MASTER EN SCIENCES INFORMATIQUES

Les réseaux de Petri et leurs applications

Hastir, Pierre; Jodocy, Bernd

Award date:
1989

Awarding institution:
Universite de Namur

Facultés Universitaires N.D. de la Paix Namur
Institut d'Informatique
Rue Grandgagnage,21
B - 5000 NAMUR (Belgium)

Année académique 1988-1989

Les réseaux de Petri et leurs applications

Pierre Hastir
Bernd Jodocy

Directeur : Jean Fichefet

Mémoire présenté en vue de l'obtention du diplôme de Licencié et Maître en Informatique

RÉSUMÉ

Ce mémoire est consacré aux réseaux de Petri et à leurs applications. Il est composé d'un premier chapitre qui expose de façon synthétique la théorie des réseaux de Petri. Un deuxième chapitre donne quelques exemples et domaines typiques des applications des réseaux de Petri: software engineering, système de production, programmes temps réel,... . Dans un troisième chapitre nous expliquons en détail l'utilisation des réseaux de Petri dans la modélisation et la vérification de protocoles de communication. Les applications des réseaux de Petri demandent des outils spécifiques qui sont détaillés dans le quatrième chapitre. Et finalement le cinquième chapitre illustre par un exemple concret l'utilisation des réseaux de Petri aux protocoles de communication: la modélisation et la vérification de la couche application du protocole F.I.P. utilisé dans les processus automatisés de production.

ABSTRACT

This final study work is dedicated to Petri Nets and their applications. It is composed by a first chapter which exposes in a synthetic manner the theory of Petri Nets. A second chapter gives some examples and tipical areas of the use of Petri Nets: software engineering, production systems, real time programs... . In a third one, we explain in details the use of Petri Nets in the modelisation and the verification of communication protocols. The applications of Petri Nets require specific tools wich are detailled in the fourth chapter. Finally, the fifth chapter shows by a concrete example the use of Petri Nets on communication protocols: the modelisation and the verification of the application layer of the F.I.P. protocol, used in automatisated production process.

AVANT-PROPOS

Le mémoire que vous allez lire est le résultat de notre travail en vue de l'obtention du diplôme de Licencié et Maître en Informatique. Il clôture le cycle de cinq années d'études que nous avons effectué au sein des Facultés Universitaires Notre-Dame de la Paix à Namur.

Ce mémoire porte sur les applications des réseaux de Petri et a été soutenu par la réalisation d'un stage effectué au Laboratoire d'Analyse et d'Automatique des Systèmes du Centre National de la Recherche de Toulouse. Là, nous avons été cordialement accueillis dans l'équipe de Michel Diaz, Pierre Azéma et Guy Juanolle.

Notre travail, lors de ce stage, consista en la modélisation et la vérification d'une couche d'un protocole de communication en voie de normalisation appelé FIP (Flux Information Processus). Cette modélisation a été construite sur le logiciel PIPN réalisé par la collaboration du LAAS et de la société VERILOG.

La modélisation du protocole de communication nous a amenés à rafraîchir nos connaissances en la matière. De plus, le logiciel PIPN (Prolog Interpreted Petri Nets) utilisant comme langage de spécification Prolog, il nous a fallu approfondir notre connaissance de ce langage.

Partant de notre expérience de stage, nous avons donné à notre mémoire un accent orienté vers les applications des réseaux de Petri et plus spécialement la vérification des protocoles de communication par l'utilisation de réseaux de Petri.

Nous tenons à remercier :

Monsieur Jean Fichefet qui nous a patronnés dans ce mémoire,
tous les membres du jury qui ont participé à notre défense de mémoire,
ainsi que toutes les personnes qui, de près ou de loin, nous ont permis, au cours de ces cinq années, d'acquérir notre formation.

Nous tenons aussi a exprimer toute notre gratitude à :

Pierre Azéma qui nous a guidés lors de la réalisation de notre travail,
Jean-Christophe Lloret qui nous a aidés à utiliser le programme PIPN,
ainsi qu'à tous les autres membres de l'équipe OLC du LAAS qui nous ont si bien intégrés au sein de leur équipe pendant notre agréable séjour à Toulouse.

PLAN DU MÉMOIRE

AVANT-PROPOS

PLAN

INTRODUCTION

CONCLUSIONS GENERALES

ANNEXES

INTRODUCTION

La théorie actuelle des réseaux de Petri provient de la théorie de l'organisation des systèmes qui trouve son origine, il y a plus de 25 ans, dans la dissertation de Carl Adam Petri. Depuis cette thèse, la théorie des réseaux a été appliquée dans de nombreux domaines, principalement dans celui de la construction de modèles de systèmes concurrents.

La conception d'un système

La construction d'un système se déroule en trois étapes principales: la spécification, la conception et la mise en œuvre. Rappelons brièvement le vocabulaire que couvrent ces trois termes.

La spécification d'un système consiste à donner son cahier des charges. On doit décrire son comportement vu de l'extérieur. Il faut dire *ce que* le système doit faire sans préciser comment il va le faire.

La conception est l'étape suivante. Elle consiste à développer une solution abstraite, à décrire une architecture logique et à proposer des algorithmes.

La mise en œuvre, également appelée implémentation, consiste à réaliser effectivement le système. Il s'agit de produire à la fois une architecture matérielle et des programmes exécutables.

Avant de passer à l'implémentation, on procède généralement à une validation de la conception. Cette validation est souvent effectuée par une modélisation. La modélisation est plus qu'une simple description. Il s'agit d'une vue abstraite, c'est-à-dire que certains détails ne sont pas pris en compte. Cette abstraction doit de plus être effectuée dans un but précis. On modélise un système pour pouvoir prévoir son comportement et obtenir une idée quant à ses performances futures.

La détection des erreurs de spécification ou de conception est primordiale. Car au plus tôt les erreurs sont détectées dans le

cycle de vie d'un système, au moins sont coûteux les changements à y effectuer.

On constate que les langages utilisés pour la spécification et la conception sont essentiellement descriptifs. On décrit un problème, on décrit une solution. La modélisation est un peu plus exigeante puisqu'elle doit permettre à la fois de faire abstraction de certains détails et de faire des prévisions sur le comportement du système.

Les approches les plus classiques pour la spécification et la conception n'insistent pas particulièrement sur le côté modélisation. Le plus souvent, on utilise des langages qui sont très proches des langages de programmation.

Les langages de conception permettent simplement de décrire des solutions incomplètes mais sans chercher à prévoir le comportement du système par une analyse. Toute évaluation est fondée sur le prototypage.

La tendance actuelle en modélisation est d'utiliser des outils de modélisation comme SADT, MERISE ou encore la conception orientée objets. Ces méthodes insistent sur la nécessité de faire un effort d'abstraction lors des premières étapes de conception.

SADT (Structured Analysis and Design Technique), méthode textuelle et graphique, a été conçue pour fournir une approche rigoureuse et disciplinée à l'analyse d'un système. C'est une approche permettant de comprendre le problème, en le structurant, avant de chercher à en exposer une solution. Cette méthode permet l'analyse et la décomposition de problèmes complexes et est également un très bon outil de dialogue entre diverses équipes concernées par un même projet. Elle est utilisée dans la première phase du cycle de vie d'un projet, c'est-à-dire lors de la spécification.

MERISE est une méthode spécifiquement développée pour les systèmes d'information, c'est-à-dire des systèmes où la

structuration des données est plus importante que celle des séquencements des opérations.

Les approches orientées objets reposent d'abord sur la notion d'encapsulation d'un ensemble de données avec un ensemble d'opérations exécutées sur ces données. L'extérieur de l'objet ne pourra accéder aux données qu'à travers l'exécution de ces opérations, ce qui permet d'éviter des opérations incohérentes et mal contrôlées.

Les méthode de modélisation décrites ci-avant mettent l'accent sur les données. Ces méthodes sont très adaptés pour les systèmes conventionnels, où une bonne décomposition fonctionnelle et la description des flux des données sont généralement suffisantes pour spécifier complètement le système. Par contre, dans les systèmes de communication, dans les protocoles de communication, dans les systèmes en temps réel,... on constate que la partie de contrôle et le séquencement des opérations est plus important que la structure et les traitements associés aux données. Les méthodes orientées sur les données ne traitent pas de manière adéquate le comportement dynamique de tels systèmes.

Les réseaux de Petri ont été spécialement développés dans la recherche d'une méthode naturelle, simple et puissante pour le description et l'analyse de tels systèmes. Les réseaux de Petri possèdent, dès le départ, un structure composée d'une partie de contrôle et d'une partie opérative. Dans les réseaux de Petri on fait abstraction de la partie opérative et de ce qui se passe pendant les activités. On a donc une position de départ contraire à des méthodes de type MERISE.

Par comparaison avec les autres modèles de systèmes, les principales caractéristiques des réseaux de Petri sont les suivantes:

•les réseaux de Petri permettent une bonne prise en compte du parallélisme sans avoir à introduire, dès le départ, une structure rigide en processus communicants.

•ils peuvent être utilisés comme approche unique dans tous les cas où le traitement des données est de peu d'importance. Dans tous les autres cas ils doivent être utilisés conjointement à d'autres approches et en particulier SADT pour une première structuration et les approches à objets pour une bonne prise en compte des données.
•ils sont indispensables lorsque la structure de contrôle est complexe et présente un fort degré de parallélisme.

Les principaux domaines d'applications des réseaux de Petri sont d'après [REISIG85] :
- les systèmes d'interrogation des bases de données distribuées,
- les systèmes temps réel pour le contrôle de production,
- le contrôle des processus dans les systèmes d'exploitation
- les protocoles de communication.

Soit tous les systèmes où l'on trouve des événements se déroulant en parallèle et des événements concurrents.

Voyons maintenant le plan de notre mémoire.

Une première partie (chapitre 1) donne la base théorique des réseaux de Petri. Ceux-ci sont utilisés comme langage de modélisation car, grâce à eux on peut représenter graphiquement des modèles faisant intervenir des événements parallèles, concurrents, simultanés,... . Ils possèdent un large éventail de possibilités de modélisation. Mais la modélisation n'est pas la seule chose réalisable grâce aux réseaux de Petri. On peut également analyser ceux-ci. On peut analyser leurs propriétés structurelles ou leurs propriétés dynamiques.

Cette partie théorique finie, nous aborderons un éventail des différentes applications possibles des réseaux de Petri (chapitre 2). Celles-ci sont divisibles en deux grandes familles, celle qui rassemble les applications faisant principalement de la modélisation et l'autre qui rassemble les applications qui font de la modélisation et, en sus, de l'analyse. A travers les différents exemples donnés, on peut s'apercevoir de la grande variété de problèmes traitables par les réseaux de Petri.

Un problème plus particulier est celui de la vérification des protocoles de communication (chapitre 3). Un bref rappel de la théorie des protocoles est indispensable dans cette partie pour fixer le vocabulaire. Ensuite, on montre quelle méthode suivre pour arriver à modéliser les protocoles de communication par des réseaux de Petri. Ce modèle réalisé, on va l'analyser afin de détecter des propriétés particulières du modèle et donc du protocole. L'exemple qui soutient cette partie est celui du protocole du bit alterné.

Pour réaliser toutes ces applications basées sur les réseaux de Petri, il faut que le concepteur dispose d'outils manipulant ces réseaux. Cette partie (chapitre 4) explique quelles doivent être les caractéristiques des logiciels qui manipulent les réseaux de Petri. On y trouvera aussi les descriptions de deux outils existants, OVIDE et PIPN.

Finalement, dans la dernière partie (chapitre 5), nous montrons ce qu'un outil manipulant les réseaux de Petri peut faire. Nous nous servons pour cela d'un compte rendu de ce que nous avons réalisé lors de notre stage au LAAS. L'outil PIPN nous a permis de modéliser la couche application du protocole de communication FIP, ceci afin de la vérifier.

Au terme de sa lecture, nous espérons que le lecteur aura, comme nous, apprécié les réseaux de Petri. Leur facilité de modélisation leur permet d'être employés par des non-informaticiens comme par des concepteurs, ce qui n'est pas donné à tous les langages de spécification. De plus les réseaux de Petri sont analysables, ce qui leur donne une qualité fondamentale pour le concepteur qui peut vérifier la véracité, soit du modèle qu'il construit, soit du réel qu'il a modélisé. Ces avantages font des réseaux de Petri un sujet qui est promis à un bel avenir.

CHAPITRE 1

LA THÉORIE DES RÉSEAUX DE PETRI

Bernd Jodocy

1.1. INTRODUCTION

Un réseau de Petri est un modèle formel et abstrait pour des procédures, organisations ou équipements où des flux régulés, en particulier des flux d'informations, jouent un rôle. Les propriétés, concepts et techniques des réseaux de Petri ont été développés dans la recherche d'une méthode naturelle, simple et puissante pour la description et l'analyse de systèmes, en particulier des systèmes qui montrent un comportement asynchrone et concurrent. Les réseaux de Petri trouvent leur principale application dans la modélisation de systèmes d'événements dans lesquels certains événements peuvent arriver simultanément mais où il y a des contraintes quant à cette compétition, quant à la précédence ou la fréquence de ces occurrences.

La théorie des réseaux de Petri a été développée à partir de la thèse de doctorat de Carl Adam Petri en 1962 [PETRI62]. Petri introduisait dans cette thèse un nouveau modèle des flux d'informations dans des systèmes. Ce modèle fut basé sur les concepts d'opérations asynchrones et concurrents des parties d'un système et la constatation que la relation entre parties pouvait être représentée par un graphe ou un réseau.

Les idées de Petri furent reprises et développées, dans les années 1970, par un groupe de chercheurs américains comme A. W. Holt et J. Dennis du M.I.T. A partir de leur travail, l'utilisation des réseaux de Petri s'est largement développée. Un grand effort de recherche a été fait sur la nature et les applications possibles des réseaux de Petri, et leur utilisation s'accroît continuellement. La simplicité et la puissance des réseaux de Petri en font un outil excellent pour le travail avec des systèmes asynchrones et concurrents. Les champs d'application des réseaux de Petri vont de la modélisation de systèmes d'ordinateurs parallèles et de protocoles de communication jusqu'à la modélisation de relations inter-humaines.

1.2. MODÉLISATION AVEC DES RÉSEAUX DE PETRI

1.2.1. Introduction

Dans de nombreuses sciences, un phénomène n'est pas étudié en examinant le phénomène réel lui-même, mais plutôt en observant un modèle du phénomène. Un modèle est une représentation, souvent en terme mathématique, de ce qu'on croit être les caractéristiques importantes de l'objet sous étude. Par la manipulation de la représentation, on espère obtenir des connaissances nouvelles sur le phénomène modélisé sans encourir le coût, les inconvénients ou le danger de la manipulation du phénomène lui-même. Par exemple, beaucoup de recherches sur l'énergie atomique se sont faites par la modélisation, à cause des frais et des dangers d'une manipulation directe des matériaux atomiques.

Les réseaux de Petri sont un outil de modélisation très puissant. Ils ont été imaginés pour la modélisation d'une classe spécifique de problèmes, la classe des systèmes à événements discrets[1] ayant des événements concurrents[2] ou parallèles. Les réseaux de Petri modélisent des systèmes et plus particulièrement les aspects de systèmes, les événements et les conditions, et la relation entre eux.

1.2.2. Un exemple de modélisation par réseaux de Petri

Un exemple simple, extrait du domaine des systèmes d'ordinateurs, peut donner une intuition de la simplicité de description par réseaux de Petri [PETERSON81].

• Des travaux (jobs) se présentent et sont mis dans une liste d'entrée. Quand le processeur est libre et si la liste n'est pas vide, le processeur commence l'exécution du travail.

---

[1] un système à événements discrets est un système pour lequel les variables d'état varient brutalement à certains instants.

[2] deux événements sont concurrents s'ils veulent accéder à une ressource commune pour accomplir leur tâche.

• Si l'exécution aboutit à sa fin, le travail est placé dans une liste de sortie et si un ou plusieurs travaux se trouvent encore dans la liste d'entrée , le processeur continue avec un autre travail, sinon il attend qu'un nouveau job se présente.

Ceci est donc un exemple très simple, composé de plusieurs éléments : le processeur, la liste d'entrée, la liste de sortie et les travaux. Nous pouvons identifier plusieurs conditions d'intérêt :

- le processeur est libre
- un travail se trouve dans la liste d'entrée
- un travail est en cours d'exécution
- un travail se trouve dans la liste de sortie

et plusieurs événements pertinents :

- un nouveau travail entre dans le système
- début d'exécution d'un travail
- fin d'exécution d'un travail
- un travail quitte le système

Convenons de représenter :

• les conditions par des cercles, que nous appellerons places

• les événements ou actions par des barres, que nous appellerons transitions

• les conditions représentées par des places qui doivent être vérifiées pour que l'action puisse avoir lieu par une flèche allant des places à la transition

• les événements représentés par des transitions qui doivent avoir lieu avant que la condition indiquée par une place puisse jouer par une flèche allant des transitions à la place.

Le réseau de Petri de la figure 1.1. donne la modélisation du système informatique décrit ci-dessus.

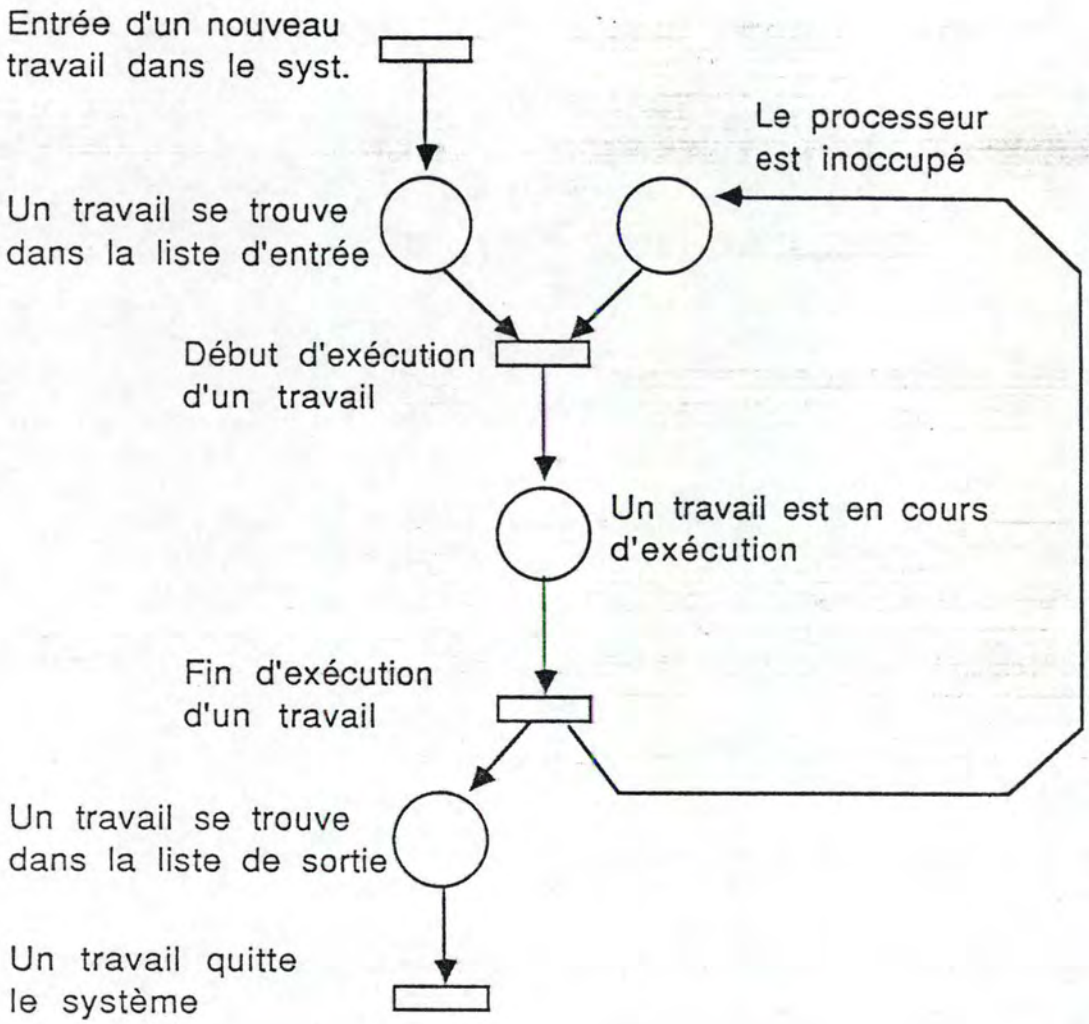


figure 1.1. Schéma d'un système informatique

1.2.3. Propriétés utiles à la modélisation

L'exemple ci-dessus illustre quelques caractéristiques importantes des réseaux de Petri et des systèmes qu'ils peuvent modéliser [FICHEFET87]. Une de ces caractéristiques est le **parallélisme et la concurrence inhérente.** Dans l'exemple ci-dessus, deux sortes d'entités indépendantes se trouvent dans le système : le travail et le processeur. Dans le modèle, les

événements qui sont relatifs à l'un ou l'autre peuvent arriver indépendamment. Il n'y a aucun besoin de synchroniser les actions des travaux et du processeur. Ainsi des travaux peuvent entrer et sortir du système à chaque instant, indépendamment de l'action du processeur. Tout de même, si la synchronisation est nécessaire, par exemple si la présence d'un travail et l'inoccupation du processeur sont requises pour commencer le traitement d'un travail, la situation est également facilement modélisable. Donc un réseau de Petri semble être idéal pour la modélisation de systèmes à contrôle distribué avec des processus multiples et concurrents.

Une autre caractéristique majeure des réseaux de Petri est leur **nature asynchrone**. Il n'existe aucune mesure explicite du temps ou du flux de temps dans un réseau de Petri. Ceci reflète une philosophie de temps qui affirme que la seule propriété de temps, d'un point de vue logique, est de définir un ordre partiel parmi les occurrences d'événements. Des événements durent un temps différent dans la réalité. Le modèle des réseaux de Petri reflète cette variabilité en restant indépendant d'une notion de temps pour contrôler la séquence des événements. Pour cette raison , la structure du réseau de Petri doit contenir toute l'information nécessaire pour définir toutes les séquences d'événements possibles du système modélisé. Ainsi, dans le réseau de la figure 1.1., à cause de la structure du réseau, l'événement "fin d'exécution du travail" doit suivre l'événement correspondant "début d'exécution du travail", même si aucune information sur la durée de cette exécution n'est fournie. D'autre part, des événements qui ne nécessitent pas d'être contraints par un ordre relatif à l'occurrence ne le sont pas. Ainsi, pendant qu'un travail est traité, l'événement "un nouveau travail entre dans le système" peut arriver, avant, après ou simultanément avec l'occurrence de l'événement "fin d'exécution du travail".

Un réseau de Petri, comme le système qu'il modélise, est vu comme une séquence d'événements discrets dont l'ordre d'occurrence est l'un des nombreux possibles permis par la structure de base. Si, à un moment donné, plus d'une transition est activée, alors chacune de ces actions peut se réaliser. Comme le

choix de l'événement ou de l'action se fait au hasard et n'est pas modélisé, ceci signifie que l'exécution d'un réseau de Petri s'effectue d'une manière **non-déterministe**. Mais si cette particularité peut s'avérer avantageuse du point de vue de la modélisation, elle introduit cependant pas mal de complexité sous l'angle de l'analyse des réseaux de Petri. Afin de réduire cette complexité, on accepte généralement d'apporter une limitation : on suppose que les actions sont instantanées, c'est-à-dire d'un temps nul.

Les événements que modélise un réseau de Petri sont considérés comme des événements primitifs. Ce fait ne pose en principe aucune difficulté pour la modélisation. Ainsi, par exemple, l'événement non-primitif "exécution d'un travail" fut modélisé dans l'exemple en le décomposant en un début et une fin (qui sont des événements instantanés) et le déroulement non instantané de l'exécution du travail.

Un aspect important des réseaux de Petri est qu'ils sont des **modèles non interprétés**. Le réseau de la figure 1.1. a été étiqueté par des commentaires qui indiquent au lecteur l'intention du modèle, mais ces étiquettes n'affectent nullement l'exécution du réseau. Nous traitons uniquement les propriétés abstraites inhérentes à la structure du réseau.

Une autre particularité intéressante des réseaux de Petri est la capacité de **modéliser d'une manière hiérarchique**. Un réseau entier peut être remplacé par une seule place ou transition en vue d'une modélisation à un niveau plus abstrait, et, inversement, des places et des transitions peuvent être remplacées par des sous-réseaux afin de fournir une modélisation plus fine.

1.3. LA THÉORIE DES RÉSEAUX DE PETRI

Ce paragraphe n'a pas l'intention de donner une vue complète de la théorie des réseaux de Petri. Nous donnerons plutôt une description synthétique des concepts et propriétés principaux de cette théorie. Pour ceci, nous nous baserons principalement sur les notes d'un cours sur les réseaux de Petri [VALETTE88] que nous avons eu l'occasion de suivre lors de notre stage à Toulouse.

1.3.1. Concepts de base

a) Réseau de Petri : Un réseau de Petri est un quadruplet

$$R = < P,T, Pre, Post> \quad (1)$$

où :

- P est un ensemble fini de places,
- T est un ensemble fini de transitions,
- Pre : P x T -> N est l'application incidence avant (places précédentes),
- Post : P x T -> N est l'application incidence arrière (places suivantes).

On utilise également la notation :

$$C = Post - Pre \quad (2)$$

b) Réseau marqué : Un réseau marqué est le couple :

$$N = < R,M> \quad (3)$$

où :

- R est un réseau de Petri,
- M est le marquage initial, c'est une application

$$M : P -> N \quad (4)$$

M(p) est le nombre de marques (jetons, tokens) contenus dans la place p.

c) Graphe associé et notations matricielles : A un réseau de Petri on peut associer un graphe qui possède deux types de nœuds : les places et les transitions. Un arc relie une place p à une transition t si et seulement si $Pre(p,t) \neq 0$. Un arc relie une

transition t à une place p à si et seulement si Post(p,t) ≠ 0. Les valeurs non nulles des matrices Pre et Post sont associées aux arcs comme étiquettes (par défaut, on prend la valeur 1). Un exemple de graphe associé à un réseau de Petri est donné par la figure 1.2..

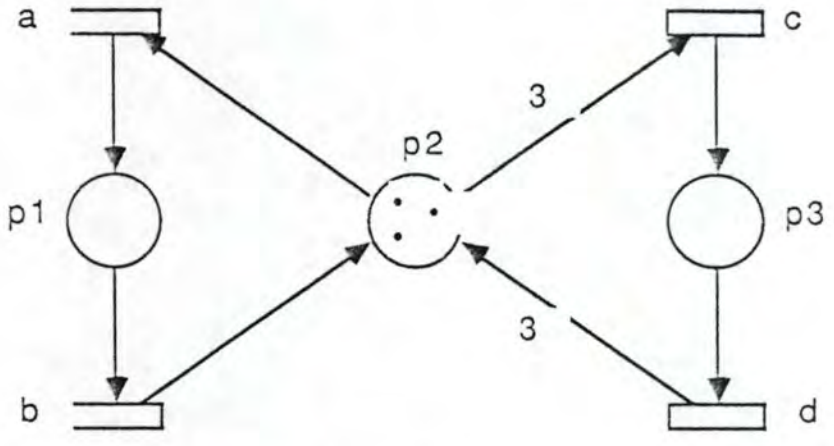


figure 1.2. Un exemple de réseau de Petri

Le marquage M peut être représenté par un vecteur ayant pour dimension le nombre de places; Pre, Post et C seront alors des matrices dont le nombre de lignes est égal au nombre de places et le nombre de colonnes est égal au nombre de transitions.

On note Pre(.,t), Post(.,t) et C(.,t) les colonnes de ces matrices associées à une transition t. Ce sont des vecteurs ayant pour dimension le nombre de places (comme M).

Considérons par exemple la figure 1.2.. Elle définit le réseau de Petri suivant :

- P = p1, p2, p3,
- T = a,b,c,d,
- Pre =

	a	b	c	d	
	0	1	0	0	p1
	1	0	3	0	p2
	0	0	0	1	p3

$$\bullet\ Post = \begin{array}{c} a\ \ b\ \ c\ \ d \\ \left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{array}\right) \end{array} \begin{array}{l} p1 \\ p2 \\ p3 \end{array}$$

$$\text{On a alors : } C = \begin{array}{c} a\ \ \ b\ \ \ c\ \ \ d \\ \left(\begin{array}{cccc} 1 & -1 & 0 & 0 \\ -1 & 1 & -3 & 3 \\ 0 & 0 & 1 & -1 \end{array}\right) \end{array} \begin{array}{l} p1 \\ p2 \\ p3 \end{array}$$

$$\text{et le marquage initial est : } M = \left(\begin{array}{c} 0 \\ 3 \\ 0 \end{array}\right) \begin{array}{l} P1 \\ p2 \\ p3 \end{array}$$

d) Transition franchissable : Une transition t est franchissable (sensibilisée, enabled) si et seulement si :

$$\forall\ p\ \epsilon\ P : M(p) \geq Pre\ (p,t) \qquad (5)$$

On peut exprimer que t est franchissable par les notations :

$$M \geq Pre\ (.,t\ ) \qquad (6)$$

ou encore $M\ (\ t >$ (7)

Par exemple dans le réseau de Petri de la figure 1.2. , les transitions a et c sont franchissables car $M > Pre(.,a\ )$ et $M = Pre\ (\ .,c)$.

e) Franchissement d'une transition : Si t est franchissable pour le marquage M, le franchissement (tir, firing) de t donne le nouveau marquage M' tel que :

$$\forall\ p\ \epsilon\ P \quad M'(p) = M(p) - Pre(p,t) + Post\ (p,t) \qquad (8)$$

On utilise également les notations :

$$M' = M - Pre\ (.,t) + Post\ (.,t) \qquad (9)$$

$$M\ (\ t > M' \qquad (10\ )$$

f) **Conflit et parallélisme :**

Conflit structurel : Deux transitions t1 et t2 sont en conflit structurel si et seulement si elles ont au moins une place d'entrée en commun :

$$\exists\, p \quad Pre\,(p,t1)\,.\,Pre\,(p,t2) \neq 0 \qquad (11)$$

Conflit effectif : Elles sont en conflit effectif pour un marquage M si et seulement si t1 et t2 sont en conflit structurel et que :

$$M \geq Pre\,(\,.,\,t1)$$
$$M \geq Pre\,(\,.,\,t2) \qquad (12)$$

Parallélisme structurel : Deux transitions t1 et t2 sont parallèles structurellement si :

$$Pre(\,.,t1)\;.\;Pre\,(\,.,t2) = 0 \qquad (13)$$

Elles n'ont donc aucune place d'entrée commune.

Parallélisme effectif : Deux transitions t1 et t2 sont parallèles pour un marquage donné M si et seulement si elles sont parallèles structurellement et :

$$M \geq Pre\,(\,.,\,t1\,)$$
$$M \geq Pre\,(\,.,\,t2\,) \qquad (14)$$

g) **Séquence de franchissement :** Si M (t1 > M' et M' (t2 > M" on dit que la séquence t1 t2 est franchissable pour M et on note :

$$M\,(\,t1\;t2\;>\,M" \qquad (15)$$

Soit s le vecteur dont les composantes s(t) sont les nombres d'occurrences des transitions t dans une séquence de franchissement S. Ce vecteur est appelé vecteur caractéristique de S. Sa dimension est égale au nombre de transitions du réseau de Petri.

Les évolutions du marquage d'un réseau de Petri sont alors données par l'équation :

$$M' = M - Pre\,.\,s + Post\,.\,s \qquad (16)$$

que l'on peut également écrire :

$$M' = M + C\,.\,s \qquad (17)$$

Cette équation est appelée l'équation fondamentale d'un réseau de Petri.

h) Ensemble des marquages accessibles : L'ensemble des marquages accessibles A (R;M) d'un réseau de Petri marqué est l'ensemble des marquages que l'on peut atteindre à partir du marquage initial par une séquence de franchissement.

$$A(R;M) = \{ Mi , \exists s : M (s > Mi \} \qquad (18)$$

On peut, lorsque cet ensemble est fini, le représenter sous la forme d'un graphe GA (R;M). Ce graphe a pour ensemble de sommets l'ensemble des marquages accessibles A (R;M); un arc orienté relie deux sommets Mi et Mj s'il existe une transition t franchissable permettant de passer d'un marquage à un autre : Mi (t > Mj.

La figure 1.3. représente le graphe des marquages accessibles pour le réseau de Petri de la figure 1.2. et le marquage initial M.

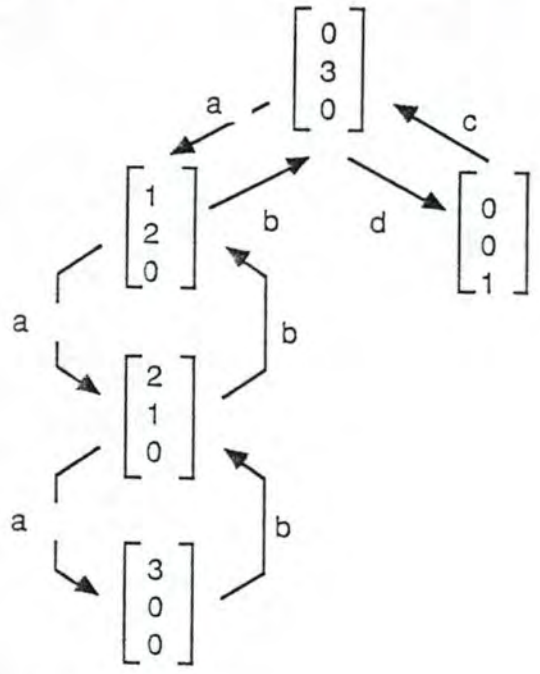


figure 1.3. Graphe des marquages accessibles

1.3.2. Propriétés

Nous allons définir un certain nombre de propriétés concernant les réseaux de Petri marqués et les réseaux de Petri non marqués. Les définitions des premières ne donnent pas directement des algorithmes permettant de déterminer si la propriété est

vérifiée ou non car elles impliquent l'ensemble des marquages accessibles qui n'est pas toujours fini. Les méthodes d'analyse de ces propriétés seront données dans le chapitre analyse. Par contre, les propriétés concernant les réseaux de Petri indépendamment de son marquage initial sont telles que des méthodes de calcul peuvent être dérivées directement des définitions.

1.3.2.1. Propriétés dues au marquage initial du réseau

a) **Place k-bornée et binaire :** Une place p d'un réseau marqué N est k-bornée si et seulement si :

$$\forall M \in A(R;M) \quad M(p) \leq k \qquad (19)$$

Si k = 1 on dit que la place est binaire (safe).
Si on considère le réseau de Petri de la figure 1.2., on voit que pour le marquage initial M, la place p3 est binaire alors que les places p1 et p3 sont 3-bornées.

b) **Réseau de Petri marqué k-borné et binaire :** Un réseau marqué N est k-borné (bounded) si et seulement si toutes ses places sont k-bornées.

Un réseau marqué est binaire (sauf, safe) si et seulement si toutes ses places sont binaires.

Le réseau de Petri de la figure 1.2. est 3-borné pour le marquage initial M. Si son marquage initial était M = p2, il serait borné (les transitions c et d ne seraient toutefois jamais franchies).

Un exemple de réseau non borné décrivant un mécanisme classique est donné par la figure 1.4.. Si, au franchissement de la transition a, on associe l'ouverture d'une parenthèse et, à celui de la transition b, la fermeture d'une parenthèse, les séquences de franchissement de transition qui, à partir du marquage initial M (M(p1) = 0) conduisent au même marquage M, décrivent toutes les expressions parenthésées licites. Il est clair que ce réseau n'est pas borné car on peut toujours ouvrir une parenthèse (franchir a) et donc ajouter un jeton dans la place p1.

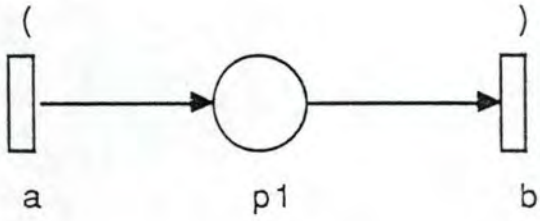


figure 1.4. Le réseau Parenthèse

c) Transition quasi-vivante : Une transition t d'un réseau marqué N avec N = < R,M > est quasi-vivante si et seulement si il existe une séquence de franchissement s telle que

$$M (S > M' \text{ et } M' (t > \qquad (20)$$

d) Transition vivante : Une transition t d'un réseau marqué N avec N = < R,M > est vivante si et seulement si il existe une séquence de franchissement S telle que

$$\forall M' \in A(R;M) \quad \exists S \quad M' (St > \qquad (21)$$

Exemple : Considérons le réseau de Petri de la figure 1.5. avec un marquage initial M = p3p4. Il est clair que la transition d est quasi-vivante (elle peut être franchie une fois), mais non vivante (elle ne peut être franchie qu'une fois car on ne peut plus sortir de la composante fortement connexe formée par les marquages p1, p2, p3 et p4).

Par contre les transitions a, b, c, e et f sont vivantes car elles figurent dans la composante fortement connexe et on pourra donc toujours trouver une séquence de franchissement les contenant.

Il faut faire attention à ce qu'une transition peut, sans être vivante, apparaître une infinité de fois dans des séquences infinies de franchissements de transitions. C'est le cas de la transition g

qui peut être franchie autant de fois que l'on veut avant le franchissement de d, mais après, elle ne peut plus être franchie.

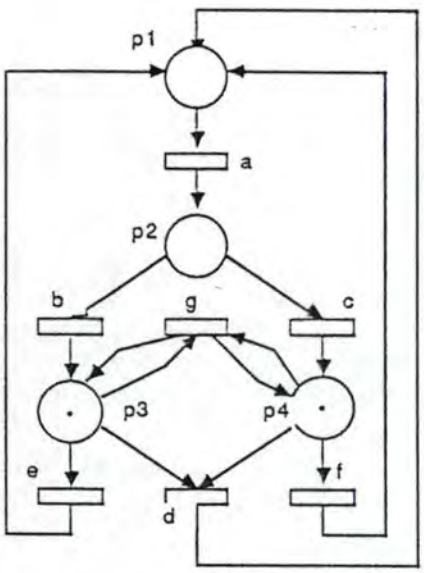


figure 1.5. Transition quasi-vivante/vivante et séquence infinie

e) Réseau marqué vivant : Un réseau de Petri marqué N = < R,M > est vivant si et seulement si toutes ses transitions sont vivantes pour un marquage initial M.

Un réseau de Petri vivant garantit qu'aucun blocage ne peut être provoqué par la structure du réseau de Petri. Par contre, il ne prouve pas l'absence d'éventuels blocages provoqués par une mauvaise interaction entre le réseau de Petri et son environnement.

Par exemple, le réseau de la figure 1.2. est vivant pour le marquage initial donné, par contre celui de la figure 1.5. ne l'est pas.

f) Réseau marqué réinitialisable : Un réseau marqué N = < R,M > est réinitialisable si et seulement si son graphe de marquages accessibles GA (R;M) est fortement connexe :

$$\forall M' \in A(R;M) \quad \exists S: M'(S > M \qquad (21)$$

La plupart des systèmes réels ont des fonctionnements répétitifs et donc les réseaux de Petri utilisés seront réinitialisables.

Remarque finale : Il est important de noter que les propriétés que nous venons de définir sont fortement liées au marquage initial. Considérons par exemple le réseau de Petri de la figure 1.6.. Pour le marquage M = p1p3, il est binaire, vivant et réinitialisable. Si on ajoute un jeton dans la place p4, alors il cesse d'être borné (voir la séquence abab...).

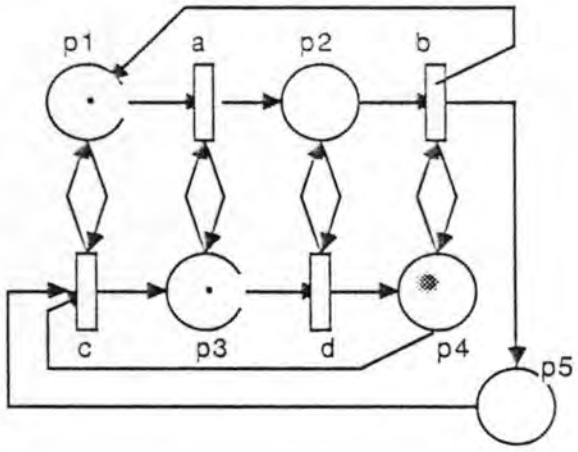


figure 1.6. Exemple de propriétés dues au marquage initial

1.3.2.2. Propriétés dues à la structure du graphe (les invariants)

Dans ce paragraphe, nous considérons d'abord des ensembles de places qui n'altèrent pas le nombre de jetons pendant des tirs de transitions. La connaissance d'un tel ensemble de places n'aide pas seulement à l'analyse de la vivance mais permet également de vérifier d'autres propriétés par exemple des "faits" du système. De tels ensembles de places sont appelés invariants de place ou encore composantes conservatives, parce qu'elles conservent l'ensemble des jetons. Comme les invariants sont caractérisés par des solutions d'équations du type C.x, il est possible de les calculer en appliquant les méthodes bien connues de l'algèbre linéaire.

Comme les invariants de place, nous obtenons les invariants de transition comme solution de systèmes d'équations linéaires. Ils indiquent combien de fois , en démarrant d'un marquage donné, chaque transition doit tirer pour reproduire ce marquage. A cause

de leur comportement cyclique, on les désigne parfois aussi comme composantes répétitives stationnaires.

a) Composante conservative, invariant de place : Considérons le réseau de Petri de la figure 1.7., et regardons le circuit formé par les places p1 et p2 et par les transitions a et b. Considérons maintenant la somme M(p1) + M(p2) qui vaut 1 pour le marquage initial M0. Le franchissement de a ne change rien à cette somme, ni celui de b. Celui des autres transitions du réseau ne modifie pas non plus cette somme. On peut donc affirmer que pour tous les marquages accessibles à partir du marquage initial on a :

$$M(p1) + M(p2) = 1$$

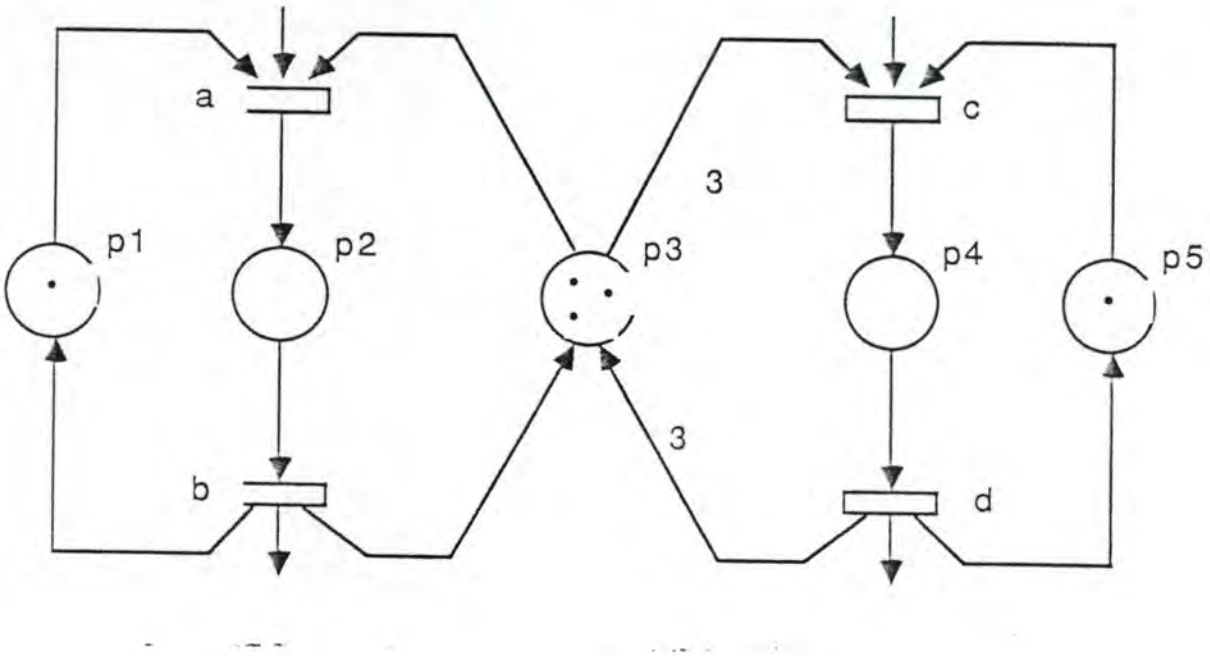


figure 1.7. Invariants

Si on considère maintenant le circuit (non élémentaire) formé par les places p2, p3 et p4 et par les transitions a, b, c et d, on pourra vérifier que

$$M(p2) + M(p3) + 3.\ M(p4) = 3 \qquad \forall\ M \text{ accessible}$$

Un invariant linéaire de place est une fonction linéaire du marquage des places dont la valeur est une constante, ne dépendant que du marquage initial du réseau. Il correspond à une contrainte

sur les états et les activités du système qui sera toujours vérifiée, quelles que soient ses évolutions.

Si nous revenons à l'équation fondamentale (équation 17) en prémultipliant chacun des termes par un vecteur :

$$f^T . M' = f^T . M + f^T . C . s \qquad (22)$$

Il est clair que la seule façon de se rendre indépendant des séquences de franchissements est d'annuler $f^T . C$.
Une composante conservative d'un réseau de Petri est une solution de l'équation :

$$f^T . C = 0 \qquad (23)$$

Si f est solution de l'équation ci-dessus alors la fonction linéaire :

$$f^T . M = f^T . M0 \qquad (24)$$

est appelée invariant linéaire de place.

b) Composante répétitive, invariant de transition : Considérons à nouveau la figure 1.7.: le franchissement de la séquence cd à partir du marquage initial redonne le même marquage. Le sous-réseau (circuit non élémentaire) à observer est cette fois formé des places p3, p4 et p5 avec les transitions c et d (multitude d'arcs entrant et sortant des transitions mais seulement un arc entrant et sortant par place).

On peut faire le même raisonnement avec la séquence ab, par exemple.

Un invariant de transition est une séquence de franchissement de transitions qui ne modifie pas le marquage du réseau. Un tel invariant correspond à des séquences cycliques d'événements qui peuvent être répétées indéfiniment. Si on reprend l'équation fondamentale, (17), il est clair que tout invariant de transition doit être une séquence s telle que s vérifie C. s = 0.

Toute solution s de l'équation :

$$C . s = 0 \qquad (23)$$

est appelée composante répétitive stationnaire. Si s est le vecteur caractéristique d'une séquence S de franchissement de transitions

effectivement franchissables à partir d'un marquage accessible alors cette séquence S est un invariant de transitions.

1.3.2.3. Relation entre "Bonnes Propriétés" et Invariants

L'utilisation des invariants de place permet de montrer que certaines places sont bornées et de calculer leurs bornes sans énumérer l'ensemble des marquages accessibles. Par contre les invariants de transition ne donnent que des conditions nécessaires mais non suffisantes.

