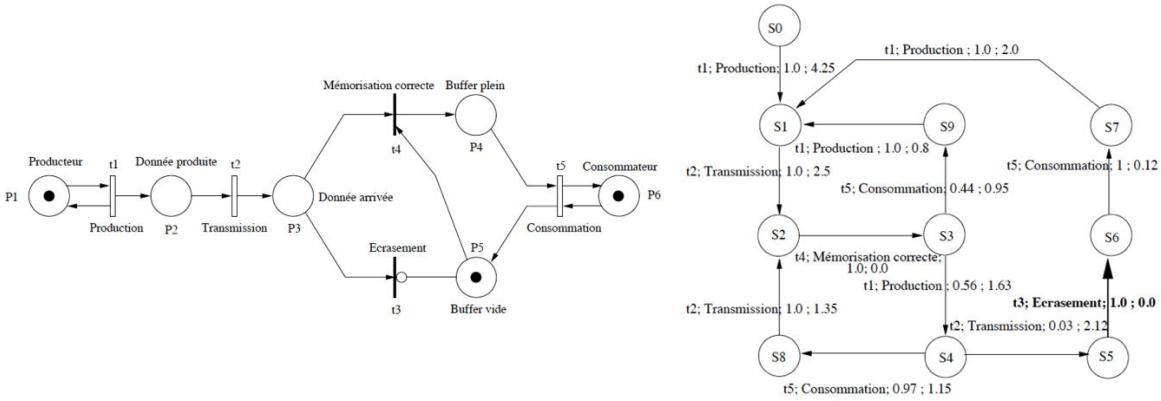


Figure 31: Modèle STPN et son graphe d'états probabilisé (Juanole & Gallo, 1995)

Les nombreuses applications des réseaux de Petri stochastiques à l'évaluation de performance, que soit dans le domaine des réseaux de communications, des systèmes informatiques ou encore de la gestion de production, montrent l'intérêt significatif que peut accorder à ces modèles la communauté scientifique mais aussi industrielle. Ces modèles permettent en effet la modélisation des tâches d'un système dont la durée d'exécution ne peut être modélisée qu'à travers une durée non déterministe. En cela, ils répondent parfaitement aux besoins exprimés pour l'évaluation de performances en phase d'avant-vente. En revanche, l'analyse formelle de ces modèles peut être confrontée à des phénomènes d'explosion combinatoire pour des modèles de grande taille et impose, dans la plupart des cas, le recours à des distributions de probabilités exponentielles permettant de se ramener dans un cadre markovien ou semi-markovien. Si le recours à d'autres types de distributions (uniformes, Weibull, Erlang, …) est nécessaire ou si les modèles à analyser sont de grande taille (plusieurs centaines de places), le recours à la simulation de Monte-Carlo peut s'avérer indispensable. Ces deux conditions sont a priori réunies dans le cadre de notre étude et orientent donc naturellement notre choix vers la simulation. Dans ce contexte et sous réserve des hypothèses énoncées précédemment, les réseaux CPN de Jensen et leur outil support CPN Tools apparaissent comme une alternative pertinente pour l'évaluation de performances temporelles à l'aide de réseaux de Petri.

[3] https://www.tu-ilmenau.de/sse/timenet/

5 Conclusions

Dans ce chapitre, nous avons cherché à évaluer l'adéquation à notre étude des différents formalismes de modélisation utilisés pour l'évaluation de performances. Nous avons ainsi organisé un état de l'art suivant trois grandes approches. Pour chaque formalisme, nous avons cherché à expliciter les éventuelles limites par rapport aux différentes exigences permettant d'une part, de couvrir la structure et la hiérarchisation des architectures à évaluer et, d'autre part, de modéliser le comportement dynamique des architectures dans un contexte incertain. Nous rappelons ainsi les points clés des éléments de synthèse mis en avant.

Pour les modèles algébriques déterministes, ils nécessitent des paramètres d'entrée déterministes pas simple à identifier en phase d'avant-vente et qui ne supportent pas explicitement une prise en compte de l'incertitude. De plus, les expressions de type bornes (minimales et/ou maximales) peuvent souffrir d'un conservatisme, qui en phase d'avant-vente, peut engendrer une forte surestimation de l'architecture. La recherche des cas critiques peut s'avérer fastidieuse et doit être intégralement relancée à chaque fois que l'architecture est modifiée, ce qui ne permettra pas de tester un large nombre de solutions. Enfin, ces approches algébriques n'intègrent pas de structuration hiérarchique.

En ce qui concerne les modèles à états à paramètres déterministes, les solutions à base de model-checking souffrent généralement d'un problème classique d'explosion combinatoire qui peut conduire à un échec du processus de vérification lorsque le nombre d'états à explorer est trop important et reste un frein pour l'évaluation de systèmes complexes de grande taille. De plus, ces techniques ne sont pas adaptées au contexte particulier de l'avant-vente où il reste très difficile de caractériser le comportement de l'architecture de contrôle-commande de manière déterministe avec des durées ou des plages de durée fixes. Quant aux réseaux de Petri et les extensions colorés et hiérarchique, ils présentent un intérêt très significatif puisque la modélisation du temps sous la forme d'intervalles dans les RdP t-temporels peut traduire une incertitude sur la durée et permettent la modélisation d'architectures de contrôle-commande caractérisées par une structuration hiérarchique et par la présence de composants dont les comportements sont similaires. En revanche, l'ensemble de ces approches laissent apparaître la complexité de la modélisation au travers de modèles comprenant plusieurs centaines d'états et de transitions. On ne peut donc qu'en déduire que la phase de modélisation est coûteuse en ressources humaines et en temps. Ce constat est incompatible avec les besoins exprimés pour l'évaluation de performance en phase d'avant-vente durant laquelle de multiples solutions d'architectures doivent être évaluées avec le support de ressources limitées.

Pour les modèles stochastiques de type chaînes de Markov, il reste néanmoins nécessaire de connaître les probabilités de transition d'un état à un autre, ce qui n'est pas toujours évident, plus spécifiquement dans le cadre de l'avant-vente. De plus, le support de multiples lois de distribution (autres qu'exponentielle) n'est pas directement pris en compte. Enfin, nous noterons qu'un large nombre de décisions (correspondant chacune à un état différent) peut conduire à un très large arbre de recherche. Pour la théorie des files d'attente, la complexité du modèle devient très vite élevée pour des réseaux de plus grande taille (et notamment multi

classes et à capacité limitée), rendant son utilisation difficilement envisageable pour des architectures de contrôle commande de grands systèmes. Et les simulateurs supportant la théorie des files d'attente ne proposent pas de structuration générique et hiérarchique, de sorte qu'il ne soit pas possible de réduire les temps d'exécution. Les réseaux de Petri stochastique permettent quant à eux la modélisation des tâches d'un système dont la durée d'exécution ne peut être modélisée qu'à travers une durée non déterministe. En cela, ils répondent parfaitement aux besoins exprimés pour l'évaluation de performances en phase d'avant-vente. En revanche, des phénomènes d'explosion combinatoire pour des modèles de grande taille sont probables et imposent, dans la plupart des cas, le recours à des distributions de probabilités exponentielles permettant de se ramener dans un cadre markovien ou semi-markovien.

Ces différentes observations nous amènent alors à promouvoir pour l'évaluation de performances d'architectures de contrôle/commande en avant-vente des réseaux CPN de Jensen et leur outil support CPN Tools (pour la simulation de Monte-Carlo). Dans le chapitre suivant, il s'agira également d'une part de rendre la modélisation des composants des architectures la plus transparente aux utilisateurs finaux (qui ne connaissent pas forcément le formalisme) et d'autre part, de fournir une génération automatisée du modèle, de sorte que ce temps de construction soit le plus réduit possible et qu'ainsi, davantage d'architectures puissent être évaluées.

Chapitre 3
Génération automatique de modèles pour l'évaluation de performances

1 Introduction

L'objectif de ce travail est de définir un cadre formel de modélisation, basé sur les réseaux de Petri colorés et temporisés, permettant la génération automatique de modèles servant de support à l'évaluation de performances des architectures de contrôle-commande en réseau. Cette évaluation, réalisée à l'aide de simulation de Monte-Carlo, doit pouvoir être effectuée pour n'importe quel type d'architecture quel que soit sa complexité, dans le contexte incertain d'avant-vente. Notre contribution se décline en trois points principaux : la définition d'un réseau de Petri « constructeur » qui coordonne la génération automatique des modèles, la définition d'une bibliothèque de modèles de composants élémentaires et enfin la génération d'observateurs permettant l'évaluation des performances. Avant d'aborder successivement ces trois points, la première section de ce chapitre présente une vue d'ensemble de la démarche.

2 Vue d'ensemble de la méthode proposée

La Figure 32 présente une vue d'ensemble de la démarche suivie : les éléments pleins de cette figure constituent nos contributions majeures et feront l'objet des différentes sections de ce chapitre (numérotations sur la figure).

2.1 Données d'entrée

Le point d'entrée du processus d'évaluation des performances est constitué de la description de l'architecture et des performances à évaluer.

Les offres d'architectures sont élaborées en réponse à un cahier des charges « client » à l'aide d'un langage spécifique au domaine (DSL) (Van Deursen, et al., 2000). Ce langage peut être formel ou informel, par exemple sous une forme graphique. Conformément aux modèles UML présentés au chapitre 1, la description des architectures contient l'ensemble des paramètres relatifs à la topologie de l'architecture (nomenclature des équipements, réseau de connexions ente équipements, …) ainsi qu'une estimation des flux de communication, qu'ils soient périodiques – par exemple entre un automate et les capteurs/actionneurs qu'il pilote – ou apériodiques – par exemple pour caractériser des sollicitations par des opérateurs humains.

D'autre part, la description des architectures doit également contenir une description exhaustive des performances qui devront être évaluées. Cette description des performances attendues sera accompagnée d'indicateurs de niveau de confiance qui permettront de construire les paramètres des simulations de Monte-Carlo (nombre d'histoire, critères d'arrêt, …).

2.2 Génération des modèles d'architecture

Le raisonnement suivi pour proposer notre méthode de génération automatique des modèles, supports à l'évaluation de performances, s'inscrit dans une logique basée sur l'assemblage de modèles d'équipements présents au catalogue constructeur (COTS – Components On The Shelves (McKinney, 2001)). Ce raisonnement est justifié si l'on considère qu'une architecture de contrôle-commande contient un grand nombre d'équipements identiques ou de la même famille dont le comportement peut être décrit à l'aide de modèles génériques et instanciables.

La mise en œuvre de ce raisonnement impose la définition :

- d'une bibliothèque de modèles élémentaires dont les interfaces doivent être standardisées afin de pouvoir les assembler dans une logique « *plug and play* » et respecter leurs contraintes d'interopérabilité (Mezini & Karl, 1998)
- d'un modèle « *constructeur* » décrivant des structures topologiques génériques pour les architectures mais aussi permettant d'instancier et de paramétrer les modèles de la bibliothèque, notamment à l'aide de fonctions d'incidence arrière « *post* » associées aux arcs de sortie des transitions ainsi qu'aux couleurs des jetons.

Une fois la bibliothèque d'équipements construite et le modèle « *constructeur* » défini, l'effort de modélisation, pour toute nouvelle architecture conforme à la topologie type décrite par le modèle « *constructeur* », se limitera à l'exécution d'un algorithme permettant de générer le modèle servant à l'évaluation des performances.

Plusieurs précautions d'usage doivent être néanmoins être soulignées :

- il est évident que l'introduction d'un nouvel équipement au catalogue constructeur nécessitera l'enrichissement de la bibliothèque de modèles élémentaires et induira donc un effort de modélisation supplémentaire ;
- le modèle « *constructeur* » est associée à une famille de typologie d'architecture (par exemple des architectures distribuées en réseau contenant des commutateurs, des PLC, des boitiers d'entrées/sorties déportés, des SCADA et des postes client) ; l'ajout d'une nouvelle famille d'équipements – par exemple, des cartes électroniques de contrôles – nécessitera une légère modification du modèle « *constructeur* » ; en revanche, la suppression d'une famille ne posera pas de problèmes particuliers dans la mesure où elle pourra être prise en charge lors de l'instanciation des composants (il suffira de ne pas instancier la famille en question) ; nous pouvons noter que, chez Schneider Electric, le modèle « constructeur » a pu être conservé à l'identique pour l'ensemble des cas d'étude traités.

Enfin, il est clair que les définitions de la bibliothèque de modèles d'équipements ainsi que du modèle « constructeur » sont intimement liées puisque les fonctions d'instanciation et de paramétrage supportées par le modèle « *constructeur* » auront un impact sur le comportement des modèles d'équipements. Autrement dit, si certaines couleurs seront spécifiques aux modèles d'équipements, certaines couleurs complexes devront être partagées et communes pour l'ensemble des modèles (équipements et « *constructeur »*). Dans la suite de ce chapitre, nous avons pris le parti de présenter successivement le modèle « *constructeur* » (Ndiaye, et al., 2016) puis les modèles d'équipements même si les effets de certains mécanismes de paramétrage du modèle « *constructeur* » ne pourront être observés qu'ultérieurement sur les modèles d'équipement.

2.3 Génération des observateurs de performances

A partir de la description des performances qui doivent être évaluées, l'objectif est de générer automatiquement les observateurs requis et de définir leur insertion dans le modèle CPN de l'architecture. Dans le formalisme défini par (Jensen, et al., 2007), ces observateurs sont appelés « moniteurs ». Ces indicateurs de performances attendus ne pouvant, par

hypothèse, être anticipés, la définition des moniteurs ne peut être réalisée de manière statique sur les modèles génériques d'équipement. Dès lors, la génération des moniteurs repose sur :

- la définition d'une structure générique de moniteurs pour chaque familles de performances définies en section 3 et leur intégration, a priori, sur des places et transitions dédiées dans les modèles d'équipements,
- le développement d'un algorithme permettant le relevé automatique des valeurs statistiques observées lors de la simulation et de leur traitement en fonction des performances à évaluer.

Enfin, l'indice de confiance souhaité par les ingénieurs en charge de l'élaboration des offres d'architectures sera pris en compte pour déterminer les paramètres d'arrêt de la simulation et le nombre d'histoires à simuler.

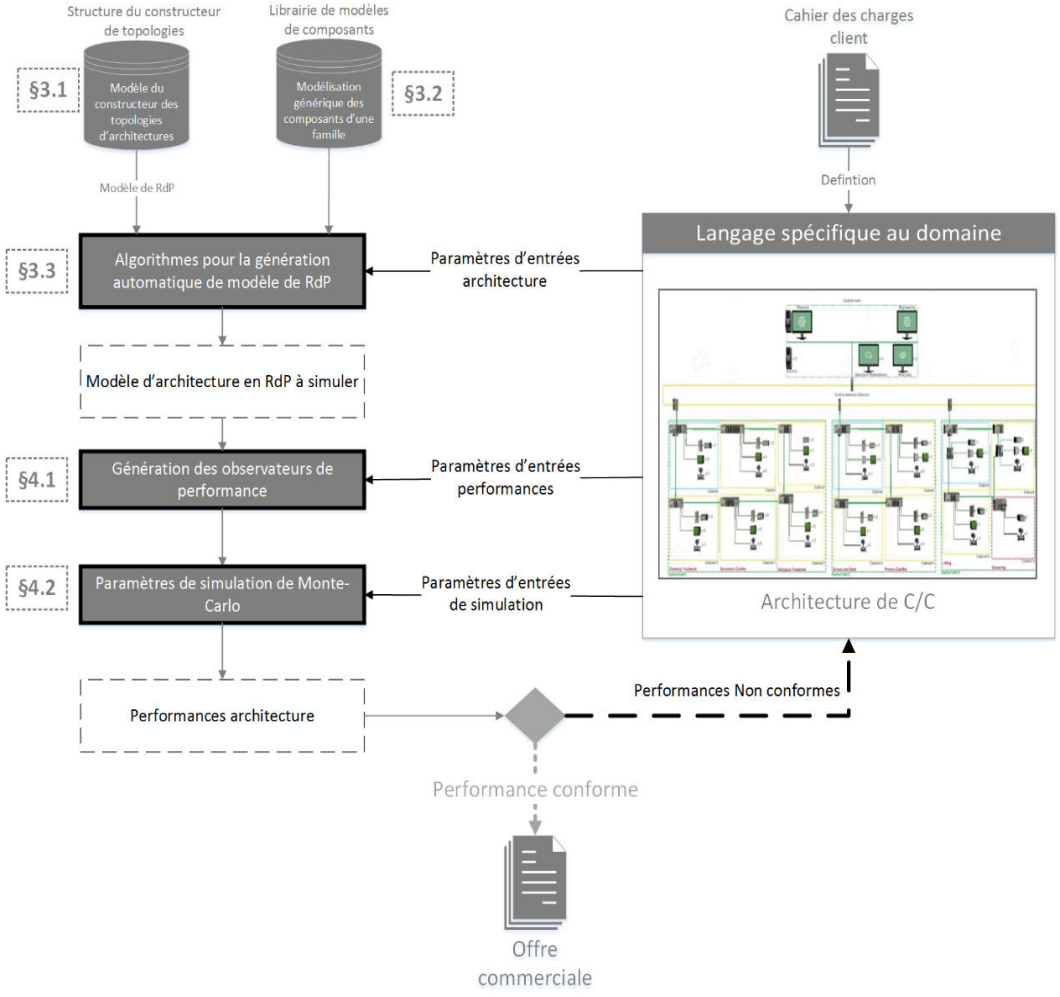


Figure 32 : Vue d'ensemble de la méthode proposée

3 Génération automatique du modèle d'architecture

Comme indiqué dans la section précédente, la génération d'un modèle CPN de l'architecture à évaluer est réalisée, d'une part, à partir d'un modèle générique décrivant une topologie type d'architecture et embarquant des mécanismes d'instanciation et de paramétrage, et, d'autre part, à partir d'une bibliothèque de composants génériques. Sur cette base, un algorithme de génération procède à l'instanciation et au paramétrage du modèle de haut niveau en fonction

des spécifications d'architecture pour obtenir, de manière automatique, le modèle CPN d'une architecture à évaluer.

3.1 Modèle du constructeur des topologies d'architectures

Le modèle « *constructeur* », dont les principes sont présentés en Figure 33, peut être caractérisé par trois éléments principaux :

- sa structure en termes de places et de transitions,
- un ensemble de couleurs associées aux jetons composant les marquages du réseau,
- un ensemble de fonctions d'incidence arrière (« *post* »).

3.1.1 Structure du modèle « *constructeur* »

Le modèle « constructeur » repose sur une classification hiérarchique des équipements d'une architecture selon trois niveaux :

- niveau *architecture* : regroupe l'ensemble des familles d'équipements d'une architecture ;
- niveau *famille* : regroupe l'ensemble des équipements d'une famille dont la mission est semblable mais dont le comportement peut varier de manière significative d'un équipement à l'autre ; autrement dit, les équipements de deux familles différentes ne pourront être décrit par un même modèle même si celui-ci est paramétrable ; une famille est identifiée par un entier strictement positif i ;
- niveau *composant :* regroupe les équipements d'une même famille dont le comportement est suffisamment proche pour être décrit par un modèle générique unique paramétrable. Les paramètres sont des spécifications techniques des équipements qui n'influent pas sur leur principe de fonctionnement ; un composant est identifié par un indice i_j où i est l'identifiant de la famille à laquelle il appartient et j un entier strictement positif.

Remarque

> Dans le cadre de cette étude, nous ferons l'hypothèse que la famille des équipements de communication est toujours présente dans le modèle « *constructeur* », qu'il n'existe qu'un seul type de composant dans cette famille – le commutateur – et que les architectures ne contiennent qu'une seule instance de commutateur. Cette hypothèse, certes restrictive, ne modifie pas fondamentalement les principes d'instanciation et de paramétrage supporté par le modèle « *constructeur* » et qu'elle est suffisamment représentative à la fois de la démarche proposée et des besoins exprimés par Schneider Electric.
>
> Dans la suite du mémoire, la famille « réseaux » sera identifiée par l'indice $i = 1$ mais sera associée à la couleur nommée *Famille_Réseaux* dans les modèles pour des raisons de lisibilité. Les autres familles porteront donc des indices croissants à partir de $i = 2$ pour leur indentification et *Famille_i* avec $i \geq 2$ pour la couleur associée. De même, le commutateur, seul composant, par hypothèse, de la famille « réseaux », sera identifié par *composant_1_1* et sera associée à la couleur *Composant_Reseaux* dans les modèles.

Exemple

> *Familles* : Automates Programmables Industriels (PLC), systèmes de supervision (SCADA), postes clients, boitiers d'entrées/sorties déportés (I/O devices).
> *Composant* : dans la famille PLC, il existe plusieurs offres dont le M340 ou le M580.
> *Paramètre* : le PLC M580 possède deux variantes ayant chacune un nombre de requêtes traitées par cycle qui diffère (12 requêtes pour la variante M580 20/20 et 50 requêtes pour la variante M580 40/40).

Le modèle « *constructeur* » peut être découpé en deux sous-ensembles distincts de places et de transitions jouant chacun un rôle différent comme l'indique la Figure 33:

- un sous-ensemble composé de transitions de substitutions et de places ayant exclusivement en entrée ou en sortie ces transitions de substitutions ; ce sous-ensemble décrit le **comportement de l'architecture** : les transitions de substitutions représentent les différents équipements de l'architecture et les places représentent les échanges d'informations entre ces équipements ;
- un sous-ensemble composé de transitions quasi-vivantes, c'est à dire franchissables une et une seule fois depuis le marquage initial et de places connectées à ces transitions ; ce sous-ensemble est dédié à **l'instanciation et au paramétrage du modèle d'architecture**, notamment à l'aide des fonctions d'incidence arrière présentes sur les arcs entre transitions et places de ce sous-ensemble.

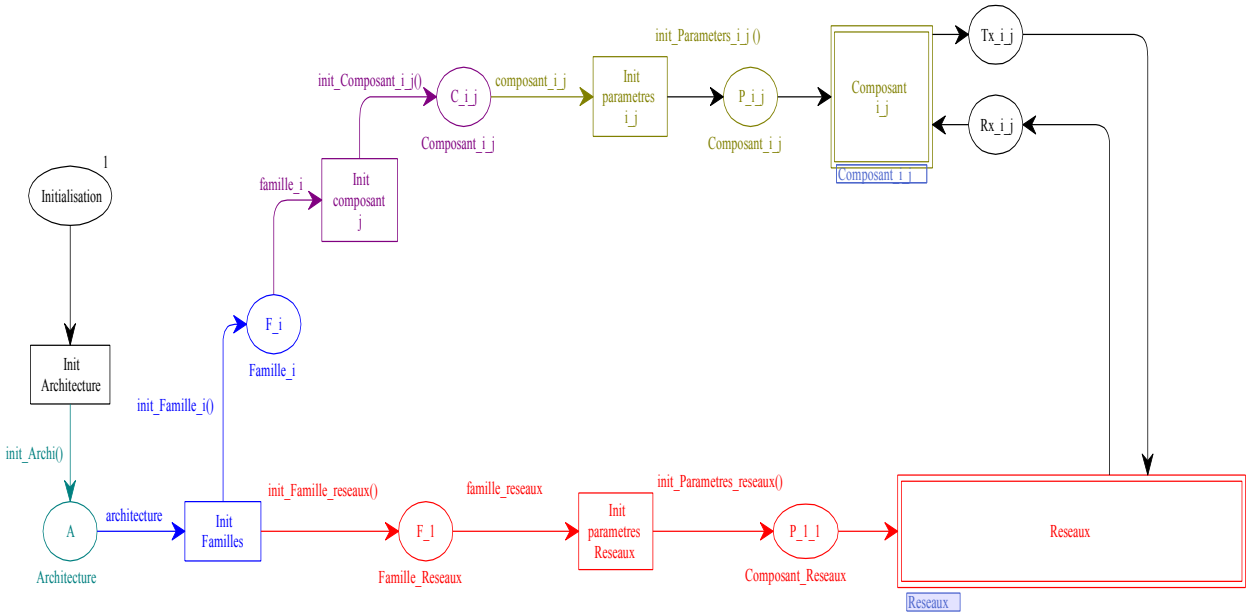


Figure 33: Structure générique du modèle « *constructeur* » de topologies d'architecture

3.1.1.1 Sous-ensemble « Modèle comportemental de l'architecture »

Ce sous-ensemble du modèle « *constructeur* » est un modèle CPN hiérarchique où chaque transition de substitution représente le modèle générique d'un équipement. Il repose sur une hypothèse forte, parfaitement justifiée pour les architectures de contrôle-commande en réseau : tous les équipements d'une architecture sont connectés à un réseau de communication.

Deux types de transitions de substitutions peuvent donc être distingués :

- les transitions de substitution décrivant les équipements de contrôle-commande,
- une transition de substitution décrivant les équipements de communication auxquels sont connectés les équipements de contrôle-commande ; contrairement aux équipements de contrôle-commande, cette transition de substitution est unique puisque nous avons fait l'hypothèse d'un composant unique dans la famille des équipements de communication.

Pour des raisons d'interopérabilité entre modèles d'équipements de contrôle-commande et modèle d'équipements de communication, l'interface entre les transitions de substitutions associées à ces deux familles est décrite de manière standard par deux places :

- les places *Tx* représentent une connexion sortante d'un équipement de contrôle-commande vers un équipement de communication
- les places *Rx* représentent une connexion entrante d'un équipement de communication vers un équipement de contrôle-commande

Sur la Figure 33, les places *Tx_i_j*, *Rx_i_j* sont les places *Tx* et *Rx* associées à un type de composant ayant *i_j* pour identifiant.

Pour rappel, une transition de substitution fait référence à un modèle CPN de niveau hiérarchique inférieur par l'intermédiaire d'une relation de correspondance entre les places d'entrées et de sorties de la transition de substitution et celles du modèle CPN (port) auquel elle fait appel. Il s'ensuit qu'une transition de substitution du modèle « constructeur » (*composant i_j*) sera associée à un et un seul modèle CPN stocké en bibliothèque. Elle représentera alors le comportement d'un type de composant présent dans l'architecture mais nécessitera un traitement complémentaire pour représenter le comportement de plusieurs équipements appartenant au même type de composant (et donc à la même famille). Ce point est pris en charge par le sous-ensemble dédié à l'instanciation et au paramétrage.

Enfin, comme indiqué en début de chapitre, le modèle « *constructeur* » est défini pour une classe d'architecture dont les familles et composants sont supposés connus. L'ajout de nouvelles familles ou de nouveaux composants imposera, bien entendu, la redéfinition d'un modèle « *constructeur* » approprié. A titre d'exemple, la Figure 34 présente un modèle « *constructeur* » proposant trois familles : *famille réseaux* (place *F_1*), une deuxième famille (place *F_2*) et une troisième famille (place *F_3*). Les couleurs associées aux places de ces familles se définissent respectivement par *Famille_Reseaux, Famille_2* et *Famille_3*. La *Famille_2* comporte deux composants nommés C_2_1 et C_2_2 dont leurs couleurs sont respectivement *Composant_2_1* et *Composant_2_2* et la *Famille_3* comporte un composant unique appelé C_ 3_1 dont la couleur est *Composant_3_1*.

Notons que, même s'ils figurent dans le modèle « *constructeur* », les couleurs des Composants *2_2* et *3_1* ne sont pas nécessairement présents dans l'architecture dont on souhaite évaluer les performances. En effet, les mécanismes d'instanciation, qui seront détaillés par la suite, permettent de choisir, parmi les familles et composants proposés par le modèle « *constructeur* », quels sont ceux qui seront réellement utilisés dans une proposition d'architecture.

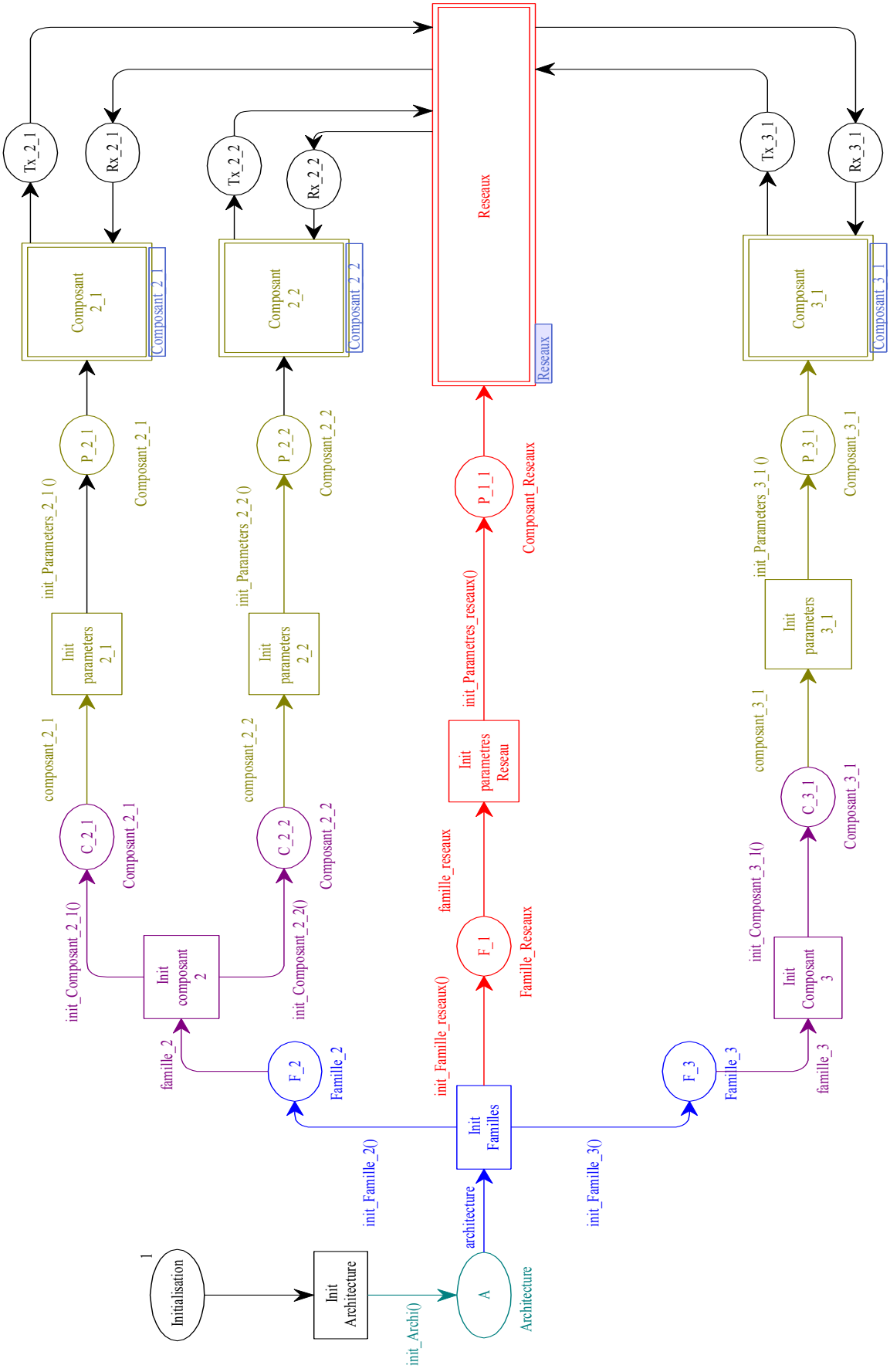


Figure 34: Exemple de modèle « *constructeur* » de topologies d'architecture

3.1.1.2 Sous-ensemble « Modèle d'instanciation et de paramétrage »

Dans le modèle « *constructeur* », trois transitions sont associées à l'instanciation de l'architecture selon les trois niveaux présentés précédemment (transitions *init_architecture*, *init_famille* et *init_composant_i_j*) et une transition est associée au paramétrage des composants (transition *init_parametres_i_j*).

