Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Khider Biskra

Faculté des Sciences et des Sciences de l'ingénieur

Département d'Informatique

N°d'ordre :……
Série :……

Mémoire

Présenté en vue de l'obtention du diplôme de **Magister en Informatique**

Option**: Intelligence Artificielle et Systèmes Distribués**

:

Modélisation et Vérification des systèmes de production distribués à l'aide d'UML et des réseaux de Petri

Par :

M^{elle} Meliouh Amel

Soutenu le : / /2009

Devant le jury :

Mr. Benmohamed Mohamed, Pr. Université de Constantine	Président
Mr. Kazar Okba, M.C. Université de Biskra.	Examinateur
Mr. Babahenini Med Chaouki, M.C. Université de Biskra	Examinateur
Mr. Chaoui Allaoua, M.C. Université de Constantine	Rapporteur

REMERCIEMENTS

Le travail présenté dans ce mémoire a été dirigé par le maître de conférences à l'université de Constantine ; Mr. **Chaoui Allaoua**. Je suis honorée et il m'est agréable de lui adresser mes remerciements les plus sincères. De qui j'ai tiré un immense profit de son savoir ; par ses conseils éclairés et sa vision objective et rationnelle de la problématique ainsi que sa bienveillante disponibilité. Ainsi, je lui exprime ici ma profonde reconnaissance pour la confiance qu'il m'a témoignée en m'acceptant pour l'encadrement.

J'adresse mes sincères remerciements à Mr. **Kerkouche Elhilali,** enseignant à l'université d'Oum El Bouagui, pour son aide précieux .Qu'il trouve ici l'expression de mon respect et de ma profonde reconnaissance.

Je tiens tout particulièrement à exprimer ma plus profonde gratitude à Mr. **Moustari Adel**, Mme. **Taalah Latifa**, Melle. **Beribeche Fethia**, qui n'ont jamais cessé de m'aider et qui m'ont été un soutien indispensable dans des moments quelquefois difficiles. Que ces trois personnes trouvent ici le témoignage de mes remerciements.

J'adresse mes vifs et sincères remerciements à Mr. **Babahenini Med Chaouki**, le chef du département d'informatique. Mr. **Kazar Okba**, maître de conférences au département d'informatique, de Biskra pour l'intérêt qu'ils ont porté à ce travail et pour leurs soutiens moraux.

Je voudrais également remercier tous les membres de jury, pour l'honneur qu'ils m'ont fait en acceptant d'examiner ce travail.

Mes sincères remerciements vont également à Mr. **Bennoui Hammadi**, enseignant au département d'informatique de Biskra, Mme. **Wafa Saadi**, étudiante en post-graduation à l'université de Biskra, Mr. **Mechkour Houari** maître assistant à l'école Centrale d'Electronique (ECE) de Paris, et Mr. **Wonham.W.M**, Professeur au département d'informatique Université de Toronto, Canada ;Pour leurs aides documentaires .

Je remercie tous les enseignants du département d'informatique et notre charmante équipe administrative, pour leur grande gentillesse. Un remerciement particulier à **Mme. Saouli Rachida**, **Mr.** pour son aide dans la lecture de ce mémoire. Un grand merci va également à Mr. **Rezeg Khaled** pour son encouragement.

Finalement, un grand merci à tous mes collègues d'études, pour leur témoignage d'amitié et pour l'ambiance de travail qu'ils ont su créer.

SOMMAIRE

TABLE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GENERALE

Contexte

Durant les trois dernières décennies, les systèmes de production ont subi des changements notables qui ont modifié leurs modes de gestion. Cette gestion est devenue un problème, qui consiste à gérer de façon intégrée les différentes fonctions du système, et ceci de façon à maximiser ses performances en fonction des objectifs établis [Habchi ,2001].

Un système de production est constitué d'un ensemble de processus visant à produire des éléments. Ces processus sont composés d'un ensemble de phases de production, correspondante chacune à une activité ou à un ensemble d'activités, permettant de transformer les éléments dans le processus de production [Toshic, 2006]. Avant, le but d'un système de production était limité à exécuter une séquence de processus pour produire un seul type de produit. Actuellement, le besoin d'offrir une variété de produits, avec une qualité optimale et dans un cycle de production très court, a augmenté les exigences du système de production pour supporter l'exécution parallèle de ses processus.

L'impératif de cette exécution, a conduit à une augmentation de la complexité des systèmes de production, ce qui a entraîné un problème de fiabilité et de sûreté de fonctionnement. Le but est donc d'élaborer des procédures de supervision et de diagnostic au dessus de la couche de commande du système de production, en vue de garantir les objectifs de sécurité, de synchronisation, de fiabilité et de disponibilité.

Un autre facteur de complexité des systèmes de production est qu'ils sont considérés comme des systèmes à événements discrets (DES). Ces systèmes sont composés d'un ensemble d'éléments indépendants, dont l'évolution dynamique est dirigée par des événements asynchrones, qui apparaissent dans un intervalle fini de temps, provoquant une transition discrète de l'état du système. Cette classe de systèmes présente certaines caractéristiques comme le parallélisme, la synchronisation, et les conflits,…etc, dues au manque de dépendances de temps en plus des séquences d'événements qui ne sont pas prédéfinies à l'avance [Toshic, 2006]. Et comme les systèmes de production sont des systèmes à événements discrets, ils exigent dans ce cas une stratégie de supervision très complexe [Ramadge, 1987].

La supervision des systèmes de production peut être définie comme une suite d'opérations qui a pour objet de détecter et de localiser les défauts et les modes de fonctionnement anormaux et de les diagnostiquer ou les éviter [Kechida, 2005]. Partant d'une spécification de la structure et du comportement du système physique et des objectifs à atteindre, un superviseur de contrôle pour un système de production consiste à spécifier une stratégie de supervision. Il doit être capable d'autoriser, d'inhiber et de synchroniser les opérations du système de production en suivant l'occurrence des événements dans le système physique.

Le coût de test des différentes structures et stratégies de supervision sur le processus de production réel, rend indispensable le passage par une étape de conception d'un superviseur de contrôle pour ces systèmes. Cette conception est basée sur l'utilisation de modèles, qui doivent présenter la structure et le comportement des systèmes à superviser et permettre l'analyse de leurs propriétés. Pour cela, la modélisation de la partie physique (partie opérative) de ces systèmes doit être effectuée au préalable.

Les formalismes les plus connus pour la modélisation des DES, sont les automates d'états finis, les réseaux de Petri, la logique temporelle etc. Plusieurs recherches ont été entreprises dans ce domaine. La technologie multi-agents a été utilisée pour le contrôle des systèmes de production distribués [Kis, 1996], l'approche utilise des agents intelligents pour la modélisation du système. La coordination et la coopération entre ces agents sont réalisées via un mécanisme de passage de messages en vue de la supervision du système. UML et le modèle de checking sont utilisés dans [Flake, 2002], l'approche consiste à modéliser le système de production par UML et vérifier ses propriétés par le modèle de checking. Les réseaux de Petri représentent le formalisme le plus utilisé pour les DES, ils sont alors utilisés pour la modélisation des systèmes de production [Uzam, 2006] et convertis à une méthodologie basée sur la logique de passage des jetons (TPLL) afin d'assurer la supervision du système. Les automates d'états finis sont utilisés pour concevoir un superviseur de contrôle pour un système de production [Quedraogo, 2006], la conception est basée sur la théorie de supervision (SCT) proposée par Ramadge et Wonham. Ainsi, les réseaux de Petri temporisés sont appliqués pour modéliser et vérifier des systèmes de production en vue de leur supervision [Gradisar, 2007].

Objectifs

En pratique dans la modélisation des systèmes à événements discrets, ces techniques formelles souffrent du problème d'explosion d'états, c'est-à-dire le nombre d'états grandit de façon exponentielle avec la taille du système à modéliser. Il est alors nécessaire d'avoir recours aux approches modulaires, qui consistent à identifier les modules ou les composants d'un

système, en modéliser chacun séparément, pour faciliter la conception de ces systèmes qualifiés de complexes et les rendre plus compréhensibles. Pour cela plusieurs chercheurs ont préconisé un changement de paradigme vers l'orienté objet [Bordbar, 2000]. Dans ce cadre, UML est le langage standard de modélisation le plus utilisé. C'est un langage visuel de modélisation d'une application, qui aide le concepteur durant le cycle de vie de conception, depuis la description fournie par les utilisateurs ou les experts vers le logiciel final. Dans [Bordbar, 2000], le travail présenté a proposé une méthode de conception manuelle d'un contrôleur à événements discrets vérifiable pour un processus de production. Cette méthode est basée sur UML pour la modélisation statique et sur les réseaux de Petri pour la modélisation dynamique et la vérification des propriétés; une simulation de la méthode a été réalisée par Matlab. En se basant sur ce travail nous proposons une approche de conception automatique d'un superviseur de contrôle pour un processus de production distribué. Cette conception est basée sur la modélisation par UML, et la vérification des propriétés par les réseaux de Petri.

Malgré son succès en étant un langage de modélisation, UML souffre du manque de capacité de vérification et d'analyse [Zhao, 2004]. Beaucoup de modèles ne peuvent réellement être vérifiés en détail, en particulier s'ils sont utilisés pour décrire des systèmes complexes et distribués. Par contre, les réseaux de Petri sont un modèle à événement discret [Ramadge, 1987]. Leur théorie permet d'analyser les caractéristiques des DES telle que la synchronisation, la concurrence, les conflits, le partage de ressources, les blocages..etc. Dans ce cas, nous signalons clairement que les transformations entre UML et les réseaux de Petri sont significatives pour l'analyse et la vérification du modèle UML.

Il s'agit dans notre approche de conception, d'une part de modéliser la structure et le comportement d'un processus de production distribué, ainsi que d'effectuer une vérification complète de ses propriétés. D'autre part notre approche permet la génération d'un graphe présentant l'enchaînement des activités à suivre afin d'assurer la sûreté de fonctionnement de ce processus. Le passage de la modélisation par UML vers le formalisme réseau de Petri est assuré par une technique de transformation de graphes.

Structure du document

Pour pouvoir procéder à une conception automatique d'un superviseur de contrôle pour un processus de production distribué, il nous faut choisir un outil de modélisation UML. Il nous faudra également choisir un outil de transformation de graphes et un autre pour vérifier les propriétés du système. Ces outils sont nécessaires puisque notre contribution concerne la modélisation de la partie opérative (partie physique) d'un processus de production, puis la

modélisation comportementale du processus, ensuite la vérification des propriétés comportementales et la génération d'un graphe d'enchaînement des activités selon une stratégie de supervision.

Pour obtenir une transformation plus générale s'approchant entre UML et les réseaux de Petri, nous avons fait des recherches sur la transformation au niveau méta-modèle. Et pour atteindre un processus de transformation automatique et correct, nous utilisons alors des grammaires de transformation de graphes.

L'outil choisit pour cela est ATOM^3 ; car il permet de supporter la multi-modélisation et les transformations de graphes en se basant sur une grammaire. Il est alors nécessaire dans ce cas, de comprendre comment cet outil génère automatiquement des modèles UML, et les transforme vers leur équivalent réseau de Petri.

Pour avoir une vérification complète et automatique des propriétés du modèle réseau de Petri généré, il nous faut choisir un outil adapté pour ce formalisme. L'outil le plus utilisé et le plus maîtrisé est INA .Une autre motivation pour le choix de cet outil, est qu'il nous offre une possibilité, d'illustration du fonctionnement du superviseur de contrôle par un graphe généré automatiquement.

Pour cela ce mémoire est organisé en trois chapitres. Le premier chapitre, est un état de l'art présentant une étude des systèmes à évènements discrets tout en abordant les systèmes de production. Une attention particulière est portée à la modélisation et la vérification des systèmes de production distribués en vue de concevoir un superviseur de contrôle pour ces systèmes.

Une étude progressive de notre approche de conception du superviseur de contrôle est présentée dans le deuxième chapitre, qui consiste à étudier en détail les étapes de cette conception.

Le troisième chapitre est une étude de cas, où une application de notre approche sur deux exemples de processus de production distribués est procédée. Le premier exemple est une chaîne d'emballage des produits et le deuxième est une chaîne d'embouteillage de l'eau minérale.

La dernière partie, conclut le mémoire et résume ce que nous avons réalisé dans notre contribution ainsi que les travaux futurs, inspirés de l'approche proposée.

ETAT DE L'ART : Modélisation et vérification en vue de la supervision des systèmes de production

Les systèmes de production, à l'initiative de l'homme, sont caractérisés par une forte complexité et flexibilité ; ils sont considérés comme des systèmes à événements discrets. L'un des objectifs des systèmes de production est d'avoir une productivité optimale. Optimiser la productivité signifie, avoir une sûreté de fonctionnement. Le but est donc d'élaborer des procédures de supervision au dessus de la couche de commande du système de production, en vue de garantir ses objectifs. Ces procédures de supervision sont basées sur la structure du système. Donc La modélisation du système de production est une étape initiative pour la supervision ainsi que la vérification de ses propriétés.

Ce chapitre présente un état de l'art sur la modélisation et la vérification des systèmes de production en vue de leur supervision. Il se décompose en quatre parties. Dans la première partie, le concept de systèmes dynamiques à événements discrets (DES), sera présenté. Une description des systèmes de production fera l'objet de la deuxième partie. La commande par supervision, qui repose sur la théorie de Ramadge et Wonham, sera étudiée dans la troisième partie. Les potentialités de modélisation et de vérification des systèmes de production seront abordées dans la quatrième partie. Dans la conclusion, il sera tenté de dégager quelques éléments de réflexion sur cet état de l'art.

1.1 Systèmes à événements discrets

1.1.1 Définition

Les Systèmes Dynamiques à Evénements Discrets (DES) sont une classe de systèmes dynamiques, et complexes.

Par opposition aux systèmes dynamiques continus, ils satisfont généralement les propriétés suivantes :

- L'espace d'état est décrit par un ensemble discret. c'est à dire le changement d'états est causé par l'occurrence des évènements à des instants non prédictibles. Dans ce cas, chaque

évènement peut être dépendant d'autres évènements et la fin d'une opération déclenche d'autres opérations. [Ferrier, 2004][Uzam, 1998].

- Les transitions d'état apparaissent seulement à des instants discrets du temps ; ces transitions d'état sont associées à des événements [Ferrier, 2004].
- Le parallélisme, c'est à dire plusieurs opérations peuvent être exécutées en même temps.
- Les opérations sont asynchrones, à l'opposé des autres systèmes où chaque changement est synchronisé par une horloge globale, dans les DES les évènements arrivent de manière asynchrone.
- Le non déterminisme, l'occurrence des évènements est non déterminée à priori, c'est à dire les différentes évolutions peuvent être possibles d'un état donné. [Uzam, 1998].

1.1.2 Exemple de Systèmes a événements discrets

L'exemple présenté concerne une souris qui se déplace de manière spontanée (elle agit sans intervention extérieure) à l'intérieur d'un labyrinthe.

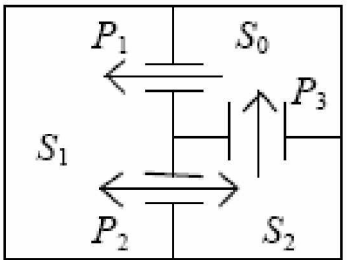

Figure 1.1 Exemple d'un Systèmes a événements discrets [Ferrier, 2004].

Les salles **Si** communiquent par des portes unidirectionnelles $\mathbf{P_1}$ et $\mathbf{P_3}$ et bidirectionnelle $\mathbf{P_2}$. On note par **"pi"** l'événement : "la souris passe par la porte **Pi**".

Soit $\sum$ l'ensemble des événements : $\sum = \mathbf{\{p_1, p_2, p_3\}}$.

On conçoit que les situations (ou comportements) possibles sont :

- La souris est dans la salle $\boldsymbol{S_0}$.
- La souris est dans la salle $\boldsymbol{S_1}$.
- La souris est dans la salle $\boldsymbol{S_2}$.

Ce qui définit trois états différents, relatifs aux possibilités d'occupation des salles.

Le passage de l'état "souris en $\mathbf{S_0}$" (état **A**) à "souris en $\mathbf{S_1}$" (état **B**) a lieu sur l'occurrence de l'événement $\mathbf{p_1}$. On peut alors construire le diagramme des états (Figure 1.2). L'espace d'états s'écrit **X = {A, B, C}.**

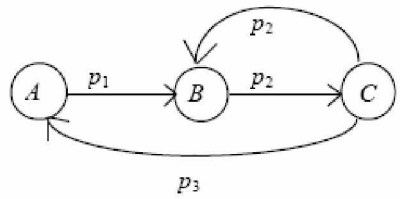

Figure 1.2 Diagramme des états pour l'exemple de la Figure 1.1

La théorie des DES peut être divisée actuellement en deux grandes approches :

- **Approche logique** qui ne s'intéresse qu'à l'occurrence des événements ou l'impossibilité de cette occurrence (*"impasse"* ou *"blocage",* en Anglais *"deadlock"*) et à la succession de ces événements, mais pas à la date précise de ces occurrences, autrement dit pas aux aspects de performances ;cette approche est initiée par W.M. Wonham et P. Ramadge, et après eux de nombreux auteurs, ont étendu à la théorie des Automates et des Langages Formels la problématique de la commande, qui agit dans ce cas sur l'inhibition de certaines transitions d'état pour éviter les comportements non désirés .
- **Approche quantitative** qui s'adresse à l'aspect "évaluation de performance" (mesurée par le nombre d'événements survenant dans un laps de temps donné), voire à l'optimisation des performances.

1.1.3 Systèmes continus, systèmes à événements discrets

Des grandeurs telles que la position, la vitesse, l'accélération, le niveau, la pression, la température, le débit, la tension, le courant,...etc, sont des variables continues, dans le sens qu'elles peuvent prendre n'importe quelle valeur dans R lorsque le temps, lui-même est "continu", évolue.

Les grandeurs telles que le nombre de produits dans un stock, le nombre de processeurs en activité, ...etc sont discrètes. L'évolution est conditionnée par l'occurrence d'événements, à "certains instants", tels que la fin d'exécution d'une tâche, le franchissement d'un seuil devant entraîner une action, l'arrivée d'un produit, d'un client, la défaillance d'un dispositif, ...etc. [Ferrier, 2004].

Donc, on peut dire qu'un Système à Evénements Discrets est un système à espace d'état discret dont les transitions entre états sont associées à l'occurrence d'événements discrets asynchrones. Par contre, le comportement d'un système continu à l'instant **t** peut être résumé par son état qui représente la mémoire minimale du passé nécessaire à la détermination du futur.

1.1.4 Différents modèles pour les DES

Un modèle est une approximation, une vue partielle plus ou moins abstraite de la réalité afin de l'appréhender plus simplement, selon un point de vue et qu'il est établi pour un objectif donné. Il peut être exprimé par des mathématiques, des symboles, des mots, des graphes,...etc.

Il existe trois formalismes de modélisation et d'analyse des DES, les réseaux de Petri, l'algèbre des dioïdes, les langages et automates [Ferrier, 2004].

1.1.4.1 Réseaux de Petri

Les réseaux de Petri (RdP), sont un outil graphique à support mathématique permettant de modéliser, visualiser et analyser des évolutions de ces systèmes [Ferrier, 2004].

Ils sont utilisés pour modéliser, analyser et concevoir des systèmes à évènements discrets, incluant les systèmes de production, les systèmes informatiques, les systèmes de communication…etc. Les réseaux de Petri présentent également deux caractéristiques principales : Premièrement, ils permettent la modélisation comportementale comprenant le parallélisme, la synchronisation et le partage de ressources. Deuxièmement, Le modèle réseau de Petri peut être utilisé pour implémenter les systèmes de contrôle en temps réel pour ces systèmes [Uzam, 1998].

L'état du système modélisé par un RdP est représenté par un vecteur de marquage définissant le nombre de jetons que contient chaque place. L'évolution de l'état (dynamique du système) correspond donc à une évolution du marquage. L'évolution du marquage se produit par le franchissement de transitions: à l'occurrence d'un événement correspond le franchissement d'une transition. L'espace des marquages accessibles tient lieu d'espace d'états dans le système.