On dit qu'il existe une couverture de composantes (conservatives ou répétitives stationnaires) si l'on trouve une composante (ou un ensemble de composantes élémentaires) qui passe par toutes les places (respectivement transitions). On a alors les résultats suivants :

1. Un réseau de Petri pour lequel existe une couverture de composantes conservatives $f^T > O$ (les composantes du vecteur f sont positives ou nulles) est k-borné quel que soit son marquage initial. En effet, la forme linéaire $f^T . M = f^T . M0$ permet de calculer une borne pour chaque place car : $M(p) \leq f^T.M0$

Il faut toutefois souligner qu'un réseau peut être borné pour un marquage donné sans posséder une couverture de composantes conservatives.

2. Tout réseau de Petri qui est à la fois vivant et borné pour au moins un marquage initial, est tel qu'une couverture de composantes répétitives stationnaires $s > 0$ existe.
Le point 2 découle du fait que

- borné => nombre fini de marquages sensibilisant une transition donnée,
- vivant => séquence de longueur infinie,
- vivant et borné => séquence répétitive stationnaire.

Il est clair que la réciproque du point 2 est fausse; il suffit en général de choisir un marquage pour lequel toutes les places sont vides.

1.4.TECHNIQUES D'ANALYSE

Deux types d'approche sont possibles pour l'analyse des bonnes propriétés :

• l'analyse par énumération : l'étude du graphe des marquages accessibles (l'établissement de la liste complète des états accessibles à partir d'un marquage initial M0) permet de vérifier la présence ou l'absence de bonnes propriétés et d'en déduire les conclusions qui s'imposent.

• l'analyse structurelle : elle permet d'étudier le comportement du système à partir des invariants de places et de transitions. Les relations obtenues sont indépendantes du marquage initial.

. les invariants de places du réseau qui sont des relations permanentes entre les nombres de jetons contenus dans différentes places (quel que soit le marquage atteint)

. les invariants de transitions permettent la détection de séquences de tirs cycliques qui sont à analyser en fonction de la sémantique associée au réseau. Tous les cycles détectés doivent avoir un sens dans la sémantique du système. Si ce n'est pas le cas, c'est qu'une erreur existe. Il convient de la détecter.

Il faut ajouter à ces méthodes, d'une part des techniques de réduction, et d'autre part des techniques de projection qui peuvent être utilisées lors de l'analyse par énumération. Ces techniques permettent notamment de travailler à différents niveaux d'abstraction.

1.4.1. Analyse par énumération

Le principe général est d'obtenir l'ensemble des états accessibles depuis l'état initial, lorsque cet ensemble est fini, ou bien de détecter en cours d'énumération que cet ensemble est infini [PETERSON81].

Pour trouver cet ensemble des états accessibles, il suffit d'appliquer la règle de tir des transitions. Chaque nœud du graphe

correspond à un marquage du système, chaque arc de ce graphe étant étiqueté par la transition qui permet le changement d'état.

Si on se contente d'énumérer les marquages accessibles, dans le cas où leur nombre est fini, il y aura arrêt de la procédure; par contre, si le nombre de marquages est infini, la procédure ne s'arrêtera jamais. Il nous faut donc un algorithme pour décider si un réseau de Petri est k- borné.

1.4.1.1. L'arbre de couverture

L'algorithme permettant de déterminer si un réseau de Petri marqué est k-borné est fondé sur la construction d'un arbre appelé "arbre de couverture".

On part du marquage initial. Chaque transition franchissable pour ce marquage donne naissance à une branche; on calcule les marquages obtenus par le franchissement des transitions et pour chaque marquage obtenu, on recommence. On arrête la construction d'une branche soit lorsqu'on trouve un marquage égal à un marquage déjà rencontré et pour lequel tous les successeurs ont déjà été calculés (sinon on réexplorerait un sous-arbre déjà calculé), soit lorsqu'on trouve un marquage strictement supérieur à un marquage de la branche en cours d'exploration.

Dans le deuxième cas, le réseau marqué n'est pas borné et l'arbre des états accessibles est infini. Si l'arbre veut être un outil d'analyse utile, alors nous devons trouver un moyen pour le limiter à une taille finie. La réduction à une représentation finie est accomplie en utilisant un symbole spécial, ω , qui peut être interprété comme "l'infini" et qui représente un nombre de jetons qui peut devenir aussi grand qu'on veut. En utilisant cette convention, on peut déterminer si le réseau marqué est borné ou non [1] et on peut connaître l'ensemble (et le graphe) des marquages accessibles.

---

[1] Si un réseau marqué est borné, alors le symbole ω ne peut pas apparaître dans l'arbre de couverture.

1.4.1.2. La terminaison de l'algorithme

Cet algorithme se termine car :

• on ne peut avoir aucune branche de longueur infinie [1]

• le nombre de branches est fini car pour chaque marquage, le nombre de transitions franchissables est fini (inférieur au nombre de transitions du réseau de Petri).

1.4.1.3. Un exemple

Afin d'illustrer ces points, considérons le réseau de Petri de la figure 1.8.. L'arbre de couverture initial est infini (fig 1.9.a.), car la transition t1 accumule les jetons et peut être tirée une infinité de fois. On a réduit le graphe à une taille finie à l'aide du symbole comme le montre la figure 1.9.b..

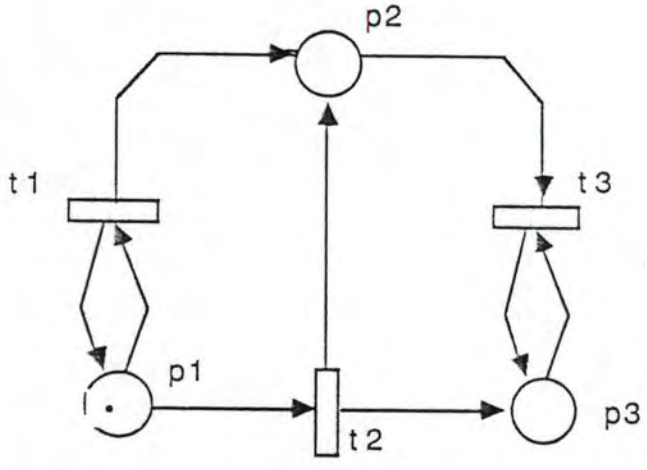


figure 1.8. Le réseau de Petri

[1] Cette affirmation découle du lemme de Karp et Miller : "Toute suite infinie de vecteurs formés d'entiers positifs ou nuls v1,...., vk,... est telle qu'elle contient au moins deux éléments vi et vj avec i< j tels que vi <vj ou bien vi = vj." Il est donc impossible d'avoir une suite infinie de vecteurs avec une suite de vecteurs strictement décroissante et dont la valeur doit rester supérieur ou égale à zéro.

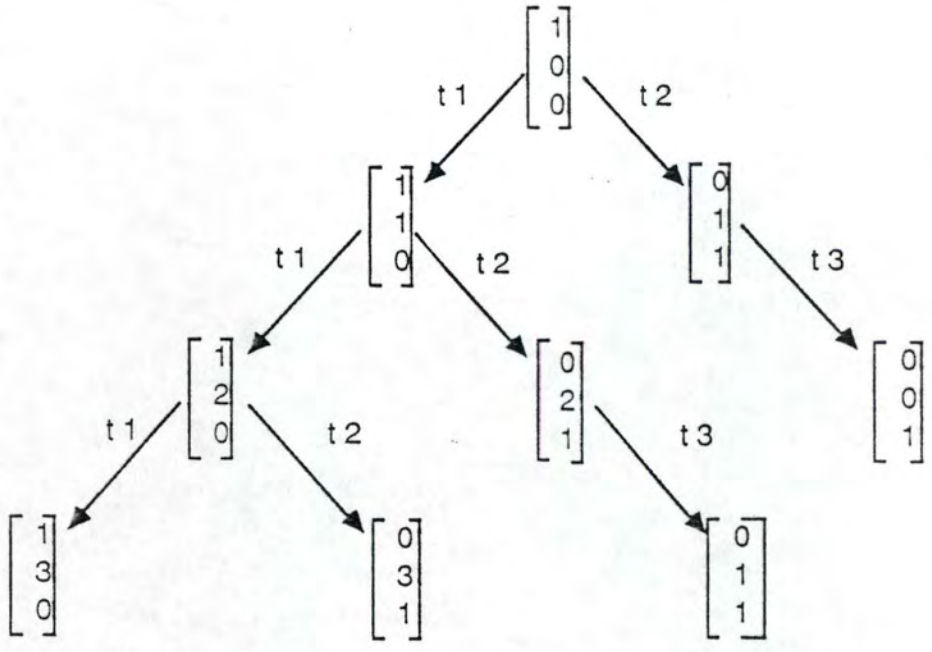


figure 1.9.a L'arbre de couverture initial infini

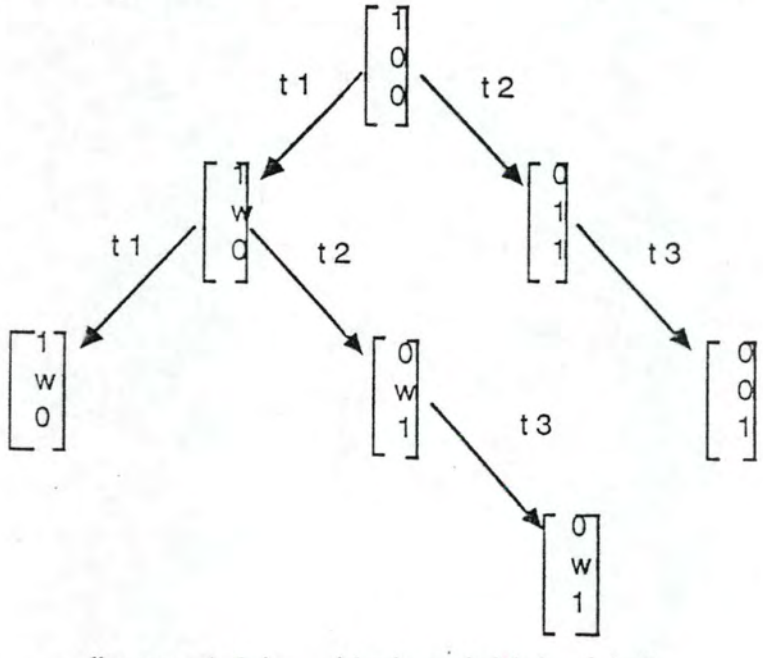


figure 1.9.b. L'arbre initial réduit

1.4.1.4. La recherche de propriétés

Nous venons de voir que la propriété de "borné" était vérifiable à l'aide du graphe de couverture. La démarche pour vérifier les autres "bonnes propriétés" à partir de l'arbre de

couverture est la suivante : on montre que le réseau est k-borné, puis qu'il est réinitialisable, puis enfin, qu'il est vivant.

En effet, on a les propriétés suivantes [VALETTE88] :
- R réinitialisable pour M0 <=> GA (R,M0) fortement connexe
- R réinitialisable pour M0 => (R quasi-vivant <=> R vivant)

Pour prouver que réinitialisable implique que GA (R,M0) est fortement connexe, il suffit de passer par le marquage initial M0 pour montrer qu'il existe au moins un chemin entre deux marquages accessibles quelconques Mi et Mj.

Pour prouver que quasi-vivant implique vivant si le réseau est réinitialisable, il suffit à partir d'un marquage accessible Mi de revenir au marquage initial M pour trouver une séquence amenant au franchissement d'une transition quelconque.

1.4.2. Analyse par réduction

Les méthodes présentées au paragraphe précédent sont extrêmement lourdes à mettre en œuvre lorsque la taille de l'ensemble des marquages accessibles devient grande (explosion combinatoire des états). L'idée est alors de trouver des règles de réduction [VALETTE88] qui soient telles que le réseau de Petri de départ et le réseau réduit aient les mêmes propriétés (équivalence vis-à-vis des "bonnes propriétés") de façon à n'appliquer la méthode décrite ci-dessus qu'à des réseaux de taille raisonnable qui donnent une ou plusieurs vues simplifiées différentes du même modèle.

Il faut remarquer que dans certains cas particuliers, certaines règles ne conservent pas toutes les bonnes propriétés et que la signification des éléments du réseau de Petri d'origine peut être altérée par ces réductions.

1.4.2.1. Place substituable

Informellement, une place substituable est une place qui sert de simple relais entre deux transitions (ou deux ensembles de

transitions). Le franchissement de la transition d'entrée (de l'une des transitions d'entrée) est une condition suffisante pour pouvoir franchir la (une) des transitions de sortie.

Considérons par exemple le réseau de Petri de la figure 1.10.a.. Le franchissement de l'une des deux transitions a ou b est une condition suffisante pour pouvoir franchir soit c soit d. La place p4 ne sert que de relais et on peut la supprimer. Sa suppression produit le réseau de Petri de la figure 1.10.b.

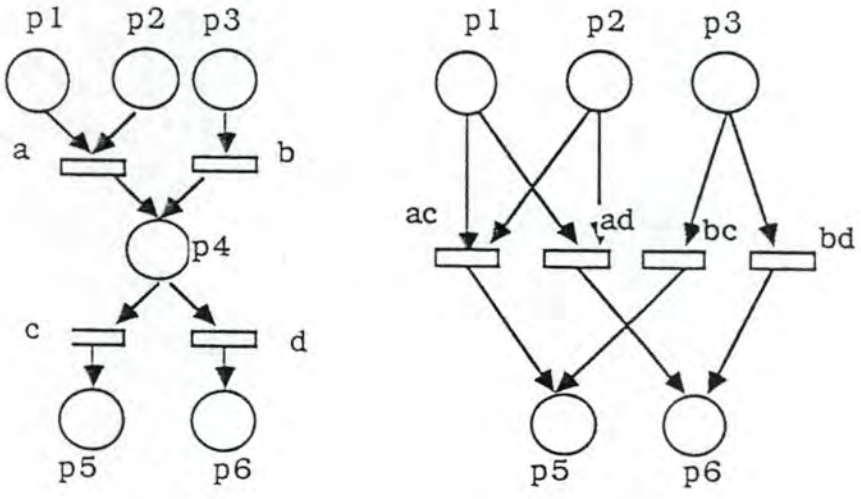


figure 1.10. Place substituable

1.4.2.2. Place implicite

Une place implicite est "une place qui ne sert à rien". En effet, son marquage est une combinaison linéaire du marquage d'un ensemble de places E et, vis-à-vis de ses transitions de sortie, elle n'introduit aucune condition supplémentaire de franchissement.

Le fait que le marquage d'une place implicite doive être une combinaison linéaire du marquage d'un ensemble E relie cette notion à celle de composante conservative.

Considérons le réseau de la figure 1.11.; nous allons montrer que la place p1 est implicite vis-à-vis de l'ensemble des places (p2,p3). Pour cela, faisons comme si nous voulions chercher les composantes conservatives.

Le jeu d'équations produit par fT . C = 0 est :

$$f1 + f2 - f4 = 0$$
$$- f2 + f3 = 0$$
$$- f1 - f3 + f5 = 0$$

Comme on ne s'intéresse qu'aux places p1, p2 et p3, on cherche d'abord des solutions pour lesquelles f4 et f5 sont égaux à zéro. On trouve alors :

$$f1 = - f2$$
$$f2 = f3$$

et on obtient l'invariant de place (en prenant par exemple f1 = 1):

$$M(p1) = M(p2) + M(p3)$$

Nous pouvons donc, à tout moment, calculer le marquage de la place p1 en fonction du marquage des places p2 et p3. La seule transition de sortie de la place p1 est la transition c. Or pour qu'elle soit franchissable, il faut que la place p3 contienne au moins un jeton et nous avons vu plus haut que cela implique que p1 contienne également au moins un jeton. Donc la place p1 "ne sert à rien" et peut être supprimée sans modification des séquences de franchissement des transitions .

Il faut faire attention à deux choses. D'abord le marquage initial du réseau modifie l'équation permettant de calculer le marquage de la place implicite. Par exemple, la présence d'un jeton initialement dans l'une des places p2 ou p3 donnera :

$$M(p1) = M(p2) + M(p3) - 1$$

En conséquence le fait que la place p2 contienne un jeton n'est pas suffisant pour affirmer que p1 contient au moins un jeton.

Le second problème auquel il faut faire attention est qu'il n'est pas suffisant que le marquage d'une place soit fonction du marquage d'un sous-ensemble E pour qu'elle soit implicite. Si dans notre exemple on avait eu une transition d suivant de p1 mais qui n'était ni suivant de p2 ni de p3, alors p1 ne pourrait pas être supprimée.

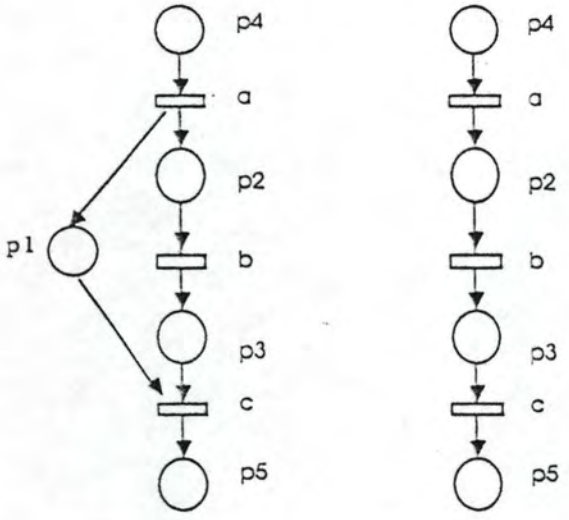


figure 1.11. Place implicite

1.4.2.3. Transition neutre ou identité

Une transition neutre est "une transition qui ne sert à rien". En effet,elle n'est connectée au réseau de Petri que par des boucles élémentaires. Son franchissement ne modifie donc pas le marquage du réseau de Petri.

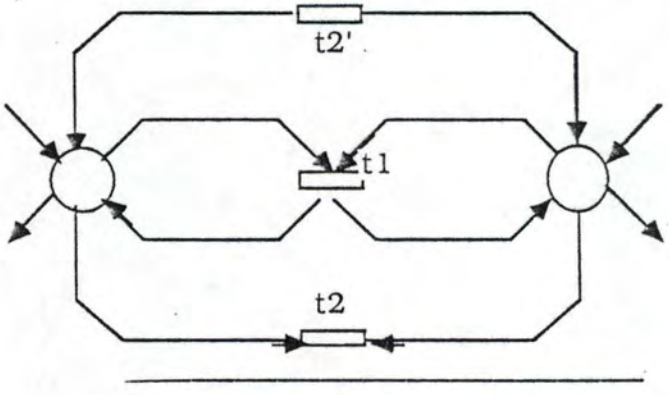


figure 1.12. Transition neutre

Pour pouvoir la supprimer, en étant assuré de conserver la propriété de réseau marqué vivant, il faut être certain qu'elle soit

vivante. Soit t1 cette transition; on peut la supprimer s'il existe une autre transition t2 avec soit : pre (.,t1) = pre (.,t2), soit : pre (.,t1) = post (.,t2).

Ceci est illustré par la figure 1.12..

1.4.2.4. Transition identique

Deux transitions sont identiques si les colonnes correspondantes des matrices pre et post sont identiques. Il est clair que l'une des deux peut être supprimée sans changer les propriétés du réseau de Petri.

Le réseau de Petri de la figure 1.13. donne un exemple de deux transitions qui sont identiques.

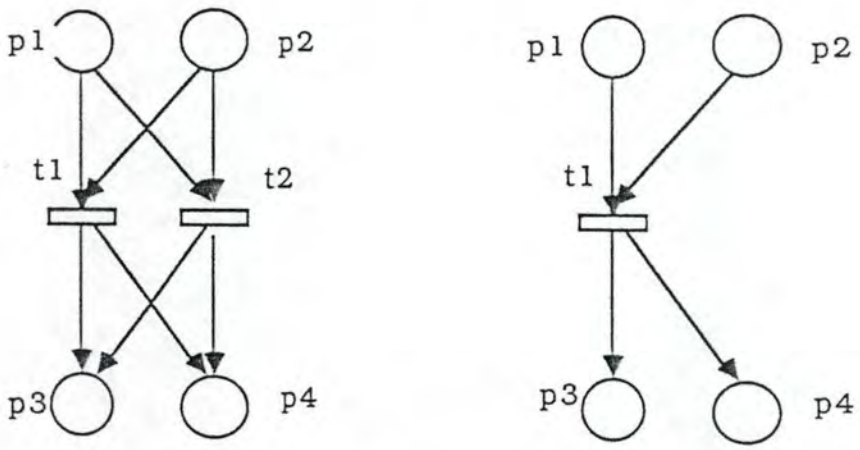


figure 1.13. Transition identique

1.4.3. Projection

La projection consiste à réduire le graphe des états après son énumération afin d'en obtenir une "vue abstraite". L'utilisateur choisit les transitions du réseau de Petri qu'il souhaite conserver "visibles", les autres éléments du réseau devenant alors internes et donc "invisibles".

La méthode de projection consiste ainsi :

• à considérer l'ensemble des événements associés aux transitions du réseau;

• à partitionner cet ensemble en deux sous-ensembles :

a) le sous-ensemble des événements internes, c'est-à-dire ceux qui ne doivent plus apparaître dans la vue abstraite; les transitions associées à ces événements seront appelées lambda-transitions;

b) le sous-ensemble des événements visibles qui doivent eux continuer à apparaître dans la vue abstraite; à un même événement visible peuvent être associées plusieurs transitions;

• à éliminer les lambda-transitions; notons que ceci peut être par exemple effectué en partant du graphe des marquages du réseau de Petri : on obtient ainsi, après avoir supprimé d'une part le non-déterminisme qui est en général introduit par l'élimination des lambda-transitions, et les états équivalents, un automate réduit ayant en général peu d'états et plus facilement analysable.

1.4.4. Analyse structurelle

Nous étudierons seulement ici l'analyse structurelle du réseau de Petri sous-jacent (comme présenté en [REISIG85] ou [VALETTE88]), quel que soit le type de modèle utilisé; certains auteurs [D'ANNA88] développent également des méthodes spécifiques, suivant le type de modèle utilisé.

Les invariants de places traduisent le fait que le marquage d'un certain nombre de places du réseau, pondéré par les composantes de l'invariant, reste constant.

Les invariants de transitions caractérisent des séquences de tir cycliques de ce réseau.

Ceci explique que ces invariants soient aussi appelés "composantes conservatives" et "composantes répétitives" d'un réseau.

Les invariants ne donnent que des informations relatives à la structure du réseau et ne nécessitent aucune énumération des états. Il est possible de trouver des invariants de places et de

transitions pour un réseau dont le nombre d'états est infini. Ceci constitue un des points les plus intéressants de cette méthode.

Les invariants de places (respectivement de transitions) sont des vecteurs dont le nombre de composantes est égal au nombre de places du réseau (respectivement de transitions). Ils sont obtenus par résolution d'un système d'équations linéaires en nombres entiers.

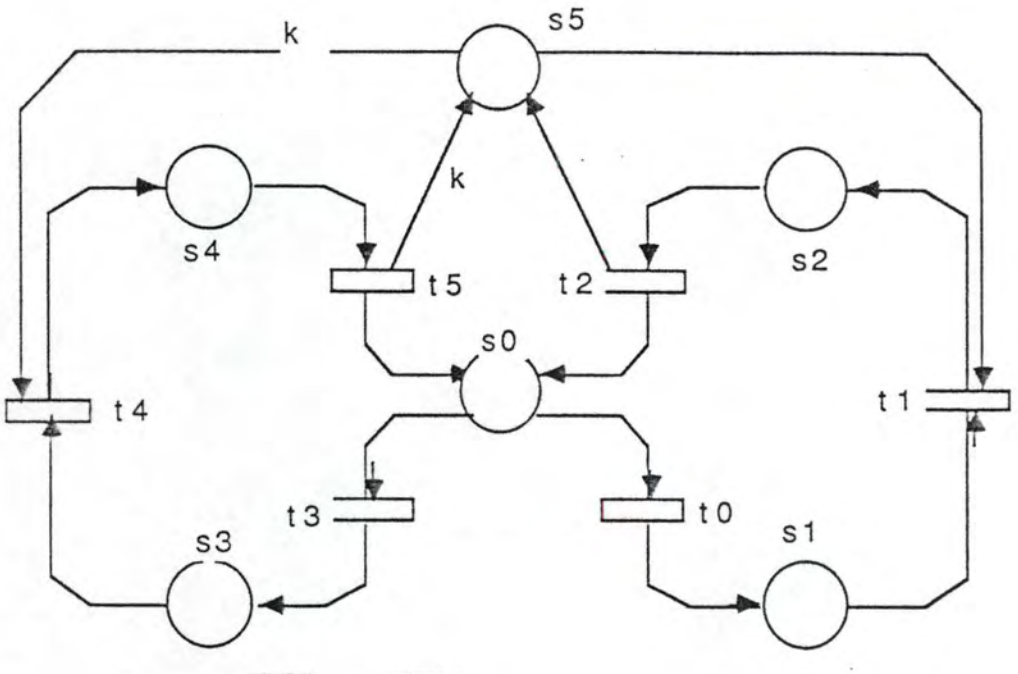
Nous considérerons à titre d'exemple le problème des lecteurs et écrivains [REISIG85] pour illustrer quelles propriétés structurelles peuvent être reconnues par une connaissance des invariants de place d'un réseau. Supposons que n processus d'un système d'exploitation peuvent accéder à un tempon en mode de lecture ou d'écriture. Pour garantir la cohérence, la lecture et l'écriture sont restreintes de la manière suivante : si aucun processus n'est en écriture alors k (<= n) processus peuvent lire dans le tempon. Mais l'accès en écriture à cette zone est évidemment uniquement permis si aucun autre processus ne lit ou n'écrit.

Un tel système d'écriture et de lecture est illustré par la figure 1.14.. Chaque processus est dans un des cinq états représentés par les places s0, ..., s4. Dans l'état initial, tous les n processus sont passifs, s0 contient donc au départ n jetons. La place s5 contient elle, par contre, k jetons qui correspondent au nombre de processus simultanément admis en lecture.

A l'aide des invariants montrés à la figure 1.15., il est possible de démontrer l'exactitude du système conçu.

En utilisant l'invariant i1, nous avons pour chaque suivant M de M0 :

$$M(s0) + M(s1) + M(s2) + M(s3) + M(s4) =$$
$$M0(s0) + M0(s1) + M0(s2) + M0(s3) + M0(s4) = n.$$



s0 : processus inactifs
s1 : processus prêts à lire
s2 : processus en lecture
s3 : processus prêts à écrire
s4 : processus en écriture
s5 : synchronisation

figure 1.14. Le problème des lecteurs et écrivains

	t0	t1	t2	t3	t4	t5	i1	i2	M0
s0	-1		1	-1		1	1		n
s1	1	-1					1		
s2		1	-1				1	1	
s3				1	-1		1		
s4					1	-1		k	
s5		-1	1		-1	k		1	k

figure 1.15. Matrice, invariants et marquage initial du réseau 1.14.

Ceci signifie que le nombre n de processus reste constant et que chacun des processus est dans un des états s0,....s4.

En prenant l'invariant i2, nous obtenons pour chaque marquage M suivant de M0 :

$$M(s2) + k.\ M(s4) + M(s5) = M0(s2) + k.\ M0(s4) + M0(s5) = k.$$

Donc, nous trouvons que s4 contient au maximum un jeton, ce qui équivaut à dire qu'il existe au maximum un processus en écriture. Si s4 contient un jeton, alors s2 et s5 sont vides, ce qui correspond bien à la spécification qui prescrit l'exclusion de processus en écriture et en lecture. s2 contient au maximum k jetons, ce qui veut dire qu'il ne peut y avoir plus que k processus simultanément en lecture. Quand aucun processus n'est en écriture (M(s4) = 0), alors s2 peut en fait porter k jetons. Dans ce cas, la place de synchronisation s5 est vide.

En particulier, nous pouvons prouver la vivance du réseau. Pour les raisons discutées ci-dessus, le marquage ne va jamais empêcher un quelconque tir de transitions. Montrons que chaque marquage M suivant de M0 sensibilise au moins une transition. Dans le cas où M(s0) + M(s2) + M(s4) > 0, nous voyons que, par la structure du réseau au moins une des transitions t0, t2, t3 ou t5 est sensibilisée. Si M(s0) + M(s2) + M(s4) = 0, nous obtenons de l'invariant i1 que M(s1) + M(s3) = n, et de l'invariant i2 que M(s5) = k. Par conséquent, soit t1, soit t4 est sensibilisé. Maintenant, si s0 est vide pour un marquage M, elle peut être marquée par une succession de tirs. Ceci implique la vivance de t0 et t3. La vivance des autres transitions s'en conclut immédiatement.

CHAPITRE 2

LES APPLICATIONS DES RÉSEAUX DE PETRI

Pierre Hastir

2.1. INTRODUCTION

Après avoir introduit la théorie des réseaux de Petri dans le chapitre précédent, voyons quelles sont les applications pratiques des réseaux de Petri.

Les réseaux de Petri sont actuellement à la mode, et depuis le début des années 80, les publications concernant l'utilité et les applications pratiques des réseaux de Petri font florès.

Les emplois des réseaux de Petri dans les domaines de l'informatique, de la productique ainsi que dans divers autres domaines sont nombreux et variés. Nous allons néanmoins essayer de donner un fil conducteur à ces différentes applications des réseaux de Petri.

Les réseaux de Petri sont utilisés pour deux tâches principales : la modélisation et la vérification de systèmes. La modélisation par réseaux de Petri a pour objectif de donner à l'utilisateur un langage de représentation lui permettant de modéliser un système où interviennent des actions en parallèle, des actions synchrones,... La vérification utilisant les réseaux de Petri donne à l'utilisateur les avantages des réseaux de Petri en ce qui concerne l'analyse. Ces analyses d'un réseaux de Petri modélisant un système peuvent donner à l'utilisateur une idée des propriétés du système et de son évolution.

Les pages qui suivent vont montrer ces applications des réseaux de Petri. Une première partie va décrire quelques applications dans l'aide à la modélisation et une seconde partie montrera des applications dans la vérification.

2.2. LES APPLICATIONS DE MODELISATION

Les descriptions des systèmes d'informations sont utilisées dans toutes les phases du développement des systèmes. Elles servent de moyen de communication entre les personnes engagées dans le processus de développement du SI.

Les descriptions doivent fournir aux lecteurs l'information nécessaire des aspects importants du système considéré. Le langage de description doit, du mieux qu'il peut, apporter un support à la clarté et à la compréhension de la description du système.

Les langages actuellement utilisés sont principalement orientés dans la perspective des spécialistes en informatique, ce qui offre une pauvre qualité de description du point de vue des autres lecteurs. C'est pourquoi l'utilisation des réseaux de Petri se répand dans le monde de la modélisation car, par sa représentation graphique, de telles modélisations utilisant les réseaux de Petri sont facilement compréhensibles pour les utilisateurs aussi bien que pour les concepteurs.

Les domaines d'utilisation des réseaux de Petri dans l'aide à la conception sont variés. Cela va de l'aide à la conception de logiciel à l'aide à la conception de système de production. Les application décrites ci-après sont l'aide à la conception de logiciel, la modélisation des relations homme-machine et l'aide à la conception de systèmes de production.

2.2.1. L'aide à la conception de logiciels

Dans ce domaine les réseaux de Petri vont être utilisés pour modéliser un "problème", ou plus exactement son environnement dans lequel on veut développer un système informatique.

L'exemple [REISIG86] qui sera développé pour illustrer ce type d'application est la location de voitures. Trois acteurs interviennent dans ce "problème"; le client qui veut louer une

voiture, le service de location qui gère les voitures et la compagnie d'assurances qui assure les voitures.

Chaque acteur possède une idée du "problème" qu'est la location de voitures; il possède un point de vue sur la "réalité globale" de la location de voitures. Ainsi le point de vue du client peut être représenté comme à la figure 2.1 . Le client veut louer une voiture au service de location. Celui-ci lui loue une voiture. Le client l'utilise puis la reconduit au service de location pour redevenir un piéton.

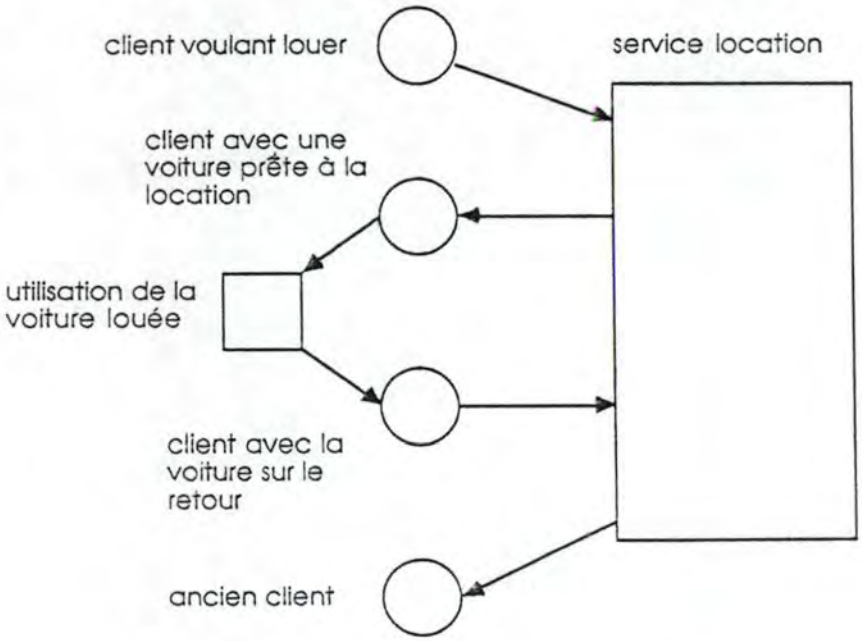


figure 2.1. Point de vue du client

Mais cette vue de la location n'est pas la seule. Le service location possède lui aussi une opinion sur ce qu'est le service qu'il rend. La figure 2.2 donne l'interprétation du service location. Ce point de vue est basé sur la gestion des index qui indiquent la disponibilité ou la location des voitures. Les deux actions importantes pour ce point de vue étant la signature du contrat lors de la prise de la voiture et le paiement de la location lors du retour de la voiture.

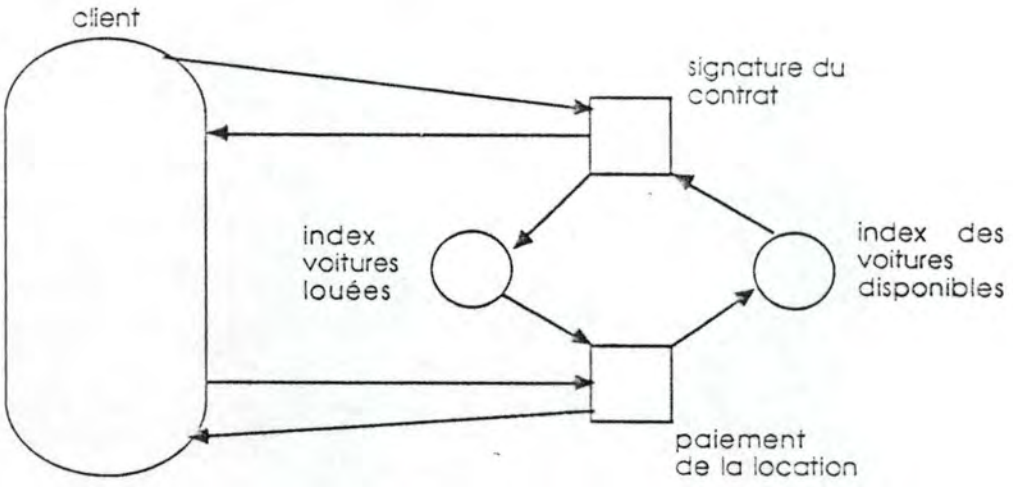


figure 2.2. Point de vue du service

Les assurances sont, quant à elles, concernées par le fait qu'elles assurent les voitures et doivent donc, de leur point de vue, connaître à tout moment qui possède la voiture; un client ou le service. La figure 2.3 exprime le point de vue des assurances.

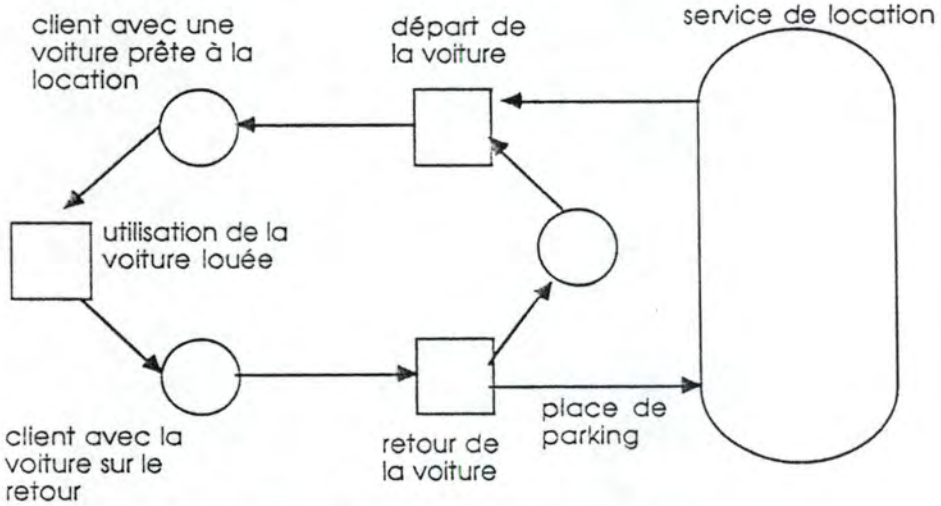


figure 2.3. Point de vue des assurances

Comme on peut le remarquer, les réseaux décrivant les points de vue utilisent des places et des transitions. En général les places (cercles) seront utilisées pour modéliser des canaux où l'on stocke des objets pour les rendre visibles. Les transitions (rectangles) sont utilisées pour modéliser des agences qui produisent, transportent ou modifient des objets.

Les réseaux de Petri possèdent les principes d'abstraction et de raffinement. Ces principes concernent les relations qui existent entre les différents réseaux qui modélisent les points de vue. Ainsi dans la figure 2.1, l'agence appelée service location est une abstraction de ce service. En effet, on ne rentre pas dans le détail de ce que fait le service lors de la location de la voiture au client. Si on raffine le service location, on peut alors obtenir le réseau exprimé par la figure 2.4 qui représente un raffinement de la figure 2.1.

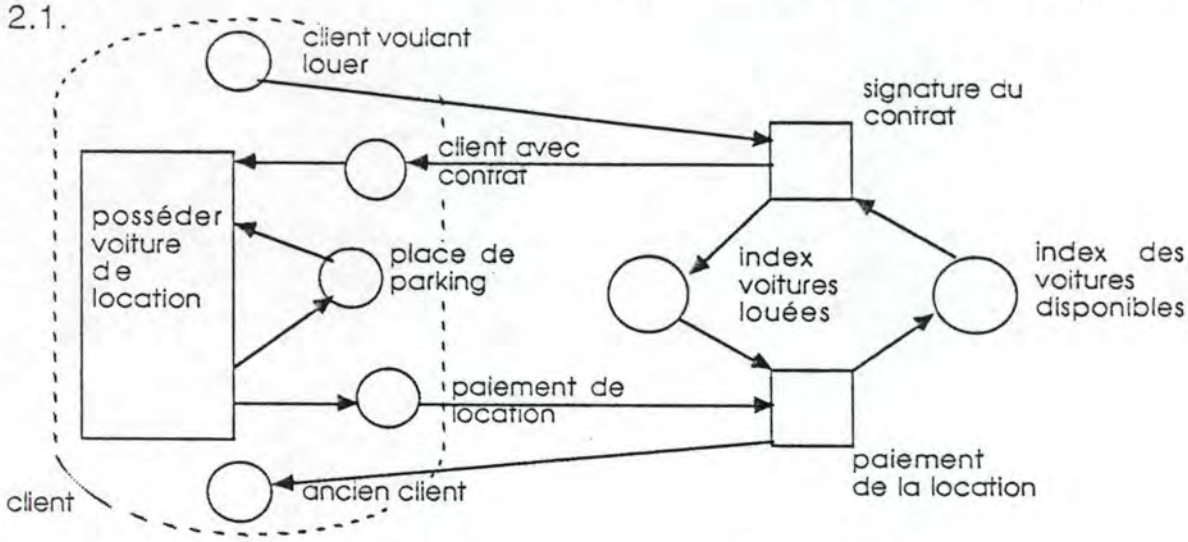


figure 2.4. Raffinement de la figure 2.1.

Le point de vue du service réalise aussi une abstraction. Celle-ci ne parle que du client sans le détailler. Le raffinement de ce canal qu'est le client donne la figure 2.5 qui est un raffinement de la figure 2.2.

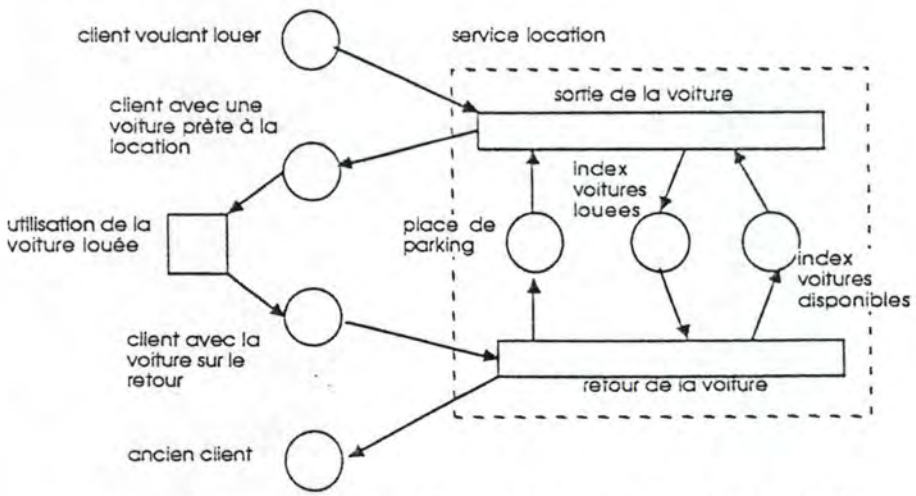


figure 2.5. Raffinement de la figure 2.2

Mais le raffinement et l'abstraction ne sont pas les seuls principes que l'on peut appliquer aux réseaux obtenus. On peut aussi effectuer des fusions ou des divisions de réseaux. Les figures 2.4 et 2.5 se prêtent bien à une fusion car elles possèdent des places communes. La figure résultante 2.6 est donc la modélisation commune du point de vue des clients et du point de vue du service de location. Elle représente donc une meilleure interprétation de la "réalité du problème".

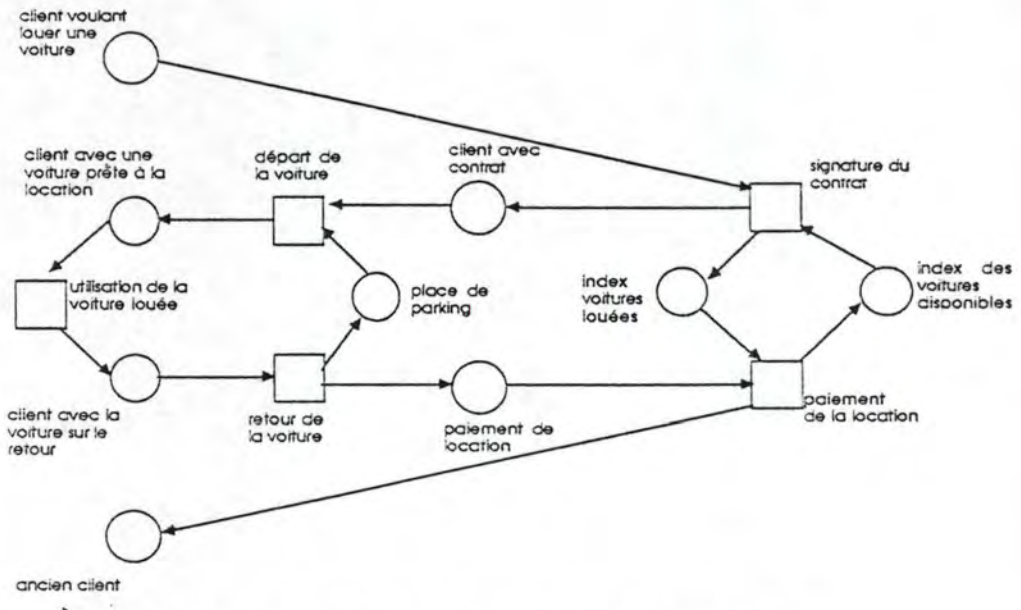


figure 2.6. Fusion des figures 2.4 et 2.5

Cette figure 2.6 traite l'aspect statique du "problème" de la location de voitures. Nous avons une description des acteurs qui modélise la location mais aucun objet n'y est représenté. On peut apporter un aspect dynamique à cette modélisation en plaçant dans les canaux des objets. La figure 2.7 représente le réseau où l'on a introduit des clients, des voitures et les index.

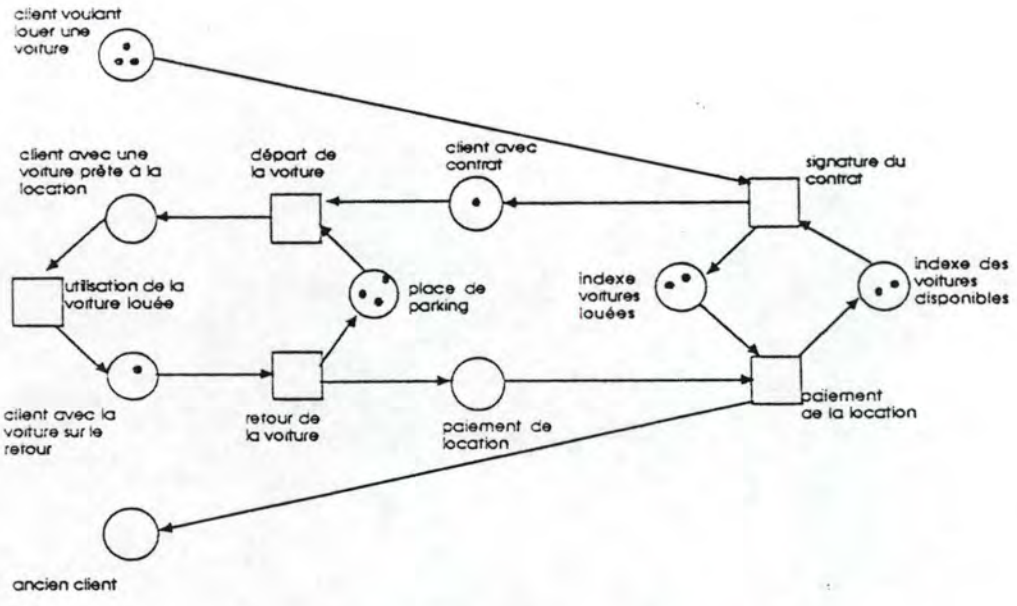


figure 2.7. Dynamisation du modèle

Mais les objets que l'on peut introduire dans les réseaux de Petri traditionnels sont non différenciables. Or il peut être intéressant de posséder un modèle dynamique où l'on peut différencier les objets. C'est ce qui est fait par l'introduction de prédicats qui donne alors le réseau de la figure 2.8.

