UTILISATION DES SYSTEMES DE REGLES POUR LA MISE EN OEUVRE ET LA SUPERVISION DES ATELIERS FLEXIBLES

Babou Bako

Ecole Supérieure Interafricaine de l'Electricité
B.P. 311 Bingerville, République de Côte d'Ivoire

Robert Valette

LAAS-CNRS, 7 Av. du Colonel Roche, 31077 TOULOUSE CEDEX, France
e-mail robert@laas.laas.fr.

RESUME : Le but de cet article est de montrer que l'approche des réseaux de Petri peut être combinée avec le concept objet et les systèmes de règles. Les systèmes de règles contrôlés par réseaux de Petri sont définis et leur mise en oeuvre basé sur l'utilisation conjointe du concept du "joueur" de réseaux de Petri [3] et des techniques de compilation de règles [7] est également présentée.

Mots clés : Réseaux de Petri, Systèmes de règles, Ateliers flexibles

ABSTRACT : The aim of this paper is to show that a Petri net based approach may be combined together with the notions of "object" and "production rule". Controlled Rule-based Systems are defined and their implementation through the "token player" principles and the rule "compilation" technique altogether is described as well.

Keywords : Petri Nets, Rule-based systems, Manufacturing systems

I INTRODUCTION

La similitude entre un réseau de Petri et un système de règles est maintenant bien connue. Les réseaux de Petri ordinaires peuvent être traduits par un système de règles de la logique propositionnelle en associant une règle à chaque transition. Les entrées correspondent aux "pré-conditions" et les sorties aux "post-conditions". Des travaux récents montrent aussi que les programmes basés sur les clauses de Horn peuvent être traduits en réseaux "Predicat-Transition" [11].

Cependant, similaire ne signifie pas équivalent. Un réseau de Petri est une représentation d'un type de connaissance bien précis. Il décrit des relations de *causalité* dans une logique non monotone basée sur les notions d'état et d'événement. Pour cela, le réseau de Petri est d'une utilisation très intéressante chaque fois que la notion de contrôle est importante dans une application.

Les systèmes de commande en temps réel des ateliers flexibles sont un exemple typique de telles applications. La décision de commencer l'exécution d'une opération sur une machine qui vient d'être libérée demande qu'on considère en

même temps une connaissance non "procedurale" (du type "c'est mieux de commencer par l'opération de durée la plus courte") et des relations de causalité (du type "l'opération op3 doit toujours suivre l'opération op2 "). C'est pour cette raison que la tendance actuelle est d'utiliser conjointement les réseaux de Petri et les systèmes de règles de production [6,10,13,16]. Le but de ce travail est de montrer la faisabilité de cette approche de façon intégrée, en combinant les avantages d'une représentation claire et sans ambiguïté du contrôle avec la possibilité de manipuler des connaissances déclaratives à travers des règles de production.

L'utilisation d'un mécanisme de compilation semblable à la compilation des systèmes de règles de production permet une implémentation efficace des réseaux de Petri. En même temps, pour ceux qui travaillent avec des systèmes de règles, la représentation explicite du contrôle d'un système de règles par un réseau de Petri permet une optimisation de la compilation.

II SYSTEMES DE REGLES CONTROLES PAR RESEAUX DE PETRI

II.1. Introduction

Un système de règles contrôlé par réseaux de Petri est un réseau de Petri dans lequel une règle est associée à chaque transition. Chaque jeton représente un n–uplet d'objets, et la règle associée à la transition ne porte que sur les objets des jetons contenus dans ses places d'entrée. Les variables contenues dans les expressions des règles sont les variables associées aux arcs d'entrée et de sortie de la transition. Ces variables ne peuvent être substituées que par les objets des jetons contenus dans les places d'entrée de la transition. Tous les objets d'un même jeton doivent être substitués globalement et de façon indivisible.

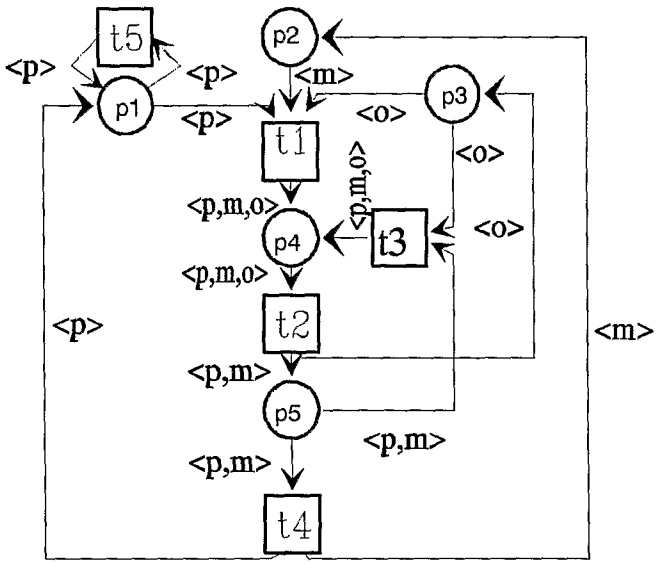
Mis à part la syntaxe des règles, ce modèle correspond exactement au modèle des réseaux de Petri à objets [14]. Si on ne prend pas en compte la structuration des données en terme de classes d'objets, il est aussi équivalent au réseaux de Petri Prédicats/Transition [16].