1.1.4.2 Algèbre des dioïdes

La structure algébrique de dioïde permet de modéliser et d'évaluer les performances de certains systèmes à événements discrets. C'est une structure algébrique munie d'une addition associative, commutative, avec élément neutre ("zéro"), et d'une multiplication associative, avec élément neutre ("un"), la multiplication étant distributive par rapport à l'addition, le zéro étant absorbant pour la multiplication (zéro fois *a* égale zéro pour tout *a*) ; et l'addition est idempotente, c'est-à-dire que *a* plus *a* égale *a* pour tout *a.* Elle est basée essentiellement sur les graphes d'événements temporisés qui sont une sous-classe des réseaux de Petri, avec des équations d'état linéaires [Ferrier, 2004].

A la différence du modèle RdP, l'état considéré ici pour aboutir à cette représentation linéaire est associé non plus aux places mais à leurs transitions. Le temps, associé aux événements, est intégré de manière naturelle à cette classe de modèles. L'état du système est modélise dans ce cas par un ensemble d'équations linéaires plus un graphe d'événements [Uzam, 1998].

1.1.4.3 Langages et Automates

Les langages et les automates permettent de traiter mathématiquement les problèmes relatifs aux DES, essentiellement d'un point de vue logique. La théorie des automates à états finis a été principalement développée avec la théorie des langages. Ces modèles reviennent à spécifier des ensembles d'états et des transitions entre ces états.

Chaque DES a un ensemble d'événements qui lui est associé, ces événements font l'évoluer dans le temps. Cet ensemble peut être vu comme un alphabet d'un langage et les séquences d'événements sont des mots (aussi appelés chaînes) de ce langage. Un automate est alors un dispositif qui engendre un langage en manipulant l'alphabet (les événements). [Ferrier, 2004].

1.1.5 Théorie de supervision pour la commande des DES

Les systèmes à événement discrets sont des systèmes qui sont caractérisés par les séquences d'évènements qu'ils peuvent accepter ou exécuter [Cantarelli, 2006].

L'objectif d'une commande est d'imposer au procédé un comportement spécifié, tout en respectant un ensemble de contraintes [Ferrier, 2004].

Le contrôle de ces systèmes est initié par Ramadge et Wonham, qui ont proposé une théorie de supervision (SCT). Le travail est développé autour des langages réguliers formels générés par des automates d'états finis [Pinzon, 1997].

1.1.5.1 Définitions

La théorie de supervision pour un Système à événements Discrets consiste à proposer des algorithmes permettant de définir automatiquement un modèle de contrôle et de commande, à partir de modèles formels de la dynamique d'un procède à contrôler [Wonham, 2008].

Dans cette théorie, le système à événements discrets à contrôler est modélisé par un système de transitions étiquetées **P** (plant model), appelé le générateur. Les étiquettes indiquent le phénomène physique qui a causé la transition [Pinzon, 1997].

Le comportement du DES est modélisé par un langage formel **L** généré par le système de transitions. L'alphabet du langage $\sum$ est un ensemble d'évènements. Ces évènements sont

identifiés à l'aide d'une spécification $\mathbf{S_p}$. La spécification est une description du comportement voulu du DES, elle est décrite sous forme d'un langage formel, généré par un système de transitions **H**.

La caractéristique du contrôle réside dans le fait que certains événements (transitions) peuvent être empêchés par un contrôleur externe. Ce sont des évènements contrôlables et les autres sont des évènements non contrôlables

Un contrôleur **C** est construit à partir d'un système de reconnaissance **G** et le système de transitions **H** [Wonham, 2008].

1.1.5.2 Démarche de conception d'un superviseur de contrôle

Soit les générateurs **G** et **H** définie chacun comme un 5-tuplet: $\mathbf{(Q,\sum,\delta,q_0,q_m);}$

- **Q:** est un ensemble d'états ;
- $\sum$ est un ensemble fini d'alphabet interprété comme l'ensemble des évènements dans le DES ;
- $\mathbf{\delta: \sum \times Q \rightarrow Q}$ est une fonction de transition **;**
- $\mathbf{q0 \in Q}$ **:** est un état initial ;
- $\mathbf{q_m \in Q}$**:** est un état final.

Le DES sera modélisé via un ensemble de sous générateurs **G1, G2…….Gn**, présentant chacun un sous système du DES. Les actions de ces générateurs sont asynchrones et indépendantes, donc leurs alphabets sont disjoints et toute interaction entre ces sous générateurs est exprimée dans la spécification $\mathbf{S_p}$ [Pinzon, 1997].

La démarche de conception d'un superviseur de contrôle pour un DES consiste en six étapes :

1- Modéliser les sous générateurs pour l'ensemble des sous systèmes du DES.

2- Construire le générateur **G = G1 ||G2 … ||Gm**. G par une concaténation de l'ensemble des sous générateurs **Gi.**

3- Construire les sous générateurs de spécifications **Hj**.

4- Augmenter les spécifications **Hj**, par des boucles étiquetées par les évènements présents dans **G** mais ne sont pas dans **Hj**.

5- Construire la spécification globale $\mathbf{H = H1 \cap H2 \ldots \cap Hm}$. par une intersection des systèmes de transitions **Hj** de l'ensemble des sous générateurs **Gi. H** va forcer le comportement de **G.**

6- Construire l'intersection **E = H ∩ G**. **E** représente le comportement globale du système qui vérifie toutes les contraintes de la spécification.

Le générateur obtenu **E** ne présente pas un superviseur parfait, due au fait qu'il ne vérifie pas les deux propriétés suivantes :

- Le Non blocage, c'est à dire le générateur **E** peut contenir des états de blocage à partir des quelles l'état final ne peut pas être atteindre.
- Contrôlabilité, c'est à dire le langage **L(E)** peut être non contrôlable par apport à **L (G)**. Cela signifie que lorsque **G** et **E** s'exécutent en parallèle, on arrive a un état à partir du quel un évènement non contrôlable est permis dans **G** mais pas dans **E** [Giua, 1992]**.**

Nous devons donc minimiser le comportement de **E**, pour obtenir un générateur contrôlable et non bloquant **S**. Le système **S** obtenu par cette procédure nous donne la structure de transition d'un superviseur de contrôle.

1.1.5.3 Exemple de conception d'un superviseur de contrôle

Nous donnerons ici un exemple de conception utilisant un modèle d'automate d'états finis.

Considérant le système **G1** et **G2** de la Figure 1.3 et la spécification **H.** Nous pouvons penser à **G1** comme un robot qui prend une partie (l'événement **a**) et porte la partie sur une machine (l'événement **b**). **G2** est une machine qui peut commencer à travailler (l'événement **c**) et peut transporter la partie produite à un transporteur **A** ou **B** (respectivement les événements **d** et **e**) avant de retourner à l'état disponible. La spécification que nous considérons, est représentée par le générateur **H**, spécifie que la machine peut commencer à travailler seulement après qu'elle a été chargé (les événements **b** et **c** doivent arriver alternativement). Et que le robot peut charger une nouvelle partie sur la machine seulement après que la partie précédemment chargée a été transportée au transporteur (des événements **d** et **b** doivent arriver alternativement). Nous supposerons que le seul événement incontrôlable est l'événement **b**.

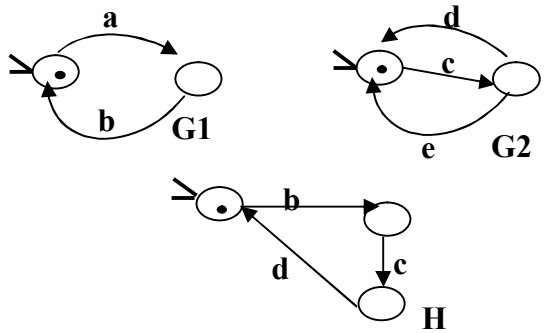

Figure 1.3 Les systèmes et la spécification pour l'exemple [Giua, 1992]**.**

La deuxième étape de conception exige la construction de **G = G1 || G2**, comme il est illustré dans la Figure 1.4.

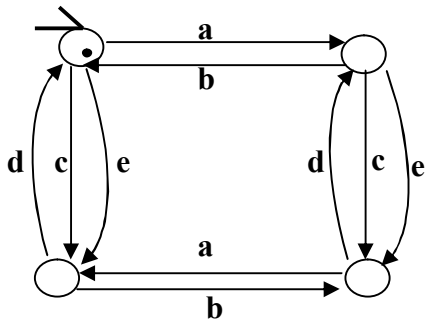

Figure 1.4 Concaténation de G1 et G2 [Giua, 1992]**.**

Dans la troisième étape nous construisons la spécification **H'** augmentée (Figure 1.5), par l'ajout des boucles à **H** avec les événements rencontrés **a** et **e**. La cinquième étape n'est pas nécessaire puisqu'il y a une seule spécification dans cet exemple.

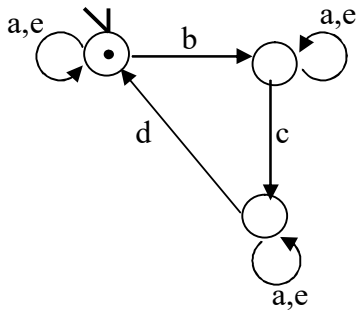

Figure 1.5 Générateur de la spécification H' [Giua, 1992]**.**

Dans la sixième étape, nous construisons le système **E = G ∩ H'** (Figure 1.6). Le système **E** n'est pas un superviseur parfait, car, il contient des états bloquants (depuis les quelles l'état final ne peut pas être accessible), ainsi que des états non contrôlables .Par exemple pour la chaîne d'événements **w = aba**, les évènements **b** et **c** peuvent apparaître dans **G**, mais le modèle de contrôle engendré par **E**, contient seulement l'évènement **c** .Bien que l'évènement **b** est non contrôlable, donc le langage généré par **E** est non contrôlable.

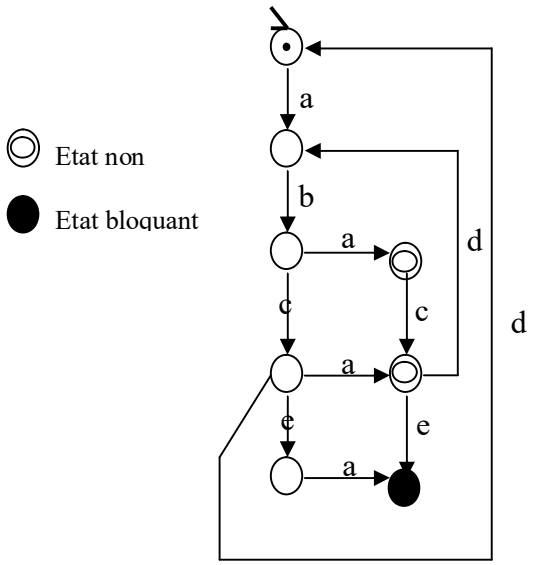

Figure 1.6 Le système E = G ∩ H' [Giua, 1992]**.**

Si nous éliminons les états non contrôlables et les états bloquants, nous obtenons un générateur **S** non bloqué et contrôlable comme il est montré dans la Figure 1.7, qui est un superviseur parfait pour cet exemple.

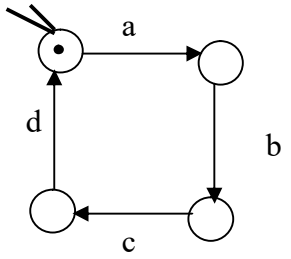

Figure 1.7 Le superviseur de contrôle S [Giua, 1992]**.**

1.2 Systèmes de productions

1.2.1 Définitions

Un système de production est un système qui vise à produire des éléments. C'est un ensemble de processus, chaque processus de production est constitué d'un ensemble de phases de production. Une phase de production est une activité ou un ensemble d'activités [Toshic, 2006], de natures différentes qui transforment les éléments dans le processus de production.

D'une part, il y a les activités de nature physique telles que le stockage, le transfert, la transformation, le contrôle ...etc qui peuvent être groupées dans un processus de production. D'autre part, il y a les activités de nature logique (décisionnelle) telles que l'ordonnancement, la définition de règles de priorité, la réaction à une perturbation, la programmation d'une action préventive ou corrective ..etc, qui peuvent être groupées dans un processus de contrôle. [Habchi, 2001].

Une autre définition est donnée par [Holmstrom, 2006], qui considère qu'un système de production est un ensemble d'objets et d'évènements qui sont connectés et contrôlés dans le temps et dans l'espace qu'ils occupent , dans le but d'atteindre des fonctions précises.

1.2.2 Composants d'un système de production

Un système de production est constitué de trois sous-systèmes qui coopèrent :

- Le sous-système physique représentant le système opérant [Habchi, 2001]. Il agit directement sur les produits en effectuant des opérations de transformation, de contrôle, de manutention et de stockage [Gaillard, 2002].
- Le sous-système d'information permettant l'acquisition, le traitement et la gestion des données du système et de son environnement.
- Le sous-système de décision**,** ce système, est appelé aussi système de conduite ou de pilotage. Il a pour rôle de modifier l'évolution du système physique [Gaillard, 2002].

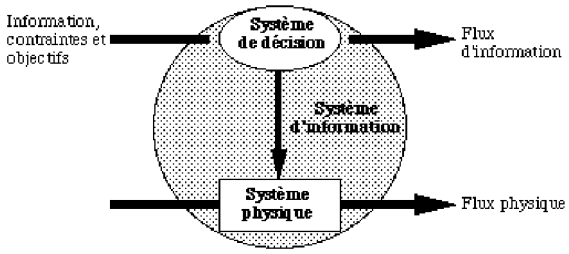

Figure 1.8 Composants d'un système de production [Gaillard, 2002].

Si cette décomposition est valable pour le système entrepris, et permet son analyse, elle est moins adaptée pour un système de production et sa modélisation. Dans ce dernier, les sous systèmes d'information et de décision n'ont pas d'existence propre, l'un sans l'autre. Ils constituent ensemble ce que nous appelons le système de pilotage ou le système d'information et de décision (SID) ou encore le système directeur [Habchi, 2001].

1.2.2.1 Système de fabrication (système physique)

Le système de fabrication est composé de ressources et d'entités. Les ressources comprennent les machines, les stocks, les opérateurs, les moyens de transfert...etc, et les entités comprennent les produits, les matières premières, les pièces, les lots...etc [Habchi, 2001]. Donc un système de fabrication est un ensemble de ressources qui effectuent des opérations de transformation sur les entités.

Les ressources peuvent être identifiées par deux types :

- Ressources principales (machines, robots, moyens de transfert, stocks,…) qui sont actives et qui ont une certaine autonomie vis-à-vis du reste du système. Elles peuvent évoluer dans certaines limites, indépendamment du reste du système.
- Ressources auxiliaires (outils, palettes,…) qui sont passives et qui permettent à des ressources principales d'accomplir une opération. Ce type de ressource est caractérisé par son exclusivité ou sa partageabilité [Habchi, 2001].

1.2.2.2 Système de pilotage

Piloter un système, c'est choisir un objectif par rapport auquel il faut définir la meilleure trajectoire. Le pilotage a pour but d'assurer la cohérence des décisions entre des ordres issus de la gestion prévisionnelle et les actions exécutées au niveau du système de production. Un système de pilotage peut être composé de:

- Des points de capture (les capteurs) pour la récupération de l'information.
- Un processus d*e* supervision pour l'analyse, le traitement de l'information, l'évaluation et la génération de décisions.
- Des points d'action qui constituent les points de passage des ordres ou des actions vers le système physique [Habchi, 2001].

La réactivité du système de production dépend en premier lieu de la capacité de réaction de son système de pilotage. Cette capacité dépend de la qualité et de la quantité des points de capture et des points d'action, d'une part, et de l'efficacité du processus de supervision, d'autre part.

1.2.3 Caractéristiques des systèmes de production

Les systèmes de production sont basés sur des caractéristiques, telles que la flexibilité, la réactivité, la proactivité et la robustesse.

1.2.3.1 Flexibilité

L'évolution croissante des besoins d'une entreprise fait que la conception du système de production est de plus en plus orientée vers des familles de produits et non vers un seul type de produit. Les systèmes correspondants à une telle exigence doivent se révéler flexibles.

La flexibilité d'un système de production se caractérise par sa capacité d'adaptation à la production de nouveaux produits pour lesquels le système n'a pas été étudié [Habchi, 2001].

Plusieurs types de flexibilité ont été mis en évidence :

- Flexibilité de produits: offre la possibilité d'une reconFigureuration du système pour la prise en compte d'un nouveau produit ou famille de produits permettant ainsi un gain de productivité.
- Flexibilité de mélange : c'est la possibilité de produire simultanément un ensemble de produits ayant des caractéristiques de base communes ; cette flexibilité peut être mesurée par le nombre de produits différents qui peuvent être fabriqués simultanément.
- Flexibilité de quantité : il s'agit de la capacité du système à faire face aux fluctuations de la quantité des produits à fabriquer en modifiant les rythmes, ainsi que les temps de passage et d'engagement des outils.
- Flexibilité d'ordre des opérations : permet de changer l'ordre des opérations en cours de production et choisir la destination suivante après chaque opération.
- Flexibilité d'expansion : autorise une extension et une modification de l'architecture du système et elle exige une phase de modélisation.
- Flexibilité des ressources : c'est la capacité des ressources à effectuer plusieurs tâches élémentaires et de permettre leur reprogrammation [Habchi, 2001].

1.2.3.2 Réactivité

Un système de production réactif, est un système qui est capable de répondre rapidement et économiquement à un changement ou à un aléa. Ces aléas peuvent provenir soit du système de production (défauts de réalisations d'une tâche, pannes des machines, …etc) soit de son environnement (approvisionnements des matières premières).

La réactivité d'un système de production est définie comme l'aptitude à répondre (réagir) dans un temps requis aux changements de son environnement interne ou externe [Habchi, 2001].

1.2.3.3 Proactivité

La réactivité est nécessaire, mais elle n'est pas suffisante et les systèmes de production doivent présenter une nouvelle propriété qui est la proactivité.

La proactivité d'un système de production se caractérise par ses capacités d'anticipation (prévoir et/ou provoquer) les changements d'état, d'apprentissage et d'enrichissement des connaissances (pour améliorer sa réactivité) [Habchi, 2001]. Donc, un système de production proactif est avant tout un système réactif.

1.2.3.4 Robustesse

La robustesse d'un système de production se définit par son aptitude à produire conformément aux résultats attendus. Cela suppose la garantie de l'obtention des performances souhaitées en présence d'incertitudes dans le système [Habchi, 2001].

1.2.4 Processus de production

Un processus de production est un élément actif dans un système de production, son but est de produire des produits [Holmstrom, 2006]. Un processus de production est un ensemble d'activités et d'évènements et des objets qui sont en relation.

Une activité est un évènement dont lequel un objet acte pour changer son état ou l'état d'un autre objet, hors un évènement est une occurrence qui cause un changement d'état d'un objet [Holmstrom, 2006].

Un objet dans un processus de production est une ressource principale ou auxiliaire .Son état est une condition ou une situation durant son cycle de vie . Cette situation peut être la réalisation d'une activité ou l'attente d'un événement ou la satisfaction d'une condition [Holmstrom, 2006].

1.2.5 Complexité des systèmes de production

Durant ces dernières années, les systèmes de production sont devenus plus complexes. Cette complexité due à plusieurs facteurs, qui sont :

- Les produits sont devenus et continuent à devenir plus complexes .Cela implique que la complexité de l'information nécessaire pour produire ces produits et le système qui produit sont ainsi complexes.
- La technologie de production est devenue elle même complexe.
- La taille des systèmes de production est grandis partiellement due à la complexité des produits et le nombre des variants de produits.
- Les besoins du marché sont augmentés et les clients demandent toujours de la marque et de la qualité des produits .Cela a conduit à une évolution dans les activités de production [Petit, 1999].

1.2.6 Classes des systèmes de production

Les systèmes de production sont classifiés en 2 classes principales :

1.2.6.1 Systèmes de production distribués

Un système de production distribué est un système composé de plusieurs entités, qui sont en coopération, reliées par un réseau de communication, chacune représente une partie opérationnelle du système de production.