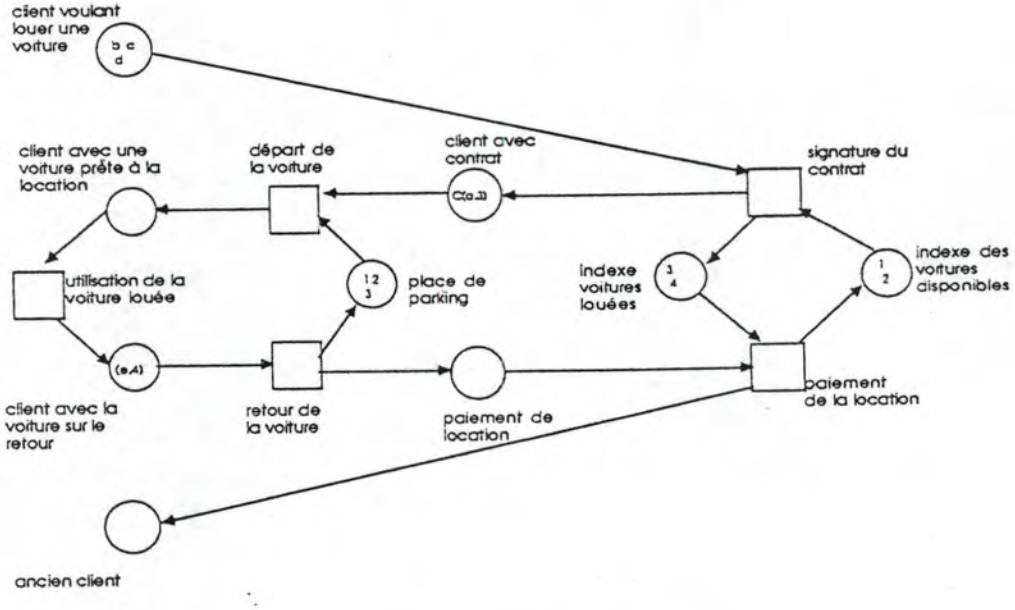


figure 2.8. Différenciation des objets

Dans le réseau ci-dessus, on a introduit deux notations: C(x,y) qui signifie que le client x passe un contrat pour la voiture y et (x,y) qui signifie que le client x est avec la voiture y. Les clients sont représentés par des lettres et les voitures par des chiffres.

En conclusion on peut dire que les réseaux de Petri ont permis de modéliser les différents point de vue des acteurs du problème; que grâce au principe d'abstraction, qu'on peut réaliser des réseaux utilisant des sous-réseaux, et grâce au principe de raffinement, on peut développer progressivement des réseaux de niveaux plus abstraits à des niveaux plus détaillés par raffinements successifs. De plus les actions de fusion et de division permettent soit de mettre en relation deux réseaux isolés pour représenter une "réalité" plus vaste, soit de séparer un réseau pour isoler certains aspects d'une réalité trop grande. Enfin la dynamisation de réseau statique permet à l'utilisateur d'étudier le comportement de sa modélisation lorsqu'il fait évoluer la dynamique de son système. Cette dynamique peut soit traiter des objets non différenciables, soit traiter des objets discernables par l'introduction de prédicats dans les réseaux de Petri.

2.2.2. La modélisation des interactions homme-machine

2.2.2.1. Introduction

Ces applications utilisant les réseaux de Petri pour modéliser des systèmes d'informations aident les concepteurs à créer leurs systèmes. Les réseaux de Petri vont représenter ces systèmes de manière graphique et ainsi aider l'utilisateur à mieux comprendre les relations entre les composantes de ses systèmes et donc à concevoir ses systèmes.

Afin d'améliorer le langage de communication entre concepteurs et utilisateurs lors de la création de systèmes, il semble pertinent de définir un langage de description de ces systèmes. Pour définir un tel langage on doit tenir compte des concepts nécessaires pour décrire le SI du point de vue des

utilisateurs. Ceux-ci désirent qu'un tel langage possède les caractéristiques suivantes [OBERQUELLE86] :

1) le langage doit offrir la description du système d'interactions entre les hommes et les machines, c'est-à-dire son environnement de travail. Ceci afin de réaliser deux objectifs :
   - la compréhension des changements dans l'organisation suite à l'introduction de machines, de programmes.
   - la détermination exacte du champ d'action des tâches automatisées.
2) le langage dot être tel que le SI décrit est aussi transparent que possible et le langage doit utiliser les avantages des textes et des graphiques. Les graphiques visualiseraient les relations complexes entre les entités du système et les textes complèteraient la compréhension du graphique.
3) et finalement le langage doit pouvoir réaliser des descriptions imprimables de grande qualité pour faciliter la lecture et l'étude de la description.

Quant aux concepteurs, ils souhaitent que ce langage, qui demande la production de graphiques et leur modification fréquente lors des étapes de développement, soit réalisé sur ordinateur afin de produire les résultats des descriptions de manière automatique.

Une production informatisée de descriptions de systèmes de graphiques semi-formels à haute qualité d'impression basée sur un ensemble de concepts orientés utilisateurs semble être un but intégrant les intérêts des concepteurs et des utilisateurs.

Le problème est donc maintenant de définir un ensemble de concepts orientés utilisateurs capables de décrire les SI ainsi que leur représentation graphique associée.

2.2.2.2. Les concepts des interactions homme-machine

Ce paragraphe décrit le cadre des interactions homme-machine [OBERQUELLE86].

On peut prétendre que le travail des personnes dans une organisation peut être décrit en terme d'ensemble de rôles-coopérants et que les aspects pertinents peuvent être traités sur trois niveaux : le niveau du rôle, le niveau de la fonction et le niveau de l'action.

1) niveau du rôle

Au niveau du rôle, on s'occupe du travail, c'est-à-dire de la relation entre les personnes et les parties du travail. Un ROLE est une tâche qui a un sens pour une personne appelée JOUEUR, et qui comprend :
- la responsabilité pour l'accomplissement d'une tâche
- la compétence pour gérer les degrés de liberté de la tâche
- les droits et les devoirs envers les autres rôles et leurs joueurs.

Les joueurs COOPERENT avec les rôles voisins et échangent objets, informations ou réalisent des opérations communes.

Une personne peut jouer plusieurs rôles.

Chaque rôle peut contenir un ou plusieurs SOUS-ROLES qui contribuent à l'accomplissement de la tâche.

2) niveau de la fonction

Au niveau de la fonction, on s'occupe de l'organisation du travail. Une FONCTION conceptuellement comprend une ACTIVITE séquentielle et ses ressources locales.

Toute fonction est réalisée par un ACTEUR, personne ou machine. Le joueur peut être l'acteur pour les fonctions non automatisées de son rôle.

Les fonctions utilisent de l'espace organisationnel, c'est-à-dire les POSITIONS où un objet peut être localisé. L'espace local d'une fonction est appelé DEPOT, et l'espace qu'une fonction a en commun avec une autre fonction, est appelé l'INTERFACE. Une

fonction peut INTERAGIR avec d'autres fonctions par l'échange d'objets ou de données à travers des interfaces ou lors d'opérations communes.

3) niveau de l'action

Au niveau de l'action, on se concentre sur la dynamique du processus de travail. On vérifie que tout se passe selon les plans que connaissent les acteurs et qui sont décrits explicitement.

La plus simple unité de l'activité d'une fonction est appelée OPERATION ELEMENTAIRE. Les opérations sont exécutées par les acteurs et peuvent être utilisées pour transformer les objets localisés dans les positions.

La relation entre opération et acteur est appelée aspect du contrôle. Il peut être caractérisé en terme d'ETAT des acteurs en fonction de leur rôle et l'état est changé par les opérations. Chaque fonction possède un état initial pour son acteur et chaque acteur doit atteindre un de ses états finals.

La relation entre opérations et objets est appelée aspect de l'objet. Il est caractérisé par les propriétés de l'objet, leurs positions dans l'état organisationnel et les effets que les opérations ont sur lui.

Une DONNEE est un attribut qui peut être copié et rangé dans un objet.

Les opérations travaillent sans flux d'objets, seules les données peuvent faire l'objet d'un flux. Les ACTIONS qui sont des parties dynamiques de fonctions consomment ou génèrent des objets.

2.2.2.3. La représentation : les réseaux RFA

Les réseaux RFA sont des réseaux Rôle/Fonction/Action. Ils modélisent les aspects théoriques décrits ci-dessus.

1) le rapport avec les réseaux de Petri

Les réseaux de Petri décrivent et analysent les systèmes distribués asynchrones et, dans cette description, le graphisme est essentiel. Différents types de réseaux de Petri ont été envisagés pour modéliser le cadre conceptuel décrit précédemment mais aucun n'y correspond parfaitement.

C'est pourquoi on a créé une nouvelle classe de réseaux, les réseaux RFA. Comme tous les réseaux, les réseaux RFA utilisent la notation "classique" : un réseau-RFA N est le triplet (S,T;F) où S est l'ensemble des "états", T l'ensemble des "transitions" et F l'ensemble des "flux".

2) la représentation graphique du cadre conceptuel

• les "états" du réseau-RFA

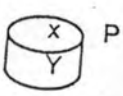
- aspect de contrôle

 = état d'un acteur

- aspect objet

= position élémentaire P d'un objet de type T

= position complexe composées de plusieurs positions élémentaires



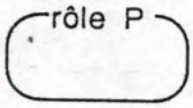
= position de stockage M de classe Y pour objet de type X

 = positions communes

 = position privée

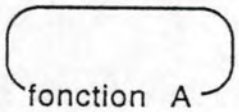
• les "transitions" du réseau-RFA

- niveau rôle

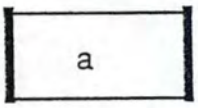


rôle P

- niveau fonction

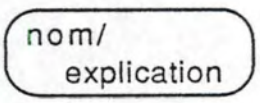


fonction A

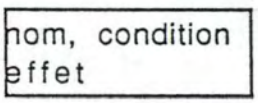


activité

-niveau action

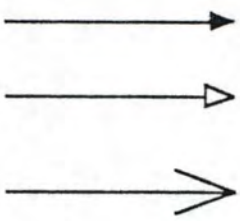


action



opération

• les "flux" du réseau-RFA



flux d'objet

flux de données

flux de contrôle

2.2.2.4. Exemple de réseau-RFA : une bibliothèque

L'exemple support aux réseaux RFA est le cas d'une bibliothèque d'entreprise. La première chose que l'on peut faire c'est décrire l'environnement de cet exemple par le réseau RFA de la figure 2.9.

Sur cette figure, on découvre 2 rôles principaux : le marchand de livres et l'entreprise. L'entreprise possède d'autres rôles secondaires comme ceux de la bibliothèque, du département finance et de la direction. La bibliothèque possède elle-même des sous-rôles répartis sur différentes personnes responsables de celle-ci. Ainsi on trouve le rôle d'administrateur, celui qui administre la bibliothèque (qui commande les livres et qui propose les livres à acheter), le rôle de leader, celui qui dirige la bibliothèque (qui décide des livres à acheter) et le rôle d'utilisateur, qui utilise les livres (qui lit les livres et propose des livres à acheter).
Entre ces trois rôles, on trouve une convention qui régit les rôles de chacun et qui se base sur des flux de données (flèches blanches). Il existe aussi des relations entre l'utilisateur et l'administrateur et entre l'administrateur et le leader. Ces relations seront plus détaillées dans la figure 2.10 où l'on introduit des actions.

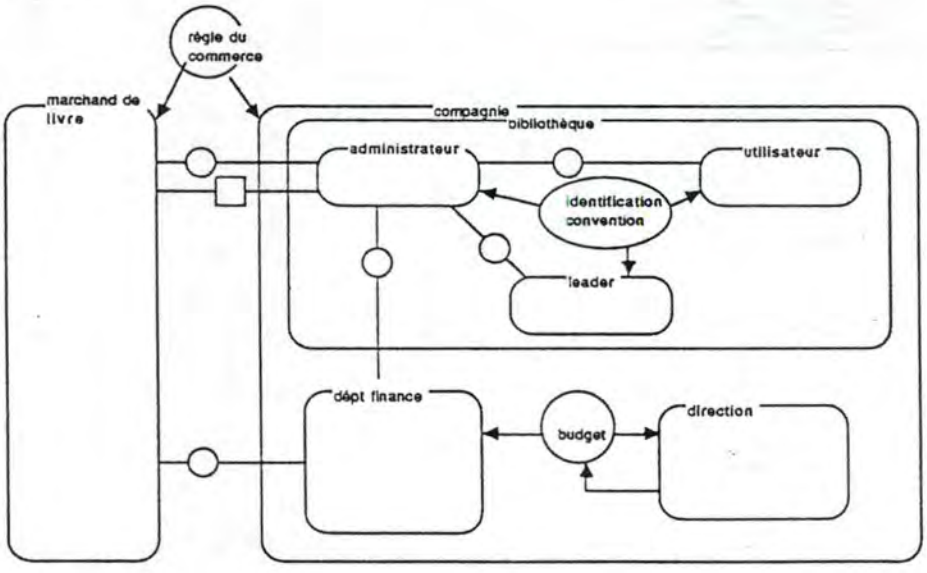


figure 2.9. Schéma de l'environnement de la bibliothèque

Le schéma 2.10 détaille davantage les rôles et les fonctions. Ainsi le rôle de l'administrateur possède deux fonctions : l'acquisition et l'extension. Ces fonctions sont en relation avec celles du leader et celles de l'utilisateur.

+ Le leader possède un sous-rôle de conseiller qui conseille ses propres fonctions d'extension (il se conseille lui-même, il

réfléchit). Il y a deux fonctions d'extension chez le leader car une est relative à une extension future (un projet d'achat de livres) et l'autre est relative à l'extension décidée (les livres que le leader a décidé d'acheter). Ces fonctions sont en relation étroite avec les fonctions de l'administrateur d'acquisition et d'extension.

+ L'utilisateur intervient de deux façons dans : une d'acquisition du livre c'est-à-dire que la bibliothèque de l'entreprise achète pour lui le livre et une fonction d'emploi c'est-à-dire qu'il emprunte un livre à la bibliothèque (ou du moins désire emprunter). Cette seconde fonction de l'utilisateur est en relation avec les fonctions de l'administrateur d'acquisition et d'extension. Car en effet l'utilisateur peut demander à l'administrateur d'acheter un livre et il ne peut utiliser qu'un livre acquis.

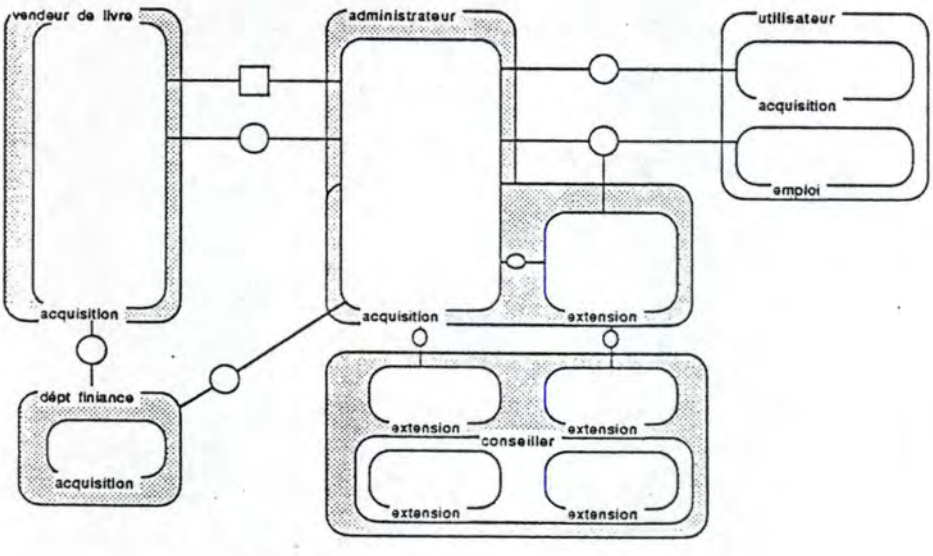


figure 2.10. Schéma des rôles et fonctions

L'étape suivante dans la description du SI est l'introduction du flux des données. La figure 2.11 décrit partiellement la façon dont on doit procéder pour commander un nouveau livre.

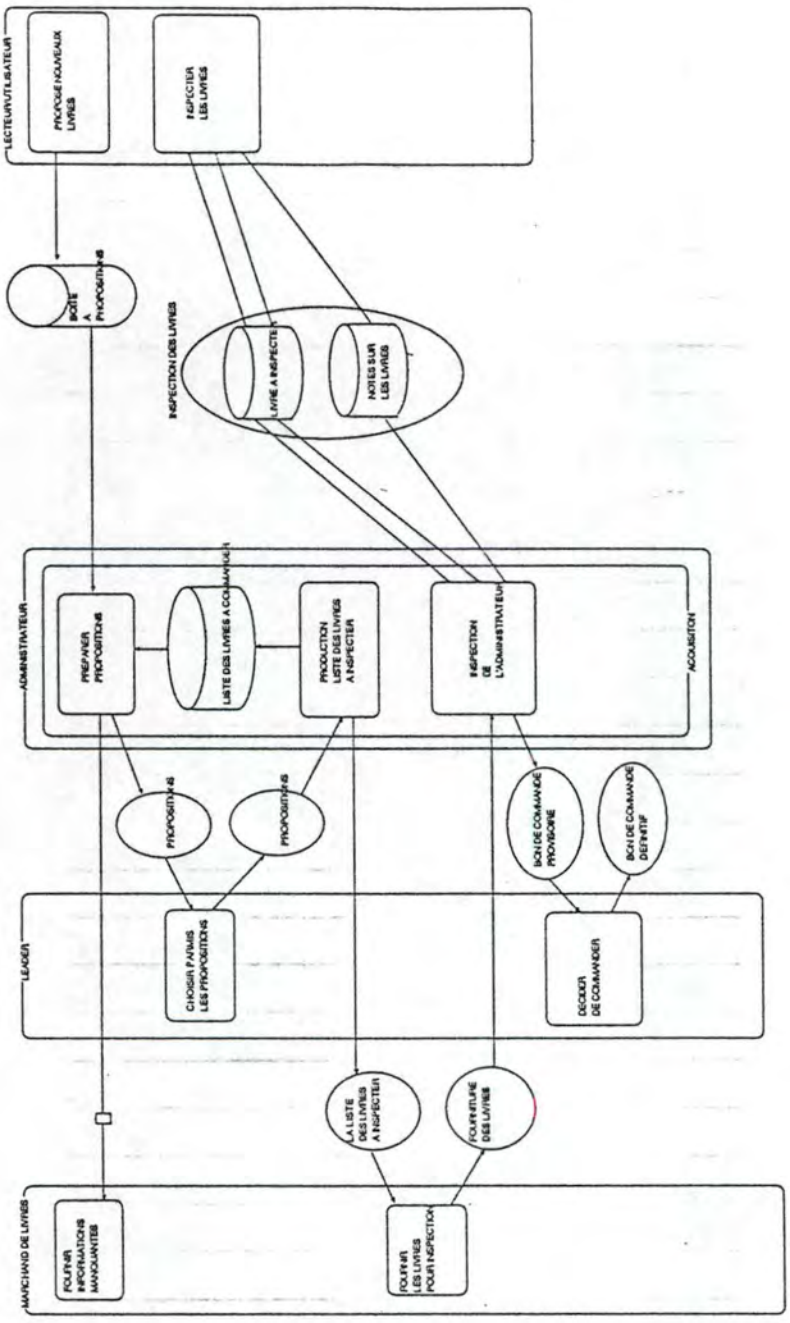


figure 2.11. Schéma de commande d'un livre

2.2.2.4. Conclusion

On a présenté un cadre conceptuel et un ensemble d'interprétations des réseaux RFA pour la description orientée utilisateurs d'un environnement de travail. Ils couvrent les interactions homme-machine comme un phénomène organisationnel dans la perspective de l'utilisateur.

La cohérence entre les descriptions partielles aux différents niveaux conceptuels et au niveau des détails est réalisée en ayant en commun le même formalisme de description et en utilisant des abstractions communes. Il ne faut donc pas "apprendre" un nouveau formalisme pour chaque niveau de modélisation: un seul langage permet de décrire du niveau le plus abstrait au niveau le plus détaillé.

Les possibilités graphiques ont été exploitées afin de mettre en évidence les concepts communs, les différences, et de rendre visibles les relations complexes.

2.2.3. La conception des systèmes de production

2.2.3.1. Introduction

Nous allons dans ce paragraphe essayer de montrer les apports des réseaux de Petri à la conception de système de production industrielle [VALETTE86, HOLLIGER85]. Nous exposerons une méthodologie de conception des systèmes de production et l'illustrerons par un exemple. L'exemple qui sert de support à cette conception est le domaine des filatures industrielles, et principalement de la manipulation des bobines de fil.

2.2.3.2. La méthode

première étape

Une première étape consiste à définir les produits et les procédés, et à observer les équipements et leur implantation ainsi que le mode d'exploitation de l'atelier et son système de commande.

a) définition des produits et des procédés

L'atelier considéré produit des bobines de fil textile. La production de bobines est regroupée par palettes de qualité homogène car les bobines sortant de cet atelier sont de tailles

variables, le fil se cassant parfois ce qui produit des bobines moins remplies.

Les objets utilisés dans cet atelier sont un présentoir sur lequel tournent plusieurs bobines (24 en général). Le long de ce présentoir se déplacent deux objets : un système automatique qui peut changer les bobines pleines et placer des tubes vides et une navette qui transporte les tubes de carton vers le système automatique et les bobines pleines vers le "stock". Cet atelier est représenté par la figure 2.12.

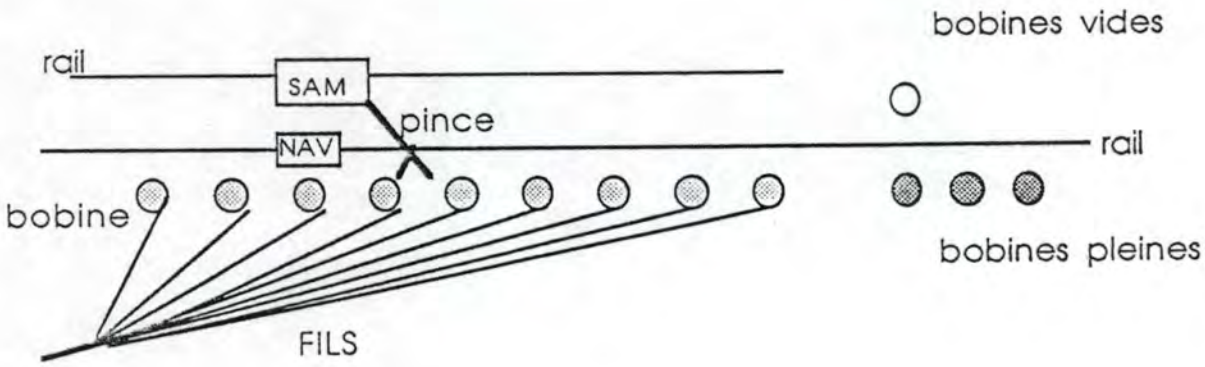


figure 2.12. Représentation de l'atelier de bobinage

On peut donner une description synthétique du problème sous la forme de la figure 2.13. Le problème c'est qu'ayant un planning à respecter, des tubes de carton, du fil, des accessoires de palettisation et des utilités à sa disposition, il faut arriver à un certain état de production avec certains déchets de fil et des ensembles de bobines stockées en palettes. Ce résultat est obtenu en réalisant les fonctions de production de bobines, de contrôle des tailles de bobines et des équipements, du tri des bobines selon leur taille de remplissage et du conditionnement en palette des bobines de même taille.

ENTREE	FONCTION	SORTIE
1 planning 2 tubes carton 3 fil 4 accessoires de palettisation 5 utilités	1 production des bobines 2 contrôle 3 tri 4 conditionnement	1 état de la production 2 déchets de fil 3 palettes de bobines identifiées

figure 2.13. Le procédé

La production de bobines s'effectue sur un bobinoir et lorsque la bobine atteint la taille voulue, le fil est coupé et dévié vers un collecteur de déchets. On retire la bobine, puis on replace un carton vide auquel on attache le fil.

Le processus de production est décrit par la figure 2.14. Connaissant la date d'achèvement du remplissage d'une bobine, ayant un tube de carton vide sur la navette et un fil qui s'enroule, il faut que l'on retire la bobine achevée, ce qui produit des déchets et initialise une nouvelle date de fin de remplissage pour le carton mis à la place de la bobine retirée. Ce processus est réalisé par l'intermédiaire de plusieurs fonctions automatiques comme la coupe du fil sur la bobine pleine, le retrait de la bobine, la pose du nouveau carton, l'accrochage du fil sur ce carton et le bobinage du carton.

ENTREE	FONCTION	SORTIE
1 date d'achèvement 2 tubes carton 3 fil	1 coupe du fil et déviation 2 retrait bobine 3 pose du carton 4 accrochage du fil 5 bobinage	1 déchets 2 bobine achevée 3 date d'initialisation bobine

figure 2.14. Le processus de production de bobine

Mais le processus décrit ci-dessus qualifié de "normal", est perturbé par de fréquentes casses de fil. Du fait de sa fréquence, ce

processus du traitement de casse du fil est pris en compte et est décrit dans la figure 2.15. Si une casse intervient, ayant un tube de carton vide et du fil, alors on produit des déchets, on retire une bobine "achevée" et on replace un nouveau carton qui initialise une nouvelle date de terminaison. Ce processus est réalisé à l'aide des fonctions automatiques de détection de la casse, par la déviation de fil vers les déchets, le retrait de la bobine "achevée", la pose du nouveau carton, l'accrochage du fil et le bobinage.

ENTREE	FONCTION	SORTIE
1 occurrence de casse	1 détection	1 déchets
2 tubes carton	2 prise du fil et	2 bobine achevée
3 fil	déviation aux	3 date d'initialisation
	déchets	bobine
	3 retrait bobine	
	4 pose du carton	
	5 accrochage du fil	
	6 bobinage	

figure 2.15. Le processus de casse

Le fonctionnement du bobinoir peut alors être représenté schématiquement par le réseau de Petri de la figure 2.16.

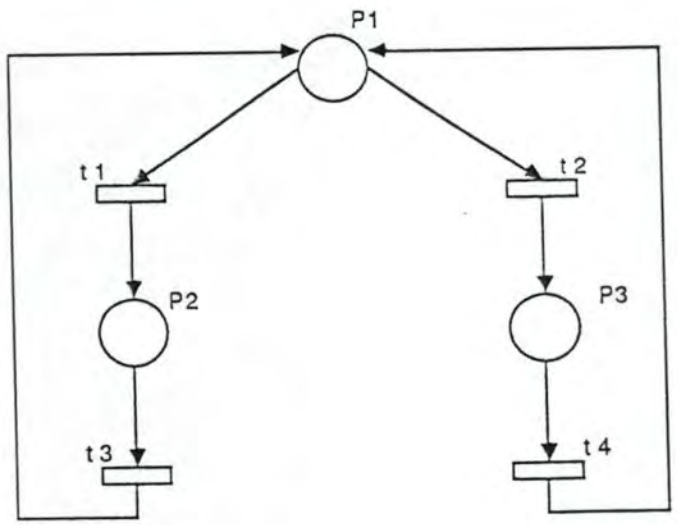


figure 2.16. Le réseau de Petri du fonctionnement d'un bobinoir

Légende de la figure 2.16

les places
p1 : bobinage en cours
p2 : exécution des fonctions 1 à 4 du traitement normal (figure 2.12)
p3 : exécution des fonctions 1 à 5 du traitement de casse (figure 2.15)

les transitions
t1 : coupe du fil
t2 : casse du fil
t3 : fin d'accrochage du fil

b) définition des équipements et implantations

On doit connaître les équipements, par exemple savoir que les bobinoirs sont regroupés en métier de 24 positions alignées. Et à partir de cela, raisonner sur la notion de position qui repère :
- l'emplacement du système automatique mobile SAM
- l'emplacement de la navette qui transporte les tubes de carton NAV
- l'emplacement de l'opérateur qui traite les casses.

c) définition de l'exploitation de l'atelier

Cette exploitation consiste à commander le système automatique SAM jusqu'à une position de bobine qui a atteint une taille nominale et de traiter le cas "normal" de changement de bobines. SAM, quant à lui "commande" la navette afin qu'elle arrive avec un carton vide et reparte avec la bobine pleine.

d) définition de l'architecture de commande

La structure liée au métier gère les chronomètres des bobines (taille nominale), active les tâches des équipements et gère SAM.

La structure liée à SAM reçoit les ordres de déplacement, de coupe,... commande les mouvements élémentaires, émet les comptes rendus et contrôle la synchronisation de NAV.
La structure liée à NAV reçoit les ordres de chargement de tubes et commande le déplacement de NAV.

deuxième phase

On va recueillir des données afin de mieux spécifier les produits, les gammes de produits et quantifier ces spécifications.Le recueil des données techniques auprès des ingénieurs permet de spécifier plus finement les fonctions décrites ci-avant (figure 2.13;2.14;2.15). Par exemple la prise de la bobine pleine doit être effectuée par l'intérieur des cartons pour ne pas abîmer la bobine de fil.

On va créer une documentation pour disposer d'une spécification littérale des fonctions où l'on va transcrire les marquages de début et de fin d'évolution du système en langage clair sur un diagramme fonctionnel. Pour cela on utilise des réseaux de Petri.

troisième phase

Cette troisième phase est dite phase de modélisation. On va réaliser une nomenclature des places et des transitions afin d'obtenir des réseaux de Petri traduisant le fonctionnement de la partie opérative du système.

exemple : la représentation du traitement normal

places :

samp1 : état de SAM au repos devant la dernière position
samp2 : SAM en déplacement
samp3 : SAM exécutant la partie préparatoire du cycle
samp4 : SAM en attente de NAV
samp5 : exécution de l'échange de la bobine pleine et du carton vide

navp1 : état de NAV au repos
navp2 : chargement de tube
navp3 : déplacement aller NAV
navp4 : attente SAM
navp5 : déplacement retour NAV
posp1 : bobine en cours de constitution
posp2 : fil évacué aux déchets

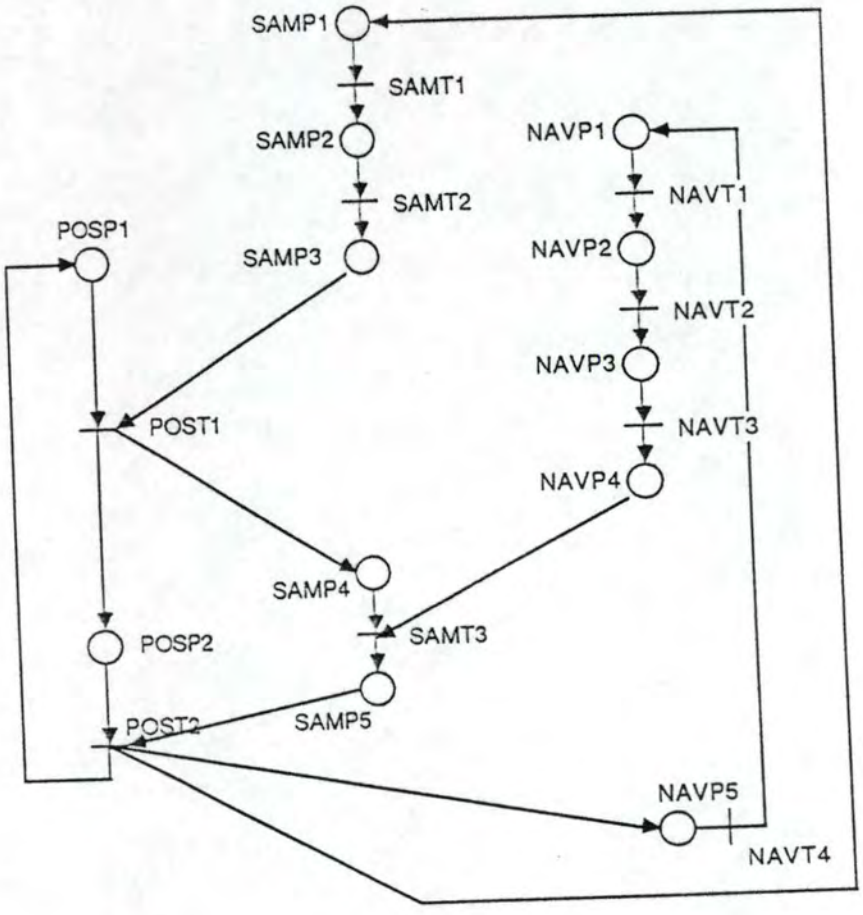


figure 2.17. Le réseau de Petri du traitement du cas normal de rebobinage

transitions :

post1 : coupe du fil par SAM
post2 : initialisation de la nouvelle bobine par SAM
samt1 : mise en route de SAM
samt2 : arrivée de SAM à la position

samt3 : synchronisation SAM-NAV
navt1 : ordre de chargement
navt2 : mise en route NAV
navt3 : arrivée de NAV en position
navt4 : début retour de NAV

Cette figure 2.17 modélise l'exécution du bobinage normal. Au départ SAMP1, NAVP1 et POSP1 sont marqués par un jeton, ce qui veut dire que SAM est en position devant la dernière bobine qu'il a changée, NAV est au repos à sa position de départ et la bobine est en cours de construction. Le chronomètre va alors commander à SAM de se déplacer (SAMT1) car une bobine est bientôt pleine. SAM commande à NAV de se charger (NAVT1) d'un carton vide puis de se mettre en route (NAVT2) vers la position où il faut changer la bobine. Le système automatique va alors arriver en position devant la bobine à changer (SAMT2) puis il se prépare au changement (SAMP3). Le système automatique peut alors couper le fil (POST1) et est en attente de l'arrivée de la navette. Dès que NAV est arrivée (NAVT3) alors SAM et NAV se synchronisent pour effectuer simultanément les opérations de permutation de bobines. Après cette permutation la navette retourne se décharger à sa position de repos (NAVT4) et on évacue les déchets (POST2). On se retrouve alors dans la situation de départ.

Cette opération sera répétée sur les différents cas observables afin d'obtenir plusieurs modélisations sous forme de réseaux de Petri relativement simples. De ces réseaux on va tirer le graphe des états accessibles qui représente le modèle du comportement du métier.

Sur ce modèle global on peut vérifier certaines propriétés classiques : réseau borné, réseau vivant, réseau réinitialisable.

A ce modèle on ajoute l'ensemble des messages échangés entre les équipements et les systèmes de contrôle. On obtient ainsi le modèle complet de la phase de bobinage.

quatrième phase

Cette phase sert à évaluer les performances du métier modélisé. Elle permet donc de vérifier certaines propriétés du réseau. On va trouver cette phase dans la partie 2.3 parlant des aspects de vérification des réseaux de Petri.

2.2.3.3. Conclusion

Dans l'application des réseaux de Petri aux phases d'étude décrites précédemment, nous pouvons relever ses points forts et ses points faibles qui peuvent être imputés, selon le cas, au formalisme des réseaux lui-même, ou à l'environnement du projet pris comme exemple.

Parmi les points forts il y a :
- la facilité de dialogue entre exploitant du procédé étudié et l'ingénieur d'étude. Après présentation des réseaux de manière intuitive, il est possible de dialoguer à l'aide de réseaux de Petri.
- la rigueur de la modélisation, l'alternance place/transition est toujours respectée et la règle de passage de transition ne varie jamais.
- la finesse de modélisation: les réseaux de Petri sont des outils de raffinement progressif et modulaire.
- un fort pouvoir de modélisation

Parmi les points faibles il y a :
- de grandes difficultés à modéliser un système traversé par des flux distincts (non différenciabilité des jetons) dans les réseaux de Petri traditionnels, d'où la nécessité de réseaux colorés.
- des difficultés à modéliser des opérations simples telles que les compteurs, d'où la nécessité de réseaux à prédicats .

2.2.4. Conclusion

Les réseaux de Petri sont bien adaptés à la modélisation de systèmes. Ils permettent de modéliser de nombreux types de systèmes et donc d'être disponibles pour de nombreuses applications.

Un de leurs grands avantages est la facilité de manipulation des réseaux de Petri, même par des personnes non spécialistes en informatique. De plus, de tels modèles sont aisément compréhensibles par toutes les personnes concernées par le projet modélisé, aussi bien l'utilisateur que l'informaticien ou le concepteur.

Les réseaux de Petri ont aussi le grand avantage d'être précis et d'être toujours interprétés de la même façon. La règle de tir des transitions est la même pour tous les modèles réalisés avec des réseaux de Petri.

Ils permettent aussi de réaliser des abstractions en donnant à certaines places la valeur d'un sous-réseau, et donc de traiter des problèmes possédant une grande taille, par parties. De plus, l'abstraction combinée au raffinement permet de partir d'un modèle abstrait et, par raffinements successifs d'obtenir le modèle désiré.

Un désavantage des réseaux de Petri classiques tels que définis par Petri est de ne pas permettre de différencier les objets que l'on y dépose (les jetons sont non différenciables). On doit donc fréquemment utiliser des extensions (abréviation) des réseaux de Petri soit des réseaux de Petri colorés, soit des réseaux de Petri à prédicats.

Un autre désavantage des réseaux de Petri dans la modélisation est la difficulté de représentation des problèmes qui possèdent une grande taille car alors la multiplicité des places et des transitions rend l'analyse et la représentation graphique difficiles.

2.3. LES APPLICATIONS DE VÉRIFICATION

Ces applications ont pour but d'aider les utilisateurs à vérifier et analyser certaines propriétés de systèmes modélisés par des réseaux de Petri. On suppose que l'utilisateur a, dans une étape précédente, réussi à modéliser de façon correcte le système dont il veut obtenir certains résultats.

On utilise notamment les réseaux de Petri pour vérifier la rentabilité de systèmes de production [HOLLIGER85], pour vérifier des programmes temps réel [LAMARCHE85], vérifier des algorithmes [KERHERVE85] et des protocoles de communication [DIAZ86].

2.3.1. La rentabilité de systèmes de production

Les systèmes de production peuvent être modélisés par des réseaux de Petri (cfr 2.2.3) et cette modélisation peut être à la base de calcul de rentabilité et de mesure de performance des métiers à bobiner modélisés. Dans la phase d'évaluation, on va réaliser ces calculs.

Cette phase sert à l'évaluation. Elle répond à l'objectif de détermination des performances requises des équipements pour un objectif de production donné et de calcul des gains de productivité justifiant l'investissement. L'établissement du rendement de l'atelier de bobinage s'effectue en écrivant un modèle de simulation de l'atelier, puis en procédant à une campagne d'essais.

Le modèle qui va permettre une telle opération est issu des modèles développés précédemment lors des phases précédentes. En y incorporant des procédures statistiques, on obtient des statistiques servant à mieux cerner le rendement de l'atelier.

On va donc passer d'une modélisation en réseaux de Petri classique à une modélisation utilisant les réseaux de Petri stochastiques. Ces réseaux de Petri stochastiques possèdent des propriétés particulières déterminant le comportement des

transitions de manière aléatoire et permettant une analyse statistique fine de l'évolution du modèle. Ces statistiques permettent d'établir à priori un rendement des métiers et donc de voir si l'emploi d'un tel métier pour un tel type de fil est optimal quant à l'investissement à réaliser.

Ces modèles ainsi obtenus permettent une meilleure gestion des métiers, en allouant de la meilleure façon possible les métiers à un plan de production donné.

2.3.2. Les tests de programmes temps réel

2.3.2.1. Introduction

Les logiciels temps réel, caractérisés par la présence de traitements effectués en parallèle avec de sévères contraintes de temps se voient souvent confier la réalisation de fonctions critiques pour le système où ils interviennent. Il importe donc de protéger ces programmes de la présence d'erreurs, ce qui se traduit par une phase de test longue et coûteuse.

De plus, si l'on sait relativement bien conduire les tests des programmes séquentiels, il n'en va pas de même pour les programmes parallèles. Ainsi certains aspects du programme ne sont pas, faute de moyens, contrôlés; il s'agit des tests portant sur :

- les contraintes de synchronisation, exclusion,...
- les contraintes de date ou de durée d'exclusion.

Le programmeur se trouve réellement démuni pour effectuer les vérifications portant sur les premières contraintes. Ainsi effectuer la simple vérification, du fait que deux séquences s'exécutent simultanément, ne pourra être effectuée qu'après une mise en oeuvre laborieuse d'un équipement.

Enfin il est nécessaire que les tests puissent s'effectuer sans perturber le programme à tester et donc, en particulier, sans instrumentation de ce dernier.

Une solution à ces problèmes est de créer un outil qui s'appuie sur le concept d'observateur qui consiste à faire évoluer en parallèle le programme et un modèle de celui-ci et à comparer ces derniers en permanence (figure 2.18).

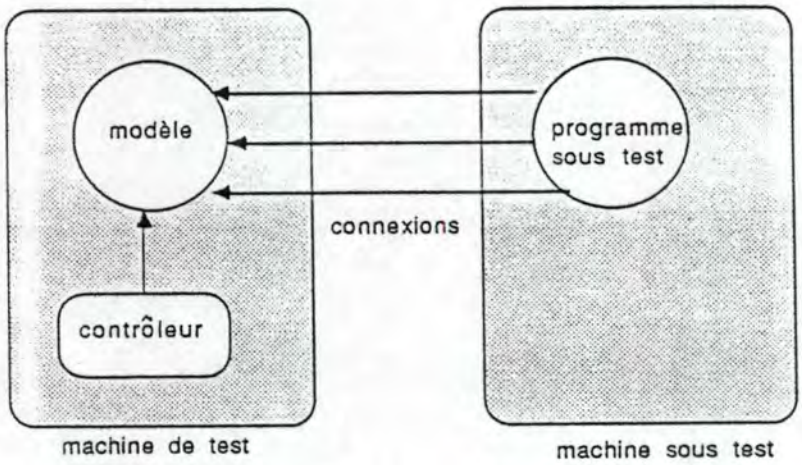


figure 2.18. Schéma de l'observateur

L'outil proposé manipule :

- le programme sous test : inchangé par rapport au programme définitif
- le modèle : décrit à l'aide d'un réseau de Petri le comportement de référence que doit respecter le programme sous test.
- les connexions : qui permettent d'établir une correspondance entre le programme et le modèle afin que le "contrôleur" puisse vérifier qu'ils évoluent de manière cohérente.
- le contrôleur : activé à chaque passage du programme sur un point de contrôle, qui compare l'évolution du programme à l'état du modèle et fait évoluer ce dernier en conséquence.

exemple : l'exclusion mutuelle

Il est parfois nécessaire dans les programmes temps réel de s'assurer que deux séquences d'instructions ne s'exécutent jamais simultanément (exclusion mutuelle). Cette propriété peut être modélisée par un réseau de Petri (voir figure 2.19). Chaque transition est connectée par un point de contrôle au début ou à la fin des séquences d'exclusion. Le contrôleur activé à chaque

passage du programme sur ces points de contrôle s'assure que les jetons sont dans les bonnes places au bon moment et les fait évoluer en conséquence.

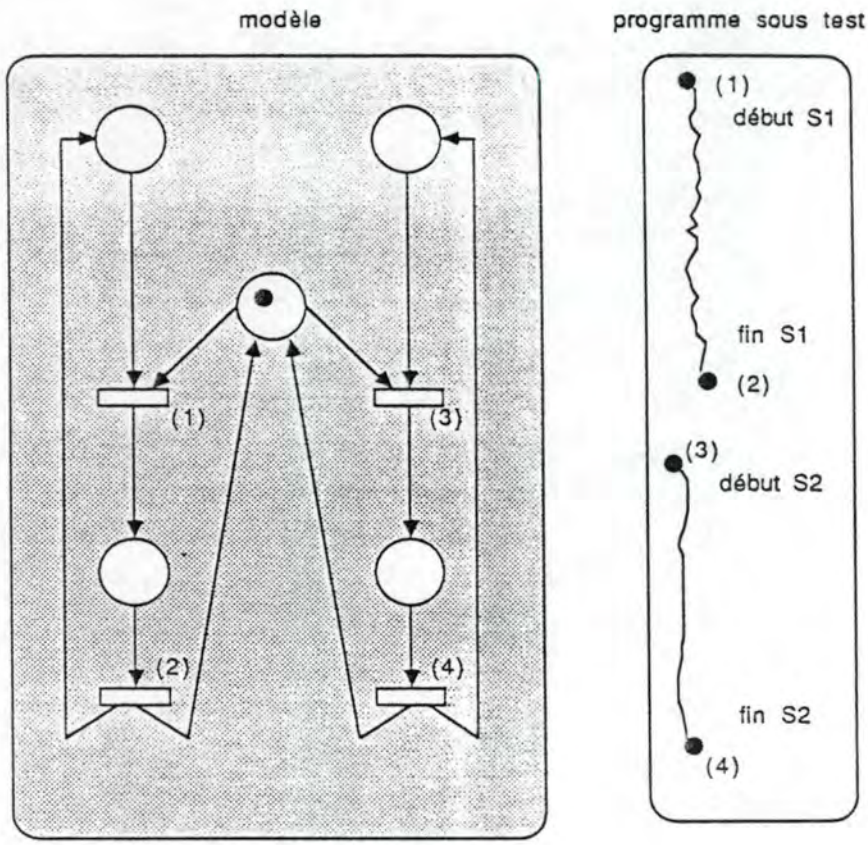


figure 2.19. Contrôle d'exclusion de deux séquences parallèles.

2.3.2.2. Le type de réseau utilisé

Afin de donner à l'outil une plus grande puissance et de lui permettre de modéliser des comportements faisant intervenir le temps, il a été nécessaire d'étendre le modèle de base des réseaux de Petri.

1) le temps

Dans un réseau de Petri, le franchissement d'une transition est instantané; il a lieu dès que la transition est franchissable c'est-à-dire au moment où toutes les places précédant la transition possèdent un nombre de marques au moins égal au poids de l'arc correspondant.

Afin de simplifier les modèles d'une part et de permettre la modélisation du temps d'autre part, l'outil définit un nouveau type de transition qui, à la différence de la transition habituelle, n'est plus à franchissement instantané.

L'intérêt principal de cette modification, outre la diminution du nombre de places et de transitions dans la réalisation du modèle, est de permettre d'associer à chaque transition un traitement et à chaque traitement une durée minimale et une durée maximale. Le contrôleur devant s'assurer que la durée effective de la transition respecte bien la fourchette souhaitée.

2) les prédicats

A chaque transition est associé un prédicat dont la valeur détermine, en plus de la présence des marques dans les places, la franchissabilité de la transition. Ce prédicat est évalué par le contrôleur lors de l'arrivée du point de contrôle correspondant au début de la transition et permet donc d'étendre les possibilités de modélisation du réseau.

De même, une action est associée à chaque transition; celle-ci étant exécutée à la fin de la transition correspondante.

3) les objets

L'extension proposée comprend donc :
- un ensemble de transitions
- un ensemble de places
- une relation transition-place
- une relation place-transition
- une pondération des arcs

A chaque transition sont associés :
- un point de contrôle de début
- un point de contrôle de fin
- un prédicat
- une action
- une durée minimale
- une durée maximale
- une variable "durée de transition"

4) l'algorithme du contrôleur

Lors de l'arrivée d'un point de contrôle de début de transition, le contrôleur
- vérifie que la transition est franchissable et la met à feu;
- évalue le prédicat et signale une erreur si celui-ci est faux;
- met la transition dans l'état "actif".