Remarque

> La séquence d'instanciation de la famille « réseaux » diffère légèrement des autres familles d'équipements. En effet, compte tenu de l'hypothèse de composant unique, la transition *init_composant_reseaux* n'est plus nécessaire et la transition *init_parametres_reseaux* ne dépose qu'un seul jeton dans la place *C_r*.
> Si cette hypothèse devait être levée, le modèle « *constructeur* » devrait être modifié de la façon suivante. Une transition *init_composant_réseau* ainsi que *n* places *C_r* en sortie de cette transition devraient être ajoutées dans le modèle « constructeur » avec *n* correspondant au nombre des types de composants « réseau » présents dans la famille « réseau ». De plus, les places *Rx* et *Tx* devraient être reliées aux composants réseaux auxquels les équipements sont reliés.

L'instanciation consiste à définir le nombre de composants réel auxquels les transitions de substitutions devront faire appel. Comme nous l'avons souligné, ce type de transition n'étant lié qu'à un modèle unique de niveau hiérarchique inférieur, l'instanciation de *n* équipements d'un même type de composant *i_j* sera réalisée à l'aide *n* jetons, chacun étant associé à une instance de composant. Le paramétrage consiste à associer, à chaque jeton coloré, l'ensemble des informations relatives aux paramètres de fonctionnement d'un équipement.

Le marquage initial du modèle « constructeur » est constitué d'un jeton unique non coloré présent dans la place nommée *Initialisation*. L'instanciation et le paramétrage du réseau CPN correspondant à l'architecture, dont on souhaite évaluer les performances, repose sur :

- la définition d'un ensemble de couleurs caractérisant les architectures, les familles, les composants ainsi que leurs paramètres,
- un ensemble de fonctions d'incidence arrière permettant, depuis le franchissement de la première transition située en amont de la place *Initialisation* d'effectuer les traitements nécessaires à la transformation des couleurs selon les niveaux, la création de jetons ou encore l'affectation de valeurs aux paramètres des composants.

Les deux sections suivantes détaillent ces deux points relatifs aux couleurs et aux fonctions d'incidence arrière.

3.1.2 Définition des couleurs

La définition des ensembles de couleurs suit la hiérarchisation des équipements en architecture, famille et composants.

Convention de notation

Les ensembles de couleurs sont désignés par une lettre grecque majuscule alors que les éléments de ces ensembles (les couleurs), représentés par une variable, sont désignés par une lettre grecque minuscule. Le marquage d'une place associée à un ensemble de couleur Φ sera défini, selon les notations utilisées par (Jensen, et al., 2007), par : $x`(\varphi)$ où x est le nombre de jeton présents dans la place et φ une couleur appartenant à Φ.

Au niveau le plus bas de la hiérarchie, un ensemble de couleurs est définie pour chaque paramètre associé à un équipement. Ces couleurs élémentaires possèdent un type (comme par exemple un entier, un booléen, une chaine de caractère, …).

Définition 1

Soit $\Lambda_{i\_j\_k}$ un ensemble de couleurs caractérisant un paramètre k d'un composant i_j, de type P_k. Les éléments de l'ensemble $\Lambda_{i\_j\_k}$ sont les couleurs $\lambda_{i\_j\_k}$; ces dernières peuvent prendre toutes les valeurs admissibles, de type P_k, du paramètre k.

Remarque

Dans l'outil CPN Tools, utilisé pour le développement du prototype présenté au chapitre 4, la primitive *Colset* du langage ML permet l'association d'un type de donnée à un ensemble de couleurs.

L'ensemble de couleurs associé aux jetons présents dans la place *C_i_j* de la Figure 33 est identique à celle des jetons présents dans la place *P_i_j*. Cet ensemble est construit comme le produit cartésien entre les ensembles élémentaires de couleurs associées aux paramètres d'un composant *i_j*.

Définition 2

L'ensemble de couleurs associé à un composant *i_j*, noté Δ_{i_j}, est défini par :

$$\text{Color } \Delta_{i\_j} = \Lambda_{i\_j\_1} \times \Lambda_{i\_j\_2} \times \ldots \times \Lambda_{i\_j\_K_{ij}}$$

où K_{ij} est le nombre de paramètres du composant *i_j*.

Les éléments de cet ensemble sont les variables δ_{i_j} qui caractériseront les jetons en sortie des transitions *init_composant_i* et *init_parametres_i_j*. La variable δ_{i_j} est un K_{ij}-uplet $\{\lambda_{i\_j\_1}, \lambda_{i\_j\_2} \ldots, \lambda_{i\_j\_K_{ij}}\}$ constituée des valeurs du jeu de paramètres associés à un composant.

Remarque

Dans l'outil CPN Tools, la primitive *Record* du langage ML permet de construire un ensemble de couleurs comme un produit cartésien d'ensemble de couleurs élémentaires.

$$\text{Color } \Delta_{i\_j} = record\ \lambda_{i\_j\_1} : \Lambda_{i\_j\_1} * \lambda_{i\_j\_2} : \Lambda_{i\_j\_2} * \ldots * \lambda_{i\_j\_K_{ij}} : \Lambda_{i\_j\_K_{ij}}$$

La différence entre les jetons des places *C_i_j* et *P_i_j* est relative à l'affectation des valeurs des variables λ_k qui n'est pas encore réalisée pour la place *C_i_j*, alors qu'elle l'est pour les jetons de la place *P_i_j*.

Exemple

Le composant M580 de la famille des PLC possèdent entre autre 2 paramètres : le premier, de type entier caractérisant le type de PLC, et le second, de type entier, caractérise le temps de traitement de la logique automate. Ces paramètres sont codés dans deux ensembles de couleurs contenant deux couleurs : *20* ou *40* pour l'ensemble de couleurs *de la gamme*, *50* ou *100* pour l'ensemble de couleurs *temps de traitement*.
L'ensemble de couleurs des jetons de la place *composant_M580* est un ensemble complexe $variante \times requetes$. Un jeton présent dans cette place est donc défini par : $x`(nil: variante, nil: requetes)$.
L'ensemble de couleurs caractérisant les jetons de la place *instance_paramétrée_M580* est identique à celui des jetons de la place *composant_M580*. En revanche, les variables associées aux couleurs sont affectées : un jeton pourra par exemple être défini par : $1`(20: gamme, 50: logique\_Automate)$ ou $1`(40: gamme, 100: logic\_automate)$.

L'ensemble de couleurs associé aux familles de composants, c'est à dire caractérisant les jetons présents dans la place *F_i* de la Figure 33, est également un ensemble complexe. A la différence de l'ensemble de couleurs associé aux composants construit comme un produit cartésien de couleurs élémentaires, celui-ci est construit comme l'agrégation des ensembles de couleurs associés à chacun des composants d'une même famille. En effet, le jeton de la place *F_i* porte l'ensemble des informations relatives à chaque composant de cette famille. Il est donc défini comme une liste de K_{ij}-uplets, chaque K_{ij}-uplet étant associé un composant donné.

Définition 3

L'ensemble de couleurs associé à une famille *i*, noté Ω_i, est défini par :

$$\text{Color } \Omega_i = \Delta_{i\_1} \cup \Delta_{i\_2} \cup \ldots \cup \Delta_{i\_C_i}$$

où Δ_{i_j} est la couleur complexe associée au composant *i_j* et C_i le nombre de composants différents présents au sein de la même famille *i*.
Les éléments de l'ensemble Ω_i sont les couleurs ω_i définies comme une liste $(\delta_{i\_1}, \delta_{i\_2}, \ldots, \delta_{i\_C_i})$, de longueur C_i, constitués des couleurs associés aux composants de la famille *i*. Ces variables ω_i caractérisent les jetons en sortie de la transition *init_famille*.

Remarque

Dans l'outil CPN Tools, la primitive *Union* du langage ML permet de construire les couleurs complexes comme des unions de couleurs élémentaires.

$$\text{Color } \Omega_i = union\ \delta_{i\_1}: \Delta_{i\_1} + \delta_{i\_2}: \Delta_{i\_2} + \ldots + \delta_{i\_C_i}: \Delta_{i\_C_i}$$

Enfin, au niveau le plus haut, l'ensemble de couleur associée à l'architecture est construit comme une agrégation des couleurs caractérisant les différentes familles présentes en son sein.

Définition 4

L'ensemble de couleurs associé à une architecture, noté Γ, est définie par :

$$\text{Color } \Gamma = \Omega_1 \cup \Omega_2 \cup \ldots \cup \Omega_F$$

où Ω_i est la couleur complexe associée à la famille *i* et *F* le nombre de familles différentes présentes au sein d'une architecture.

Une variable γ, portée par le jeton en sortie de la transition nommée *init_archi*, est associée à la couleur Γ. Cette variable est une liste $(\omega_1, \omega_2, \ldots, \omega_F)$ de longueur F constitué des couleurs associées aux familles de l'architecture.

Le processus d'instanciation et de paramétrage du modèle « *constructeur* » CPN consiste à affecter les valeurs des couleurs $\lambda_{i\_j\_k}, \delta_{i\_j}, \omega_i, \gamma$ (appartenant aux ensembles$\Lambda_{i_j_k}$, $\Delta_{i\_j}, \Omega_i, \Gamma$) aux jetons franchissant les transitions nommées respectivement *init_archi*, *init_famille*, *init_composants_i*, *init_parametres_i_j et* init_parametre_réseau de la Figure 33. Ces mécanismes d'affectation sont supportés par des fonctions d'incidence arrière portées par les arcs sortant de ces quatre transitions. Elles sont décrites dans la section suivante.

3.1.3 Définitions des fonctions d'incidence arrière

L'affectation des couleurs caractérisant l'architecture, les familles, les composants et enfin les paramètres dépend bien sûr de la topologie de l'architecture dont on souhaite évaluer les performances. Nous faisons donc l'hypothèse que les informations suivantes sont connues (par exemple importées depuis un outil dédié à l'élaboration d'offre commerciale comme nous le verrons au chapitre 4) :

- les variables $\delta_{i\_j}, \omega_i, \gamma$ caractérisant une architecture « type », c'est-à-dire la déclaration des familles et des composants qui peuvent la composer,
- le nombre d'instances de chaque composant présents dans l'architecture, défini par la fonction *instance (composant_i_j)* qui associe à chaque composant une valeur entière,
- les valeurs des variables $\lambda_{i\_j\_k}$, fournies sous la forme d'une liste ordonnée des jeux de paramètres $\delta_{i\_j} = (\lambda_{i\_j\_1}, \lambda_{i\_j\_2}, \ldots, \lambda_{i\_j\_k})$ associés à chaque instance de composant. Cette liste de variables δ_{i_j} est notée *param_i_j*.

Exemple

Reprenons le modèle « *constructeur* » de la Figure 34 pour lequel nous supposons que chaque composant est caractérisé par 2 paramètres à valeur entière. Considérons une architecture, dont on souhaite générer le modèle, constituée d'une instance du *composant_1_1* (composant réseau) – dont les paramètres ont pour valeur (a_1, b_1) – et de deux instances du *composants 2_1* – dont les paramètres ont pour valeur (x_1, y_1) et (x_2, y_2). Aucune instance du *composant_2_2* ni aucune instance de composant de la *famille_3* ne sont utilisées dans l'architecture.

Ces données, supposées connues et utilisées comme paramètres d'entrée des fonctions, correspondent aux variables suivantes :

Variables associées aux couleurs

$\gamma = (\omega_1, \omega_2, nil)$, $\omega_1 = (\delta_{1\_1})$, $\omega_2 = (\delta_{2\_1}, nil)$,

$\delta_{1\_1} = \{\lambda_{1\_1\_1}, \lambda_{1\_1\_2}\}$, $\delta_{2\_1} = \{\lambda_{2\_1\_1}, \lambda_{2\_1\_2}\}$,

Déclaration des instances

instance(composant_1_1) = 1, *instance(composant_2_1)* = 2,

instance(composant_2_2) = instance(composant_3_1) = 0

Paramètres

$param\_1\_1 = \{\lambda_{1\_1\_1} = a_1, \lambda_{1\_1\_2} = b_1\}$

$param\_2\_1 = (\{\lambda_{2\_1\_1} = x_1, \lambda_{2\_1\_2} = y_1\}, \{\lambda_{2\_1\_1} = x_2, \lambda_{2\_1\_2} = y_2\})$

3.1.3.1 Fonction init_architecture ()

Cette fonction est placée sur l'arc sortant de la transition *init_archi*. Cette transition a pour place d'entrée une place unique – la place *initialisation* – et pour place de sortie une place unique – la place *architecture*. Dans le marquage initial, la place *initialisation* contient un jeton unique non coloré. Le franchissement de la transition *init_archi* provoque le dépôt d'un jeton coloré unique dans la place *architecture* en lui affectant la couleur γ.

Définition 5

La fonction *init_archi()* est définie par :

$$init\_archi\ () = 1`(\gamma : \Gamma) = 1`(\omega_1 : \Omega_1, \omega_2 : \Omega_2, \dots, \omega_F : \Omega_F)$$

3.1.3.2 Fonction init_famille_i ()

La fonction *init_famille_i()* est associée à chaque arc sortant de la transition *init_famille*. Cette transition a, pour place d'entrée, une place unique contenant un jeton unique de couleur Γ (place *architecture*) et plusieurs places avales *F_i* correspondant à l'ensemble des familles proposées par le modèle « *constructeur* ». Pour rappel, la Figure 33 et la Figure 34 adoptent la convention de notation suivante : la famille *F_1* est associée à la couleur *Famille_Réseaux*.

Le rôle des fonctions *init_famille_i()* est, d'une part, d'extraire de l'ensemble de couleurs Γ les composantes Ω_i caractérisant la famille *i* et, d'autre part, d'affecter des couleurs δ_{i_j} au jeton unique déposé dans la place *F_i*. Ces couleurs δ_{i_j} définissent l'ensemble des types de composants présents au sein de la famille. Cette fonction correspond donc à une fonction de projection.

Définition 6

La fonction *init_famille_i()* est définie par :

$init\_famille\_i\ () = 1`(proj_i(\omega_1, \omega_2, \dots, \omega_F)) = 1`(\delta_{i\_1} : \Delta_1, \delta_{i\_2} : \Delta_2, \dots, \delta_{i\_C_i} : \Delta_{i\_C_i})$ où $proj_i$ est la fonction de projection sur le i[ème] terme.

Remarque

Si une *famille_i*, proposée dans le modèle « *constructeur* », n'est pas utilisée dans une architecture donnée soumise à évaluation, alors la variable ω_i qui lui est associée sera égale à *nil*. En conséquence, l'exécution de la fonction *init_famille_i()* aura pour effet de déposer dans la place *F_i* un jeton caractérisé une liste vide qui ne validera pas le franchissement des transitions placées en aval de cette place. En d'autres termes, cela revient à forcer le marquage de la place *F_i* à zéro.

3.1.3.3 Fonction init_composant_i_j ()

La fonction *init_composant_i_j()* est associée à chaque arc sortant de la transition *init_composant_i*. Cette transition a, pour place d'entrée, une place unique contenant un jeton unique de couleur ω_i (place *F_i*). Cette transition possède plusieurs places avales *C_i_j* correspondant à l'ensemble des composants de cette famille qui sont proposés par le modèle « *constructeur* ».

Deux rôles sont associés aux fonctions *init_composant_i_j()* :

- le premier est de déposer dans la place avale *C_i_j* autant de jetons qu'il y a d'instances du composant *i_j* ; en d'autres termes, cela revient à affecter un poids variable et égal à *instance(composant_i_j)* à l'arc reliant la transition *init_composant_i_j* et la place *C_i_j* ;
- le second est comparable à celui des fonctions *init_famille_i()* ; il s'agit pour chaque jeton déposé dans la place *C_i_j*, d'une part, d'extraire de la couleur complexe ω_i les couleurs δ_{i_j} caractérisant les composants *i_j* et, d'autre part, de les affecter aux jetons déposés dans la place *C_i_j*. Toutefois, les valeurs des variables $\lambda_{i\_j\_k}$, qui composent une couleur δ_{i_j}, ne sont pas affectées à cette étape. Celle-ci se limite à préparer la structure des jetons associés à chaque instance de composants. Selon les mêmes principes que la fonction *init_famille_i()*, cette opération est réalisée à l'aide d'une fonction de projection.

Définition 7

La fonction *init_composant_i_j()* est définie de la manière suivante :

init_composant_i_j()

$$= instance(composant\_i\_j)\text{`}(proj_{ij}(\delta_{i\_1}, \delta_{i\_2}, \dots, \delta_{i\_C_i}))$$

$$= instance(composant\_i\_j)\text{`}(\lambda_{i\_j\_1}: \Lambda_{i\_j\_1}, \lambda_{i\_j\_2}: \Lambda_{i\_j\_2}, \dots, \lambda_{i\_j\_k}: \Lambda_{i\_j\_K_{ij}})$$

où $proj_{ij}$ est la fonction de projection sur le $ij^{ème}$ terme.

Remarques

a) L'affectation des valeurs des paramètres n'étant pas réalisée à cette étape, les variables $\lambda_{i_j_k}$seront affectées de la valeur *nil*.

b) Pour la famille « réseaux », la transition *init_composant_1* (ou *init_composant_réseaux*) n'existe pas dans le modèle « *constructeur* » dans la mesure où nous n'avons considéré qu'un composant unique, le commutateur. Par

conséquent, la fonction *init_composant_1* (ou *init_composant_réseaux*) n'est pas définie.

3.1.3.4 Fonction init_parametres_i_j ()

La fonction *init_parametres_i_j()* est associée à l'arc sortant de la transition '*Init_parametres_i_j'*. Cette transition a, pour place d'entrée, une place unique *C_i_j* de couleur (*composant_i_j*) contenant autant de jetons, de couleur δ_{i_j}, qu'il y a d'instances de composants *i_j* dans l'architecture à évaluer. Cette transition possède une place de sortie unique nommée *P_i_j* qui est la place d'entrée de la transition de substitution décrivant le comportement du *composant i_j*.

Le rôle de cette place est d'affecter, à chaque jeton déposé dans la place *P_i_j*, les variables $\lambda_{i\_j\_k}$ avec les valeurs extraites de la liste de jeux de paramètres *param_i_j*. Conformément aux définitions précédentes, cette liste est de longueur *instance(composant_i_j)*.

La fonction *init_parametres_i_j()* est exécutée à chaque franchissement de la transition *init_parametres*, c'est à dire autant de fois qu'il y a de jetons dans la place *C_i_j* et donc d'instances du composant *i_j*.

Définition 8

La fonction *init_parametres_i_j()* est définie de la manière suivante :

$$init\_parametres\_i\_j() = 1`(param\_i\_j :: \delta_{i\_j})$$

où $\delta_{i\_j} = (\lambda_{i\_j\_1}, \lambda_{i\_j\_2}, \dots, \lambda_{i\_j\_k})$ et *liste* :: *item* désigne la fonction d'extraction (et de suppression) du dernier élément de la liste *liste*.

Remarque

Dans le cas particulier de la famille réseau, *param_1_1* est une liste composée d'un seul élément (puisque nous avons fait l'hypothèse d'une seule instance de commutateur), la valeur $\lambda_{1\_1\_1}$ sera définie comme une liste de couples composée de l'identifiant unique d'un composant et du port de communication auquel il est associé. Ce point sera plus amplement détaillé dans la section 3.2.2.

3.1.3.5 Scénario d'instanciation et de paramétrage

Afin d'illustrer les définitions des couleurs et des fonctions d'instanciation et de paramétrage, considérons le scénario suivant, construit sur le modèle « *constructeur* » de la Figure 34 et les jeux de données proposés par l'exemple cité en début de section 3.1.3 de ce chapitre :

a) Marquage initial : un jeton non coloré est contenu dans la place *initialisation* (Figure 35-a).
b) Franchissement de la transition *init_archi* : il provoque le dépôt d'un jeton de couleur $\gamma = (\omega_1, \omega_2, nil)$ dans la place *architecture* (Figure 35-b). Ce jeton porte donc

l'ensemble des informations relatives aux familles présentes dans l'architecture dont on cherche à générer le modèle.

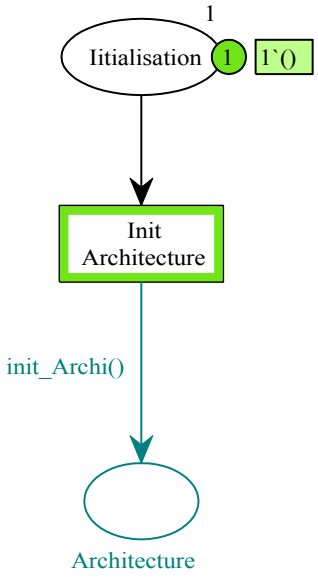
c) Franchissement de la transition *init_familles* : un jeton, de couleur $\omega_1 = (\delta_{1\_1})$ est déposé dans la place *F_1 (famille_reseaux)* et un jeton de couleur $\omega_2 = (\delta_{2\_1}, nil)$ est déposé dans la place *F_2* (Figure 35-c). Aucun jeton n'est déposé dans la place *F_3*.

d) Franchissement des transitions *init_composant_i* : la transition *init_composant_2* provoque le dépôt de deux jetons de couleur $\delta_{2\_1} = \{\lambda_{2\_1\_1} = nil, \lambda_{2\_1\_2} = nil\}$ dans la place *C_2_1* alors qu'aucun jeton n'est déposé dans la place *C_2_2* (Figure 35-d). La transition *init_composant_3* n'est pas franchissable. Dans cet exemple la variable $\lambda_{i\_j\_k}$ est nommée parametre_i_j_k.

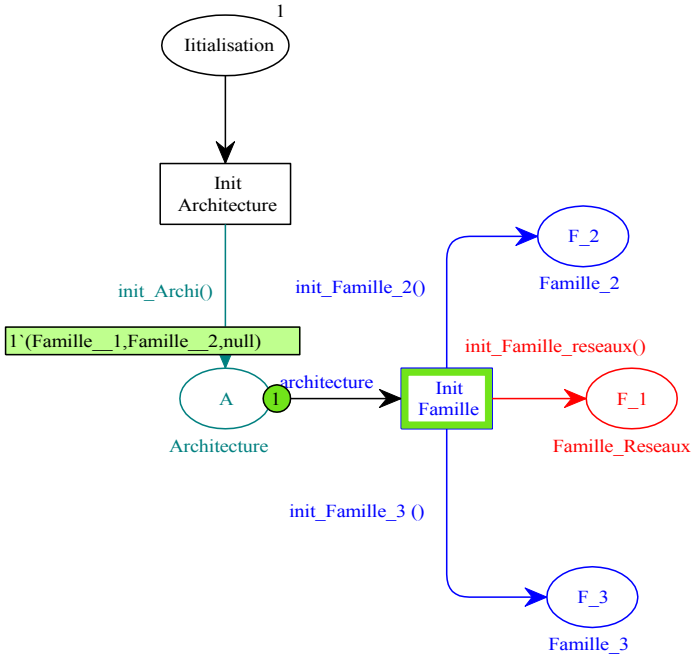
e) Franchissement des transitions *init_parametres_i_j* : la transition *init_parametres_1_1* provoque le dépôt d'un jeton de couleur $\{\lambda_{1\_1\_1} = a_1, \lambda_{1\_1\_2} = b_1\}$ dans la place *P_1_1* tandis que la transition *init_parametres_2_1* dépose deux jetons de couleur respective $\{\lambda_{2\_1\_1} = x_1, \lambda_{2\_1\_2} = y_1\}$ et $\{\lambda_{2\_1\_1} = x_2, \lambda_{2\_1\_2} = y_2\}$ dans la place *P_2_1*. Les transitions *init_parametres_3_1* et *init_parametres_2_2* ne sont pas franchissables, il n'y a donc pas de composant de ce type dans l'architecture à évaluer (Figure 35-e).

Remarques

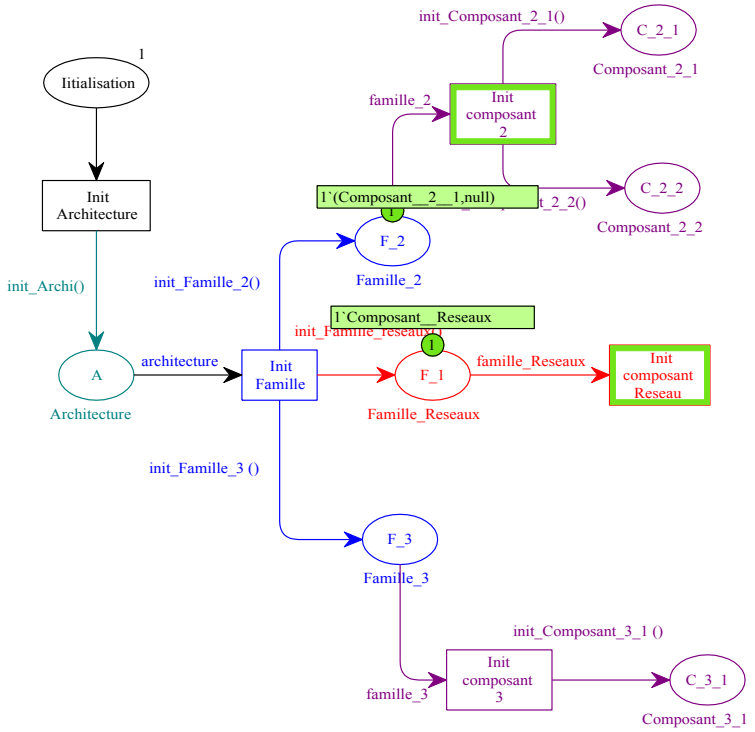
> Les franchissements des transitions décrites dans le scénario ci-dessus s'effectuent à temps nul. Cette propriété est exploitée pour s'assurer que le cycle d'instanciation et de paramétrage soit terminé – c'est à dire que toutes ses transitions validées soient tirées – avant toute évolution des modèles génériques de composant associés aux transitions de substitutions. La solution adoptée est d'associer aux premières transitions des modèles génériques de composant une temporisation minimale d'une unité de temps.



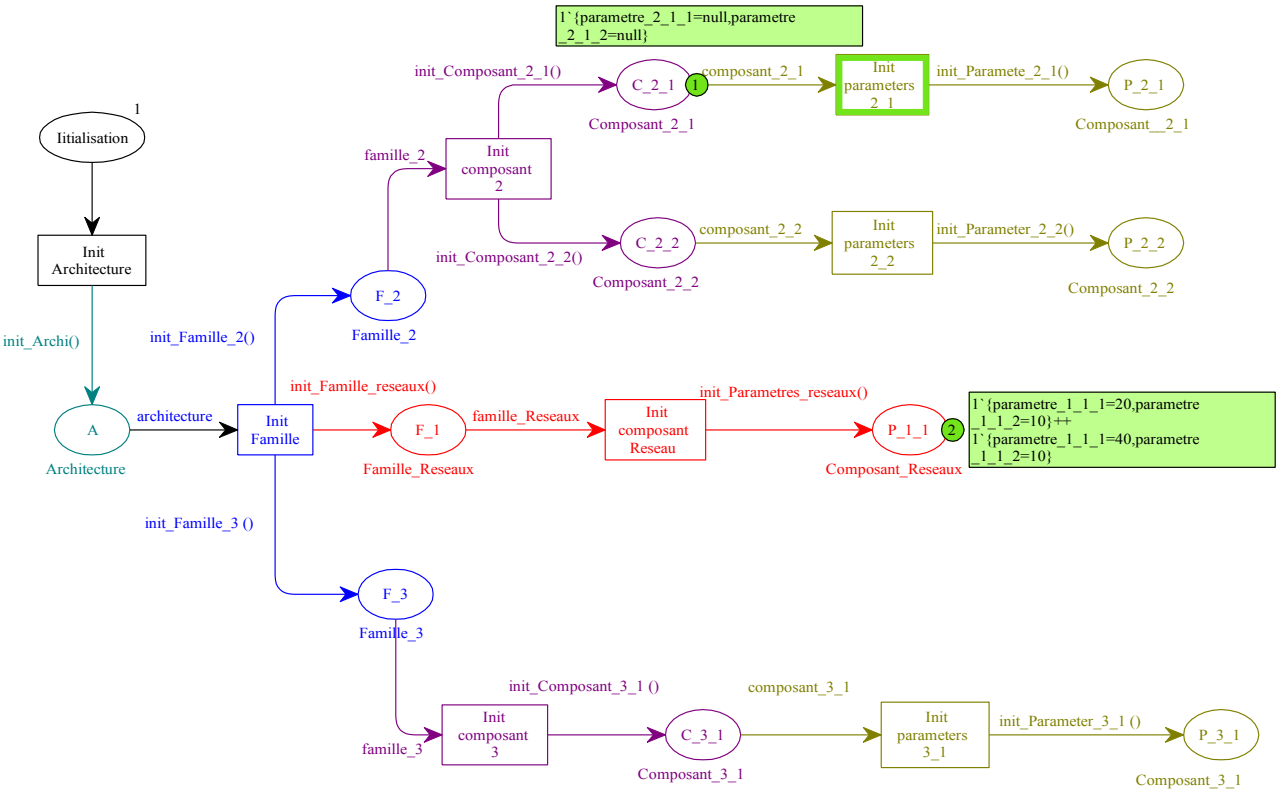
(a) Marquage initial



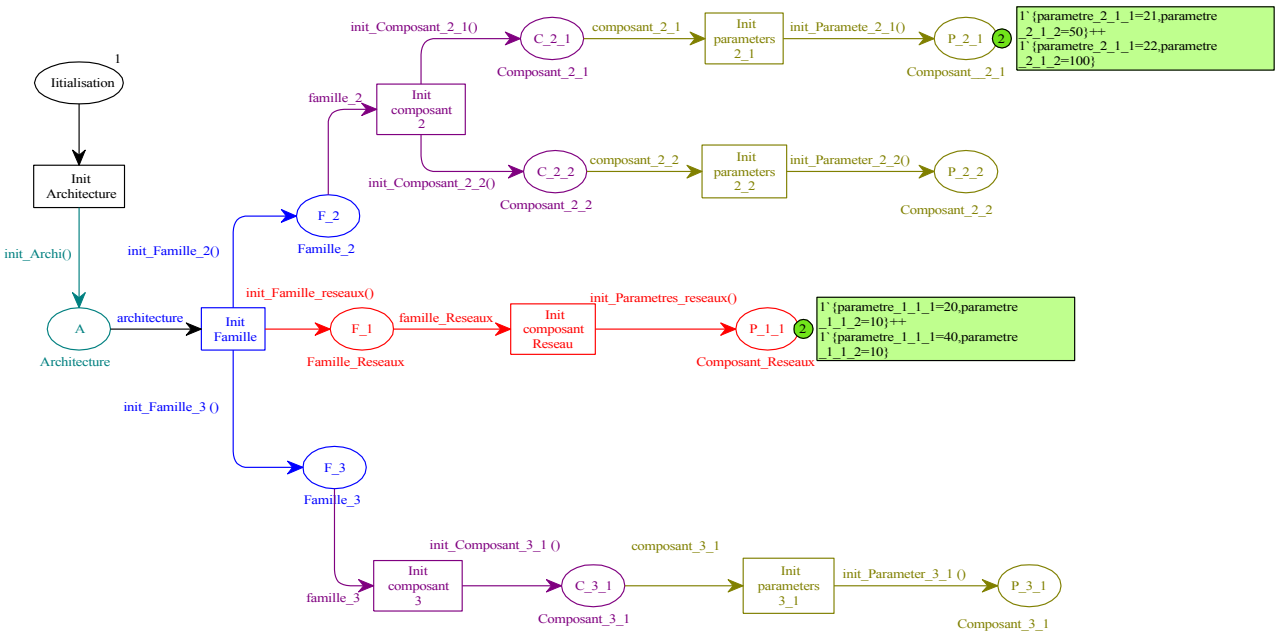
(b) Transition *init_archi*



(c) Transition *init_famille*



(d) Transitions *init_composant_i*



(e) Transitions *init_parametres_i_j*

Figure 35: Scénario d'instanciation et de paramétrage d'une architecture

3.2 Modélisation générique des composants d'une famille

La structuration hiérarchique en architecture, familles et composants proposées pour le modèle « *constructeur* » repose sur une hypothèse forte qui établit que toutes les instances d'un même composant peuvent être caractérisées par un comportement similaire (par exemple en termes de services assurés, de modes de fonctionnement, …) même si elles peuvent différer de par leurs caractéristiques techniques (par exemple, capacités de traitement, performances temporelles, …). En conséquence, nous avons fait l'hypothèse qu'un composant pouvait être décrit par un modèle générique, instanciables et paramétrable.