Un système de production distribué est caractérisé par un ensemble de propriétés, parmi ces propriétés :

- L'autonomie, qui est la possibilité de créer et de contrôler l'exécution du plan de production.
- La distribution, qui permet a toutes les entités du système d'opérer dans le système.
- La décentralisation, indique qu'une opération peut être exécutée par plusieurs entités.
- La dynamique, qui signifie le changement de la structure et du comportement du système durant l'exécution d'une opération [Sousa, 2008].

1.2.6.2 Systèmes de production flexibles

Un système de production flexible FMS est un ensemble de machines multi-fonctionnaires et d'équipements qui peuvent être réorganisés périodiquement. Il permet de lancer un nouveau programme de production à chaque réorganisation [Andrea, 2006].

L'objectif des FMS est de réaliser :

- Une adaptation rapide aux types multiples de produits et aux programmes multiples de production.
- Un taux maximum d'utilisation (100% si possible) des machines et d'équipements.

Un FMS est composé de deux systèmes, un système physique et un système de contrôle. Chacun est divisé en des sous systèmes et chaque sous système en des activités. Deux types d'activités sont considérés, des activités de décision et des activités d'exécution [Bérard, 1998].

1.2.7 Supervision des systèmes de production

1.2.7.1 Réflexions sur les terminologies Commande, Surveillance et Supervision

a. Commande

La commande a pour rôle de faire exécuter un ensemble d'opérations en fixant des consignes de fonctionnement en réponse à des ordres d'exécution. Il peut s'agir de réaliser :

- Une séquence d'opérations constituant une gamme de fabrication dans le but de fabriquer un produit en réponse à une demande d'un client,

- Une séquence d'actions corrective destinée à rendre au système de production toute ou partie des fonctionnalités requises pour assurer sa mission. La perte d'une partie des fonctionnalités initialement disponibles faisant suite à l'occurrence d'une défaillance,
- Des actions prioritaires et souvent prédéfinies sur le procédé dans le but d'assurer la sécurité de l'installation et du personnel.
- Des opérations de test, de réglage, de nettoyage permettant de garantir que le système de production pourra continuer d'assurer sa fonction.

b. Surveillance

La surveillance est limitée aux fonctions qui collectent des informations, les archivent, font des inférences, etc. sans agir réellement ni sur le procédé ni sur la commande. La surveillance a donc un rôle passif vis-à-vis du système de commande et du procédé.

c. Supervision

La supervision permet de contrôler et surveiller l'exécution d'une opération ou d'un travail effectué par d'autres sans rentrer dans les détails de cette exécution. La supervision recouvre l'aspect fonctionnement normal et anormal.

- En fonctionnement normal, son rôle est surtout de prendre en temps réel des décisions. Pour cela elle est amenée à faire de l'ordonnancement en temps réel, à modifier la commande et à gérer le passage d'un algorithme de surveillance à l'autre.
- En présence de défaillance, la supervision va prendre toutes les décisions nécessaires pour le retour vers un fonctionnement normal. Après avoir déterminé un nouveau fonctionnement, il peut s'agir de choisir une solution curative, d'effectuer des réordonnancements locaux, de prendre en compte la stratégie de supervision de l'entreprise, de déclencher des procédures d'urgence, etc. [Combacau, 2000]

Toutes ces terminologies s'intègrent dans le pilotage ou la conduite des systèmes de production.

1.2.7.2 Superviseur pour les systèmes de production

L'objectif principal de la supervision de la production est la bonne exécution du programme prévisionnel de production par le système physique, en garantissant sa sûreté de fonctionnement.

La supervision d'un système inclut des fonctions de collecte et de visualisation d'informations, de surveillance, de diagnostic et d'aide à la prise de décision pour

l'accommodation, la reconfiguration ou la maintenance. Un superviseur peut être décrit comme le montre la Figure 1.9.

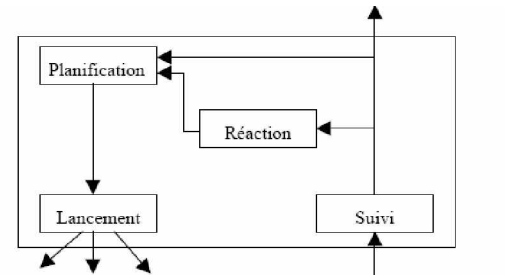

Figure 1.9 Modèle d'un superviseur [draghupb, 1998].

- **La planification**, qui consiste à mettre en oeuvre des techniques d'ordonnancement. Intégrées dans le processus global de supervision, la planification propose une affectation pour les différentes opérations, dans le temps et l'espace. C'est un moyen unique et incontournable pour s'assurer du respect des objectifs fixés.
- **Le lancement,** qui répartit et transmet les ordres au système physique en tenant compte de l'état des entités de production.
- **Le suivi**, qui recueille l'ensemble des événements survenant dans le système et qui met à jour une image interne du système opérant en autorisant ou inhibant certaines opérations toutes en assurant le bon fonctionnement du système.
- **La réaction**, qui corrige les déviations induites par les aléas de production. Cette activité est très liée à celle de la planification, car il s'agit de prendre des mesures correctives tout en s'assurant du respect des objectifs de production [draghupb, 1998]

Dans la littérature des systèmes de contrôle, un superviseur est un contrôleur qui utilise les données disponibles pour caractériser le comportement actuel du système.

La supervision d'un système de production consiste à utiliser un système de décision pour faire exécuter par le système physique l'ensemble des opérations de fabrication qui lui sont affectées :

- En respectant au mieux les objectifs de production fixés, tout en satisfaisant les contraintes spatiales, temporelles et de coûts.
- En s'assurant que chaque ordre transmit est cohérent vis-à-vis l'environnement dans lequel évolue le système.
- En utilisant un système d'information cohérent réalisant une interface robuste entre le système physique et le système de décision.

1.2.7.3 Structures de supervision d'un système de production

On peut distinguer essentiellement cinq structures différentes de supervision :

- **Structure centralisée**, caractérisée par un superviseur localisé au sein d'une ressource unique qui supervise la production.
- **Structure hiérarchisée**, dont chaque niveau coordonne les unités de supervision du niveau inférieur, et ce jusqu'au niveau le plus bas.
- **Structure coordonnée**, correspond à un ensemble de structures hiérarchisées où une coopération est possible au sein d'un même niveau de supervision.
- **Structure distribuée**, fondée sur une distribution totale des capacités de supervision.
- **Structure distribuée supervisée**, caractérisée par un ensemble d'entités coopérantes sous le contrôle d'une entité superviseur dont le rôle est d'imposer, de conseiller ou de modifier une décision afin de respecter un objectif plus global.

Le choix de la structure de supervision dépend essentiellement d'un compromis entre le degré d'autonomie et la cohérence de décision dans le système de production.

1.2.8 Modélisation des systèmes de production en vue de leur supervision

La modélisation des systèmes de production est indispensable pour la compréhension et l'analyse des phénomènes mis en jeu dans ces systèmes. Donc La supervision de tels systèmes repose également sur l'utilisation de modèles. Ces modèles doivent rendre compte de la structure et du comportement du système et permettre l'analyse de ses propriétés structurelles et comportementales.

Ainsi, la mise en œuvre des différentes fonctions de supervision s'appuie sur des modèles de natures différentes: structurels ou fonctionnels, comportementaux, Ils font intervenir des informations de nature également différentes: numérique, logique, symbolique ou textuelle, ...etc [Draghupb, 1998].

La partie supervision d'un système de production doit disposer d'un modèle de la partie opérative (ou physique) du système, afin de réaliser son objectif. Dans ce but, plusieurs approches de modélisation des systèmes de production ont été envisagées. Nous considérons deux types de modélisation :

1.2.8.1 Modélisation structurelle

La modélisation structurelle est relative à l'expression des besoins du système et à sa structure. Parmi les approches de modélisation existantes :

a. Approche orientée données : Entité / Association

C'est une méthode conceptuelle Apparue vers le milieu des années 60, elle définit une structure générale de données, indépendante de programmes qui les manipulent. Elle s'intéresse aux aspects structurels et informationnels. Néanmoins cette approche n'intègre pas les aspects dynamiques, son modèle étant par définition statique ; ainsi les conditions de déclenchement, l'ordonnancement dans le temps sont difficilement représentables.

b. Approche orientée objets

L'approche objet est adaptée à la problématique de l'étude des systèmes de production. En effet, les mécanismes objets tels que l'abstraction, l'encapsulation, la modularité, la classification, et l'héritage permettent de définir une méthodologie s'appuyant sur des variations mineures d'entités préétablis et de les adapter aux besoins du système étudié.

La démarche de modélisation par objets a pour objectif de définir les objets du système de production identifiés lors du processus de conception. Une première étape correspond à la définition des entités génériques ou objets abstraits, ensuite l'exploitation de ces entités génériques. L'utilisateur de l'approche pourra adapter les modèles proposés à son besoin, en définissant de cette manière l'architecture de supervision du système de production considéré. Parmi les formalismes utilisés, le langage UML que nous allons l'étudier dans la suite.

c. Approche multi-agents

Les systèmes multi-agents ont été développés dans le cadre de l'intelligence artificielle distribuée. L'intérêt qu'ils suscitent est lié à leur capacité d'aborder les problèmes complexes d'une manière distribuée et de proposer des solutions réactives et robustes.

Dans la supervision des systèmes de production, la coopération entre les agents et la communication par passage de messages est indispensable pour résoudre des conflits pouvant apparaître, pour l'allocation des ressources limitées, pour réconcilier des préférences différentes et rechercher des solutions dans un espace globale à partir des informations locales.

Les systèmes multi-agents ajoutent à la localité de comportements, présentée dans l'approche objet, l'autonomie et la répartition de prise de décisions. En effet, l'agent, à l'encontre de l'objet, peut effectuer un certain travail, le refus pouvant s'expliquer par son manque de compétence (il ne possède pas le savoir-faire nécessaire) ou par sa trop grande occupation à une autre tâche ou par toute autre raison [draghupb ,1998].

1.2.8.2 Modélisation comportementale

La modélisation comportementale est relative à la spécification du comportement dynamique de l'ensemble des objets du système de production et sa stratégie de production.

Plusieurs types de modèles sont adaptés pour ce type de modélisation, parmi ces modèles, nous pouvons citer :

a. Les modèles semi-formels, comme les diagrammes dynamiques d'UML, ou REMORA, qui permettent de représenter différentes activités, leurs flux d'entrées et de sorties, informationnels et matériels, à différents niveaux de détail.

b. Les modèles formels, comme les automates d'états finis , les réseaux de Petri, les state-charts, les processus communicants de type CSP basés sur la logique temporelle etc , qui permettent de modéliser la dynamique des systèmes et de les analyser de manière formelle .

1.2.8.3 Langage de modélisation UML

a. Présentation

En 1994, Rumbaugh et Booch unissent leurs efforts pour mettre au point la méthode unifiée(Unified Method 0.8), La méthode unifiée à partir de la version 1.0 devient UML (Unified Modeling Language) [Frédéric, 2001]. UML est le standard de l'OMG (Object Groupe Management) pour la modélisation orientée objet des systèmes logiciels. Il propose un ensemble de notations, sous forme de diagrammes.

UML n'est pas une méthode (i.e. une description normative des étapes de la modélisation), c'est un langage graphique qui permet de représenter, de communiquer les divers aspects d'un système [Frédéric, 2001], il a été pensé pour permettre la modélisation des activités de l'entreprise, pas pour les régir.

UML est un langage pseudo-formel, il est fondé sur un méta-modèle, qui est une description très formelle de tous les concepts d'un langage,

UML 2.0 définit treize types de diagrammes qui peuvent être divisés en deux catégories. Six types de diagrammes pour la modélisation de la structure statique du système, sept sont pour les différents aspects de la modélisation dynamique du système.

Catégorie	Diagrammes
Diagrammes structurels ou	▪ Diagramme de classes (Class diagram)

diagrammes statiques	▪ Diagramme d'objets (Object diagram) ▪ Diagramme de composants (Component diagram) ▪ Diagramme de déploiement (Deployment diagram) ▪ Diagramme de paquetages (Package diagram) ▪ Diagramme de structures composites (Composite structure diagram)
Diagrammes comportementaux ou diagrammes dynamiques	▪ Diagramme de cas d'utilisation (Use case diagram) ▪ Diagramme d'activités (Activity diagram) ▪ Diagramme d'états-transitions (State machine diagram)
	• **Diagrammes d'interaction (Interaction diagram)** ▪ Diagramme de séquence (Sequence diagram) ▪ Diagramme de communication (Communication diagram) ▪ Diagramme global d'interaction (Interaction overview diagram) ▪ Diagramme de temps (Timing diagram)

Tab 1.1 Les différents diagrammes d'UML [Audibert, 2006].

Ces diagrammes, sont d'une utilité variable selon les cas, ils ne sont pas nécessairement tous produits à l'occasion d'une modélisation. Les plus utiles sont les diagrammes d'activités, de cas d'utilisation, de classes, d'objets, de séquence et d'états-transitions.

b. Sémantique à base de méta-modélisation

Depuis ses premières versions, le standard UML est caractérisé par sa sémantique définie par une approche de méta-modélisation. Un méta-modèle est la définition des constructions et des règles de création des modèles. Le méta-modèle d'UML définit donc la structure que doit respecter tout modèle UML [UML, 2007].

L'approche de méta-modélisation adoptée par l'OMG est connue comme une hiérarchie à quatre niveaux :

▪ Niveau méta-méta-modèle (M3), il définit le langage de spécification du méta-modèle. Le MOF (Meta Object Facility) est un exemple d'un méta-méta-modèle.

- Niveau méta-modèle (M2),le méta-modèle d'UML se situe à ce niveau et il est spécifié en utilisant le MOF, c.à.d. les concepts du méta-modèle d'UML sont des instances des concepts de MOF.
- Niveau modèle (M1), qui correspond au niveau des modèles UML des utilisateurs.
- Niveau objets (M0), qui correspond au niveau des objets à l'exécution [Ziadi, 2004].

Le méta-modèle d'UML est décrit en utilisant une partie de la notation d'UML lui même. Les concepts suivants sont utilisés :

- Les classes d'UML, pour décrire les méta-classes.
- Les attributs, pour décrire les propriétés attachées à une méta-classe.
- Les associations, pour décrire des liens entre les méta-classes.
- Les paquetages (packages), pour regrouper les méta-classses par domaine.

c. Diagrammes statiques d'UML

UML utilise des représentations par des diagrammes, pour modéliser les aspects statiques (structurels). Nous présentons ici deux diagrammes qui nous intéressent dans notre travail, qui sont le diagramme de cas d'utilisation et le diagramme de classes.

c.1 Diagramme de cas d'utilisation (Use Cases Diagram**)**

Un diagramme de cas d'utilisation est un diagramme qui est utilisé pour donner une vision globale du comportement fonctionnel d'un système. Il permet d'identifier les possibilités d'interaction entre le système et les acteurs (extérieurs au système), c'est-à-dire toutes les fonctionnalités que doit fournir le système.

C'est le premier diagramme du modèle UML, celui où s'assure la relation entre l'utilisateur et les objets que le système met en œuvre. C'est donc une vue du système dans son environnement extérieur [Federic, 2001] [Audibert, 2006] [UML, 2007].

➢ **Eléments des diagrammes de cas d'utilisation**

Le diagramme de cas d'utilisation est constitué de deux éléments, comme le montre la Figure suivante :

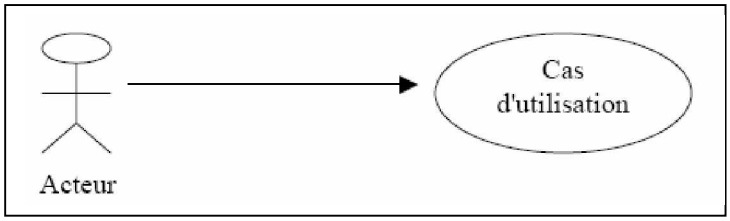

Figure 1.10 Les éléments constituants le diagramme de cas d'utilisation [Federic, 2001].

- **Acteur**

Un acteur est une entité externe qui agit sur le système ; Le terme acteur ne désigne pas seulement les utilisateurs humains mais également les autres systèmes. [Federic, 2001]. Un acteur se représente par un petit bonhomme (Figure 1.10) avec son nom (i.e. son rôle) inscrit dessous, il est également possible de le représenter sous la forme d'un classeur(rectangle) stéréotypé (étiqueté) par le mot « actor » et son rôle est inscrit dessous (Figure 1.11)

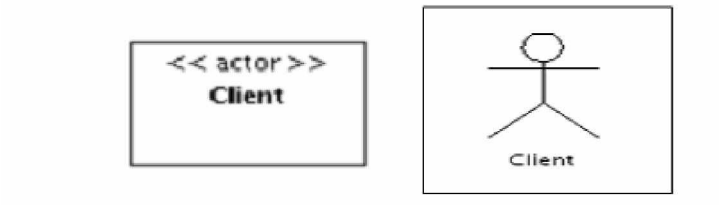

Figure 1.11 Représentation d'un acteur [Audibert, 2006].

Il existe deux catégories d'acteurs :

1. Les acteurs principaux : Cette catégorie regroupe les personnes qui utilisent les fonctions principales du système. Par exemple dans un système distributeur de billets, il s'agit des clients [UML, 2007].

2. Les acteurs secondaires. Cette catégorie regroupe les personnes qui effectuent des tâches administratives ou de maintenance. Dans le même exemple du distributeur de billets, il s'agit de la personne qui recharge la caisse du distributeur [UML, 2007].

- **Cas d'utilisation**

Un cas d'utilisation est un ensemble d'actions réalisées par le système en réponse à une action d'un acteur [Federic, 2001]. Il décrit les objectifs du système et modélise un service rendu par le système, sans imposer le mode de réalisation de ce service. Le service est visible de l'extérieur.

Un cas d'utilisation se représente par une ellipse (Figure 1.12) contenant le nom du cas (un verbe à l'infinitif), et optionnellement, au-dessus du nom, un stéréotype (Une étiquette). Dans le cas où l'on désire présenter les attributs ou les opérations du cas d'utilisation, il est préférable de le représenter sous la forme d'un classeur« use case.

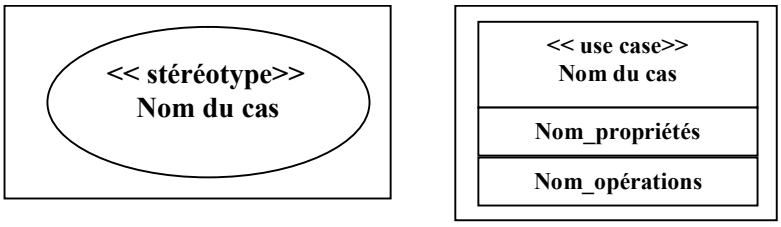

Figure 1.12 Représentations d'un cas d'utilisation [Audibert, 2006].

- **Représentation d'un diagramme de cas d'utilisation**

Comme le montre l'exemple de la Figure (1.13), la frontière du système est représentée par un cadre. Le nom du système Figure à l'intérieur du cadre, en haut. Les acteurs sont à l'extérieur et les cas d'utilisation sont à l'intérieur.

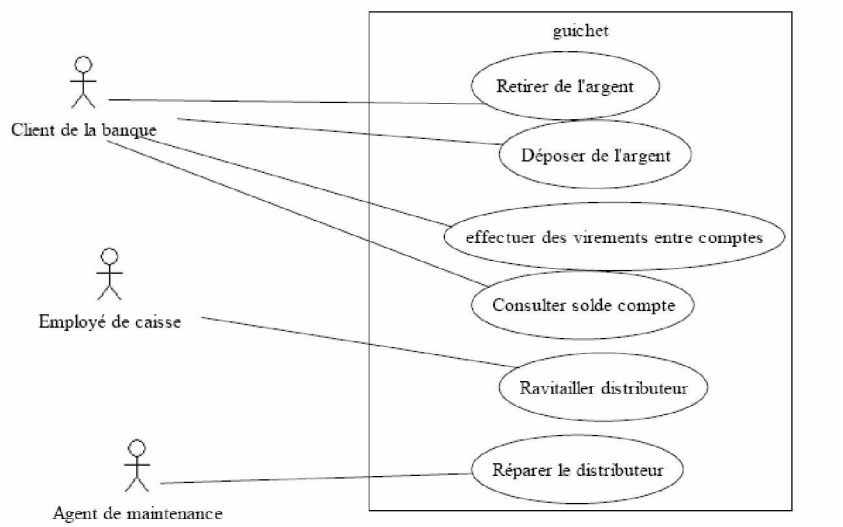

Figure 1.13 Exemple d'un diagramme de cas d'utilisation d'un guichet automatique bancaire [Federic, 2001].