Lors de l'arrivée d'un point de contrôle de fin de transition, le contrôleur :
- vérifie que la transition est dans l'état actif et la désactive;
- ajoute, dans chaque place, suivant la transition, un nombre de jetons égal au poids de l'arc correspondant;
- calcule la durée de la transition et vérifie qu'elle est comprise dans la fourchette;
- exécute l'action associée à la transition.

5) La mise en oeuvre

La mise en oeuvre de la méthode de test qui vient d'être proposée nécessite :
- d'établir un modèle servant de référence de test et de le connecter au programme à tester;
- de disposer d'un environnement matériel et logiciel pour décrire le test et réaliser les observations nécessaires.

L'établissement du modèle et la preuve de la validité de celui-ci restent de l'entière responsabilité du programmeur. Cette construction peut être envisagée avec l'aide d'un outil manipulant les réseaux de Petri et pouvant aider à la démonstration de certaines propriétés du réseau mais pas à démontrer que le modèle correspond effectivement à la propriété que souhaite tester l'utilisateur.

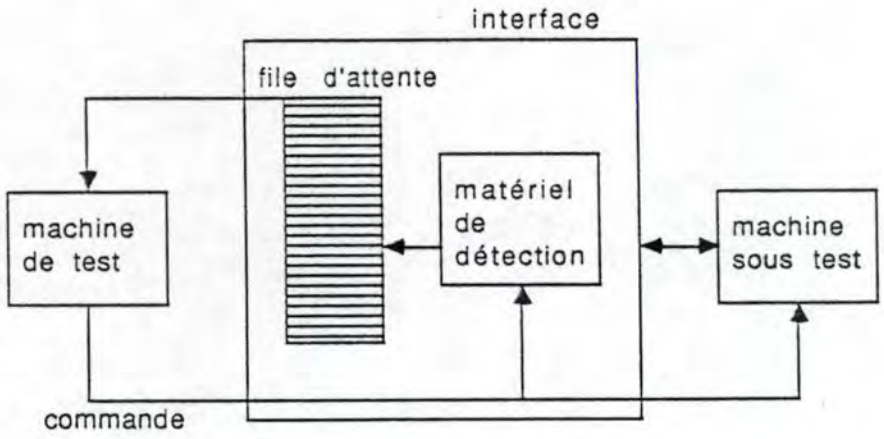


figure 2.20. Configuration de mise en oeuvre

L'environnement matériel nécessaire est composé de deux machines (figure 2.18), celle qui tourne sous le programme à tester et la seconde qui tourne sous le modèle. Un flux d'informations va de la première à la seconde et est constitué des différents événements rencontrés ainsi que de la datation de ceux-ci (figure 2.20). Ce flux est stocké dans une file d'attente avant d'être analysé par la machine sur laquelle tourne le modèle. Ce flux est analysé dans la machine du modèle et le comportement du programme sous test est comparé avec l'évolution du modèle suite aux événements reçus.

2.3.2.3. Conclusion

L'utilisation des réseaux de Petri permet au programmeur d'envisager des tests jusqu'ici irréalisables. Les modèles que celui-ci doit écrire ne concernent que le contrôle particulier à effectuer

et non pas l'ensemble du programme. Les réseaux manipulés restent donc simples.

Une utilisation très prometteuse consiste à définir une fois pour toutes les propriétés que l'on veut contrôler (par exemple le respect de certaines contraintes de temps) et à installer ce contrôle dans la machine de test pour toute la durée des essais, voire même jusqu'à la phase d'exploitation. Si, au cours des essais, la vérification échoue ne serait-ce qu'une fois, une alarme pourra être déclenchée. On dispose donc d'un moyen puissant de contrôle de l'intégrité du système en rejoignant ainsi les motivations initiales ayant conduit à "l'observateur".

2.3.3. La vérification de protocoles de communication

Les réseaux de Petri peuvent aussi être utilisés pour modéliser et vérifier les protocoles de communication. Nous consacrerons à cette utilisation typique des réseaux de Petri un chapitre entier, le chapitre 3, car ce fut cette application que nous avons traitée lors de notre stage au LAAS de Toulouse.

2.3.4. Conclusion

L'analyse des modèles réalisés sous la forme de réseaux de Petri permet de tirer certaines conclusions quant aux propriétés des systèmes modélisés. Les propriétés des réseaux de Petri (cfr chapitre 1) peuvent, dans certains cas être interprétées et transposées dans le modèle et donner des bribes de vérification du modèle.

2.4. CONCLUSION

Les réseaux de Petri permettent de modéliser aisément des systèmes de productions, des relations homme-machine, des environnements de problèmes, des programmes temps réel répartis et encore un bon nombre d'autres systèmes.

La modélisation graphique par réseaux de Petri permet une compréhension aisée des problèmes représentés et une évolution facile des modèles par fusion, abstraction, raffinement et division, mais ne permet pas de traiter des problèmes dont la taille est trop élevée.

Les réseaux de Petri traditionnels, comme définis par Petri, sont généralement insuffisants et doivent être étendus pour être applicables à des problèmes réels. On utilise comme principales extensions les réseaux de Petri colorés, les réseaux de Petri à prédicats, les réseaux de Petri temporisés et les réseaux de Petri stochastiques. Le type d'extension choisi est en rapport avec le problème à traiter.

Les réseaux de Petri ne sont pas seulement un outil de modélisation mais également un outil de vérification. En effet les réseaux de Petri possèdent certaines propriétés remarquables qui peuvent être analysées. La transposition de ces propriétés sur les systèmes modélisés permet de vérifier en tout et en partie certaines propriétés du système.

CHAPITRE 3

LES RÉSEAUX DE PETRI ET LES PROTOCOLES DE COMMUNICATION

Bernd Jodocy

3.0. INTRODUCTION

Un courant actuel est la décomposition de grands systèmes de télécommunication en sous-systèmes qui communiquent entre eux. Dans ce type d'environnement, il est possible de développer de nouvelles applications tirant profit de la répartition des données et des ressources de traitement dans des domaines importants tels que les applications bancaires, les bases de données réparties, le contrôle de procédés industriels, la gestion de production, ...

Etant donné la diversité des applications, un besoin croissant de réseaux de communication de tous types, toutes dimensions et toutes caractéristiques s'est manifesté. Comme les sous-systèmes composant le système global sont souvent réalisés par des constructeurs différents et possèdent fréquemment des architectures différentes, essayer de faire travailler ces éléments ensemble est une tâche difficile.

La plupart des constructeurs d'équipements de télécommunication ont depuis longtemps reconnu que, pour interconnecter efficacement leurs produits variés, ils doivent fixer et respecter une architecture bien définie. Des organismes nationaux et internationaux, comme l'Organisation Internationale de Normalisation (ISO), ont ainsi essayé de répondre à cette nécessité en publiant un modèle de référence pour l'interconnexion de systèmes de communication ouverts, le modèle OSI.

Un autre point important à souligner est que, dans le cas général, les systèmes distribués nécessitent l'échange d'informations nombreuses et complexes entre les calculateurs concernés. Ces interactions impliquent que le fonctionnement du système global ne peut être appréhendé en considérant uniquement les fonctionnements séparés des sous-systèmes constituant le système; il est indispensable, pour obtenir le comportement global, de prendre en compte l'ensemble de toutes les interactions possibles pouvant exister dans le système. Les méthodes de description informelle étant trop imprécises, on a éprouvé , très rapidement, le besoin d'une méthodologie de conception.

La base fondamentale d'une telle méthodologie consiste à spécifier de façon formelle le fonctionnement distribué, en incluant les comportements séquentiels et les interactions entre les composants du système global, car, plus que dans le domaine de la programmation séquentielle, on cherchera à simuler, à tester, mieux, à être assuré du bon fonctionnement de l'application distribuée. Ceci signifie que l'on puisse disposer d'un modèle formel permettant la description et l'analyse de l'application. De plus, pour être opérationnel, ce modèle doit être supporté par un outil logiciel qui non seulement contienne les algorithmes d'analyse que le modèle a permis d'élaborer, mais aussi permette de manipuler des modèles de taille industrielle.

Les réseaux de Petri sont un des tout premiers modèles formels dont la vocation affirmée est de permettre d'exprimer, puis d'analyser des applications parallèles, en particulier d'étudier les problèmes de synchronisation et de concurrence. Il existe d'autres modèles : le calcul de Milner, la logique temporelle, les automates communicants, les types abstraits algébriques.

Dans une première partie, nous considérerons les concepts entourant les applications réparties (concurrence, synchronisation,...). Ensuite, nous expliquerons de manière succincte le modèle de référence de l'ISO. Dans la partie suivante, nous soulignerons la nécessité d'une approche formelle pour la conception de protocole de communication. Dans une quatrième section, nous traiterons de l'utilisation des réseaux de Petri dans le domaine de la conception de protocoles.

3.1. LES SYSTEMES DE TRAITEMENT RÉPARTI .

3.1.1. Présentation générale des systèmes de traitement réparti

Dans les années 1970, alors que la puissance des gros ordinateurs ne cessait de croître, le développement de la mini- et de la micro-informatique, ainsi que l'abaissement des coûts de télécommunication ont permis d'envisager la mise en place d'un nouveau type de systèmes informatiques : les systèmes dits distribués ou répartis, ainsi nommés parce qu'ils sont constitués de plusieurs ordinateurs répartis dans des lieux différents et reliés entre eux soit par le réseau téléphonique général, soit par un réseau spécialisé privé ou public, voire par un réseau satellite. Chaque ordinateur du réseau jouit d'une certaine autonomie et aucun de ses ordinateurs ne doit jouer, en principe, un rôle particulier. Cette décentralisation des moyens de traitement et de stockage sur les différents sites justifie le nom de systèmes répartis ou distribués.

Les exemples suivants montrent comment le contrôle et les données peuvent être distribués sur les différents composants d'un système de traitement de données et comment on peut avoir, d'une part, des systèmes où la distribution n'est que très limitée et, d'autre part, des systèmes où on a recherché à maximiser aussi bien la distribution des ressources matérielles que logicielles. Cette classification a été reprise dans [BOCHMANN83] et complétée par celle de [De BAKKER86] et [HWANG84].

3.1.1.1. Système distribué sur longue distance

a) Accès distant (remote access)

Avec l'avènement de systèmes multi-programmes, qui permettent des services de traitement à plusieurs applications simultanément, il est devenu désirable d'obtenir l'accès à de telles facilités de terminaux se trouvant à différents endroits. Cet accès est en général réalisé via le réseau téléphonique en utilisant des modems pour l'adaptation des interfaces digitales des terminaux et systèmes d'ordinateurs à la nature analogique du support de

transmission téléphonique. Ce type de traitement est utilisé, entre autres dans des applications bancaires, des systèmes de réservation de places d'avion,...

b) Réseaux d'ordinateurs

Tandis que les systèmes discutés ci-dessus permettent l'accès à un seul ordinateur hôte, les réseaux d'ordinateurs procurent l'accès à plusieurs ordinateurs hôtes. Les raisons principales pour la construction de réseaux d'ordinateurs sont :

- donner un moyen d'accès distant à une variété de ressources, par exemple à des processeurs normaux, à des moyens spéciaux comme des calculateurs de grande puissance ou de grande précision, à des applications graphiques, à des bases de données, à des facilités d'échange de messages personnels, ...
- partager ces ressources parmi un grand nombre d'utilisateurs. Si les ordinateurs font tous partie d'un système distribué singulier, les applications et moyens spécialisés peuvent être partagés parmi tous les ordinateurs du réseau
- procurer des possibilités de sauvetage dans le cas de panne d'une des ressources, par exemple en ayant des données redondantes distribuées sur le réseau
- procurer un support de communication fiable pour l'accès de ressources et pour le traitement réparti impliquant plusieurs ressources.

Un aspect important des réseaux d'ordinateurs est l'hétérogénéité des ordinateurs connectés et de leurs systèmes d'exploitation. Pour réaliser des communications entre des programmes d'applications, terminaux, fichiers et base de données sur les différents ordinateurs, un certain nombre de conventions doivent être établies et respectées. Ces conventions dites protocoles, implémentées dans les programmes de communication des ordinateurs et terminaux connectés assurent le transport point-à-point des données entre les entités de communication. Il est visiblement très important de développer des normes de protocoles qui soient appropriées aux systèmes actuels et futurs. En l'absence de telles normes, il est difficile, voire impossible,

d'implémenter des communications entre systèmes d'ordinateurs hétérogènes, c'est-à-dire de types différents.

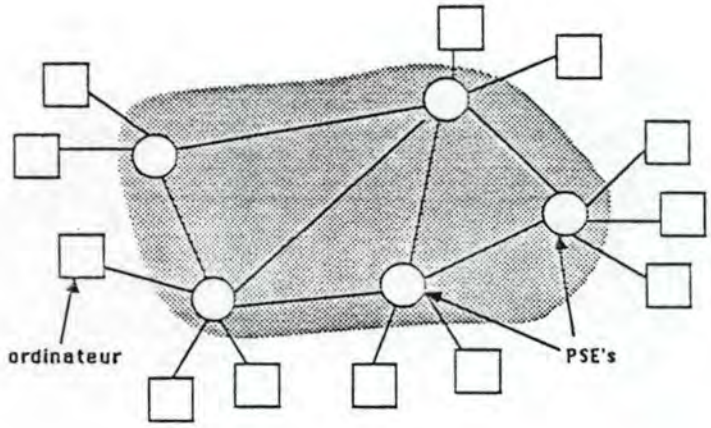


figure 3.1. Réseau d'ordinateurs

c) Système de traitement réparti

Nous considérons dans cette section des systèmes dans lesquels le traitement d'une tâche donnée n'est pas limité à un seul ordinateur hôte mais distribué à travers plusieurs ordinateurs se trouvant à des endroits différents; ceci pour des raisons de performances et/ou de contraintes géographiques.

Un premier principe du traitement réparti est de faire le traitement de l'information là où elle se trouve. Comme, en général, l'information entre à des endroits différents dans le système informatique, ceci signifie que le traitement des données est également distribué. Un exemple d'application de ce principe est que la validation des informations se fait près du terminal d'entrée. Dans le cas d'une base de données distribuée, une enquête pourra accéder à des informations stockées à plusieurs endroits. A nouveau, le traitement est fait où l'information se trouve. Ce principe vise une réduction des échanges de messages. Il est important de réduire les échanges de messages au maximum et d'avoir des liens entre les différents ordinateurs aussi lâches et faibles que possible, car, premièrement, les délais de propagation des messages peuvent être très importants par rapport à la vitesse

de calcul des ordinateurs et, deuxièmement, en cas de défaillance d'un ordinateur local, le fonctionnement d'une grande partie du système ne sera pas empêché.

Un second principe du traitement réparti est la redondance. Comme le traitement est exécuté à plusieurs endroits, il n'est souvent pas trop difficile d'organiser le système tel que les différents centres de traitement peuvent se remplacer mutuellement dans l'exécution du travail. Ceci donne des systèmes très flexibles caractérisés par une "dégradation progressive" dans le cas de défaillances.

Un troisième principe, qui peut être appliqué au traitement réparti, est la construction de systèmes spécialisés. Au lieu d'utiliser des ordinateurs généraux pour exécuter une variété de tâches diverses, les différents composants d'un système de traitement réparti peuvent chacun être spécialisé pour l'exécution d'une tâche particulière. Ceci facilite souvent la conception de chaque composant singulier. La simplification de la conception et l'installation des systèmes répartis grâce à la décomposition du système global en sous-systèmes, chacun d'eux implémenté sur un ordinateur, est un point important car la complexité signifie des coûts élevés de développement, erreurs de logiciels et frais de maintenance accrus.

3.1.1.2. Système distribué à courte distance

En contraste avec les systèmes considérés ci-dessus, les systèmes localement distribués utilisent des moyens de transmissions ne pouvant pas dépasser quelques centaines de mètres. Ils fonctionnent généralement à des vitesses nettement supérieures à ceux des réseaux à longue distance.

Les raisons pour la distribution du traitement sont toujours les mêmes que celles évoqués ci-dessus, à savoir, traitement des données où elles se trouvent, sécurité et disponibilité accrues par la redondance et l'interchangeabilité, et spécialisation des

composants. Cette dernière raison semble être la plus importante pour les systèmes locaux.

Des exemples typiques pour de tels systèmes sont les systèmes de contrôle de processus en temps réel, des systèmes bureautiques intégrés,... La figure 3.2. donne un exemple d'une configuration possible d'un réseau local pour un système distribué.

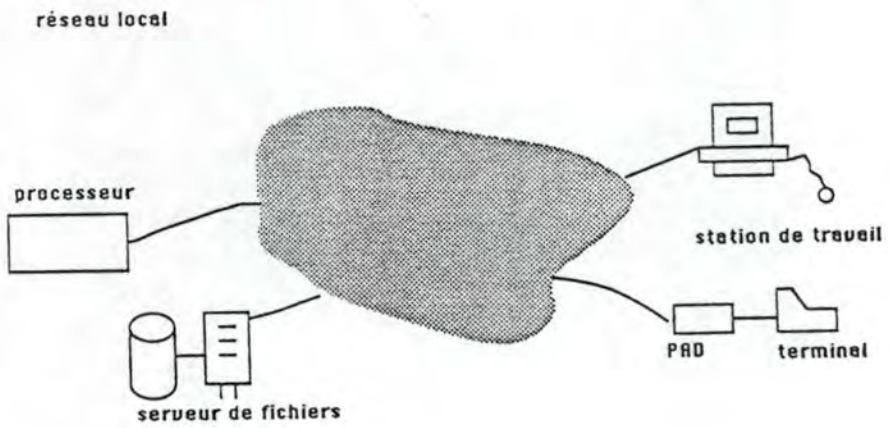


figure 3.2. Configuration possible d'un réseau local

3.1.1.3. Système multi-processeurs

Ce type de systèmes inclut, comme son nom l'indique, un certain nombre de processeurs dans un système hardware intégré sous le contrôle d'un seul système d'exploitation. Le système d'exploitation alloue les processeurs et l'espace mémoire à des "tâches utilisateurs" et leur permet de s'exécuter parallèlement. L'environnement hardware inclut une mémoire partagée ou une connexion à très haute vitesse entre plusieurs systèmes processeur/mémoire avec un système virtuel d'adressage unifié. L'utilisation de la mémoire partagée ou de l'espace d'adressage virtuel permet aux tâches utilisateurs de communiquer entre elles et avec le système d'exploitation à travers des variables et tableaux partagés, tout comme des systèmes conventionnels mono-processeur.

Des systèmes multi-processeurs récents offrent jusqu'à 32 processeurs partageant un unique espace mémoire et d'adressage.

La figure 3.3. illustre la différence entre les principes de systèmes distribués et ceux de systèmes multiprocesseurs.

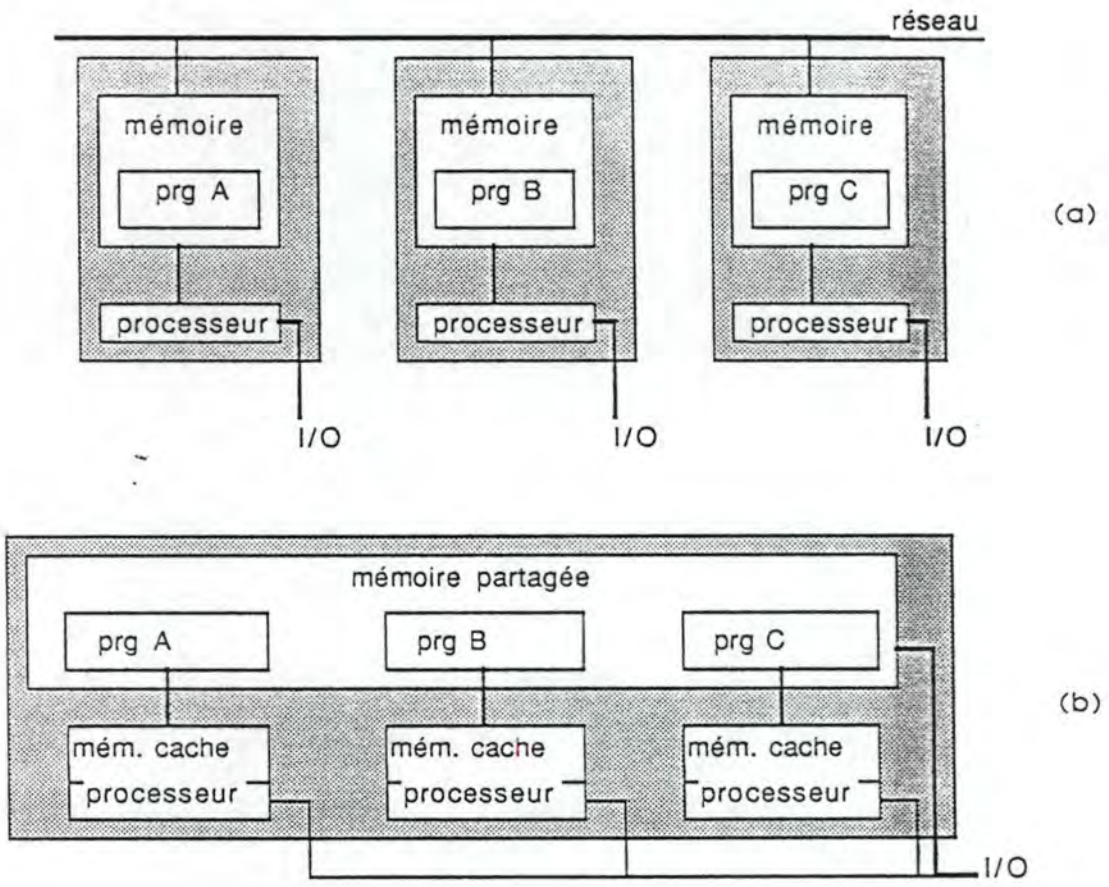


figure 3.3. Système distribué (a) et système multi-processeurs (b)

3.1.1.4. Distribution virtuelle

Nous appelons "distribution virtuelle du contrôle et des données" l'introduction de processus conceptuellement indépendants dans un système dont la réalisation physique ne correspond pas à cette organisation conceptuelle. Un exemple typique est un système d'exploitation, pour un système mono-processeur, qui est conçu comme un ensemble de processus, chacun

exécutant une tâche spécifique et interagissant par un mécanisme de communication donné. Le logiciel est généralement structuré en plusieurs couches où la première assure (directement au niveau hardware) le multiplexage du processeur entre les processus du système et le mécanisme de communication inter-processus.

Remarque importante : dans ce qui suit, nous nous restreindrons généralement à la définition la plus courte et la plus usuelle des systèmes répartis et ne considérerons que les systèmes répartis à longue ou courte distance .

3.1.2. Les avantages et la problématique des systèmes répartis

3.1.2.1. Les avantages

Les systèmes répartis présentent des nombreux avantages par rapport à un système centralisé sur un gros ordinateur.

a) **Performance :** Le travail simultané de plusieurs ordinateurs, chacun équipé de sa propre mémoire et se situant à proximité des données à traiter, permet de meilleures performances à un coût moins élevé.

b) **Extensibilité :** Il est très facile d'étendre le système, si la demande de service augmente, sans remplacer les composants existants.

c) **Accessibilité:** La panne d'un gros ordinateur arrêtera complètement le système, tandis que la défaillance d'un des composants locaux n'empêchera pas une grande partie du fonctionnement du système. Disposer de plusieurs ordinateurs est une garantie en cas de défaut de l'un d'eux.

d) **Conception :** Ce mode de fonctionnement facilite souvent la conception et l'installation des systèmes informatiques grâce à la décomposition du système global en sous-systèmes, chacun d'eux implémenté sur un ordinateur.

Mais à l'inverse, les problèmes qu'il faut résoudre pour mettre en œuvre un système réparti sont loin d'être triviaux.

3.1.2.2. Les problèmes

Dans un ordinateur classique, le processeur trouve dans l'unique mémoire centrale l'ensemble des informations dont il a besoin pour assurer le bon fonctionnement des tâches qu'il doit effectuer. Il réalise son programme en mode séquentiel en exécutant les instructions les unes après les autres. A l'inverse, les ordinateurs organisés en réseau d'un système réparti doivent agir ensemble pour la réalisation d'un objectif commun tout en conservant le maximum d'autonomie possible. Tout se résume donc à un problème de prise de décision : chaque ordinateur doit, à chaque instant, prendre la décision lui permettant de réaliser la partie du travail qui lui est assignée. Il faut donc assurer la synchronisation des différents ordinateurs coopérant au sein du réseau, compte tenu des délais de transmission des messages, tout en garantissant fiabilité et sécurité.

a) **la synchronisation :** Les ordinateurs d'un réseau de système réparti travaillent d'une manière parallèle et en général asynchrone, c'est-à-dire à des vitesses indépendantes et non égales. Si un composant du système requiert de l'information non momentanément accessible, qui doit être fournie par un autre composant, alors il doit se mettre en attente. La vitesse du système global est donc déterminée par la vitesse du composant le plus lent. Pour un tel système asynchrone, le mécanisme de communication doit fournir une facilité de "routage" ou de "contrôle de flux" pour adapter les vitesses effectives des différents composants. Quelle que soit la complexité du système réparti, synchroniser les programmes parallèles sur plusieurs ordinateurs n'est pas un problème facile à résoudre.

L'exemple le plus significatif et le plus important à signaler est celui du partage de ressources entre plusieurs ordinateurs du système. La plupart des ressources imposent des restrictions (par exemple l'exclusion mutuelle) si elles peuvent être accédées par

les différents composants du système. Ces ressources doivent être attribuées d'une manière équitable et de façon qu'il n'y ait pas d'attente infinie ou de blocage. Pour éviter ces inconvénients, on doit construire un algorithme équitable qui garantira le fonctionnement correct. Dans un système centralisé comportant un seul ordinateur, l'ordre d'entrée des demandes est généralement assuré par une mise en file d'attente unique des demandes. Dans un système réparti, il faut déterminer un ordonnancement des demandes, autrement dit un ordre global valable pour tous les composants du système. Cet ordonnancement peut être assuré de trois manières différentes :

•1. allocation centralisée : Un composant, le "scheduler" est le responsable unique pour l'allocation de toutes les ressources du système. Cette approche est la plus simple à réaliser mais elle n'est pas fiable car elle ne résistera pas à la disparition, en cas de panne inopinée, de ce contrôleur privilégié et le système entier se trouvera bloqué.

•2. allocation à la ressource : Un module d'allocation est associé à chaque ressource partageable. Il traite les demandes d'accès à la ressource venant de tous les composants du système.

Du point de vue d'une ressource, les deux approches sont relativement similaires. Tout de même, l'évitement de blocage semble être plus difficilement réalisable avec cette dernière approche. Pour s'en rendre compte, il suffit de s'imaginer deux composants c1 et c2 utilisant tous les deux les ressources r1 et r2 simultanément pour l'accomplissement de leur tâche. Si le module d'allocation associé à la ressource r1 alloue cette dernière à c1 et si par contre le module d'allocation associé à la ressource r2 alloue celle-ci à c2, alors on arrive à une situation où aucun des deux composants ne peut travailler.

•3. algorithme distribué d'allocation de ressources : Ce sont essentiellement E.W. Dijkstra et L. Lamport qui ont contribué, dans les années 1970, à développer des techniques distribuées d'allocation des ressources. Tous les processus en concurrence pour une ressource exécutent cet algorithme distribué qui

détermine l'allocation. Une des techniques les plus connues est celle développée par L. Lamport, la technique d'ordonnancement par une horloge logique.

b) la communication : Qui dit système distribué dit système communicant. Or tous les grands constructeurs de système de traitement de l'information ont développé leur propre architecture de réseau, définissant moyens et protocoles permettant à leurs ordinateurs de communiquer entre eux, mais pas nécessairement avec les ordinateurs construits par un autre constructeur. Chaque constructeur s'est attaché à développer une solution au problème de la communication. Dès la fin des années 1970, nombreux sont les ordinateurs qui pouvaient dialoguer entre eux et échanger des données entre eux. Mais ces conversations restaient souvent trop "familiales", c'est-à-dire les machines IBM comprenaient les machines IBM, pas les autres, d'une part à cause des différences entre les systèmes d'exploitation, d'autre part parce que chaque constructeur choisissait une méthode de transmission des données, un protocole d'échange qui n'était pas forcément celui qu'avait retenu le concurrent. Ce problème pouvait parfois être résolu en construisant des "passerelles" des interfaces jouant le rôle de traducteur entre les systèmes hétérogènes. Une grande avance et percée ne fut accomplie qu'avec la publication du modèle de référence de l'ISO à la fin des années 1970.

3.2. LE MODELE DE REFERENCE DE L'ISO

Dans le contexte d'un foisonnement croissant des méthodes de communication, il est très vite apparu qu'une certaine réglementation était devenue nécessaire et inévitable. Les organismes de normalisation, comme l'ISO et le CCITT, se sont penchés très tôt sur ce problème fondamental des communications entre matériaux hétérogènes.

La solution trouvée peut se résumer dans la phrase : "Séparons les problèmes et traitons-les un par un". On proposa de séparer les spécifications concernant le plus bas niveau des communications (niveau physique) des protocoles de communication et des logiciels nécessaires à la présentation et à la compréhension des données.

L'objectif fut une structure en couches qui communiquent entre elles mais qui sont néanmoins indépendantes. En "bas", le support, le lien physique sur lequel transitent les informations; juste "au-dessus", la procédure de communication, les opérations d'adressage, bref tout ce qui régit le transport même de l'information. Au "milieu", toutes les procédures réglant l'établissement correct et efficace d'un dialogue. En "haut", les éléments nécessaires à la présentation de l'information et au traitement même que subira cette information. Ainsi, l'ISO proposa vers la fin des années 1970 un "modèle" théorique en sept couches définissant les aspects tant matériels que logiciels nécessaires à l'interconnexion de divers équipements informatiques.

Aujourd'hui, ce modèle connu sous le nom Open System Interconnexion (OSI), mis au point par l'ISO, est respecté par la plupart des constructeurs informatiques. Des interfaces, quoique bien plus simples à développer et à mettre en oeuvre, seront tout de même toujours nécessaires pour relier les réseaux des différents constructeurs entre eux. Car le modèle OSI n'est pas une structure excessivement figée; il laisse la possibilité aux constructeurs de bâtir leurs architectures sur des fondations de différentes natures. Ainsi, si on prend un exemple dans le domaine des réseaux locaux, un constructeur peut aussi bien opter pour un réseau dit

Ethernet que pour un réseau à passage de jeton, tout en restant fidèle et conforme au modèle OSI.

3.2.1. Les concepts de base du modèle de référence de l'ISO

3.2.1.1. Introduction

En 1977, l'International Standards Organization (ISO) reconnaissait le besoin urgent de normalisation pour des réseaux informatiques hétérogènes construits par des constructeurs différents et décidait de créer un nouveau sous-comité pour la connexion de systèmes ouverts. Le terme ouvert fut choisi pour souligner le fait qu'en respectant ces normes internationales, un système sera ouvert à tous les autres systèmes du monde respectant ces mêmes normes.

On arriva très rapidement à un consensus sur une architecture en couches qui devrait satisfaire la plupart des exigences de l'interconnexion de systèmes ouverts avec la capacité de pouvoir être étendue plus tard pour rencontrer des besoins futurs.

Ainsi fut publié en fin 1979 le modèle de référence de l'ISO pour l'interconnexion de système ouverts, le OSI (= Open System Interconnection) Architecture Model [ZIMMERMANN80], qui est actuellement un modèle accepté dans le monde entier. L'architecture de ce modèle est représentée à la figure 3.4..

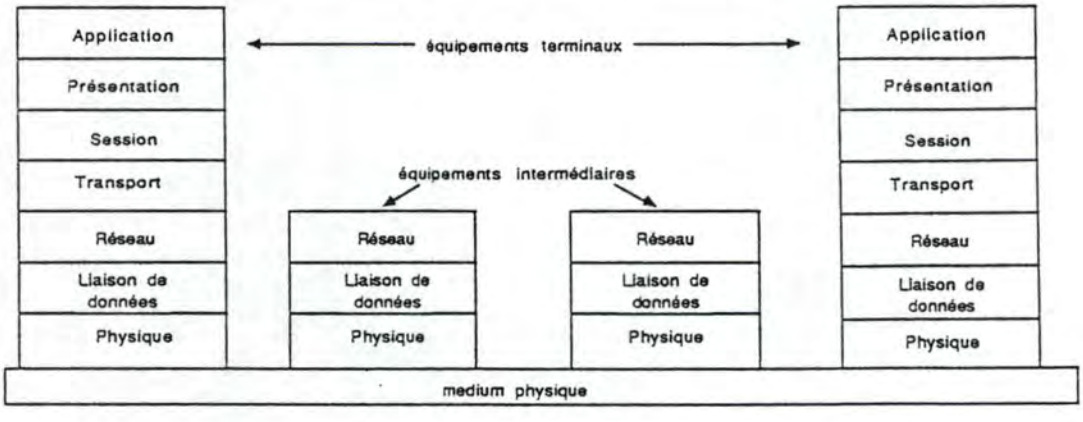


figure 3.4. Le modèle de référence OSI de l'ISO

3.2.1.2. Les principes généraux d'une structuration en couches

Les systèmes distribués sont généralement très complexes. A côté de la partie du système qui gère l'application proprement dite, une large partie du système doit être consacrée à la gestion de la communication entre les différents composants du système distribué (échange de données sur grande distance, contrôle de la synchronisation et de la cohérence des opérations exécutées à des endroits éloignés).

A cause de cette grande complexité, la conception de système distribué est généralement structurée en un certain nombre de couches hiérarchiques. Chaque couche fournit un certain service spécifique, qui est utilisé par la couche supérieure adjacente. En utilisant ces services, elle ignore les détails de leur implémentation dans les couches inférieures.

Une structure en couches est une technique de structuration qui permet à un réseau de systèmes ouverts d'être logiquement vu comme une succession de couches, chacune incluant les couches plus basses et l'isolant de couches plus hautes, comme illustré sur la figure 3.5. .

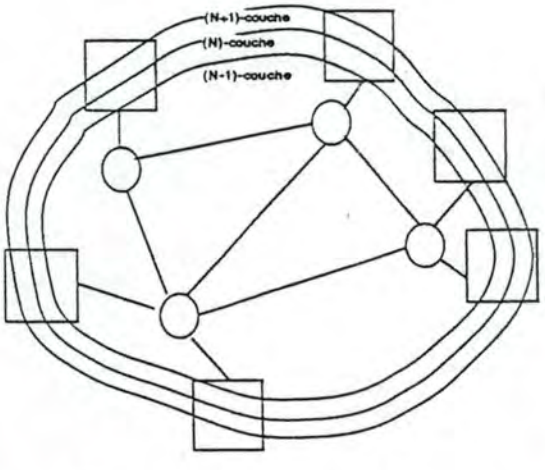


figure 3.5. Structuration en couches

Chaque système pris individuellement est considéré comme étant logiquement une succession de sous-systèmes, chacun correspondant à l'intersection du système avec la couche. En d'autres mots, une couche peut être vue comme logiquement composée de sous-systèmes du même rang de tous les systèmes interconnectés. Chaque sous-système est, par contre, considéré comme constitué d'une ou de plusieurs entités. En d'autres mots, chaque couche est faite d'entités, chaque entité appartenant à un système. Les entités dans la même couche sont appelées entités paires.

Pour des raisons de simplicité, chaque couche est référencée comme (N)-couche tandis que ses couches inférieure et supérieure sont référencées respectivement comme (N-1)-couche et (N+1)-couche. La même notation est utilisée pour désigner tous les concepts relatifs à des couches, par exemple les entités de la (N)-couche sont nommées (N)-entités. L'idée de base de la structuration en couches est que chaque couche ajoute de la valeur à des services procurés par l'ensemble des couches inférieures d'une telle façon que la couche supérieure offre les services nécessaires à faire tourner des applications réparties. Le principe de découpage en couches divise donc le problème global en morceaux plus petits.

Un autre principe de base est d'assurer l'indépendance de chaque couche en définissant des services à la couche supérieure, indépendamment de la manière dont ces services sont réalisés. Ceci permet des changements dans la manière qu'une couche ou un ensemble opère, à condition qu'ils offrent le même service à la couche suivante. Cette technique est similaire à celle utilisée dans la programmation structurée où uniquement les fonctions réalisées par un module (et pas le fonctionnement interne) sont connues de l'utilisateur.

Excepté la couche la plus haute qui opère pour son "propre compte", les (N)-entités distribuées parmi les systèmes ouverts interconnectés travaillent collectivement à procurer le (N)-service aux (N+1)-entités comme illustré par la figure 3.6.. En d'autres

mots, les (N)-entités ajoutent de la valeur au (N-1)-service qu'elles reçoivent de la (N-1)-couche et offrent ce service avec une valeur ajoutée, c'est-à-dire le (N)-service, aux (N+1)-entités.

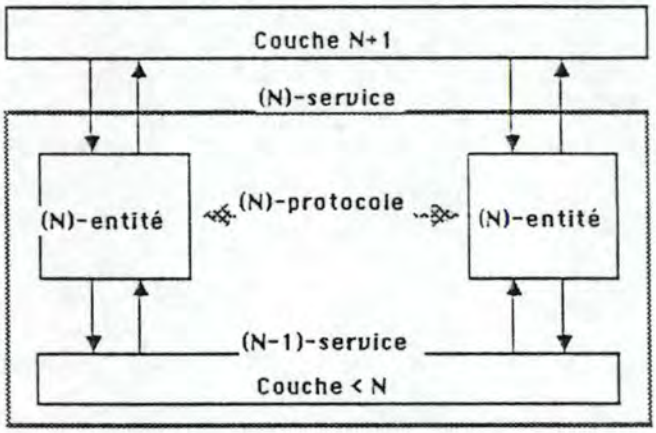


figure 3.6. Les concepts de Service, Protocole et Entité

La communication entre les (N+1)-entités utilise exclusivement des (N)-services. Les entités dans la couche la plus basse communiquent par le médium physique de l'interconnexion de systèmes ouverts, qui peut être considéré comme la (0)-couche de l'architecture OSI.

La coopération parmi (N)-entités est réglée par les (N)-protocoles qui précisent comment (N)-entités travaillent ensemble, utilisant le (N-1)-service pour procurer les (N)-fonctions pour ajouter de la valeur au (N-1)-service afin de fournir le (N)-service aux (N+1)-entités.

Les (N)-services sont offerts aux (N+1)-entités aux (N)-points-d'accès-de-service (ou (N)SAP's pour (N)-service-access-points) qui représentent l'interface logique entre les (N)-entités et les (N+1)-entités. Un (N)-SAP peut être servi par uniquement une (N)-entité et être utilisé par uniquement une (N+1)-entité, mais une (N)-entité peut servir plusieurs (N)-SAP's et une (N+1)-entité peut utiliser plusieurs (N)-SAP's.

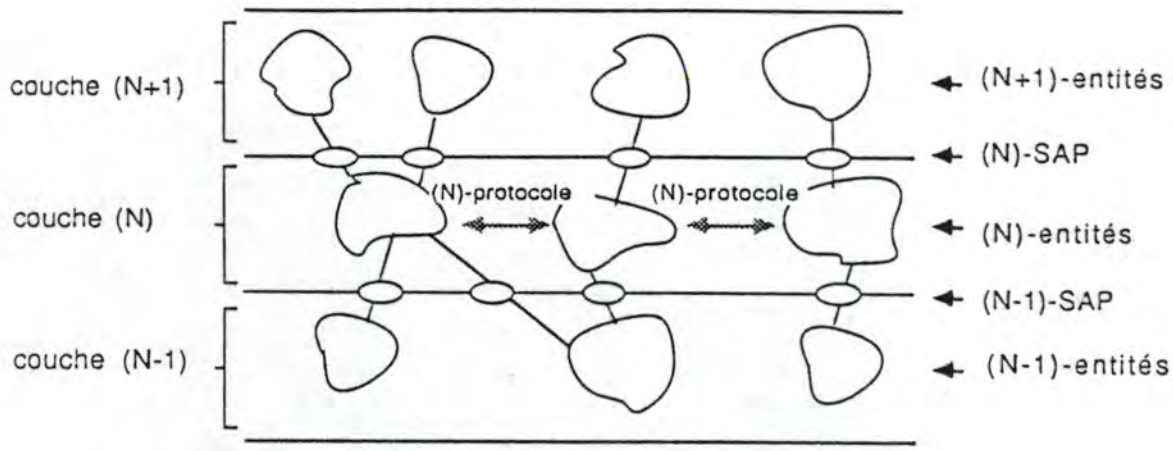


figure 3.7. Entités, points d'accès de services et protocoles

3.2.1.3. Les sept couches de l'architecture OSI

Après avoir défini les concepts de base, le modèle OSI définit le rôle de chacune des couches, résumé ci-dessous [MACCHI87] :

a) Couche Physique : La Couche Physique joue un double rôle. Elle est tout d'abord chargée de l'interface entre les systèmes et le support physique pour l'interconnexion de systèmes ouverts. La Couche Physique est également chargée d'assurer le relais des éléments binaires transmis, c'est-à-dire qu'elle réalise la fonction d'interconnexion entre les circuits de données.

b) Couche Liaison de Données : La fonction de base de la Couche Liaison de Données est de gérer les trames ainsi que d'effectuer, le cas échéant, la détection et la reprise des erreurs entre systèmes ouverts adjacents.

c) Couche Réseau : La fonction essentielle de la couche Réseau consiste à effectuer le relais des paquets ainsi que le routage des paquets et des circuits de données. De plus, la couche peut effectuer le multiplexage, le contrôle d'erreurs et le contrôle de flux lorsque cela permet d'optimiser l'utilisation des ressources de transmission.

d) **Couche Transport :** Les fonctions essentielles de la Couche Transport sont d'effectuer le contrôle de bout en bout et l'optimisation de bout en bout du transport des données entre systèmes extrémités. La Couche Transport opère toujours de bout en bout. Toutes les fonctions relatives au transport de données entre systèmes sont réalisées au sein de la Couche Transport et des couches inférieures.

e) **Couche Session :** La Couche Session réalise les fonctions qui sont nécessaires au support du dialogue entre processus telles l'initialisation, la synchronisation et la terminaison du dialogue.

f) **Couche Présentation :** La Couche Présentation prend en charge les problèmes associés à la représentation des informations que les applications désirent échanger ou manipuler. En d'autres termes, la Couche Présentation s'occupe de la syntaxe des données échangées permettant ainsi aux entités d'application de ne se préoccuper que des aspects sémantiques des informations.

g) **Couche Application :** La Couche Application prend en charge toutes les fonctions nécessaires aux applications réparties et qui ne sont pas déjà fournies par le Service de Présentation.

3.3. LA NECESSITE D'UNE METHODOLOGIE POUR LA CONCEPTION DE SYSTEMES DISTRIBUES

3.3.1. Introduction

L'importance actuelle des réseaux locaux et des ordinateurs nécessite la conception de programmes sophistiqués afin de concevoir et d'utiliser les applications en question. Dans le cas général, on aura une communication entre un nombre n d'activités de processeurs séquentiels dont la conception des interactions entre les ordinateurs doit tenir compte d'actions se passant à des endroits différents, peut-être même dans des pays différents. Chacun des logiciels des processus du réseau pouvant lui-même être complexe, la difficulté du logiciel global ne peut alors être perçue dans sa totalité, car elle découlera de la nécessité de manipuler à la fois l'ensemble de la modularité (les modules, un par processus) et l'ensemble de la distribution (les interactions entre ces modules).

Par conséquent, la conception de programmes interactifs du système global semble être une tâche très difficile et une méthodologie semble être nécessaire. Dû à leur complexité et importance, c'est dans le domaine des réseaux d'ordinateurs qu'on a mené les premières études sur la spécification formelle, la validation et l'implémentation de logiciels [SUNSHINE80] [BOCHMANN79] avant la reconnaissance générale de leur intérêt.

Une étude américaine, dont on citera quelques chiffres clés, a éclairé de quelle importance primordiale peut être la détection des erreurs de spécification. Selon cette enquête [BOEHRM 75] effectuée sur un nombre représentatif de gros projets logiciels, les erreurs de spécification représentent (avec les erreurs de conception) 66 pour cent du nombre total des erreurs contenues dans les logiciels étudiés. Ces erreurs, vu qu'elles ont des répercussions sur tout le cycle de vie d'un logiciel, sont aussi les plus coûteuses à corriger. Le coût de correction des erreurs de spécification représente en moyenne 66 pour cent du coût total de correction, contre 25 pour cent pour les erreurs de conception et 9

pour cent pour les erreurs de codage. Au plus tard ces erreurs sont détectées, au plus le coût est élevé. Ainsi, une erreur de spécification repérée lors de la conception coûte 2,5 fois plus, lors du codage 5 fois plus et lors de l'intégration même 36 fois plus qu'une erreur trouvée lors de la phase de spécification même. L'intérêt de la validation préalable de la spécification devient donc évident si on sait en plus que la maintenance forme parfois plus que 60 pour cent du coût total d'un logiciel.

Il convient de préciser la différence fondamentale entre la spécification et l'implémentation de cette spécification dans un programme. La spécification doit décrire et définir uniquement la fonctionnalité d'un système, en d'autres mots la relation des entrées et des sorties. Le programme, de l'autre côté, est une description explicite et détaillée d'un processus opérationnel qui réalise effectivement les relations entrée-sortie. Une spécification doit établir toutes les exigences auxquelles un objet doit satisfaire et rien de plus. Pour être abstrait, elle doit séparer l'essentiel de l'accessoire, en couvrant le premier et en omettant le second.

3.3.2. Les étapes de conception d'un système distribué

De la section 2, il apparaît clairement que des systèmes distribués sont constitués d'un ensemble de services et de protocoles. Comme ils sont conçus par couches, le concepteur est confronté à une hiérarchie de couches. Généralement, on adopte la méthodologie suivante, en quatre étapes, pour la conception d'un système distribué.

3.3.2.1. Spécification

Une première étape est constituée par la *spécification* du système. Cette étape de spécification qui est le point de départ de la conception peut, dans le cas de programmes de communication, être sous-divisée en plusieurs sous-étapes :

• définition d'une architecture hiérarchique en fixant l'ensemble des couches du système distribué. Ceci est une tâche difficile et les choix ne sont pas toujours évidents.
• définition des services offerts ((N)-service) par les couches fonctionnelles du protocole. Cette spécification précise donc la fonction à réaliser.
• définition, pour chaque couche, des entités de protocole qui communiquent entre elles, donc du protocole. Cette spécification exprime ce que réalise la couche.
• spécification pour la (N)-couche, du service ((N-1)-service) fourni par la couche inférieure, utilisé par la couche considérée, et qui est nécessaire pour assurer que les entités réalisent bien le service de référence ((N)-service) à fournir. Cette définition donne donc les fonctions de base déjà existantes sur lesquelles la couche est construite.