Ces modèles génériques de composants sont représentés dans le modèle « *constructeur* » (Figure 33 et Figure 34) par les transitions de substitutions (transitions *composant_i_j*) qui définissent un lien hiérarchique vers un modèle CPN. L'instanciation et le paramétrage de ces modèles sont réalisés à l'aide de la place *P_i_j*, située en amont de chaque transition de substitution, qui contient :

- un nombre de jetons *n* égal au nombre d'instances de composants présents dans l'architecture à évaluer ; une couleur identifiera chaque instance de composant dans le modèle générique de composant ; celui-ci aura donc un comportement équivalent, par une opération de dépliage, à *n* modèles génériques,
- le jeu de paramètres qui caractérise le fonctionnement de chaque instance de composant et qui est porté par les couleurs associées aux jetons (variables $\lambda_{i\_j\_k}$ définies dans les sections précédentes).

Les paramètres des composants peuvent se décomposer en deux grandes familles :

- les paramètres associés à des caractéristiques techniques intrinsèques à chacun des composants qui traduisent essentiellement les capacités du composant (par exemple, une taille mémoire, un nombre de requêtes traitées par cycle, …) ;
- les paramètres associés à l'intégration du composant au sein d'une architecture donnée (par exemple, les flux de messages émis ou reçus par le composant, ses liens de connexion avec les autres éléments de l'architecture, …).

Comme nous l'avons indiqué précédemment, les composants seront traités de manière différente selon qu'ils appartiennent à une famille d'équipements supportant un service d'automatisation ou à une famille d'équipements de communication. Les deux sections suivantes abordent la présentation des modèles génériques associés à ces deux classes.

3.2.1 Modèles génériques pour un composant d'automatisation

Chaque modèle générique est défini de manière structurée et hiérarchique à l'aide de 2 niveaux : un modèle comportemental décrivant les modes de fonctionnement intrinsèques des composants, un modèle d'intégration définissant les interfaces standard d'un composant au sein d'une architecture. L'ensemble des modèles génériques est stocké en bibliothèque.

3.2.1.1 Modèle comportemental

Le modèle comportemental représente le fonctionnement général d'un composant. Il décrit les différentes modalités des services proposés (par exemple pour un automate programmable, les modes I/O scanning ou messagerie), l'enchainement des traitements réalisés, les conditions dans lesquelles ceux-ci peuvent s'opérer ou encore les résultats qu'ils produisent.

Un modèle comportemental sera dit générique dans la mesure où il permettra la description de composants dont les principes de fonctionnement général sont similaires même si leurs caractéristiques techniques peuvent être légèrement différentes. A titre d'exemple, dans la gamme Schneider Electric, les automates M340 ou M580 ont des fonctionnements identiques et ne diffèrent que par leur capacité (par exemple mémoire CPU, nombre de requêtes pouvant être traitées en un cycle, …).

Pour prendre en compte ces variations techniques de gamme, un modèle comportemental générique est paramétrable. L'ensemble de ces paramètres est traduit en termes de couleurs associées aux jetons de paramétrage. Chaque jeu de paramètres étant spécifique à un type de composant, nous ne dresserons pas ici une liste exhaustive des paramètres associés à l'ensemble des modèles comportements développés dans le cadre de cette étude et stockés en bibliothèque.

En termes de modélisation, la spécificité de chaque modèle comportemental pour un type de composant donné a pour conséquence que leur élaboration, contrairement au modèle « *constructeur* », nécessite l'intervention d'un expert. Si l'offre « catalogue » s'enrichit d'un nouveau composant dont le comportement ne peut être instancié à partir de ceux stockés en bibliothèque, il sera nécessaire de procéder à une modélisation CPN de ce nouveau comportement à l'aide d'un expert.

Dans le contexte avant-vente, les modèles comportementaux des composants contiendront des incertitudes relatives notamment à leurs performances. En effet, celles-ci peuvent être sujettes au contexte d'utilisation du composant (notamment de sa charge). Certaines places des modèles comportementaux seront donc P-temporisées avec une durée de temporisation déterminée par une distribution de probabilité dépendant de la nature de l'incertitude. A titre d'exemple, le temps de traitement d'une requête par le processeur d'un automate programmable sera déterminé à l'aide d'une loi uniforme.

Enfin, pour des motifs relatifs à la lisibilité, la maintenabilité et la réutilisabilité des modèles, les modèles génériques pourront éventuellement être structurés hiérarchiquement à l'aide de transitions de substitution encapsulant différents niveaux de modélisation et d'abstraction.

Ces modèles étant spécifiques pour chaque composant, ils ne seront pas décrits de manière plus détaillées dans le cadre de ce chapitre, mais plutôt dans le chapitre suivant.

3.2.1.2 Modèle d'intégration

Le modèle comportemental d'un composant est associé à une transition de substitution nommée *comportement_composant* qui permet son encapsulation au sein du modèle d'intégration présenté par la Figure 36. Celui-ci introduit trois éléments nouveaux :

- la génération de flux de messages émanant du composant,
- la définition d'une interface standard de communication,
- la définition des points d'observation requis pour l'évaluation des performances.

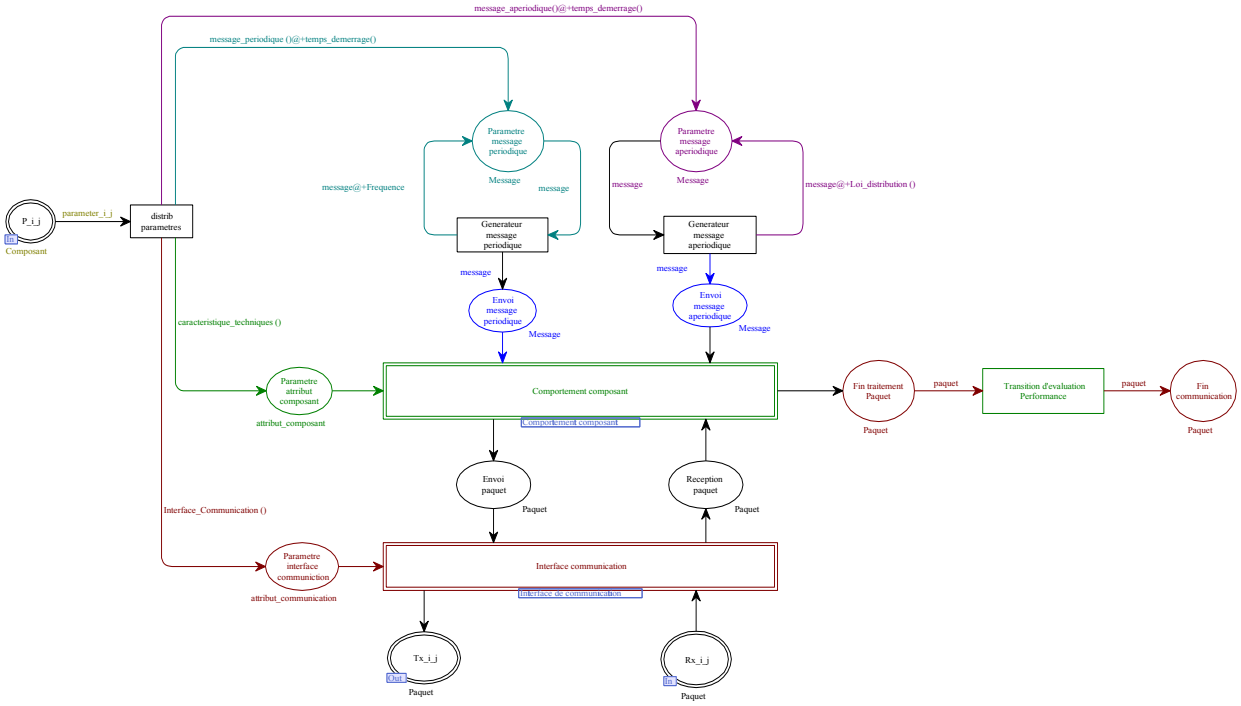


Figure 36: Modèle d'intégration d'un composant d'automatisation

La fonction *caracteristique_techniques ()* affecte les valeurs des caractéristiques techniques des composants à travers un jeton déposé dans la place nommée

Parametre_attribut_composant en entrée de la transition de substitution nommée *comportement_composant*. Ces paramètres sont exploités dans les modèles comportementaux des équipements.

Les places *Rx_i_j* et *Tx_i_j* sont les mêmes que sur le modèle « *constructeur* » (RdP hiérarchique selon (Jensen, et al., 2007))

3.2.1.2.1 Génération des flux de messages

Dans le cadre de ce mémoire, tous les composants sont considérés comme des clients ou des serveurs : un client émet des messages à destination d'un serveur qui les reçoit, les traite et envoie une réponse au client initiateur de la requête. En mode client, l'émission de messages peut être réalisée selon deux modes :

- un mode périodique où les dates d'émission des messages suivent une temporisation périodique ; ce mode est utilisé pour l'exécution périodique de services tels que la mise à jours et le rafraichissement des valeurs du procédé à partir du système d'acquisition et de commande SCADA ;
- un mode apériodique où les émissions des messages peuvent survenir à une date quelconque donc aléatoire; ce mode est utilisé dans le cas de services apériodiques tels qu'une sollicitation émanant d'un opérateur de conduite ou encore une acquisition de variable lors du changement d'état d'un capteur.

Ces flux de messages sont caractérisés par un ensemble de couleurs, permettant d'identifier notamment les typologies de flux et leurs destinataires, associé aux places nommées *parametre_message_apériodique*, *envoi_message_apériodique*, *envoi_message_périodique* et *parametre_message_périodique*.

Définition 9

L'ensemble de couleurs associé à un jeton de couleur *message*, est définie par :

$$\text{Color } message = source \times Destination \times typeMessage \times interArrivee$$

où :

- *source* est un ensemble de couleurs (de type entier) caractérisant l'identifiant unique de l'émetteur du message,
- *destination* est un ensemble de couleurs (de type entier) caractérisant l'identifiant unique du destinateur du message,
- *typeMessage* est un ensemble de couleur (de type énumération) caractérisant le type de message envoyé
- *interArrivee* est un ensemble de couleurs (de type entier) qui définit la durée entre l'émission de deux messages. Cette durée peut être fixe (communication périodique) ou aléatoire (communication apériodique).

Exemple

Les composants de type SCADA génèrent de manière périodique 3 types de message (Alarms, Display, Trend) à destination des automates afin de récupérer les différents états du système. Ces messages sont envoyés suivant des périodes différentes en fonction

de la criticité du groupe. Par exemple les messages du groupe *Alarms* dont le rôle est de remonter au niveau du SCADA les alarmes et les défaillances du système sont envoyés plus fréquemment que ceux du *Display* dont le rôle est de fournir le statut d'un équipement (*On*, *Off*, *Running*, etc…).

Deux fonctions de projection associées respectivement à deux des arcs de sortie de la transition *distrib_parametres* permettent d'extraire les composantes des ensembles de couleurs relatifs à la génération de messages périodiques et la génération de messages apériodiques.

La fonction *message_periodique ()* extrait, des ensembles de couleurs des jetons présents dans la place *parametrage_composant*, les paramètres requis pour une émission de messages en mode périodique, notamment la fréquence d'émission, et les associe à un jeton déposé dans la place *parametre_message_periodique* caractérisé par l'ensemble de couleurs *message.*

La fonction *message_apériodique()* extrait, des ensembles de couleurs des jetons présents dans la place *parametrage_composant*, les paramètres requis pour une émission de messages en mode apériodique, et les associe à un jeton déposé dans la place *parametre_message_apériodique* caractérisé par l'ensemble de couleurs *message*. Dans la mesure où le niveau de connaissance disponible en phase d'avant-vente reste faible, ces paramètres seront donnés sous la forme d'une loi de distribution. Enfin, si le composant ne propose qu'un fonctionnement périodique en mode serveur, cela signifie que les couleurs du jeton de paramétrage associées au trafic apériodique ont une valeur *nil*. Dans ce cas, la fonction *message_apériodique()* ne déposera aucun jeton dans la place *générateur_message_apériodique*, ce qui aura pour effet d'inhiber ce type d'émission.

Ces deux fonctions, *message_periodique()* et *message_aperiodique(),* ont un rôle supplémentaire : elles incluent une fonction de temporisation notée *temps_demarrage()*, dont la durée est aléatoire, ayant pour effet de retarder la date de disponibilité des jetons déposés dans les places avales *générateur_message_apériodique, générateur_message_apériodique*, avec un double objectif :

- il s'agit, d'une part, de s'assurer que l'ensemble des transitions du modèle « *constructeur* », franchies à la date $t = 0$, aient bien été franchies avant de procéder à l'initialisation du composant,
- d'autre part, cette temporisation permet de garantir la désynchronisation de l'ensemble des messages émis, qu'ils soient périodiques ou apériodiques, et d'éviter ainsi l'émission simultanée de tous ces messages à la date $t = 0$.

Le tir des transitions *générateur_message_périodique* ou *générateur_message_apériodique* a un double effet. D'une part, un jeton est déposé dans les places *envoi_message_periodique* ou *envoi_message_aperiodique* ; il représente l'émission d'un message.

D'autre part, un jeton est redéposé dans les places *paramètre_message_périodique* ou *paramètre_message_apériodique*, situées en amont de ces transitions ; ce jeton est destiné à permettre la génération de messages ultérieurs lors d'un nouveau tir des transitions *générateur_message_periodique* ou *générateur_message_apériodique*. Afin de respecter les fréquences d'émission, ces deux places doivent donc être temporisées.

En accord avec la notation de Jensen (voir chapitre 2), les paramètres de temporisation sont portés par les arcs entrant sur ces places depuis les transitions *générateur_message_périodique* et *générateur_message_apériodique* selon les principes suivants :

- en mode périodique, l'émission de messages est cadencée par l'introduction d'une temporisation à durée fixe correspondant à la période d'émission ; cette temporisation est définie par la primitive *message@+Frequence* portée par l'arc reliant la transition *generateur_message_periodique* et la place *paramètre_message_periodique* ; cette primitive a pour effet de ne rendre disponible un jeton déposé dans cette place qu'à la date correspondant à la date de dépôt incrémentée de la fréquence. La valeur de *Frequence* découle de la valeur de la couleur *interArrivee* contenue dans le jeton *message* ;
- en mode apériodique, les principes restent les mêmes en remplaçant la durée fixe par une durée dont les valeurs est obtenue par un tirage aléatoire selon une distribution de probabilité donnée sous la forme : *generation_message@+Loi_distribution()*. La valeur de la distribution de probabilité *Loi_distribution()* est contenue dans la couleur *interArrivée* contenue dans le jeton message.

Les jetons ainsi générés et déposés dans les places *envoi_message_periodique* et *envoi_message_aperiodique* sont ensuite traités par le modèle comportemental.

3.2.1.2.2 Interface de sortie du modèle comportemental

Les jetons sortant du modèle comportemental représentent les messages de communication à transmettre sur le réseau. Ils sont déposés dans les places non bornées *envoi_paquet* et *reception_paquet*. Par rapport aux jetons de couleur *message* présents dans les places en entrée du modèle comportemental, les jetons déposés dans les places *envoi_paquet* et *reception_paquet* subissent une transformation de couleur ayant essentiellement pour but de définir le type de service requis en fonction du type de message, d'en déduire une taille de paquet dont la valeur a un impact sur les performances, d'associer un numéro de séquence afin d'identifier le paquet lorsque celui-ci transitera sur le réseau (ou en cas de perte) et enfin, d'associer au jeton une information permettant son horodatage.

Les places *envoi_paquet* et *reception_paquet* seront ainsi caractérisées par un ensemble de couleurs *paquet* défini comme suit.

Définition 10

L'ensemble de couleur *paquet* est défini de la manière suivante :

$$\text{Color } paquet = source \times Destination \times typeService \times sequence \times taille \times TimeStamp$$

où :

- *Source et destination* définissent les identifiants uniques du composant émetteur et du composant récepteur découlant du jeton de couleur *message*.
- *typeService* est le type de service demandé par l'émetteur du message ; il résulte du type de message émis.

- *Séquence* définit le numéro de séquence du paquet afin de permettre son identification ultérieure (notamment lorsqu'il transitera sur le réseau).
- *Taille* définit la taille du paquet, la valeur de celle-ci dépend du type de service.
- *TimeStamp* définit l'heure et la date auquel le paquet a été émis.

Exemple

Les messages envoyés par le SCADA correspondent à des services fournis par les automates programmables. Lors de l'envoi du paquet vers le réseau, le modèle comportemental du SCADA transforme l'information relative au type de message (par exemple *Alarm*, *Display*, *Trend*) vers un type de service requis (par exemple *Messaging fast* ou *Messaging Low*). La table de conversion est fixe et définie pour chaque composant. Par exemple, les messages de type « *Alarm* » ou « *Display* » correspondent à un type de service « *Messaging Fast* » alors que les messages de type « *Trend* » seront associés à un type de service « *Messaging Low* ».

Remarque

La couleur horodatage est un attribut nécessaire pour calculer la durée entre une date d'émission et une date de réception, et qui sera donc très utile pour l'évaluation des temps de réponse. La valeur de celle-ci est obtenue grâce à la fonction *Time(),* qui une fonction native dans (Jensen, et al., 2007).

3.2.1.2.3 Interface de communication

L'interface de communication est représentée par une transition de substitution nommée *interface_communication.* Celle-ci est spécifique à chaque composant et nécessite une modélisation à l'aide d'un expert. Elle possède deux places d'entrée *envoi_paquet* et *Rx* ainsi que deux places de sortie *reception_paquet* et *Tx*. Les ensembles de couleurs associés à ces quatre places sont identiques et conformes à la *définition 10*. Cette transition de substitution a essentiellement trois rôles.

Il s'agit, d'une part, de distribuer les paquets émis ou reçus sur les différents dispositifs de communication internes et spécifiques au composant (pour un composant ayant plusieurs ports de communication). Les informations permettant cette distribution sont fournies par la fonction d'incidence arrière *Interface_Communication()* qui rassemble les paramètres permettant l'aiguillage des jetons entre les ports internes du composant. Cette transition de substitution n'est pas détaillée de manière plus approfondie car spécifique à chaque composant. Cependant, le chapitre 4 en donne un exemple pour un composant de type automate programmable (PLC).

Il s'agit, d'autre part, de fournir les services requis pour la transmission sur/depuis les ports du commutateur, notamment en intégrant l'ordonnancement des transmissions à l'aide de buffers. Les ensembles de couleurs associés à la place représentant un buffer sont de type Liste en accord avec les définitions de (Jensen, et al., 2007). A titre d'exemple, la Figure 37 présente une transition de substitution décrivant un comportement de type FIFO pour un buffer de sortie. Dans cette figure la place nommée '*Buffer FIFO'* est associée à une couleur nommée *'BUFFER'* de type liste (primitive CPN ML) dont la variable associée à celle-ci est nommée *'buffer'*. Le franchissement de la transition nommée *'entrée Buffer'* induit à l'ajout d'un jeton

de type paquet dans la queue de la liste grâce à la primitive *buffer^^[paquet]*. Le franchissement de la transition '*Sortie Buffer'* induit à l'extraction du premier élément de la liste grâce à la primitive *paquet::Buffer*.

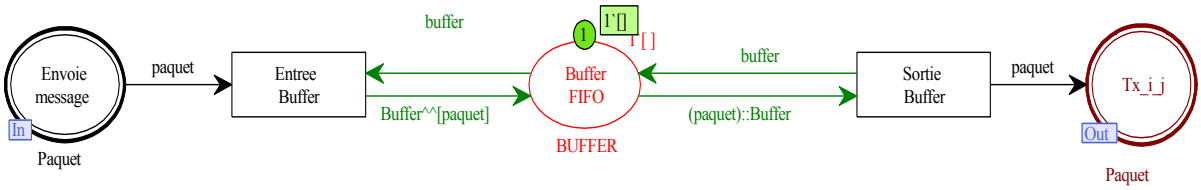


Figure 37: Modèle buffer FIFO

Enfin, il s'agit de jouer un rôle de passerelle entre différents réseaux si et seulement si l'émetteur et le destinataire se trouvent sur deux réseaux différents et que le composant est connecté à ces deux réseaux. Les informations nécessaires sont obtenues à partir de la fonction *interface_communication ()* (voir Figure 36). Le comportement associé à cette fonction de passerelle sera identique à celui du commutateur décrit en section 3.3.2, notamment en ce qui concerne la fabrique de commutation, et ne sera donc pas détaillé dans la présente section.

3.2.1.2.4 Points d'observation pour l'évaluation de performance

L'évaluation de performance à l'aide d'un réseau de Petri repose sur la définition d'observateurs permettant d'enregistrer le comportement du réseau au cours du temps en termes de marquage des places ou de franchissement des transitions.

Dans l'optique de génération automatique du réseau utilisé pour une telle évaluation de performance et, en particulier, la définition et le placement des observateurs requis, nous avons choisi d'introduire, par défaut, une transition et une place caractérisant l'activité du composant et sur lesquels des observateurs pourront être ultérieurement si besoin placés. Ces « *réceptacles* » pour observateurs seront (ou ne seront pas) utilisés en fonction des performances à évaluer. La place ainsi introduite est notée sur la Figure 36 *fin_traitement_message* et la transition est notée *transition_evaluation_performance*. Le modèle comportemental du composant (transition de substitution) déposera un jeton dans la place *fin_traitement_message* lorsqu'il aura terminé le traitement d'un message reçu à partir de la place *reception_paquet* (lorsque le composant est en mode client donc reçoit des réponses à des requêtes), et/ou lorsqu'il aura terminé le traitement d'un message reçu à partir de la place *reception_paquet,* puis déposé dans la place *emission_paquet* un jeton de type message (lorsque le composant est en mode serveur donc fourni une réponse à des requêtes).

La section 4 de ce chapitre détaille la définition des observateurs et leur placement sur les composants de l'architecture.

3.2.2 Modèle du composant réseau

Le modèle du composant réseau se base sur une modélisation d'un commutateur réseau dont le nombre de port correspond au nombre de composants génériques dont dispose le modèle « *constructeur* ». Le rôle de ce composant est d'aiguiller (commutation) l'ensemble des jetons

envoyés/reçus vers les interfaces de communication des composants génériques. Le modèle du composant réseau est englobé au sein de la transition de substitution réseaux dans le modèle « *constructeur* » (Figure 34).

3.2.2.1 Vue d'ensemble

Le modèle général en RdP du composant réseau peut être défini comme l'indique la Figure 38. Celle-ci représente un composant réseau ne comportant qu'un seul port de connexion associé au $j^{ème}$ composant appartenant à l'$i^{ème}$ famille de composant. Ce port est représenté par une transition de substitution. Celle-ci est connectée avec les deux ports de transmission et de réception de ce composant respectivement *Tx_i_j* et *Rx_i_j* ainsi qu'en sortie à une place *'buffer commutateur'* qui constitue le buffer interne d'un commutateur où les jetons sont stockés avant d'être commutés vers le port approprié. La place *'table de commutation'* contient l'ensemble de jetons permettant la commutation. Ceux-ci sont générés et paramétrés lors du franchissement de la transition *'init table_commutation'*.

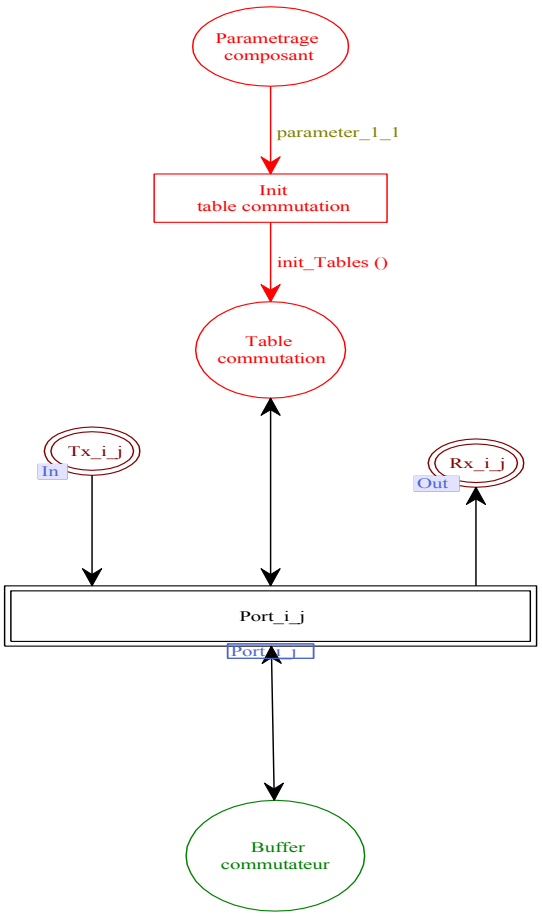


Figure 38: Définition générale d'un commutateur réseau

Cette création de jetons se fait à travers la fonction *init_Tables ()* qui récupère sur la place nommée *'Parametrage composant'* l'ensemble des informations liées à la topologie de l'architecture. Les jetons contenus sur cette place découle de la fonction d'instance arrière *'init_Parametres_reseaux()'* présente sur le modèle de constructeur de la Figure 33 dont le rôle est le paramétrage du composant réseau.

Les jetons résultants générés par la fonction *init_Tables ()* sont composés de l'identifiant du $j^{ème}$ composant appartenant à l'$i^{ème}$ famille ainsi que de **l'identifiant unique du port** (Ndiaye, et al., 2016) reliant le $j^{ème}$ composant appartenant à l'$i^{ème}$ famille au modèle générique du réseau. En définitive, la fonction *init_Tables()* est une fonction dédiée au composant réseau qui :

- dépose dans la place *Table commutation* autant de jetons qu'il y a d'instances de composants de toutes les familles d'équipements d'automatisation (c'est à dire exceptée la famille réseau) dans une architecture donnée,
- associe à chaque jeton déposé une couleur constituée d'un couple (identifiant du composant, identifiant de son port de connexion).

Définition 11

La fonction *init_table()* est définie de la manière suivante :

$$init\_table() = \sum_{i=2}^{i=F} \sum_{j=1}^{j=C_i} instance(composant\_i\_j) \text{ `} (\lambda_{1\_1\_1} = id\_comp, \lambda_{1\_1\_2} = id\_port)$$

Pour définir un modèle de réseau correspondant au nombre de composants définis dans le modèle « constructeur », il suffira de reproduire ce modèle de base, en considérant autant de transitions de substitution *Rx_i_j* et *Tx_i_j* qu'il y a de type de composants dans le modèle « *constructeur* ». Afin d'illustrer le principe du modèle générique du commutateur, la Figure 39 fourni un exemple basé sur le modèle « *constructeur* » défini sur la Figure 35.