- **Description textuelle des cas d'utilisation**

Le diagramme de cas d'utilisation décrit les grandes fonctions d'un système du point de vue des acteurs, mais n'expose pas de façon détaillée le dialogue entre les acteurs et les cas d'utilisation. Bien que de nombreux diagrammes d'UML permettent de décrire un cas, il est recommandé de rédiger une description textuelle car c'est une forme souple qui convient bien dans des situations.

Une description textuelle couramment utilisée se compose de trois parties :

1. La première partie permet d'identifier les cas, elle doit contenir les informations suivantes :

- **Nom :** Utiliser une tournure à l'infinitif (exp : Réceptionner un colis).
- **Objectif :** Une description résumée permettant de comprendre l'intention principale du cas d'utilisation.
- **Acteurs principaux :** Ceux qui vont réaliser le cas d'utilisation (la relation avec le cas d'utilisation est illustrée par le trait liant le cas d'utilisation et l'acteur dans un diagramme de cas d'utilisation).

- **Acteurs secondaires :** Ceux qui ne font que recevoir des informations à l'issue de la réalisation du cas d'utilisation.
- **Dates :** Les dates de création et de mise à jour de la description courante.
- **Responsable :** Le nom des responsables.
- **Version :** Le numéro de version.

2. La deuxième partie contient la description du fonctionnement du cas sous la forme d'une séquence de messages échangés entre les acteurs et le système. Elle contient toujours une séquence nominale qui décrit le déroulement normal du cas. A la séquence nominale s'ajoutent fréquemment des séquences alternatives (des embranchements dans la séquence nominale) et des séquences d'exceptions (qui interviennent quand une erreur se produit).

- **Préconditions :** elles décrivent dans quel état doit être le système avant que ce cas d'utilisation puisse être déclenché.
- **Scénarios :** Ces scénarii sont décrits sous la forme d'échanges d'évènements entre l'acteur et le système. On distingue le scénario normal, qui se déroule quand il n'y a pas d'erreur, et les scénarios d'exception qui décrivent les cas d'erreurs.
- **Des postconditions :** Elles décrivent l'état du système à l'issue des différents scénarios.

3. La troisième partie de la description d'un cas d'utilisation est une rubrique optionnelle, elle contient généralement des spécifications non fonctionnelles (spécifications techniques,...). Elle peut éventuellement contenir une description des besoins en termes d'interface graphique [Audibert, 2006].

- **Modélisation avec le diagramme de cas d'utilisation**

Un bon diagramme de cas d'utilisation doit être simple avec un nombre d'acteurs limité, la démarche de construction d'un diagramme de cas d'utilisation doit contenir les points suivants :

1. Lister les acteurs.
2. Déterminer le rôle de chaque acteur.
3. Déterminer les cas d'utilisation.
4. Identifier le flot d'évènements auxquels le système doit réagir.
5. Structurer les cas d'utilisation.
6. Finaliser un ou plusieurs diagrammes par package [UML, 2007].

- **Identification des acteurs dans un diagramme de cas d'utilisation**

Les acteurs d'un système sont les entités externes à ce système qui interagissent (saisie de données, réception d'information, ...) avec lui. Chaque acteur doit être nommé. Ce nom doit refléter sont rôle car un acteur représente un ensemble cohérent de rôles joués vis-à-vis du système.

Pour trouver les acteurs d'un système, il faut identifier quels sont les différents rôles que vont devoir jouer ses utilisateurs (ex: responsable clientèle, responsable d'agence, administrateur, approbateur, ...).

Pour faciliter la recherche des acteurs, nous pouvons imaginer les frontières du système. Tout ce qui est à l'extérieur et qui interagit avec le système est un acteur, tout ce qui est à l'intérieur est une fonctionnalité à réaliser. Par exemple, l'hôtesse de caisse d'un magasin de grande distribution est un acteur pour la caisse enregistreuse, par contre, les clients du magasin ne correspondent pas à un acteur car ils n'interagissent pas directement avec la caisse [Audibert, 2006].

- **Identification des cas d'utilisation**

L'ensemble des cas d'utilisation doit décrire exhaustivement les exigences fonctionnelles du système. Chaque cas d'utilisation correspond donc à une fonction métier du système, selon le point de vue d'un de ses acteurs. Aussi, pour identifier les cas d'utilisation, il faut se placer du point de vue de chaque acteur et déterminer comment et surtout pourquoi il se sert du système. Par exemple, un distributeur de billets aura probablement un cas d'utilisation Retirer de l'argent et non pas Distribuer de l'argent [Audibert, 2006].

c.2 Diagramme de classes

Le diagramme de classes est généralement considéré comme le plus important dans un développement orienté objet. Il représente l'architecture conceptuelle du système : il décrit les classes que le système utilise, ainsi que leurs relations [Audibert, 2006].

Alors que le diagramme de cas d'utilisation montre un système du point de vue des acteurs, le diagramme de classes en montre la structure interne. Il permet de fournir une représentation abstraite des objets du système qui vont interagir ensemble pour réaliser les cas d'utilisation [UML, 2007].

- **Eléments du diagramme de classes**

Un diagramme de classes est constitué d'un ensemble de classes qui permettent de décrire un ensemble d'objets (attributs et comportement), et de relations entre ces classes.

- **Classes**

Une classe est une description d'un ensemble d'objets partageant la même sémantique, ainsi que les mêmes attributs, opérations et relations [Federic, 2001].

Une classe est un concept abstrait représentant des éléments variés comme :
- Des éléments concrets (ex : des avions),
- Des éléments abstraits (ex : des commandes),
- Des composants d'une application (ex : les boutons des boîtes de dialogue),
- Des structures informatiques (ex : des tables de hachage),
-Des éléments comportementaux (ex : des tâches), etc [Audibert, 2006].

Une classe est représentée graphiquement par un rectangle divisé en trois compartiments. Le premier indique le nom de la classe, le deuxième ses attributs et le troisième ses opérations.

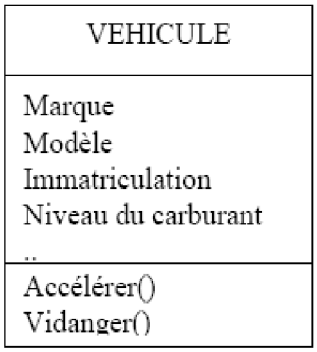

Figure 1.14 Exemple d'une classe [Federic, 2001].

1. **Nom d'une classe :** Le nom de la classe doit évoquer le concept décrit par la classe. nous pouvons ajouter des informations subsidiaires comme le nom de l'auteur de la modélisation, la date, etc [Federic, 2001].
2. **Attributs :** Les attributs définissent des informations qu'une classe ou un objet doit connaître. Ils représentent les données encapsulées dans les objets de cette classe. Chacune de ces informations est définie par un nom, un type de données, une visibilité et peut être initialisé. Le nom de l'attribut doit être unique dans la classe [Audibert, 2006].

3. **Opérations :** La définition d'une classe est complétée par l'ensemble des opérations qu'elle peut exécuter. Une opération est une fonctionnalité assurée par la classe. [Federic, 2001]. Dans une classe, une opération a un nom et des paramètres, elle doit être unique.

UML définit trois niveaux de visibilité pour les attributs et les opérations d'une classe :

1. **Public** qui rend l'élément visible à tous les clients de la classe,
2. **Protégé** qui rend l'élément visible aux sous classes de la classe,
3. **Privé** qui rend l'élément visible à la classe seule [Federic, 2001].

- **Association entre classes**

Une association représente une relation structurelle entre classes d'objets. La plupart des associations sont binaires, c'est à dire qu'elles connectent deux classes. Nous présentons une association en traçant une ligne entre les classes associées. Les associations peuvent être nommées afin de faciliter la compréhension des modèles. Il est d'usage de nommer les associations par une forme verbale, comme le montre la Figure suivante :

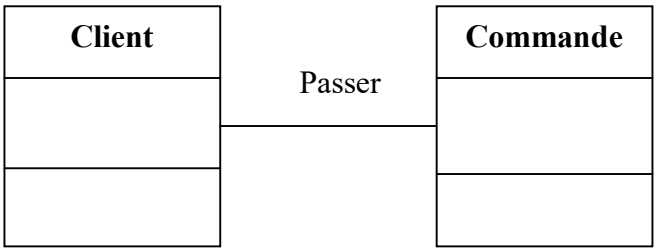

Figure 1.15 Association entre deux classes [Federic, 2001].

1. Arité des associations

On appelle arité d'une association le nombre de classes qui participent à l'association. Il y a deux types d'arités selon le nombre de classes associés à l'association. Il existe deux types d'association :

- **Association binaire :** Une association binaire est matérialisée par un trait plein entre les classes associées Elle peut être ornée d'un nom, avec éventuellement une précision du sens de lecture .Quand les deux extrémités de l'association pointent vers la même classe, l'association est dite réflexive.

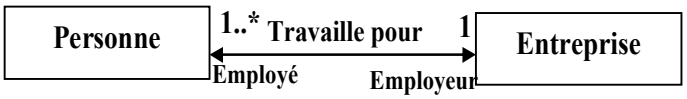

Figure 1.16 Une association binaire entre deux classes [Audibert, 2006].

- **Association n-aire :** Une association n-aire lie plus de deux classes. Elle est représentée par un grand losange avec un chemin partant vers chaque classe participante. Le nom de l'association, le cas échéant, apparaît à proximité du losange.

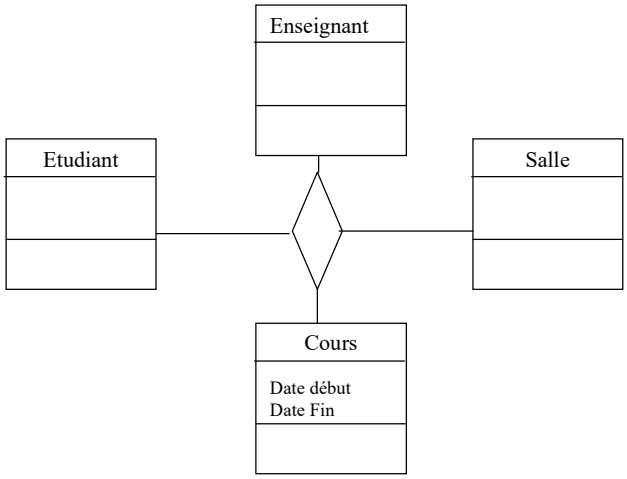

Figure 1.17 Une association d'arité égale à 4 [Federic, 2001].

2. Rôle d'une association

Les extrémités d'une association sont appelées rôles et peuvent porter un nom. Le rôle décrit comment une classe voit une autre classe au travers une association. Un rôle est nommé au moyen d'une forme nominale.

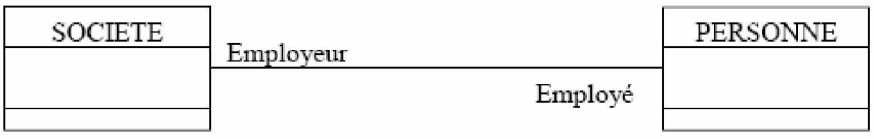

Figure 1.18 Identification d'un rôle d'association [Federic, 2001].

3. Multiplicité des associations

Chaque rôle peut porter une multiplicité montrant combien d'objets de la classe considérée (celle qui joue ce rôle) peuvent être liés à une instance de l'autre classe par l'association. La multiplicité est représentée sous la forme d'un couple de cardinalités [Federic, 2001]. Voici quelques exemples de multiplicité :

1..1 noté 1	Un et un seul
0..1	Zéro ou un
0..* noté *	De Zéro à n
1..*	De un à n
n..m	De n à m

Tab 1.2 Différents multiplicités d'une association [Audibert, 2006].

4. Types d'associations

- **Classes associations** : il peut arriver que l'on ait besoin de garder des informations (attributs ou opérations) propres à une association. Une classe de ce type est appelée classe association. [Federic, 2001].Une classe association possède les propriétés des associations et des classes, elle se connecte à deux ou plusieurs classes et possède également des attributs et des opérations.

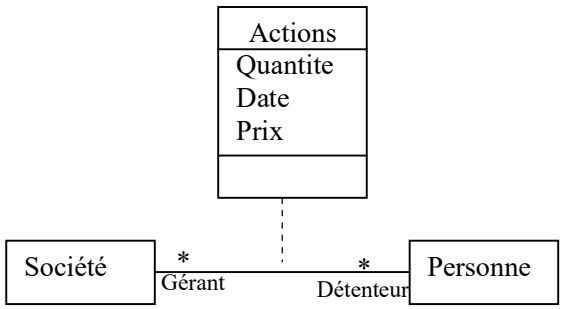

Figure 1.19 Exemple d'une classe association [Audibert, 2006].

- **Agrégation :** Une agrégation est une association qui représente une relation d'inclusion structurelle ou comportementale d'un élément dans un ensemble. Graphiquement, on ajoute un losange vide du côté de l'agrégat [Audibert, 2006].

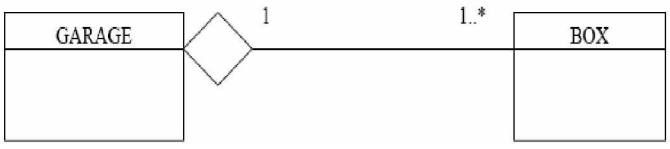

Figure 1.20 Exemple d'une classe association d'agrégation [Federic, 2001].

- **Composition :** La composition, également appelée agrégation composite, décrit une contenance structurelle entre instances. Ainsi, la destruction de l'objet composite implique la destruction de ses composants. Une instance de la partie appartient toujours à au plus une instance de l'élément composite.

Graphiquement, on ajoute un losange plein du côté de l'agrégat [Audibert, 2006].

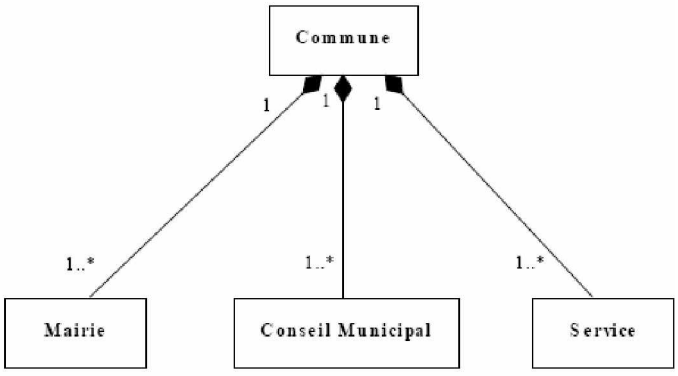

Figure 1.21 Exemple d'une classe association de composition [Federic, 2001].

- **Généralisation** : L'association de généralisation est une relation de classification entre un élément plus général et un élément plus spécifique. La relation de généralisation signifie «est un» ou « est une sorte de ».

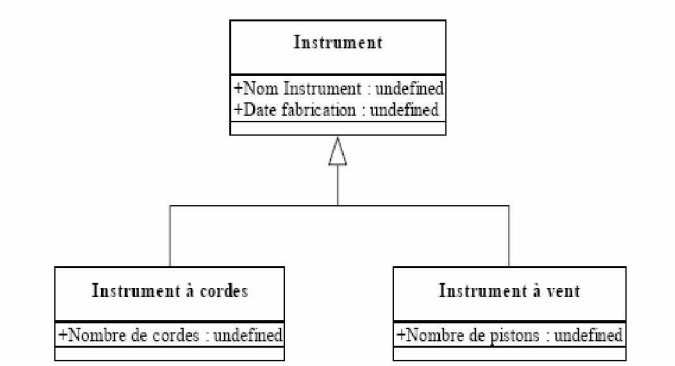

Figure 1.22 Exemple d'une classe association de composition [Federic, 2001].

- **Elaboration d'un diagramme de classes**

Pour élaborer un diagramme de classes, il y a au moins trois points de vue qui guident la modélisation:

- Le point de vue spécification, qui met l'accent sur les interfaces des classes plutôt que sur leurs contenus.
- Le point de vue conceptuel, qui capture les concepts du domaine et les liens qui les lient. Il s'intéresse à la manière éventuelle d'implémenter ces concepts et leurs relations et aux langages d'implémentation.
- Le point de vue implémentation, qui est le plus courant, il détaille le contenu et l'implémentation de chaque classe.

Une démarche couramment utilisée pour bâtir un diagramme de classes consiste à :

1. Trouver les classes du domaine étudié**, c**ette étape empirique se fait généralement en collaboration avec un expert du domaine. Les classes correspondent généralement à des concepts ou des substantifs du domaine.

2. Trouver les associations entre classes**,** les associations correspondent souvent à des verbes, ou des constructions verbales, mettant en relation plusieurs classes, comme «est composé de», «pilote», «travaille pour».

3. Trouver les attributs des classes**.** les attributs correspondent souvent à des substantifs, ou des groupes nominaux, tels que «la masse d'une voiture» ou «le montant d'une transaction ». Les adjectifs et les valeurs correspondent souvent à des valeurs d'attributs.

4. Organiser et simplifier le modèle en éliminant les classes redondantes et en utilisant la d'héritage [Federic, 2001].

1.2.9 Vérification des systèmes de production en vue de leur supervision

La vérification d'un système de production en vue de sa supervision est couverte par la connaissance d'un ensemble d'informations concernant les différentes fonctions du système ainsi que ses propriétés.

La vérification des systèmes de production repose sur l'existence d'un modèle qui modélise le système à vérifier. L'objectif est de vérifier ses propriétés. Les contraintes de sûreté de fonctionnement imposées aux systèmes de production conduisent à préconiser la mise en place de méthodes formelles pour la spécification, et la vérification des propriétés tels que: l'absence de blocage, vivacité, etc.

Le choix du formalisme de vérification est un compromis entre pouvoir d'expression et moyens de vérification existants.

Les formalismes les plus utilisés pour la vérification des systèmes de production sont : les modeles de checking, les automates d'états finis, et les réseaux de Petri, que nous les présentons dans la suite.

1.2.9.1 Modèle de Checking

Le modèle de checking est une technique de vérification formelle complètement automatique. Un modèle checker est un algorithme qui permet de vérifier si un modèle spécifié par un système d'états-transitions satisfait une propriété exprimée dans une logique temporelle.

Les systèmes de transitions désignent un ensemble de notations permettant de décrire des comportements dits « orientés contrôle ».Ces notations sont des états et des transitions entre les

états décrivant le comportement du système. Un exemple du modèlede checking est (la structures de Kripke, CTL) [Hammani, 2006].

Soit **P** un ensemble fini de propositions booléennes. Une structure de Kripke sur **P** est un quadruple **M = (S; T; I; L)** où :

- **S** est l'ensemble des états .
- **T** $\subseteq$ **S** × **S** est la relation de transition ; $\forall$ **s** $\in$ **S,** $\exists$ **s'** $\in$ **S; (s; s') $\in$ T**.
- **I** $\subseteq$**S** est l'ensemble des états initiaux.
- **L : S** $\rightarrow$**2^{P}** est la fonction d'étiquetage [Deharbe, 2002].

Un exemple du modèle de Checking illustrant une partie d'une ligne de production est présenté dans la Figure 1.24 :

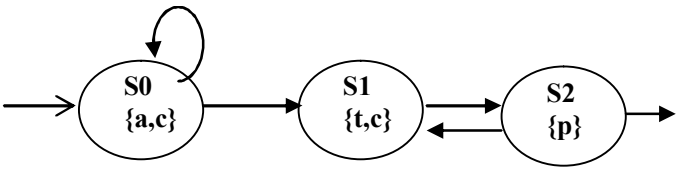

Figure 1.23 Exemple d'un modèle de Checking.

- **P = {a, c, t, p}**, où: **a** : amener une pièce, **t** : travailler sur une pièce, **p** : passer une pièce a une autre machine, **c** : contrôler une pièce.
- **S = {s0,s1,s2}**, **T = {(s_0, s_0); …….}**, **I = {s_0) et L : s_0 $\rightarrow${a}**….[Deharbe, 2002].