3.3.2.2. Modélisation et validation

Si des erreurs sont commises à la première étape et non détectées à l'étape suivante de la *modélisation et validation* de la spécification, alors les conséquences sont, en général , très graves et génèrent des coûts de correction qui peuvent atteindre des sommes importantes, voire astronomiques. Avant de pouvoir valider de façon aussi complète que possible une (N)-couche, il est nécessaire de dériver un modèle formel pour les (N-1)-services, pour les (N)-protocoles, pour les (N)-entités et pour le (N)-service (cfr. figure 3.6.).

Il est généralement assez difficile de comprendre intégralement le comportement global d'un protocole complexe. En plus, modéliser une telle complexité n'est pas facile. Ceci explique pourquoi, souvent, on ne modélise qu'une partie spécifique d'un protocole. Il faut aussi noter que le protocole inclut des aspects nombreux tel que le temps, le contrôle et les données qu'il est difficile de considérer toutes à la fois. Ainsi, par exemple, le temps est très rarement considéré explicitement.

En ce qui concerne la validation, on distingue deux types de validation. D'une part, la validation générale, qui est d'intérêt quel que soit le modèle, vérifie un certain nombre de "bonnes propriétés" comme l'absence de blocage, le comportement cyclique, le caractère borné,... . D'autre part, la validation spécifique dépendant du comportement spécifique du protocole tente de prouver que la spécification fournit bien le service demandé. L'utilisation d'invariants est préconisée pour définir les propriétés jugées nécessaires.

3.3.2.3. Implémentation

La troisième étape est constituée par l'*implémentation* du protocole. Cette implémentation doit être aussi immédiate que possible. Si une implémentation directe n'est pas possible, il faut passer encore à la validation de l'implémentation.

3.3.2.4. Tests

Dans la quatrième et dernière étape, on procède aux *tests* du système. Les tests et la vérification "on-line" sont, vu la distribution géographique, d'une grande importance et une approche formelle et méthodique, d'un grand intérêt.

3.3.3. Méthodes formelles pour la description de système

L'architecture globale de la distribution, composée d'interactions, peut être décrite d'une manière informelle en utilisant un pseudo-langage ou la langue naturelle. Mais il devient assez difficile à exprimer des mécanismes sophistiqués de coopération. Cela conduit à des ambiguïtés et des spécifications incomplètes qui contiennent potentiellement des erreurs de spécifications. Cette possibilité est clairement inacceptable pour des systèmes ouverts pour qui des implémentations très différentes - réalisées par des constructeurs différents - doivent aboutir à un comportement totalement compatible, quel que soit le domaine d'application.

Par conséquent, des techniques de description formelle doivent être utilisées, car il est obligatoire dans des systèmes ouverts d'avoir

• une description non ambiguë, car des implémentations séparées seront dérivées d'elle et ces implémentations doivent être totalement compatibles. Par conséquent, le modèle formel utilisé doit reposer sur une fondation mathématique ferme pour permettre l'application de techniques d'analyse rigoureuses.

• une spécification de haut niveau sans contenir des contraintes d'implémentation pour qu'elle soit implémentée si efficacement que possible sur les différents systèmes cibles par les concepteurs. Le modèle formel doit donc être totalement indépendant par rapport à l'implémentation.

• une validation off-line, avant l'implémentation, car la correction d'erreurs est très difficile dans des systèmes distribués si tous les processeurs tournent en parallèle et envoient une série de messages erronés. Pour pouvoir gérer et maîtriser la complexité, le modèle doit également être capable d'exprimer des idées de spécifications à de nombreux niveaux de détails (d'abstraction) afin de pouvoir procéder par raffinements successifs.

La base fondamentale d'une telle méthodologie consiste à spécifier de façon formelle le fonctionnement distribué, en incluant les comportements séquentiels et les interactions entre les composants du système global. La description formelle pourra alors servir de base à l'ensemble des travaux et équipes participant à la conception.

En effet, une description formelle est indispensable si l'on veut, premièrement, exprimer de façon non ambiguë et complète les fonctions nécessaires et, deuxièmement, analyser les spécifications et donc détecter, très tôt, des erreurs. L'importance de la spécification, déjà existante dans le cas des systèmes séquentiels, est encore accrue car, dans le cas général, des logiciels distribués sont réalisés sur des équipements informatiques différents, par des équipes différentes, utilisant des supports logiciels et matériels différents pour la réalisation. Une spécification formelle pourra servir de base commune et de

référence à l'ensemble des équipes concernées par la conception globale.

3.3.4. Des outils pour la conception de systèmes distribués

La description formelle permet en effet l'utilisation d'outils logiciels déduits du formalisme utilisé dans toutes les phases de conception. Ces logiciels devraient selon [RUDIN87] permettre la validation (détection d'erreurs syntaxiques ou de formes) et la vérification (démonstration de la fonction souhaitée) de protocoles. Plusieurs formalismes et outils permettent également de donner une indication sur les performances très tôt dans la phase de conception.

Le protocole implémenté doit être une instanciation fidèle du protocole spécifié, sinon tous les efforts de validation sur la spécification seraient vains. Ceci pourrait être garanti par une procédure de compilation qui, à partir de la spécification formelle du protocole, génère le code dans le langage de la machine cible. En absence d'une telle - invraisemblable - procédure, l'implémentation doit être testée ou certifiée par rapport à la spécification formelle. Ici, le principe de l'observateur s'est démontré fort utile. L'observateur compare le comportement de l'implémentation avec le comportement attendu décrit dans la spécification formelle [MONDAIN87].

Comme il existe beaucoup de standards de protocoles soit planifiés, soit déjà utilisés, il serait également intéressant d'avoir une procédure de conversion d'un système de protocole à un autre.

La figure 3.8. donne une illustration d'un environnement de conception de protocoles offrant des fonctionnalités de validation, de vérification, d'analyse de performances et de conversion de protocole à partir d'une description formelle faite avec l'aide du support de conception. Egalement inclus dans cet environnement, un générateur de séquences de test[1].

---

[1] La question des outils sera approfondie au chapitre 4.

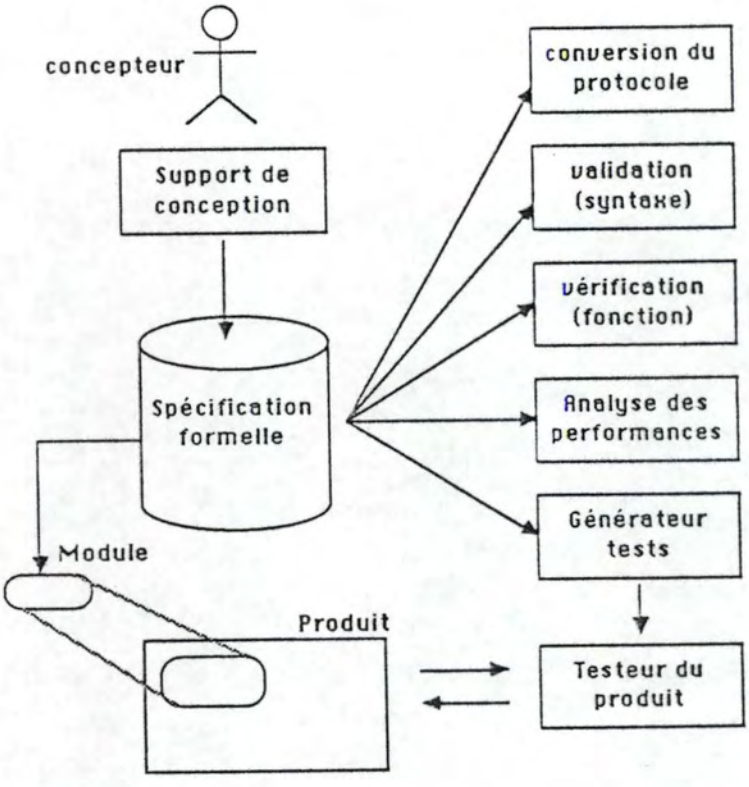


figure 3.8. Un environnement de conception de protocoles

3.4. LES RÉSEAUX DE PETRI DANS LA CONCEPTION DE PROTOCOLES

3.4.1. Introduction

Nous l'avons déjà affirmé à plusieurs reprises : une application répartie doit être décrite avec rigueur, car plus que dans le domaine de la programmation séquentielle, on cherchera à la simuler, à la tester, mieux, à être assuré de son bon fonctionnement avant de la mettre en oeuvre. Ceci implique que l'on puisse disposer d'un modèle formel permettant la description et l'analyse de l'application.

Les réseaux de Petri ont le mérite d'avoir été un des tout premiers modèles formels dont la vocation affirmée est de permettre d'exprimer et puis d'analyser des applications parallèles, en particulier d'étudier les problèmes de synchronisation et de concurrence [COURTIAT84].

3.4.2. La modélisation de protocoles par réseaux de Petri

La conception d'un protocole [DIAZ85] complexe est une tâche très dure et pleine de risque d'erreurs. Plus le protocole est complexe, plus nombreuses sont généralement les erreurs. C'est la raison pour laquelle des méthodes formelles sont nécessaires. Chaque approche formelle doit reposer sur une base bien reconnue par tout le monde. Dans le cas des protocoles, le support est le concept de "couche" dans l'architecture OSI. La conséquence est que chaque modèle doit tenir compte de cette organisation architecturale de protocoles. La figure 3.6. donne la structure globale et bien connue des trois couches dont on doit tenir compte en considérant une (N)-couche quelconque.

D'une telle approche, il apparaît que pour un protocole et une (N)-couche donnés, le modèle doit tenir compte des éléments suivants :

• le service fourni par la (N)-couche, i.e. les interfaces avec la couche supérieure N+1

• le service utilisé par la (N)-couche, i.e. le modèle du mécanisme fourni par la couche inférieure N-1
• le mécanisme spécifique de communication qui définit le protocole de la (N)-couche

L'obtention du fonctionnement global du (N)-protocole nécessite la modélisation des différents éléments de l'architecture considérée. Nous devons donc modéliser le comportement local de chaque entité, le (N-1)-service et les interfaces reliant ces éléments [AYACHE85].

3.4.2.1. Modèle local d'une entité de protocole

La modélisation du comportement d'une entité dans l'étude d'un protocole de communication ne nécessite pas de représenter tous les mécanismes mis en place dans l'entité (gestion des ressources, mémoires,...), ni de représenter toutes les communications existantes [1].

La modélisation pratique du comportement local des entités nécessite la considération des points suivants [NOVALI86]:

a) **La fragmentation des modèles :** Lors de l'étude de protocoles complexes, le protocole sera en général fragmenté pour être modélisé. La représentation, sur un même réseau de Petri, du comportement complet d'une entité, aboutirait à une explosion du nombre d'états.

Cette fragmentation doit séparer les parties les plus indépendantes, comme les différentes phases du protocole, par exemple : l'ouverture-fermeture de la connexion, le transfert de données sur la connexion établie.

b) **Les temporisations :** Pour surveiller les dialogues, les entités de protocole utilisent des temporisations. L'expiration

[1] Par exemple chaque (N)-entité de protocole, est en principe impliquée dans plusieurs (N)-connexions; ces communications sont indépendantes les unes des autres, mais elles peuvent être liées pour des partages de ressources. Ces problèmes n'étant pas modélisés, car nous nous intéressons particulièrement à l'échange des PDU's entre les entités; pour l'étude d'un (N)-protocole, nous ne représentons que le comportement local de deux entités qui coopèrent dans le cadre d'une seule connexion.

d'une temporisation provoque le déclenchement d'une procédure de récupération d'erreurs, qui consiste généralement à retransmettre l'information. Dans une modélisation de protocole, il est donc important de tenir compte des temporisations. L'utilisation des réseaux de Petri classiques ne permet pas d'introduire explicitement le paramètre temps mais nous pouvons simuler les causes qui provoquent l'expiration de la temporisation, permettant ainsi d'introduire les mécanismes de récupération d'erreur dans les modèles des entités.

Cette simulation s'effectue par exemple par la modélisation de la perte du message dans le médium de communication, comme nous le montrerons dans un exemple de la section suivante.

c) Les variables : Les variables doivent être représentées par des places supplémentaires. D'une part, les variables doivent être des variables entières, et d'autre part, l'ensemble des valeurs prises par ces variables doit être le plus limité possible

Afin d'avoir un modèle global de taille acceptable, il vaut mieux ne représenter que les variables importantes, celles dont la valeur influe sur la suite des échanges entre entités.

3.4.2.2. Modélisation du service utilisé ((N-1)-service)

La spécification du service fourni par la couche adjacente inférieure peut être plus ou moins complexe. Ceci est dû au fait que le service a la possibilité ou non d'agir sur les interactions entre entités homologues et d'engendrer lui-même des interactions.

Une première approche dans la modélisation du service consiste à spécifier seulement le transfert de données où chaque interaction est résultat simplement de l'usage d'une primitive d'envoi d'un message. Une telle modélisation du service sera dénommée par la suite "médium virtuel".

La seconde approche consiste en une modélisation complète d'un service orienté connexion, c'est-à-dire où la connexion est explicite et qui implique la prise en compte dans le modèle des interactions associées à l'ouverture et la fermeture de connexion.

Cependant la fermeture de connexion ne résulte pas nécessairement d'une interaction engendrée par l'entité homologue distante et ceci implique donc que le service puisse l'engendrer sur sa propre initiative . Il en sera de même pour le signalement d'erreurs.

a) Spécification par médium virtuel : Dans ce premier type d'approche, on ne modélise que le transfert de données. Chaque interaction est simplement le résultat d'un envoi d'un message. Différents comportements peuvent être associés à un médium virtuel : médium parfait, perte de message, duplication de message, ordonnancement Fifo, taille du canal bornée ou non bornée, etc.. Considérons à titre d'illustration le modèle simplifié d'une communication entre deux entités homologues, chacune d'elles modélisée par un réseau de Petri comme représenté sur la figure 3.9.a. La spécification du médium virtuel sera utilisée pour interconnecter des transitions étiquetées par l'envoi de message (!m) et des transitions étiquetées par la réception du message (?m). La transition étiquetée par le libellé Perte tient compte du fait que le médium peut perdre le message en transit. On verra au paragraphe suivant (3.4.2.3.) que plusieurs types d'interconnexions sont possibles pour modéliser un médium virtuel en fonction des propriétés projetées. Dans cet exemple, on a décidé de procéder par fusion de transitions pour obtenir le modèle global (figure 3.9.b.).

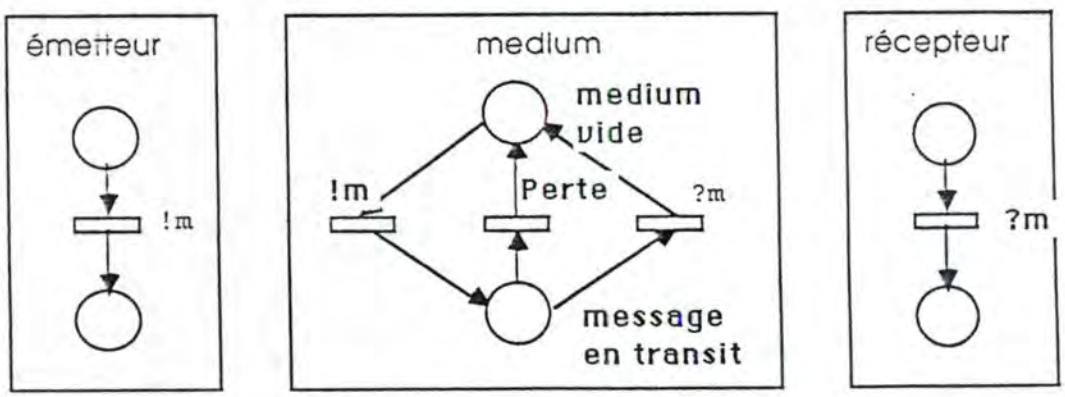


figure 3.9.a. Un exemple de modélisation du médium virtuel

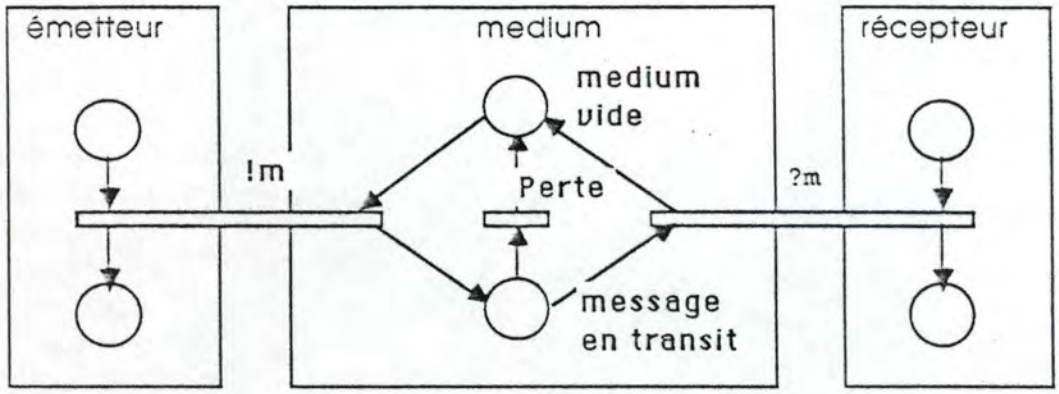


figure 3.9.b. Un exemple de modélisation du médium virtuel

b) Service orienté connexion : Cette approche consiste à représenter le service comme deux processus communiquant par un médium virtuel (figure 3.10.). Chaque processus est un modèle local du comportement du service à un bout de connexion (CEP), tel qu'il est perçu par la couche supérieure. Le modèle local de CEP peut être obtenu à partir des spécifications informelles de service. Le médium virtuel est chargé de propager les événements de service d'un modèle à l'autre. Cette propagation n'est pas forcément parfaite, si le (N-1)-service modélisé ne rend pas un service parfait.

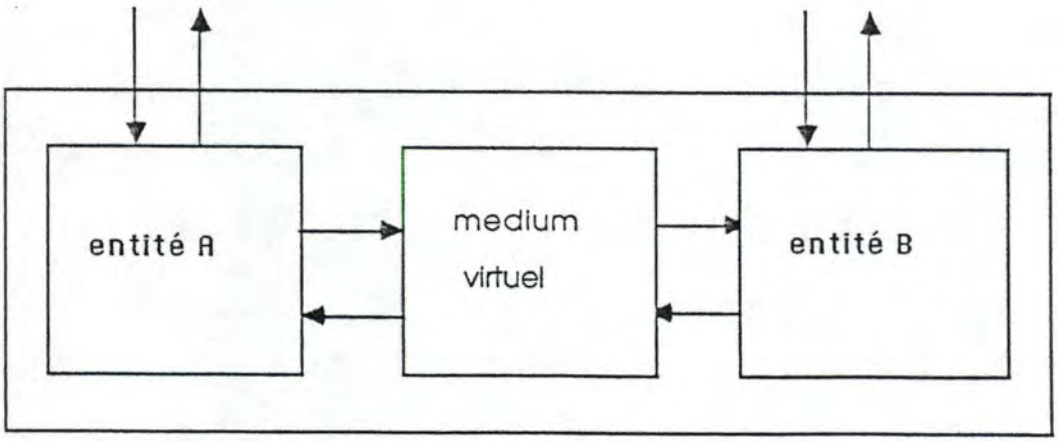


figure 3.10. Modèle de service orienté connexion

3.4.2.3. Modèle global de la couche du protocole

Le modèle global de la couche de protocole est obtenu en interconnectant chaque entité locale par le modèle du service fourni par la couche inférieure. Chaque entité aura été préalablement spécifiée par un réseau de Petri dont toutes les interactions seront relatives à des interactions externes avec la couche supérieure ou inférieure. Le modèle de service sera aussi modélisé de la même façon. Le but du processus d'interconnexion est, dans ce cas, d'obtenir un réseau de Petri dans lequel les étiquettes concerneront seulement des interactions avec la couche supérieure.

L'interconnexion peut être réalisée de plusieurs manières différentes. On distingue principalement les trois modes de synchronisation et communication [VALETTE88] suivants : l'échange synchrone ou par rendez-vous, l'échange asynchrone et l'appel avec confirmation.

a) **L'échange synchrone (ou par rendez-vous)** correspond au cas où l'entité émettrice tout comme l'entité destinatrice doivent suspendre leur exécution jusqu'à réalisation de l'échange. L'entité qui envoie le message ne le fera donc effectivement que lorsqu'elle aura la garantie que le destinataire est prêt à le recevoir.

figure 3.11. Echange par rendez-vous

Ce mécanisme correspond à l'introduction d'une transition commune entre les deux entités comme cela est figuré par la figure 3.11. La parfaite symétrie du réseau de Petri décrit bien le fait qu'il n'y a plus aucune différence fonctionnelle entre l'émetteur et le récepteur.

b) L'échange asynchrone correspond au cas où le processus envoyant le message ne se met pas en attente et poursuit donc son exécution. Par contre le destinataire sera mis en attente s'il essaie d'accéder au message avant que celui-ci soit arrivé.

Ce mécanisme est parfaitement décrit par la création d'une place commune entre les deux entités comme cela est représenté sur la figure 3.12. Franchir la transition a et mettre un jeton dans la place p1 correspond bien à une émission non bloquante. Franchir la transition b en consommant un jeton dans la place p2 correspond bien à une réception bloquante (en l'absence de jeton, on ne peut franchir la transition). Il suffit de confondre les places p1 et p2 en une seule place pe pour représenter correctement le mécanisme.

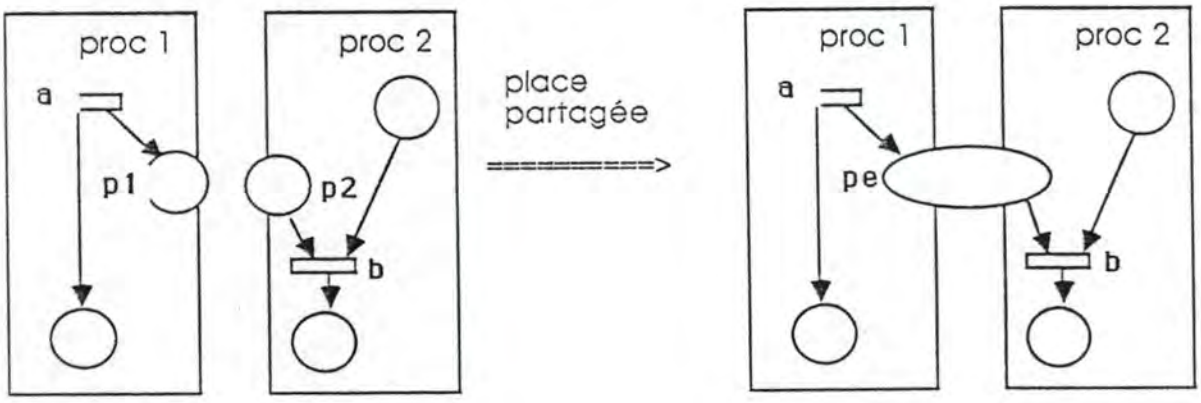


figure 3.12. Echange asynchrone

c) L'appel avec confirmation correspond au fait que l'émetteur va suspendre son exécution non seulement jusqu'à la réception de son message par le récepteur, mais qu'il va rester en attente jusqu'à ce que le récepteur ait envoyé une réponse.

Le réseau de Petri de la figure 3.13. décrit ce mécanisme.

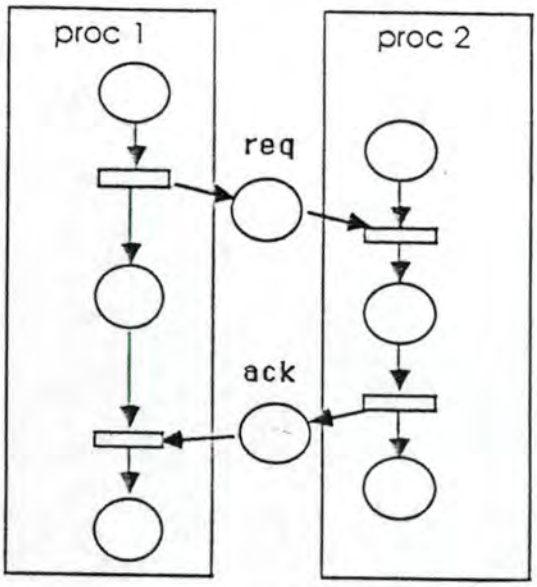


figure 3.13. Appel avec attente de la confirmation

Le point fondamental à souligner est la sémantique de l'interaction : le modèle d'interaction représente la vue abstraite de la couche inférieure, c'est-à-dire comment, vue de la (N)-couche, la (N-1)-couche agit. Le choix de cette sélection de la vue abstraite est d'une grande importance. Chacun des différents modes d'interactions peut mener à des résultats différents lors de l'analyse (réseau non borné ou borné, vivant ou non vivant selon le choix du modèle d'interaction).

3.4.3. La validation de protocoles modélisés par réseaux de Petri

Le but de ce paragraphe est de montrer comment il est possible d'utiliser les résultats classiques sur la validation des réseaux de Petri pour analyser les propriétés d'un protocole donné. L'analyse du protocole qui alors porte sur le modèle global, tel qu'il a été développé dans le paragraphe précédent, permet de prouver deux classes de propriétés, c'est-à-dire :

• d'une part des propriétés générales qui, a priori, doivent être nécessaires pour tout protocole. Ces propriétés fournissent des informations sur le protocole, son fonctionnement et sa structure, mais sont indépendantes de la mission confiée au système : elles

garantissent que le modèle défini a un comportement qui paraît cohérent,
• d'autre part des propriétés spécifiques qui dépendent du protocole considéré et donc de la sémantique associée aux places et aux transitions du réseau, c'est-à-dire de l'interprétation du modèle.

3.4.3.1. Validation de propriétés générales

La première caractéristique que l'on peut souhaiter est que le modèle global du protocole soit **borné**. En effet, si le modèle est non borné cela implique que le protocole a un nombre infini d'états et ne peut donc être implanté.

La deuxième propriété que l'on peut désirer est que le modèle global du protocole soit **vivant**. La vivance du modèle implique différentes propriétés du protocole à savoir :
• il n'y a pas de blocage; un protocole est dans l'état de blocage lorsqu'aucune transmission n'est possible dans l'état courant de chaque entité de protocole et qu'aucun message n'est en transit dans le médium virtuel (ou service de la couche inférieure) interconnectant les entités.
• il n'y a pas de réceptions non spécifiées. Supposons que le médium virtuel n'altère pas l'ordre des messages qu'il transmet; sous cette hypothèse classique, les réceptions non spécifiées peuvent être caractérisées par une situation de blocage, où cependant certains messages sont en transit dans le médium virtuel; en effet, si le message en tête de file ne peut pas être reçu lorsque l'entité réceptrice est dans un certain état, les autres messages en transit dans le médium virtuel ne peuvent doubler le premier, ce qui conduit à une situation de blocage.
• il n'y a pas d'interactions non exécutables; c'est le cas dans un réseau sans blocage où cependant certaines transitions ne sont jamais sensibilisées.

La troisième propriété que l'on peut vouloir montrer est que le modèle global du protocole soit **réinitialisable**, ce qui correspond bien au fonctionnement souvent répétitif et cyclique d'un protocole.

3.4.3.2. Validation de propriétés spécifiques

Une propriété spécifique est une propriété qui dépend de l'interprétation c'est-à-dire de la sémantique que le concepteur du protocole associe aux différents éléments du modèle global de ce protocole, c'est-à-dire les transitions, les places et les événements qui peuvent être en étiquette de certaines transitions.

Dans le domaine des protocoles, il est courant de distinguer :
• les propriétés liées à la correction partielle du protocole ("safety properties") qui permettent de montrer que, si le protocole progresse, alors il rend le service attendu,
• les propriétés liées à la progression du protocole ("progress properties") qui démontrent que le protocole progresse effectivement.

a) **Correction partielle :** Afin d'être en mesure de prouver la correction partielle d'un protocole, il faut disposer d'une spécification du service que le protocole est censé rendre. La spécification du service attendu étant sous la forme d'un modèle explicite du service, on devra vérifier l'adéquation entre ce modèle de service de référence et le modèle de service réellement fourni par le protocole conçu et modélisé. Dans ce cas, on procède par projection, car le service fourni par un protocole n'est en fait, comme nous l'avons vu, rien d'autre qu'une vue abstraite du modèle global du protocole par rapport à un ensemble d'événements visibles que sont les réceptions (respectivement émissions) des requêtes (respectivement indications) de service du (respectivement vers le) niveau supérieur. Ce modèle du service obtenu par projection doit alors être comparé avec le modèle du service attendu.

Si le service à fournir est défini sous la forme d'un ensemble d'assertions sur les marquages qui devront être vérifiés pour l'ensemble des états du protocole, alors une première approche consiste à effectuer directement la vérification de l'assertion pour chacun des états du graphe des marquages accessibles du réseau de Petri global. Une deuxième approche, plus intelligente, procède par

l'interprétation des invariants de place du réseau; elle est plus intéressante car elle ne nécessite pas l'énumération de l'ensemble des états du protocole et peut ainsi s'appliquer même à un réseau non borné.

b) Progression du protocole : La progression du protocole peut être prouvée en utilisant les invariants de transition du réseau de Petri spécifiant le modèle global du protocole. Les propriétés typiques à prouver sont celles d'accessibilité par rapport à un certain événement E.

3.4.4. Un exemple : le protocole du bit alterné

Afin d'illustrer les points précédents, on présentera l'exemple le plus souvent cité dans la littérature [AYACHE85], celui du protocole du bit alterné, premièrement modélisé par G. v. Bochmann [BOCHMANN79].

Le protocole du bit alterné met en œuvre un transfert de données très simple. Une entité émettrice, à la suite de la réception d'une requête de transfert de données du niveau supérieur (↓ DT-Req (Mes)) , procédera au transfert de la donnée Mes à laquelle elle adjoindra un bit de contrôle 0 ou 1. [1] L'entité réceptrice quant à elle acquittera systématiquement Mes0 par un message Ack0 et Mes1 par Ack1.

Le bit de contrôle permettra d'éviter les duplications (plusieurs messages se suivant et ayant le même bit de contrôle). Dans ce cas, uniquement un seul message fera l'objet d'une indication (î DT-Ind(Mes)) vers l'entité distante.

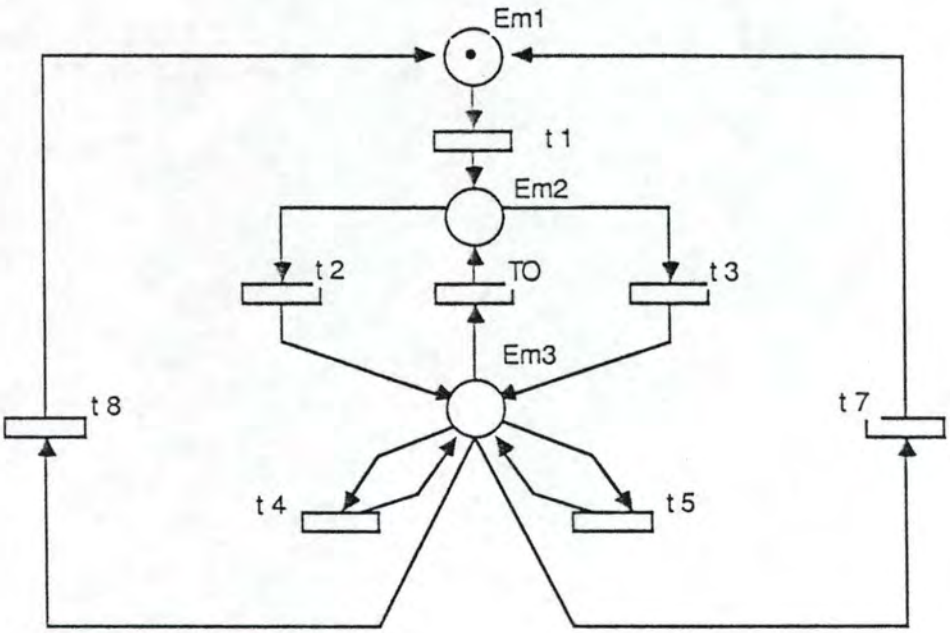
3.4.4.1. La modélisation

La modélisation de ce protocole sera effectuée par réseaux prédicats/actions, une extension des réseaux de Petri de base.

---

[1] Dans la suite, afin de simplifier les notations, nous assimilerons les requêtes et indications du transfert de donnée aux données elles-mêmes

Commençons d'abord avec la modélisation de l'émetteur telle que représentée sur la figure 3.14. Pour des raisons de lisibilité, nous avons séparé le réseau de prédicat/action en deux parties : le réseau prédicat/action d'une part et les prédicats et actions sous forme de tableau d'autre part.



transition	prédicat	action
t 1	↓? DT-Req (Mes)	-
t 2	E = 1	! Mes1
t 3	E = 0	! Mes0
t 4	? Ack0 et E = 1	-
t 5	? Ack1 et E = 0	-
t 6	? Ack1 et E = 1	E = modulo(E+1)
t 7	? Ack0 et E = 0	E = modulo(E+1)
TO	-	-

figure 3.14. Modèle de l'émetteur

Etant donné la réception d'une requête d'émission du message Mes (↓ DT-Req(Mes)) par la couche supérieure, on émet sur le médium selon la valeur du switch E soit Mes0 soit Mes1 (tir des transitions

libellées Mes0 et Mes1). Après cette émission, cinq cas peuvent survenir :

• le médium transmet une confirmation Ack1 à l'émetteur et E = 1 : on retourne à l'état initial après avoir mis E à 0,

• le médium transmet une confirmation Ack0 à l'émetteur et E = 0 : on retourne à l'état initial après avoir mis E à 1,[1]

• le médium transmet une confirmation Ack1 à l'émetteur tandis que E = 1 : on ne réagit pas (tir d'une transition de boucle),

• le médium transmet une confirmation Ack1 à l'émetteur tandis que E = 1 : on ne réagit pas (tir d'une transition de boucle), [2]

• la transition libellé TO tire : cette transition n'est pas contrainte par la réception d'un message ou par le fait qu'un prédicat soit vrai; elle correspond au déclenchement d'une alarme de temporisation (armée à l'émission du message Mesi). Le franchissement de cette transition conduit à la réémission de ce message.

De la même manière, nous pourrons modéliser le récepteur (figure 3.15.). A nouveau, plusieurs situations peuvent se présenter :

• le médium transmet le message Mes0 au récepteur et le switch R = 0 : on passe le message à la couche supérieure (î DT-Ind (Mes)), on émet un Ack0 sur le médium et un met R à 1,

• le médium transmet le message Mes1 au récepteur et le switch R = 1 : on passe le message à la couche supérieure (î DT-Ind (Mes)), on émet un Ack1 sur le médium et un met R à 0,

• le médium transmet le message Mes0 au récepteur tandis que R = 1 : on ne le passe pas à la couche supérieure et émet un Ack0 sur le médium pour provoquer une réémission,

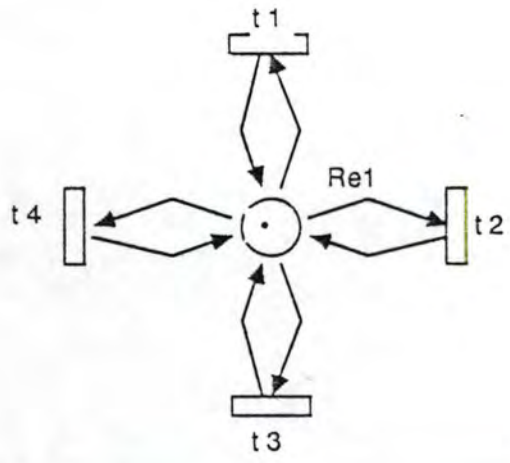
• le médium transmet le message Mes1 au récepteur tandis que R = 0 : on ne le passe pas à la couche supérieure et émet un Ack1 sur le médium pour provoquer une réémission [3].

---

[1] Ces deux premières situations correspondent donc aux cas normaux où aucune anormalité ou problème n'est survenue.

[2] Ces deux situations correspondent au cas où un problème est survenu (duplication ou perte d'un message ou d'une indication)

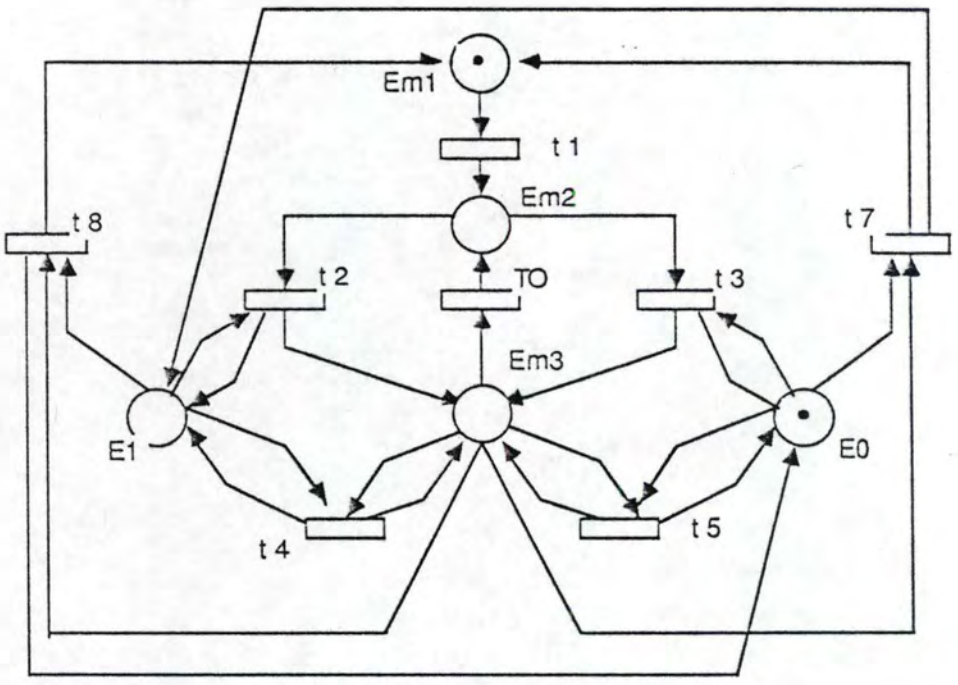
[3] Initialement E et R valent 0.



transition	prédicat	action
t 1	? Mes0 et R = O	R = 1; ! Ack0; ! DT-Ind (Mes)
t 2	? Mes1 et R = O	! Ack1
t 3	? Mes1 et R = 1	R = 0; ! Ack1; ! DT-Ind (Mes)
t 4	? Mes0 et R = 1	! Ack0

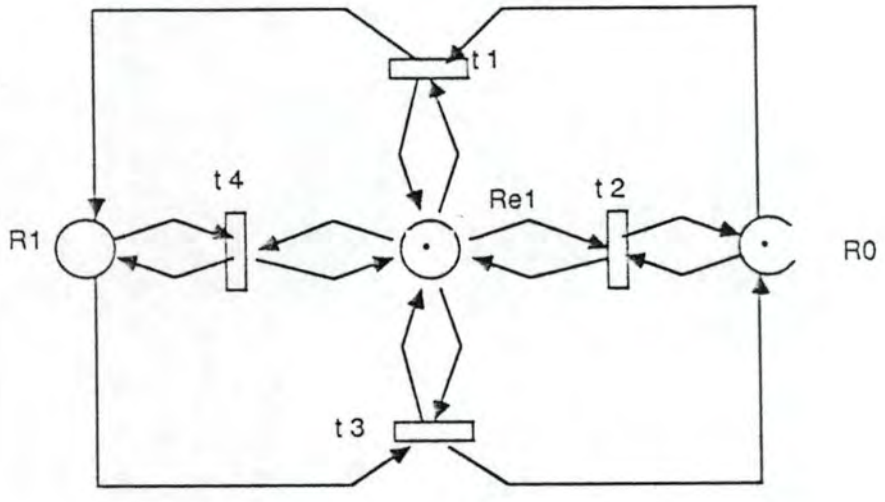
figure 3.15. Modèle du récepteur

Dans une deuxième phase, nous éliminerons les variables apparaissant dans les étiquettes (ici les variables E et R) en associant une ou plusieurs places à chacune de ces variables. Ainsi E et R prenant les valeurs 0 et 1, chaque valeur pourra être représentée par une place (E0,E1,R0,R1). Les opérations sur ces variables sont très simples (l'incrémentation modulo et test d'égalité) et peuvent être représentées par de simples transitions. Ainsi l'opération E = modulo(E + 1) consiste tout simplement à exprimer par une transition chaque changement de valeur de E (E0->E1 et E1->E0). De même le prédicat E = 1 sera traduit par une boucle sur la place E1. La traduction complète des modèles prédicats/actions donne lieu aux figures 3.16. et 3.17..



transition	prédicat	action
t 1	↓ ? DT-Req (Mes)	-
t 2	-	! Mes1
t 3	-	! Mes0
t 4	? Ack0	-
t 5	? Ack1	-
t 6	? Ack1	-
t 7	? Ack0	-
TO	-	-

figure 3.16. Modèle de l'émetteur avec prise en compte des variables internes



transition	prédicat	action
t 1	? Mes0	! Ack0; ! DT-Ind (Mes)
t 2	? Mes1	! Ack1
t 3	? Mes1	! Ack1; ! DT-Ind (Mes)
t 4	? Mes0	! Ack0

figure 3.17. Modèle du récepteur avec prise en compte des variables internes

Ce qui manque encore dans ces modèles, ce sont les interactions entre émetteur et récepteur. La phase suivante consiste donc à assembler les deux modèles par un modèle du service de transfert de données fourni par la couche inférieure. Cette couche peut recevoir (émettre) un message de (vers) la couche supérieure mais aussi le perdre. Le résultat de cette interconnexion nous conduit au schéma de la figure 3.18.. Les places EtAck et EtMes indiquent l'état du médium. Si une de ces

deux places est marquée, ceci signifie que le médium est vide. Les transitions intitulés PerteAck et PerteMes tiennent compte du fait que des messages peuvent être perdus par le médium. Au maximum, le médium peut contenir un message et une confirmation. Un message (respectivement une confirmation) ne peut donc être émis sur le médium que si la place EtMes (respectivement EtAck) est marquée. On peut augmenter la taille du canal en augmentant le nombre de jetons dans les places EtAck et EtMes.

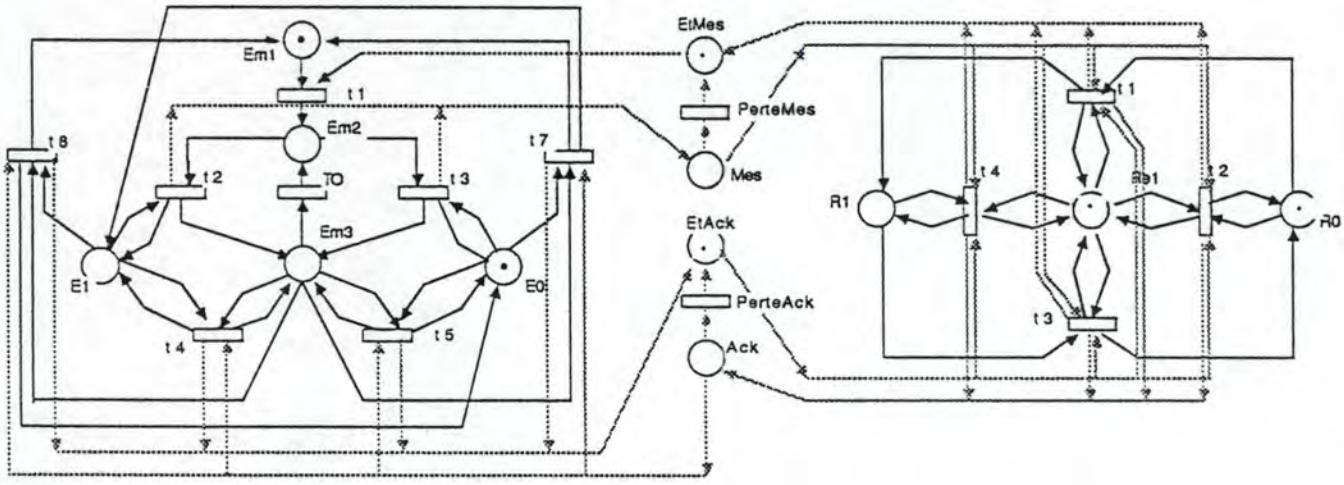


figure 3.18. Modèle global du protocole du bit alterné

3.4.4.2. La validation

Sur le modèle global nous pourrons effectuer un certain nombre de vérifications. Dans un premier temps, le réseau révélera qu'il est binaire (ou sauf) et vivant assurant donc la cohérence du modèle (pas de blocage, nombre fini d'états). Cependant, le point fondamental à vérifier dans le cas du bit alterné est de montrer que le transfert de données s'effectue sans perte ni duplication de messages. Cette propriété peut être exprimée autrement en disant que la requête de données (↓ DT-Req(Mes)) et son indication (î DT-

Ind(Mes)) à l'autre bout alternent. La projection du modèle global vis-à-vis de la requête et l'indication de la primitive de transfert nous donnent une réponse positive à cette question.

Néanmoins, le réseau de la figure 3.17. n'assure pas que l'indication de la primitive DT-Ind aura effectivement lieu dans la mesure où il peut y avoir un nombre infini d'occurrences de transitions entre Dt-Req et Dt-Ind. Ainsi, on pourra exhiber un invariant de transitions (donc une séquence cyclique de tir de transitions) qui ne contienne pas de transitions étiquetées par DT-Ind. Pour assurer la propriété de progression du protocole, il faudrait faire l'hypothèse (réaliste d'ailleurs) que le nombre de pertes est borné et que l'on ne se trouve pas dans un cas catastrophique de rupture de connexion entre l'émetteur et le récepteur.

Cet exemple encore relativement simple, nous montre déjà que la modélisation mène très rapidement à une grande complexité. L'analyse "à la main" se montre vite impossible en manipulant des modèles de taille industrielle et le recours à des outils logiciels de conception et d'analyse devient une nécessité pour rester opérationnel. Au chapitre suivant, nous exposerons les fonctionnalités requises d'un tel outil et nous présenterons quelques outils commercialisés.

CHAPITRE 4

LES OUTILS DES RÉSEAUX DE PETRI

Bernd Jodocy

4.0. INTRODUCTION

Jusqu'au début des années 80, l'utilisation industrielle des réseaux de Petri a été freinée par le manque d'outils logiciels adaptés. L'utilisation pratique des réseaux de Petri est - comme celle de chaque autre technique de description - très dépendante de l'existence d'outils adéquats qui peuvent aider l'utilisateur à affronter les nombreux détails et la complexité d'une large description. Pour les réseaux de Petri, il existe un besoin d'outils supportant aussi bien la construction que la modification et l'analyse de réseaux.

Dans ce chapitre, nous décrirons quel genre d'outils sont utiles et nécessaires et quelles sont les exigences auxquelles ces outils doivent satisfaire pour pouvoir supporter le travail de l'utilisateur d'une façon naturelle et effective. Finalement, nous présenterons quelques exemples d'outils existants. Il s'agit de deux outils que nous avons eu l'occasion de voir et d'utiliser à Toulouse. D'autres outils sont décrits en [BILLINGTON88].