Dans le domaine des ateliers flexibles, les jetons qui sont des n–uplets d'instances d'objets, représentent les objets physiques de l'ateliers et leurs interactions. Par exemple, une pièce sur une machine, une pièce sur une palette, etc. Ces relations sont établies et détruites par des franchissements de transitions.

Les règles associées aux transitions peuvent être considérées comme des méthodes des classes d'objets. Elles représentent les transformations que les objets doivent subir. Dans les approches à objets, une méthode est associée à une classe d'objet. Dans notre approche, une méthode est associée à plusieurs classes d'objet à la fois.

II.2. Exemple

Cet exemple représente la spécification à l'aide de systèmes de règles contrôlés par réseaux de Petri d'une cellule d'usinage constituée d'un ensemble de machines devant réaliser des opérations sur un ensemble de pièces à l'aide d'outils.



Classe: *Piece*
attributs:
identificateur
ops_à_subir
op_courante
ordre_op
méthodes:
prochaine_op()

Classe: *Outil*
attributs:
identificateur
ops_réalisables

Classe: *Machine*
attributs:
identificateur
ops_réalisables

figure 1

La place *p1* contient les pièces qui attendent de subir une opération, la place *p2*, les machines libres et la place *p3*, les outils non en cours d'utilisation. La place *p4*, elle, représente l'ensemble des opérations d'usinage en cours. La place *p5* contient toutes les pièces qui sont sur des machines en attente d'un déchargement éventuel ou du lancement de leur prochaine opération.

La transition *t1* correspond au lancement d'une opération d'usinage et la transition *t2* à la fin d'une opération d'usinage. Chaque tir de *t2*, sera suivi d'un tir de *t3* si à la fin de l'opération d'usinage, la prochaine opération, sur la même pièce est réalisable par la même machine. Dans le cas contraire, *t4* est franchie, ce qui met la machine dans l'état libre. La transition *t5* sera franchie chaque fois qu'une pièce aura terminé sa gamme (ensemble d'opérations qu'elle doit subir) pour faire entrer une nouvelle pièce.

Trois classes d'objets sont manipulées dans ce réseau de Petri: les objets des classe *Piece*, *Machine* et *Outil*. Chacune de ces classes représente les objets physiques existant dans la cellule d'usinage. La variable *p* est de classe *Piece*, *m* de classe Machine et *o* de classe Outil.

Supposons les règles suivantes associées aux transitions du réseau de Petri

t1:
condition:
(p.op_courante ≠ ())
(p.op_courante ∈ m.ops_réalisables)
(p.op_courante ∈ o.ops_réalisables)
action:
()

t3:
condition:
(p.op_courante ≠ ())
(p.op_courante ∈ m.ops_réalisables)
(p.op_courante ∈ o.ops_réalisables)
action:
()

t2:
condition:
()
action:
(p.op_courante = p.prochaine_op())

t4:
condition:
(p.op_courante ∉ m.ops_réalisables)
action:
()

t5:
condition:
(p.op_courante == ())
action:
(p.op_courante = p.prochaine_op())

II.3. La syntaxe des règles

Une règle est représentée par un couple **condition – action**. La partie condition est un ensemble de tests portant sur les objets des jetons contenus dans les places d'entrée de la transition et qui détermine la possibilité de pouvoir franchir la transition. La partie action représente tout ce qui peut être déduit du franchissement de la transition. C'est elle qui représente la liste des modifications qu'on va apporter au marquage des places de sortie chaque fois que l'on franchira la transition.

Voyons plus en détail ces deux parties.

II.3.1. La partie condition

La partie condition d'une règle consiste en un ensemble de prédicats portant sur une ou plusieurs classes d'objets. Les prédicats servent à filtrer des jetons ayant certaines caractéristiques. On dit qu'un ensemble de jetons constitue une instance consistante d'une transition si et seulement si ses jetons vérifient la liste de prédicats constituant la partie condition de la règle associée à cette transition.