Les propriétés à vérifier sont exprimées dans la logique temporelle CTL.

Basant sur ce principe, le modèle de Checking permet de vérifier un ensemble de propriétés pour un système de production, suivant les étapes suivantes :

- Modéliser le système de production par un modèle d'états-transitions.
- Ecrire une spécification du système sous forme d'une logique temporelle.
- Faire appel à un algorithme du modèle de Checker pour vérifier les propriétés du système et pour générer des scénarios qui mènent aux problèmes de fonctionnement ou les scénarios qui évitent ces problèmes [Garmhausen, 1998].

Cependant, cette méthode se heurte à la taille, souvent gigantesque, des graphes générés, et pose le problème d'explosion d'états dans le cas d'un système de production d'une grande taille. Ainsi ce formalisme ne permet pas de prendre en compte l'aspect temporel du fonctionnement du système.

1.2.9.2 Automates à états finis

Les automates à états finis ou machines à états finis, ont été employés pour spécifier le comportement dynamique de certains systèmes.

Les automates à états finis sont représentés graphiquement par des diagrammes d'états-transitions, des graphes orientés dont les noeuds sont des états et les arcs des transitions (Figure 1.24). Un état est un ensemble de valeurs qui caractérise le système à un moment donné dans le temps. Une transition d'état est une relation entre deux états indiquant un changement d'état possible, et qui peut être annotée pour indiquer les conditions et les sources de déclenchement (événements) et les opérations qui en résultent (sorties). On appelle automate fini le quintuplet $\mathbf{A = < \Sigma, Q, \delta, I, F >}$, où :

- **Σ** est un alphabet.
- **Q** est un ensemble d'états stables.
- **I** est une partie de **Q** appelée ensemble des états initiaux.
- **F** est une partie de **Q** appelée ensemble des états finaux.
- **δ** est une partie de **Q×Σ ×Q** appelée ensemble des transitions. C'est une fonction. de transition qui à un état du système et un élément de l'alphabet associe le passage à un autre état [Pinzon, 1997].

Un exemple d'un automate à états finis illustrant une partie d'un système de production de pièces est présenté dans la Figure suivante. Il consiste simplement en deux marteaux (Pushers), qui ont deux degrés de prolongation demi-prolongé, et pleinement-prolongé. Quand le **Pusher 1** est entièrement prolongé, la pièce en position 1 s'est déplacée à la position 2. S'il y a un objet en position 2 et le **Pusher 2** se prolonge entièrement, alors la pièce se déplace à la position 3.

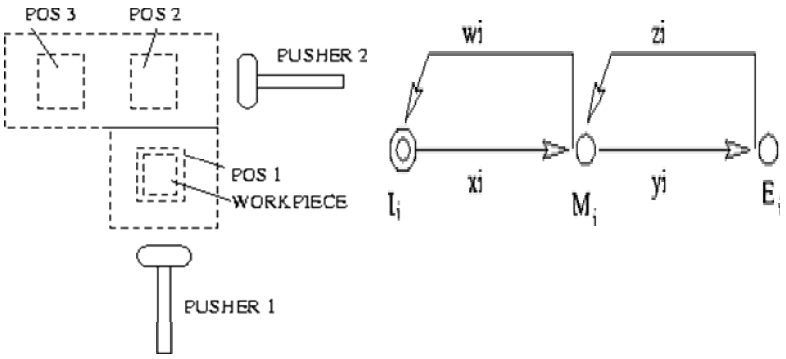

Figure 1.24 Exemple d'un automate a états finis [Pinzon, 1997].

Les états **Ii**, **Mi**, et **Ei**, indiquent les positions du Pusher : initial, demi-prolongé, et pleinement-prolongé respectivement, l 'état **Ii** est un état initial et finale au même temps.

L'ensemble des événements est **Σ= {xi,yi,wi : i=1,2}.**

Comme le modèle de Checking, les automates à états finis mènent rapidement à une explosion du nombre d'états et de transitions suivant la taille du système de production.

Ainsi, la complexité de vérification des systèmes de production par ce formalisme vient de:

- La coopération : les automates décrivent des processus ayant un but commun.
- La compétition : les automates partagent des ressources
- Le pseudo parallélisme ou l'entrelacement (interleaving) : les événements sont tous ordonnés par un ordre total [Valette, 1999].

1.2.9.3 Réseaux de Petri

Les réseaux de Pétri sont un outil efficace pour modéliser, et vérifier les systèmes de production. Ils peuvent manipuler les problèmes qui ne peuvent pas être modélisé par les autres formalismes.

Contrairement aux formalismes précédents, Les réseaux de Pétri sont bien adaptés pour modéliser des systèmes de production parce qu'ils capturent les relations et les interactions entre les événements. En outre, une base mathématique solide existe pour l'analyse des propriétés de système telles que le blocage, le parallélisme et les conflits…etc.

a. Définition

Un réseau de Petri (RdPs) est un graphe biparti orienté qui a deux types de nœuds : les places (notées graphiquement par des cercles) et les transitions (notées graphiquement par un rectangle ou une barre) ; reliées par des arcs (qui sont des flèches).Un arc est un lien soit d'une place à une transition, soit d'une transition à une place.

Chaque arc est étiqueté par une valeur (ou un poids), qui est un nombre entier positif. L'arc ayant **k** poids peut être interprété comme un ensemble de **k** arcs parallèles. L'étiquette du poids égale à **1** est ignorée.

Formellement, un réseau de Petri est un graphe orienté biparti valué **< P, T, Pré, Post >** avec :

1. **P** est l'ensemble des places.
2. **T** est l'ensemble des transitions.
3. **Pré** = **P** x **T**→**N**, est une application d'incidence avant où : **Pré (p, t)** contient la valeur entière « **n** » associée à l'arc allant de « **p** » à « **t** ».

4. **Post** = **P** x **T** → **N**, une application d'incidence arrière où : **Post (p, t)** contient la valeur entière « **n** » associée à l'arc allant de « **t** » à « **p** ».

Remarques :

1- « **p** » est une place d'entrée de la transition « **t** » si **Pré (p, t) > 0.**

2- « **p** » est une place de sortie de la transition « **t** » si **Post (p, t) > 0** [Valette ,2002].

b. Modélisation par les RdPs

Un réseau de Petri est un outil pour modéliser une classe spécifique de problèmes qui sont les systèmes contenant des évènements concurrents ou parallèles. Il modélise en particulier deux aspects dans un système, les conditions et les événements et la relation entre les deux.

Dans ce cadre, dans un système à n'importe quel moment, certaines pré-conditions (une description logique d'un état du système nécessaires au déclenchement d'un événement) devenues vrai, cela va causer le déclenchement de certains événements qui vont changer l'état du système. Lorsqu'un événement se produit, certaines de ses pré-conditions peuvent cesser d'être vraies alors que d'autres post-conditions de l'événement deviennent vraies. [Peterson, 1999].

L'exemple de la Figure 1.25 présente un lecteur de cartes

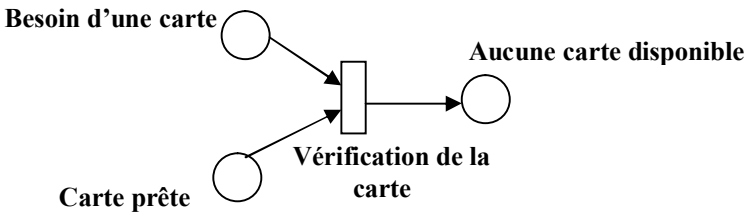

Figure 1.25 Un modèle simple de trois conditions et un événement [Peterson, 1999].

Où il y a les pré-conditions suivantes :

- Besoin d'une carte pour la lecture
- Carte Prête.

La pré-condition **''**Carte Prête'' va permettre l'apparition de l'événement '' Vérification d'une carte''. L'occurrence de cet évènement va mettre les deux pré-conditions précédentes à faux et rendre la post-condition '' Aucune carte disponible'' vrai.

Comme il est montré dans la Figure 1.25, les places sont utilisées pour représenter les conditions et les transitions pour représenter les événements [Peterson, 1999].

c. Marquage d'un RdP

Un réseau de Petri marqué est un couple :

N=<R, M_o > Où : **R** est un RdP, **M_o** est le marquage initial du réseau R.

Le marquage d'un RdP est une application **M : P → N** donnant pour chaque place le nombre de jetons qu'elle contient [Valette ,2002].

d. Evolution d'un RdP

L'évolution d'un RdP correspond à l'évolution de son marquage au cours du temps (évolution de l'état du système) : il se traduit par un déplacement des jetons pour une transition **t** de l'ensemble des places d'entrée vers l'ensemble des places de sortie de cette transition. Ce déplacement s'effectue par le franchissement de la transition **t** [Scorletti, 2006].

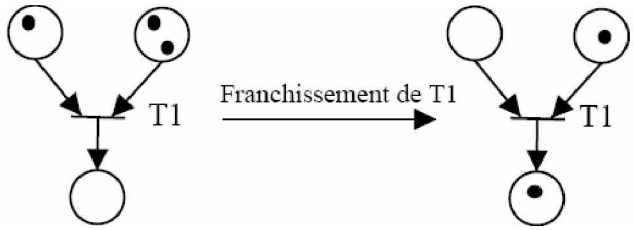

Figure 1.26 Franchissement de la transition T1 [Scorletti, 2006].

Pour un marquage **M**, une transition **t** est dite tirable (ou franchissable ou sensibilisée) si et seulement si:

∀ pi∈ P , on a M(pi) >= Pré (pi, t)

C'est-à-dire, pour toutes les places d'entrées **pi** de **t**, le nombre de jetons dans **pi**, **M(pi)**, est supérieur ou égal au poids de l'arc allant de **pi** à **t**.

Le franchissement de **t** depuis **M** donnant le nouveau marquage **M'**, se notera : **M[t>M'** [Scorletti , 2006].

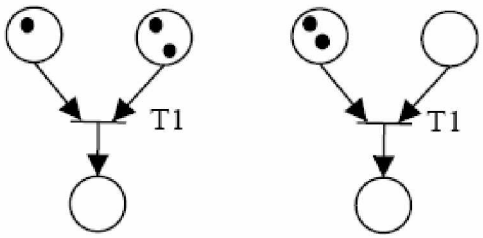

Figure 1.27 La Transition T1 (gauche) est franchissable, T1 (droite) est non franchissable [Scorletti , 2006].

L'évolution d'un RdP se fait par le franchissement d'une seule transition à la fois. Quand plusieurs transitions sont simultanément franchissables, on ne peut pas savoir dans quel ordre elles seront effectivement franchies. L'évolution n'est donc pas unique [Scorletti , 2006].

e. Structures fondamentales pour la modélisation des systèmes

Les RdPs permettent de modéliser un certain nombre de comportements importants dans les systèmes tels que le parallélisme, la synchronisation, le partage de ressources, la mémorisation et la lecture d'information, la limitation d'une capacité de stockage. [Scorletti , 2006].

1. Parallélisme

Le parallélisme représente la possibilité que plusieurs processus évoluent simultanément au sein du même système. On peut provoquer le départ simultané de l'évolution de deux processus à l'aide d'une transition ayant plusieurs places de sortie.

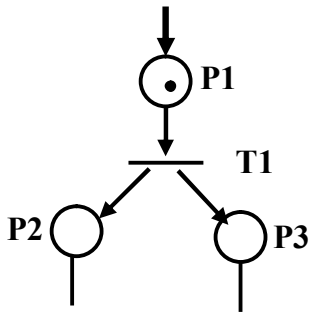

Figure 1.28 Structure du parallélisme [Scorletti, 2006].

Le franchissement de la transition **T1** met une marque dans la place **P2** (ce qui marque le déclenchement du processus 1) et une marque dans la place **P3** (ce qui marque le déclenchement du processus 2) [Scorletti , 2006].

2. Synchronisation

La synchronisation est modélisée sous cinq formes :

- **Synchronisation mutuelle (Par rendez vous)**

La synchronisation mutuelle ou par rendez-vous permet de synchroniser les opérations de deux processus. La Figure 1.29 montre un exemple de deux processus, le franchissement de la transition **T7** ne peut se faire que si la place **P12** du **processus 1** et la place **P6** du **processus 2** contiennent chacune au moins un jeton. Si ce n'est pas le cas, par exemple la place **P12** ne contient pas de jetons, le **processus 2** est bloqué sur la place **P6** : il attend que l'évolution du **processus 1** soit telle qu'au moins un jeton apparaisse dans la place **P12**.

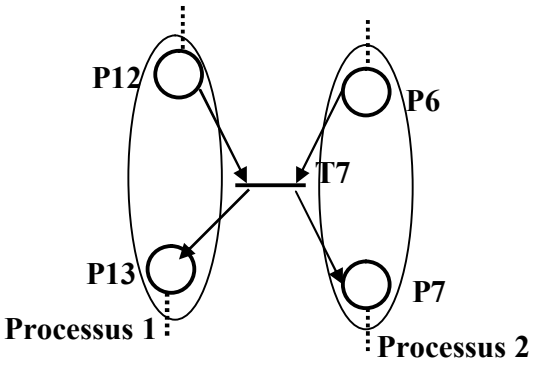

Figure 1.29 Synchronisation mutuelle [Scorletti, 2006].

- **Synchronisation par signal (sémaphore)**

Dans la synchronisation par signal Les opérations du **processus 2** ne peuvent se poursuivre que si le **processus 1** a atteint un certain niveau dans la suite de ses opérations. Par contre, L'avancement des opérations du **processus 1** ne dépend pas de l'avancement des opérations du **processus 2**.

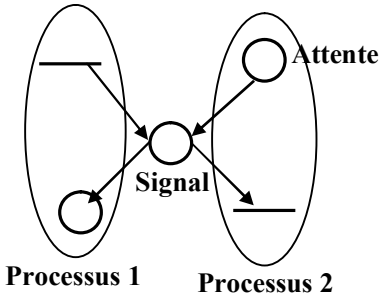

Figure 1.30 Synchronisation par sémaphore [Scorletti , 2006].

Si la place **Signal** est marquée et la place **Attente** ne l'est pas, cela signifie que le processus **P1** a envoyé le signal mais le processus **P2** ne l'a pas encore reçu. Si, par contre, la place **Signal** n'est pas marquée et que la place **Attente** est marquée, cela signifie que le processus **P2** est en attente du signal [Scorletti , 2006].

- **Synchronisation par Partage de ressources**

Cette structure va modéliser le fait qu'au sein du même système plusieurs processus partagent une même ressource [Scorletti, 2006], en utilisant le principe de l'exclusion mutuelle.

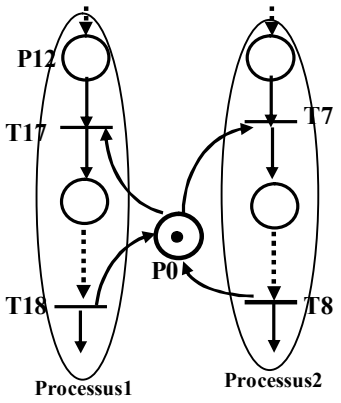

Figure 1.31 Synchronisation par Partage de ressources [Scorletti, 2006],

La marque dans la place **P0** présente une ressource mise en commun entre le **processus 1** et le **processus 2**. Le franchissement de la transition **T17** lors de l'évolution du **processus 1** entraîne la consommation du jeton présenté dans la place **P0.** La ressource que constitue cette marque n'est alors plus disponible pour l´évolution du **processus 2.** Lorsque la transition **T18** est franchie, un jeton est alors est placé dans la place **P0** : la ressource devient alors disponible pour l'évolution des deux processus.

- **Synchronisation par Mémorisation**

Dans tous les modèles, on peut compter le nombre de tirs d'une transition en utilisant une place sans sortie [Scorletti , 2006].

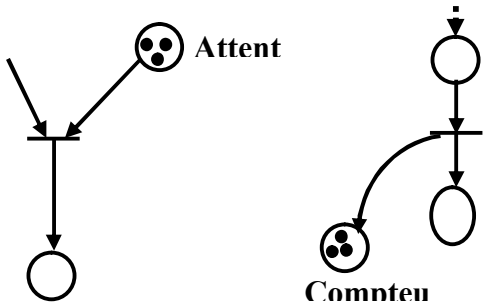

Figure 1.32 Synchronisation par Mémorisation [Scorletti, 2006].

Dans La Figure 1.32 les places **attente** et **Compteur** peuvent servir à indiquer combien d'instances d'un processus sont en attente.

- **Capacité limitée**

Dans la Figure 1.33, pour que la transition **T3** soit franchissable, il est nécessaire que la place **P5** contienne des jetons. Le marquage de **P5** ne permet que deux franchissements successifs de **T3**. La transition **T3** sera à nouveau franchissable si le franchissement de la

transition **T4** permet de mettre des jetons dans la place **P5.** Au total, la place **P4** ne pourra pas contenir plus de 3 jetons [Scorletti , 2006].

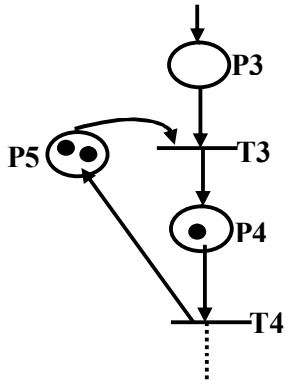

Figure 1.33 Synchronisation par Capacité limité [Scorletti , 2006].

f. Propriétés qualitatives des RdP

Après la modélisation d'un système, la question qui se pose qu'est qu'on peut faire avec ce modèle ? Les réseaux de Petri ont une grande capacité d'analyse des propriétés et des problèmes associés avec les systèmes concurrents .Deux types de propriétés peuvent être étudiées :

- Les propriétés qui dépendent du marquage initial qui sont les propriétés comportementales.
- Les propriétés qui dépendent de la structure du réseau qui sont les propriétés structurelles [Murata ,1989].

Nous nous intéressons ici aux propriétés comportementales

1. Propriétés Comportementales d'un RdP

- **Accessibilité**

L'accessibilité est une propriété fondamentale pour étudier les propriétés dynamiques du système. Le franchissement d'une transition va changer la distribution des jetons sur le réseau, partant d'un marquage **M** et aboutissant à un autre marquage **M'**.

Un marquage $\mathbf{M_1}$ est dit accessible à partir d'un marquage $\mathbf{M_0}$ s'il existe une séquence de franchissement qui transforme $\mathbf{M_0}$ vers $\mathbf{M_1}$.

L'ensemble des marquages accessibles depuis $\mathbf{M_0}$ par une séquence de franchissement est noté par **A (N, $\mathbf{M_0}$)** ou **A ($\mathbf{M_0}$).**

L'ensemble de tous les marquages possibles accessibles depuis $\mathbf{M_0}$ est noté par **L (N,**

M_0) ou **L (M_0)** [Murata ,1989].

Le problème concernant l'accessibilité pour le réseau de Petri est de trouver si **M_n** $\in$ **A(M_0)** en donnant **M_n** et un réseau **(N , M_0)** [Vinh Duc,2005].

- **Bornitude**

Un RdP **(N, M_0)** est K-borné ou simplement borné si le nombre de jetons dans chaque place ne dépasse pas un nombre fini **K** pour tout marquage accessible de puis **M_0** [Murata ,1989].

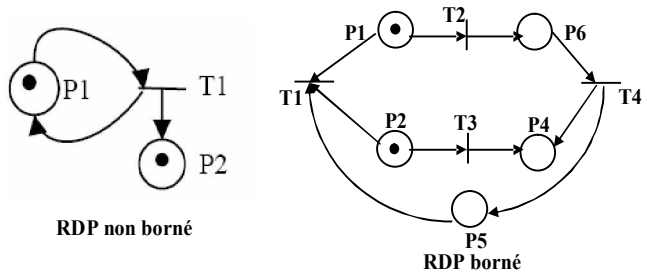

Figure 1.34 Exemple d'un RdP non borné et borné [Scorletti , 2006][Murata ,1989].

Pour le RdP non borné, la transition **T1** admet la place **P1** comme l'unique place d'entrée. La place **P1** a un jeton, la transition **T1** est franchissable. A chaque franchissement de **T1** s'ajoute un jeton dans la place **P2**, donc le marquage de celle-ci peut donc tendre vers l'infini [Scorletti , 2006].