4.1. DES OUTILS POUR LA CONSTRUCTION ET L'ANALYSE DES RÉSEAUX DE PETRI

4.1.1. Nécessité d'outils réseaux de Petri

Les réseaux de Petri, pour être utilisables d'une manière confortable et efficace, ont besoin de différents outils informatiques [JENSEN87] pour assister l'utilisateur lors de la construction, la modification et l'analyse de larges descriptions.

En appliquant de tels outils, l'utilisateur peut profiter de plusieurs avantages [JENSEN87]. Le plus important est la possibilité de créer de meilleurs résultats. Par exemple, des systèmes graphiques de conception procurent à l'utilisateur une précision et une qualité de dessin qui excède de loin les capacités manuelles de la plupart des êtres humains. D'une manière analogue, le support ordinateur pour des techniques d'analyse compliquées permet d'obtenir des résultats qui pourraient difficilement être obtenus par des techniques manuelles (à cause d'un manque de temps ou d'un trop grand risque d'erreurs).

Un autre avantage important est la possibilité de créer les résultats plus rapidement qu'à la main. Ainsi certaines techniques d'analyse peuvent être complètement ou partiellement automatisées; par exemple, construire un graphe des marquages accessibles à la main est un processus très fastidieux; par contre le créer par ordinateur, même pour des graphes très grands, ne prend que quelques instants et est sans risque d'erreur.

Un troisième avantage est la possibilité de faire des présentations interactives des résultats. Un simulateur permet de montrer une séquence de franchissement d'un réseau qui décrit, par exemple, un protocole de communication, d'une manière très animée. A chaque étape, l'utilisateur peut - sur la représentation graphique - voir les transitions qui sont sensibilisées et choisir celle à franchir pour expérimenter les comportements alternatifs d'un protocole.

Un quatrième avantage est la possibilité de cacher certains aspects plus techniques de la théorie des réseaux de Petri dans l'outil. Ceci permet à l'utilisateur d'appliquer certaines méthodes d'analyse sans avoir une connaissance approfondie de la théorie sous-jacente.

Finalement, des systèmes informatiques peuvent aider l'utilisateur à structurer le processus de conception. Ainsi certains éditeurs graphiques pour réseaux de Petri limitent la taille maximale d'un réseau et forcent de cette façon l'utilisateur à dessiner un réseau d'une taille raisonnable. L'utilisateur peut raffiner alors certaines places dans ce réseau en dessinant un nouveau réseau pour chacune d'elles. Ce processus mène à une hiérarchisation en réseaux de petite taille, ce qui rend les réseaux plus lisibles et compréhensibles.

Les différents types d'avantage ne sont bien sûr pas indépendants. Ceci est particulièrement vrai pour des applications industrielles. Dans ce cas, la possibilité de produire rapidement des résultats - d'une bonne qualité - sans connaissance approfondie de la théorie des réseaux de Petri est un prérequis absolu pour l'utilisation des réseaux de Petri comme technique de description et d'analyse dans ce domaine.

Des fonctionnalités de construction, de modification et d'analyse font partie intégrante de presque chaque outil réseaux de Petri et elles sont en fait très proches l'une de l'autre. Il est difficile d'établir une frontière stricte entre construction et modification. De la même façon, des résultats de l'analyse mènent souvent à la modification du réseau et ces modifications, à nouveau, exigent une nouvelle analyse.

4.1.2. Les qualités d'un bon outil réseaux de Petri

Quelles sont donc les qualités qui permettent de caractériser un bon outil réseaux de Petri? [BILLINGTON88] Une première qualité importante est certainement que l'outil est sous le contrôle de

l'utilisateur et pas l'inverse. L'utilisateur peut lui-même décider quand il faut appliquer un certain outil d'une certaine manière. Il peut choisir parmi les différentes techniques de construction, de modification et d'analyse.

Une seconde qualité est que l'outil est facile à manipuler. Mais notons que des tels outils ne peuvent jamais être efficacement utilisés après seulement quelques heures d'apprentissage. Les tâches qu'exécute un tel outil sont généralement tellement compliquées qu'un certain temps d'entraînement et d'apprentissage est nécessaire pour devenir un utilisateur expert. Il est certes important que l'utilisateur, après quelques heures d'apprentissage, puisse commencer à exécuter des tâches faciles, mais il est aussi important que l'utilisateur - avec l'expérience et l'utilisation quotidienne - soit capable d'améliorer sa façon de maîtriser le système. D'une manière analogue, un bon outil devrait - en parallèle - procurer une interface très conviviale pour l'utilisateur occasionnel et une interface efficace pour l'utilisateur fréquent.

Une autre qualité est la généralité de l'outil. L'outil devrait permettre à l'utilisateur de construire la plupart des choses qu'il souhaite produire. D'autre part, nous souhaitons que les outils garantissent une certaine qualité des produits. Normalement, nous attendons d'un éditeur graphique de réseaux de Petri de nous empêcher de violer des règles fondamentales de la théorie comme: connecter deux transitions par un arc sans interposer une place ou laisser un arc en suspension quand une place a été détruite [1].

De bons outils nous empêchent de commettre les types d'erreurs décrits ci-dessus, mais un très bon outil nous permet également - d'une manière explicite - de court-circuiter les règles standards et de nous permettre ainsi de créer le produit exactement de la manière que nous souhaitons. Un bon outil est

---

[1] Ce type de contrôle structurel est similaire à l'utilisation d'éditeurs syntaxiques pour des langages de programmation.

donc caractérisé par une balance entre une structure imposée par le système et une liberté d'expression offerte à l'utilisateur.

Notons que beaucoup des qualités recherchées pour un outil réseaux de Petri qu'on vient d'énoncer ci-dessus sont également souhaitables pour un outil logiciel quelconque.

4.1.3. Les outils pour les réseaux de Petri

Pour travailler pratiquement avec des réseaux de Petri, l'utilisateur a besoin de plusieurs types d'outils :

• un éditeur graphique par lequel nous pouvons construire et modifier des réseaux de Petri en travaillant directement sur la représentation graphique. De tels éditeurs doivent fournir une grande puissance d'expression et une assistance réelle dans la manipulation de vastes réseaux.

• un éditeur textuel par lequel nous pouvons construire et modifier des réseaux de Petri, en travaillant sur leur représentation textuelle (exprimée sous forme de matrices d'incidence et/ou de relations de flux).

• des programmes d'analyse qui aident à l'application de différentes techniques d'analyse et donnent donc des informations sur les propriétés des réseaux de Petri.

Il est important de souligner que ces trois types d'outils forment un ensemble cohérent. Il doit être possible de traduire automatiquement une représentation graphique en une représentation textuelle équivalente qui alors peut être modifiée et analysée. D'une manière analogue, certains des résultats d'analyse peuvent directement être représentés graphiquement tandis que d'autres sont plus facilement exprimés sous forme textuelle.

Voyons, dans les sections suivantes, ces différents outils d'une façon plus détaillée.

4.2. LES ÉDITEURS GRAPHIQUES DE RÉSEAUX DE PETRI

L'utilisateur devrait être capable de travailler directement avec la représentation graphique des réseaux de Petri. Cette exigence implique que l'utilisateur devrait disposer d'un éditeur graphique implémenté sur une station de travail graphique (ou au moins un terminal graphique). L'utilisateur peut alors reconfigurer et déplacer les éléments individuels d'un réseau à l'aide d'une souris. Les résultats sont immédiatement affichés sur un écran graphique dans une forme qui est proche de celle du produit imprimé.

Les éditeurs graphiques devraient avoir une très large puissance d'expression, c'est-à-dire l'utilisateur devrait être libre de choisir parmi les différentes manières de dessiner un réseau. Idéalement, un utilisateur devrait être capable de dessiner les places avec des formes et tailles diverses, des arcs qui sont courbés, droits ou en pointillé et d'associer des textes en divers styles et tailles à des places et arcs .

Les éditeurs doivent être capable de manipuler de vastes réseaux d'une manière structurée. Ceci devrait de préférence être atteint en remplaçant une place par un nouveau sous-réseau qui donne une description plus détaillée du système représenté par la place.

Egalement, il devrait être possible de générer la représentation textuelle automatiquement à partir de la représentation graphique.

Finalement, l'éditeur devrait remplir les exigences normales qu'on attend d'une bonne interface utilisateur d'un système graphique général, à savoir offrir des menus, être orienté objet, donner une rétroaction visuelle suffisante, afficher des informations d'aide,... .

4.3. LES ÉDITEURS TEXTUELS DE RÉSEAUX DE PETRI

Nous avons besoin d'éditeurs textuels pour pouvoir travailler directement sur la représentation textuelle de réseaux de Petri. Cette représentation peut aussi bien être réalisée sous forme d'une matrice d'incidence que par l'intermédiaire de "relations de flux".

La matrice d'incidence décrit pour chaque place l'effet du tir de chaque transition du réseau. Cet effet s'exprime pour les réseaux de Petri classiques sous forme d'un entier (le nombre de jetons ajoutés moins le nombre de jetons retirés) et pour des réseaux de haut niveau - comme les réseaux prédicat/transition - sous forme de fonctions ou d'expressions formelles contenant des variables libres.

Une relation de flux peut être représentée sous forme de tuples constitués du nom de la place, du nom de la transition et d'un entier/d'une fonction/d'une expression décrivant la relation entre la place et la transition.

Les deux représentations textuelles sont équivalentes et il est facile d'opérer la traduction automatique de l'une à l'autre. On peut considérer tout de même la représentation par relations de flux comme étant la plus préférable parce qu'elle est plus proche de la langue humaine et des langages de programmation. La représentation matricielle, par contre, est plus compacte et ceci est utile quand on traite des réseaux d'une taille moyenne ou grande et si on veut afficher un réseau sur un seul écran ordinateur.

Un éditeur textuel de réseaux de Petri devrait proposer beaucoup des fonctionnalités qu'on trouve également chez des éditeurs normaux. En plus, un tel éditeur devrait reconnaître la structure du réseau et vérifier que le réseau construit soit cohérent avec le type de réseau spécifié par l'utilisateur. De même l'éditeur devrait faciliter les opérations qui ont une signification bien précise pour des réseaux de Petri, comme par exemple pouvoir détruire ou réordonner des places sans devoir réécrire la matrice d'incidence ou la relation des flux.

4.4. LES PROGRAMMES D'ANALYSE POUR RÉSEAUX DE PETRI

Les techniques d'analyse décrites au chapitre 1 (section 4) sont utilisables sans moyen automatique de calcul. Cependant ceci n'est vrai que pour de très petits réseaux. Il n'est pas possible sans assistance ordinateur d'analyser des réseaux représentant des systèmes réels. C'est pourquoi nous avons besoin d'un outil qui offre des fonctionnalités d'analyse. Dans cette section, nous allons décrire les types de programmes qu'un tel outil devrait proposer.

4.4.1. Simulation de séquences de tirs singuliers

Ce type de programme permet à l'utilisateur d'exécuter différentes séquences de franchissement du réseau étudié. La simulation peut être soit interactive, soit automatique [1].

Dans les deux cas, les séquences de marquages sont calculées par le système et le marquage individuel est affiché soit par une représentation textuelle, soit par une représentation graphique. Pour des réseaux de Petri classiques, le marquage peut être indiqué directement sur la représentation graphique par des jetons. Pour les autres types de réseaux, il peut être nécessaire ou plus commode d'utiliser des formules textuelles.

4.4.1.1. Simulation interactive

La simulation interactive est principalement utile pour des tests pendant la conception d'un système compliqué, ainsi que pour des fins de démonstration. Elle ne permet pas de prouver qu'un système est correct, mais elle peut être extrêmement précieuse pour détecter des erreurs pendant la conception. La simulation permet d'offrir une meilleure compréhension de systèmes compliqués.

---

[1] Dans le premier des cas, l'utilisateur choisit à chaque étape les transitions à franchir, tandis qu'au deuxième cas, les transitions à tirer sont sélectionnées par le système (normalement d'une façon non déterministe).

En utilisant une représentation graphique, le système peut faire ressortir les transitions sensibilisées. Ceci permet un dialogue agréable où l'utilisateur voit les transitions sensibilisées et sélectionne celle à franchir à l'étape suivante et aperçoit le résultat de ce tir directement à l'écran.

Dans une simulation interactive, il devrait être possible de revenir en arrière. L'utilisateur pourrait alors examiner comment un certain marquage a été atteint ou essayer les différentes façons de résoudre le conflit entre un ensemble de transitions sensibilisées simultanément.

4.4.1.2. Simulation automatique

La simulation automatique peut être appliquée pour exécuter des simulations longues - en testant de cette façon les nombreuses combinaisons d'états possibles qui peuvent exister, par exemple, dans un protocole de communication.

La simulation automatique peut également être utilisée pour l'implémentation et le prototypage rapide. Alors, à chaque transition, est attaché un module du programme qui est invoqué chaque fois que la transition est sensibilisée. De cette manière il est possible de programmer les "parties concurrentes" au moyen de réseaux de Petri tandis que les "parties séquentielles" sont programmées par des langages de programmation classiques.

4.4.2. Le calcul du graphe des marquages

Un programme devrait permettre à l'utilisateur de construire automatiquement le graphe des marquages accessibles d'un réseau de Petri. L'idée de base est d'avoir pour chaque marquage un nœud d'un graphe. Deux nœuds sont connectés par un arc dirigé s'il existe une transition permettant d'atteindre le second marquage à partir du premier par le franchissement d'une transition du réseau. Nous avons tout de même fait remarquer (cfr. chapitre 1) que le graphe ainsi construit pouvait être infini. Pour résoudre ce problème, la méthode de l'arbre de couverture permet de concentrer un groupe de

marquages similaires dans un seul nœud et d'obtenir ainsi un graphe d'une taille finie. Même si la méthode de l'arbre de couverture aboutit à un graphe fini, il est tout de même généralement assez grand. Afin de pouvoir calculer un graphe en pratique, il nous faut donc un programme de construction du graphe.

Ces graphes - nous l'avons vu - sont utilisés pour prouver certaines propriétés (bornage,...) du réseau de Petri correspondant. Certaines de ces preuves peuvent être automatisées ou au moins assistées par l'ordinateur. Pour atteindre cette fin, nous avons besoin de programmes qui collectent des statistiques sur les graphes, qui nous indiquent par exemple l'existence de branches mortes, le nombre maximal de jetons dans chaque place,... .

Egalement des programmes assistant l'utilisateur dans des recherches compliquées de marquages ou séquences de transitions ayant certaines propriétés seraient fort précieux.

4.4.3. La réduction de réseaux de Petri

Ces programmes permettent de transformer un réseau dans un réseau équivalent, d'une plus petite taille. La transformation est accomplie au moyen de règles de réduction (cfr. chapitre 1).

La réduction peut soit être automatique, soit être interactive. Dans le dernier cas, l'utilisateur détermine comment appliquer les règles de réduction tandis que le système calcule le réseau réduit et veille à la validité des réductions exécutées. Les réductions peuvent soit être effectuées sur la représentation graphique, soit sur une des représentations textuelles.

Pour toutes les méthodes de réduction il est important que le résultat d'analyse du réseau réduit puisse immédiatement être interprété en terme du réseau originel. Il n'est souvent pas suffisant de savoir qu'il y a un blocage ou une place non bornée dans le réseau réduit, l'utilisateur veut aussi savoir comment arriver à cette situation de blocage et où se trouvent les places non bornées en terme du réseau originel.

4.4.4. La projection de réseaux de Petri

Nous avons vu au premier chapitre qu'il est possible de réduire le graphe des marquages accessibles en définissant un ensemble de transitions "intéressantes" et de retenir pour le graphe uniquement les séquences contenant ces transitions en rendant les autres invisibles.

Pouvoir faire ceci d'une manière automatique offre certainement une facilité fort utile pour l'analyse de réseaux ayant un graphe de marquages d'une grande taille.

4.4.5. Le calcul des invariants

Pour certains types de réseaux, le calcul des invariants de places et de transitions peut être entièrement automatisé. Pour d'autres, le calcul peut être plus compliqué. Il n'est alors plus possible de construire un programme qui calcule tous les invariants pour tous les réseaux possibles.

Un grand effort de recherche pour le développement de techniques de calcul de réseau de haut niveau a été accompli et les différents systèmes poursuivent des techniques parfois très différentes. Certains d'entre eux procèdent par calcul automatique mais en restreignant les types d'invariants et de réseaux à prendre en compte. D'autres calculent les invariants à l'aide d'un dialogue, où l'utilisateur décide ce qu'il faut faire à chaque étape pendant que l'ordinateur exécute les calculs et vérifie la validité des opérations.

Une fois les invariants calculés, l'utilisateur peut les appliquer pour prouver diverses propriétés du réseau de Petri. De telles preuves sont normalement faites manuellement, mais il peut être possible - et aussi attractif - d'utiliser des systèmes automatiques de preuves pour déduire des propriétés du réseau à partir des invariants.

4.4.6. La vérification d'invariants proposés

Souvent, il n'est pas nécessaire de calculer les invariants, tout simplement parce que, dès le départ, le concepteur a une idée très nette sur l'ensemble des invariants qu'il attend que le système respecte.

Dans ce cas, il est plus adéquat de vérifier des invariants proposés par l'utilisateur. Le système examine si l'invariant est vrai ou non pour le réseau en question. Dans le dernier cas, la réponse doit être plus constructive au sens que l'outil indique les transitions qui violent l'invariant. Généralement, il est alors facile pour le concepteur de changer - soit l'invariant proposé, soit le réseau - de façon à obtenir un invariant valide.

4.4.7. La vérification de propriétés structurelles

Ce type de programme est apte à effectuer une vérification automatique de certaines propriétés structurelles qui sont connues et à impliquer certaines bonnes propriétés.

Ainsi on pourrait vérifier pour un réseau place/transition qu'il est marqué. En plus, si c'est le cas, on pourrait vérifier que chaque circuit contient au moins un jeton et que chaque arc est sur un circuit avec au maximum un jeton (ce qui implique que le réseau est vivant et sauf). Ce type d'outil est très important et relativement facile à implémenter et utiliser.

Après avoir défini ce que serait un bon outil réseaux de Petri, quelles seraient les fonctionnalités souhaitées? Voyons quelques outils existants. Le premier outil, OVIDE, qu'on présentera est un outil manipulant les réseaux de Petri classiques tandis que le deuxième, PIPN, permet aussi la manipulation de réseaux de haut niveau, les réseaux prédicat/transition.

4.5. OGIVE/OVIDE

Un des tout premiers outils logiciels soutenant la tâche du concepteur de réseaux de Petri fut un outil développé au LAAS à Toulouse, intitulé OGIVE (et commercialisé sous le nom OVIDE) [OVIDE82]. Cet outil était lors de sa publication relativement révolutionnaire et connut - et connaît toujours - un grand succès. Il offrait déjà la plupart des fonctionnalités énoncées ci-dessus et regroupait toutes les techniques d'analyse connues à ce jour au sein d'un environnement complet.

Les grandes fonctions de cet outil sont les suivantes :

- description de réseaux (création/modification) par une approche graphique ou textuelle (au choix de l'utilisateur),
- analyse classique de réseaux par construction du graphe des marquages accessibles,
- réduction de réseau conservant les propriétés de vivance et de bornage avec possibilités de contraintes,
- recherche des invariants de places et de transitions.

4.5.1. Caractéristiques générales

OVIDE est piloté à partir d'un terminal graphique équipé d'un écran de visualisation, d'un clavier alphanumérique, d'un moyen de désignation écran (light-pen, souris ou tablette de digitalisation).

L'appel aux différentes fonctions d'OVIDE se fait par choix dans un menu d'aiguillage présenté sur l'écran. Au niveau d'une fonction, l'appel à une sous-fonction se fait aussi par choix dans un menu local qui déclenche, le cas échéant, une chaîne de dialogues homme-machine guidée et contrôlée par l'outil.

4.5.2. La construction et modification de réseaux

Deux méthodes sont proposées par OVIDE à l'utilisateur pour lui permettre de décrire un réseau de Petri.

4.5.2.1. Description directe

Exclusivement textuelle (entrées/sorties de type alphanumérique uniquement), cette méthode permet de décrire et de modifier directement le réseau de Petri complet qui doit modéliser le système étudié.

A titre d'exemple, on citera les sous-fonctions de création et de suppression de place, transition ou d'arc, de remise à zéro du marquage...

4.5.2.2. Description par mosaïque

Par cette méthode, on construit le réseau complet en décrivant morceau par morceau, puis en fusionnant les morceaux pour former le réseau global.

Chaque morceau de réseau est un sous-réseau qui est décrit de manière interactive par un graphique. Pour garder une certaine lisibilité au dessin, les dimensions d'un sous-réseau sont limitées à 20 places, 20 transitions et 50 arcs.

La fusion de réseaux pour former un réseau complet se fait par les sommets (places ou transitions ayant le même nom). Pour des raisons évidentes de lisibilité, le réseau complet obtenu ne peut être dessiné.

On notera qu'un réseau décrit par la seconde méthode (graphique) peut être manipulé par la première (textuelle).

4.5.3. L'analyse du réseau

Le réseau ayant été décrit, l'utilisateur peut faire appel à la fonction d'analyse. Une première approche consiste à construire le graphe de marquages et à en déduire des propriétés dynamiques comme le bornage, la vivance et la réinitialisabilité du réseau. Les sous-fonctions proposées sont les suivantes :

• analyse par construction du graphe des marquages accessibles

• recherche d'une séquence de tir conduisant du marquage initial à un marquage donné.

L'utilisateur peut également demander une réduction de la structure du réseau. Cette réduction conserve les propriétés de vivance et de bornage du réseau avant réduction, sur le réseau réduit. L'utilisateur peut demander que certaines places ou transitions ne soient pas réduites, même si elles sont réductibles.

Une autre fonction permet de rechercher les invariants de places et de transition du réseau.

4.5.4. Conclusion

OVIDE est un environnement relativement complet et offre une grande diversité de fonctionnalités assistant le concepteur dans sa tâche de création et d'analyse de réseaux de Petri. Particulièrement, le calcul automatique des invariants est fort précieux. Un désavantage de cet outil est son incapacité à traiter des réseaux de haut niveau, cependant fort utilisés dans des nombreuses applications. On explicitera, dans la section suivante, un outil qui est capable de manipuler une classe de réseaux de haut niveau.

6. PIPN

Un outil plus récent est PIPN (**P**rolog **I**nterpreted **P**etri **N**ets) (voir [AZEMA87], [AZEMA88], [LLORET88] et [PIPN88]). Comme OVIDE, il a été réalisé par l'équipe OLC du LAAS en collaboration avec la société VERILOG. Comme son nom le suggère, il a été développé dans un environnement PROLOG. Le but principal de ce logiciel est d'offrir une aide aux informaticiens impliqués dans la spécification et vérification de protocoles de communication.

PIPN est à la fois un langage de description et un environnement de prototypage rapide et de vérification d'algorithmes distribués.

Le langage permet une approche modulaire et hiérarchique d'un problème. Il offre plusieurs modes de synchronisation (rendez-vous, file FIFO) entre modules (sous-réseaux). Le langage PIPN est une extension des réseaux de Petri de haut niveau, les réseaux prédicat/transition ou réseaux à prédicats tout court.

L'environnement PIPN poursuit trois objectifs :

- offrir un prototypage rapide avec une convivialité maximale,
- offrir des facilités de vérification permettant d'augmenter la confiance dans le modèle construit,
- offrir un environnement ouvert aux deux techniques de spécification formelle retenues par l'ISO (LOTOS et ESTELLE).

4.6.1. Les caractéristiques du langage PIPN

Le langage PIPN est un langage reposant sur le formalisme des réseaux de Petri à prédicats. Les réseaux à prédicats sont construits sur la logique du premier ordre. Ce langage est particulièrement adapté à la modélisation concise et précise de systèmes communicants, à base de processus parallèles, distribués et synchrones. Les protocoles de télécommunication conçus dans le cadre conceptuel du modèle ISO constituent un domaine d'application particulièrement adapté à la modélisation par réseaux à prédicats [LLORET 88].

Les réseaux à prédicats ont été étendus par la notion de "label". Un label est une étiquette des transitions dont le rôle est de décrire les communications de la transition avec son environnement. L'introduction du label comme nouvel élément sémantique justifie l'adoption de la nouvelle terminologie de "réseaux à prédicats labellés".

La motivation principale de cette approche est une meilleure prise en compte des données[1] par rapport aux réseaux de Petri classiques. Un autre intérêt de cette sous-classe est d'autoriser une description et une simulation (mise en œuvre de la règle de tir) de façon immédiate dans un environnement PROLOG.

4.6.1.1. Architecture modulaire

Un réseau, appelé dans le vocabulaire PIPN entité, est une hiérarchie statique père-fils de modules ou sous-réseaux (cfr figure 4.1.) . Le module "père" est constitué d'un ensemble de "fils" dont les comportements, mis en coopération par interconnexion, induisent le comportement du père. Ainsi le module STATION a deux fils (CONSOMMATEUR et PRODUCTEUR) qui, a leur tour, ont plusieurs fils.

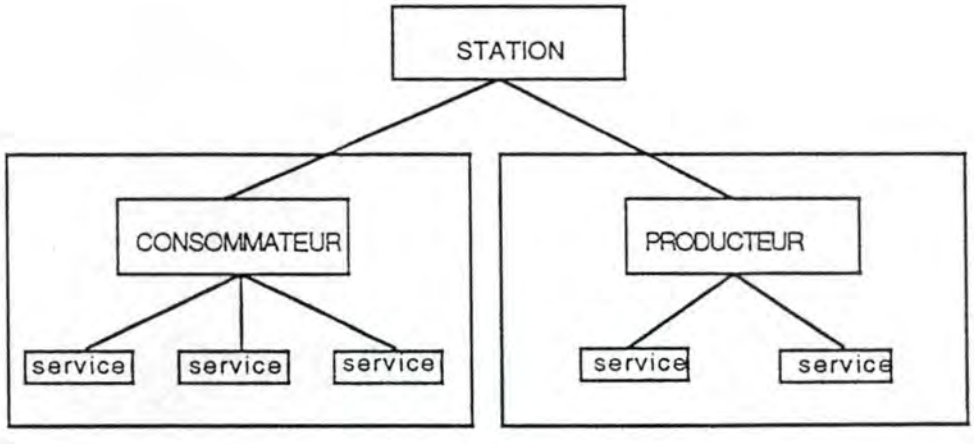


figure 4.1. Exemple de modèle

[1] C'est notamment nécessaire pour les protocoles qui utilisent des estampilles ou numérotations.

Chaque module comporte un corps et une interface : l'interface donne la visibilité externe du module, et le corps sa dynamique.

Cette architecture entre donc tout à fait dans la philosophie ISO en reprenant les idées d'encapsulation et d'hiérarchisation des modules (services). Cette architecture modulaire permet de construire le modèle de façon ascendante, par intégration successive des modules ou de façon descendante en réalisant un affinage de la structure. Chaque module peut être considéré comme une boîte noire.

4.6.1.2. Un module : une boîte noire communicante

Un module peut être considéré comme un processus qui n'interagit avec son environnement que par des points d'interaction. Le principe de communication est le rendez-vous : un module communique avec son environnement lorsque simultanément l'un est prêt à recevoir l'interaction, l'autre à l'émettre.

Le concepteur peut tout de même spécifier entre deux points d'interaction de modules distincts un mode de communication différent du rendez-vous, comme par exemple une liste FIFO bornée ou non, avec perte ou fiable.

4.6.1.3. Une transition PIPN : une structure complexe

Une transition PIPN est un modèle d'événement local à un module caractérisé par :

• un ensemble d'actions : la transition ne peut être tirée que si l'environnement du module est prêt à répondre d'une manière indivisible à toutes les interactions élémentaires (par exemple l'émission ou la réception d'un message) en garde de la transition.

• un ensemble d'instances de places en précondition : les préconditions sont des instances de places qui doivent correspondre chacune à un élément d'état de l'état courant différent afin que la transition soit autorisée. Les éléments d'état correspondant aux préconditions seront retirées de l'état courant pour obtenir le nouvel état.

• un ensemble d'instances de place en postcondition : les postconditions sont les instances de place dont une instanciation de base donnera les nouveaux éléments d'état qui figureront dans l'état atteint après tir de la transition.
• les conditions sur les données: elles sont un ensemble de relations sur les variables apparaissant en garde, en pré- ou postcondition, qui doivent être vérifiées pour que le tir de la transition soit autorisé. Les relations sont données par des procédures PROLOG.

4.6.2. La structure et la syntaxe du langage PIPN

4.6.2.1. La structure du modèle

Une entité (module) est composée d'***un en-tête*** (header), d'un ***fichier de conditions*** (cond) et d'***un corps*** (body). Chaque entité a un en-tête particulier, mais plusieurs entités peuvent se partager le même fichier de conditions et (ou) le même corps.

L'en-tête d'un module donne son état initial et une description de sa communication avec les autres modules. Cet entête est placé sur un fichier <NAME>.HEADER.PRO où NAME est le nom de l'entité.

Le corps comprend les transitions du réseau de prédicats et donne ainsi le comportement du module. Ce corps se trouve sur le fichier <NAMEBODY>.BODY.PRO où NAMEBODY est le nom du corps.

Le fichier de conditions contient les conditions sur les données qui doivent être remplies pour que les transitions puissent être tirées. Ce fichier se trouve dans le fichier <NAMECOND>.COND.PRO où NAMECOND est le nom du fichier de conditions.

La figure ci-dessous illustre cette structure d'un modèle.

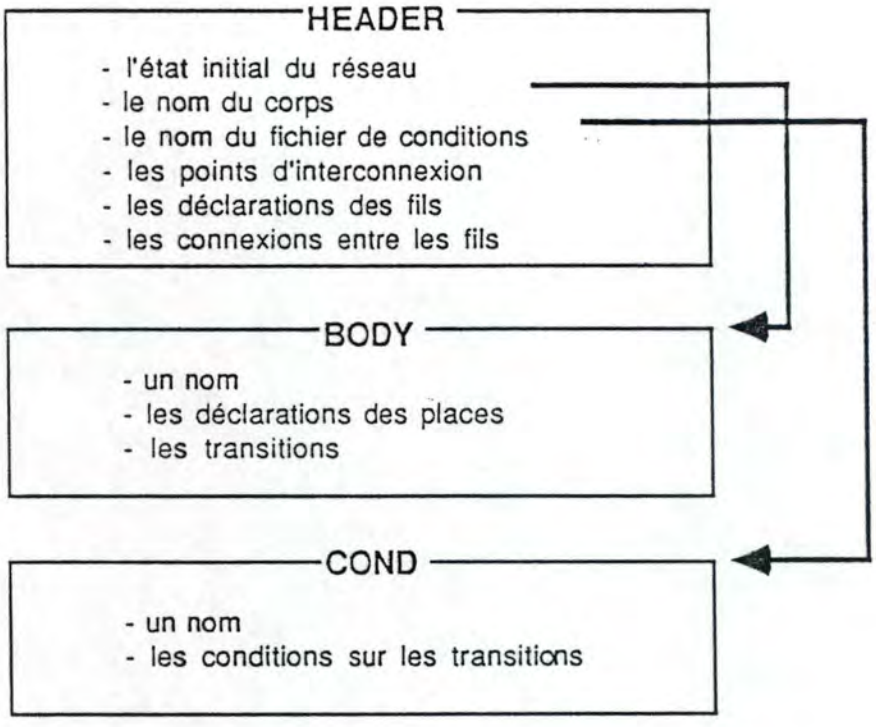


schéma 4.2. La composition d'un modèle PIPN

Il se peut qu'une entité n'utilise pas de fichier de conditions. Dans ce cas, les conditions sont écrites directement dans les transitions. Cependant, si elles sont longues et complexes, il est plus pratique de les écrire dans un fichier de conditions et de les appeler par leur nom.

Une transition PIPN peut être décrite par un terme comprenant :

- le *nom de la transition* : **tr** (tr_name)[1]
- un *label qui décrit la réception* : **input** (ip_name(x))
  le message x est reçu en entrée de la transition au point d'interconnexion ip_name.
- un *label qui décrit l'émission* : **output** (ip_name(y))
  le message y est émis, si la transition est tirée, au point d'interconnexion ip_name.
- une *précondition* : **pre** ([place(i_1),..])
  une suite de places, avec leurs paramètres i_j, qui doivent être marquées pour que la transition puisse être tirée.

[1] Les caractères gras désignent les mots clés du langage.

- une *postcondition* : **post** ([place(j_1),..])
  une suite de places, avec leurs paramètres j_i, qui sont vérifiées après que la transition ait été tirée.
- des *conditions* : **cond** ([fonction(x,i_1,...,y,j_1,...)])
  une fonction qui à partir des données x (reçues en input) et i_1,...(paramètres des places de la précondition), crée les résultats y (émis en output) et j_1,...(paramètres des places en postcondition).[1]

Pour éclaircir ceci nous donnons ci-dessous un exemple d'une transition. Cette transition est extraite d'un modèle qui décrit la couche application d'un protocole de communication inspiré du modèle de l'ISO . Nous avons réalisé cette modélisation lors de notre stage de fin d'étude au LAAS à Toulouse. Au chapitre 5 nous reviendrons d'une manière plus précise sur ce travail.

```
tr(get_conf_readloc_conf,
    label ([ input(I(I_get_conf(id_var,data_liaison))),
             output(a(a_readloc_conf(a_nom,data_application))) ]),
    pre ([ wait_get_conf(id_var) ]),
    post ([ idle ]),
    cond ([  correspondance(id_var,a_nom),
                  création_data(data_liaison,data_application) ])).
```

Cette transition représente la réception en entrée sur le point d'interconnexion *I* d'une confirmation de lecture *I_get_conf* de la variable *id_var* ayant la valeur *data_liaison*. Cette confirmation ne peut être reçue que si la précondition est vérifiée, à savoir que la place *wait_get_conf(id_var)* est marquée. Place qui indique qu'on a précédemment émis une demande de lecture portant sur la variable *id_var*.

L'entité se prépare maintenant à émettre un message *a_readloc_conf* sur le point d'interconnexion. Cette émission ne peut avoir lieu qu'après avoir vérifié la condition.

---

[1] La fonction de la condition peut être décomposée en fonctions plus petites et plus simples à réaliser.

La condition est constituée de deux fonctions. La première recherche le nom *a_nom* correspondant à l'identificateur *id_var* de la variable. La seconde extrait les données de *data_liaison* qui sont nécessaires pour former les données *data_application*.

Après émission du message *a_readloc_conf*, la place *idle* est marquée pour indiquer que l'entité est en état de repos. Pour plus de détails, voir la section suivante.

A l'aide de ce petit exemple, on voit clairement qu'un tel langage de type déclaratif est nettement supérieur du point de vue lisibilité et compréhension à des descriptions de type matrice d'incidence par exemple.

4.6.3. L'outil PIPN

PIPN offre comme tout environnement de réseaux de Petri des fonctionnalités facilitant la mise au point et l'analyse de réseaux de Petri, ou plus précisément de réseaux à prédicats labellés décrits dans le formalisme PIPN.

Notons que le logiciel dans sa version actuelle ne permet pas une description graphique. Une interface graphique est néanmoins en cours d'étude.

La figure 4.3. donne un aperçu sur les fonctionnalités offertes par PIPN.

4.6.3.1. Un outil de développement

Le logiciel PIPN est un outil de développement où l'on peut accéder, à tout moment de l'élaboration d'un modèle, aux fonctions de compilation, simulation et vérification. Il possède tout ce qui est nécessaire à la mise au point d'un modèle, c'est-à-dire :
- un *compilateur* : qui fournit un modèle exécutable et indique les erreurs syntaxiques et sémantiques du modèle.

- un *simulateur* : qui permet de voir, pas à pas, les évolutions possibles du système modélisé, de façon interactive ou en fournissant un scénario.
- un *vérificateur* : qui permet de s'assurer des propriétés du modèle formulé à l'aide d'opérateurs de la logique temporelle.

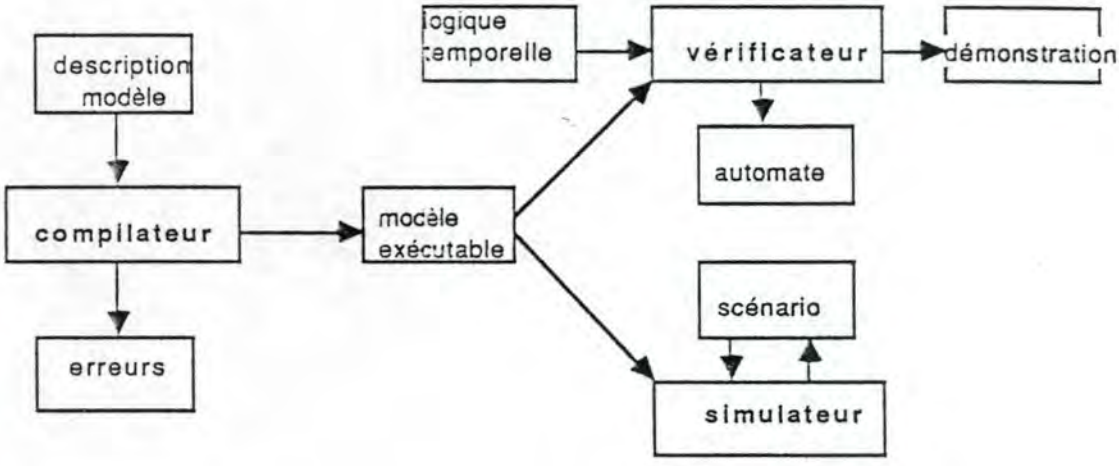


figure 4.3. L'architecture générale de l'outil PIPN

4.6.3.2. Un outil de validation

La première des validations disponibles dans l'outil PIPN, c'est d'utiliser le simulateur afin de vérifier le comportement du modèle par rapport au fonctionnement désiré.

Une seconde validation consiste à utiliser le vérificateur qui permet d'obtenir différents résultats :
- le *graphe des marquages* : qui constitue l'automate du modèle et qui permet de calculer un ensemble de caractéristiques de la dynamique de celui-ci : les états puits, la vivacité et les cycles.
- l'*automate réduit* qui est obtenu par projection de l'automate en fonction de certains critères d'observabilité et d'équivalence. Cette réduction peut être utile pour valider plus simplement une propriété particulière.
- la *démonstration* directe de propriétés exprimées en terme de logique temporelle. Ces propriétés concernent la potentialité ou l'inévitabilité entre certains événements décrits par l'automate. Ainsi on peut, par exemple, vérifier que l'événement A ne peut avoir lieu que précédé par l'événement B, à condition que C n'apparaisse

pas après B et avant A ou d'une manière plus concrète qu'une confirmation de connexion ne peut que suivre une requête de connexion, à la condition qu'aucun site ne se déconnecte.

4.6.4. Conclusion

PIPN est un outil qui vise surtout la vérification de protocoles de communication. Dans sa version actuelle, il offre un grand nombre des fonctionnalités requises pour pouvoir manipuler les réseaux complexes que le concepteur et testeur de protocoles rencontre dans la réalité. Il n'offre pas les fonctionnalités de calcul automatique d'invariants. Les invariants doivent être proposés et explicitement introduits sous forme de formules de la logique temporelle.

Un problème majeur, lors de nos premières expériences avec cet outil, était l'explosion combinatoire de l'espace des états dans le cas d'une étude exhaustive. Pour pouvoir procéder à une étude exhaustive de modèles d'une taille réelle, PIPN offre deux voies de solution. La première est de restreindre la vérification à un sous-modèle (ce qui est facilement réalisable si on a construit un modèle bien hiérarchisé et équilibré). La deuxième solution est de procéder à la réduction du graphe des marquages en exprimant une relation d'équivalence.

CHAPITRE 5

LA COUCHE APPLICATION DE F.I.P.

Pierre Hastir, Bernd Jodocy

5.1. INTRODUCTION

Lors de notre stage au LAAS-CNRS de Toulouse, l'équipe dans laquelle nous étions intégrés nous a confié la tâche de réaliser la modélisation et la vérification de la couche application de la norme FIP en utilisant l'outil PIPN.

L'objectif de ce travail était de montrer l'adéquation de l'utilisation de l'outil PIPN [1] pour la modélisation et la vérification de protocoles de télécommunication et de réaliser cela par son utilisation sur la couche application du protocole FIP (Flux Information Processus). Nous devions, grâce à ce travail, clarifier les mécanismes de la norme écrits dans un langage semi-formel parfois incomplet et réaliser la modélisation des évolutions des services et des éléments de la couche application.

Nous allons décrire dans les sections qui suivent l'environnement de ce travail. Cet environnement est composé de FIP qui est un protocole relatif à la transmission d'informations entre capteurs, actionneurs et automates dans une installation industrielle automatisée (section 5.2.). Cette norme FIP est une collaboration entre les centres de recherches et l'industrie. Elle est actuellement en voie de normalisation.

Cette analyse de la couche application du protocole FIP est réalisée par la modélisation de cette couche. Cette modélisation a été effectuée dans le formalisme des réseaux de Petri labellés à prédicats. Un logiciel qui manipule ce formalisme, PIPN (voir section 4.6. 4) a été utilisé pour la modélisation et la vérification. Un bref rappel de ce logiciel est donné dans la section 5.3.

---

[1] logiciel développé par une collaboration entre le LAAS et la société VERILOG

La modélisation de la couche application a demandé la création d'un modèle de représentation de cette couche. La recherche de ce modèle (section 5.4.), compte tenu des impératifs de PIPN, nous a permis d'arriver à un modèle simplifié qui décompose la couche application en un ensemble de services distincts.

Ayant le modèle, il a fallu traduire la norme en transitions PIPN (section 5.5). Nous y décrivons une méthode de déduction de ces transitions compte tenu du modèle. Cet ensemble de transitions forme le modèle écrit dans la langage PIPN de la couche application du protocole FIP. L'analyse de ces transitions et les automates que l'on peut en déduire sont utilisés pour vérifier le protocole FIP.

En annexe on trouvera le listing du modèle du service de lecture locale A_READLOC. Ce listing est composé des différentes parties du modèle: l'en-tête, le corps et le fichier de conditions.

5.2. FIP

5.2.1. Introduction

F.I.P. (**F**lux **I**nformation **P**rocessus) est un système de communication adapté à la transmission d'informations entre capteurs, actionneurs et automates dans une installation industrielle automatisée [GRFIP[1]].

Actuellement, si les divers réseaux de communication entre la salle de commande et les équipements d'automatismes FSC (**F**lux **S**alle **C**ommande) semblent relativement satisfaisants, il n'en est pas de même pour la communication entre équipements d'automatismes, capteurs et actionneurs (**FIP**).

L'objectif de FIP est de proposer un moyen de communication qui permette de réduire globalement le coût du câblage afin d'améliorer les performances du système tout en satisfaisant aux besoins de l'application et en garantissant la qualité de service, principalement en donnant un meilleur accès aux informations.

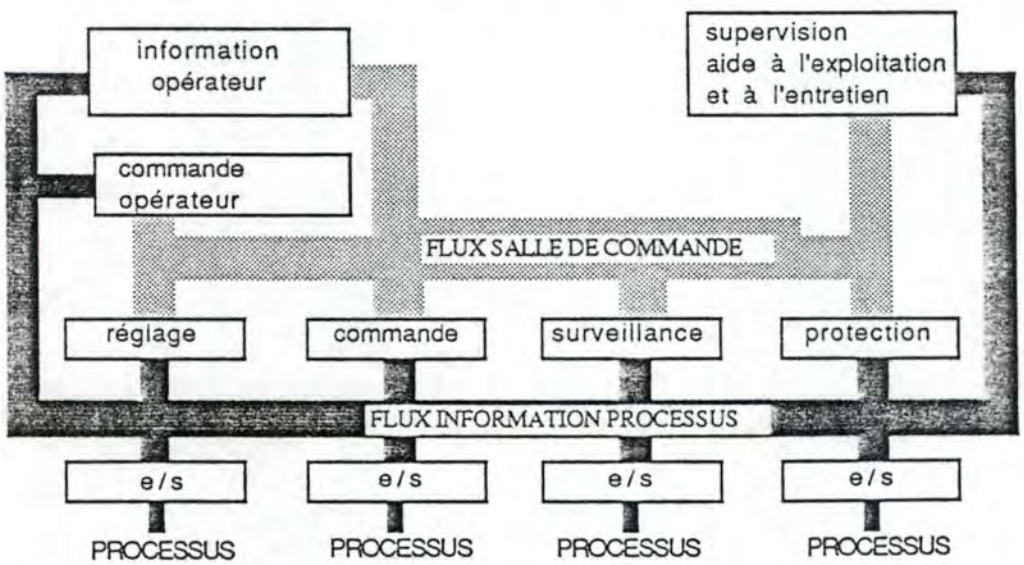


figure 5.1. Les flux dans une installation industrielle automatisée.

[1] groupe de réflexion FIP [GRFIP]

Il n'existe actuellement aucune proposition de système de communication de type FIP au niveau international; le protocole FIP propose une solution.

Le club FIP, une association qui comprend les représentants des différentes sociétés d'utilisateurs, des fabricants et des services de recherche, assure la régie du travail de normalisation du protocole FIP.

5.2.2. Le protocole FIP

Le protocole FIP s'inspire du modèle de référence d'interconnexion des systèmes ouverts de l'ISO. Il reprend les couches 1, 2 et 7 (physique, liaison de données et application) du modèle de l'ISO. La simplicité de FIP exclut l'utilisation de réseaux de communication de type FSC qui utilisent trop de couches logicielles.

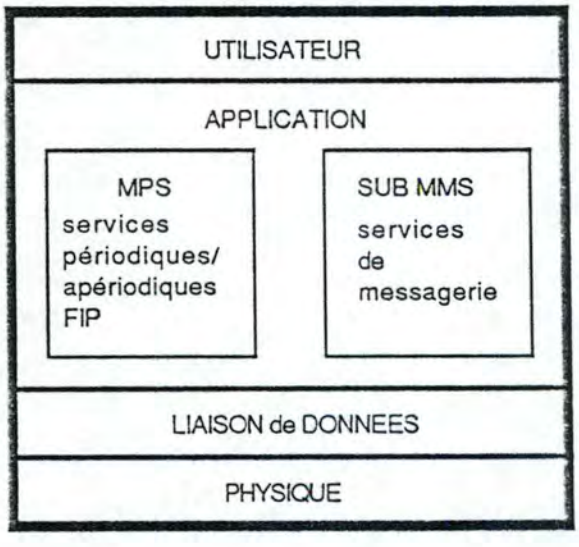


figure 5.2. La structure du protocole FIP

Le modèle retenu par le système FIP met en évidence au niveau de la couche application l'existence de deux familles de services :

1) les services propres à la gestion d'une base de données répartie qui constitue l'originalité de FIP.
2) les services de messagerie.

5.2.3. Le modèle général

Le modèle général permet de décrire les relations entre les services MPS (services périodiques/apériodiques FIP) assurant la gestion de la base de données répartie et l'environnement FIP. L'environnement FIP est celui des systèmes de production continus et discontinus dans lequel les équipements sont interconnectés à l'aide d'un système de commande (**CS**), permettant d'assurer les fonctions d'automatisme.

Les traitements associés aux fonctions d'automatisme sont répartis sur plusieurs stations qui interopèrent.