Par exemple, la règle associée à la transition *t1* permet de choisir une machine et un outil pour la réalisation d'une opération d'une pièce donnée.

condition (t3):
(p.op_courante ≠ ())
(p.op_courante ∈ m.ops_réalisables)
(p.op_courante ∈ o.ops_réalisables)

On peut noter la présence de deux types de prédicats dans la partie condition d'une règle:

- Les prédicats qui servent à réaliser des tests portant sur des objets d'un même jeton, et sont pour cela appelés des prédicats mono–jeton. Ils servent à réaliser une comparaison entre un attribut d'un objet avec une constante ou entre des attributs de deux objets d'un même jeton.
- Les prédicats qui servent à réaliser des tests faisant intervenir des objets de deux jetons différents et sont pour cela appelés prédicats multi–jetons.

Par exemple, le prédicat *(p.op_courante ≠ ())* qui vérifie qu'il reste encore une opération à réaliser sur la pièce est un prédicat mono–jeton, tandis que le prédicat *(p.op_courante ∈ o.ops_réalisables)* est un prédicat multi–jetons.

Il convient de noter que le prédicat *(p.op_courante ∈ m.ops_réalisables)* est un prédicat mono–jeton dans le cas de la règle associée à la transition *t3* alors que ce même prédicat est un prédicat multi–jetons dans la cas de la règle associée à la transition *t1*.

II.3.2. La partie action

La partie action est une séquence d'affectations (modifications) des attributs des objets contenus dans les jetons. La syntaxe de la partie action est similaire à celle de la partie condition sauf que les prédicats sont remplacés par des affectations.

Par exemple, lorsqu'une machine vient de terminer une opération sur une pièce donnée (franchissement de la transition *t2*), l'exécution de la règle associée à cette transition à pour effet de mettre à jour l'attribut *op_courante*. Dans cet exemple la partie action est réduite à une seule affectation. Une illustration plus complète de la syntaxe de la partie action est la suivante :

action:

(m.état = p.identificateur)
(o.état = p.identificateur)
(p.op_courante = p.prochaine_op())

II.4. Systèmes de règles contrôlés par réseaux de Petri et Systèmes de règles classiques

Un système de règles contrôlé par réseau de Petri est composé d'une base de faits qui contient tous les faits de l'état actuel (marquage), et d'une base de règles (couple **transition–règle**) qui décrit tous les moyens de déduire de nouveaux faits. La moteur d'inférence doit parcourir, à chaque cycle d'inférence, toutes les règles. On doit comparer la condition d'application de chaque règle à la base des faits (marquage des places d'entrée de la transition), en remplaçant les variables intervenant dans la règle par des faits. Cette opération est appelée "pattern–

matching" des faits par les règles. A la fin de chaque cycle d'inférence, on doit choisir un élément de l'ensemble de conflits et l'appliquer.

Par exemple, la partie de l'ensemble des conflits associée à la règle de la transition t1 doit être composée de triplets <pièce, machine, outil> vérifiant la condition de cette règle. On choisira, après, un de ces triplets pour franchir la transition (résolution de conflits).

La taille de l'ensemble des conflits peut être très grande (explosion combinatoire) et donc il serait pénalisant en temps d'avoir à le recalculer totalement à chaque cycle. D'où l'idée d'utiliser, pour la mise en oeuvre des systèmes de règles contrôlés par réseaux de Petri, des méthodes basée sur le principe de compilation [7,12]. Ce principe est de construire un graphe qui limitera le nombre de tests, en conservant entre deux cycles d'inférence la partie de l'ensemble des conflits qui reste toujours vraie. De cette façon la construction de l'ensemble des conflits devient totalement incrémentale.

L'utilisation directe de cette approche nécessite de traduire le système de règles contrôlé par réseau de Petri en un système de règles classique (où le contrôle n'est pas explicite), en transformant la structure de contrôle décrite par le réseau de Petri en conditions élémentaires spécifiques portant sur l'état (localisation dans les places) des jetons. A une transition et sa règle associée, on substitue une règle d'un système de règles de production classique. Par exemple, à la règle associée à la transition t1, on substitue la règle suivante :

condition :

- *(classe(p) == piece)(p.état == p1)(p.op_courante == ϕ)*
- *(classe(m) == machine)(m.état == p2)(m.identificateur == ϕ')(ϕ ∈ m.ops_réalisables)*
- *(classe(o) == outil)(o.état == p3)(o.identificateur == ϕ")(ϕ ∈ o.ops_réalisables)*

action :

- *(classe(p) = piece)(p.état = p4)(p.machine_courante = ϕ')(p.outil_courant = ϕ")*
- *(classe(m) = machine)(m.état = p4)*
- *(classe(o) = outil)(o.état = p4)*