- **Vivacité et blocage**

La notion de vivacité est liée à l'absence d'interblocage dans un système opérationnel. Une transition **T_j** est vivante pour un marquage initial **M_0** si pour tout marquage accessible **M_k**, il existe une séquence de franchissements à partir de **M_k** contenant **T_j**.

$$\forall M_k \in {}^*M_0, \exists\ S, M_k \mid S> \text{ et } S = \ldots.. Tj \ldots.$$

Un réseau **(N, M0)** est vivant si toutes ses transitions sont vivantes. [Scorletti, 2006]. La vivacité exprime le fait qu'à tout instant de l'évolution du système, le franchissement à terme d'une transition vivante **T** n'est jamais définitivement impossible [Vinh Duc,2005].

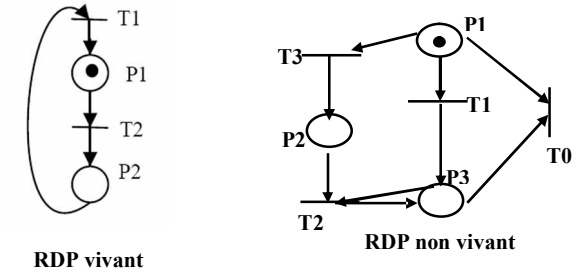

Figure 1.35 Exemple d'un RdP vivant et non vivant [Scorletti , 2006][Murata ,1989].

Un blocage (ou état puits) est un marquage pour lequel aucune transition n'est validée. Un RdP marqué est dit sans blocage pour un marquage initial $\mathbf{M_0}$ si aucun marquage accessible n'est un blocage. Le RdP marqué de la Figure 1.36 a pour blocage le marquage M3= [1, 0, 0,4].

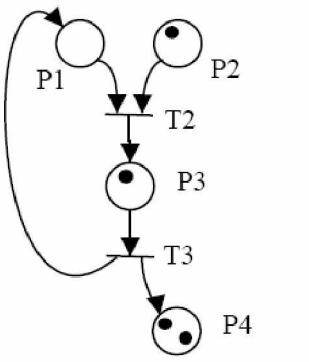

Figure 1.36 Exemple d'un RdP avec un blocage [Scorletti , 2006].

- **Réseau de Petri Réinitialisable et Home d'etat**

Un réseau de Petri est réinitialisable si pour chaque marquage **M** dans $\mathbf{R(M_0)}$, $\mathbf{M_0}$ est accessible à partir de **M**.

Un marquage $\mathbf{M_0}$ est Home d'état si pour chaque marquage **M** dans $\mathbf{A\ (M_0)}$, **M** est accessible à partir de $\mathbf{M_0}$ [Vinh Duc, 2005].

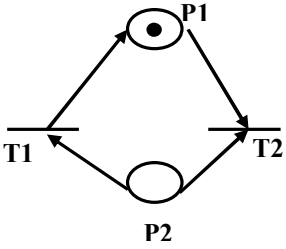

Figure 1.37 Exemple d'un RdP réinitialisable [Murata ,1989].

Un marquage $\mathbf{M_0}$ est Home d'état si pour chaque marquage **M** dans $\mathbf{A(M_0)}$, **M** est à partir de $\mathbf{M_0}$.

- **Couverture**

Un marquage **M** dans un RdP $\mathbf{(N,M_0)}$ est dit couvré s'il existe un marquage **M'** dans $\mathbf{R(M_0)}$ telque $\mathbf{M'(p) \geq M(p)}$ pour chaque place du réseau [Murata ,1989].

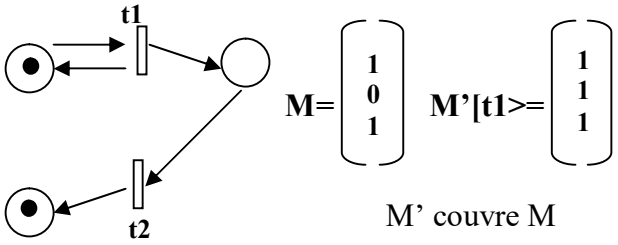

Figure 1.38 Exemple d'un RdP couvré [Murata ,1989].

▪ **Persistance**

Un RdP **(N, M_0)** est dit persistant pour n'importe quel deux transitions, si le franchissement d'une transition ne doit pas inhiber les autres. Une transition dans un RdP persistant une fois elle est validée, elle doit rester validée jusqu'à son franchissement [Murata ,1989].

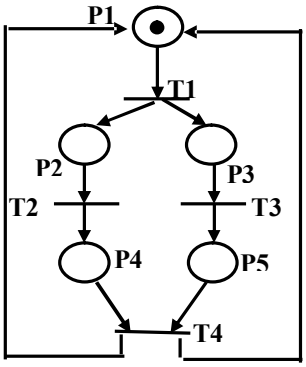

Figure 1.39 Exemple d'un RdP persistant [Murata ,1989].

g. Méthodes d'analyse

Les principales méthodes d'analyse des propriétés peuvent être classées en trois groupes :

▪ Méthode d'analyse par graphe de marquage ou de l'arborescence de couverture. Une arborescence est un graphe particulier composé d'arcs orientés qui divergent progressivement à partir d'un noeud appelé racine de l'arborescence.

▪ Méthode d'analyse par algèbre linéaire ou équation de matrices. L'algèbre linéaire rend possible l'étude de propriétés structurelles, en particulier par la recherche des invariants linéaires de places (composantes conservatives) et des invariants linéaires de transitions (composantes répétitives) et donne aussi des résultats dépendant du marquage. On peut en déduire si le réseau est borné ou sauf, sans blocage et s'il est réinitialisable ou vivant.

▪ Méthode d'analyse par Réduction de RdP en appliquant certaines règles de réduction particulières.

Nous nous intéressons ici à la méthode d'analyse par arbre de marquage, qui consiste à construire le graphe de tous les marquages du réseau et déduire les propriétés grâce aux techniques de la théorie de graphes.

1.Graphe de marquages

Le graphe des marquages accessibles **G(N, Mo)** est défini comme le graphe dont les nœuds sont les marquages accessibles de **A(N, Mo)** et dont les arcs sont les noms des transitions impliquées dans les franchissement menant d'un marquage à un autre [Vinh Duc,2005].

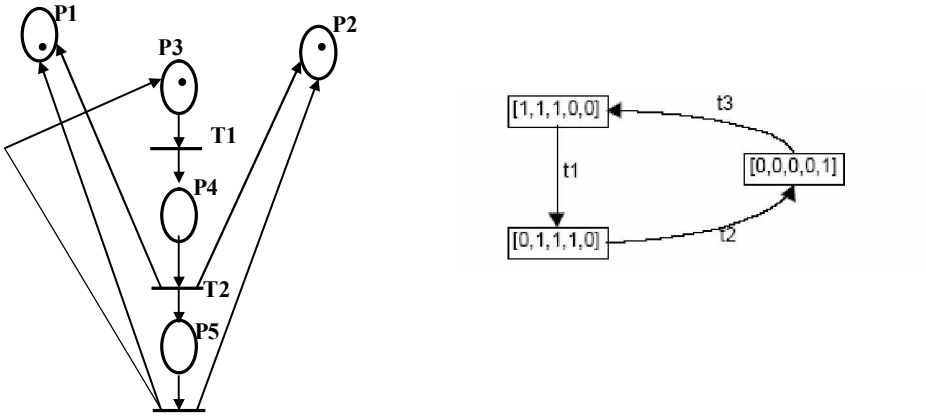

Figure 1.40 Exemple d'un graphe de marquage [Vinh Duc, 2005].

Le graphe des marquages permet d'explorer tous les marquages possibles dans un RdP. Si le graphe de marquages accessibles est fini, c'est la situation la plus favorable car toutes les propriétés peuvent être déduites simplement. L'alternative, si le réseau n'est pas borné, le graphe de marquages sera infini et on aura le problème d'explosion d'espace d'états.

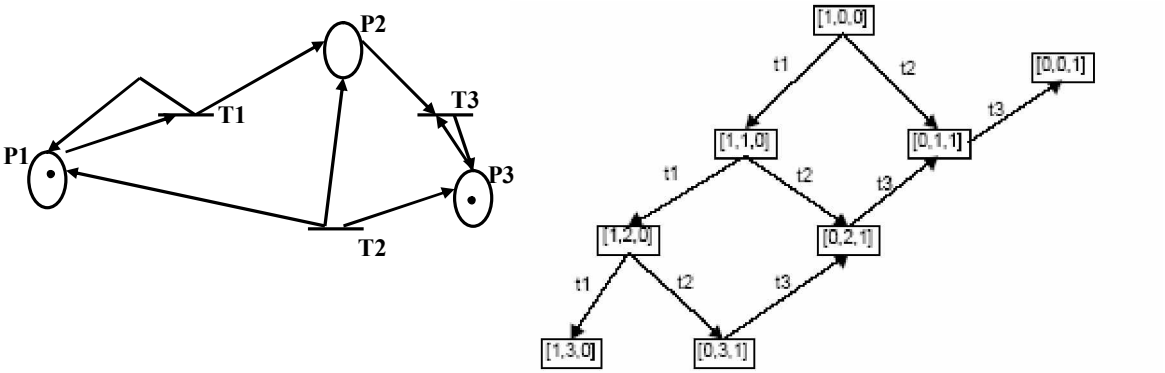

Figure 1.41 Exemple d'un graphe de marquage infini [Vinh Duc, 2005].

La solution pour ce problème consiste à construire le graphe de couverture. On doit accepter la perte des informations. Pour l'arbre infini, le nombre de jetons dans les places pouvant avoir une quantité arbitrairement grande de jetons, elle va être considérée comme une valeur particulière, notée par **ω**. Elle a les propriétés suivantes pour chaque entier **n.**

$$\omega + n = \omega$$

$$\omega - n = \omega$$

$$n \leq \omega$$

$$\omega \leq \omega$$

L'algorithme pour construire le graphe de couverture consiste à :

1. Assigner la racine à $\mathbf{M_0}$ et mettre **tag($\mathbf{M_0}$)** = **"nouveau"** ;

2. Tant qu'il existe un marquage ayant **tag = "nouveau",** nous faisons les étapes suivantes :

2.1 Choisir un **"nouveau"** marquage **M** ;

2.2 Si **M** est égal à un autre noeud dans le chemin de noeud à **M**, donc **tag(M)** = **"old"** et on continue (l'étape 2) avec un autre nouveau marquage ;

2.3 S'il n'y a pas de transition franchissable dans **M**, donc on assigne **tag(M)** = **"terminal"** ;

2.4 Tant qu'il existe une transition franchissable **t** dans **M**, on fait les étapes suivantes :

2.4.1 Calculer **M'** tel que **M[t > M'** ;

2.4.2 S'il existe un noeud **M'** dans le chemin de la racine à **M'** tel que :

$$\forall p \in P : M'(p) \geqslant M\ (p) \text{ et } M \neq M'$$

Càd **M < M'**, donc pour chaque **p** tel que **M'(p) > M (p)** on replace **M'(p)** par **ω.**

2.4.3 Ajouter le noeud **M'** à l'arbre et dessiner un arc de **M** à **M'** avec l'étiquette t, et on assigne **tag(M')** = **"nouveau"**.

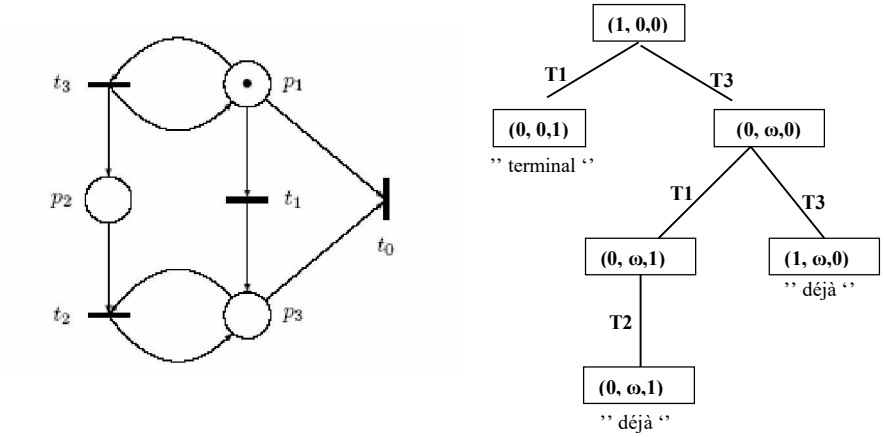

Figure 1.42 Exemple d'un graphe de couverture [Vinh Duc, 2005].

1.4 Conclusion

Dans ce chapitre nous avons présenté les concepts et les principes fondamentaux des systèmes à événements discrets et plus particulièrement les systèmes de production.

La supervision de ces systèmes, repose sur l'existence d'un modèle qui modélise la partie opérationnelle du système. Pour cela nous avons abordé les différentes méthodes de modélisation et nous avons mis l'accent sur UML.

Ainsi la vérification de certaines propriétés des systèmes de production fait une partie importante pour leur supervision, pour cette raison, nous avons présenté les différents formalismes apportés pour ce but et les réseaux de Petri sont le formalisme le plus adapté pour ce type de systèmes.

Dans le prochain chapitre notre intérêt sera sur la proposition d'une approche de modélisation d'un processus de production, utilisant UML avec un passage vers les réseaux de Petri, pour un but de vérification. Toute la procédure est en vue de concevoir un superviseur de contrôle pour ce processus.

Proposition d'une méthode de conception d'un superviseur de contrôle pour un processus de production distribué

Dés que le but d'un système de production est d'avoir une productivité optimale, l'existence d'un système de supervision devienne nécessaire. Un superviseur pour un système de production vise à assurer le bon déroulement, de l'ensemble des opérations exécutées par les processus de production. L'opération de supervision consiste à réaliser une synchronisation et une vérification de certaines propriétés relatives à l'aspect comportemental du système.

Tant que le rôle du superviseur est d'assurer la bonne synchronisation entre les opérations de production et de vérifier les propriétés relatives à la dynamique du système, nous avons recours à un modèle illustrant l'aspect statique de la partie opérative du système, tant qu'elle est le responsable de l'exécution réelle de l'ensemble des processus de production. Ainsi un autre modèle exprimant l'aspect comportemental, ce modèle doit être inspiré du celui de l'aspect statique. La conception de tel superviseur se base alors sur ces deux modèles.

Dans ce qui suit, nous présentons notre méthode de conception d'un superviseur de contrôle pour un processus de production distribué .Nous commençons par l'étude de transformation de modèles et nous étudions un outil adapté pour cela. Ensuite une démarche pour la modélisation et la vérification des processus de production en vue de leur supervision est présentée, dans laquelle nous expliquons comment la notion de transformation de graphes est utilisée via l'outil de transformation. Nous concluons par un rappel sur les principes de la méthode proposée.

2.1 Transformation de modèles

2.1.1 Pourquoi modéliser ?

Modéliser un système avant sa réalisation permet de mieux comprendre son fonctionnement. C'est également un bon moyen de maîtriser sa complexité et d'assurer sa cohérence.

Depuis longtemps, divers langages et techniques de modélisation sont utilisés lors des phases d'analyse et de conception, entre autres en développement logiciel. Se situant à un niveau

intermédiaire entre l'expression des besoins et le code développé pour répondre à ces besoins. Les modèles permettent une meilleure communication entre un informaticien et un client.

Du point de vue du développeur, un modèle permet de garder tout au long du cycle de vie du logiciel une vision globale sur le système à traiter, en offrant plusieurs niveaux d'abstraction.

2.1.2 Méta-modélisation

Tout comme un programme, un modèle destiné à être traité par une machine ne doit pas permettre d'ambiguïtés d'interprétation, et doit être exprimé en respectant des règles bien définies. Il s'agit là de l'objet de la démarche de méta-modélisation, qui s'attache à définir des formalismes pour les langages de modélisation. Une fois le formalisme défini, on peut garantir que tout modèle conforme à ce formalisme pourra être traité correctement [Meylan, 2006].

Un méta-modèle est un modèle d'un langage de modélisation. Le terme ''méta '' indique un niveau plus élevé, soulignant le fait qu'un méta-modèle décrit un langage de modélisation à un plus haut niveau d'abstraction que le langage de modélisation lui-même.

Un méta-modèle a deux caractéristiques principales. Premièrement, il doit capturer les caractéristiques essentielles du langage de modélisation, ainsi, il devrait être capable de décrire la syntaxe concrète, et la sémantique de ce langage [Clark, 2004].

Le tableau suivant définit quelques concepts qui différencient les niveaux d'abstraction de la méta-modélisation.

Méta-méta-modèle	Langage de spécification des méta-modèles
Méta-modèle	Définition du langage utilisé pour exprimer le modèle
Modèle	Abstraction du système
Système	Information et flux de contrôle d'un domaine

Tab 2.1 Niveaux d'abstraction de la méta-modélisation [Meylan, 2006].

- Au premier niveau, se trouve le système à étudier.
- Au second niveau, se trouve le modèle qui décrit certains aspects du premier niveau que nous voulons l'étudier: il peut s'agir d'un diagramme de classes UML, d'un modèle conceptuel de traitement MERISE, qui représente une vue abstraite des objets modélisés.
- Au troisième niveau se trouve le langage de modélisation ou méta-modèle : par exemple les définitions d'un MCD de MERISE d'un diagramme d'objet UML. C'est ce langage qui doit faire l'objet d'une spécification formelle.

- Au quatrième niveau se trouve le méta-méta-modèle : il s'agit d'un langage qui doit être assez générique pour définir les différents langages de modélisation existants, et assez précis pour exprimer les règles que chaque langage doit respecter pour pouvoir être traité automatiquement [Meylan, 2006]. Des exemples de méta-méta-modèles : DSL Tools de Microsoft, MOF (Meta Object Facility), KM3 (Kernel MetaMetaModel).

2.1.3 Transformation de modèles

2.1.3.1 Définitions

La notion de transformation de modèles est centrale pour L'ingénierie dirigée par les modèles ou (**MDI** : Model Driven Engineering). Une transformation de modèles prend comme entrée un modèle conformément à un méta-modèle donné et produit comme sortie un autre modèle conformément à un méta-modèle donné.

La transformation de modèles est définie comme étant le processus de convertir un modèle d'un système à un autre modèle [Czarnecki, 2006].

Deux types de transformations sont réalisables :

- Les transformations endogènes (si le méta-modèle source et cible sont identiques).
- Les transformations exogènes (si le méta-modèle source et cible sont différents).

Une autre classification indique qu'il y a les transformations de type modèle vers code et celles de type modèle vers modèle.

Une transformation de modèles peut également avoir plusieurs modèles source et plusieurs modèles cibles. Une caractéristique de transformation de modèles est qu'elle est un modèle puisque elle doit être conforme à un méta-modèle donné [Czarnecki, 2006].

Une présentation générale des principaux concepts impliqués dans la transformation de modèles est illustrée dans la Figure suivante :

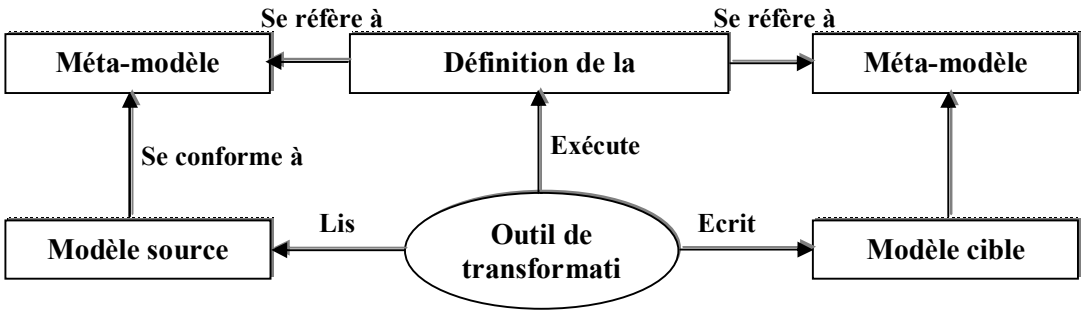

Figure 2.1 Concepts de base de la transformation de modèles [Czarnecki, 2006].