Un système de commande (CS) est composé d'équipements informatiques (**CSD**) et d'un système de communication permettant d'assurer les fonctions d'automatisme du système de production. Les équipements informatiques (CSD) correspondent aux abonnés raccordés au système de communication.

Un équipement du système de commande (CSD) dispose d'une image virtuelle de plusieurs équipements virtuels du système de commande (**VCD**).

Chaque équipement virtuel du système de commande (VCD) regroupe un ensemble d'équipements virtuels de production (**VMD**) associés à une même application dans un équipement du système de commande (CSD).

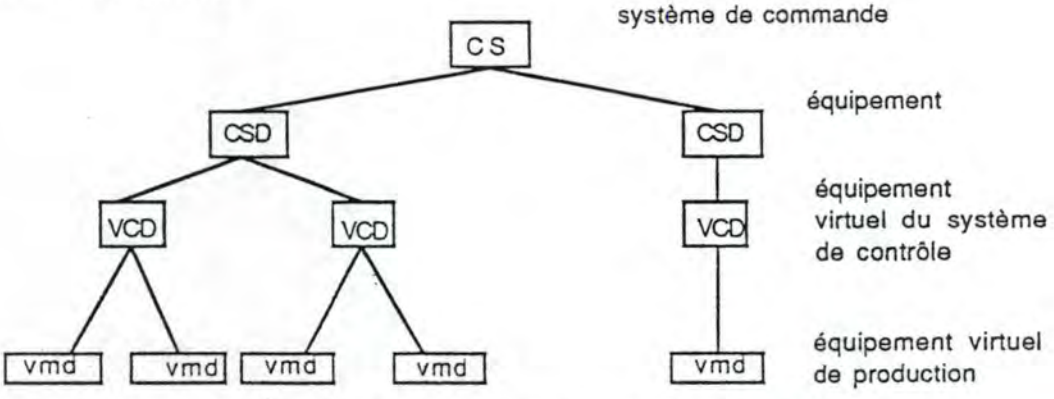


figure 5.3. Le modèle général de FIP

Un équipement virtuel du système de commande (VCD) est constitué :

• d'un *processus application* (**AP**), qui est un modèle abstrait regroupant des équipements virtuels de production (VMD) associés à une même application.

• d'une *entité de communication* (**CE**) qui met à la disposition de l'utilisateur des fonctions de communication élaborées à partir des *éléments de service application* (**ASE**) et à partir des *éléments de service de contrôle liaison de données* (**DLSE**) pour communiquer avec d'autres VCD qui participent à la réalisation d'une application répartie.

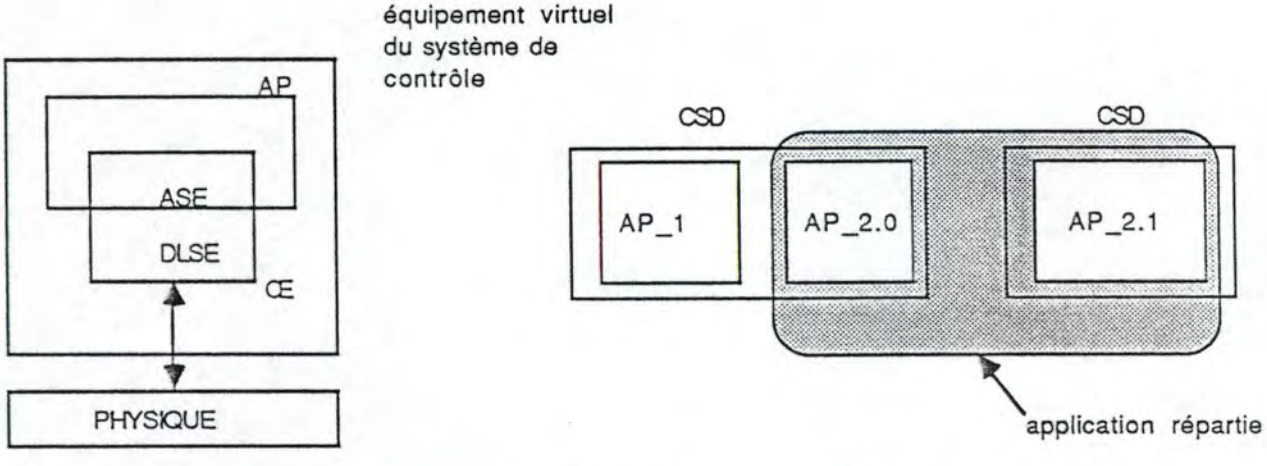


figure 5.4. Un VCD et une application répartie

Une *entité application* (**AE**) représente un sous-ensemble de fonctions de communication associées à un équipement virtuel de production (VCD), fonctions assurées par un ensemble d'éléments de service application (ASE).

5.2.4. Portée du travail réalisé au LAAS

Au LAAS on s'est occupé de la modélisation et de la vérification de la couche liaison de données [RLIDO[1]] et de la couche application du protocole FIP . Notre travail portait sur les résultats de *la modélisation et de la vérification des services propres à la gestion d'une base de données répartie* de la couche application du protocole FIP.

5.2.5. Caractéristiques de la couche application

5.2.5.1. Les objets

La description des services périodiques/apériodiques (MPS) de FIP nécessite la description des objets application de l'environnement FIP. Ceux-ci peuvent être regroupés en deux ensembles :
- les objets relatifs aux variables
- les objets relatifs aux listes de variables.

Les équipements du système de commande (VCD) mettent à disposition d'un processus application (AP) une valeur de chacune des variables reconnues localement.

Dans une application répartie faisant intervenir plusieurs AP, une variable est produite par un AP et consommée par un ou plusieurs autres AP. Une seule entité application (AE) peut être déclarée productrice de la variable alors que plusieurs peuvent être déclarées consommatrices. Certaines AE ne se reconnaissent ni productrice, ni consommatrice d'une variable.

On dira qu'une variable est locale à une entité de communication si elle appartient à ce VCD. On dira qu'une variable est distante à une entité de communication si elle appartient à un autre VCD.

---

[1] rapport sur la couche liaison de données [RLIDO]

La mise à jour d'une valeur de variable dans une AE peut être périodique ou apériodique. Elle peut être demandée par le producteur de la variable, par un consommateur de la variable ou par un tiers.

5.2.5.2. Les informations de validité

A chaque valeur d'une variable sont associées des informations portant sur la validité ("*date de fraîcheur*") de la valeur présente.

Les entités applications asynchrones élaborent les informations concernant la validité des valeurs indépendamment de l'environnement FIP. Elles travaillent isolées du reste du réseau sans avoir besoin de connaître des informations extérieures.

Les entités applications disposant d'un caractère synchrone élaborent les informations de validité lors de la réception des ordres de synchronisation du réseau, elles travaillent en fonction du réseau et ont toujours besoin d'avoir connaissance des informations se trouvant sur le réseau.

Quelques définitions sur les informations concernant *les informations de validité des valeurs* des variables FIP.

a) PROMPTITUDE

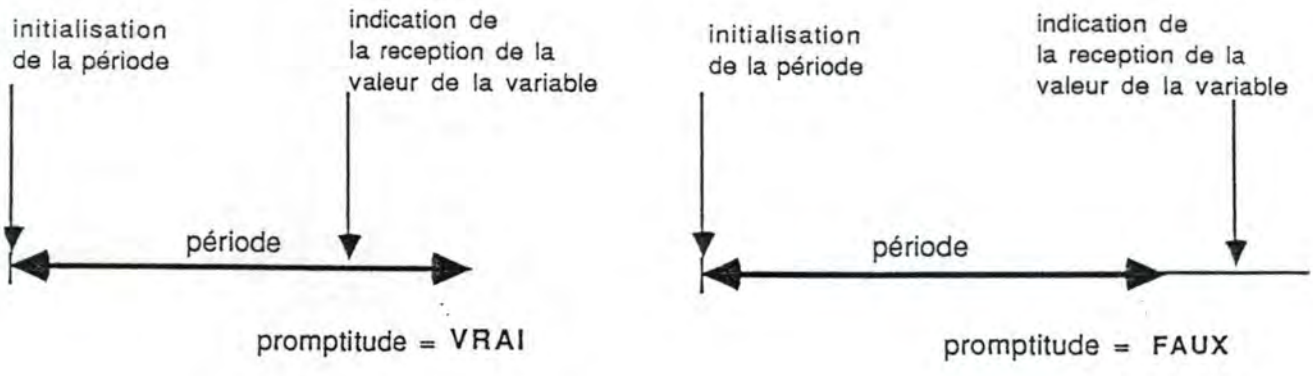


figure 5.5. Service d'élaboration du statut de promptitude d'une variable

La promptitude est une information de validité relative à la mise à disposition de l'utilisateur consommateur, par le réseau, de la valeur de la variable. Elle représente la fiabilité de transmission du bus dans un certain délai.

b) RAFRAICHISSEMENT

Le rafraîchissement est une information de validité relative à la validité de mise à disposition du réseau, par l'utilisateur producteur, de la valeur d'une variable. Elle représente la fiabilité de production du producteur dans un certain délai.

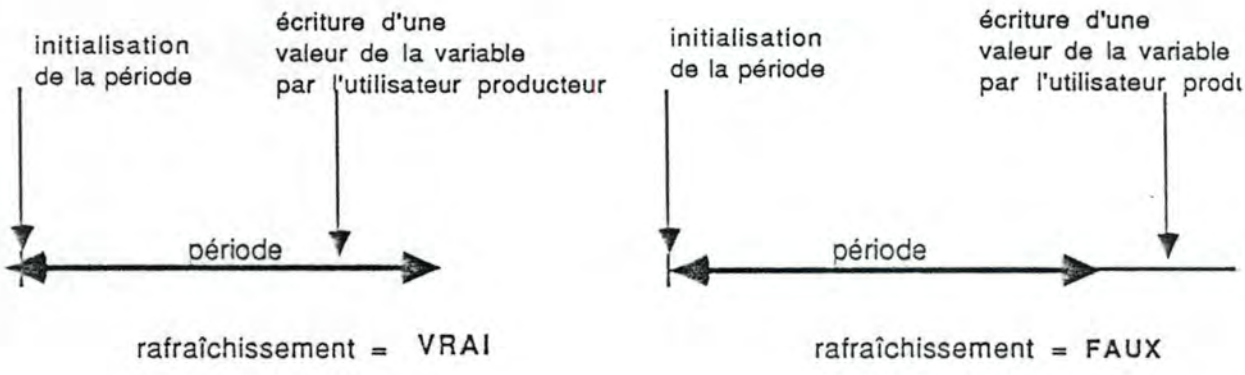


figure 5.6. Service d'élaboration du statut de rafraîchissement d'une variable

Les informations de validité ont leur période de mise à jour initialisée soit par une écriture ou une réception de valeur de variable (cas asynchrone), soit par réception variable de synchronisation (cas synchrone).

Une variable de synchronisation indique le début d'un nouveau cycle de la couche liaison de données et provoque la mise à jour des statut de rafraîchissement ou de promptitude concernés ainsi que l'armement des temporisateurs associés à ces statuts.

5.2.5.3. Le service de resynchronisation

Le service de resynchronisation permet à des Processus Application purement asynchrones (c'est-à-dire fonctionnant indépendamment du réseau) de participer à des applications réparties synchrones (c'est-à-dire dont l'exécution est associée à des indications issues du réseau).

1) resynchronisation chez un consommateur : lors de la réception d'une variable de resynchronisation (top de positionnement), on transfère la valeur et les statuts des promptitudes publiques dans la valeur et dans les promptitudes privées.

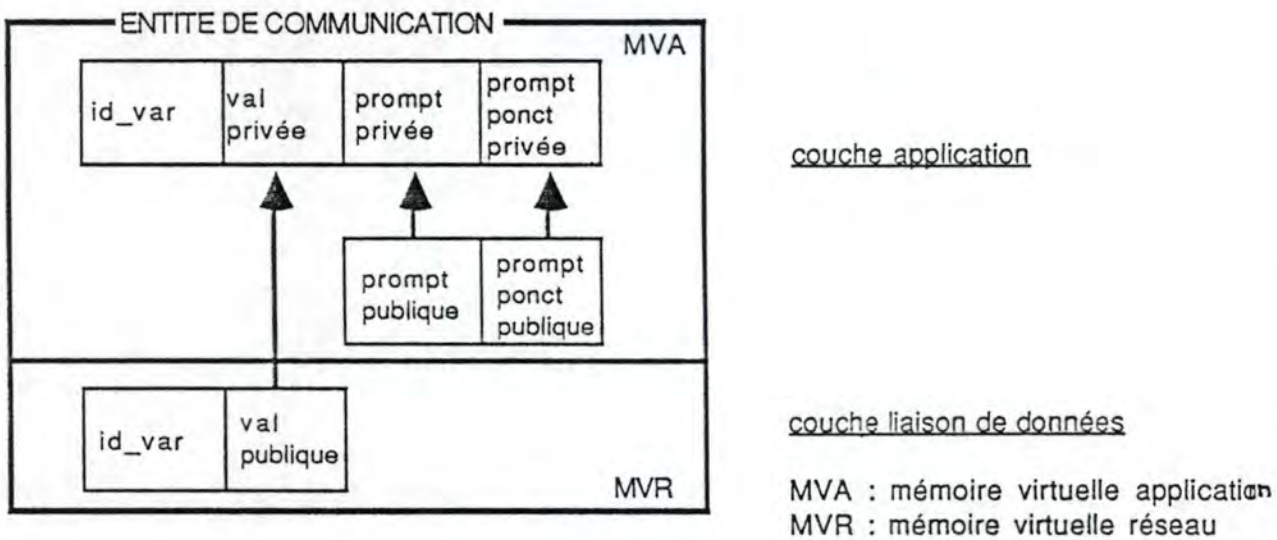


figure 5.7. Service de resynchronisation chez un consommateur.

2) resynchronisation chez un producteur : lors de la réception d'une variable de resynchronisation (top de photo), on transfère la valeur privée et les statuts de rafraîchissement privés des variables concernées dans la valeur et dans les statuts de rafraîchissement publics de ces variables.

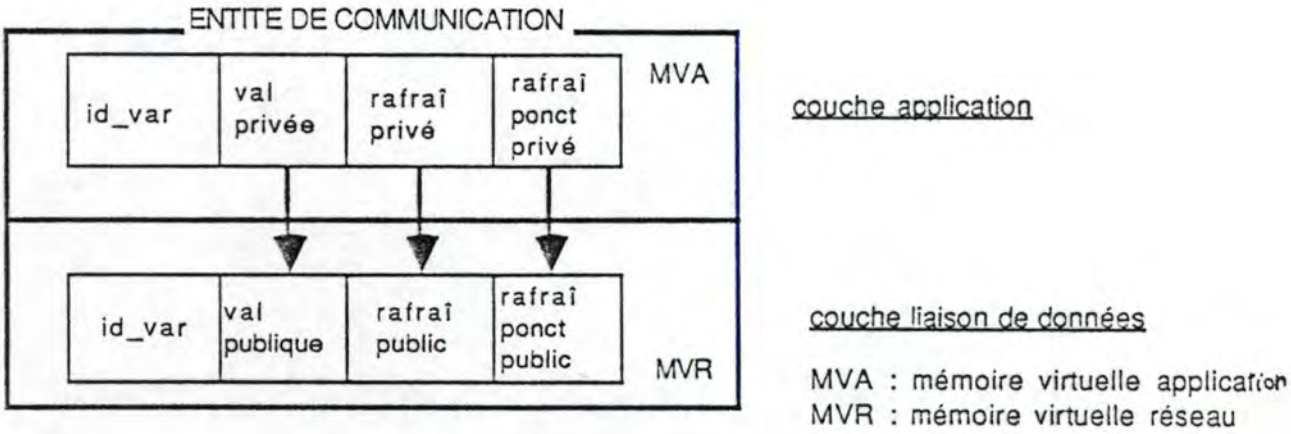


figure 5.8. Mécanisme de resynchronisation chez un producteur.

5.2.6. Les services de la couche application

Les services que la couche application doit rendre sont :

•A_READLOC_dem (A_NOM)
A_READLOC_conf (A_NOM,DATA)
par ce service l'utilisateur effectue une lecture de la variable A_NOM disponible dans l'entité de communication locale.

•A_READFAR_dem (A_NOM)
A_READFAR_conf (A_NOM,DATA)
par ce service l'utilisateur effectue une lecture de la variable A_NOM disponible dans une entité de communication distante.

•A_READ_dem (A_NOM)
A_READ_conf (A_NOM,DATA)
par ce service l'utilisateur effectue une lecture de la variable A_NOM disponible soit dans l'entité de communication locale, soit dans une entité de communication distante.

•A_READLIST_dem (A_LIST)
A_READLIST_conf (A_LIST,DATALIST)
par ce service l'utilisateur effectue une lecture des variables de la liste A_LIST disponibles dans l'entité de communication locale.

•A_WRITELOC_dem (A_NOM,VAL)
A_WRITELOC_conf (A_NOM,STATUS)
par ce service l'utilisateur effectue une écriture d'une valeur VAL dans la variable A_NOM disponible dans l'entité de communication locale.

•A_WRITEFAR_dem (A_NOM,VAL)
A_WRITEFAR_conf (A_NOM,STATUS)
par ce service l'utilisateur effectue une écriture d'une valeur VAL dans la variable A_NOM disponible dans une entité de communication lointaine.

•A_WRITE_dem (A_NOM,VAL)
A_WRITE_conf (A_NOM,STATUS)
par ce service l'utilisateur effectue une écriture d'une valeur VAL dans la variable A_NOM disponible soit dans l'entité de communication locale, soit dans une entité de communication lointaine.

•A_UPDATE_dem (A_NOM)
A_UPDATE_conf (A_NOM,STATUS)
par ce service l'utilisateur déclenche la mise à jour des variables A_NOM chez les entités consommatrices de A_NOM, à partir de la valeur de cette variable disponible localement dans l'entité de communication productrice.

•A_RECEIVED_ind (A_NOM)
ce service indique à l'utilisateur la mise à disposition d'une variable par le réseau.

•A_SENT_ind (A_NOM)
ce service indique à l'utilisateur qu'une variable a été émise sur le réseau.

5.3. PIPN

La modélisation de la couche application du protocole FIP a été réalisée sur le logiciel PIPN. Ce logiciel constitue un outil offrant à l'utilisateur un environnement de prototypage de modèles hiérarchisés écrits sous la forme de réseaux de Petri à prédicats. Ce langage PIPN a été décrit précédemment dans le chapitre 4.

Rappelons que PIPN signifie Prolog Interpreted Petri Net, et que ce langage est basé sur les réseaux de Petri à prédicats labellés. Les réseaux de prédicats sont basés sur la logique du premier ordre. Ce langage est particulièrement adapté à la modélisation des systèmes communicants.

Le protocole FIP étant construit sur la base du modèle OSI de l'ISO et le langage PIPN étant bien adapté à la modélisation de telles architectures à couches, l'emploi de ce langage pour modéliser la couche application de FIP est justifié.

En effet l'architecture modulaire de PIPN fait de ce langage un langage adapté à la modélisation en couche des protocoles de communication. De plus le langage donne aux modules de niveaux supérieurs l'impression que les modules de niveaux inférieurs sont des boîtes noires qui rendent un service. L'architecture PIPN, par ces caractéristiques, correspond avec l'architecture en couche des protocoles, où une couche de niveau supérieur utilise les services des couches inférieures sans savoir comment ils ont été implémentés.

Et le langage PIPN permet à l'utilisateur de modéliser des processus possédant une structure complexe. Chaque transition écrite dans ce langage est un modèle d'événement local et peut être caractérisée pour représenter un événement complexe.

5.4. LES MODELES

5.4.1. Introduction

Les pages qui suivent sont un exposé de la configuration choisie, des grandes lignes de la structure de la "base de données", de la manipulation des éléments de la "base de données" ainsi que des hypothèses émises pour faciliter la construction des modèles.

5.4.2. Configuration

La recherche de la bonne configuration du modèle ne s'est pas réalisée du premier coup, il a fallu plusieurs itérations pour trouver la configuration adéquate.

Dans cette recherche, nous n'avons pas respecté la modélisation classique présentée dans le chapitre 3. Cette représentation classique modélise les entités de la couche application communiquant par l'intermédiaire de la couche liaison. Ici nous n'avons pas pris en compte la couche liaison qui avait déjà été modélisée, testée et vérifiée précédemment.

Cette couche liaison modélisée par K. Nordgard ne prend pas en compte les erreurs pouvant survenir dans la couche liaison. L'intégration de ce modèle dans celui de la couche application aurait donc nécessité une refonte et une modification des notations du modèle déjà réalisé. C'est pourquoi les modèles que nous avons développés n'ont pas tenu compte de cette modélisation précédente.

5.4.2.1. Première configuration

La première configuration envisagée fut celle reprenant la couche liaison de données à laquelle deux stations étaient raccordées. Chacune des stations rendant à son utilisateur tous les services de la couche application.

figure 5.7 Configuration 1

Une telle configuration, très proche de la réalité, s'est avérée impossible à modéliser car le nombre de transitions (> 100) est trop élevé pour permettre d'être rentré en mémoire. De plus l'automate d'un tel système aurait été gigantesque (plusieurs centaines de places et quelques milliers d'arcs) et donc incompréhensible.

5.4.2.2. Deuxième configuration

La deuxième configuration étudiée ne reprenait plus qu'une station reliée à la couche application. Cette station rendait à son utilisateur l'ensemble des services de la couche application.

Mais l'ensemble des services de la couche application représente 19 sous-modules, chacun étant composé de plusieurs transitions. Ce qui donnait un nombre de transitions encore trop élevé. L'automate obtenu, quoique plus petit, restait encore inabordable.

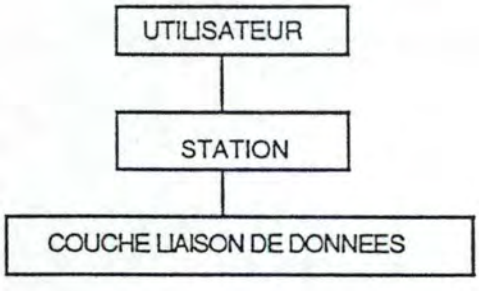


figure 5.8. Configuration 2

5.4.2.3. Troisième configuration

La configuration finalement envisagée est celle d'une station décomposée en deux parties, une partie relative au producteur et une autre relative au consommateur. Le producteur et le consommateur étant aussi divisés en services.

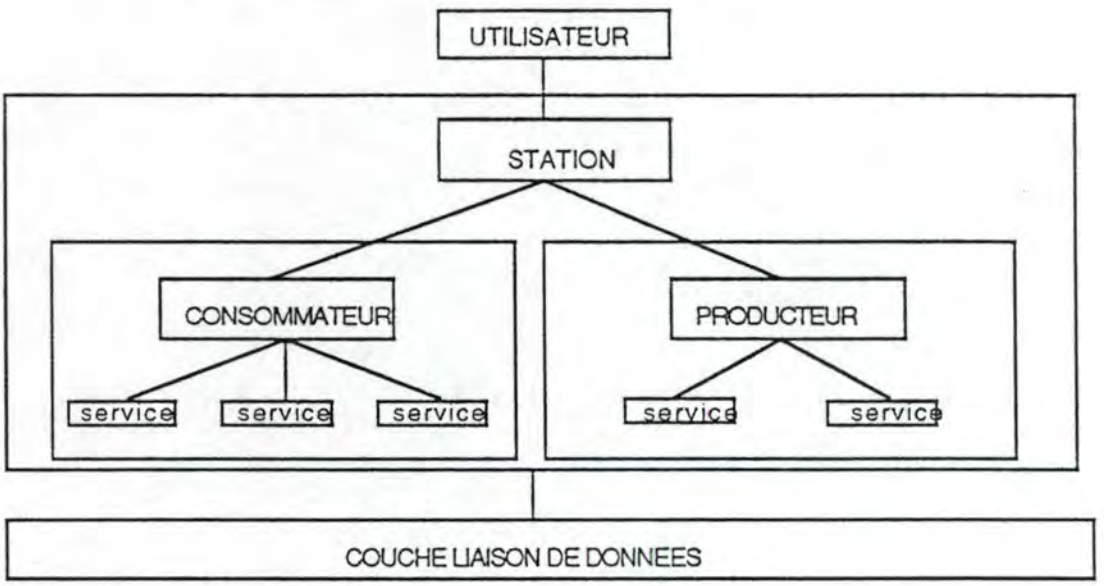


figure 5.9. Configuration 3

Cette structure "consommateur-service", "producteur-service" a l'avantage de pouvoir être facilement simulée car le nombre de transitions y est très petit (<10). Ce qui nous permet d'avoir des automates facilement analysables.

De plus, changer le service associé à un producteur ou à un consommateur est très simple. En effet, il suffit d'indiquer la référence au nom du service que l'on veut simuler dans le prédicat "child" et d'initialiser le prédicat "init" du producteur ou du consommateur (et éventuellement une partie des "tableaux fixes" du fichier "cond").

Notons que nous n'avons pas modélisé la couche liaison de données, mais que nous avons émis l'hypothèse qu'elle rend le service (même de façon erronée) qu'on attend d'elle.

5.4.2.4. Le consommateur

Le consommateur offre les services suivants qui sont représentés dans le modèle par le nom en italique :

- service de lecture d'une variable de type local, A_READLOC : *readloc*
- service de lecture d'une variable de type lointain, A_READFAR : *readfar*
- service de lecture d'une variable, A_READUNIV : *readuniv*
- service de lecture d'une liste, A_READLIST : *readlist*
- service de mise à jour d'une variable, A_UPDATE : *update*
- service de réception ,A_RECEIVED, d'une variable de classe normale : *recnorm*
- service de réception ,A_RECEIVED, d'une variable de classe synchronisation : *recsync*
- service de réception ,A_RECEIVED, d'un ordre de resynchronisation (top de positionnement) : *recresync*
- service de réception ,A_RECEIVED, d'une variable de classe normale appartenant à une liste : *recnormlist*
- service de réception ,A_RECEIVED, d'une variable de classe cohérence : *reccoh*
- service de déclenchement d'une alarme associée à une promptitude : *alarmprompt*
- service de déclenchement d'une alarme associée à une liste de reprise : *alarmrepr*

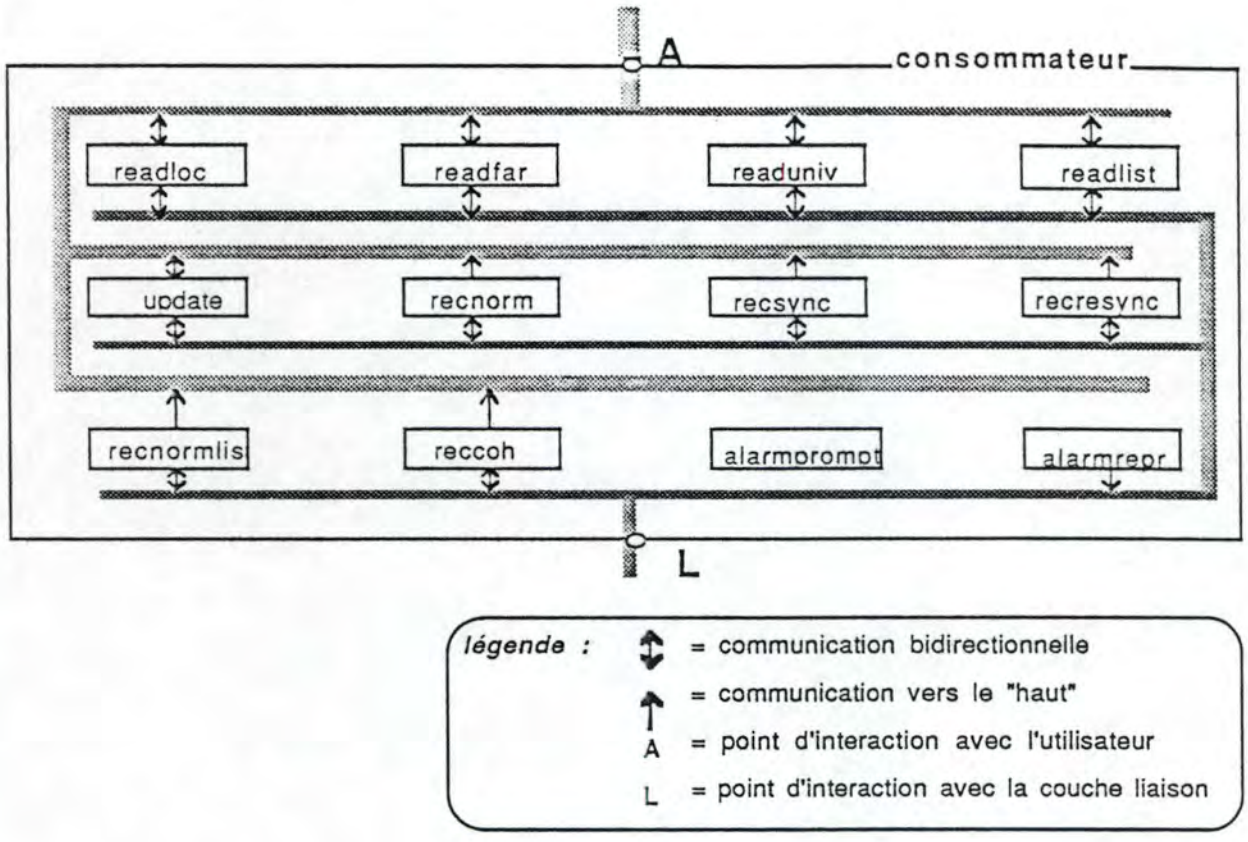


figure 5.10 L'architecture du modèle consommateur

5.4.2.5. Le producteur

Le producteur offre les services suivants qui sont représentés dans le modèle par le nom en italique :

- service d'écriture de la valeur d'une variable de type local WRITELOC : *writeloc*
- service d'écriture de la valeur d'une variable de type lointain WRITEFAR : *writefar*
- service d'écriture de la valeur d'une variable WRITEUNIV :*writeuniv*
- service de réception, A_RECEIVED, d'une variable de classe synchronisation : *syncprod*
- service de réception, A_RECEIVED, d'un ordre de resynchronisation (top de photo) : *top_photo*
- service de réception, A_SENT, d'une indication d'émission : *recsent*

- service de déclenchement d'une alarme associée à un rafraîchissement : *alarmrafr*

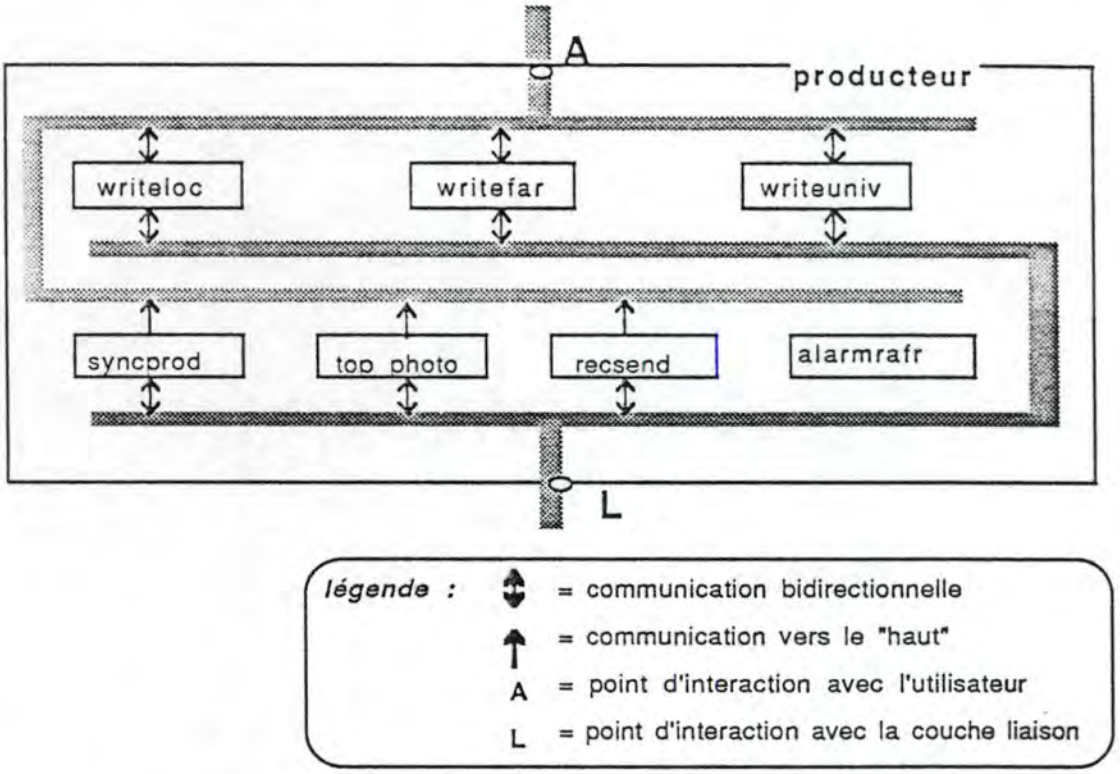


figure 5.11 L'architecture du modèle producteur

5.4.3. La base de données

Une des caractéristiques principales de FIP est sa base de données répartie. Cette base de données contient les informations nécessaires au fonctionnement de la couche application. Ces informations donnent des renseignements sur l'état de FIP.

5.4.3.1. Les places globales

Pour modéliser la base de données répartie, on a créé des places qui contiennent la "base de données" du consommateur ou du producteur. Ces places sont partagées par l'ensemble des modules (services) du consommateur ou du producteur afin qu'ils manipulent les mêmes données. Chaque service possède en plus des places globales des places locales qui lui sont propres.

Les informations de cette base de données sont accessibles à l'aide de fonctions Prolog, écrites dans le fichier de conditions, qui permettent la consultation et la mise à jour des informations de la base de données. Ces primitives sont construites de telle façon qu'elles cachent la structure du modèle de la base de données afin qu'une modification de la structure de la base de données, tel que l'ajout d'un champ, n'ait que des effets au niveau de ces primitives élémentaires d'accès aux informations.

L'écriture des fonctions sous cette forme s'est avérée fort utile lors de la conception de nos modèles pendant laquelle la structure de la base de données a subi de nombreuses modifications.

Grâce à la transparence de la structure de la base de données, une modification de celle-ci n'entraîne aucune modification au niveau des fonctions du fichier de conditions, des transitions des modules et des prédicats utilisant les primitives élémentaires d'accès à la base de données. Et l'utilisation de ces primitives d'accès rend les fonctions des fichiers conditions très facilement compréhensibles et d'une grande lisibilité.

exemple de place globale:
La place "list_var_cons(I)" contient toutes les variables connues en consommation du consommateur. Chaque module du consommateur qui manipule les variables consommées importe cette place. Il peut alors consulter et modifier cette place.

La consultation et la modification des places globales s'effectuent dans l'environnement Prolog. On a réalisé pour cela quatre types de fonction qui manipulent les places globales : des fonctions de construction, de consultation, de modification et de sauvetage des éléments des places globales.

5.4.3.2. Les places globales du consommateur

a) **list_var_cons(L)** : la liste des variables consommables

Cette place représente l'ensemble des variables consommables par le consommateur et les renseignements sur celles-ci.

b) **list_prompt(L)** : la liste des promptitudes

Cette place représente l'ensemble des promptitudes associées aux variables consommables par le consommateur et les renseignements sur ces promptitudes.

c) **list_l_repr(L)** : la liste des listes de reprise

Cette place représente l'ensemble des listes de reprises associées aux listes consommables par un consommateur. Une liste de reprise est une liste d'identificateurs de variables pour la couche liaison, qui représente les valeurs non encore reçues des éléments d'une liste dont on fait la lecture.

d) **list_var_coh(L)** : la liste des variables de cohérence

Cette place représente l'ensemble des variables de cohérence associées aux listes consommables par un consommateur. Une variable de cohérence est une variable, associée à une liste, qui indique quel est l'état de la cohérence de cette liste, en fonction des "*dates de fraîcheur*" des variables de la liste.

e) **idle_cons** : l'état de repos du consommateur

Cette place représente l'état de repos du consommateur; elle est consommée par chaque module au début du traitement et n'est restituée qu'à la fin du traitement. Ceci afin de rendre les services atomiques, ce qui n'est pas nécessairement un choix très réaliste. En effet, ceci signifie , par exemple, que pendant le traitement d'une lecture locale "A_READLOC", la station ne peut être appelée par aucun autre service, par exemple recevoir un

"RECEIVED_IND" de la couche liaison de données. On a tout de même opté pour le blocage des autres services pendant un traitement d'un service afin de limiter l'explosion combinatoire des automates qui rendrait leur compréhension très ardue (cfr 5.4.2).

f) **wait_update_free_conf (N)** : le nombre de confirmations en attente

Cette place représente le nombre de confirmations de mise à jour en attente pour le consommateur. Cette place est modifiée; par la demande d'une mise à jour, elle augmente de 1 et par la confirmation d'une demande de mise à jour, elle décroît de 1. Cette place est bornée par le nombre maximum de confirmations en attente qui est défini à la configuration.

g) **liste_liste (L)** : la liste des listes du consommateur

Cette place représente l'ensemble des listes consommables par le consommateur et les renseignements sur ces listes.

5.4.3.3. Les places globales du producteur

a) **list_var_prod(L)**: la liste des variables productibles
Cette place représente l'ensemble des variables productibles par le producteur et les renseignements sur ces variables.

b) **list_rafr(L)** : la liste des rafraîchissements
Cette place représente l'ensemble des rafraîchissements associés aux variables productibles par le producteur et les renseignements sur ces rafraîchissements.

c) **idle_prod** : l'état de repos du producteur
Cette place représente l'état de repos du producteur, elle est consommée par chaque module au début du traitement et n'est restituée qu'en fin de traitement. Ceci afin de rendre les services atomiques pour les mêmes raisons que chez le consommateur.

5.5. DEMARCHE D'OBTENTION DES RESULTATS

Nous allons développer un service pour montrer comment nous avons obtenu les résultats de notre travail. Il faut noter que le rapport que nous avons réalisé au LAAS comprend une seconde partie exposant tous les résultats.

Le service développé est le service de lecture locale A_READLOC. Ce service est particulièrement simple à modéliser et c'est pourquoi il a été choisi pour servir de support à la méthodologie proposée. Il ne faudrait pas en conclure que la norme FIP est aussi simple. Il nous a fallu pour arriver à un premier résultat beaucoup d'efforts car la lecture du document contenant la norme est loin d'être évidente.

La norme que nous avons analysée possède certaines incohérences ou manques de clarté, notamment en ce qui concerne l'évolution de certains éléments de la base de données. Elle ne possède pas non plus de schéma global de la base de données mais seulement des définitions dans un langage naturel, ce qui laisse parfois le lecteur perplexe. De plus, la norme a été modifiée alors que nous avions déjà réalisé une première version de notre modèle, ce qui nous a amené a réviser notre travail.

5.5.1. Le réseau d'évaluation

Nous allons décrire comment nous sommes passés des spécifications de la norme [FIPAPP88] aux transitions PIPN. Nous nous sommes basés premièrement sur le réseau d'évaluation qui est rappelé ci-après.

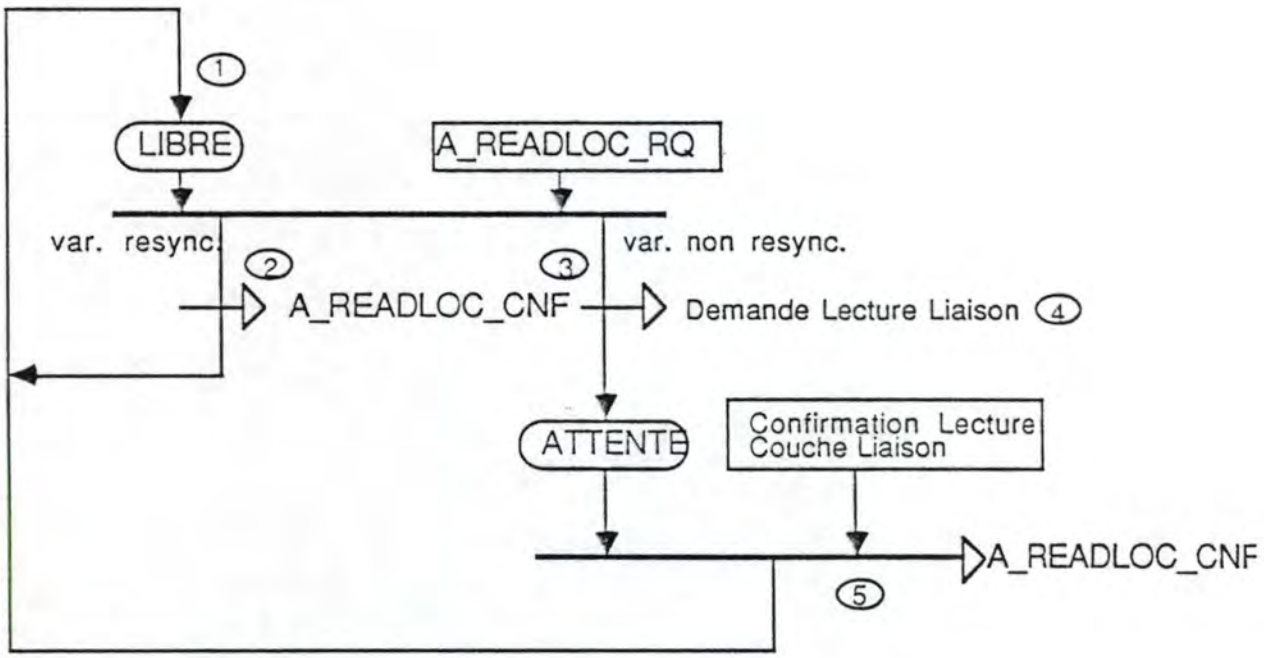


figure 5.12. Réseau d'évaluation du service A_READLOC

5.5.2. Les mécanismes

De la figure 5.12. du réseau d'évaluation du service A_READLOC, nous avons déduit trois mécanismes différents, associés à la lecture locale.

Les mécanismes ne tiennent pas compte de tous les éléments spécifiés dans la norme. Ainsi aucun code d'erreur n'est spécifié, car la correspondance entre les erreurs de la couche liaison et la couche application ne nous semble pas évidente.

Le mécanisme d'accès aux champs des variables structurées n'est pas décrit. Nous nous sommes limités à l'accès de la variable en entier.

1) Un premier mécanisme dans le cas d'une lecture d'une **variable resynchronisée** (2 sur le figure 5.12. du réseau d'évaluation de A_READLOC).

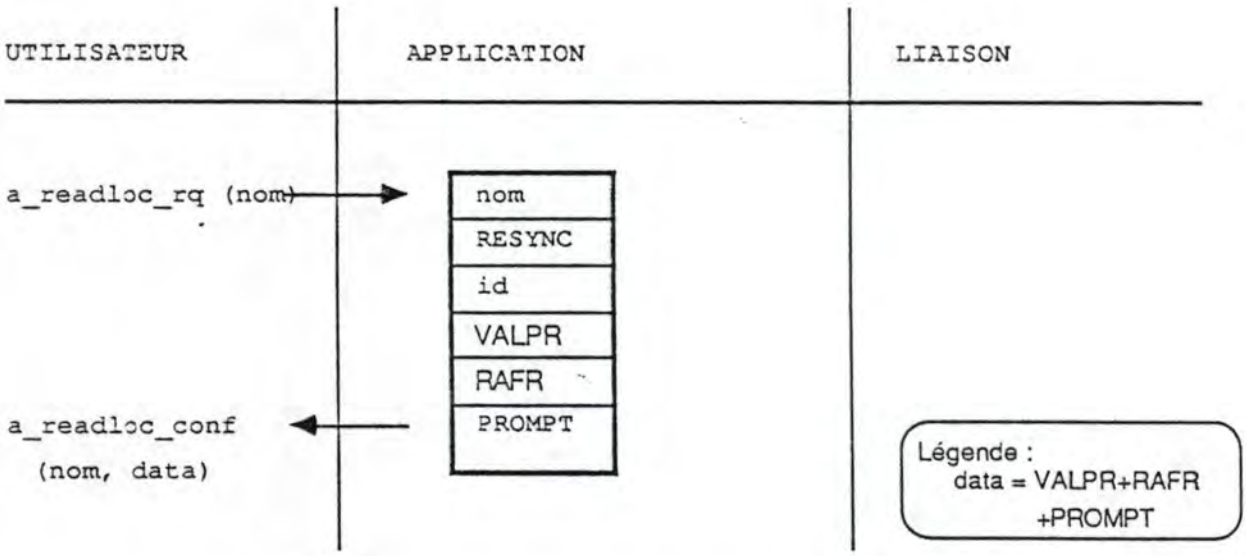


figure 5.13. Mécanisme de lecture locale pour une variable resynchronisée

Dans le cas d'une variable resynchronisée, on transmet directement la confirmation de lecture avec les données demandées (data) sans faire de lecture sur la couche liaison. Il n'y a qu'une étape nécessaire.

2) Un second mécanisme dans le cas d'une lecture locale de **variable non resynchronisée** (3 sur le figure 5.12. du réseau d'évaluation).

Ce mécanisme nécessite deux étapes: une de demande de lecture et une de confirmation de lecture.
Dans le cas d'une variable non resynchronisée, on effectue une demande de lecture sur la couche liaison afin de recevoir la valeur publique et les rafraîchissements de la variable (4 sur le figure 5.12. du réseau d'évaluation).
On reçoit alors la confirmation de lecture de la couche liaison avec la valeur publique et les rafraîchissements. Et on transmet les données (data) à l'utilisateur (5 sur le figure 5.12. du réseau d'évaluation).