On note qu'il est nécessaire d'augmenter le nombre d'attributs des objets. L'attribut "*état*" contient le nom de la place dans laquelle est localisée le jeton contenant les objets. De plus, les attributs "machine_courante" et "outil_courant" servent à décrire les relations dynamiques existant entre les pièces, les outils et les machines. Ces relations étaient décrites dans le système de règles contrôlé par réseau de Petri par le 3_uplet <p,m,o>. Le rôle des variables ϕ' et ϕ" est de capter l'identificateur de la machine et de l'outil pour ensuite attribuer ces valeurs aux attributs "*machine_courante*" et "*outil_courant*". Ces variables introduisent des conditions toujours vraies mais qui seront tout de même évaluées. De plus, cette méthode ne permet pas de prendre en compte les relations de causalité décrites par la structure de contrôle (le réseau de Petri).

III COMPILATION DES SYSTEMES DE REGLES CONTROLES PAR RESEAUX DE PETRI

Les principes de compilation des systèmes de règles contrôlés par réseaux de Petri ont été introduits dans [19]. Cette technique est basée sur l'exploitation d'un graphe obtenu après compilation du système de règles contrôlé par réseau de Petri . Ce graphe est constitué de deux types d'arbres:

- arbre de tests,
- réseau de jointures.

Nous allons présenter succinctement à partir de l'exemple précédent la construction de ce graphe et son exploitation par un programme "Joueur".

III.1. Construction des arbres de tests

Pour la construction de ces arbres, on va considérer chaque place avec l'ensemble de ses transitions de sortie. A chaque n-uplet de variables associé à un arc de sortie de la transition, on associe une branche de l'arbre. Les noeuds correspondent aux prédicats portant sur ces variables. C'est dans cet arbre que seront propagées des copies des jetons qui seront déposés dans la place considérée. Si le prédicat associé à un noeud est vrai, une copie du jeton est envoyée à chacun des successeurs du noeud. Dans le cas contraire, la copie est simplement détruite.

En considérant la place p1, on obtient le graphe de la figure 2.

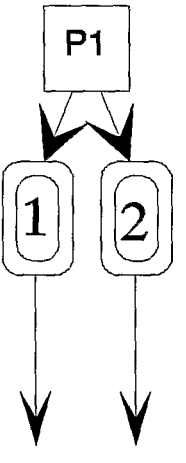


figure 2

Le noeud racine de cet arbre de tests représente la place p1. Chacun des deux noeuds suivants de ce noeud possède un test. Au noeud 1 est associé le test *(p.op_courante* $\neq$ *())* et au noeud 2 le test *(p.op_courante* == *())*. Ceci correspond parfaitement à ce qui se passe au niveau du système de règles contrôlé par réseau de Petri. En effet, lorsqu'un jeton est déposé dans la place *p1*, ce jeton peut être utilisé soit pour le franchissement de *t1* si *(p.op_courante* $\neq$ *())* soit pour le franchissement de *t5* si *(p.op_courante* == *())*.
Les jetons arrivant à chaque feuille de cet arbre correspondent aux jetons qui ont une possibilité d'être utilisés pour le franchissement de la transition correspondante. Cet arbre est en général, un arbre qui effectue un filtrage "non exclusif" des jetons de la place p1.

III.2. Construction des réseaux de jointures

Le but des réseaux de jointures est de construire pour chaque transition l'ensemble des n-uplets de jetons qui peuvent être utilisés pour son franchissement et de les conserver. Voyons comment ces réseaux de jointures sont construits. Pour cela, considérons le réseau de jointures associé à la transition *t1*.

Il est constitué de deux noeuds (noeud 5 et 6) et d'un noeud particulier représentant la transition t1. La seule différence entre les noeuds 5 et 6, et le noeud *t1* est que les deux premiers possèdent chacun une relation de jointure alors que l'autre n'en possède pas. Tous ces trois noeuds possèdent chacun deux mémoires (mémoire gauche et mémoire droite). Ces mémoires servent à conserver les instances partielles et totales de la transition *t1*.