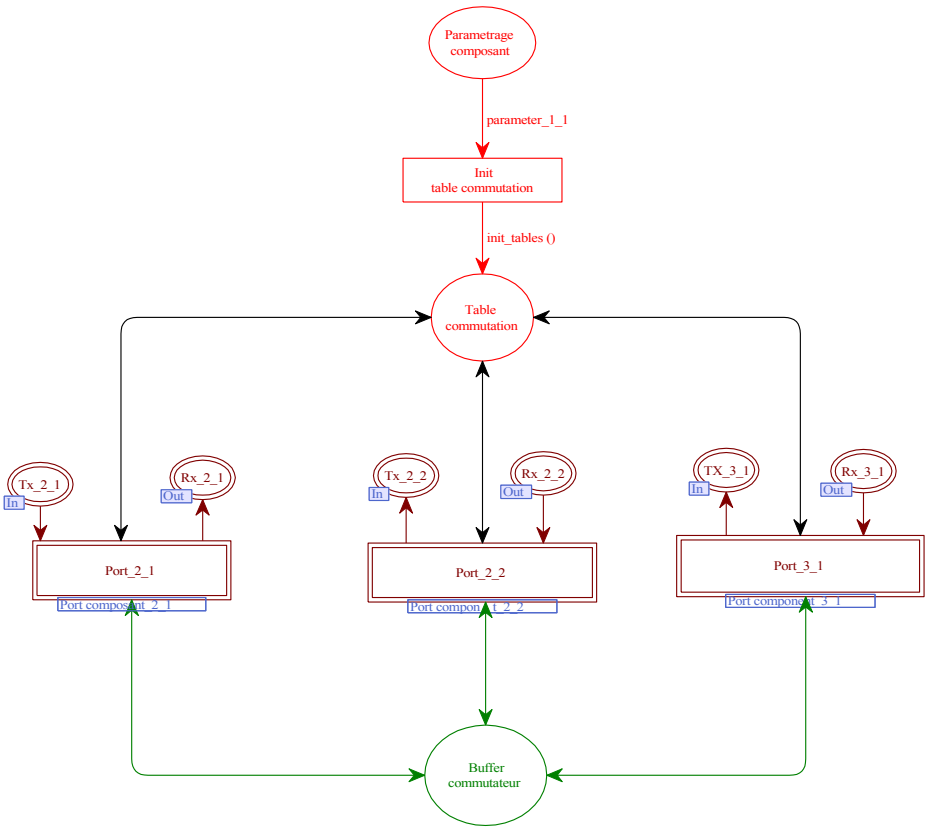


Figure 39: Exemple modèle de commutateur

Ce modèle est composé de 3 transitions de substitutions appelées *'port_2_1', 'port_2_2', 'port_3_1'* représentant les connexions entre les composants génériques *'composant_2_1'*, *'composant_2_2'*, *'composant_2_1'* et le modèle générique du commutateur. Cette connexion se fait à travers leurs places respectives *Tx_2_1* et *Rx_2_1, Tx_2_2* et *Rx_2_2 et Tx_3_1* et *Rx_3_1* qui représentent l'interface de communication des différents composants génériques. Chacun de ces ports est relié à la place *'buffer commutation'* représentant le buffer interne du commutateur. Ainsi, pour chaque composant générique modélisé, un port lui est associé dans le modèle générique de composant. Chaque port possède un numéro unique permettant son identification dont la valeur est affectée lors de la construction du modèle de constructeur.

3.2.2.2 La fabrique de commutation

La commutation des jetons vers le port de destination se fait dans le modèle de fabrique de commutation auquel fait appel la transition de substitution *Port_i_j*. Comme indiqué dans la section précédente, ce modèle est associé à un et un seul type de composant *i_j*. Le modèle de fabrique de commutation, présenté sur la Figure 40, est composé de cinq places :

- une place d'entrée Tx_i_j et une place de sortie Rx_i_j, associées à l'ensemble de couleurs paquet,
- une place *Buffer commutateur*, associée à une couleur de type *Liste* contenant une liste ordonnée de jetons de couleur *paquet* transmis par un composant,
- une place *'Table commutation'* où sont stockés les jetons contenant les informations de commutation obtenu par exécution de la fonction *init_table ()* ; les couleurs de ces jetons sont des couples (identifiant du composant, identifiant du port)
- une place *id Port i_j* qui contient le jeton portant comme couleur, de type entier, l'identifiant unique du port associé aux $j^{ème}$ composant de la $i^{ème}$ famille. La valeur de cet identifiant unique est définie de manière statique lors de l'élaboration du modèle de constructeur pour l'ensemble des composants génériques.

La transition *Entree_buffer* a pour rôle de stocker dans un buffer, selon une logique FIFO, les messages émis par les composants et en attente de commutation.

La transition *Sortie_buffer* réalise l'aiguillage des jetons de couleurs *paquet*. Elle ne sera franchissable que pour les jetons contenus dans la place *Buffer commutateur* à destination d'un composant *IDcomposant* ayant comme identifiant de port *IDport* (association définie dans la table de commutation), et dont celui-ci équivaut au numéro de port *numeroPort* auquel le composant est connecté dans la topologie de l'architecture instanciée. Cette condition de franchissement est matérialisée par la garde *[IDPort = numeroPort]* portée par la transition *Sortie_Buffer*. En d'autres termes, cette garde permet le franchissement de la transition *Sortie_Buffer* par les jetons paquet par rapport aux valeurs des couleurs destinations, de l'identifiant du port et du numéro de port. C'est à dire les jetons contenus dans la place *Table_commutation* ayant, pour identifiant de port, la valeur du numéro du port auquel le composant est connecté dans la topologie.

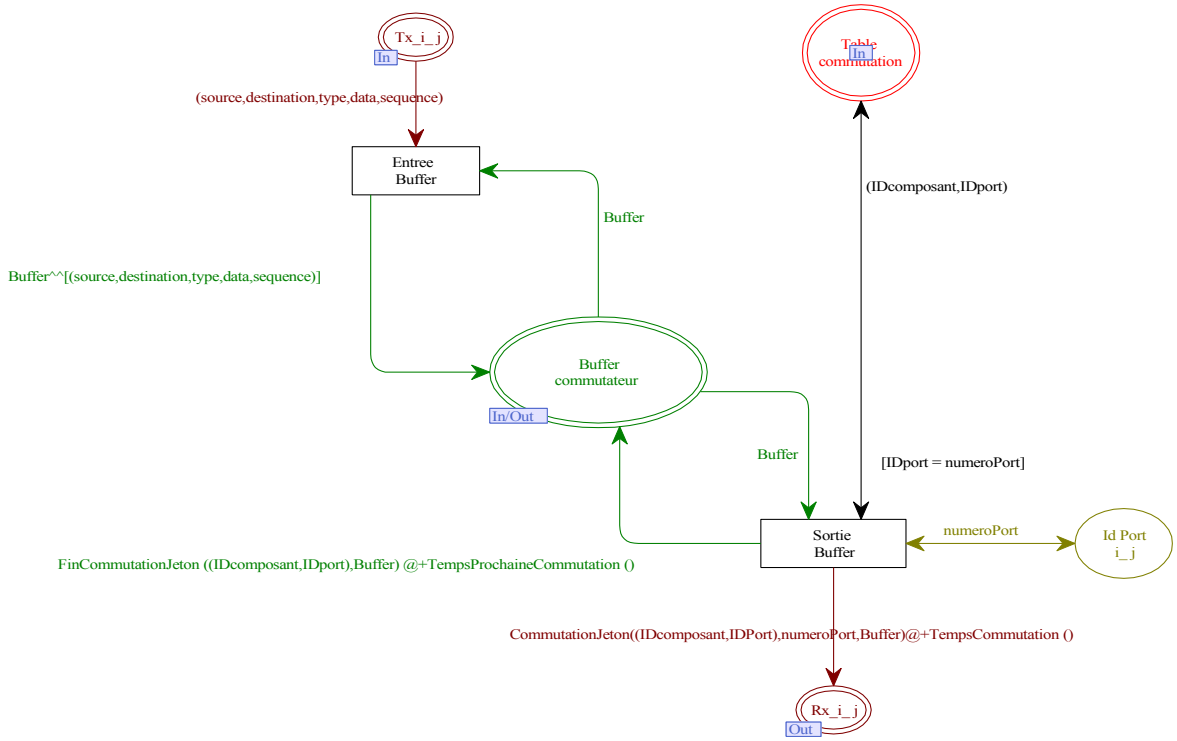


Figure 40: Modèle de la fabrique de commutation

Sur l'arc de sortie reliant la transition *Sortie_buffer* et la place *Rx_i_j*, la fonction d'incidence arrière *CommutationJeton((IDComposant,IDPort), numeroPort, Buffer)* effectue la commutation. Pour cela, la fonction sélectionne, à l'aide d'une fonction de filtrage ML, le premier élément de la liste *Buffer* ayant comme destinataire un composant *i_j*, puis le dépose dans la place *Rx_i_j* affecté d'une temporisation *TempsCommutation()* qui correspond au temps de commutation.

Sur l'arc de sortie reliant la transition *Sortie_buffer* et la place *Buffer_commutateur*, la fonction d'incidence arrière *FinCommutationJeton((IDcomposant,IDport),Buffer)* prépare la commutation suivante en modifiant la liste de messages à commuter de la manière suivante :

- le premier message à destination d'un composant *i_j*, commuté par la fonction *CommutationJeton((IDComposant,IDPort),numeroPort,Buffer)* est retiré de la liste,
- les messages suivants, à destination d'un composant *i_j*, sont affectés d'une temporisation qui correspond au délai d'attente avant une prochaine commutation pour un même composant,
- les autres messages de la liste (ayant des destinataires différents que les composants *i_j*) demeurent inchangés.

3.3 Algorithmes pour la génération automatique de modèle de RdP

A partir d'un modèle « *constructeur* » tel que défini en section 3.1 et d'une bibliothèque de composants génériques telles que définie en section 3.2, la génération automatique d'un modèle CPN représentant le comportement d'une architecture donnée repose sur les phases suivantes :

- *Sélection et adaptation d'un modèle « constructeur »* : pour rappel, un modèle « *constructeur* » est construit pour un ensemble d'architectures dont les familles et composants sont connus ; par construction, un modèle « *constructeur* » intégrant la

totalité des équipements d'un fournisseur sera donc adapté pour toute architecture faisant appel (ou non) à ces composants. Si toutefois, le découpage en familles / composants devait ne pas convenir, il serait alors nécessaire d'élaborer un nouveau modèle « *constructeur* » en suivant les mêmes principes que ceux indiqués précédemment.

- *Importation de la description semi-formelle d'une architecture :* l'import consiste à mettre les données issues de la conception d'architectures en avant-vente (composants, topologie, paramètres, flux de communication) sous le format des ensembles de couleurs associés aux places du modèle « *constructeur* » (et par conséquence aux places d'entrée des modèles de composant).
- *Tir des transitions du modèle « constructeur » :* les fonctions d'incidence arrière présentes dans ce modèle contribuent, lors du tir des transitions, à l'instanciation et au paramétrage des composants via l'affectation de couleurs aux jetons.
- *Tir des transitions d'initialisation dans les modèles génériques de composant :* le paramétrage des composants est poursuivi par le tir de ces transitions qui contribuent à distribuer aux places des modèles de composant les éléments de paramétrage, sous forme de couleurs, dont celles-ci ont besoin.

Comme indiqué précédemment l'effort pour la génération d'une architecture une fois que le modèle de constructeur, les composants génériques ainsi que l'ensemble des couleurs ont été définis se situe donc sur cette assignation de valeurs. Celle-ci se fait automatiquement à travers l'algorithme défini ci-dessous.

Algorithme de génération automatique d'un modèle d'architecture

Input Données semi-formelles de description d'architecture, Bibliothèque de composants génériques

1. Sélection du modèle « constructeur »

2. Import de la description d'architecture

/ Variables associées aux couleurs */*

$(\gamma, \omega_i, \delta_{i\_j}, \lambda_{i\_j\_k})$,

/ Déclaration des instances */*

instance(composant_i_j)

/ liste des jeux de paramètres associés aux instances du composant i_j */*

$param_{i\_j} = \{(\lambda_{i\_j\_1} \dots \lambda_{i\_j\_K}), (\lambda_{i\_j\_1} \dots \lambda_{i\_j\_K}), \dots, (\lambda_{i\_j\_1} \dots \lambda_{i\_j\_K})\}$

3. Tir des transitions du modèle « constructeur »

4. Tir des transitions d'initialisation des modèles de composants

Algorithme 1 : Génération automatique des modèles d'architecture par instanciation et paramétrage

Il est important de souligner, à ce stade, que :

- les phases 2, 3 et 4 de l'algorithme sont entièrement automatiques : l'import de la description d'architecture en phase 2 est supportée par un script qui sera décrit dans le chapitre 4 et les phases 3 et 4 seront supportées par l'exécution du modèle en simulation à l'aide d'un outillage qui sera, lui aussi, décrit au chapitre 4 ;

- la phase 1 peut nécessiter l'intervention d'un expert pour ajuster le modèle CPN « constructeur » si celui-ci ne correspond pas à une topologie standard évaluée en phase d'avant-vente (notamment si de nouvelles familles ou de nouveaux types de composants sont considérés).

L'effort de modélisation peut donc être considéré comme négligeable en phase d'avant-vente dans la mesure où il se limite à la description informelle d'une architecture selon un format prédéfini compatible avec la définition du modèle CPN « constructeur ». Ce format correspond, en fait, aux modèles UML présentés en fin de chapitre 1 et fera l'objet d'une modélisation XML dans le chapitre 4. Cette automatisation de la phase de construction des modèles supportant l'évaluation des performances présente donc un potentiel significatif en termes de ratio entre le nombre d'architectures évaluables et l'effort de modélisation (en temps et en coût).

3.4 Evaluation de performances

Même si un modèle d'architecture en réseau de Petri coloré et temporisé peut-être construit automatiquement à l'aide du modèle « constructeur », associé à une bibliothèque de modèle de composant, il demeure incomplet pour procéder à une évaluation de performances. En effet, le modèle d'architecture doit être enrichi par un ensemble d'observateurs permettant d'effectuer des mesures associées aux places (marquage moyen par exemple) ou aux transitions (fréquence de franchissement par exemple). De manière similaire à l'approche mise en œuvre en section 3, l'objectif de cette section est :

- de permettre la génération automatique de ces observateurs à partir d'une description informelle des performances à évaluer
- générer les paramètres des simulations de Monte-Carlo requis pour l'évaluation des performances (critères d'arrêt, nombre d'histoires simulées, …).

3.4.1 Génération des observateurs de performance

La génération des observateurs de performance consiste à :

- formaliser un cadre de représentation des différentes performances à évaluer,
- définir, pour chaque type de performance, un observateur complexe, exploitant les concepts de base relatifs au marquage moyen ou aux fréquences de franchissement pour couvrir la mesure d'une performance, en l'associant au modèle d'architecture, et notamment aux places et/ou transitions placées sous surveillance.

3.4.2 Cadre de représentation des performances à évaluer

Comme défini dans le chapitre 1, les performances à évaluer dans les architectures de contrôle-commande sont les temps de réponses entre un émetteur et un récepteur et le temps cycle des composants.

3.4.2.1 Temps de réponse

Le temps de réponse peut se mesurer selon deux modes :

- mode aller simple pour lequel le temps de réponse se mesure entre l'envoi d'un message par un émetteur et sa réception par un destinataire ; conformément aux besoins exprimés au chapitre 1, cette performance est évaluée dans le cas spécifique d'une remontée d'alarme entre des composants de type entrées/sorties (I/O device) et un destinataire (de type SCADA ou automate programmable) ;
- mode aller-retour pour lequel le temps de réponse se mesure entre l'envoi d'un message par un émetteur et la réception d'un réponse par ce même émetteur, c'est à dire la somme de deux temps de réponse (envoi par l'émetteur – réception par le destinataire et envoi par le destinataire, réception par l'émetteur) et d'un temps de traitement de la requête par le destinataire ; conformément aux besoins exprimés au chapitre 1, cette évaluation de performance concerne des messages de commande pour lesquels un émetteur envoie une requête à un destinataire et attend, en retour, un accusé de réception.

Pour chaque flux de messages dont il souhaite évaluer le temps de réponse, le système intégrateur devra indiquer : le mode à évaluer (aller simple pour les messages de type alarme ou aller-retour pour les messages de type commande), l'identifiant de l'émetteur et l'identifiant du récepteur.

3.4.2.2 Temps de cycle des composants

Cette évaluation ne s'applique qu'aux composants ayant un comportement cyclique (automates programmables industriels par exemple). Deux cas de figures peuvent être envisagés :

- temps de cycle de durée fixe (mode périodique) : ce temps est fixe et défini dans les paramètres du composant; cependant, le composant traite toutes les tâches et messages, même si le temps de cycle imparti est dépassé, mais émet, dans ce cas, un message d'alerte indiquant un dépassement de cycle (chien de garde dans les automates programmables industriels) ; la performance évaluée se limite donc au nombre de dépassements constatés sur une période d'observation.
- temps de cycle variable : temps nécessaire pour le traitement de l'ensemble des tâches et messages ; le temps de cycle correspondra à la durée entre la date de début de traitement de la première tâche et la date de fin de traitement de la dernière tâche.

Dans tous les cas, l'ingénieur en avant-vente devra indiquer les identifiants des composants dont il souhaite évaluer les temps de cycle ainsi que le mode de fonctionnement.

3.4.3 Définition des observateurs de performance

L'analyse de performance utilisant les réseaux de Petri colorés et temporisés définis par (Jensen, et al., 2007) s'appuie notamment sur la simulation. Au cours de celle-ci, le réseau peut contenir et générer beaucoup d'informations quantitatives sur la performance d'un système,

telles que la longueur de file d'attente, le temps de réponse, le débit, etc. Ces informations peuvent être extraites du modèle en examinant les marquages ou les tirs des transitions puis enregistrées à l'aide de **moniteurs** (Jensen, et al., 2007).

3.4.3.1 Les moniteurs

Un moniteur est chargé de collecter des données statistiques lors de la simulation Il constitue un mécanisme utilisé pour observer, inspecter, contrôler ou modifier une simulation d'un réseau de Petri coloré et temporisé. De nombreux moniteurs différents peuvent être définis pour un réseau de Petri donné.

Les moniteurs peuvent inspecter à la fois les marques des places et les tirs de transitions lors d'une simulation. La collecte des informations peut s'effectuer périodiquement ou bien sur occurrence d'un événement particulier (tir d'une transition par exemple).

Sur la base des informations collectées, les moniteurs peuvent également effectuer des traitements ayant un intérêt vis à vis de la performance à évaluer.

Par exemple, un collecteur de données qui mesure la longueur d'une file d'attente de paquets peut calculer à la fois la longueur moyenne de la file d'attente ainsi que la longueur maximale de la file d'attente. Parmi les traitements réalisables par un moniteur, nous pouvons citer, à titre d'exemple :

- arrêter une simulation lorsqu'un endroit particulier est vide,
- compter le nombre de fois qu'une transition se produit,
- mettre à jour un fichier lors du franchissement d'une transition avec une variable liée à une valeur spécifique à observer,
- calculer le nombre moyen de jetons dans une place.

(Jensen, et al., 2007) ont défini dans leur outil CPN-Tools des moniteurs natifs dont les fonctions ont été prédéfinies, qui sont:

- *Breakpoint monitor* : sont utilisés pour arrêter une simulation,
- *Marking size* : enregistre le marquage d'une place,
- *Count transition* : compte le nombre de tir d'une transition,
- *List lenght* : mesure la longueur d'une liste,
- *Generic data collector* : traitement spécifique défini par le modélisateur.

Il est également possible de définir des moniteurs personnalisables en fonction des besoins de l'utilisateur final, qui sont :

- *Data collector monitor* pour les lesquels les conditions d'extraction (*predicate*) et les traitements (*observer*) doivent être précisés en fonction du contexte,
- *User define monitor* qui enrichit les moniteurs de type *Data collector* par des fonctions dédiées aux post-traitements comme par exemple l'enregistrement personnalisé des observations dans des fichiers.

En revanche, tous les moniteurs partagent le mode de fonctionnement commun suivant :

Moniteur

Répéter jusque *fin simulation*
Examiner périodiquement certaines marques et/ou les transitions en cours,
Vérifier si une condition particulière est remplie
Si la condition est remplie
alors
Effectuer un ou plusieurs traitements à partir des informations observées (informations contenues dans le marquage et/ou les éléments de liaison)
Effectuer un enregistrement
Sinon ne rien faire
Fin_si
Fin Répéter

Fonctionnement général d'un moniteur

Dans le cadre de cette étude, le type de moniteur utilisé est *User define monitor*. En effet, d'une part les moniteurs prédéfinis nécessite de renseigner de manière statique l'ensemble des informations permettant le relevé statistique. Ce qui entraine une modification manuelle de ceux-ci à chaque nouvelle architecture, ce qui en contraire à nos exigences définie dans le chapitre 1. D'autre part les *User define monitors* qui offrent le plus de latitudes pour spécifier précisément la performance à évaluer et surtout définir automatiquement les données enregistrées modulo de définir l'ensemble de ses fonctions.

En effet ce moniteur est défini par un 5-uplet de fonctions (*init*, *predicat*, *observer*, *action*, *stop*) où :

- ***Init***, est la fonction fournissant une action déterminée lors du début de la simulation.
- ***Predicat*** est une fonction qui fixe les conditions dans lesquelles une observation doit être réalisée (ce qui conditionne notamment les instants où la valeur surveillée doit être relevée),
- ***Observer*** est une fonction qui permet d'effectuer la relevée de la valeur désirée,
- ***Action*** est une fonction déterminant des post-traitements à effectuer avec la valeur relevée et sur les enregistrements,
- ***Stop*** est la fonction définissant l'action à réaliser à la fin de la simulation.

Les données numériques enregistrées sont ensuite utilisées pour calculer des statistiques et produire les différents indicateurs de performances.

3.4.3.2 Définition des moniteurs associés à la mesure des temps de réponse

Cas du temps de réponse en mode « Aller simple »

Dans le modèle d'architecture, le temps de réponse d'un aller simple entre un émetteur et un récepteur sera défini comme la différence entre la date d'émission d'un message et sa date de réception par le destinataire. Lors de l'émission d'un message, nous avons indiqué dans les

sections précédentes que les jetons de couleur paquet déposés dans la place *envoi_paquet* étaient horodatés.

Le moniteur permettant l'évaluation du temps de réponse pour un aller simple doit donc être en mesure de collecter la date de réception des messages par les destinataires. Par hypothèse, cette date de réception correspond au franchissement de la transition nommée *transition_evaluation_performance* et le dépôt d'un jeton dans la place *fin_traitement_paquet* de la Figure 36.

Le type de service concerné par une évaluation du temps de réponse en mode aller simple est le service *alarm*. Ce service correspond à une remontée d'alarme, de type défaillance, en provenance d'un équipement à des fins d'informations et de surveillance.

Le moniteur associé à cette transition aura donc la structure suivante :

Moniteur temps de réponse aller simple	
Transition observée : *transition_évaluation_performance*	/* Figure 36*/
Init : création fichier moniteur	/* Création du fichier pour les relevés statistiques du moniteur lors du démarrage de la simulation (action réalisée une seule et unique fois). Puis ouverture de celui-ci pour l'écriture des relevés statistiques. */
Predicat : *typeService = alarm*	/* *typeService* est une couleur qui est un élément du produit cartésien de l'ensemble de couleur *paquet*. La transition ne sera observée que si et seulement si la valeur contenue dans cette couleur est égale à *alarm*. */
Observer :	
id_s = *source*	/* *TimeStamp*, *Source*, *destination* sont des couleurs associées au jeton ayant provoqué le franchissement de la transition observée (*TimeStamp* correspond à l'horodatage du message émis, *source* au composant émetteur et *destination* au composant récepteur). *CurrentTime* est la date de franchissement de la transition *observée* qui est fournie par la fonction ML native *Time()*.*/
id_d = *destination*	
T_{aller} =*CurrentTime – TimeStamp*	
Action : Enregistrer le triplet (id_s , id_d , T_{aller})	/* enregistrement des valeurs dans le fichier de moniteur créé. */
Stop : Fermeture fichier	/* Fermeture du fichier de moniteur lorsque le moniteur break point survient marquant la fin de la simulation.*/

Moniteur pour les temps de réponse aller simple

Par définition, ce moniteur enregistre dans un fichier externe au fil de l'eau tous les messages de type *alarm* reçus par un composant donné.

Pour obtenir les temps de réponse souhaités par les ingénieurs en avant-vente, l'algorithme suivant sera exécuté :

Algorithme d'évaluation des temps de réponse en mode aller simple

Sélectionner les enregistrements associés au moniteur du composant *i_j* dont le « **destinataire** » est une instance,
/ Post-traitement */*

Répéter pour chaque couple (émetteur, récepteur) d'intérêt */* *couples pour lesquels les ingénieurs en avant-vente souhaitent évaluer les temps de réponse */*

Extraire les données relatives à (identifiant émetteur, identifiant récepteur, T_{aller}),
Calcul de la performance attendue (temps moyen, bornes inférieures ou supérieures, écart type, …) ; */*ce point fera l'objet de la section 3.5.*/*

Fin Répéter

Algorithme 2: post-traitement des moniteurs pour les temps de réponse aller simple

Cas du temps de réponse en mode « Aller/Retour »

Dans le modèle d'architecture, le temps de réponse aller/retour entre un émetteur et un récepteur est composé du temps mis par le jeton depuis sa génération par le générateur de paquet d'un composant en mode client jusqu'à son retour au niveau de l'expéditeur en incluant son temps de traitement par le composant destinataire.

Ce type de performance est évalué pour les messages de type commande et correspond au temps de réponse entre l'émission d'une requête et la réception d'un accusé de réception. Dans le modèle d'architecture, le message aller/retour est porté par un jeton unique de couleur paquet. Ce message étant horodaté lors de son émission par le composant en mode client, le moniteur associé à la mesure de temps de réponse aller/retour sera donc similaire à celui associé au temps de réponse en mode aller simple. La seule différence portera sur le prédicat qui, dans le cas d'un temps de réponse aller/retour, sélectionnera les messages de type commande (par exemple requête entre un système SCADA et un I/O device, requête entre un PLC et un I/O device ou encore requête entre deux PLC).

Moniteur temps de réponse aller/retour

Transition observée : *transition_évaluation_performance* /* Figure 36*/
Init : création fichier moniteur
Predicat : vérifier si *typeService = command*
Observer : id_s = *source* ; id_d = *destination*
$T_{aller/Retour}$ =CurrentTime – TimeStamp
Action : Enregistrer le triplet (id_s , id_d , $T_{aller/Retour}$)
Stop : Fermeture fichier

Moniteur pour le temps de réponse aller-retour

De la même manière, l'algorithme d'évaluation des temps de réponse en mode aller/retour sera similaire à celui en mode aller simple, excepté pour le moniteur sélectionné qui devra, dans ce cas, être le moniteur associé au composant source.

Algorithme d'évaluation des temps de réponse en mode aller/retour

Sélectionner les enregistrements associés au moniteur du composant *i_j* dont la « **source**» est une instance,

/ Post-traitement */*

Répéter pour chaque couple (émetteur, récepteur) d'intérêt

Extraire les données relatives au couple (émetteur, récepteur, $T_{aller/Retour}$),

Calcul de la performance attendue (temps moyen, bornes inférieures ou supérieures, écart type, …) ; */*ce point fera l'objet de la section 3.5.*/*

Fin Répéter

Algorithmes 3: post-traitement des moniteurs pour les temps de réponse aller/retour

3.4.3.3 Définition des moniteurs associés à la mesure des temps de cycle

La performance relative au temps de cycle des composants se définit comment étant le temps mis par le processeur du composant pour réaliser l'ensemble des tâches qui lui sont allouées. La réalisation de ces tâches suit un processus cyclique de durée fixe et prédéterminée ou bien de durée variable en fonction des tâches à traiter (par exemple pour un automate programmable industriel, lecture des entrées, traitements, écriture des sorties).

L'évaluation de ces performances nécessite une modélisation des composants telle qu'elle puisse faire apparaître explicitement la représentation des différentes étapes d'un cycle à l'aide d'un ensemble de places P_{cycle} constituant une composante conservative :

$$\sum_{P_i \in P_{cycle}} m(P_i) = 1.$$

L'ensemble de couleur associé à l'ensemble de places P_{cycle} est l'ensemble *paquet* de telle sorte à ce qu'il soit compatible avec l'ensemble de couleur de la place *fin_traitement_paquet*. Parmi les ensembles de couleurs entrant dans le produit cartésien de l'ensemble *paquet*, seules deux couleurs seront renseignées pour les jetons modélisant le cycle de traitement d'un composant :

- *TimeStamp* : lors du dépôt d'un jeton dans la place initiale du cycle, la couleur prend la valeur $t_{current}$ correspondant au temps courant de la simulation,
- *typeService* : lors du dépôt d'un jeton dans la place initiale du cycle, la couleur prend la valeur *Cycle_composant*, ce qui permettra l'identification de ce type de jeton par le moniteur associé au temps de cycle.

La transition associée à l'événement « fin de cycle » dépose un jeton dans la place initiale du début de cycle (pour en débuter un suivant) et en dépose également un dans la place *fin_traitement_paquet*. Ce dépôt permet le franchissement de la transition *fin_evaluation_performance* et enclenche le traitement nécessaire à l'évaluation du temps de

cycle à l'aide d'un moniteur construit sur les mêmes bases que ceux utilisés pour les évaluations des temps de réponse.

Moniteur temps de cycle

Transition observée : *transition_évaluation_performance* /* Figure 36*/
Init : création fichier moniteur
Predicat : vérifier si *typeService = Cycle_composant*
Observer :
Id_comp = ID_{PLC}
T_{cycle} = CurrentTime – TimeStamp
Action : Enregistrer le doublet (Id_comp, T_{cycle})
Stop : Fermeture fichier

Moniteur pour les temps de cycle

L'algorithme d'évaluation du temps de cycle sera basé sur la sélection du moniteur associé au composant dont on veut évaluer le temps de cycle et sur l'exécution d'un post-traitement permettant de différencier les résultats selon que le composant est dans un mode cycle de durée fixe ou de durée variable.

Algorithme d'évaluation du temps de cycle

Sélectionner les enregistrements associés au moniteur du composant *i_j* dont on veut évaluer le temps de cycle,
/ Post-traitement */*
Nbre_dépasssement ← 0 ;
Si mode fonctionnement PLC = périodique
Répéter pour chaque enregistrement
Si $T_{cycle} > D_{max}$ */* D_{max} est la durée de cycle fixe et prédéterminée */*
alors Nbre_dépasssement ← Nbre_dépasssement +1 ;
Sinon ne rien faire
Fin_si
Fin_répéter
Sinon Calcul de la performance attendue (temps moyen, bornes inférieures ou supérieures, écart type, …) ; */*ce point fera l'objet de la section 3.5.*/*
Fin_si

Algorithme 4: post-traitement des moniteurs pour l'évaluation des performances aller simple

3.5 Paramètres de simulation de Monte-Carlo

Les moniteurs présentés dans la section précédente permettent d'obtenir un ensemble de valeurs brutes relatives à une performance donnée. Nous avons montré que leur mise en œuvre au sein d'un algorithme d'évaluation requiert un post-traitement complémentaire permettant d'exhiber la performance attendue (temps moyen, bornes inférieures ou supérieures, écart type, intervalle de confiance, …).

Par ailleurs, les simulations de Monte-Carlo sont basées sur la simulation d'un ensemble d'histoires (que l'on appellera *réplications*), chacune d'entre elles ayant une durée bornée à l'aide d'un critère d'arrêt. Compte tenu du fait que les histoires sont indépendantes et identiquement distribuées, les résultats obtenus pour chaque scénario seront différents.