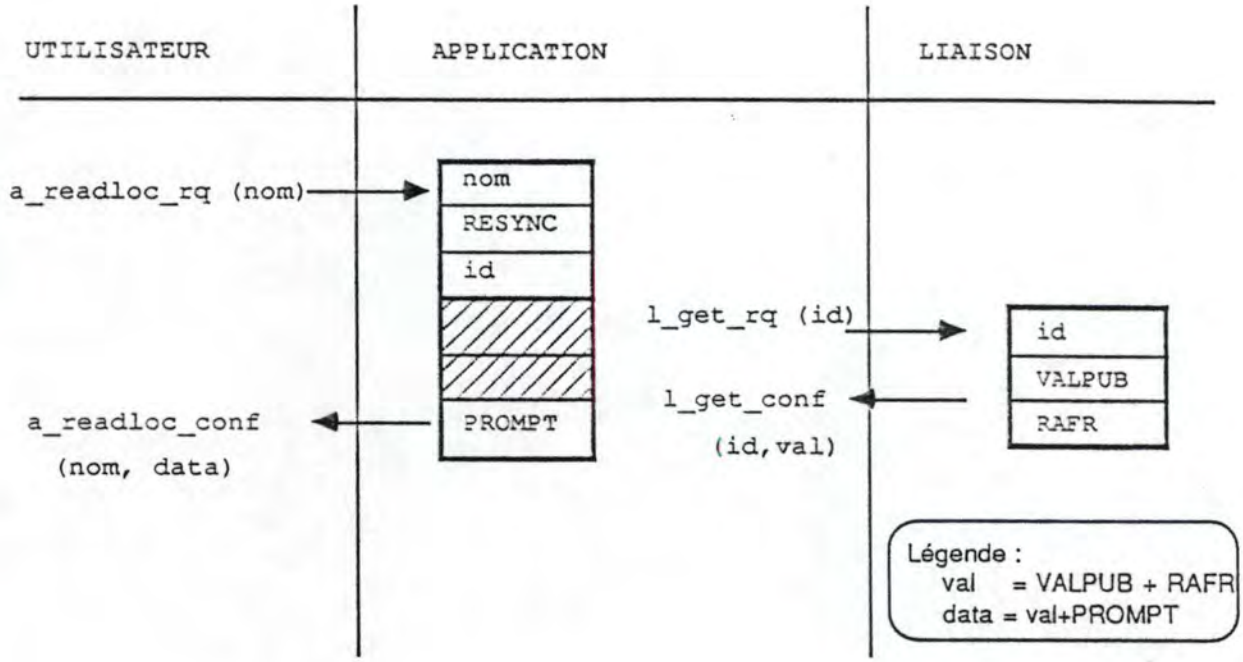


figure 5.14. Mécanisme de lecture locale pour une variable non resynchronisée

3) Un troisième mécanisme implicite, en cas d'une demande de lecture locale d'une variable **non consommable.**

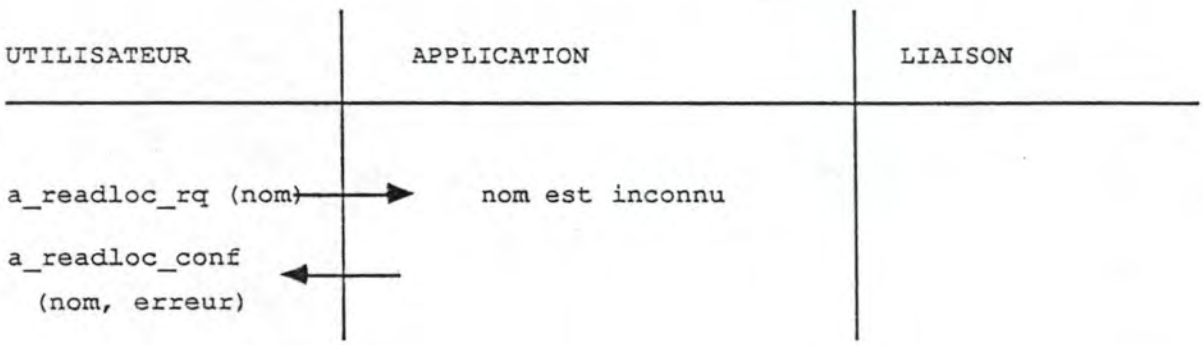


figure 5.15. Mécanisme de lecture locale pour une non consommable

5.5.3. Les places importées

On importe à partir du module père "cons" les places :

- **idle_cons** : état de repos du module "cons". Cette place se déduit de l'état "LIBRE" du réseau d'évaluation (figure 5.12.) .

- **list_var_cons(_l), list_prompt(_l)**: les informations nécessaires de la base de données pour ce service, à savoir l'ensemble des variables consommées par la station et l'ensemble des promptitudes élaborées pour ces variables.

5.5.4. Les places locales

- **test_valid_rl_rq(_a_nom)** : mémorise le nom de la variable sur laquelle porte la demande. Cette place s'est avérée nécessaire pour recevoir la confirmation en erreur de l'application dans le cas où la demande porte sur un nom d'une variable non locale.

- **wait_get_conf(_id_var)** : mémorise l'identificateur de la variable de laquelle on attend la confirmation suite à un l_get_dem. Cette place est déduite de l'état "ATTENTE" du réseau d'évaluation.

5.5.5. Les transitions

Nous allons maintenant expliciter les différentes transitions. En annexe, nous donnons les listings des fichiers: cons.header.pro (le module père), readloc.header.pro (l'en-tête du service readloc pour le consommateur), readloc.body.pro (le corps du service readloc qui contient les transitions et les places locales) et readloc.cond.pro (fichier des conditions du service readloc). Ces fichiers contiennent toute l'information nécessaire à la modélisation du service en question.

5.5.5.1. Transition 1 : demande de lecture locale

objectif :

Réception d'une demande de lecture locale effectuée par l'utilisateur portant sur la variable ayant le nom *"_a_nom"*.

condition :

Ce nom est connu par le consommateur. Ceci est vérifié à l'aide de la fonction *"rech_nom_acceptee(_a_nom)"* qui s'instancie à tous les noms connus par l'utilisateur.

place :

On consomme la place globale *"idle_cons"* et on produit la place *"test_valid_rq(_a_nom)"*.

réseaux d'évaluation :

Cette transition est une étape intermédiaire entre le point 1 et les points 2 et 3 du réseau d'évaluation 5.12..

5.5.5.2. Transition 2 : demande rejetée pour une variable inconnue

objectif :

Rejet de la demande parce que le nom *"_a_nom"* ne correspond pas à une variable consommable dans l'entité de communication locale (mécanisme 3).

condition :

Cette constatation est faite - après la construction de la base de données Prolog à l'aide de la fonction *assertion_var_cons(_L)* où *"_L"* est la liste de toutes les variables consommables de la station - par le prédicat *"verif_accept_readloc(_a_nom)"* qui ne réussit pas, c'est-à-dire ne trouve aucune variable ayant le nom *"_a_nom"* et qui est de type "LOCAL".

place :

On envoie, par conséquent, une confirmation négative à l'utilisateur *"a_readloc_conf(_a_nom,erreur)"* et on retourne à l'état de repos *"idle_cons"*.

réseau d'évaluation :

Cette transition n'est pas déductible du réseau d'évaluation (figure 5.12.) , mais correspond au mécanisme 3 (figure 5.4).

5.5.5.3. Transition 3 : demande acceptée pour une variable non resynchronisée

objectif :

Acceptation de la demande portant sur un nom d'une variable de type "local" qui est non resynchronisée (mécanisme 2, étape 1).

condition :

Ceci est vérifié à l'aide de la fonction *verif_accept_readloc(_a_nom)* et de la fonction

rech_resync_var_cons(_id_var,NONRESYNC) [1] où *"_id_var"* est l'identificateur de la variable de nom *"_a_nom"*.

place :

Comme la variable est non resynchronisée , il faut envoyer une demande d'obtention de la valeur publique *"l_get_dem(_id_var)"* et se mettre en état d'attente de la confirmation *"wait_get_conf (_id_var)"*.

réseau d'évaluation :

Cette transition correspond au point 3 du réseau d'évaluation.

5.5.5.4. Transition 4 : demande acceptée pour une variable resynchronisée

objectif :

Acceptation de la demande portant sur un nom d'une variable de type "local" qui est resynchronisée et transmission de la confirmation à l'utilisateur (mécanisme 1).

condition :

Ceci est vérifié à l'aide de la fonction verif_accept_readloc(_a_nom) et de la fonction rech_resync_var_cons(_id_var,RESYNC) où *"_id_var"* est l'identificateur de la variable de nom *"_a_nom"*. Il suffit donc de rechercher dans la base de données la valeur privée, les statuts des rafraîchissements et les statuts des promptitudes

place :

On restitue la place *"idle_cons"*.

réseau d'évaluation :

Cette transition correspond au point 2 du réseau d'évaluation (figure 5.12.).

---

[1] Note : Une des caractéristiques des fonctions Prolog est qu'on peut les utiliser de plusieurs manières. Ainsi, il est possible d'utiliser la fonction rech_resync_var_cons(_id_var,NONRESYNC) aussi bien pour vérifier si la variable *"_id_var"* est non resynchronisée (en instanciant le deuxième paramètre à la constante NONRESYNC) que pour rechercher si la variable est resynchronisée ou non (en laissant le deuxième paramètre libre), etc..

5.5.5.5. Transition 5 : réception de la confirmation de lecture locale

objectif :

Réception de la confirmation de lecture locale et transmission de la confirmation à l'utilisateur, cette transition est la suite du traitement initié par la transition 3 (suite du mécanisme 2, étape 2).

condition :

Ceci est vérifié à l'aide de la fonction Rech_prompt qui recherche les valeurs des promptitudes dans la base de données.

place :

On obtient en input du point d'interconnexion, la valeur publique et les statuts de rafraîchissements de la couche liaison de données. On envoie ces valeurs à l'utilisateur.

On restitue la place *"idle_cons"*.

réseau d'évaluation :

Cette transition correspond au point 4 du réseau d'évaluation (figure 5.12.).

5.5.5.6. Transition 6 : réception de l'erreur de lecture locale

objectif :

Réception de l'erreur de lecture sur la couche liaison et transmission de la confirmation avec erreur à l'utilisateur. Cette transition fait également suite au traitement initié par la transition 3 (mécanisme 2,étape 2).

place :

Le *"l_get_rq"* a échoué et on reçoit une confirmation en erreur de la couche liaison (input *"l_get_conf (erreur)"* . Après la confirmation en erreur à l'utilisateur (output *"a_readloc_conf(_a_nom,erreur)"*), on restitue la place *"idle_cons"*.

réseau d'évaluation :

Dans le réseau d'évaluation (figure 5.12.) on ne tient pas compte d'un éventuel échec du "L_GET". Cette transition n'est donc pas reprise dans la figure.

5.5.6. Les automates

5.5.6.1. Introduction

L'outil PIPN permet de réaliser l'automate de transitions. Ci-après, on va retrouver les automates du service A_READLOC obtenus à partir des 6 transitions décrites précédemment.

Trois automates sont décrits, un pour chaque mécanisme:
- l'automate de la lecture locale d'une variable non resynchronisée (mécanisme 2)
- l'automate de la lecture locale d'une variable resynchronisée (mécanisme 2)
- l'automate de la lecture locale d'une variable inconnue (mécanisme 3).

5.5.6.2. Légende des trois automates de lecture locale d'une variable

a) légende sur les paramètres

V	= identificateur de la variable pour l'utilisateur
ID	= identificateur de la variable pour la couche liaison
VAL	= valeur de la variable + rafraîchissements
DATA	= valeur de la variable + rafraîchissements + promptitudes

b) légende sur les transitions

?<mess> = réception du message <mess>
!<mess> = émission du message <mess>
***<mess>** = réception interne du message <mess>
a_mess = message à destination ou en provenance de l'utilisateur
l_mess = message à destination ou en provenance de la couche liaison de données

5.5.6.3. L'automate du service A_READLOC pour une variable non resynchronisée

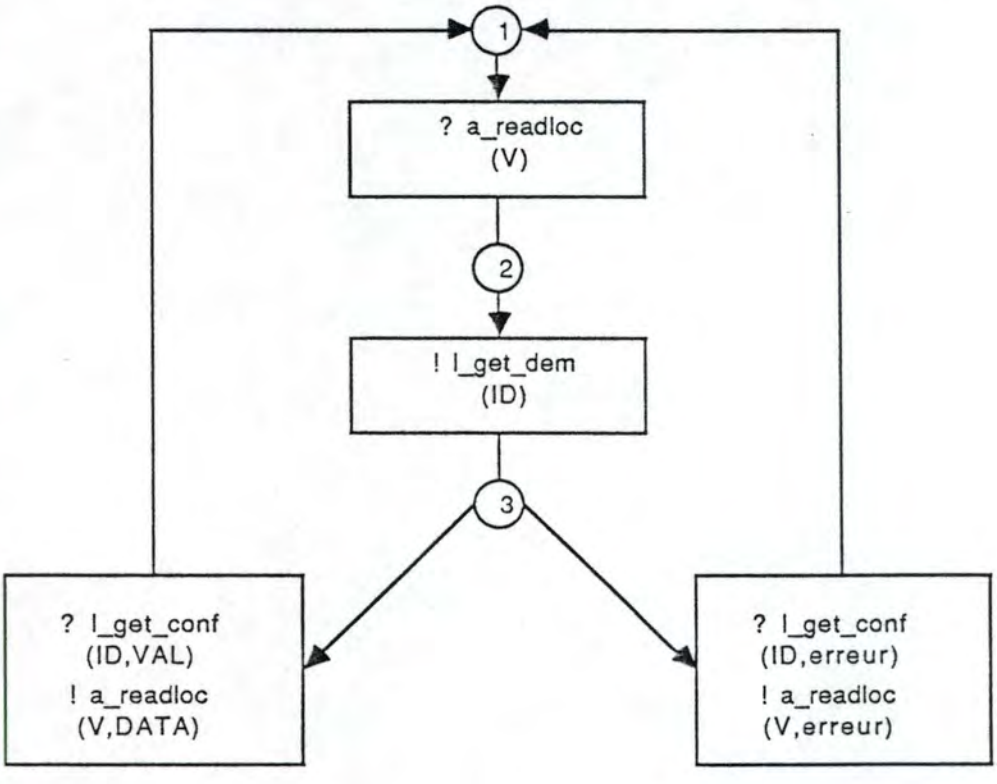


figure 5.16. L'automate du service A_READLOC pour une variable non resynchronisée

L'automate du service de lecture locale pour une variable locale non resynchronisée est déclenché par la demande de lecture "? a_readloc", ce qui lui fait demander à la couche liaison la valeur par la demande "!l_get_dem". La réponse de la couche liaison peut

être double : soit la couche liaison donne la valeur , soit la couche liaison indique une erreur, ce qui crée deux alternatives dans l'automate. Selon la réponse de la couche liaison, on donne à l'utilisateur une réponse différente puis on retourne dans l'état de départ.

5.5.6.4. L'automate du service A_READLOC pour une variable resynchronisée

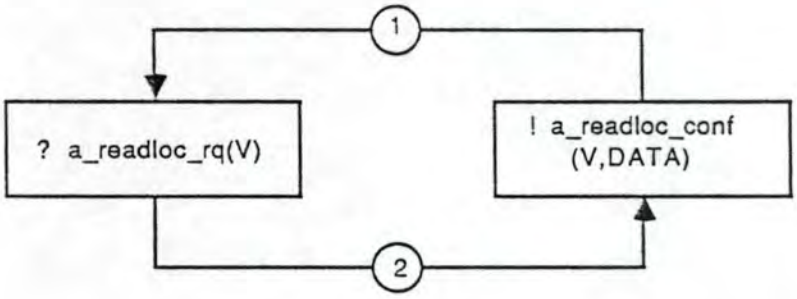


figure 5.17. L'automate du service A_READLOC pour une variable resynchronisée

L'automate du service de lecture locale pour une variable resynchronisée est fort petit. Il est déclenché par la demande de lecture "? a_readloc_rq", et comme la couche liaison possède la valeur de cette variable (de par son caractère resynchronisé), il ne faut faire aucune lecture sur la couche liaison. On transmet directement la valeur connue dans la base de données de la couche application. Il ne peut donc y avoir qu'une réponse possible, car aucune erreur ne peut intervenir.

5.5.6.5. L'automate du service A_READLOC pour une variable non consommable

L'automate du service de lecture locale pour une variable inconnue en consommation est fort court. Il est déclenché par une demande de lecture "? a_readloc_rq" sur une variable inconnue (dont il n'existe pas de référence dans la base de données de la couche application pour ce consommateur). La réponse est unique, c'est l'erreur de la lecture locale.

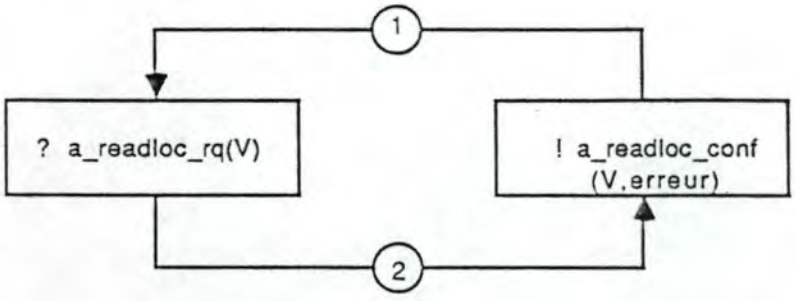


figure 5.18. de l'automate du service A_READLOC pour une variable inconnue en consommation

5.5.6.6. Conclusion sur les automates

Les trois automates que nous avons obtenus sont réinitialisables, c'est-à-dire que l'on peut toujours revenir à l'état de départ. Cela signifie que le service de lecture locale se termine toujours et qu'il n'y a pas de blocages.

Par opposition, s'il n'était pas réinitialisable, cela voudrait dire que le service de lecture locale ne se termine jamais. En effet l'état initial est l'état de repos qui est consommé par tous les services lors de leur première transition. Si on ne revenait jamais dans cet état, alors cela voudrait dire que l'on reste dans le service de lecture locale sans en sortir (bouclage ou état puits).

S'il y avait eu un état puits, alors on aurait pu dire que le service de lecture locale possède un blocage. C'est-à-dire qu'il existe une suite d'événements qui amène le service de lecture locale dans un état tel qu'il ne peut plus rien faire.

5.6. CONCLUSION

L'utilisation de PIPN nous a permis de modéliser la couche application de FIP ce qui montre l'adéquation d'utiliser cet outil pour modéliser des protocoles de télécommunication. On a donc pu recommander PIPN (LAAS-VERILOG) pour la modélisation et la vérification de protocoles.

La taille du modèle d'une station ne nous autorisant pas à la modéliser dans son entièreté, ce que nous avons pu vérifier se limite à des services isolés ou à des regroupements limités de services. Il ne nous est donc pas permis de dire que la couche application ne possède pas de blocage dans son ensemble, mais par contre nous pouvons dire que notre modélisation ne fait apparaître aucun blocage au niveau des services isolés ou des regroupements limités.

Mais à ce problème, on peut apporter deux types de réponses : une réponse théorique, qui utilisant les théories de la communication des systèmes répartis (Milner CCS, Hoare CSP), permettra de vérifier la couche dans son entièreté, et une réponse pratique qui serait de lier FIP à une station de travail qui permettrait de modéliser le protocole dans son intégralité.

D'autre part, la modélisation et la vérification des modèles nous a amené à nous poser certaines questions sur la norme. Ces questions nous ont permis de découvrir certaines anomalies et oublis de la norme. Nos observations ont donné lieu à un rapport de recherche [HASTIR88] dans lequel on décrit d'une manière systèmatique le protocole de service (ses fonctions, ses mécanismes, les hypothèses de construction du modèle, les événements pris en compte par l'utilisateur et les événements pris en compte par la couche liaison). On y trouve aussi les automates se rapportant à ce protocole. Le rapport se termine par des remarques sur les services :ambiguïté, silence, contradiction; ainsi que par les observations d'anomalies et des propositions d'améliorations.

CONCLUSIONS GÉNÉRALES

L'objectif de ce mémoire a été la mise en évidence des qualités et des défauts des réseaux de Petri. Nous y avons exposé quelques applications caractéristiques des réseaux de Petri en approfondissant leur utilisation dans la conception et la vérification des protocoles de communication.

Notre contribution pratique à ce domaine fut notamment la modélisation et la vérification de la couche Application du protocole FIP pendant notre stage au LAAS de Toulouse.

Lors de la rédaction de ce mémoire, nous avons tenté d'établir une vue synthétique -illustrée par des exemples- sur les applications actuelles des réseaux de Petri.

Notre expérience des réseaux de Petri, nous a permis de dégager leurs avantages et certains de leurs défauts.

Parmi les avantages, on citera :

- la représentation aisée du parallélisme et de la concurrence
- la clarté de la représentation graphique,
- la facilité d'apprentissage du formalisme,
- le large domaine d'applicabilité,
- la capacité d'abstraction,
- la possibilité d'une modélisation par raffinements successifs,
- la précision des réseaux de Petri,
- le non-déterminisme des tirs des transitions.

On a pu remarqué que les réseaux de Petri - comme toute autre méthode de spécification - possédaient certains inconvénients :

- l'explosion combinatoire des états lors de la création du graphe de marquage
- la non-différenciation des données dans les réseaux de Petri de base
- la difficulté d'analyse si on passe des "cas d'école" à des problèmes réels
- la difficulté de représentation de problèmes de grande taille
- l'inexistence d'une méthodologie générale de modélisation.

- l'inexistence d'une méthodologie générale de modélisation.

Ces inconvénients peuvent être atténués par diverses solutions (réseaux de Petri de haut niveau, outils d'aide à la conception, outils d'aide à l'analyse, méthode de modélisation par domaine d'application,....).

Ayant fait le bilan de ces inconvénients et avantages, nous pensons que ces derniers priment sur les premiers. Cette constatation est soutenue par une abondante publication et des conférences fréquentes au niveau international qui témoignent du vif intérêt que suscitent les réseaux de Petri. Ainsi, aura déjà lieu à Bonn, au mois de juin, la dixième conférence internationale sur la théorie et les applications des réseaux de Petri. Les thèmes principaux en seront des sujets aussi variés que les applications dans la bureautique, dans les systèmes flexibles de production, dans le domaine des langages de programmation, des systèmes temps réels, des mesures de performances, des systèmes intégrés,... . Le fait qu'on passe actuellement de plus en plus des systèmes séquentiels aux systèmes parallèles et concurrents, assure un bel avenir aux réseaux de Petri comme langage de modélisation et de vérification particulièrement adapté à de telles applications.

RÉFÉRENCES

[AYACHE85] J. M. Ayache, J. P. Courtiat, M. Diaz, G. Juanole : *Réseaux de Petri pour la modélisation et validation de protocoles*, TSI vol. 4, n° 1 1985

[AZÉMA87] P. Azéma, B. Berthomieu, J. P. Courtiat, M. Diaz, G. Juanole, B. Pradin : *Specification formelle et conception des systèmes informatiques distribués* , Dernier développement en Automatique, Informatique, Robotique, Micro-Electronique, Cepedues-Editions 1987

[AZÉMA88] P. Azéma, J. C. Lloret, G. Papanagiotakis, F. Vernadat : *ESTELLE validation and PROLOG Interpreted Petri Nets*, Rapport LAAS n° 88235, juin 1988

[BILLINGTON88] J. Billington, G. R. Wheeler, M. C. Wilbur-Ham : *PROTEAN - A high-level Petri Net tool for the specification and verification of communication protocols* , IEEE Trans. on S. E. ,14(3) 1988

[BOCHMANN79] G. v. Bochmann :*Architecture of distributed computer systems,* LNCS 77, Springer 1979

[BOCHMANN83] G.v. Bochmann : *Concepts for the distributed systems design*, Springer 1983

[COULOURIS88] G.F. Coulouris, J. Dollimore : *Distributed Systems : concepts and design* , Addison-Wesley Publishing Company 1988

[COURTIAT84] J.P. Courtiat, J.M. Ayache, B. Algayres : *Petri Nets are good for protocols*, Sigcomm 84 Montreal; June 1984

[DE BAKKER86] J. W. de Bakker : *Current trends in concurrency - overviews and tutorials*, Springer 1986

[DIAZ82] M. Diaz : *Modeling and analysis of communication and cooperation protocols using Petri net based models,* Tutorial Paper 2nd International Workshop on Protocol Specification, Testing andVerification Idyllwild (Los Angeles), mai 1982

[FICHEFET88] J. Fichefet : *Introduction aux réseaux de Petri*, Notes de cours 1ère licence 1988

[FIPAPP88] Normalisation Française C-46-602, *Bus FIP pour un échange d'information entre transmetteurs, actionneurs et automates - couche application.,* Union Technique de l'Electricité, mai 88

[GRFIP85] D. Galara, J.P. Thomese, *Groupe de réflexion FIP : Réseaux locaux industriels,* Ministère de l'industrie et de la recherche, 1985

[HASTIR88] P. Hastir, B. Jodocy, P. Azéma : *La couche Application de FIP en réseaux de Petri labellés à prédicats*, Rapport LAAS n° 88362, déc.1988

[HOLLIGER85] D. Holliger, *Utilisation pratique des réseaux de Petri dans la conception des systèmes de production*, Technique et Science Informatique, Vol4, n°1, p509-522, 1985.

[HWANG84] K. Hwang, F. A. Briggs : *Computer Architecture and parallel processing,* McGraw-Hill 1984

[JENSEN87] K. J. Jensen : *Computer tools for construction, modification and analysis of Petri Nets,* LNCS 255, Springer 1987

[JUANOLE83] G. Juanole, B. Algayres, J. Dufau : *Protocol Design and Modelling*, 4th European Workshop on Applications and theory of Petri NetsToulouse, sept. 1983

[KERHERVE85] B. Kerherve, D. Mermet, F. Pasquer, L. Verlaine, *Modélisation des algorithmes de Rosenkrantz par les réseaux de Petri,* Technique et Science Informatique, Vol4, N°1, p31-50, 1985

[LAMARCHE85] G. Lamarche et P. Taillibert, *Utilisation des réseaux de Petri pour le test des programmes temps réel,* Technique et Science Informatique, Vol4 n°1, p83-87, 1985

[LLORET88] J. C. Lloret, P. Azéma, F. Vernadat : *Réseaux Prédicat-Transition labellés : Structuration et Composition,* Rapport LAAS n° 88350, nov.1988

[MACCHI87] C. Macchi, J. F. Guilbert : *Téléinformatique,* Dunod 1987

[MONDAIN87] P. Mondain-Monval : *Conception et Vérification d'un service d'appel de procédure à distance dans les réseaux hétérogènes*, Thèse de doctorat Université Paul Sabatier Toulouse , nov. 1987

[NOVALI86] J. M. Novali : *Modèles d'observation pour les architectures multicouches de protocoles de communication*, Thèse de doctorat INSA Toulouse, nov. 1986

[OBERQUEL86] H. Oberquelle, *Human-Machine Interaction and Role/Fonction/Action Nets*, Lecture Notes in Computer Science, Springer 86

[OVIDE82] *OVIDE - Outil de Validation de Réseaux de Petri (manuel d'utilisation)*, Syseca-Temps Réel 1982

[PETERSON77] J.L. Peterson : *Petri Nets* , Computing Surveys vol. 9 n° 3 1977

[PETERSON81] J. L. Peterson : *Petri Net Theory and the Modelling of Systems* , Prentice-Hall 1981

[PETRI62] C.A. Petri : *Kommunikation mit Automaten*, Schriften des Institutes für Instrumentelle Mathematik, Bonn 1962

[PIPN] LAAS-VERILOG,*PIPN : environnement de prototypage et de vérification d'algorithmes répartis Manuel de référence version1* , janvier 88

[REISIG85] W. Reisig : *Petri Nets - An Introduction*, Springer 1985

[REISIG86] W. Reisig, *Petri Nets in Software Engineering*, Lecture Notes in Computer Science 255, Springer 1986

[RLIDO] Club FIP-LAAS, *Rapport Couche liaison de données*, TOULOUSE octobre 88

[RUDIN87] H. Rudin : *Tools for Network Protocols,* Annuel Review of Computer Science vol. 21987

[SHARP87] J.A. Sharp : *An introduction to distributed and parallel processing*, Computer Sciences Tests 1987

[VALETTE 88] R. Valette : *Réseaux de Petri - définitions de base et propriétés*, Notes de cours INSA Toulouse 1988

[VALETTE86] R. Valette, *Nets in production systems*, Lecture Notes in Computer Science 255, Springer 86

[ZIMMERMANN80] H. Zimmermann : *OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection*, IEEE Trans. on Comm. 28(4) 1980

ANNEXES

```
{ CONSOMMATEUR }


{ HEADER }

entity (cons).


child ([readloc]).

ip ( [  input (l (_)),
        output (l (_)),
        { l est le point d'interconnexion avec la couche liaison }
        input (a (_)),
        output (a (_)) ] ).
        { a est le point d'interconnexion avec l'utilisateur.    }


{ INITIALISATION DES PLACES }
init ( [ idle_cons,
        list_prompt([]),
        list_var_cons([
                tab(id,param([],[],VAL,RAF,RAFPONCT,vl,NONRESYNC,LOCAL,NORMAL))])]).
```

```
{ CONSOMMATEUR }

{ HEADER }

entity (cons).


child ([readloc]).

ip ( {  input (l (_)),
        output (l (_)),
        { l est le point d'interconnexion avec la couche liaison }
        input (a (_)),
        output (a (_)) } ).
        { a est le point d'interconnexion avec l'utilisateur.    }


{ INITIALISATION DES PLACES }
init ( { idle_cons,
        list_prompt([]),
        list_var_cons([
                tab(id,param([],[],VAL,RAF,RAFFONCT,v1,NONRESYNC,LOCAL,NORMAL))]) ]).
```

```
[ READLOC ]

entity (readloc).

cond (cons).

body (readloc).

ip ([   input(a(_)),
        output(a(_)),
        input(l(_)),
        output(l(_)) ]).
```

```
{ fichier readloc.body.pro }

{*************************}
{*  SERVICE LECTURE LOCALE  *}
{*************************}

{-----------------------------------------------------------------------}
{ PLACES IMPORTEES : on importe du module principal cons les places     }
{       idle_cons : l'etat de repos du consommateur                     }
{       list_prompt(l) : la liste des promptitudes associees aux        }
{              variables consommables par le consommateur.              }
{       list_var_cons(l) : la liste des variables consommables par le   }
{              consommateur.                                            }
{-----------------------------------------------------------------------}

imported_place ({
        cons(idle_cons),
        cons(list_prompt(_l)),
        cons(list_var_cons(_l)) }).


{-----------------------------------------------------------------------}
{ PLACES LOCALES :                                                      }
{       test_valid_rl_rq(a_nom) : la place ou l'on copie le nom de la   }
{              variable dont on demande la lecture locale.              }
{       wait_get_conf(id) : l'etat d'attente de la confirmation de la   }
{              confirmation de la lecture sur la couche liaison.        }
{-----------------------------------------------------------------------}

place ({ test_valid_rl_rq (_a_nom),
         wait_get_conf (_id_var) }).


{____SERVICE__A_READLOC____}


{-----------------------------------------------------------------------}
{ TR 1 : reception de la demande a_read_loc                             }
{       On recoit de l'utlisateur la demande de a_read_loc et on        }
{       va analyser la validite de la demande. On va regarder si la     }
{       variable possede un nom connu chez le consommateur.             }
{-----------------------------------------------------------------------}

tr(readloc_rq,
       label ({ input( a( a_readloc_rq(_a_nom))) }),
       pre ({ cons(idle_cons)}),
       post ({ test_valid_rl_rq(_a_nom) }),
       cond ({ rech_nom_accepte (_a_nom) })).



{-----------------------------------------------------------------------}
{ TR 2 : la demande de a_read_loc est refusee car le consommateur ne    }
{       connait pas de variable ayant ce nom en consommation locale.    }
{       On envoie a l'utilisateur une confirmation avec erreur de la    }
{       demande de lecture locale.                                      }
{-----------------------------------------------------------------------}

tr(readloc_rq_rejetee,
       label ({ output( a( a_readloc_conf(_a_nom,erreur))) }),
       pre  ({ test_valid_rl_rq(_a_nom),
               cons(list_var_cons(_l)) }),
               cons(list_var_cons(_l)) }),
       cond ({ assertion_var_cons(_l),
               \+ (verif_accept_readloc(_a_nom)) })).



{-----------------------------------------------------------------------}
{ TR 3 : la demande de a_read_loc est acceptee  pour une variable       }
{       non resynchronisable.                                           }
{       On envoie vers la couche liaison un l_get_dem et on passe dans  }
{       un etat d'attente de la confirmation de lecture.                }
{-----------------------------------------------------------------------}

tr(rq_acc_var_nonresync_et_get_rq,
       label ({ output (l (l_get_dem(_id_var))) }),
       pre  ({ test_valid_rl_rq(_a_nom),
                cons(list_var_cons(_l))}),
       post ({ wait_get_conf (_id_var),
               cons(list_var_cons(_l)) }),
       cond ({ assertion_var_cons(_l),
               verif_accept_readloc(_a_nom),
               rech_corresp_var_cons(_id_var,_a_nom),
               rech_resync_var_cons(_id_var,NONRESYNC)})).

{-----------------------------------------------------------------------}
{ TR 4 : la demande de a_read_loc est acceptee pour une variable        }
{       resynchronisable.                                               }
{       On recherche la promptitude privee, la promptitude ponctuelle   }
{       privee, la valeur privee, le rafraichissement et le             }
{       rafraichissement ponctuel , puis on transmet le tout a          }
{       l'utilisateur                                                   }
{-----------------------------------------------------------------------}

tr(rq_acc_var_resync_et_rl_conf,
       label ({ output (a (a_readloc_conf(_a_nom,
                       data( _valpr,_rafr,_rafrp,_prompt,_promptp)))) }),
       pre  ({ test_valid_rl_rq(_a_nom),
               cons(list_prompt(_l1)),
               cons(list_var_cons(_l2)) }),
       post ({ cons(idle_cons),
               cons(list_prompt(_l1)),
               cons(list_var_cons(_l2)) }),
       cond ({ assertion_prompt(_l1),
               assertion_var_cons(_l2),
               verif_accept_readloc(_a_nom),
               rech_corresp_var_cons(_id_var,_a_nom),
               rech_resync_var_cons(_id_var,RESYNC),
               rech_valpr_var_cons(_id_var,_valpr),
               rech_status_rafr_var_cons(_id_var,_rafr),
               rech_status_rafrp_var_cons(_id_var,_rafrp),
               rech_statprpr_var_cons(_id_var,_prompt,_promptp) })).



{-----------------------------------------------------------------------}
{ TR 5 : reception de la confirmation de lecture sur la couche liaison  }
{       et envoi de la valeur vers l'utilisateur                        }
{       On recoit de la couche liaison la valeur publique et les status }
{       de rafraichissement et de rafraichissement ponctuel. On recher- }
{       che les status de promptitude et de promptitude ponctuelle, puis }
{       on transmet le tout vers l'utilisateur.                         }
{-----------------------------------------------------------------------}
```

```
        label ([ input( l( l_get_conf(_id_var,data(_val,_rafr,_rafrp)))) ,
                 output( a( a_readloc_conf(_a_nom,
                          data(_val,_rafr,_rafrp,_prompt,_promptp)))) ]),
        pre  ([ wait_get_conf (_id_var),
                cons(list_prompt(_l1)),
                cons(list_var_cons(_l2)) ]),
        post ([ cons(idle_cons),
                cons(list_prompt(_l1)),
                cons(list_var_cons(_l2)) ]),
        cond ([ assertion_prompt(_l1),
                assertion_var_cons(_l2),
                rech_corresp_var_cons(_id_var,_a_nom),
                tab_data(_val,_rafr,_rafrp),
                rech_statprpu_var_cons(_id_var,_prompt,_promptp) ])).


|-------------------------------------------------------------------------|
| TR 6 : reception de la confirmation avec erreur de la lecture sur la    |
|        couche liaison et envoi de la confirmation avec erreur de la     |
|        lecture locale vers l'utilisateur.                               |
|-------------------------------------------------------------------------|

tr(get_err_var_nonresync_et_rl_err,
        label ([ input( l( l_get_conf(status))) ,
                 output( a( a_readloc_conf(_a_nom,erreur))) ]),
        pre  ([ wait_get_conf (_id_var) ]),
        post ([ cons(idle_cons) ]),
        cond ([ rech_corresp_var_cons (_id_var,_a_nom) ])).
```

```
| extrait du fichier cons.cond.pro : conditions necessaires pour le     |
|                                    service A_READLOC                  |

|*************************************************************************|
|*************************************************************************|
|**                                                                     **|
|**            CONDITIONS POUR LE CONSOMMATEUR                          **|
|**                                                                     **|
|*************************************************************************|
|*************************************************************************|

|*************************************************************************|
|*                                                                       *|
|*         FONCTIONS DE CONSTRUCTION ET DE SAUVETAGE DE LA               *|
|*                        BASE DE DONNEES                                *|
|*                                                                       *|
|*************************************************************************|

|-------------------------------------------------------------------------|
| FONCTION DE CONSTRUCTION DE LA BASE DE DONNEES POUR LES PROMPTITUDES     |
|-------------------------------------------------------------------------|

assertion_prompt(_liste) :-
        retractall(prom(_,_)),
        ajout_prompt(_liste).

ajout_prompt ([]).
ajout_prompt([tab(_id,_p)|_T]) :-
        assert(prom(_id,_p)),
        ajout_prompt(_T).

|-------------------------------------------------------------------------|
| FONCTION DE SAUVETAGE DE LA BASE DE DONNEES POUR LES PROMPTITUDES       |
|-------------------------------------------------------------------------|

construire_list_prompt(_l_prompt) :-
        id_prompt(_l_id_prompt),
        lire_prompt(_l_id_prompt,_l_prompt).

lire_prompt([],[]).
lire_prompt([_id|_T],[tab(_id,_p)|_Q]):-
        prom (_id,_p),!,
        lire_prompt(_T,_Q).


|-------------------------------------------------------------------------|
| FONCTION DE CONSTRUCTION DE LA BASE DE DONNEES POUR LES VARIABLES       |
| CONSOMMEES                                                              |
|-------------------------------------------------------------------------|

assertion_var_cons(_liste) :-
        retractall(var_cons(_,_)),
        ajout_var_cons(_liste).

ajout_var_cons([]).
ajout_var_cons([tab(_id,_p)|_t]) :-
        assert(var_cons(_id,_p)),
        ajout_var_cons(_t).
|-------------------------------------------------------------------------|
| FONCTION DE SAUVETAGE  DE LA BASE DE DONNEES POUR LES VARIABLES         |
| CONSOMMEES                                                              |
|-------------------------------------------------------------------------|

        id_var_cons(_l_id_var_cons),
        lire_var_cons(_l_id_var_cons,_l_var_cons).

lire_var_cons([],[]).
lire_var_cons([_id|_q1],[tab(_id,_p)|_q2]) :-
        var_cons(_id,_p),!,
        lire_var_cons(_q1,_q2).


|*************************************************************************|
|*                                                                       *|
|*      FONCTIONS DE RECHERCHE ET DE MISE A JOUR D'INFORMATIONS          *|
|*              CONCERNANT DES VARIABLES CONSOMMEES                      *|
|*                                                                       *|
|*************************************************************************|

|-------------------------------------------------------------------------|
| FONCTIONS MANIPULANT DES VARIABLES CONSOMMEES :                         |
| fonctions de recherche a) des references aux promptitudes privees       |
|                        b) des references des promptitudes publiques     |
|                        c) de la valeur privee                           |
|                        d) du status du rafr. (a)synchrone               |
|                        e) du status du rafr. ponct.                     |
|                        f) du nom de correspondance avec l'utilisateur   |
|                        g) du caractere (resync. ou non resync.)         |
|                        h) du service a utiliser (local ou lointain)     |
|                        i) du type de variable (normale,...)             |
|                        j) des status des promptitudes privees           |
|                        k) des status des promptitudes publiques         |
| fonctions de mise a jour a) de la valeur privee                         |
|                          b) du status du rafr. (a)synchrone             |
|                          c) du status du rafr. ponct.                   |
|-------------------------------------------------------------------------|

rech_ref_prpr_var_cons(_id,_L_pr_pr) :-
        var_cons (_id,param(_L_pr_pr,_L_pr_pu,_val_pr,_rafr,_rafrp,_a_nom,
                _resyn,_serv,_type)).
rech_ref_prpu_var_cons(_id,_L_pr_pu) :-
        var_cons (_id,param(_L_pr_pr,_L_pr_pu,_val_pr,_rafr,_rafrp,_a_nom,
                _resyn,_serv,_type)).
rech_valpr_var_cons(_id,_val_pr) :-
        var_cons (_id,param(_L_pr_pr,_L_pr_pu,_val_pr,_rafr,_rafrp,_a_nom,
                _resyn,_serv,_type)).
rech_status_rafr_var_cons(_id,_rafr) :-
        var_cons (_id,param(_L_pr_pr,_L_pr_pu,_val_pr,_rafr,_rafrp,_a_nom,
                _resyn,_serv,_type)).
rech_status_rafrp_var_cons(_id,_rafrp) :-
        var_cons (_id,param(_L_pr_pr,_L_pr_pu,_val_pr,_rafr,_rafrp,_a_nom,
                _resyn,_serv,_type)).
rech_corresp_var_cons(_id,_a_nom) :-
        var_cons (_id,param(_L_pr_pr,_L_pr_pu,_val_pr,_rafr,_rafrp,_a_nom,
                _resyn,_serv,_type)).
rech_resync_var_cons(_id,_resyn) :-
        var_cons (_id,param(_L_pr_pr,_L_pr_pu,_val_pr,_rafr,_rafrp,_a_nom,
                _resyn,_serv,_type)).
rech_service_var_cons(_id,_serv) :-
        var_cons (_id,param(_L_pr_pr,_L_pr_pu,_val_pr,_rafr,_rafrp,_a_nom,
                _resyn,_serv,_type)).
rech_type_var_cons(_id,_type) :-
        var_cons (_id,param(_L_pr_pr,_L_pr_pu,_val_pr,_rafr,_rafrp,_a_nom,
                _resyn,_serv,_type)).
rech_statprpr_var_cons(_id,_pr,_prp) :-
        rech_ref_prpr_var_cons(_id,_l_refpr),
        rech_prompt_var_cons(_l_refpr,_pr,_prp).
```

```
rech_statprpu_var_cons(_id,_pr,_prp) :-
        rech_re(_prpu_var_cons(_id,_l_refpr),
        rech_prompt_var_cons(_l_refpr,_pr,_prp).

( fonction auxiliaire pour la recherche des status de promptitudes      )
( privees et publiques                                                  )
        rech_prompt_var_cons([],NONELAB,NONELAB):- !.
        rech_prompt_var_cons([_id_pr],NONELAB,_statuspr):-
                rech_car_prompt(_id_pr,PONCTUEL),!,
                rech_status_prompt(_id_pr,_statuspr).
        rech_prompt_var_cons([_id_pr],_statuspr,NONELAB):-
                \+ (rech_car_prompt(_id_pr,PONCTUEL)),!,
                rech_status_prompt(_id_pr,_statuspr).
        rech_prompt_var_cons([_id_pr1,_id_pr2],_statuspr,_statusprp):-
                rech_car_prompt(_id_pr1,PONCTUEL),!,
                rech_status_prompt(_id_pr1,_statusprp),
                rech_status_prompt(_id_pr2,_statuspr).
        rech_prompt_var_cons([_id_pr1,_id_pr2],_statuspr,_statusprp):-
                rech_car_prompt(_id_pr2,PONCTUEL),!,
                rech_status_prompt(_id_pr2,_statusprp),
                rech_status_prompt(_id_pr1,_statuspr).
( fin fonction auxiliaire                                               )

maj_valpr_var_cons(_id,_newval) :-
        retract(var_cons (_id,param(_L_pr_pr,_L_pr_pu,_val_pr,_rafr,_rafrp,
                _a_nom,_resyn,_serv,_type))),
        assert(var_cons (_id,param(_L_pr_pr,_L_pr_pu,_newval,_rafr,_rafrp,
                _a_nom,_resyn,_serv,_type))).
maj_status_rafr_var_cons(_id,_newval) :-
        retract(var_cons (_id,param(_L_pr_pr,_L_pr_pu,_val_pr,_rafr,_rafrp,
                _a_nom,_resyn,_serv,_type))),
        assert(var_cons (_id,param(_L_pr_pr,_L_pr_pu,_val_pr,_newval,_rafrp,
                _a_nom,_resyn,_serv,_type))).
maj_status_rafrp_var_cons(_id,_newval) :-
        retract(var_cons (_id,param(_L_pr_pr,_L_pr_pu,_val_pr,_rafr,_rafrp,
                _a_nom,_resyn,_serv,_type))),
        assert(var_cons (_id,param(_L_pr_pr,_L_pr_pu,_val_pr,_rafr,_newval,
                _a_nom,_resyn,_serv,_type))).

|-------------------------------------------------------------------------|
| FONCTIONS POUR LES PROMPTITUDES :                                       |
|fonctions de recherche  a) de la reference a la variable concernee par  |
|                           une promptitude                               |
|                        b) de l'etat du temporisateur associe a la       |
|                           promptitude                                   |
|                        c) du caractere de la prompt.                    |
|                        d) du status de la prompt.                       |
| fonctions de m-a-j     a) de l'etat du temporisateur associe a la       |
|                           promptitude                                   |
|                        b) du status de la promptitude                   |
|-------------------------------------------------------------------------|

rech_id_var_prompt(_id_prompt,_id_var) :-
        prom(_id_prompt,param(_id_var,_etat,_car,_status)).
rech_etattempo_prompt(_id_prompt,_etat) :-
        prom(_id_prompt,param(_id_var,_etat,_car,_status)).
rech_car_prompt(_id_prompt,_car) :-
        prom(_id_prompt,param(_id_var,_etat,_car,_status)).
rech_status_prompt(_id_prompt,_status) :-
        prom(_id_prompt,param(_id_var,_etat,_car,_status)).
maj_etattempo_prompt(_id_prompt,_etatnew) :-
        retract(prom(_id_prompt,param(_id_var,_etat,_car,_status))),
        assert(prom(_id_prompt,param(_id_var,_etatnew,_car,_status))).
maj_status_prompt(_id_prompt,_statusnew) :-
        retract(prom(_id_prompt,param(_id_var,_etat,_car,_status))),
        assert(prom(_id_prompt,param(_id_var,_etat,_car,_statusnew))).



|*************************************************************************|
|*                                                                       *|
|*      FONCTIONS DE VERIFICATION DE LA VALIDITE D'UNE DEMANDE           *|
|*                  DE LECTURE LOCALE                                    *|
|*                  DE LECTURE LOINTAINE                                 *|
|*                  DE LECTURE D'UNE LISTE                               *|
|*                  DE LECTURE UNIVERSELLE                               *|
|*                  DE MISE A JOUR (UPDATE)                              *|
|*                                                                       *|
|*************************************************************************|

|-------------------------------------------------------------------------|
| FONCTION DE VERIFICATION DE LA VALIDITE D'UNE DEMANDE DE READLOC :      |
|       Une demande est valide si la demande porte sur une variable       |
|       consomme de type "local"                                          |
|-------------------------------------------------------------------------|

verif_accept_readloc(_a_nom) :-
        rech_corresp_var_cons(_id_var,_a_nom),
        rech_service_var_cons(_id_var,LOCAL).


|*************************************************************************|
|*                                                                       *|
|*      FONCTIONS SERVANT A INSTANCIER DES VARIABLES RECUES EN           *|
|*                              "INPUT"                                  *|
|*                                                                       *|
|*************************************************************************|

val_ech(val_plus_un).
        ( predicat servant a instancier la valeur d'une variable de     )
        ( synchronisation lors d'un i_get_conf                           )

tab_data(VAL,RAF,RAFPONCT).
        ( predicat servant a instancier la valeur, le rafraichissement   )
        ( et le rafraichissement ponctuel lors d'un i_get_conf           )

rech_nom_accepte(_a_nom) :-
        tab_nom(_a_nom).
        ( fonction servant a instancier les noms des variables donnes en)
        ( input par l'utilisateur lors d'une lecture ou un update chez   )
        ( le consommateur                                                )



|*************************************************************************|
|*                                                                       *|
|*                  LES "TABLES FIXES" DU CONSOMMATEUR                   *|
|*                                                                       *|
|*************************************************************************|



tab_nom(v1).
tab_nom(v2).
                ( ensemble de tous les noms connus par le systeme        )
```

```
id_var_cons({id]).
                { liste des identificateurs des variables en consommation }

id_prompt({]).
                { liste des identificateurs des promptitudes des variables }
```