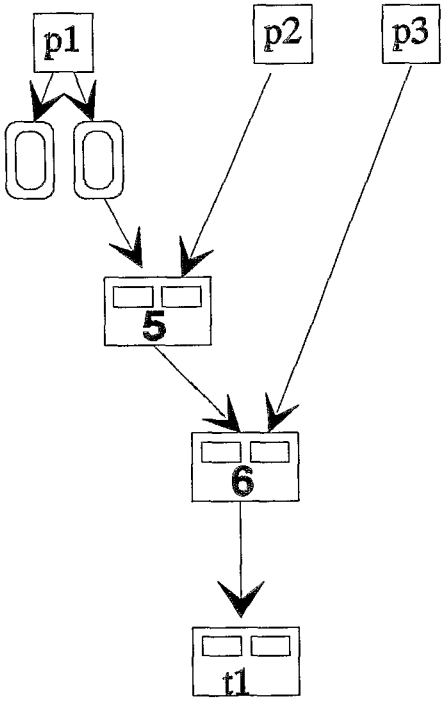


figure 3

Par exemple, la mémoire gauche du noeud 5 contient tous les jetons *(pièce)* de la place *p1* qui ont satisfait le test *(p.op_courante* ≠ *())* (car seules ces pièces peuvent être utilisées pour le franchissement de *t1*). La mémoire droite du noeud 5 contient tous les jetons *(machine)* de la place *p2*.

Le rôle du noeud 5 est de déterminer, à partir du contenu de ses deux mémoires, tous les couples *(pièce machine)* qui sont tels que la machine est capable de réaliser l'opération courante de la pièce et de les transmettre au noeud 6. C'est pour cette raison qu'est associée au noeud 5 la relation de jointure suivante:

p.op_courante ∈ *m.ops_réalisables.*

Le noeud 6 qui reçoit ces couples, les stocke dans sa mémoire gauche. La mémoire droite du noeud 6 contient tous les outils de la place *p3*. Le rôle du noeud 6 est de déterminer, à partir du contenu de ses deux mémoires, tous les triplets *(pièce machine outil)* qui sont tels que la machine est capable de réaliser

l'opération courante de la pièce avec l'outil, puis de les transmettre à son successeur. La relation de jointure du noeud 6 est la suivante:

(p.op_courante ∈ o.ops_réalisables)

Ce qui fait qu'on obtient dans la mémoire gauche du noeud *t1* la liste des instances totales de la transition *t1*.

Vu le volume des tests réalisés par les jointures, on comprend bien que la majeure partie du temps de cycle de l'interpréteur du graphe (le programme "joueur") sera consacrée à l'énumération des instances de transition. La compilation permet d'optimiser ce temps.

Il convient de noter qu'un réseau de jointures peut avoir pour noeuds d'entrée, des feuilles de plus d'un arbre de tests. C'est ce qui permet de conserver la structure du réseau de Petri.

III.3. Interprétation du graphe compilé

La fonction principale du programme "joueur" est de choisir une transition parmi les transitions franchissables et de la franchir. Dans cette approche, la recherche des transitions franchissables devient immédiate une fois que les jetons constituant le marquage ont été propagés dans le graphe.
La figure 4 représente le graphe obtenu après la compilation du système de règles contrôlé par réseau de Petri présenté dans le paragraphe II.2.

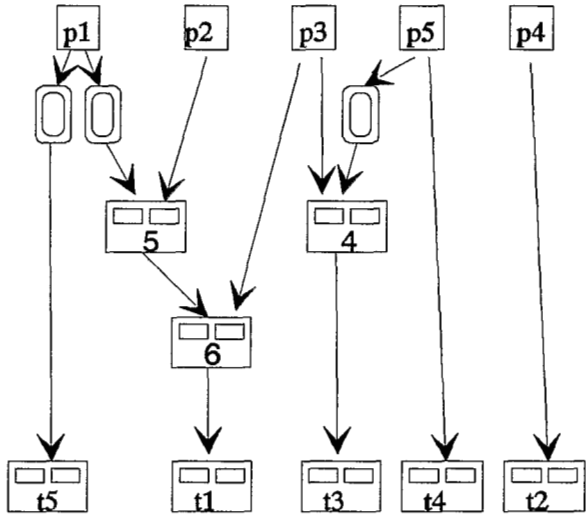


figure 4

Supposons le marquage initial suivant:

marquage(p1) = <$p1> + <$p2> marquage(p2) = <$m1> + <$m2>

marquage(p3) = <$o1> + <$o2>

avec les contraintes suivantes:

1. la machine m1 est capable de réaliser l'opération op1 de la pièce p1 avec l'outil o1

2. la machine m2 est capable de réaliser l'opération op2 de la pièce p2 avec l'outil o2
3. la machine m2 est capable de réaliser l'opération op1 de la pièce p1 avec l'outil o2

En propageant les jetons du marquage initial (<$p1>, <$p2> , <$m1> , <$m2> , <$o1> et <$o2>) dans le graphe, on obtient:

jointure 5:gauche = {<$p1>, <$p2>}

jointure 5:droite = {<$m1> , <$m2>}

jointure 6:gauche = {(<$p1> , <$m1>) , (<$p1> , <$m2>) , (<$p2> , <$m2>)}

jointure 6:droite = {<$o1> , <$o2>}

jointure 4:gauche = {<$o1> , <$o2>}

jointure 4:droite = { }

transition t1:gauche = {(<$p1> , <$m1>, <$o1>) , (<$p1> , <$m2>, <$o2>) , (<$p2> , <$m2>, <$o2>)}

transition t1 :droite = { }

Toutes les autres mémoires sont vides.