Ces trois éléments (traitements statistiques à réaliser, nombre d'histoires et critère d'arrêt) dépendent fortement du type et de la précision des performances que l'on souhaite évaluer. Elles sont donc spécifiques à chaque étude d'avant-vente.

Dans le cadre de cette étude, et compte tenu du besoin exprimé par la société Schneider Electric, nous avons fait le choix de présenter chaque mesure de performance à l'aide de quatre indicateurs :

- Valeur moyenne sur l'ensemble des histoires,
- Valeur maximale sur l'ensemble des histoires,
- Valeur minimale sur l'ensemble des histoires,
- Intervalle de confiance à 95%

Soit n le nombre d'histoires (réplications) et x_i l'ensemble des valeurs observées pour chaque histoire i. La moyenne estimée est la somme de l'ensemble des valeurs observées par histoire, divisé par le nombre d'histoires ; elle est donnée par :

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

La variance et l'écart type permettent de s'assurer de la distribution de la valeur estimée par rapport à la valeur moyenne réelle. L'estimation de l'écart type est donnée par :

$S_x = \sqrt{S_x^2}$ où S_x^2 représente l'estimation de la variance donnée par $S_x^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x_1})^2}{n-1}$

Il est alors possible d'obtenir un intervalle de confiance (dans notre cas fixé à 95%) par :

$$\left[\bar{x} - t_\alpha \frac{S_x}{\sqrt{n}}, \bar{x} + t_\alpha \frac{S_x}{\sqrt{n}}\right]$$

où t_α est le paramètre associé au degré de confiance $\alpha = 0.05$. Dans notre cas, l'information transmise aux ingénieurs d'avant-vente sera la demi-longueur de l'intervalle de confiance à 95%.

En ce qui concerne le critère d'arrêt et le nombre de réplications, nous avons l'hypothèse que leur définition était du ressort des ingénieurs en phase d'avant-vente. Pour le critère d'arrêt, ce choix est justifié par le retour d'expérience des ingénieurs, notamment sur les durées d'observations des architectures dans le cadre des tests pratiqués en plate-forme. En ce qui concerne le nombre de réplications, celui-ci dépend de la précision souhaitée qui peut être évaluée par le rapport entre la demi-longueur de l'intervalle de confiance et la moyenne estimée. Ces deux paramètres (critère d'arrêt, nombre de réplications) sont donc, dans le prototype présenté au chapitre suivant, définis par les ingénieurs d'avant-vente sur leur

interface dédiée à la description des architectures. Une évolution vers la prise en charge de ces deux paramètres au sein de ce prototype, par exemple à partir d'une information de précision fournie par les ingénieurs d'avant-vente, ne présente aucune difficulté majeure et pourra être intégrée dans nos perspectives de transfert technologique.

4 Conclusion

Notre contribution, dans le cadre de ce chapitre, porte sur la définition d'un modèle « constructeur » dans le formalisme des réseaux de Petri colorés et temporisés définis par Jensen. Ce modèle décrit une architecture de contrôle-commande de manière générique, hiérarchique et structurée et embarque des fonctions d'instanciation et de paramétrage. Cet ensemble, associé à une bibliothèque de modèles de composants, constitue le socle de la génération automatique des modèles d'architecture. Le principal intérêt réside dans une réduction très significative de l'effort de modélisation requis pour élaborer les modèles des nombreuses architectures évaluées en phase d'avant-vente. Enfin, pour que ce modèle soit utilisable pour l'évaluation de performances, il a été nécessaire de définir des observateurs que nous avons souhaités aussi génériques que possible afin de rendre leur élaboration transparente pour les ingénieurs d'avant-vente.

Le chapitre suivant est consacré au développement d'un prototype, basé sur l'outil CPN Tools, et sur l'illustration de notre contribution au travers de différents cas d'étude.

Chapitre 4
Application sur un cas d'étude

1 Introduction

L'objectif de ce chapitre est, d'une part, de montrer la faisabilité de la méthode de génération automatique et d'évaluation des modèles d'architectures proposée au chapitre précédent, et d'autre part, de mettre en évidence l'intérêt de l'approche proposée, notamment en termes d'effort de modélisation.

La démonstration de faisabilité est réalisée à travers le développement d'un prototype d'évaluation de performance permettant la description d'une architecture de contrôle-commande par un ingénieur en avant-vente, la génération et la simulation des modèles d'architectures et l'élaboration des indicateurs de performance fournis aux ingénieurs en avant-vente. Ce prototype a été réalisé sur la base de deux outils existants : l'outil Schneider de définition des architectures PSX builder[4] et l'outil CPN Tools[5] permettant l'édition et la simulation de réseaux de Petri colorés et temporisés.

La mise en évidence de l'intérêt de l'approche proposée est réalisée au travers du traitement d'un cas d'étude nécessitant l'élaboration et l'évaluation d'une architecture de contrôle commande pour une usine de traitement d'eaux usées. Nous montrons, en particulier, comment à partir d'un modèle « *constructeur* » unique et d'une bibliothèque de composants, les fonctions d'instanciation et de paramétrage permettent le traitement de multiples architectures de manière masquée pour l'ingénieur en phase d'avant-vente et donc sans effort de modélisation et expertise dans le domaine des RdP. Enfin, l'architecture de l'usine de traitement d'eaux usées évaluée en simulation est également déployée, en partie, en laboratoire. Une comparaison entre les résultats de la simulation et ceux recueillis en laboratoire est présentée et contribue à la validation de notre approche.

2 Développement d'un prototype pour l'évaluation de performances

L'architecture fonctionnelle du prototype d'évaluation de performance est donnée par la la Figure 41. Elle comprend trois entités distinctes :

- Le *Serveur de Simulation* contient la librairie de modèles de composants génériques, le modèle *constructeur* muni de ses fonctions d'instanciation et de paramétrage ainsi que qu'un simulateur de réseau de Petri temporisé et coloré tel que défini par (Jensen, et al., 2007). Cette partie intègre également l'ensemble des algorithmes d'instanciation, de paramétrage et de calcul des moniteurs.
- Le *Client* offre une interface graphique permettant la définition formelle de l'architecture, la spécification des performances à évaluer ainsi que la restitution des valeurs des performances une fois évaluées par le *Serveur de Simulation*.
- L'interface de *Communication* entre les parties *Serveur de Simulation* et *Client* assure, dans un sens, la génération d'un fichier alimentant le *Serveur de Simulation* par les données de configuration issues de la partie *Client* (caractérisation des architectures et des performances à évaluer, paramètres de simulation, …) et, dans l'autre sens, la transmission des valeurs des performances évaluées.

[4] PSX builder est un outil de cotation de matériel développé par Schneider-Electric
[5] CPN Tools est développé et distribué par Eindhoven University of Technology

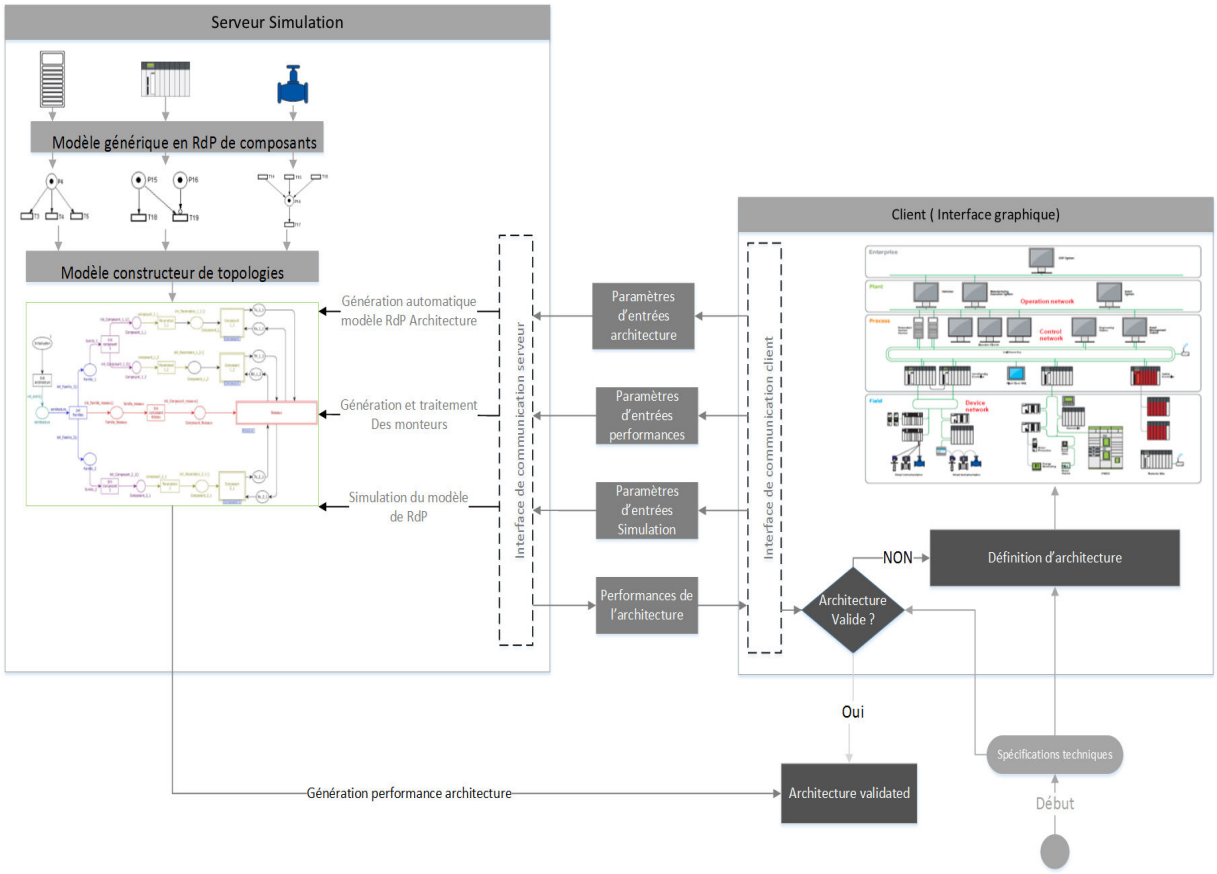


Figure 41: Architecture fonctionnelle du prototype d'évaluation de performances

Le développement du prototype repose sur :

- l'outil CPN Tools pour la partie *Serveur de Simulation*,
- l'outil PSX builder pour la partie *Client*,
- l'échange de fichiers XML, dont la structure reprend les éléments des modèles UML présentés au chapitre 1, pour la partie *Communication*.

2.1 Le Serveur de Simulation

Le serveur de simulation est l'élément central de l'outil d'estimation de performance. Son rôle est d'une part d'intégrer le modèle de constructeur de topologies ainsi que les différents modèles génériques de composants. D'autre part, à partir du fichier de description du modèle de données de l'architecture, il met à jour les fonctions d'instanciation et de paramétrage afin de générer le modèle correspondant à l'architecture. A partir de ce modèle, il permet l'exécution de simulation de Monte-Carlo afin d'estimer les performances. Enfin, il renvoie les relevés statistiques des moniteurs vers le client sous la forme d'un fichier XML.

Le serveur de simulation repose sur l'outil CPN Tools. L'outil CPN Tools est composé de deux parties distinctes : un éditeur sous la forme d'une interface graphique permettant la définition des modèles et des fonctions, et un simulateur permettant l'exécution des modèles préalablement définis et l'obtention des performances.

Les différents éléments de l'outil CPN Tools sont utilisés par le prototype d'évaluation de performance de la manière suivante :

- l'éditeur '*Design CPN*' (Christensen, et al., 1997) est utilisé pour définir la bibliothèque de modèles de composants génériques ainsi que le modèle « constructeur » ; cette phase de définition peut être considérée comme une phase de personnalisation du prototype tenant compte des spécificités et de l'offre catalogue du fournisseur de composants d'automatismes ; par conséquent, cette phase doit être prise en charge par un expert impliqué dans le développement du prototype et doit être totalement transparente pour les utilisateurs finaux (ingénieurs en avant-vente) ;
- le simulateur '*Server CPN*' (Jensen, et al., 2007) est utilisé pour effectuer les différentes simulations de Monte-Carlo et enregistrer les résultats obtenus par les différents moniteurs pour l'évaluation de performances.

2.1.1 Configuration et personnalisation du prototype à l'aide de *Design CPN*

2.1.1.1 Modèle constructeur

La Figure 42 représente le modèle de constructeur de topologie qui a été développé pour le prototype d'évaluation de performance conforme aux besoins exprimés par Schneider Electric.

Celui-ci est composé des 4 grandes familles de composants présentes généralement dans les architectures de C/C de Schneider Electric et 1 famille réseau. Celles-ci se décrivent à travers :

- La famille *SCADA* regroupe l'ensemble des composants dédié à l'acquisition et à la commande opérateur ainsi que les IHM,
- La famille *client* regroupe l'ensemble des composants de type station de travail,
- La famille *device* regroupe l'ensemble des équipements terminaux,
- La famille *réseau* qui regroupe l'ensemble des commutateurs Ethernet,
- La famille *PLC* qui regroupe l'ensemble des automates programmables.

Nous pouvons remarquer que le modèle « *constructeur* » ne contient qu'un seul type de composant par famille (un seul arc en sortie des transitions *init_composant*). Cette restriction est un choix délibéré de Schneider Electric motivé par le fait que le prototype doit permettre de démontrer la faisabilité de l'approche et que le modèle « *constructeur* » peut aisément être enrichi dans les versions ultérieures en phase d'industrialisation. Par ailleurs, il correspond à une réalité industrielle à l'exception de la famille des PLC. En effet, l'offre PLC contient des automates programmables ayant des comportements similaires aux paramètres de performances près (comme par exemple les M340 et M580) mais également des automates ayant des comportements très différents et non modélisables de manière générique. Pour ce cas particulier, le choix d'un seul type de composant pour la famille des PLC (en l'occurrence les PLC de type M340 ou M580) a été justifié par le fait que ces automates sont au cœur de l'offre Schneider alors que d'autres (comme par exemple le Premium) sont considérés en fin de vie.

Par ailleurs, rappelons que le modèle « *constructeur* » contient la définition des différents ensembles de couleurs définis au chapitre 3 ainsi que la définition des fonctions d'incidence arrière utilisées sur les arcs sortants des transitions du modèle « *constructeur* ». Néanmoins, les valeurs des couleurs assignées aux variables caractérisant les jetons circulant dans le réseau

du modèle « *constructeur* » ne sont bien sûr pas définies à stade. Elles correspondent en effet aux différents paramètres d'instanciation des architectures et de paramétrage des composants et dépendront de la topologie et des caractéristiques de l'architecture à évaluer. Ces différents paramètres devront être importés depuis la partie *Client* et faire l'objet d'un traitement particulier par le serveur CPN lors de la simulation, comme nous le verrons par la suite.

Enfin, chacune des familles de composants contient un modèle générique CPN représentant le comportant des équipements de cette famille. Chacun de ces modèles de composants génériques est connecté à la famille réseau à travers leurs places de transmission (*Tx*) et de réception (*Rx*). Afin d'illustrer la démarche, la section suivante présente un exemple de composant générique associé à la famille des PLC.

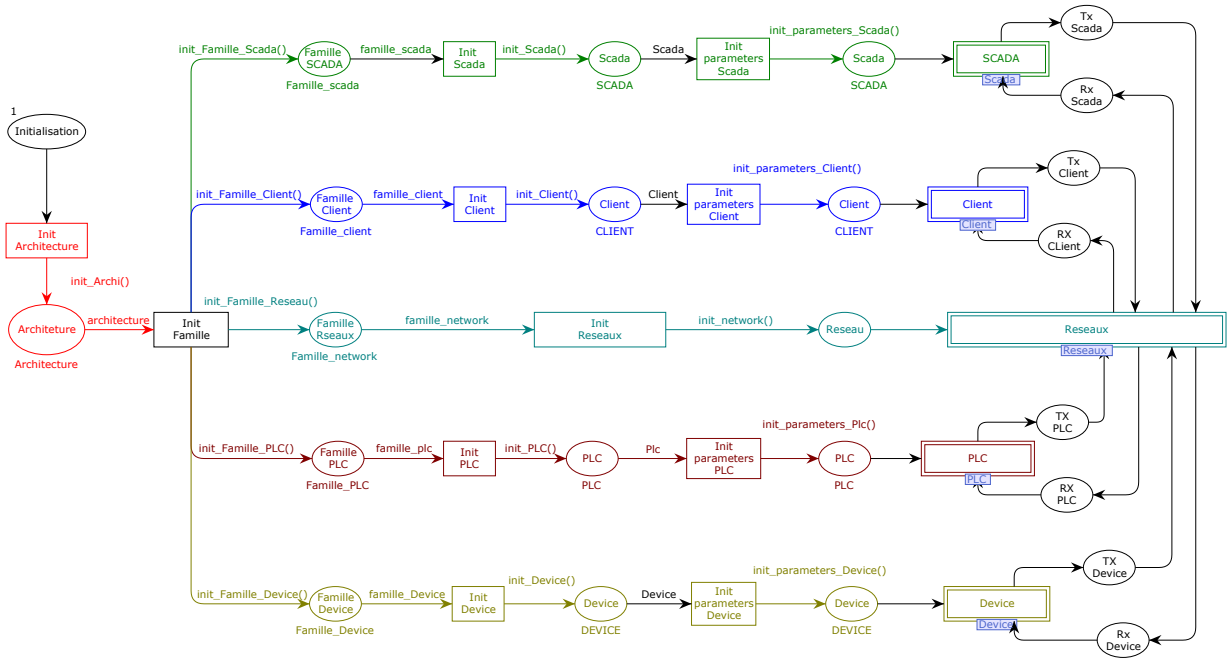


Figure 42: Modèle « *constructeur* » de topologie pour le prototype

2.1.1.2 Bibliothèque de composants génériques : exemple du PLC

Les PLC compris dans l'offre de Schneider-Electric sont des équipements complexes offrant de multiples services : client, serveur et passerelle de communication.

En effet l'automate fonctionne en model *serveur* lorsqu'il traite l'ensemble des requêtes envoyé par le SCADA et les clients. Le traitement de ces requêtes est réalisé par un processeur de manière cyclique ou périodique. Le nombre de requêtes que le processeur est capable de traiter par cycle est limité et dépend de la gamme du PLC. Les requêtes non traitées durant un cycle ou une période sont stockées dans un buffer dont la capacité de stockage dépend aussi de la gamme.

Un PLC peut également fonctionner en mode *client* lorsqu'il envoie des requêtes de mise à jour aux équipements auxquels il est connecté (les I/O devices) ou de synchronisation aux autres PLC présents dans l'architecture.

Enfin les PLC possèdent différentes interfaces de communication, comme indiqué sur la Figure 43 ; ils peuvent donc être connectés à différents réseaux.

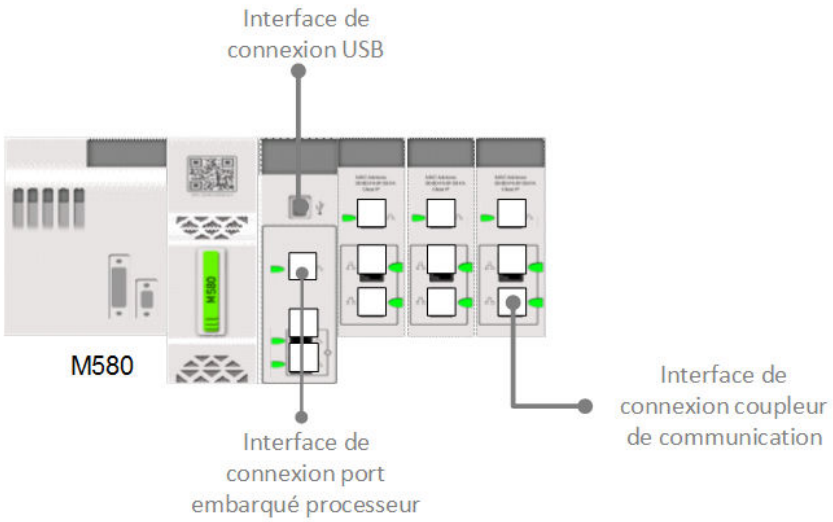


Figure 43: Interface de connexions des PLC

Concernant son architecture technique, un PLC est composé d'un processeur et d'un ou plusieurs coupleurs de communications. Ces deux types d'éléments sont reliés et communiquent à travers un bus de communication.

Le processeur et le(s) coupleur(s) de communication possèdent également un ensemble de ports de communication Ethernet. Ainsi, les requêtes peuvent être reçues et envoyées via les ports de réseau embarqué directement sur le processeur ; dans ce cas, ces requêtes ne sont pas affectées par le temps de traversée du bus de communication. Dans le cas où les requêtes sont envoyées et reçues via les coupleurs de communication, le temps de traversée du bus de communication devra être pris en compte. Par ailleurs, il faut noter que le processeur d'un automate peut aussi recevoir des requêtes à traiter à partir d'un équipement connecté en USB (e.g. un HMI) ; dans ce cas, les requêtes envoyées et reçues n'ont aucun lien avec le réseau car ne le traversant pas. Cette fonctionnalité de communication est donc liée au comportement intrinsèque du composant.

Dès lors le modèle générique d'un automate est représenté sur la Figure 44. Ce modèle suit la définition d'un composant générique (voir chapitre 3, section 3.2.1) où l'on retrouve :

- une transition de substitution nommée *'Comportement PLC'* qui représente le modèle comportemental de l'automate,
- une transition de substitution nommée '*Interface communication'* qui représente le modèle de l'interface de communication,
- des fonctions d'incidences arrières *message_periodique(), message_aperiodique (), caracteritiques_techniques () et interface_Communication ()* qui représentent respectivement les fonctions de paramétrage de la communication périodique, de la communication apériodique, des spécifications techniques de l'automate et enfin des propriétés relatives à la communication du composant. Comme indiqué au chapitre 3, ces quatre fonctions réalisent une extraction des couleurs définies lors de l'exécution de la fonction *init_parametres()* du modèle « *constructeur* ».

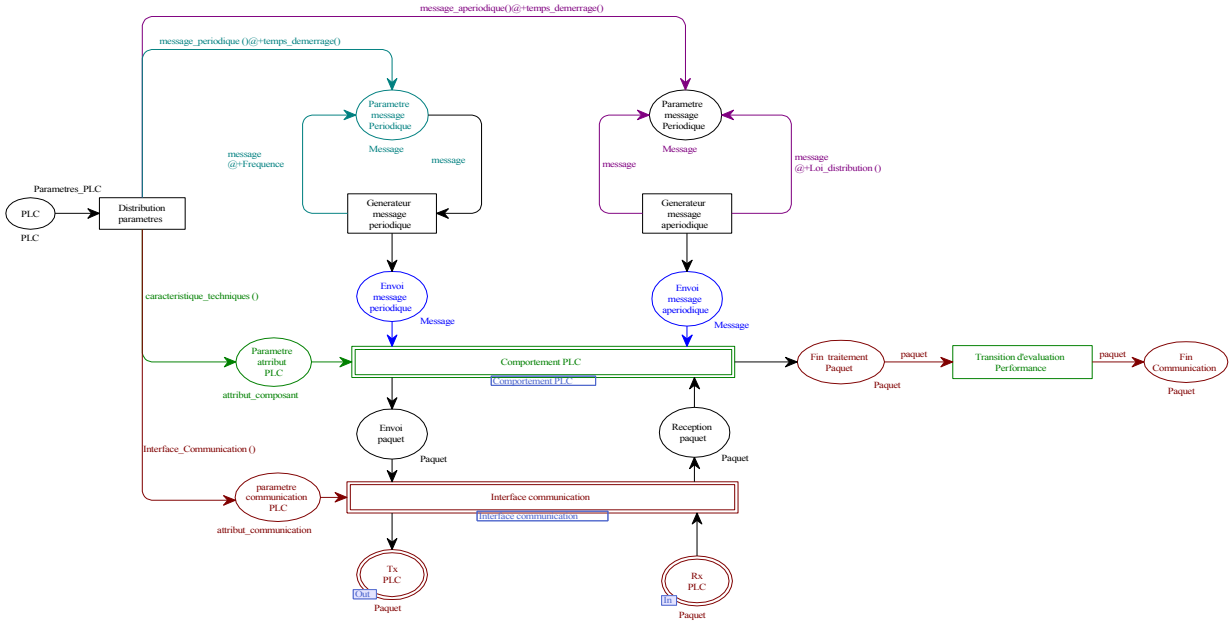


Figure 44: Modèle générique d'un PLC

Le modèle associé à la transition de substitution *Comportement_PLC* (Figure 45-a), qui représente le modèle comportemental de l'automate, contient lui-même trois transitions de substitution :

- la transition de substitution *processeur_automate* décrit l'enchaînement des tâches réalisées par le processeur : traitement des différents paquets envoyés par le SCADA et les clients, envoi de manière périodique des messages de mise à jour aux équipements terminaux et enfin envoi de manière apériodique des messages vers les autres automates afin de synchroniser les différents procédés. La capacité de traitement de ces différents paquets dépend de la taille du buffer interne de l'automate ainsi que de ses spécificités. Les différents attributs du processeur et des buffers sont renseignés respectivement par les fonctions d'incidences arrières *init_processeur_PLC ()* et *init_Buffer_PLC ()* qui extrait des couleurs définies par la fonction *caractéristiques_techniques()* les couleurs respectivement associées au processeur et au buffer.
- les transitions de substitution *Traitement_reception* et *Traitement_envoi* correspondent aux traitements effectués sur les messages respectivement reçus et générés par le processeur de l'automate.

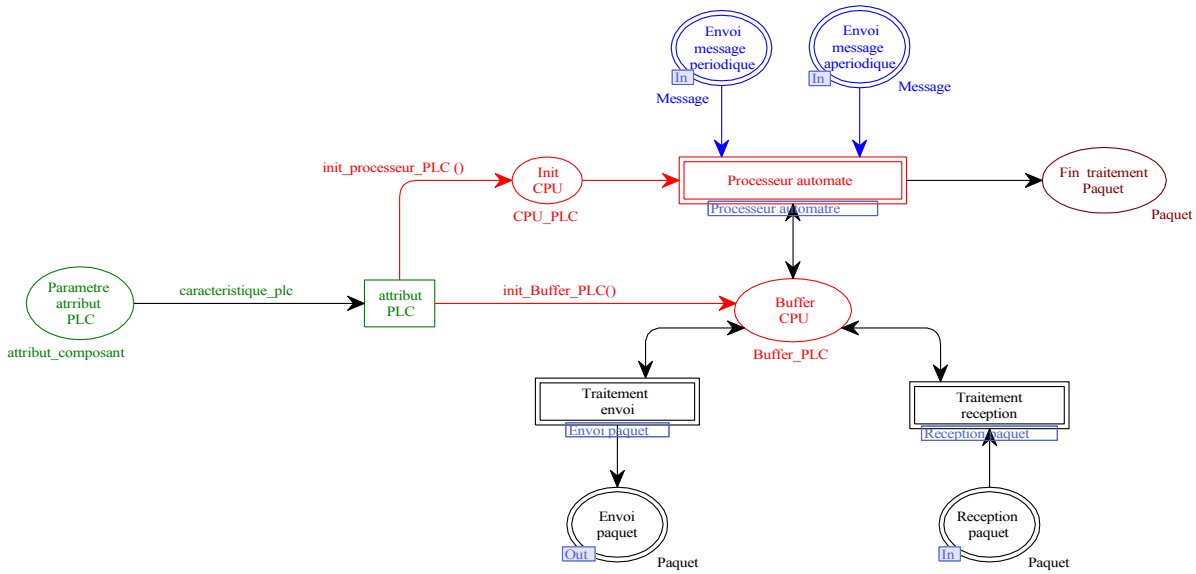
Si le message reçu par le processeur est une réponse à un message apériodique généré par l'automate lui-même alors, une fois traité celui-ci est transféré vers la place *fin_traitement*.

Le modèle associé à la transition de substitution *Interface_communication* représente le modèle de l'interface de communication de l'automate (Figure 45-b). Celle-ci permet l'aiguillage des paquets entrant et reçu au niveau de la place *Rx_PLC* vers les transitions de substitution *port_coupleur_communication* et *port_embarque* qui représentent les modèles respectifs des ports du coupleur de communication et du port embarqué du processeur d'un automate. Cet aiguillage se fait grâce aux fonctions d'incidences arrières *init_Port_embarque() et init_coupleur()* qui apportent les informations liées à la table de commutation. Ainsi, ces fonctions ainsi que les informations contenues dans le jeton permettent d'aiguiller les jetons

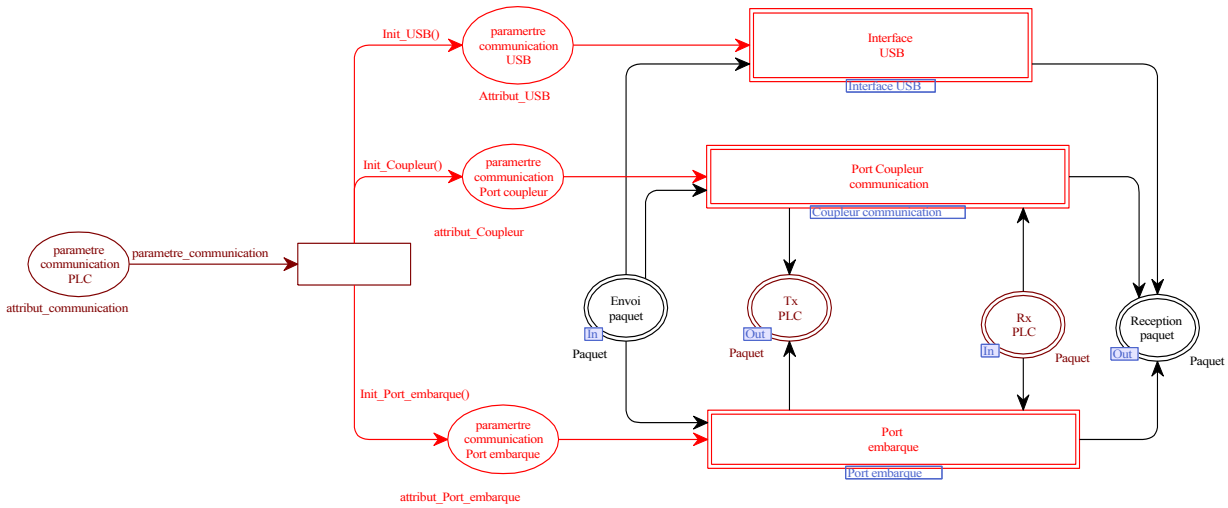
reçus à partir du réseau vers le bon port de communication de l'automate. En ce qui concerne les paquets envoyés par l'automate vers le réseau ceux-ci sont aiguillés vers la place de transmission nommée *Tx_PLC* par les différentes transitions de substitution des ports.