2.1.3.2 Transformations de type modèle vers modèle

La transformation de type modèle vers modèle est une transformation basée sur les modèles dans les deux cotés, parmi les types de cette transformation est les transformations de graphes qui nous intéressent dans notre méthode de conception.

2.1.3.3 Structure d'une transformation de type modèle vers modèle

Une transformation de modèles est principalement caractérisée par la combinaison des éléments suivants :

- Des règles de transformation, une règle de transformation est composée de deux parties: une partie gauche (**LHS**, Left Hand Side) qui accède au modèle source, et une partie droite (**RHS**, Right Hand Side) qui accède au modèle cible. Ces deux parties sont composées de variables, de modèles (patterns) et d'expressions logiques.
- Une relation entre le modèle source et le modèle cible. Pour certains types de transformations, la création d'un nouveau modèle cible est nécessaire, pour d'autres, la source et la cible sont le même modèle, ce qui revient en fait à une modification de modèle.
- Une spécification, certaines approches de transformation fournissent un mécanisme dédié de spécifications, tel que des pré-conditions et des post-conditions exprimées dans un langage. Des spécifications particulières de transformation peuvent représenter une fonction entre les modèles source et cible et peuvent être exécutables. Mais généralement, les spécifications décrivent des relations et elles ne sont pas exécutables.
- La planification d'application des règles, détermine l'ordre dans lequel sont appliqués les règles, sa forme est implicite, où l'ordre est défini par l'outil de transformation ou explicite où l'ordre d'exécution des règles est contrôlé à l'extérieur de l'outil de transformation.
- L'organisation des règles, définit comment composer plusieurs règles de transformation. Elle est soit modulaire, c'est la possibilité pour un module d'importer un autre, soit un mécanisme de réutilisation par l'héritage entre règles , soit les règles peuvent être organisées selon une structure dépendante du modèle source ou du modèle cible.
- La traçabilité, permet de garder la trace des liens entre éléments source et cible, d'analyser l'impact d'un changement d'un modèle sur un autre et de synchroniser entre les modèles

▪ La direction, il y a deux types, des règles unidirectionnelles, qui s'exécutent dans une seule direction et des règles bidirectionnelles, qui peuvent être exécutées dans les deux sens [Helsen, 2006] [Czarnecki, 2006].

2.1.3.4 Transformation de graphes

Comme les grammaires de Chomsky pour des textes, la transformation de graphes se fait grâce à des règles de transformation [Guerra, 2006]. Ces règles sont appliquées sur un graphe, et si les règles trouvent plusieurs concordances, elles effectuent les changements déterminés en opérant des remplacements, des ajouts ou des suppressions. Donc le processus de transformation s'effectue, en partant d'un graphe de départ, produire un graphe d'arrivée qui pourra être modifié ensuite [Scolas, 2007].

Une règle est constituée de deux parties, le Left Hand Side (**LHS**) et le Right Hand Side (**RHS**). Le LHS est une partie du graphe, destinée à être mise en concordance avec les parties du graphe (appelé host graph) où nous voulons appliquer la règle. Le RHS quant à lui décrit la modification qui sera effectuée, elle substitue la partie identifiée dans le host graph [Scolas, 2007].

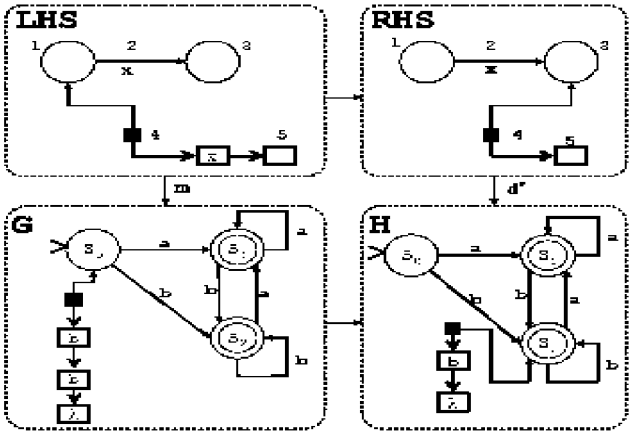

Figure 2.2 Exemple d'application d'une règle sur un graphe [Guerra, 2006].

La Figure 2.2 montre un exemple de dérivation, où une règle indiquant une étape dans la simulation d'un automate d'état finis est appliquée à un host graph ***G***. Le LHS montre deux états de l'automate (marqué en tant que 1 et 3) reliés par une transition (marquée 2). Le noeud 4 a un arc vers l'état actuel et un lien à une séquence des entrées. La valeur du premier élément de la séquence des entrées doit être égale à la valeur de la transition (variable **x**). La règle décrit la consommation de la première entrée et le changement de l'état actuel. Dans la dérivation, la variable **x** dans la règle est instanciée par la valeur **b** et l'état actuel est déplacé de $\mathbf{s_0}$ à $\mathbf{s_2}$ [Guerra, 2006]. Les résultats d'application de règle sont illustrés dans le graphe **H** [Guerra, 2006].

L'ordre dans lequel les règles d'une grammaire de graphes sont appliquées est arbitraire. Cependant, il est possible de spécifier un ordre en équipant les règles d'une priorité ou d'une couche. Dans le cas des priorités, des règles avec des priorités plus élevées sont appliquées d'abord .Dans le cas des couches, Les règles dans la première couche sont appliquées tant que possible. Puis, les couches suivantes sont consécutivement exécutées dans l'ordre croissant.

Parmi les outils permettant les transformations de graphes : **VIATRA**, **ATOM3**, **GreAT**, **UMLX**, **BOTL et AGG.** Nous nous intéressons dans notre méthode par l'outil ATOM3.

2.2 ATOM^3

2.2.1 Présentation

AToM^3 (A Tool for Multi-formalism and Meta-Modelling) est un outil de modélisation multi-paradigmes écrit en Pyton ,développé par le laboratoire MSDL (Modelling, Simulation and Design Lab) à l'université de McGill Montréal, Canada. Cet outil a été conçu en collaboration avec Juan de Lara de l'université de Madrid (UAM), Espagne [ATOM3, 2007].

Les deux principales fonctionnalités d'ATOM^3 sont la méta-modélisation et la transformation de modèles.

Dans AToM^3 les formalismes et les modèles sont décrits comme des graphes. Cet environnement génère un outil de manipulation graphique des modèles décrits dans un formalisme donné, à partir d'une méta-spécification de ce formalisme (généralement décrite dans le formalisme Entité/Relation). Les transformations entre modèles sont ensuite effectuées par réécriture des graphes (utilisant des grammaires de graphes) [De Lara, 2002].

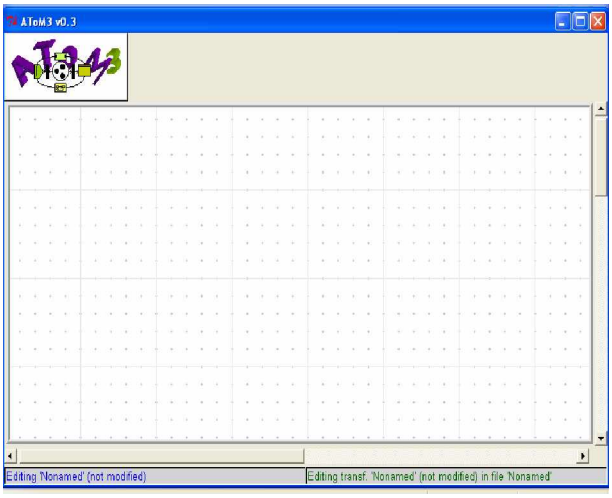

Figure 2.3 Interface d'ATOM^3.

2.2.2 Architecture d'ATOM^3

Le composant principal d'ATOM3 est le noyau (Kernel), qui est responsable du chargement, de la sauvegarde, de la création et de la manipulation des modèles (à tout méta-niveau). Par défaut, les méta-modèles et les méta-méta-modèles sont chargés quand ATOM^3 est lancé. Ces méta-méta-modèles permettent de modéliser les méta-modèles (modélisation des formalismes). Le formalisme Entité-Relation avec des contraintes est utilisé dans le méta-méta-niveau .Les contraintes sont du code en Pyton et le concepteur doit spécifier si ces contraintes sont (pré, post et sur quel événement une contrainte est évaluée).

Dans la modélisation à ce niveau, les entités qui doivent apparaître sur les modèles sont spécifiées ensemble avec leurs attributs, leurs apparences graphiques, leurs contraintes et leurs actions. Chaque relation entre deux entités peut être caractérisée par une contrainte en terme de cardinalités [De Lara, 2002].

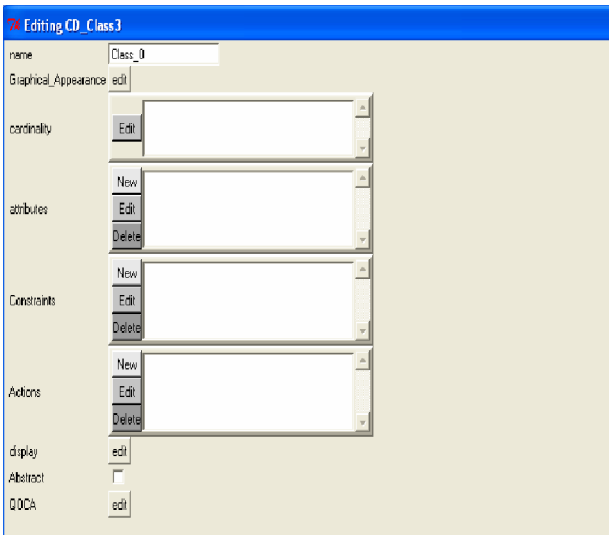

Figure 2.4 Editions des caractéristiques d'une entité.

2.2.2.1 Attributs

En générale, dans ATOM^3, il y a deux types d'attributs :

- Les attributs réguliers, qui sont utilisés pour identifier les caractéristiques d'une entité.
- Les attributs générateurs, qui permettent de générer d'autres propriétés [De Lara, 2002].

Chaque attribut a un type et un nom et une valeur initiale, comme il est montré dans la Figure 2.5.

Dans ATOM^3, chaque type a une classe Pyton, qui est responsable à l'édition des valeurs et les tests de leur validité. Il y a deux types de base :

- Type régulier tel que String, Integer, Float, Boolean…etc.

- Type générateur qui permet de générer des attributs, des contraintes, des attributs graphiques [De Lara, 2002].

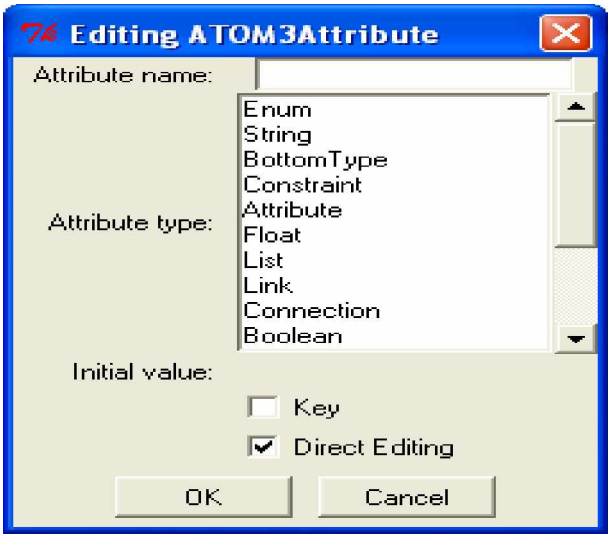

Figure 2.5 Edition des valeurs d'un attribut.

2.2.2.2 Contraintes et actions

Dans ATOM[3] Une contrainte est un code en Pyton ou des expressions en OCL, il y a deux types de contraintes :

- Des contraintes locales associées à une entité, elle doit contenir que des attributs pour cette entité.
- Des contraintes globales, ou les informations de toutes les entités du modèle sont utilisées.

La Figure 2.6 montre la structure d'une contrainte. Une contrainte est composée de :

- Un nom de contrainte.
- Un événement déclenchant l'évaluation de la contrainte, l'événement peut être sémantique tel que la sauvegarde d'un modèle, création d'une entité ou graphique ou structurel, tel que le déplacement ou la sélection d'une entité.
- Une spécification quand la contrainte doit être évaluée, avant l'événement (pré-condition) ou après (post-condition).
- Une zone pour écrire un code Pyton ou en OCL [De Lara, 2002].

Si la pré-condition d'un événement échoue alors ce dernier ne va pas être exécuté, si la post-condition d'un événement échoue ce dernier ne va pas être accompli.

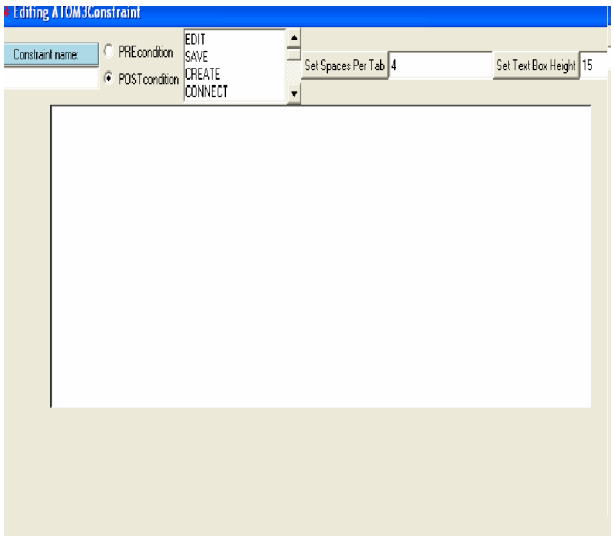

Figure 2.6 Structure d'une contrainte.

Une action est similaire à une contrainte sauf qu'elle a d'autres effets et elle est un code en Pyton seulement.

Une action est composée de :

- Un nom d'action.
- Un événement déclenchant l'action, l'événement peut être sémantique tel que la sauvegarde d'un modèle, création d'une entité ou graphique ou structurel tel que le déplacement ou la sélection d'une entité.
- Une spécification quand le code de l'action doit être exécuté, avant l'événement (pré-condition) ou après (post-condition).
- Une zone pour écrire un code Pyton [De Lara, 2002].

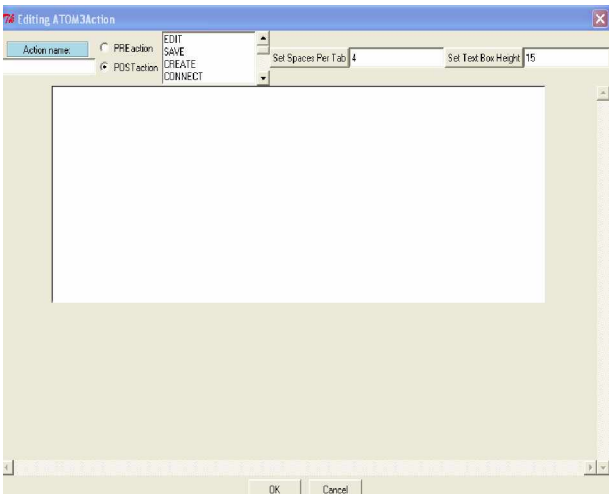

Figure 2.7 Structure d'une action.

2.2.2.3 Transformation de graphes

Dans ATOM[3], une fois un modèle est chargé il peut être transformé à un autre modèle équivalent, exprimé par un autre formalisme ou dans le même formalisme.

Dés que les modèles sont représentés à l'intérieur sous forme de graphes alors les transformations entre modèle s'effectue par des grammaires de graphes.

Une grammaire de graphes est un ensemble de règles, avec une action initiale et une action finale. Elle a un nom et un ensemble de menus permettant sa manipulation, tel que la sauvegarde, l'exécution…etc. comme le montre la Figure suivante :

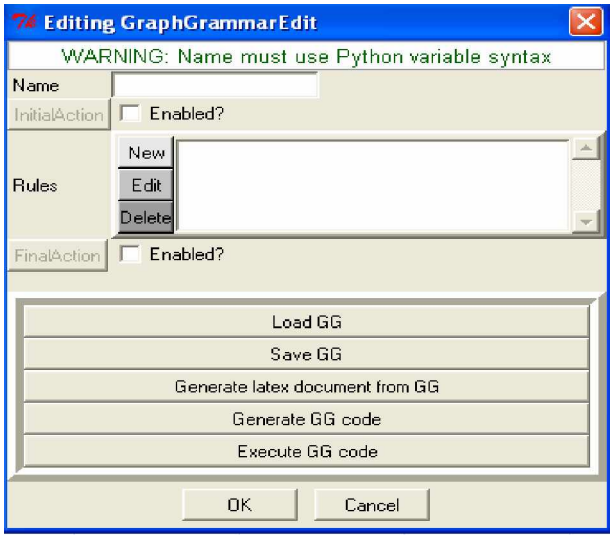

Figure 2.8 Structure d'une grammaire.

Chaque règle est constituée de:

- Un nom spécifique pour la règle.
- Une priorité indiquant l'ordre dans lequel la règle est appliquée.
- Une partie gauche (Left Hand Side : LHS) qui est un graphe.
- Une partie droite (Right Hand Side : RHS) qui peut être un graphe.
- Une condition (Un code en Pyton) qui doit être vérifiée avant que la règle soit appliquée.
- Une action (Un code en Pyton) qui doit être exécutée une fois que la règle est appliquée.

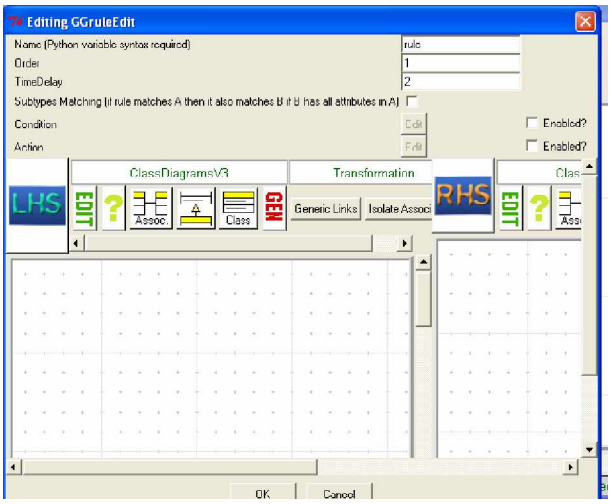

Figure 2.9 Structure d'une règle.

Les deux parties gauche et droite peuvent être des graphes illustrés par des formalismes différents, et pour faire la transformation il faut charger les méta-modèles qui concernent ces formalismes.

L'exécution de règles peut être continue (aucune interaction d'utilisateur) ou étape par étape ou l'utilisateur doit intervenir après chaque exécution d'une règle, ou d'une manière arbitraire [De Lara, 2002].

2.2.2.4 Langage Pyton

Python, est un langage autorisant la programmation impérative et la programmation orientée objet, inspiré du langage C, portable sur la plupart des systèmes d'exploitation (Unix, Mac Os, Windows, ..), placé sous la licence BSD (pour Berkeley Software Distribution) et disponible gratuitement, aussi bien utilisable pour écrire des programmes de quelques lignes que des projets très complexes [Pyton, 2008].

La syntaxe de Python est très simple et combinée à des types de données évolués (listes, dictionnaires,...), conduit à des programmes à la fois très compacts et très lisibles. A fonctionnalités égales, un programme Python est souvent de 3 à 5 fois plus court qu'un programme C ou C++ (ou même Java) équivalent, ce qui représente en général un temps de développement de 5 à 10 fois plus court et une facilité de maintenance largement accrue.

Python est orienté-objet. Il supporte l'héritage multiple et la surcharge des opérateurs. Dans son modèle objets, et en reprenant la terminologie de C++, toutes les méthodes sont virtuelles. Il intègre, un système d'exceptions, qui permettent de simplifier considérablement la gestion des erreurs.

Un programme écrit en Python est aussi bien interprète que compilé. Les programmes sources seront écrits dans des fichiers dont le nom aura l'extension ''**.py''**, et seront exécutés par la commande **Python nom_prograamme.py**, lancé dans un terminal [Pyton, 2008].

Notre utilisation de ce langage est limitée à certaines instructions, mais la plupart des autres instructions sont propres à ATOM^3, mais sur la base de ce langage.