Ainsi nous pouvons dire que seule la transition t1 est franchissable et avec trois n-uplets de jetons différents.

Lorsque le joueur franchit une transition, des jetons sont déposés dans les places de sortie et des jetons sont enlevés des places d'entrée de la transition. Dans le graphe, cela se traduit par le dépôt de jetons estampillés "+" dans les noeuds correspondant aux places marquées et de jetons estampillés "-" dans les noeuds correspondant aux places démarquées.

Les estampilles des jetons propagés dans le graphe ne sont pris en compte que dans les réseaux de jointures. Les noeuds de jointures utilisent ces estampilles pour déterminer les modifications à apporter à leurs mémoires internes. Lorsque l'estampille est égale à "+", le jeton ou l'instance partielle de transition est stocké dans l'une des mémoires internes du noeud. Et lorsque l'estampille est égale à "-", les instances partielles ou totales qui contiennent le jeton sont supprimées.

Supposons par exemple que nous décidons de franchir la transition t1 avec le n-uplet (<$p1> , <$m1>, <$o1>).

Dans ce cas nous introduirons :

1. dans le noeud p1 le jeton <$p1> estampillé "-"
2. dans le noeud p2 le jeton <$m1> estampillé "-"
3. dans le noeud p3 le jeton <$o1> estampillé "-"
4. dans le noeud p4 le jeton <$p1, $m1, $o1> estampillé "+"

Ce qui produira les modifications suivantes :

jointure 5:gauche = {(<$p2>)}	jointure 5:droite = {(<$m2>)
jointure 6:gauche = {(<$p2> , <$m2>)}	jointure 6:droite = {(<$o2>)}
jointure 4:gauche = {(<$o2>)}	jointure 4:droite = { }
transition t1:gauche = {(<$p2>,<$m2>, <$o2>)}	transition t1:droite = { }
transition t2:gauche = {(<$p1, $m1, $o1>)}	transition t2:droite = { }

Toutes les autres mémoires sont vides.

Dans ce nouvel état, seules les transitions t1 et t2 sont franchissables.

III.4. Comparaison avec la compilation classique

La simplification apportée par la compilation classique est mauvaise [1]. L'utilisation du contrôle et des relations entre les objets (sur le point d'être étudiée en vue d'une amélioration de la compilation des systèmes de règles) est perdue. L'utilisation des réseaux de Petri pour formaliser la description du contrôle rend ce type d'approche plus systématique. Un point important à noter est la simplification de certaines jointures résultant de l'utilisation de la structure de contrôle et des relations dynamiques entre les objets. Cette simplification est très importante car le temps de propagation des objets dans les jointures est beaucoup plus important que celui de leur propagation dans les arbres de tests.

III.5. Comparaison avec le "joueur" de réseau de Petri classique

Comme nous l'avons vu plus haut, la structure du graphe compilé correspond exactement à la structure graphique du réseau de Petri. Dans le cas du "joueur" de réseau de Petri classique comme dans le cas de la compilation respectant la structure de contrôle, on travaille à partir de la liste des transitions franchissables et de la structure de contrôle.

En dépit de cela, les deux approches sont différentes. La deuxième approche permet un processus de construction de la liste des transitions franchissables totalement incrémentale. Cela est possible à cause des mémorisations qui sont réalisées dans les mémoires des jointures. Ce qui n'est pas le cas de l'approche basée sur le concept du "joueur" de réseaux de Petri classique. On peut noter que la compilation traite les places d'entrée des transitions d'une manière totalement symétrique, ce qui résout le problème du choix des places de déclenchement et des places pivots [5].