La transition de substitution *interface_USB* représente le modèle de l'interface USB d'un automate. Contrairement aux deux autres transitions de substitutions des ports, l'interface USB n'est pas connectée au réseau donc non connectée aux places de transmission et de réception.

Pour des raisons de confidentialité, les modèles détaillés de ces différentes transitions de substitution (*processeur_automate, Traitement_reception, Traitement_envoi, port_coupleur_communication, port_embarque,* et *interface_USB*) ne sont pas fournies dans le cadre de ce mémoire.



(a) Modèle du comportement d'un PLC



(b) Modèle de l'interface de communication d'un PLC

Figure 45: Modèle de comportement et interface de communication PLC

Lors de la définition des modèles de composant, la définition des ensembles de couleurs caractérisant l'ensemble de leurs paramètres doit être précisée. A des fins d'illustration, l'ensemble des déclarations relatives à l'instanciation et au paramétrage du processeur PLC a été présentées ci-dessous.

Déclaration couleurs processeur automate

Colset ID *= INT* ; */*Identifiant unique du PLC. */*

Colset AppL *= INT; /* durée pour traiter la logique automate. */*

Colset USB = *INT ; /* Nombre de requêtes traité pour une connexion en USB par scan */*

Colset CoupleurComm = *INT; /*Nombre de requêtes traité pour une connexion avec le coupleur de communication par scan*/*

Colset PortServeur *= INT; /*Nombre de requêtes traité pour une connexion en port embarqué du processeur*/*

Colset Buffer = *INT ; /* taille buffer */*

/ Déclaration paramètre processeur*/*

Colset processeur = *record idPLC:ID * app_Logic:AppL * usb_Scan:USB * Coupleur_Scan:CoupleurComm * serveur_Scan * PortServeur *buffer*Buffer;*

Déclaration des paramètres des processeurs d'un PLC

Comme indiqué précédemment pour le modèle « constructeur », cette définition se limite à la déclaration des ensembles de couleurs et ne contient en aucun cas les valeurs des couleurs (variables) qui dépendent de la définition de l'architecture à évaluer. L'assignation de ces valeurs sera réalisée ultérieurement à partir des informations fournies par la partie *Client*.

2.1.2 Gestion des simulations à l'aide de *Server CPN*

Le serveur de simulation *Server CPN* a été développé avec le langage de programmation Standard ML (SML) par (Jensen, et al., 2007). Dans la pratique, le simulateur exécute un fichier, possédant une extension *.sml*, issu d'une compilation à partir d'un fichier source au format ML possédant une extension *.ml*. Par défaut, ce fichier au format ML est généré par l'éditeur de CPN Tools, compilé puis exécuté par le simulateur *Server CPN*.

Dans notre cas, un traitement préalable est nécessaire afin d'intégrer au modèle « *constructeur* » et à la bibliothèque de composants, les informations relatives à l'instanciation des composants au sein de l'architecture à évaluer, à leurs différents paramètres ainsi qu'aux performances à mesurer (moniteurs et paramètres de simulation). Pour l'instant, nous faisons l'hypothèse que l'ensemble de ces informations peut être fourni par la partie *Client* de notre prototype sous la forme d'un fichier XML. Pour prendre en compte ces informations d'instanciation et de paramétrage, il est nécessaire de modifier le flot de tâches réalisées de manière native par CPN Tools de telle sorte à adapter le processus de compilation.

Dans un premier temps, à l'issue de la phase de configuration et de personnalisation de notre prototype, le modèle « *constructeur* » et la bibliothèque de composants sont exportés de *Design CPN* dans un fichier au format ML (extension *.nj*). Ce fichier constituera une base unique pour toutes les simulations réalisées quel que soit l'architecture et les performances à évaluer.

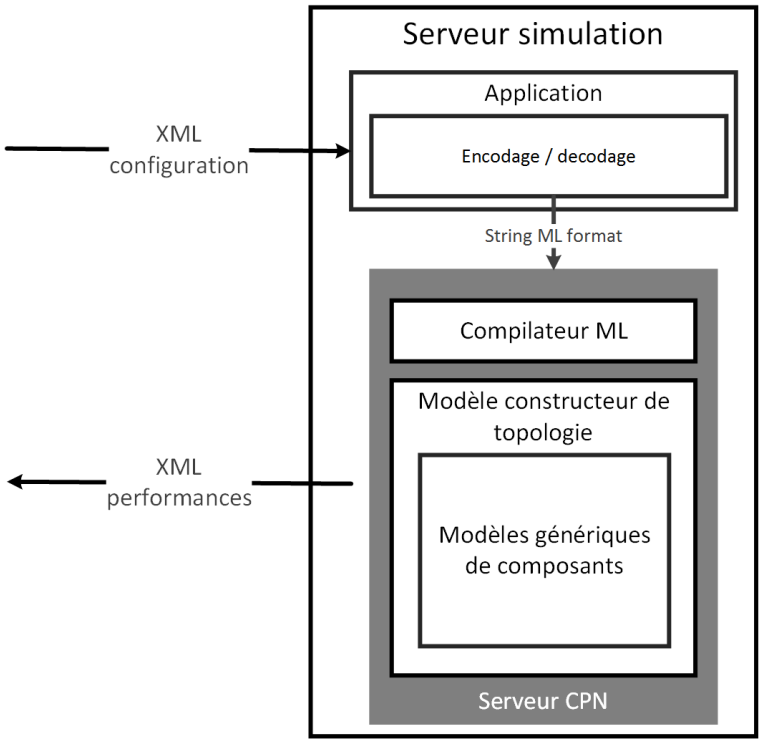


Figure 46: Architecture du serveur de simulation

Par la suite, pour chaque nouvelle architecture à évaluer, notre serveur de simulation, dont l'architecture fonctionnelle est donnée par la Figure 46, suivra le processus suivant :

- le fichier XML, contenant les données d'instanciation et de paramétrage des composants ainsi que les performances à évaluer, est transformé au format ML (avec une extension *.ml*) intelligible par le compilateur ; cette phase est nommée *encodage/décodage* dans la Figure 46 ; le fichier ML ainsi généré devra contenir l'assignation des couleurs (variables) pour les fonctions d'incidences arrière pour ainsi que l'assignation des paramètres des moniteurs ; à titre d'exemple, le Tableau 2 donne un exemple d'assignation en langage ML pour la fonction d'incidence arrière *init_parametres()* associée au composant PLC ;
- le compilateur SML procède à la compilation de deux fichiers au format ML (un contenant le modèle « constructeur » et la bibliothèque de composants, l'autre contenant les paramètres) pour générer un exécutable ;
- le simulateur du Server CPN exécute le fichier issu de la compilation et réalise les simulations requises.

En sortie, le serveur de simulation produit un fichier au format XML contenant les observations réalisées par les moniteurs pour tous les jeux de simulations effectuées. Ce fichier XML devra être transmis à la partie Client de notre prototype.

Variables et fonction d'incidence arrière du processeur
*/*déclaration variable d'instanciation et de paramétrage d'un PLC avec ses valeurs */*
Val var _PLC *=*
ref [PlcCommPeriodique({idPLC=10, idDevice=100, typeMessage= scanning, interArrivee= 50}), PlcCommNoPeriodique({idPLC=10, destPlc=11, typeMessage= PLC_request, interArrivee = sendRate}), PLCProcesseur ({idPLC=10, app_Logic=50, usb_Scan=4, Coupleur_Scan =8, serveur_Scan = 8, buffer=16}), PlcInterfaceCommunication ({idComposant=11, idPort=1})] ;
*/*Paramètres de la fonction init_parametres pour le PLC */*
Fun init_parametres () *= (! var_PLC)*

Tableau 2 : Assignation des paramètres de la fonction init_parametres() pour le PLC

2.2 Le *Client*

La partie *Client* du prototype constitue l'interface destinées aux ingénieurs en avant-vente. Cette partie repose sur l'outil PSX builder de Schneider Electric et plus particulièrement de son interface de visualisation PSX viewer. Sur cette interface (Figure 47), nous avons développé deux modules offrant :

- une interface utilisateur composée de deux modules en charge :
  - d'éditer/visualiser une architecture, de définir les caractéristiques des flux de communication et de définir les performances que l'on souhaite évaluer ;
  - de visualiser les performances mesurées par le serveur de simulation ;
- une interface de transcription permettant de :
  - produire le fichier de sortie XML exploité par le serveur de simulation pour instancier et paramétrer le modèle d'architecture à partir des informations fournies par le module d'édition;
  - transformer le fichier d'entrée XML produit par le serveur de simulation dans un format intelligible par le module de visualisation des performances.

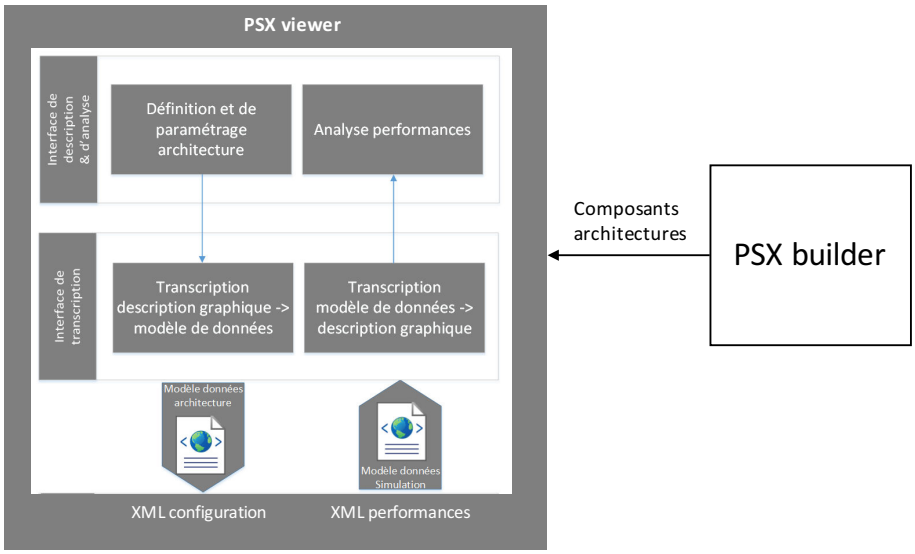
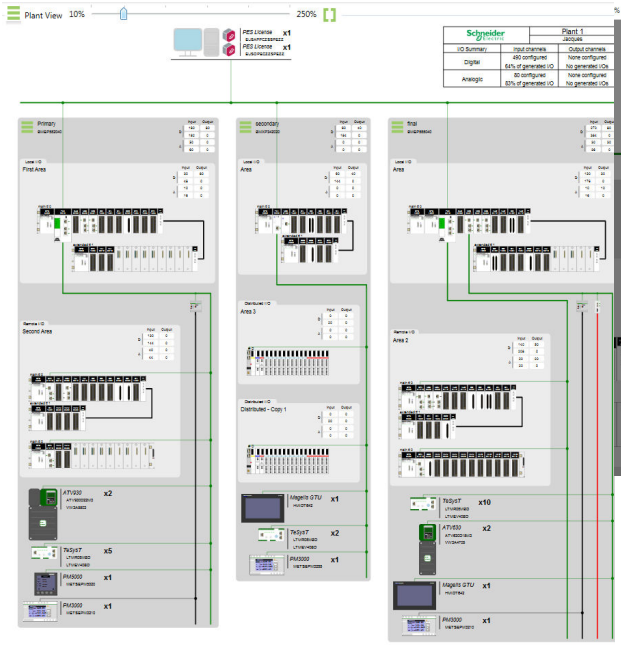
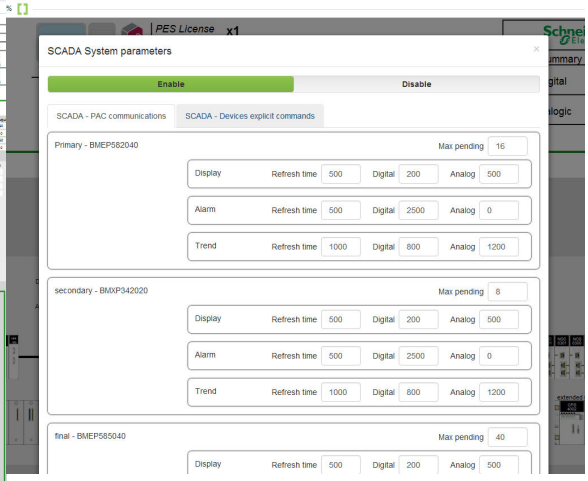


Figure 47: Architecture du *Client*

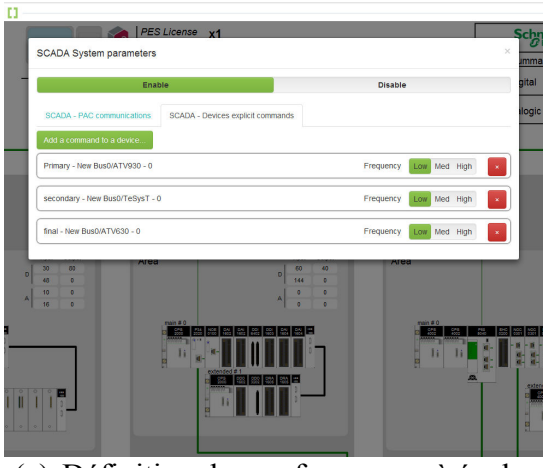
A titre d'exemple, la Figure 48 présente (a) l'interface de visualisation de la proposition d'architecture élaborée par les ingénieurs en avant-vente, (b) l'interface de définition des paramètres de fonctionnement (exemple avec le composant SCADA), (c) l'interface de définition des performances à évaluer (*enable* pour les communications apériodiques que l'on souhaite évaluer, *disable* pour les autres), (d) l'interface de définition des paramètres de simulation. A noter que l'interface de définition des performances à évaluer sert également à saisir les caractéristiques des flux de communication apériodiques aléatoires selon trois niveaux de fréquence (*low*, *medium*, *high*).



(a) Définition de l'architecture à évaluer



(b) Définition des paramètres de fonctionnement SCADA et du flux de communication périodique



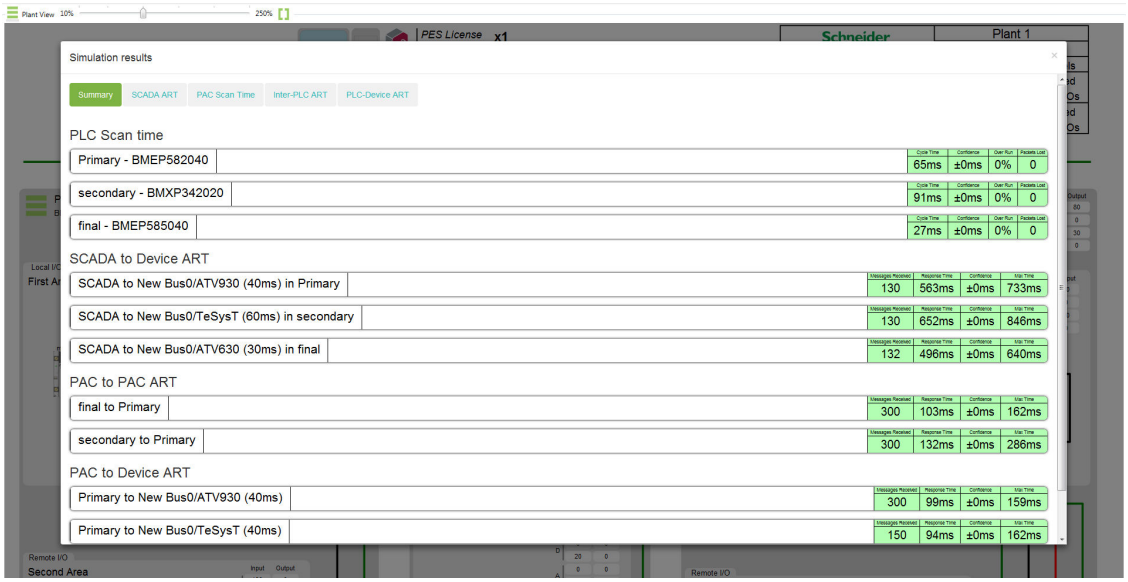
(c) Définition des performances à évaluer et paramétrage des flux de communication apériodique

(d) Paramètres de simulation

Figure 48: Interface de description et d'analyse des architectures

Le deuxième rôle de l'interface de description et d'analyse est l'affichage des résultats de simulation. En effet une fois que la configuration de l'architecture a été simulée par le serveur de simulation et que les relevés statistiques des moniteurs renvoyés au format XML ont été traités, alors l'interface affiche les résultats de simulation afin qu'ils puissent être analysés et interprétés par l'ingénieur en avant-vente. Cet affichage se fait sous la forme de tableau avec les valeurs minimales, maximales et moyenne (Figure 49 a), et sous la forme de courbe de tendance en fonction du temps de simulation (Figure 49 b).

(a) Courbes de tendance



(b) Tableau de synthèse

Figure 49: Affichage graphique des performances de l'architecture

L'interface de transcription a deux rôles. Premièrement, elle transforme, grâce à un parseur, les représentations graphiques des architectures en modèle de données XML. Celui-ci va regrouper l'ensemble des informations liées à l'architecture à simuler (nombre de composants,

types, topologies, paramètres d'entrées, moniteurs, temps de simulation etc..). Le deuxième rôle est la transformation de modèles XML contenant les performances vers une représentation graphique. Cette transformation s'effectue toujours via un parseur et éventuellement via des fonctions de post-traitement.

2.3 L'interface de communication entre Client et Serveur de simulation

Les deux composants de l'outil d'estimation de performances, c'est-à-dire le *Client* et le *Serveur de Simulation* communiquent grâce à un mécanisme de type client-serveur à travers le protocole TCP/IP (Fall & Stevens, 2011) qui représente la couche de transport de données (Figure 50).

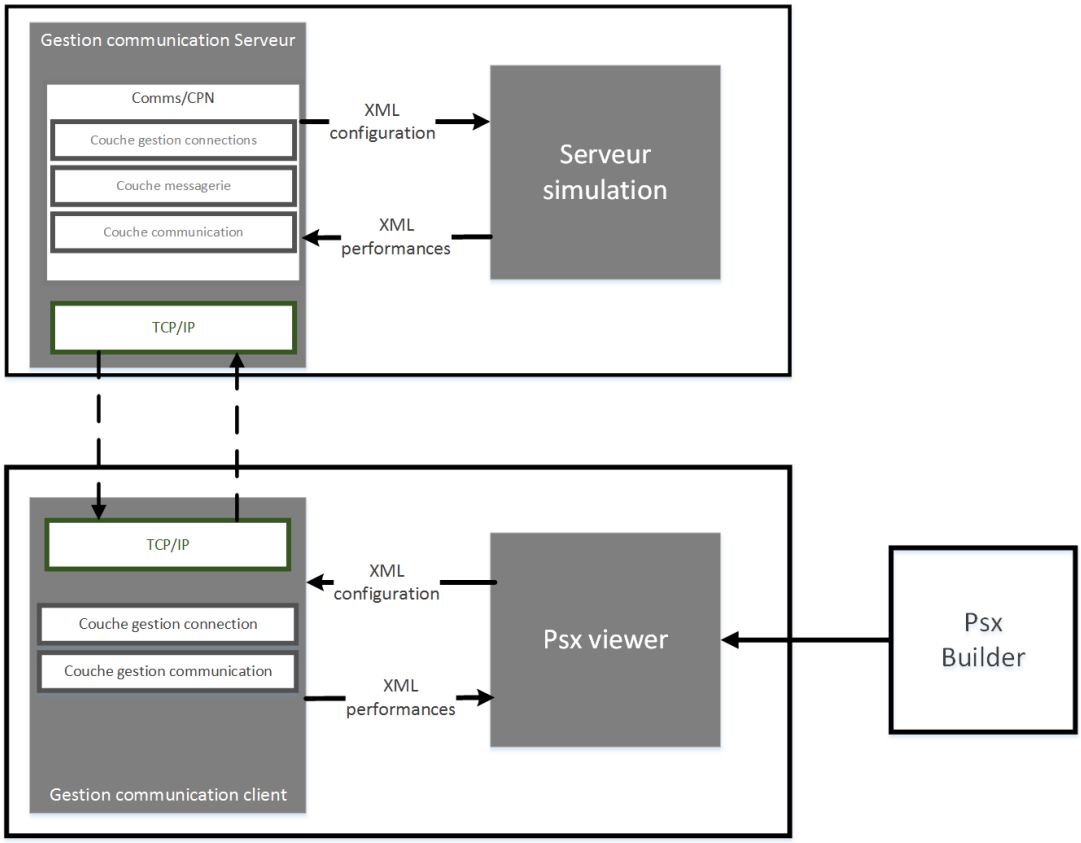


Figure 50: Vue générale de l'architecture fonctionnelle du prototype

Le choix d'une implémentation en TCP/IP pour gérer les échanges entre *Client* et *Serveur de simulation* a été motivé, d'une part, par le fait que la transmission de données entre ces deux entités doit être robuste et fiable afin d'éviter les éventuelles pertes de connections et de données (le protocole TCP/IP de manière native intègre un mécanisme de contrôle de données et de détection d'erreur) et, d'autre part, par le fait qu'un grand nombre d'applications et d'équipements intègrent de manière native ce protocole. En cas d'évolution ou de changement des choix techniques pour la définition des architectures au niveau du client, aucun effort supplémentaire n'est à fournir pour permettre la communication entre le serveur et le client, tant que l'application client intègre le protocole TCP.

Conformément au protocole TCP/IP, deux couches dédiées à la gestion de la connexion et de la communication sont déployées sur le *Client* et le *Serveur de simulation*. La couche de gestion de la connexion initie la communication à la demande du *Client*, maintient la communication suivant les échanges client-serveur et fermera la connexion à l'initiative du client ou du serveur. La couche gestion de la communication a pour rôle de gérer l'émission et la réception des paquets échangés entre *Client* et *Serveur de simulation*.

En revanche, et pour tenir compte de la spécificité du langage ML dans lequel a été développé le simulateur CPN, une couche supplémentaire doit être intégrée sur le *Serveur de simulation* (couche de messagerie). Cette couche fournit des fonctions génériques permettant la transcription des données d'envoi et de réception en séquences d'octets (vice-versa). La couche messagerie intégrée dans le *Serveur de simulation* est la librairie Comms/CPN développée par (Gallasch & Kristensen, 2001). Elle permet au simulateur CPN de communiquer avec une application externe quel que soit le type tant qu'elle intègre le protocole TCP/IP.

En résumé, avec le protocole TCP/IP et Comms/CPN, une application externe peut interagir avec notre *Serveur de simulation*. Elle peut ainsi envoyer le fichier de configuration XML de l'architecture qui est encapsulé dans une séquence de données. Celui-ci sera décodé par un module du Comms/CPN afin de présenter le fichier à l'application serveur pour traitement.

3 Application sur un cas étude

L'application de la démarche de génération d'un modèle d'architecture à des fins d'évaluation de performance poursuit un double objectif. Il s'agit :

- d'une part, de démontrer la faisabilité de l'approche en déroulant ses différentes étapes pour une architecture donnée : définition de l'architecture, génération de son modèle et évaluation de ses performances ; la validation de l'approche sera réalisée par comparaison entre les performances évaluées via notre approche et les performances mesurées sur l'architecture déployée en laboratoire.
- d'autre part, de mettre en évidence l'intérêt de l'approche relatif à l'effort de modélisation minime nécessaire pour évaluer les performances de plusieurs architectures répondant au cahier des charges.

3.1 Présentation du cas d'étude

Le cas d'étude porte sur la définition d'une architecture de C/C pour une usine de traitement d'eaux usées pour une ville de 20000 habitants comme l'indique la Figure 51. Le fonctionnement de cette usine met en œuvre 3 procédés principaux :

- le « *Primary Treatment*» : pré-traitement des eaux usagées,
- le « *Secondary Treatment*» dont le but est de faire un traitement biologique des eaux usées,
- le « *Chemical Treatment* » dont le but est de faire un traitement chimique.

Dans le cadre de ce cas d'étude, nous nous focalisons sur le procédé du « *primary clarifier* » qui se décompose à son tour en 3 sous-procédés :

- Le « *lifting/screening* » (Pompage et extraction) dont le rôle est de pomper les eaux usées et d'extraire les déchets solides lourds,
- Le « *grease & sand removal* » (suppression de la graisse et du sable) pour le traitement des déchets solides légers et de la boue,
- Le « *primary clarifier* » (traitement primaire) l'unité de prétraitement des eaux usées.

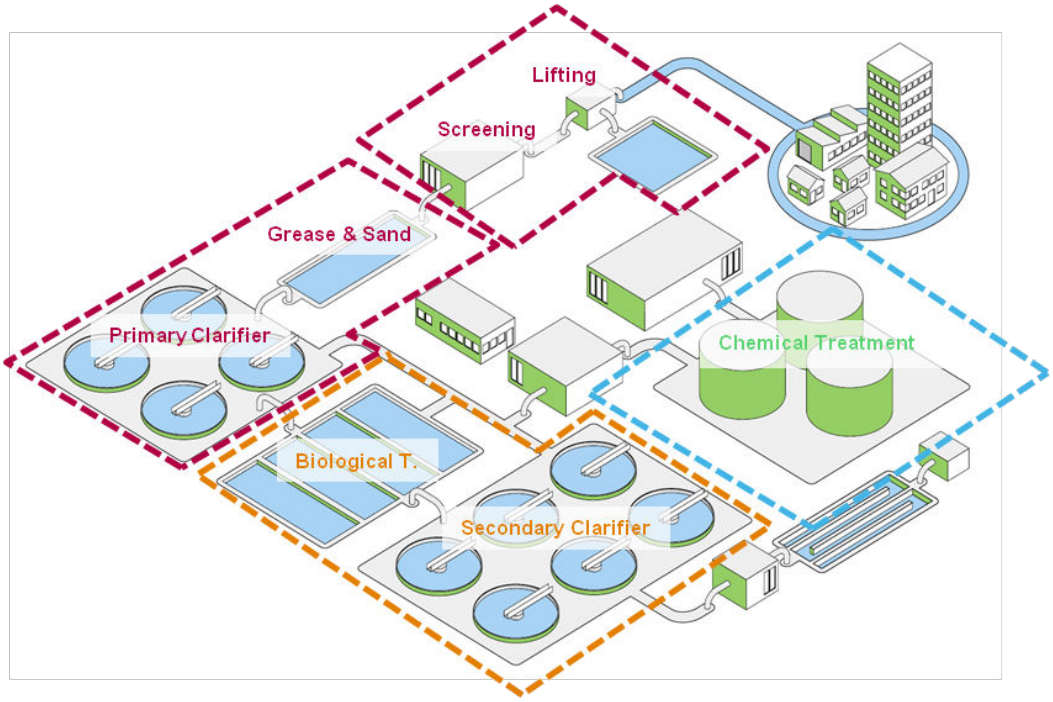


Figure 51: Usine de traitement des eaux usées

L'architecture fonctionnelle de conduite et de contrôle-commande se décompose en trois niveaux :

- un niveau « équipement de terrain » composé essentiellement des actionneurs et des capteurs de l'installation ;
- un niveau « contrôle-commande » réflexe contenant des algorithmes de régulation ou de protection
- un niveau « conduite » où des opérateurs de supervision analyse l'état des différents procédés de transformation et effectuent, si besoin, des actions à destination soit du niveau « contrôle-commande » soit du niveau « équipement de terrain ».

Les principales exigences « client » exprimées en termes de performances sont les temps de réponse aller/retour entre une action réalisée par un opérateur sur un poste de supervision à destination d'un équipement de terrain et la réponse de ce dernier. Les temps de cycle des dispositifs présents au niveau « contrôle-commande » constituent également un indicateur que le client souhaite observer.

3.2 Illustration de la démarche proposée sur le cas d'étude

L'objectif de cette section est d'illustrer les différentes étapes de notre approche :

- description des données d'entrée (architecture proposée et performances à évaluer),
- instanciation et paramétrage du modèle d'architecture,
- évaluation des performances.

3.2.1 Description des données d'entrées

Comme définit précédemment, les données d'entrées sont constituées, d'une part, par la définition de l'architecture de contrôle-commande, de ses paramètres de fonctionnement ainsi que des flux de communications et, d'autre part, par les performances attendues ainsi que les paramètres relatifs à la simulation. La définition de ces données d'entrée est à la charge des ingénieurs en avant-vente et repose sur leur retour d'expérience.

3.2.1.1 Architecture de contrôle commande proposée

L'architecture de C/C pouvant répondre au besoin du procédé du « *Primary treatment* » de l'usine de traitement d'eaux usées est proposée par la Figure 52.

Equipements composant l'architecture

L'architecture proposée comprend 3 niveaux d'équipements. Le niveau dédié à la supervision, au monitoring et à la conduite de l'installation est composé d'un serveur d'acquisition de données et de supervision (SCADA) et de deux interfaces hommes/machines HMI dédiées à la surveillance et le contrôle au niveau du terrain et communiquant avec les automates programmables via un port USB.

Le niveau dédié au contrôle-commande de l'installation comprend 3 unités fonctionnelles correspondant à chaque sous- procédé contenant chacune un automate programmable de l'offre Schneider Electric. La commande des sous-procédés « *lifting/screening* » et « *primary clarifier* » est supportée par un automate de la série Modicon M580 mais ayant des gammes de processeurs différentes donc des puissances et des spécifications techniques différentes (processeur de la gamme 20 pour le premier, processeur de la gamme 40 pour le second) ; la commande du sous-procédé « *grease & sand removal* » est assurée par un automate de la série Modicon M340 associé à un processeur de la gamme 20.

Le dernier niveau contient les équipements terminaux dont la gestion est assurée par une des unités fonctionnelles de contrôle-commande décrites ci-dessus. Les équipements terminaux sont de quatre types:

- Tesys ; équipement de contrôle moteur pour gérer et contrôler les moteurs de l'usine,
- STB : Module d'interface d'entrées / sorties pour se connecter aux procédés,
- Altivar : Variateurs de vitesses pour des moteurs ou des vannes,
- Power management ; contrôle de la puissance et de l'énergie, pour la gestion du voltage et de la consommation électrique.

Ainsi pour l'unité fonctionnelle gérée par l'automate M580-20 celle-ci est constituée de 3 équipement terminaux (device) de types Tesys, 2 STB, 2 Altivar, 1 HMI et 1 power

management. Concernant l'unité fonctionnelle géré par l'automate M340-20 celle-ci est quand elle composée de 2 STB, 2 Tesys, 3 Altivar. Enfin L'unité fonctionnelle gérée par le M580-40 est composée de 6 Tesys, 2 Altivar, 2 STB, 1 HMI et 1 power management.

Enfin, l'ensemble des équipements des différents niveaux sont connectés et communiquent à travers un réseau Ethernet.

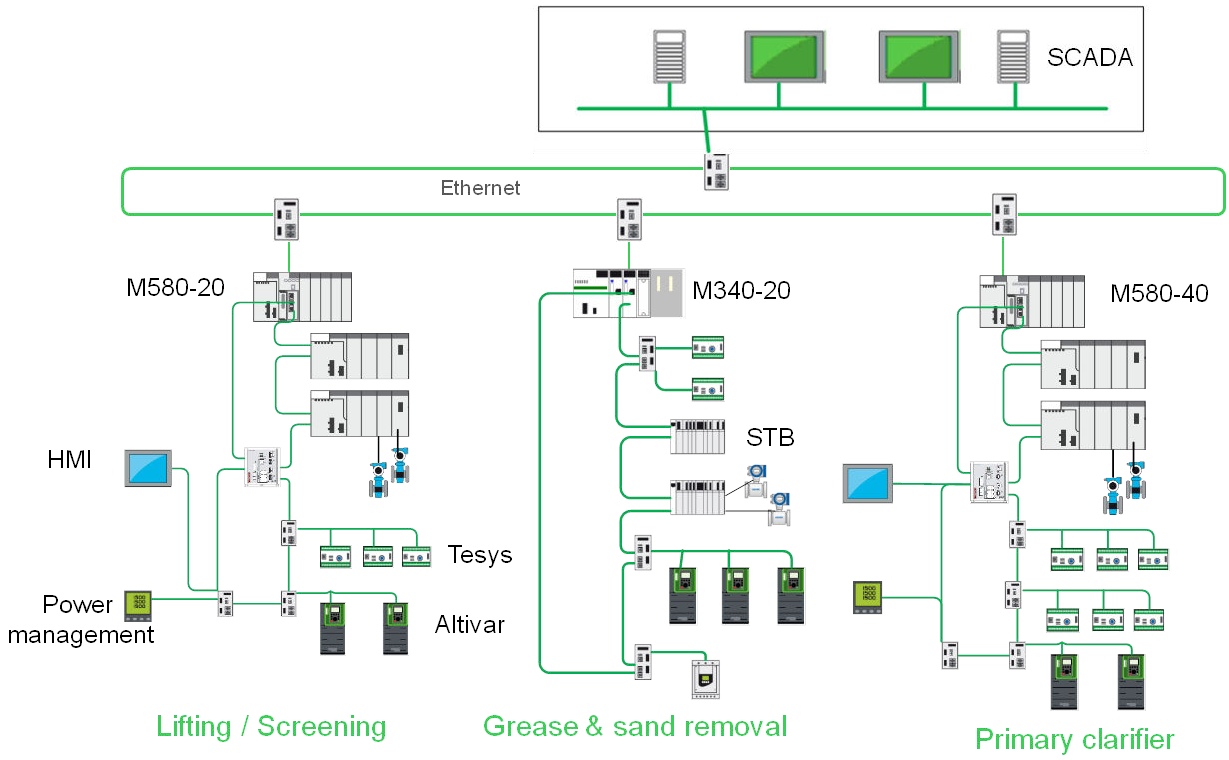


Figure 52: Architecture de C/C définie pour le 'traitement primaire'

Caractéristiques de fonctionnement

En ce qui concerne le SCADA, ses caractéristiques techniques sont relatives d'une part, à la mise à jour des informations du procédé et, d'autre part à la génération d'actions et de commande à destination du procédé.

Pour la mise à jour des informations de procédé, un serveur de supervision SCADA possède 3 groupes de services dont chacun a un rôle spécifique pour la gestion et la supervision de l'architecture :

- le groupe *Alarm* gère les remontées d'alarmes déclenchées par les équipements terminaux (franchissement de seuil) ; attention, ce service correspond à des alertes, par exemple relatives à un franchissement de seuil et n'aucun rapport avec les messages apériodiques de type *alarm* correspondant, par exemple, à des défaillances et surveillées dans les moniteurs du chapitre 3 ;
- le groupe *Display* gère la visualisation des états des composants,
- le groupe *Trend* gère l'ensemble des courbes de tendances de l'architecture.

Deux paramètres caractérisent le fonctionnement de ces services. Il s'agit, d'une part, de la période de rafraichissement (associée à l'émission périodique de requêtes permettant de collecter l'information auprès des automates) et d'autre part, à un ordonnanceur de requêtes qui regroupe les requêtes à émettre en fonction des capacités des automates.

Les paramètres des systèmes SCADA, renseignés via une interface telle que celle présentée en Figure 48 b, sont donnés dans les deux tableaux suivants.

Scada groupes

Groupe	Nombre requetes	Periode (millisecondes)
Alarme	8	500
Display	5	500
Trend	5	1000

Tableau 3: Paramètres des groupes d'échange sur un SCADA

Max Pending

Automates	Nombre de requetes
M580 - 20	16
M580 - 40	32
M340-20	8

Tableau 4: Valeur des Max Pending pour le SCADA

Les commandes émises depuis le SCADA vers le procédé sont émises au fil de l'eau de manière apériodique et sont non affectées par le *Max Pending*. Elles sont représentées dans le modèle CPN du composant SCADA par une distribution uniforme de probabilité avec trois valeurs de paramètres possibles correspondant à trois niveaux de fréquence : low, medium et high. L'ingénieur en avant-vente doit donc définir les niveaux souhaités via l'interface présentée sur la Figure 48-d.

Les principaux paramètres caractérisant les automates programmables sont donnés dans le tableau suivant. Le temps alloué à l'application correspond au temps mis par l'automate durant son cycle pour traiter le programme utilisateur d'automatisation (section 3.1.2 chapitre 1). Ce temps est grossièrement évalué par l'ingénieur en avant-vente sur la base d'un calcul simple tenant compte du type de traitement à effectuer (traitement de variables logiques ou traitement de variables analogiques) et du nombre d'entrées/sorties associés au traitement. L'ingénieur en avant-vente fixe également les temps de scrutation (temps de *scan*) des équipements terminaux associés à un automate. Les paramètres correspondant au cas d'étude sont fournis dans le tableau suivant.

Paramètre automates

Automates	Application logique (milliseconde)	Tesys (milliseconde)	STB (milliseconde)	Altivar (milliseconde)
M340-20	50	25	35	50
M580-20	100	50	75	100
M580-40	150	75	100	150

Tableau 5 : Paramètres PLC

L'ingénieur en avant-vente devra également spécifier si l'automate fonctionne en mode cyclique ou périodique (avec un temps de cycle fixe et préalablement défini). Pour notre cas d'étude, les trois automates fonctionnent en mode cyclique.

A noter que le temps de cycle de l'automate inclut, en supplément de ces temps relatifs au *scan* des équipements et au traitement de l'application, les temps nécessaires pour exécuter des tâches de lecture des entrées, d'écriture des sorties ou de services exécutés en mode serveur. Ces temps, inconnus, dépendront du contexte applicatif et seront donc évalués via la simulation du modèle d'architecture et les moniteurs de temps de cycle.

Enfin, chaque gamme d'automate possède des caractéristiques techniques, telles que le nombre de requêtes pouvant être traitées par cycle en fonction de leur provenance. Sur l'exemple de l'architecture de commande du traitement des eaux usées, les automates peuvent recevoir des requêtes en provenance du port *serveur* envoyées par le SCADA, du port USB en provenance des interfaces HMI et du coupleur Ethernet en provenance des équipements terminaux. Les requêtes ne pouvant être traitées dans un cycle automate, Si le nombre de requêtes dépasse les capacités de traitement définis pour un automate donné, les requêtes non traitées sont stockées dans un buffer ayant une capacité limitée. Ainsi, le traitement de certaines requêtes peut être différé au cycle suivant, ou, dans le pire des cas, certaines requêtes peuvent être perdues. Ces paramètres, donnés dans le tableau le Tableau 6 pour notre cas d'étude, influent donc de manière significative sur les performances.

A noter que ces différents paramètres sont directement dépendant de la gamme d'automate sélectionnée. Ils sont stockés dans l'outil *PSX builder* et ne peuvent donc être modifiés par les ingénieurs en avant-vente.

Type	USB	Serveur Scan	Coupleur Scan	Buffer
M340-20	4	8	8	16
M580-20	4	16	16	32
M580-40	4	32	16	50

Tableau 6: Spécifications techniques des PLC

En ce qui concerne les équipements de terrain, nous avons fait l'hypothèse, dans le cadre de cette étude, qu'ils ne disposent d'aucun paramètre de fonctionnement. En d'autres termes, l'ensemble de leurs caractéristiques comportementales sont directement intégrées dans les modèles CPN de chaque équipement et ne sont pas paramétrables.

3.2.1.2 Les performances à évaluer

Les performances à évaluer dans ce cas d'étude sont les suivantes :

- le temps aller et retour entre l'émission d'une commande au niveau du SCADA, son traitement par un groupe d'équipements terminaux et son retour (conformément aux exigences définies dans la section 3.1.1 du chapitre 1),
- le temps de cycle des automates programmables de l'architecture.

Conformément à la définition des moniteurs proposées au chapitre 3, l'ensemble des communications apériodiques est surveillé (en mode aller/retour pour les messages de type commande mais également en mode aller simple pour les messages de type alarme). La définition des performances, que les ingénieurs en avant-vente souhaitent évaluer, permettra

donc de filtrer les résultats de simulation pour ne fournir sur l'interface de visualisation des performances que les performances ayant un intérêt.

Pour notre cas d'étude, les performances à mesurer sont données dans le tableau suivant et définies sur notre prototype d'évaluation des performances via l'interface présentée en Figure 48-d (en déclarant les communications que l'on veut mesurer comme *enable* et les autres comme *disable*).

Performances évaluées

Type performances	équipement concernés	équipement passerelle
Tems aller Retour SCADA - device - SCADA	SCADA - Tesys	automate M340-20
Tems aller Retour SCADA - device - SCADA	SCADA - STB	automate M340-20
Tems aller Retour SCADA - device - SCADA	SCADA - Altivar	automate M340-20
Tems aller Retour SCADA - device - SCADA	SCADA - Tesys	automate M580-20
Tems aller Retour SCADA - device - SCADA	SCADA - STB	automate M580-20
Tems aller Retour SCADA - device - SCADA	SCADA - Altivar	automate M580-20
Tems aller Retour SCADA - device - SCADA	SCADA - Tesys	automate M580-40
Tems aller Retour SCADA - device - SCADA	SCADA - STB	automate M580-40
Tems aller Retour SCADA - device - SCADA	SCADA - Altivar	automate M580-40
Temps de cycle automates	M340-20	aucun
Temps de cycle automates	M580-20	aucun
Temps de cycle automates	M580-40	aucun

Tableau 7: Performances à évaluer

De manière similaire, les temps de cycle automate sont systématiquement observés par les moniteurs « temps de cycle » définis au chapitre 3. L'ingénieur en avant-vente a la possibilité de demander (ou pas) l'affichage de ces performances sur son interface de visualisation. Dans notre cas d'étude, tous les temps de cycle sont remontés sur cette interface.

3.2.2 Génération du modèle d'architecture

L'ensemble de tous les paramètres de fonctionnement et des performances à évaluer, saisi via la partie *Client* de notre prototype par les ingénieurs en avant-vente, génère un fichier XML qui sera transmis au Serveur de Simulation via les procédures d'échange et d'encodage définies en section 2.3.

La première étape consiste donc à compiler le fichier de paramètres définissant les variables (couleurs) avec le modèle « *constructeur* » présenté en section 2.1.1.1 de ce chapitre. Les valeurs des paramètres permettront d'instancier et de paramétrer, via l'ensemble des fonctions d'incidence arrière définies au chapitre 3, le modèle CPN correspondant à l'architecture de commande du traitement primaire des eaux usées.

Le tir des transitions du modèle « *constructeur* » aboutira au dépôt :

- d'un jeton dans la place située en amont de la transition de substitution SCADA (un seul équipement de type SCADA dans l'architecture de la Figure 52;
- de deux jetons dans la place située en amont de la transition de substitution CLIENT (correspondant aux deux HMI de l'architecture de la de la Figure 52;

- de trois jetons dans la place située en amont de la transition de substitution PLC (correspondant aux trois automates de l'architecture de la de la Figure 52;
- de vingt-six jetons dans la place située en amont de la transition de substitution DEVICE (correspondant à l'ensemble des équipements terminaux de l'architecture de la Figure 52;

Pour rappel, le modèle « constructeur » implanté dans une première version de notre prototype ne présente, par hypothèse, qu'un seul type d'équipement par famille (seule la transition init_famille présente plusieurs arcs en sortie).

Les valeurs des couleurs associés à ces différents jetons sont définies par les différentes fonctions d'incidence arrière en fonction des paramètres renseignés à l'aide de variables CPN conformément à la méthode d'assignation proposée sur le Tableau 2. Cette phase d'instanciation et de paramétrage suit en tout point le scénario présenté au chapitre 3 par la Figure 35.

La simulation peut alors se poursuivre en accord avec les paramètres relatifs au nombre de réplication (nombre d'histoires simulées, dans notre cas 10) et les critères d'arrêt relatifs à la durée de chaque histoire (dans notre cas 15 minutes). Ces paramètres de simulation permettent d'obtenir des résultats avec un niveau de précision suffisant (mesuré notamment via l'intervalle de confiance à 95%).

3.2.3 Evaluation des performances

Les moniteurs définis au chapitre 3 permettent alors d'obtenir les performances brutes (temps minimum, temps maximum, valeur moyenne, intervalle de confiance à 95%). A titre indicatif, mes valeurs obtenues sont données dans les colonnes 4 à 7 du Tableau 8 (la colonne 8 intitulée CI fournit l'intervalle de confiance à 95%) mais n'ont pas d'intérêt intrinsèque dans le cadre de ce mémoire. Elles ne seront donc discutées qu'à des fins de validation de nos modèles dans la section suivante.

3.3 Validation du modèle en plate-forme expérimentale

Afin de valider les principes fondateurs de notre proposition ainsi que les modèles CPN intégrés dans notre prototype (modèle « *constructeur* » et bibliothèque de composants), une comparaison est effectuée entre :

- les résultats en termes de performances évalués par notre prototype sur la base d'un modèle CPN de l'architecture de la Figure 52,
- les résultats en termes de performances mesurées sur une plate-forme expérimentale reproduisant l'architecture de la Figure 52;

Pour que ces résultats soient comparables, les tests en plate-forme ont été réalisés avec des paramètres identiques : 10 campagnes de test ayant chacune une durée de 15 minutes. Dans le même ordre d'idée, les sollicitations réalisées manuellement sur le système SCADA l'ont été avec une fréquence compatible avec les distributions uniformes implantées sur le simulateur.

Le Tableau 8 ci-dessous reprend les résultats obtenus pour les trois performances évaluées : les colonnes 4 à 7 correspondent aux résultats évalués sur le simulateur tandis que les colonnes 8 à 10 fournissent les résultats mesurés lors des tests en plate-forme.

Performances évaluées									
			Résultat outils (en millisecondes)				résultat en laboratoire (en milliseconde)		
Type performances	équipement concernés	équipement passerelle	Min	Max	Avrg	Ci	Min	Max	Avrg
Tems aller Retour SCADA - device - SCADA	SCADA - Tesys	automate M340-20	330	507	419.5	0.8	341	518	430.5
Tems aller Retour SCADA - device - SCADA	SCADA - STB	automate M340-20	340	577	459.5	0.1	348	583	466.5
Tems aller Retour SCADA - device - SCADA	SCADA - Altivar	automate M340-20	355	607	482	0.15	401	613	508
Tems aller Retour SCADA - device - SCADA	SCADA - Tesys	automate M580-20	405	807	607	0.79	408	809	609.5
Tems aller Retour SCADA - device - SCADA	SCADA - STB	automate M580-20	430	857	644.5	0.86	436	861	649.5
Tems aller Retour SCADA - device - SCADA	SCADA - Altivar	automate M580-20	455	907	682	1.08	452	910	682
Tems aller Retour SCADA - device - SCADA	SCADA - Tesys	automate M580-40	480	957	719.5	1.63	478	886	683
Tems aller Retour SCADA - device - SCADA	SCADA - STB	automate M580-40	505	1007	757	1.86	501	942	722.5
Tems aller Retour SCADA - device - SCADA	SCADA - Altivar	automate M580-40	554	1107	831.5	2.03	519	1016	768.5
Temps de cycle automates	M340-20	aucun	55	80	68.5	0.05	56	87	72.5
Temps de cycle automates	M580-20	aucun	105	142	124.5	0.75	108	144	127
Temps de cycle automates	M580-40	aucun	155	228	192.5	0.09	149	206	178.5

Tableau 8: Performances de l'architecture en simulation et en test de laboratoire

La comparaison des résultats entre les valeurs fournies par le simulateur et ceux fournis par les mesures en laboratoire montre que pour les automates d'entrée de gamme de type M340 et M580-20, les résultats fournis par le simulateur sont tout à fait comparables à ceux obtenus en plate-forme. En effet, le simulateur fournit une valeur moyenne des temps de réponse légèrement plus optimiste que celle obtenue en plate-forme avec toutefois un écart inférieur à 5%. La valeur maximale des temps de réponse présente, quant à elle, un écart inférieur à 2%. Ces écarts minimes peuvent s'expliquer par le paramétrage de nos modèles CPN de composants, et en particulier des temps de traitement des requêtes au niveau du modèle des automates. En effet, les paramètres de la distribution uniforme retenus dans nos modèles représentent un temps de traitement compris entre 1 et 3 millisecondes alors que, lors des tests effectués, cet intervalle se situe plutôt entre 1 et 5 millisecondes. Ce résultat montre que l'ensemble des paramètres de fonctionnement intégrés dans nos différents modèles CPN de composants devra peut-être être affiné lors de l'industrialisation de notre prototype sur la base de nouveaux tests en plate-forme.

Pour l'automate haut de gamme M580-40, l'écart entre performances évaluées par le simulateur et performances mesurées en plate-forme est plus important puisqu'il peut atteindre 7,5% pour les valeurs moyennes et 8,2% pour les valeurs maximales. Cet écart s'explique par une évolution entre les automates utilisés en fin de thèse pour les tests en plate-forme et ceux décrits dans les documentations techniques à notre disposition en début de thèse. En effet, dans les nouvelles versions du M580-40, un deuxième cœur vient suppléer le cœur principal lors de fortes charges, ce qui améliore de manière significative les performances dans ce cas de figure. En présence d'une charge moyenne, le deuxième cœur est inutilisé et n'a donc aucune incidence sur les temps de traitement des requêtes.

Cet écart met en exergue un point particulièrement important de notre approche. Dans les modèles CPN des composants de type PLC, le mécanisme de décharge sur un deuxième cœur en présence de fortes charges n'a pas été modélisé. Ce comportement complexe étant très différent de ceux mis en œuvre pour les M340 et M580-20, le M580-40 devra faire l'objet d'un modèle CPN spécifique. L'impact en termes d'évolution de notre prototype serait le suivant :

- ajout d'un modèle CPN dédié au M580-40 dans la bibliothèque de composants
- modification du modèle « constructeur » pour y intégrer deux types de composants pour la famille PLC (deux arcs en sortie de la transitions *init_PLC* du modèle « *constructeur* ».

Même si quelques écarts ont pu être constatés (qui pourront être réduits par un réglage plus fins des paramètres de nos modèles), ils restent néanmoins relativement faibles, ce qui constitue un argument en faveur de l'approche proposée et des modèles CPN développés.

3.4 Evaluation de multiples architectures

L'application de notre prototype d'estimation des performances, sur le cas d'étude du traitement des eaux usées, a montré que l'effort de modélisation que les ingénieurs en phase d'avant-vente doivent consentir pour évaluer les performances d'une architecture de contrôle-commande se limite maintenant à la définition de celle-ci sur une interface graphique. Même si l'architecture de la Figure 52 reste de taille modeste, la démarche proposée est applicable pour des architectures beaucoup plus complexes.

A titre d'exemple, faisons l'hypothèse que les ingénieurs en avant-vente souhaitent évaluer une seconde architecture, beaucoup plus complexe, pour répondre au cahier des charges du traitement primaire des eaux usées. Cette seconde architecture est définie sur l'interface *Client* de notre prototype et présentée sur la Figure 53. Elle fait apparaître un découpage en unité fonctionnelle différent de la première version, conserve le découpage en trois niveaux (conduite, contrôle-commande et équipements de terrain) mais les équipements présents dans ces trois niveaux sont beaucoup plus nombreux.

Pour évaluer cette nouvelle architecture l'ingénieur en avant-vente devra renseigner les informations suivantes sur son interface graphique (de manière identique à la démarche présentée dans les sections précédentes) : description de l'architecture, de ses paramètres, des flux de communications, des performances à évaluer et des paramètres de simulation.

Cet effort de modélisation est le seul qui sera déployé par les ingénieurs en avant-vente puisque la procédure de simulation et d'évaluation des performances est ensuite complètement automatisée :

- transformation du modèle de données graphiques au format XML,
- connexion du client au serveur de simulation et envoi des paramètres au serveur de simulation,
- compilation des modèles CPN munis de leurs paramètres,
- exécution en simulation des modèles CPN (et notamment de la phase d'instanciation et paramétrage),

- relevés statiques des moniteurs et génération des résultats au format XML,
- envoi au *Client* et présentation des résultats de simulation.

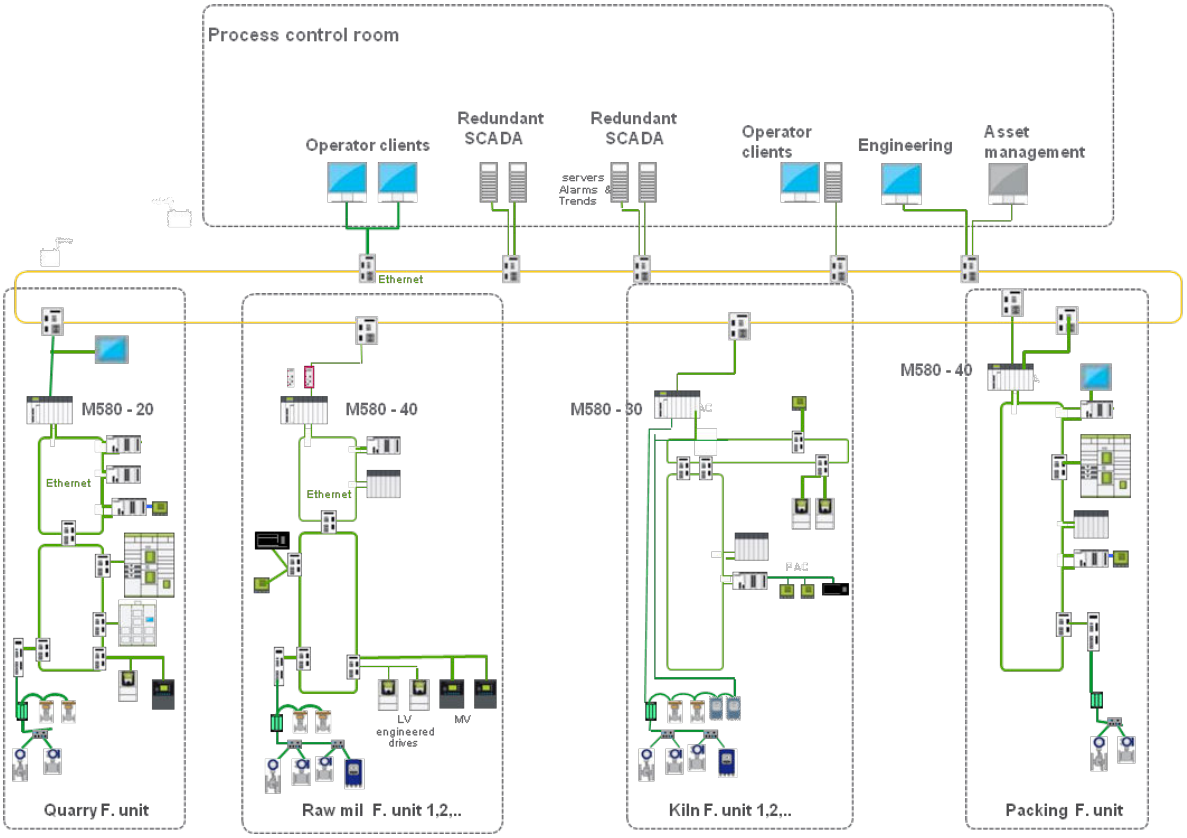


Figure 53: Architecture complexe de C/C

Cette procédure est identique à celle mise en œuvre précédemment puisque le modèle « *constructeur* » demeurera inchangé et seuls les paramètres d'instanciation et de paramétrage contenus dans le fichier XML issu de l'interface *Client* seront différents. A titre d'exemple, le tir des transitions du modèle « *constructeur* » conduira au dépôt de deux jetons dans la place située en amont de la transition de substitution SCADA (puisque cette architecture contient deux SCADA) dont les paramètres correspondront à ceux définis par les ingénieurs en avant-vente. Le même raisonnement peut être appliqué pour les composants CLIENT, PLC et DEVICE.

En revanche, si l'architecture à évaluer fait apparaître de nouveaux types de composants non disponibles en bibliothèque, il sera nécessaire d'enrichir cette dernière et d'adapter, si besoin, le modèle « *constructeur* ». Comme nous l'avons montré en section 3, cette démarche sera peut-être nécessaire pour créer un composant M580-40. En tout état de cause, ces modifications de la bibliothèque et/ou des modèles de composants ne sont en aucun cas du ressort des ingénieurs en avant-vente. Elles rentrent dans le cas de la maintenance des versions de l'outil d'estimation de performances et requièrent l'appel à un expert CPN. Cependant, les études industrielles développées dans le cadre de notre travail chez Schneider Electric montrent que cet effort de maintenance devrait rester très raisonnable.

4 Conclusion

Ce chapitre a présenté le prototype d'évaluation des performances des architectures de contrôle-commande qui supporte la démarche automatisée de construction des modèles CPN présentée au chapitre 3 de ce mémoire. L'utilisation de ce prototype sur un cas d'étude industriel fourni par Schneider Electric a permis :

- de démontrer la faisabilité de l'approche proposée et de valider les modèles CPN intégrés au prototype, notamment au travers de la comparaison entre les performances évaluées par simulation et les performances réelles mesurées sur une plate-forme de laboratoire,
- de mettre en évidence l'intérêt principal de notre proposition qui réside dans la réduction de l'effort consenti par les ingénieurs en avant-vente pour évaluer différentes architectures puisque celui-ci est limité à la description sur une interface graphique des architectures et de leurs paramètres,
- de préciser le périmètre d'applicabilité de notre proposition contraint par la topologie d'architecture, qui doit être compatible avec la structure de notre modèle « *constructeur* », et par la disponibilité en bibliothèque des modèles CPN des composants impliqués dans l'architecture; nous avons néanmoins montré que ce périmètre pouvait être aisément élargi par l'intégration au modèle « *constructeur* » de nouveaux types de composants (et de leurs ensembles de couleurs associés.

De manière générale, les résultats obtenus confortent les principes de notre proposition (notamment de notre stratégie d'instanciation et de paramétrage d'un modèle d'architecture via un modèle « *constructeur* ») et constituent une base scientifique solide pour l'industrialisation de notre prototype. Ils ont néanmoins montré que les modèles CPN que nous avons développés devraient certainement être affinés ultérieurement, notamment au niveau des paramètres associés aux distributions de probabilités, pour mieux prendre en compte la réalité des applications industrielles.

Conclusion & perspectives

L'évaluation des performances des architectures de contrôle-commande en phase d'avant-vente représente un enjeu important pour dimensionner des solutions au plus juste des besoins des utilisateurs. Le compromis entre niveau de garantie des performances et coût de l'architecture est en effet un vecteur essentiel pour le gain de certains marchés ou appels d'offre.

Dans ce contexte, l'entreprise Schneider Electric, fournisseur et intégrateur de solutions d'automatismes a souhaité se doter d'un outil d'aide au dimensionnement des architectures et notamment d'évaluation de leurs performances temporelles. L'objectif était clair : fournir un outil permettant d'évaluer rapidement les performances d'une architecture de contrôle-commande dans un laps très court et en mobilisant le minimum de ressources tant humaines que financières. Dans le contexte particulier de l'avant-vente, cet objectif a rapidement fait émerger trois corolaires :

- l'évaluation de performances devait pouvoir être réalisée par un personnel ayant la maîtrise des solutions d'architecture sans être toutefois être expert dans les outils et modèles nécessaires à sa mise en œuvre,
- un grand nombre d'architectures devaient pouvoir être évaluées dans un temps court afin de trouver les meilleures réponses aux exigences client tout en générant le maximum de plus-value pour l'entreprise,
- la connaissance nécessaire pour réaliser les évaluations devait bien sûr comprendre une description de l'architecture matérielle (composants, réseaux, …) mais également contenir de l'information, même très succincte et entachée d'incertitudes, relative à l'architecture fonctionnelle (listes des fonctions de contrôle-commande, listes des entrées/sorties en termes d'actionneurs/capteurs, …) et à son contexte d'usage (notamment des flux de communication, des sollicitations externes, …).

L'étude de la littérature a montré que de nombreux travaux ont été réalisés dans le domaine de l'évaluation des performances à l'aide de techniques d'analyse formelle ou bien par simulation de Monte-Carlo. Pour un certain nombre d'entre eux, les informations nécessaires à l'élaboration des modèles ou à la vérification des performances n'étaient pas disponibles en phase d'avant-vente. Pour d'autres, la complexité des modèles à construire pour des architectures de taille significative nous est apparue comme un frein à leur acceptation en phase d'avant-vente. Notre choix s'est donc porté sur les Réseau de Pétri colorés, temporisés et hiérarchiques (CPN) définis par (Jensen, 1994) qui allient pouvoir d'expression avec modularité (hiérarchique) et compacité (couleurs) des modèles. L'analyse de la littérature abondante, consacrée à l'usage des CPN en milieu industriel, montre néanmoins que l'élaboration des modèles CPN reste une tâche peu compatible avec les contraintes de la phase avant-vente puisque couteuse en temps et nécessitant le recours à un expert.

Dans ce contexte, notre contribution principale a donc porté sur la génération automatique des modèles CPN représentant le comportement d'une architecture de contrôle-commande ainsi que les observateurs permettant l'évaluation de ses performances. La démarche proposée repose sur trois éléments principaux :

- une bibliothèque de modèles CPN génériques et standardisés représentant le comportement des composants présents dans une architecture,
- un modèle « *constructeur de topologie* » qui structure les architectures selon différents niveaux hiérarchiques et qui met en œuvre un ensemble de fonctions permettant l'instanciation et la paramétrage du modèle CPN,
- un ensemble d'observateurs génériques permettant l'évaluation de temps de réponse aller-retour, de temps de réponse aller simple ou de temps de cycle PLC.