IV APPLICATION A LA SUPERVISION DES ATELIERS FLEXIBLES

IV.1. Prise en compte de variables globales

L'élaboration d'un système de commande d'ateliers flexibles implique une connaissance du processus de fabrication (gamme de fabrication, ressources, etc.) et l'utilisation de règles de décision. Dans le cas d'un système réactif c'est-à-dire qui interagit en temps réel avec son environnement, la liste des instances de transitions franchissables (qui est disponible à tout instant) est utilisée pour rechercher les instances effectivement utilisables pour effectuer des franchissements (prise en compte des réceptivités des transitions). Les systèmes de règles contrôlés par réseaux de Petri, tel qu'ils ont été présentés, ne permettent pas la représentation de règles de décision comprenant une connaissance globale de l'atelier (une règle ne peut comprendre que les instances d'objets contenus dans les places d'entrée). Au niveau de la compilation ces variables seront prises en compte en introduisant un nouveau type de jointure. Ces jointures seront insérées en fin des réseaux de jointure classiques, juste avant les noeuds représentant les transitions. De cette façon, leur mémoire gauche contiendra la liste des n-uplets utilisables pour un franchissement (sans prise en compte des variables globales). La mémoire droite sera une variable booléenne représentant le résultat d'une fonction logique portant sur les variables globales. Ainsi, une copie du contenu de la mémoire gauche sera transmis au successeur dès que la variable est mise à vrai. On dispose alors des n-uplets effectivement utilisables pour effectuer des franchissements (prise en compte des réceptivités).

De façon plus générale, notre approche permet d'ajouter aux systèmes de règles contrôlés par réseaux de Petri un système de règles de production ordinaire. La liaison entre les deux sera faite à travers ces nouvelles jointures. On remarque que le seul changement est l'introduction de branches supplémentaires dans le but de prendre en compte les règles de la logique propositionnelle.

IV.2. Prise en compte explicite du temps

Le temps est une variable très importante pour la supervision des ateliers flexibles. Elle est utilisée pour représenter des durées, des délais et même des "watchdogs".

Au niveau du modèle, le temps sera pris en compte en associant une temporisation aux transitions[1].

Cela se traduit, au niveau de la compilation par la définition d'un nouveau type de noeud transition. La différence essentielle entre ce type noeud et le précédent réside à l'ajout d'un attribut *temporisation* contenant la durée de la temporisation associée à la transition.

Ces modifications vont entraîner une modification de l'algorithme de l'interpréteur du graphe. En effet, lorsque le joueur franchira une transition, avant de déposer un jeton dans une place d'entrée de la transition, il lui associera deux estampilles :

- l'estampille "+" pour dire qu'il s'agit d'un marquage,
- et une e*stampille temporelle*", dont la valeur représente la date courante.

Ces deux estampilles ne seront pris en compte qu'aux niveaux des jointures et des transitions pour les modifications éventuelles des mémoires.

Au niveau de chaque jointure, avant de transmettre un n-uplet vérifiant la relation de jointure au noeud suivant, on va lui associée une estampille temporelle dont la valeur est égale à *Sup(Estampillle_1, Estampillle_2)*. *Estampillle_1* et *Estampillle_2* représentant les estampilles temporelles respectives des deux jetons (ou n-uplets) qui ont servis à construire ce n-uplet.

Cette nouvelle estampille donne exactement la date réelle de validité de la nouvelle instance partielle ou totale qu'on vient de construire.

Au niveau des transitions, avant de mémoriser un n-uplet dans sa mémoire gauche, on va lui associée une estampille temporelle dont la valeur est égale à *Estampillle + Temporisation*. *Estampillle* et *Temporisation* représentant respectivement l'ancienne estampille temporelle du n-uplet et la temporisation associée à la transition.

On obtient ainsi, au niveau de chaque transition un échéancier définissant tous ses instants de franchissements possibles.

IV.3. Prise en compte de règles de priorité

Les règles de priorité ne peuvent pas être présentées avec la syntaxe décrite ci-dessus. Cependant elles sont indispensables dans la commande des ateliers flexibles. Une solution pour l'utilisation de ces règles a été proposée dans [16]. Il s'agit d'ordonner les jetons dans les places en respectant les règles de priorité établies. Cette approche peut être transposée ici en mettant en ordre le contenu des mémoires des jointures. L'*ensemble des conflits* sera, donc, ordonné et la décision prioritaire sera la première transition avec le premier n-uplet.

IV.4. Prise en compte de connaissances imprécises

L'introduction de connaissances imprécises est étudiée également par d'autres auteurs [4, 18]. La notion de marquage imprécis permet de travailler avec des faits mal connus. Dans un marquage imprécis, une certaine instance d'objet peut être localisée en plusieurs endroits. La construction des arbres de tests et des réseaux de jointures n'est pas modifiée. Cependant, la propagation des jetons peut suivre des règles plus complexes. Lorsque l'imprécision augmente à travers l'application d'une règle, des instances d'objets sont insérées dans les arbres de

tests sans que d'autres ne soient enlevées des réseaux de jointures. Si la connaissance devient plus précise, on retire des n-uplets des réseaux de jointures, sans les propager dans les arbres de tests.