Le développement d'un prototype implantant cette démarche, et son application à quelques exemples d'architectures fournis par Schneider Electric, ont permis de mettre en évidence l'intérêt de notre contribution pour les ingénieurs en avant-vente. Car, l'évaluation d'une architecture est réalisée de manière complètement transparente pour les utilisateurs finaux. En effet, une interface graphique permet à ces derniers de saisir la description de l'architecture matérielle, des paramètres de fonctionnement connus et des performances à évaluer. Ces informations sont automatiquement traitées par le prototype (au travers des fonctions d'instanciation et de paramétrage du modèle « *constructeur* ») pour lancer des simulations de Monte-Carlo et fournir aux utilisateurs finaux une estimation statistique des performances.

Au terme de ces travaux, plusieurs perspectives peuvent être envisagées en termes d'industrialisation du prototype développé mais aussi d'élargissement des champs et domaines couverts par notre approche.

Même si la comparaison entre les performances évaluées par le prototype celles mesurées sur une plate-forme de laboratoire a donné d'excellents résultats permettant de valider notre contribution, l'industrialisation du prototype nécessite toutefois encore quelques développements pour :

- **consolider** la bibliothèque de modèles :
    - en validant les modèles de composant à partir de tests approfondis en laboratoire pour obtenir des réglages de paramètres (notamment de temporisation) plus fins permettant d'augmenter la précision des résultats,
    - en enrichissant la bibliothèque avec les différents composants de l'offre Schneider-Electric; ce travail, qui peut être long (mais facilité par la généricité de nos modèles), est indispensable pour mettre à disposition des ingénieurs en avant-vente un outil qui leur permettra de concevoir une large gamme d'architectures,
- **optimiser** les temps de calculs lors des simulations de Monte-Carlo :
    - en cherchant en paralléliser les exécutions ou les réplications,
    - en optimisant le code des fonctions ML contenues dans nos modèles ; à titre d'exemple, les fonctions d'instanciation et de paramétrage du modèle « *constructeur* » sont exécutées à chaque simulation alors qu'une seule fois pourrait suffire à condition de conserver le marquage du modèle après initialisation et d'exécuter les autres simulations à partir de ce marquage.

Au-delà des développements nécessaires à l'industrialisation de notre prototype, plusieurs perspectives de recherche permettraient d'enrichir notre contribution.

Lors de cette étude, la modélisation des comportements relatifs aux réseaux de communication est une version simplifiée des topologies réseaux (Switch, Routeur, composants d'infrastructure, etc…) et de leurs protocoles. En effet le réseau a été simplement modélisé comme un commutateur induisant des retards. Pourtant, l'étude de la littérature a montré que la charge des composants réseau peut se révéler pénalisante pour les performances temporelles du système pouvant même rendre celui-ci complétement instable. Même, si cet impact tant à être de moins en moins fréquent, la modélisation de l'ensemble des ressources partagées, des files d'attentes dont sont constitués les composants réseau d'une architecture de contrôle-commande, des protocoles de communication permettraient d'obtenir une évaluation plus fine des performances.

Par ailleurs, nous nous sommes concentrés dans cette étude sur la modélisation des performances temporelles. Il serait intéressant d'ajouter également l'évaluation des performances de charge des composants critiques des architectures, comme par exemple le taux de disponibilité moyen des ressources et le taux d'occupation moyen des files d'attente. Ces informations de charge constituent en effet un critère pertinent à prendre en compte lors de la conception des architectures, notamment pour comprendre et expliquer certaines performances temporelles.

Enfin, la caractérisation des incertitudes liées au contexte d'avant-vente a été traduite sous la forme de distributions de probabilités. Ces incertitudes, traduisant dans certains cas une absence d'informations, dans d'autre une imprécision, pourraient être appréhendées sous d'autres formes : ensembliste, théorie des possibilités, fonction de croyances. Leur intégration dans notre prototype nécessiterait des développements complémentaires pour définir les primitives requises sur CPN-Tools.

Références bibliographiques

Abdelli, A., Serrai, W., Mokdad, L. & Hammal, Y., 2015. Time Petri Nets for performance evaluation of composite web services architectures. *IEEE Symposium on Computers and Communication (ISCC),* pp. 122-127.

Addad, B., 2011a. *Évaluation analytique du temps de réponse des systèmes de commande en réseau en utilisant l'algèbre (Max,+),* Cachan: École normale supérieure de Cachan.

Addad, B., 2011a. *Évaluation analytique du temps de réponse des systèmes de commande en réseau en utilisant l'algèbre (Max,+),* Cachan: École normale supérieure de Cachan.

Addad, B., Amari, S. & Lesage, J.-J., 2010. Analytic Calculus of Response Time in Networked Automation Systems. *IEEE Transactions on Automation Science and Engineering,* 7(4), pp. 858 - 869.

Addad, B., Amari, S. & Lesage, J.-J., 2011b. Client-Server Networked Automation Systems Reactivity: Deterministic and Probabilistic Analysis. *IEEE Transactions on Automation Science and Engineering,* 8(3), pp. 540 - 548.

Al-Jaar, R. & Desrochers, A., 1990. Performance evaluation of automated manufacturing systems using generalized stochastic Petri nets. *IEEE Transactions on Robotics and Automation,* 6(6), pp. 621-639.

Amari, S., Demongodin, I., Loiseau, J. J. & Martinez, C., 2012. Max-Plus Control Design for Temporal Constraints Meeting in Timed Event Graphs. *IEEE Transactions on Automatic Control,* 57(2), pp. 462-467.

Ammour, R. & Amari, S., 2015. Modelling and temporal performances evaluation of networked control systems using (max,+) algebra. *International Journal of Systems Science,* 46(1), pp. 18-30.

Anderson, E. & Richard, L. O., 1987. Perspectives on behavior-based versus outcome-based salesforce control systems. *The Journal of Marketing,* pp. 76-88.

Auroy, M., 2000. *Elabortaion des Schémas de Procedes Industriels.* 1 éd. Paris: Société Française du Génie des procédés.

Baarir, S. et al., 2009. The GreatSPN tool: recent enhancements. *CM SIGMETRICS Performance Evaluation Review,* 36(4), pp. 4-9.

Baccelli, F., Cohen, G., Olsder, G. J. & Quadrat, J.-P., 1992. *Synchronization and Linearity An Algebra for Discrete Event Systems.* New York: Wiley & Sons.

Baier, C., Katoen, J.-P. & Larsen, K. G., 2008. *Principles of model checking.* Cambridge: MIT press.

Barger, P., 2003. *Evaluation et validation de la fiabilité et de la disponibilité des systèmes d'automatisation à intelligence distribuée,* Nancy: Université de Nancy 1.

Barger, P., Schön, W. & Bouali, M., 2009. A study of railway ERTMS safety with colored Petri nets. *The European Safety and Reliability Conference (ESREL'09),* Volume 2, pp. 1303-1309.

Bauer, H., Scharbarg, J.-L. & Fraboul, C., 2009. Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network. *IEEE Conference on Emerging Technologies & Factory Automation,* pp. 1-8.

Bauer, H., Scharbarg, J.-L. & Fraboul, C., 2010. Improving the worst-case delay analysis of an AFDX network using an optimized trajectory approach. *IEEE Transactions on Industrial informatics,* 6(4), pp. 521-533.

Béla, L., 2013. *Process Control: Instrument Engineers' Handbook.* Oxford: Butterworth-Heinemann.

Ben Hedia, B., Jumel, F. & Babau, J.-P., 2005. Formal evaluation of quality of service for data acquisition systems. *Proc. of FDL 05.*

Berthomieu, B. & Diaz, M., 1991. Modeling and verification of time dependent systems using time Petri nets. *IEEE transactions on software engineering,* 17(3), p. 259.

Berthomieu, B., Ribet, P.-O. & Vernadat, F., 2004. The tool TINA–construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research,* 42(14), pp. 2741-2756.

Berthomieu, B. & Vernadat, F., 2006. Réseaux de Petri temporels: méthodes d'analyse et vérification avec TINA. *Traité IC2 Système temps réel.*

Bordbar, B. & Anane, R., 2005. An architecture for automated QoS resolution in wireless systems. *19th International Conference on Advanced Information Networking and Applications (AINA'05),* pp. 774 - 779.

Bouillard, A., Gaujal, B., Lagrange, S. & Eric, T., 2008. Optimal routing for end-to-end guarantees using Network Calculus. Performance Evaluation. *Performance Evaluation,* 65(11), pp. 883-906.

Boyer, M., Navet, N., Olive, X. & Thierry, E., 2010. The PEGASE project: precise and scalable temporal analysis for aerospace communication systems with Network Calculus. *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation,* pp. 122-136.

Bozga, M. et al., 1998. Kronos: A Model-Checking Tool for Real-Time Systems. *International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems,* pp. 298 - 302.

Brahimi, B., 2007. *Proposition d'une approche intégrée basée sur les réseaux de Petri de Haut Niveau pour simuler et évaluer les systèmes contrôlés en réseau,* Nancy: Université Henri Poincaré-Nancy I.

Brinzei, N., Deleuze, G., Villaume, N. & Pétin, J.-F., 2014. Common cause failures modelling by means of coloured Petri nets for dependability assessment of a control system of nuclear power plant. *European Safety and Reliability Conference ESREL,* pp. 2121-2129.

Cervin, A. et al., 2003. How does control timing affect performance?. *Control Systems Magazine,* 23(3), pp. 16-30.

Chang, C.-S., 2000. *Performance guarantees in communication networks.* 1 éd. Berlin: Springer Science & Business Media.

Charara, H., 2007. *Évaluation des performances temps réel de réseaux embarqués avioniques,* Toulouse: Institut National Polytechnique de Toulouse.

Charara, H., 2007. *Évaluation des performances temps réel de réseaux embarqués avioniques,* Toulouse: institut national polytechnique de toulouse.

Charara, H., Scharbarg, J.-L., Ermont, J. & Fraboul, C., 2006. Methods for bounding end-to-end delays on an AFDX network. *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*.

Christensen, S., Jørgensen, J. B. & Lars, M. K., 1997. Design/CPN—A computer tool for coloured Petri nets. *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems,* pp. 209-223.

Clarke, A., Grumberg, O. & Peled, D., 1999. *Model Checking.* 1 éd. Boston: MIT press.

Clarke, E. et al., 2001. Progress on the state explosion problem in model checking. *Informatics,* pp. 176 - 194.

Cohen, G., 1993. *Two-dimensional domain representation of timed event graphs,* Belgique: Summer School on Discrete Event Systems.

Cohen, G., Gaubert, S. & Quadrat, J.-P., 1996. Kernels, images and projections in dioids. *Proceedings of WODES'96,* pp. 151-158.

Cottet, F. & Grolleau, E., 2005. *Systemes temps réel de contrôle-commande-Conception et implémentation: Conception et implémentation.* 1 éd. Paris: Dunod.

Cruz, R., 1991. A calculus for network delay part 1 Network elements in isolation. *IEEE Transactions on information theory,* 37(1), pp. 114-131.

Davey, B. & Priestley, H., 2002. *Introduction to lattices and order.* 1 éd. Cambridge: Cambridge university press.

De Figueiredo, J. C. & Kristensen, L., 1999. *Using coloured Petri nets to investigate behavioural and performance issues of TCP protocol,* Aarhus: Department of Computer Science Aarhus University.

Denis, B., 1994. *Aide à la conception d'architectures de conduite des systèmes de production,* NANCY: Université Henri Poincaré-Nancy I.

Dugan, J., Trivedi, K., Geist, R. & Nicola, V., 1984. *Extended Stochastic Petri Nets: Applications and Analysis,* Wisconsin: Wisconsin Univ-Madison Motor Behavior.

Estévez, E., Marcos, M. & Orive, D., 2007. Automatic generation of PLC automation projects from component-based models. *The International Journal of Advanced Manufacturing Technology,* 35(6), pp. 527-540.

Fall, K. & Stevens, R., 2011. *TCP/IP illustrated, volume 1: The protocols.* 1 éd. Boston: addison-Wesley.

Felser, M., 2005. Real-time ethernet-industry prospective. *Proceedings of the IEEE,* 93(6), pp. 1118-1129.

Gallasch, G. & Kristensen, L. M., 2001. Comms/CPN: A Communication Infrastructure for External Communication with Design/CPN. *3rd Workshop on Practical Use of Coloured Petri Nets and the CPN Tools.,* pp. 79-93.

Georges, J.-P., 2005. *Systèmes contrôlés en réseau: évaluation de performances d'architectures Ethernet commutées,* Nancy: Université Henri Poincaré-Nancy.

Gertler, J., 1988. Survey of model-based failure detection and isolation in complex plants. *Control Systems Magazine,* 8(6), pp. 3-11.

Ghanaim, A., Borges, G. & Georg, F., 2009. Estimating delays in networked control systems using colored Petri nets and Markov chain models. *IEEE Conference on Emerging Technologies & Factory Automation,* pp. 1-6.

Goldfarb, C. & Rubinsky, Y., 1990. *The SGML handbook.* Oxford: Oxford University Press.

Goodwin, G. C., Graebe, S. & Salgado, M., 2001. *Control system design.* New Jersey: Prentice Hall.

Greifeneder, J. & Frey, G., 2007. Probabilistic timed automata for modeling networked automation systems. *IFAC Proceedings Volumes ,* 40(6), pp. 1-6.

Grieu, J., 2004. *Analyse et évaluation de techniques de commutation Ethernet pour l'interconnexion des systèmes avioniques,* Toulouse: Institut National Polytechnique de Toulouse.

Grieu, J., Frances, F. & Fraboul, C., 2003. Preuve de déterminisme d'un réseau embarqué avionique. *10ème Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'03),* pp. 287-303.

Habib, G., 2010. *Qualité de service et qualité de contrôle d'un Système Discret Contrôlé en Réseau Sans Fil: proposition d'une approche de co-conception appliquée au standard IEEE 802.11,* Nancy: Université Henty Poincaré, Nancy 1.

Haffar, M., 2011. *Approche de Co-Simulation en vue de la validation et de l'évaluation de performance des systèmes de communication: Application à des architectures de distribution électriques,* Grenoble: Université de Grenoble.

Hardouin, L., 2004. *Sur la commande linéaire de systèmes à événements discrets dans l'algèbre (max,+),* Angers: Université Angers .

Hatley, D. & Pirbhai, I., 2013. *Strategies for real-time system specification.* 2 éd. Boston: Addison-Wesley.

Havelund, K. & Thomas, P., 2000. Model checking java programs using java pathfinder. *International Journal on Software Tools for Technology Transfer,* 2(4), pp. 366-381.

Heindl, A. & Reinhard, G., 2001. Performance modeling of IEEE 802.11 wireless LANs with stochastic Petri nets. *Performance Evaluation,* 44(1), pp. 139 - 164.

Hirel, C., Tuffin, B. & Trived, K. S., 2000. Spnp: Stochastic petri nets. version 6.0. *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation,* pp. 354-357.

Ishak, M. K., Herrmann, G. & Pearson, M., 2016. Performance evaluation using Markov model for a novel approach in Ethernet based embedded networked control communication. *Systems Conference (SysCon), 2016.*

Itsuo, H., Yamagata, K. & Tamura, H., 1991. Modeling and online scheduling of flexible manufacturing systems using stochastic Petri nets. *IEEE Transactions on Software Engineering ,* 17(2), pp. 126-132.

Jasperneite, J. & Neumann, P., 2001. Performance evaluation of switched ethernet in realtime applications. *Fourth international conference on Fieldbus systems and their applications,* pp. 144-151.

Jasperneite, J. & Neumann, P., 2001. Switched Ethernet for Factory Communication. *Emerging Technologies and Factory Automation,* pp. 205-212.

Jensen, K., 1981. Coloured Petri nets and the invariant-method. *Theoretical computer science,* 14(3), pp. 317-336.

Jensen, K., 1994. An Introduction to the Theoretical Aspects of Coloured Petri Nets. *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems),* pp. 230-272.

Jensen, K., Lars, M. K. & Wells, L., 2007. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer,* 9(4), pp. 213-254.

Juanole, G. & Gallo, L., 1995. Formal modelling and analysis of a critical time communication protocol. *Proceedings of the lSt IEEE workshop on factory communication system WFCS,* Volume 95, pp. 107-I.

Khansa, W., 1997. *Reseaux de Petri p-temporels contribution a l'etude des systemes A evenements discrets,* Annecy: Université de Savoie.

Kleinrock, L., 1976. *Queueing Systems Volume 2: Computer Applications.* 1 éd. New York: John Wiley.

Klir, G. J., 2005. *Uncertainty and information: foundations of generalized information theory.* 1 éd. New York: John Wiley & Sons.

Krakora, J. & Zdenek, H., 2004. Timed automata approach to CAN verification. *11th IFAC Symposium on Information Control Problems in Manufacturing,* April.Volume 1.

Kwiatkowska, M., Norman, G. & Parker, D., 2011. PRISM 4.0: Verification of probabilistic real-time systems. *International Conference on Computer Aided Verification,* pp. 585-591.

Labadi, K., 2005. *Contribution à la modélisation et à l'analyse de performances des systèmes logistiques à l'aide d'un nouveau modèle de réseaux de Petri stochastiques,* Troyes: Université de Troyes.

Lalouette, J. et al., 2010. Evaluation des performances du systeme de signalisation ferroviaire européen superpose au systeme français, en présence de défaillances. *17e Congrès de Maîtrise des Risques et de Sûreté de Fonctionnement Lambda-Mu'2010.*

Le Boudec, J.-Y. & Thiran, P., 2001. *Network calculus: a theory of deterministic queuing systems for the internet.* 1 éd. Berlin: Springer Science & Business Media.

Lee, K.-C. & Lee, S., 2002. Performance evaluation of switched Ethernet for real-time industrial communications. *Computer standards & interfaces,* 24(5), pp. 411 - 423.

Lenzini, L., Mingozzi, E. & Stea, G., 2008. A methodology for computing end-to-end delay bounds in FIFO-multiplexing tandems. *Performance Evaluation,* 65(11), pp. 922-943.

Limal, S., 2009. *Architectures de contrôle-commande redondantes à base d'Ethernet Industriel: modélisation et validation par model-checking temporisé,* Cachan: École normale supérieure de Cachan.

Li, X., Cros, O. & George, L., 2014. The Trajectory approach for AFDX FIFO networks revisited and corrected. *IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications,* pp. 1-10.

Lotito, P., Mancinelli, E. & Quadrat, J.-P., 2005. A Minplus Derivation of the Fundamental Car-Traffic Law. *IEEE Transactions on Automatic Control,* 50(5), pp. 699 - 705.

Maffezzoni, C., Ferrarini, L. & Carpanzano, E., 1999. Object-oriented models for advanced automation engineering. *Control Engineering Practice,* 7(8), pp. 957-968.

Marsal, G., 2006a. *Evaluation of time performances of Ethernet-based automation systems by simulation of high-level petri nets.,* Cachan: Ecole normale supérieure Cachan.

Marsal, G., Denis, B., Faure, J.-M. & Frey, G., 2006b. Evaluation of response time in Ethernet-based automation systems. *IEEE Conference on Emerging Technologies and Factory Automation,* pp. 380 - 387.

Marsan, A., Gianni Conte, M. & Balbo, G., 1984. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems (TOCS),* 2(2), pp. 93-122.

Marsan, M. A. & Chiola, G., 1986. On Petri nets with deterministic and exponentially distributed firing times. *European Workshop on Applications and Theory in Petri Nets,* pp. 132-145.

Martin, S. & Minet, P., 2006a. Schedulability analysis of flows scheduled with FIFO: application to the expedited forwarding class. *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium.*

Martin, S. & Minet, P., 2006b. Worst case end-to-end response times of flows scheduled with FP/FIFO. *International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06),* pp. 54-61.

McKinney, D., 2001. *Impact of Commercial Off-The-Shelf (COTS) Software and Technology on Systems Engineering,* Melbourne: Presentation to INCOSE.

Mekki, A., 2012. *Contribution à la Spécification et à la Vérification des Exigences Temporelles: Proposition d'une extension des SRS d'ERTMS niveau 2,* Lille: Ecole Centrale de Lille.

Mekki, A., Ghazel, M. & Toguyeni, A., 2010. Time-constrained systems validation using MDA model transformation. A railway case study. *Proceedings of the 8th International Conference of Modeling and Simulation (MOSIM'10).*

Merlin, P. M., 1974. *A study of the recoverability of computing systems,* Irvine: University of California.

Meunier, P., 2006. *Evaluation de performance d'architectures de commande de systèmes automatisés industriels,* Cachan: École normale supérieure de Cachan.

Mezini, M. & Karl, L., 1998. Adaptive plug-and-play components for evolutionary software development. *ACM Sigplan Notices,* 33(10), pp. 97-116.

Migge, J., 1999. *L'ordonnancement sous contraintes temps-réel: un modèle à base de trajectoires,* Sophia Antipolis : INRIA .

Moncelet, G. et al., 1998. Analysing a mechatronic system with coloured Petri nets. *International Journal on Software Tools for Technology Transfer (STTT),* 2(2), pp. 160-167.

Muller, P.-A. & Gaertner, N., 2000. *Modélisation objet avec UML.* 2 éd. Paris: Eyrolles.

Murata, T., 1989. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE,* 77(4), pp. 541 - 580.

Natkin, S. O., 1980. *Les reseaux de Petri stochastiques et leur application a l'evaluation des systemes informatiques,* Paris: Conservatoire National des Arts et Metiers.

Ndiaye, M., Petin, J.-F. & Camerini, J. G. J.-P., 2016. Performance assessment of industrial control system during pre-sales uncertain context using automatic Colored Petri Nets model generation. *Control, Decision and Information Technologies (CoDIT),* pp. 671-676.

Ndiaye, M., Petin, J.-F., Georges, J.-P. & Camerini, J., 2016. Practical use of coloured Petri nets for the design and performance assessment of distributed automation architectures. *International Workshop on Petri Nets and Software Engineering,* 1591(1), pp. 113-131.

Pinna, B., Babykina, G., Brinzei, N. & Pétin, J.-F., 2013. Using Coloured Petri Nets for integrated reliability and safety evaluations. *IFAC,* 43(22), pp. 19-24.

Process automation, 2013. *Select PlantStruxure reference architectures,* Carros: Schneider Electric.

Ramchandani, C., 1973. *Performance evaluation of asynchronous concurrent systems by timed Petri nets,* Cambridge: Massachusetts Institute of Technology.

Robert, J., 2012. *De l'usage d'architectures Ethernet commutées embarquées dans les lanceurs spatiaux,* Nancy: Université de Lorraine.

Rodriguez-Navas, G., Proenza, J. & Hansson, H., 2006. An Uppaal model for formal verification of master/slave clock synchronization over the controller area network. *6th IEEE International Workshop on Factory Communication Systems,* pp. 3-12.

Rony, G., Thiriet, J.-M. & Aubry, J.-F., 2013. Dependability evaluation of networked control systems under transmission faults. *IFAC Proceedings Volumes,* 39(13), pp. 1068 - 1073.

Rossi, O., 2003. *Validation formelle de programmes Ladder Diagram pour Automates Programmables industriels,* Cachan : Ecole normale supérieure Cachan.

Royer, A., 2006. *Evaluation de performances de réseaux de communication à l'aide de chaînes de Markov hybrides,* Grenoble: Institut National Polytechnique de Grenoble-INPG.

Ruel, S., 2009. *Évaluation des bornes des performances temporelles des Architectures d'Automatisation en Réseau par preuves itératives de propriétés logiques,* Cachan: École normale supérieure de Cachan.

Sünder, C. et al., 2006. Usability and Interoperability of IEC 61499 based distributed automation systems. *4th IEEE International Conference on Industrial Informatics,* pp. 31-37.

Schmitt, J., Zdarsky, F. & Fidler, M., 2008. Delay Bounds under Arbitrary Multiplexing: When Network Calculus Leaves You in the Lurch.... *The 27th Conference on Computer Communications INFOCOM 2008,* pp. 1669-1677.

Schmitt, J., Zdarsky, F. & Martinovic, I., 2006a. The disco network calculator: a toolbox for worst case analysis. *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools,* Volume 180, p. 8.

Schmitt, J., Zdarsky, F. & Martinovic, I., 2006b. *Performance bounds in feed-forward networks under blind multiplexing,* Kaiserslautern: University of Kaiserslautern.

Scholten, B., 2007. *Integrating ISA-88 and ISA-95,* Rosmalen: Ordina ISA-95 & MES competence centre.

Seno, L., Vitturi, S. & Zunino, C., 2009. Real Time Ethernet Networks Evaluation Using Performance Indicators. *IEEE Conference on Emerging Technologies & Factory Automation,* pp. 1-8.

Sifakis, J., 1980. Use of Petri nets for performance evaluation. *Acta Cybernetica,* 4(1978), pp. 185-202.

Smith, R., 2016. *Chemical process design and integration.* 2 éd. New York: John Wiley & Sons.

Song, Y., 2004.. *Qualité de service dans les réseaux et systèmes temps réel distribués : contributions à l'évaluation de performances temporelles et à l'ordonnancement,* Nancy: Université Henri Poincaré Nancy 1, LORIA.

Song, Y., Koubaa, A. & Simonot, F., 2002. Switched Ethernet for real-time industrial communication: Modelling and message Buffering delay evaluation. *4th IEEE International Workshop on Factory Communication Systems,* pp. 27-35.

Thramboulidis, K., 2004. Using UML in control and automation: a model driven approach. *Industrial Informatics, INDIN'04,* pp. 587-593.

Tindell, K. W., Hansson, H. & Wellings, A., 1994. Analysing real-time communications: controller area network (CAN). *Real-Time Systems Symposium,* pp. 259 - 263.

Van Deursen, A., Klint, P. & Visser, J., 2000. Domain-Specific Languages: An Annotated Bibliography. *Sigplan Notices,* 35(6), pp. 26-36.

Vanit-Anunchai, S., Billington, J. & Gallasch, G. E., 2006. Sweep-line analysis of DCCP connection management. *Seventh workshop and tutorial on practical use of coloured petri nets and the CPN tools,* pp. 157-175.

Walsh, G., Ye, H. & BushNell, L., 2001. Stability analysis of networked control systems. *Control Systems Technology, IEEE Transactions,* 10(3), pp. 438-446.

Wang, J., 1998. *Timed Petri nets: Theory and application.* 1 éd. New York: Springer Science & Business Media.

Ward, P., 1986. *Structured Development for Real-Time Systems: Vol. I: Introduction and Tools.* 1 éd. New Jersey: Pearson Education.

Wheels, J., 1993. Process control communications: Token bus, CSMA/CD, or token ring?. *ISA transactions,* 32(2), pp. 193-198.

Witsch, D., Vogel-Heuser, B., Faure, J. & Marsal, G., 2006. Performance analysis of industrial Ethernet networks by means of timed model-checking. *IFAC Proceedings Volumes,* 39(3), pp. 101-106.

Zhang, W., Branicky, M. & Phillips, S., 2001. Stability of networked control systems. *IEEE Control Systems,* 21(1), pp. 84 - 99.

Zimmermann, A., 2012. Modeling and evaluation of stochastic Petri nets with TimeNET 4.1. *Performance Evaluation Methodologies and Tools (VALUETOOLS),* pp. 54-63.

Zwick, R. & Walisten, T., 1989. Combining stochastic uncertainty and linguistic inexactness: theory and experimental evaluation of four fuzzy probability models. *International Journal of Man-Machine Studies,* 30(1), pp. 69-111.

Résumé

Ce mémoire, réalisé dans le cadre d'une thèse sous convention CIFRE avec la société Schneider-Electric et l'Université de Lorraine à travers le laboratoire du CRAN, porte sur l'évaluation des performances temporelles des architectures de contrôle-commande distribuées sur un réseau de communication. Le besoin industriel s'exprime sous la forme d'un outil d'aide au dimensionnement des architectures en phase d'avant-vente caractérisée par une connaissance partielle de ces dernières. Le problème scientifique sous-jacent est relatif à la génération automatique des modèles servant de support à l'évaluation. En effet, l'évaluation des performances doit être réalisée pour un ensemble important d'architectures, dans un temps court, difficilement compatible avec une construction manuelle des modèles. Notre contribution porte sur la définition formelle, à l'aide de réseaux de Petri colorés et temporisés, d'un modèle « constructeur » d'architectures embarquant des mécanismes de configuration, d'instanciation et de paramétrage. Plusieurs algorithmes sont proposés pour, d'une part, construire automatiquement le modèle d'une architecture donnée, à partir d'une description formelle de sa topologie et d'une librairie de modèles d'équipements de contrôle-commande, et, d'autre part, pour générer les observateurs requis à partir d'une description formelle des performances à évaluer. Ces différents algorithmes ont été implantés dans un outil interfacé, d'une part avec l'outil Schneider de description des architectures, et, d'autre part avec le simulateur de l'outil CPN Tools qui fournit une estimation des performances via des simulations de Monte-Carlo. L'intérêt de cette approche a été illustrée sur la base de quelques architectures types fournies par la société Schneider-Electric.

Mots-clés : évaluation de performances, architecture distribuée en réseau, réseaux de Petri colorés et temporisés, simulation de Monte-Carlo

Abstract

This PhD dissertation, supported by CIFRE convention between the company Schneider-Electric and the University of Lorraine through the CRAN laboratory, deals with the assessment of temporal performances for a networked distributed control system. The industrial need was the development of a quotation and sizing tool of industrial control architecture during pre-sales stage. This stage is characterized by limited information about the process and the customers' needs. The underlying scientific problematic was the ability to generate automatically models serving as support for the evaluation. In fact, performance assessment is realized for a wide range of architecture during a small amount of time, which is not compliant with a manual definition of the models. Our contribution is mainly based on a formal definition of a "builder" model with Colored and Timed Petri Nets which embeds mechanisms for configuration, instantiation and parameters setting of the architecture models. Several algorithms have been proposed for firstly build automatically the architecture Petri Nets model from a formal description of the topology and from a component model library and, secondly, for generating performance observers. Theses algorithms have been implemented on a tool gathering a user interface developed by Schneider –Electric and the Petri Nets simulator called CPN Tools which provides the performance assessment through Monte-Carlo simulation. The added value of this approach has been illustrated through case studies provided by Schneider-Electric.

Keywords : Performance assessment, industrial networked control system, Colored and Timed Petri Nets, Monte-Carlo simulation.