V CONCLUSION

Pour la commande des ateliers flexibles, il est nécessaire d'utiliser en même temps des règles de production et un modèle de représentation du contrôle. Les systèmes de règles de production contrôlés représentent une bonne solution. Cet article présente une approche pour appliquer ces systèmes d'une façon efficace.
Notre travail se borne maintenant à l'élaboration d'un prototype de machine d'inférence spécialisée comme un exemple d'application. En même temps, nous considérerons des modèles qui permettent de prendre en compte non seulement le fonctionnement normal du système, mas aussi certains types de fonctionnements anormaux et des pannes qui rendent imprécise l'information utilisée pour prendre les décisions.

REMERCIEMENTS

Ce travail a été réalisé sous la direction de Monsieur Robert Valette, Directeur de recherche au C.N.R.S. pendant mon séjour au Laboratoire d'Automatique et d'Analyse des Systèmes du CNRS.

BIBLIOGRAPHIE

[1] B. Bako: *Mise en oeuvre et Simulation du niveau coordination de la commande des ateliers flexibles: Une approche mixte réseaux de Petri et Systèmes de règles*, Thèse de Doctorat de l'Université Paul Sabatier, Toulouse, Octobre 90.

[2] R. Bauman, T.A. Turano: *Production based language simulation of Petri nets*, Simulation Vol. 47, Numéro 5, P.191-198 (Nov 1986).

[3] D.S. Barbalho: *Conception et mise en oeuvre de la fonction coordination pour une commande distribuée d'atelier*, Thèse de Doctorat de l'Université Paul Sabatier, Toulouse, (Déc 1987)

[4] J. Cardoso, R. Valette, D. Dubois: *Petri nets with uncertain markings*, 10th Int. Conf. on Application and Theory of Petri Nets, Paris, 27-29, juin 1990.

[5] J.M. Colom, M. Silva, J.L. Villarroel: *On software implementation of Petri nets and colored Petri nets using high-level concurrent languages*, 7th European Workshop on Application and Theory of Petri nets, Oxford, July 1986.

[6] H. Fleischack, A. Weber: *Rule based programming, Predicate/Transition nets and the modeling of office procedures and flexible manufacturing systems*,10th Int. Conf. on Application and Theory of Petri Nets, Paris, 27-29, juin 1990.

[7] C. Forgy: *RETE: a fast algorithm for many pattern/ many object pattern match problem*, Artificial Intelligence, 19:17–37, (1982)

[8] H.J. Genrich: *Predicate/Transition nets*, Lecture notes in computer sciences 254 Springer Verlag P. 207–247 (1986)

[9] M.P. Georgeff: *Procedural control in production systems*, Artificial Intelligence, 18:175–201, (1982)

[10] J. Martinez and al.: *Merging artificial intelligence and Petri nets for real–time scheduling and control of production systems*, 12th IMACS World Congress in Scientific Comp., PARIS, July, Vol. 3, P. 528–531 (1988).

[11] T. Murata, D.Zhang: *A predicate–Transition net model for parallel interpretation of logic programs*, IEEE Trans. on software. eng., Vol. 14, numéro 1, (April 1988)

[12] H. Phillipe: *Algorithmes pour la compilation de bases de connaissances en logiques propositionnelle et du premier ordre – les systèmes KHEOPS et CLOPS*, Thèse de Doctorat de l'Université Paul Sabatier, Toulouse, (Mai 1989)

[13] M. Rillo: *Using Petri nets and rule based system in manufacturing systems*, 12th IMACS World Congress on Scientific Comp., PARIS, p.III–535–537, July 1988).

[14] C. Sibertin–Blanc: *High–level Petri Nets with data structures*, 6th European Workshop on Application and Theory of Petri Nets, Helsinki, Finland, June 85.

[15] R. Valette: *Nets in production systems*, Lecture Notes in Computer Science 255 Springer Verlag p. 191–217 (1986)

[16] R. Valette and al.: *Petri nets and production rules for decision levels in FMS control*, 12th IMACS World Congress on Scientific Comp., PARIS, p.522–524, July 1988).

[17] R. Valette, D. Dubois, J. Cardoso.: *Représentation de l'état d'un atelier de fabrication avec prise en compte des incidents*, In Congrès AFCET Automatique 1988, Grenoble, France, (Oct. 1988)

[18] R. Valette, J. Cardoso, D. Dubois.: *Monitoring manufacturing systems by means of imprecise markings*, IEEE Int. Symp. on Intel. Control, Albany N.Y., USA, (25–24 Sept. 1989)

[19] R. Valette, B. Bako: *Software Implementation of Petri nets and Compilation of Rule–Based systems*, Eleventh Int. Conf. on Application and Theory of Petri Nets, Paris, 27–29, juin 1990.

[20] M.D. Zisman : *Use of productions systems for modelling asynchronous concurrent processes*, In patern directed inference systems D.A. Watterman and F. Hayes–Roth (Eds), Academic Press, London, p.53–68 (1978)