2.3 Modélisation de processus de production distribué

Un processus de production distribué est un ensemble de relations entre des événements et des activités dans le système de production .Une description d'un processus de production est utilisée pour décrire comment le système de production fonctionne , dans le sens de contrôler son fonctionnement .

La distribution signifie que le processus de production est composé d'un ensemble de composants, chacun a son propre contrôle et capable d'effectuer certaines activités.

Comme il est mentionné dans le chapitre précèdent, pour concevoir un superviseur de contrôle pour un système de production, il faut disposer d'un modèle modélisant la partie physique, qui est capable d'exécuter l'ensemble des processus. Ces modèles doivent présenter la structure et le comportement du système et permettent l'analyse de ses propriétés. Donc la modélisation de cette partie est une étape initiative pour notre méthode de conception.

2.3.1 Choix des moyens de modélisation

Les systèmes de production sont considérés comme des systèmes a événements discrets. Les formalismes les plus connus pour la représentation des DES, sont les automates d'états finis, les réseaux de Petri, la logique temporelle…etc. En effet ces techniques souffrent des problèmes d'explosion d'états puisque les DES ont des structures complexes. Il est alors nécessaire de faire recours aux approches modulaires.

Une approche modulaire consiste à identifier les composants d'un système de production, représenter chacun séparément, dans le but de faciliter la modélisation des systèmes complexes et les rendre plus compréhensibles [Bordbar, 2000].

UML étant le langage standard de modélisation orientée objets .Il offre une gamme d'outils étendue, permettant de représenter sous forme graphique divers aspects d'un système à traiter. Il prend le concepteur durant le cycle de vie de conception, commençant par la description fournie par des utilisateurs ou des experts vers le logiciel final.

Notre méthode de conception est basée alors sur le langage UML, pour la modélisation de l'aspect statique du processus de production distribué.

Pour offrir une assistance réelle dans les niveaux de Modélisation, il est nécessaire qu'un modèle puisse être traité automatiquement. Ce traitement est effectué par l'outil ATOM^3.

Cet outil permet une phase de méta-modélisation .La méta-modélisation correspond à une phase d'abstraction de haut niveau. Elle permet de mettre en place les concepts fondamentaux qui seront utilisés au niveau utilisateur dans la phase de modélisation du processus.

2.3.2 Démarche de modélisation d'un processus de production distribué

Comprendre un processus de production revient à présenter les deux concepts caractérisant ce processus :

- Son pôle-ontologique : ce que le processus est, décrivant sa structure.
- Son pôle fonctionnel : ce que le processus fait, décrivant ses activités.

Nous considérons dans ce cas deux niveaux de modélisation :

- **Niveau structurel**

  Dans ce niveau, on considère deux aspects distincts :

  - Un aspect décrivant d'une manière textuelle de quoi consiste le processus et de quoi il est constitué ; et comment se fait l'enchaînement des activités du processus. Nous utilisons pour cela le diagramme de cas d'utilisation d'UML.
  - Un aspect décrivant d'une manière graphique la structure du processus de production en modélisant les composants du processus et leurs relations. Nous utilisons pour cela le diagramme de classes d'UML.

- **Niveau Comportemental**

Dans le niveau comportemental, nous considérons l'enchaînement d'activités du processus de production, pour le comportement normal et avec l'évitement du comportement anormal. Ce niveau doit permettre la vérification des propriétés comportementales du processus de production. Pour cela, nous utilisons le formalisme réseau de Petri.

2.3.2.1 Modélisation structurelle

a. Diagramme de cas d'utilisation

Le diagramme des cas d'utilisation pour un processus de production, est une description des objectifs du processus et comment les activités constituant ce processus sont planifiées.

L'étude du diagramme des cas d'utilisation permet au concepteur de reconnaître les individus d'un système qui sont des objets dans la terminologie UML.

Le diagramme des cas d'utilisation se présente sous forme textuelle contenant :

- Des membres réalisant des opérations, qui sont présentés par des noms, se sont des acteurs du diagramme de cas d'utilisation.
- Des préconditions qui décrivent dans quel état doit être le processus de production avant qu'un événement puisse être déclenché.
- Des scénarios qui sont décrits sous la forme d'échanges d'évènements entre les acteurs. On distingue le scénario normal, qui se déroule quand il n'y a pas d'erreur, et le scénarii d'exception qui décrit les cas d'erreurs.
- Des postconditions qui décrivent l'état du système à l'issue des différents scénarios.

Avec ces points, le diagramme de cas d'utilisation va permettre au concepteur de déterminer la stratégie de synchronisation qui doit la suivre dans la conception du superviseur de contrôle.

Le scénario de fonctionnement du processus de production présenté dans le diagramme de cas d'utilisation correspond à la spécification H de la théorie de supervision de Ramadge &Wonham définie dans le chapitre précédent.

b. Diagramme de classes

Le but d'un diagramme de classes est d'identifier les classes dans un modèle. Comme nous avons mentionné dans le chapitre précédent, les classes dans UML sont représentées graphiquement par des rectangles contenant trois parties. La première est le nom de la classe, la deuxième est une liste d'attributs et la dernière est un ensemble de méthodes. Les classes sont connectées par des liens de type association ou de généralisation.

Pour notre méthode de conception du superviseur de contrôle, le diagramme de classes pour un processus de production distribué permet de modéliser la structure statique de ce processus, il est composé de classes illustrant les acteurs du diagramme de cas d'utilisation. Pour modéliser la synchronisation entre les différents acteurs du processus, nous avons proposé que :

- Chaque classe du diagramme doit avoir quatre variables booléennes indiquant, l'état de l'acteur dans le processus de production à un instant précis. Ces attributs sont :
  - **dp:** (décision point) un point de décision sur la prochaine activité à réaliser.
  - **wait** :(Attendre) arrêter l'activité en cours en attendant qu'une autre activité se termine.

- **out:** (sortir) l'activité en cours est terminée, en sortant de la zone de travail d'un processus de production.
- **work:** Une activité d'un processus de production est en cours d'exécution.

➢ Chaque classe doit avoir cinq méthodes qui permettent de changer l'état un acteur. Ces méthodes sont :

- **new ():**Une nouvelle activité vient d'être commencé.
- **ab ():** (abort) stopper une activité d'un processus de production en attendant l'arrivée d'un événement.
- **start ():** reprendre une activité stoppée par la méthode **ab ().**
- **go ():** pas d'attente, exécuter une activité après une décision de la commencer.
- **exit ():** sortir de la zone de travail, une activité en cours est terminée normalement.

Une classe dans le diagramme de classe pour un processus de production est illustrée dans la Figure suivante :

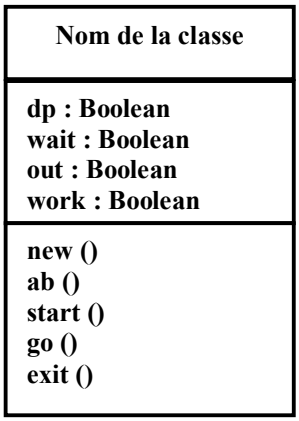

Figure 2.10 Structure d'une classe d'un processus de production distribué.

Dans notre travail, les codes de ces méthodes ne nous nous intéressent pas puisque dans la politique de supervision, ce qui est important c'est la synchronisation entre l'exécution de ces méthodes et pas comment chaque méthode est implémentée.

Les relations entre les classes sont des associations indiquant depuis le diagramme de cas d'utilisation s'il y a une synchronisation entre deux acteurs ou non.

Le diagramme de classes pour un processus de production distribué correspond au modèle Plant de la théorie de supervision de Ramadge &Wonham.

c. Modélisation automatique avec ATOM^3

Pour automatiser la modélisation de la structure du processus de production et plus précisément, avoir un moyen de modélisation du diagramme de classes pour ce processus, nous avons utilisé l'outil ATOM^3.

La modélisation avec ATOM^3 correspond à la phase d'utilisation des fonctions génériques ou objets mis en place à une étape de méta-modélisation pour développer un modèle représentant un système .Donc la méta-modélisation nous permet de définir de manière abstraite les différents concepts communs aux éléments d'un modèle.

La modélisation automatique du diagramme de classes avec ATOM^3, doit passer par les étapes suivantes:

1. Chargement d'un méta-modèle pour les diagrammes de classes, qui se trouve dans les formalismes principaux d'ATOM^3.
2. Concevoir le méta-modèle du processus de production selon la structure définie dans la Figure 2.10. Ce méta- modèle est composé d'une méta-classe présentant un composant du processus de production qui possède un nom ,les attributs et les méthodes mentionnées auparavant et une association avec la même méta-classe, avec une cardinalités égale à 0-N, comme le montre la Figure suivante :

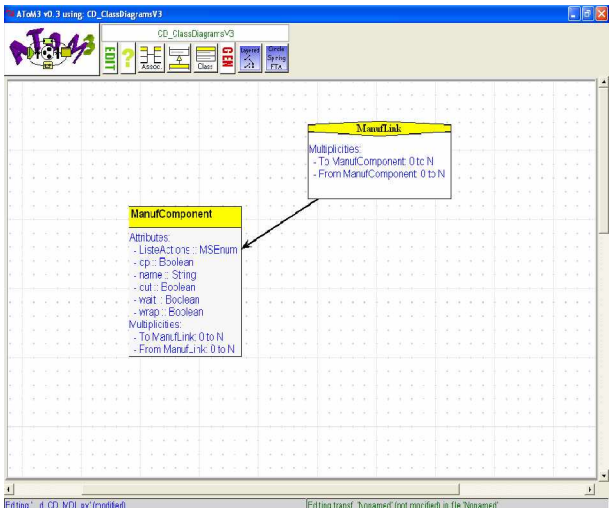

Figure 2.11 Méta-modèle pour le diagramme de classes.

3. A partir de ce méta-modèle, ATOM^3 génère un outil de modélisation de diagramme de classes pour un processus de production distribué. Cet outil offre un ensemble de boutons de manipulation de ce diagramme. Cet outil est illustré dans la Figure suivante :

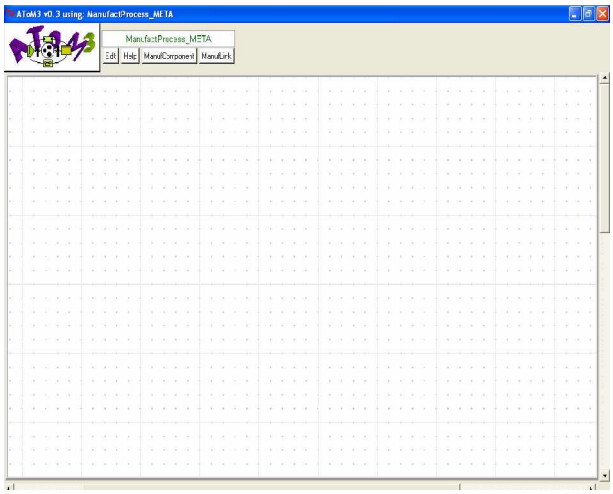

Figure 2.12 Outil de modélisation généré par ATOM[3].

2.4 Vérification du processus de production distribué

La vérification du processus de production est une étape très importante dans la conception du superviseur de contrôle, car elle s'intéresse à vérifier les propriétés comportementales qui concernent l'aspect dynamique du processus, ainsi que la définition de la bonne démarche de fonctionnement que doit respecter le processus pour éviter certains problèmes.

Pour cela, cette étape nécessite un modèle illustrant la dynamique du processus de production d'une part, et d'autre part il doit permettre une vérification efficace des propriétés comportementales de ce processus.

2.4.1 Choix des moyens de vérification

Tant que les systèmes de productions sont des systèmes a événements discrets .La plupart du temps, leur comportement est déterminé par des événements discrets qui apparaissent dans ces systèmes. Ce qui engendre un comportement complexe, nécessitant un formalisme très efficace pour modéliser le fonctionnement du système.

Pour ce type de systèmes est utilisé plusieurs formalismes tel que les statecharts d'UML, les automates d'états finis, le modèle de checking…etc. Mais ces formalismes n'ont pas satisfait la vérification de toutes les propriétés tel que, le non déterminisme, le parallélisme, le blocage…etc [Uzam, 1998] et n'ont pas la capacité suffisante pour modéliser la dynamique de ces systèmes.

Les réseaux de Petri comme un outil graphique et formel, sont de plus en plus employés dans la modélisation, l'analyse, la conception et la commande des systèmes à événement discrets [Ramadge, 1987]. Leur théorie permet d'analyser les propriétés de base des DEDS telle que la synchronisation, la concurrence, les conflits, le partage de ressources, les blocages [Bordbar,

2000]. Ainsi la simplicité de leur aspect graphique leur permet de simplifier la modélisation des systèmes complexes. Ainsi nous pouvons également évaluer l'exécution du système modéliser par un réseau de Petri. Un autre point c'est que l'outil ATOM[3] supporte ce formalisme, et permet sa méta-modélisation.

Notre méthode de conception se base donc sur le formalisme réseau de Petri pour la modélisation de l'aspect dynamique du processus de production et la vérification de ses propriétés comportementale et la génération de la bonne stratégie de fonctionnement de ce processus.

Pour, bénéficier de l'automatisation de cette étape, nous avons introduit un autre outil de vérification basé sur les réseaux de Petri qui est l'outil INA.

2.4.2 INA

INA (Integrated Net Analyzer) est un programme qui permet d'analyser des modèles réseau de Petri, il est développé par le Professeur Peter H. Starke. Avec son menu interactif, on peut éditer, réduire, exécuter et analyser un modèle réseau de Petri [Roch, 1999], comme le montre la Figure suivante :

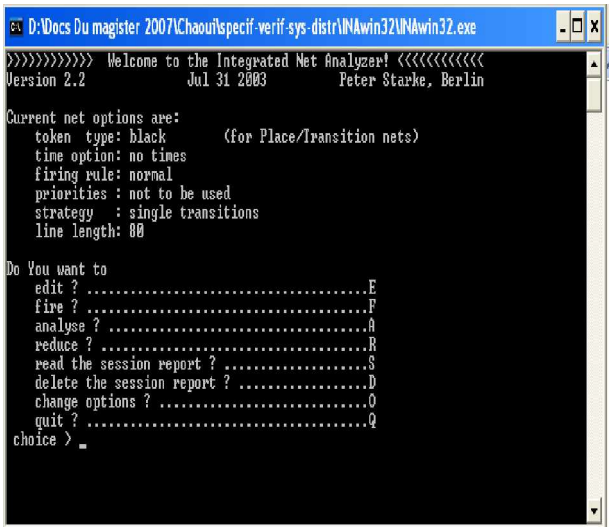

Figure 2.13 Interface de l'outil INA.

Pour l'analyse d'un modèle, cet outil doit accepter en entrée un fichier texte qui porte l'extension ''**.pnt**''.Ce fichier est une description de la structure du modèle à analyser, il doit suivre la syntaxe suivante :

- Chaque ligne est utilisée pour décrire une place du réseau a analyser.

- Chaque ligne commence par un numéro de place suivi par un espace ensuite le nombre de marquage dans la place, si cette place a des transitions d'entrée et de sortie alors ils sont présentées comme des listes séparées par des virgules.

Pi nb_jeton_Pi [T_Pré],[T_Post]
Pj nb_jeton_Pj [T_Pré],[T_Post]
.
.

Figure 2.14 Structure du fichier d'entrée pour l'outil INA.

L'analyse s'effectue en tapant le caractère ''**A**'', ensuite le nom du fichier texte à analyser, l'outil va présenter des choix sur le type d'analyse que l'utilisateur veut faire (Figure 2.15) et ensuite, il affiche les résultats demandés concernant les propriétés du modèle analysé.

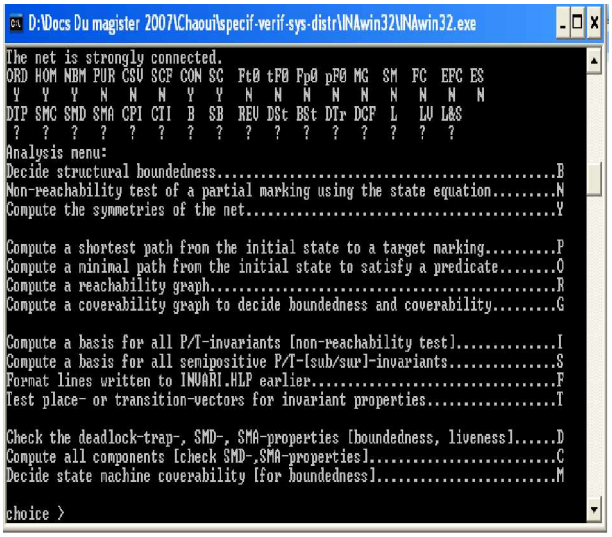

Figure 2.15 Différents choix d'analyse d'un modèle.

2.4.3 Démarche de vérification d'un processus de production distribué

La vérification du processus de production, repose sur un modèle réseau de Petri, modélisant la dynamique de ce processus. Ce modèle doit être inspiré du modèle illustrant la structure du processus, c'est-à-dire le diagramme de classes. Pour cela il faut trouver une équivalence entre le diagramme de classes et les réseaux de Petri.

Dans la modélisation avec les réseaux de Petri, chaque place représente l'état dans le quel se trouve l'élément modélisé et chaque transition représente la réalisation d'une opération par cet élément.

Dans notre travail, et pour faire la correspondance entre le diagramme de classes et les réseaux de Petri, nous avons proposé que :

- Chaque classe du diagramme de classes est représentée par un réseau de Petri formé de cinq transitions et quatre places, tel que, chaque attribut de la classe est une place et chaque méthode est une transition, comme le montre la Figure suivante :

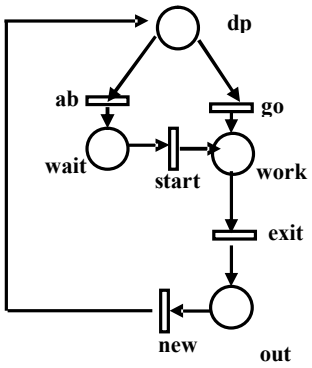

Figure 2.16 Un RdP correspondant à une classe du diagramme de classes.

Après un état de prise de décision **dp**, un composant d'un processus de production peut effectué un **ab** s'il n y a pas les conditions suffisantes pour exécuter une activité, ou **go** dans le cas contraire. Après un **ab()**, ce composant passe à l'état **wait** et s'il y a l'apparition d'un événement attendu dans le système le composant reprend son travail avec un **start()** et passe à l'état **work**. Après un **go()**, le processus passe à l'état **work**. Une fois une activité est exécutée, le composant fait un **exit ()** et passe a l'état **out**. Le cycle de production se répète en effectuant **new().**

Les associations entre les classes sont représentées par un ensemble de places de synchronisation suivant la stratégie de supervision utilisée.

La vérification du processus de production est réalisée suivant un ensemble d'étapes :

- Génération d'un outil de modélisation avec les réseaux de Petri.
- Définition d'une grammaire de graphes permettant de faire une transformation du diagramme de classes vers les réseaux de Petri.
- Vérification automatique des propriétés comportementales du modèle réseau de Petri généré par la grammaire de graphes.
- Génération automatique d'un graphe des états désirables (GDS), présentant le scénario de bon fonctionnement du processus de production.

2.4.3.1 Génération d'un outil de modélisation avec les réseaux de Petri

Pour générer un outil de modélisation avec les réseaux de Petri, nous avons utilisé ATOM^3. Et comme nous avons mentionné que la modélisation avec cet outil nécessite une étape de méta-modélisation, la génération de cet outil de modélisation, doit passer par les étapes suivantes:

1. Chargement d'un méta-modèle pour les diagrammes de classes, qui se trouve dans les formalismes principaux d'ATOM^3.
2. Concevoir le méta-modèle du réseau de Petri selon la structure définie dans la Figure 2.17. Ce méta- modèle est composé de quatre méta-classes :
   - Une méta-classe pour les places du réseau de Petri, chaque place a deux attributs : **name** de type string et **Tokens** de type Integer.
   - Une méta-classe pour les transitions, chaque transition a un attribut : **name** de type string.
   - Deux associations, présentant les arcs, une pour les arcs entre les places et les transitions et l'autre pour les arcs entre les transitions et les places. Ces associations ont des cardinalités 0-N.

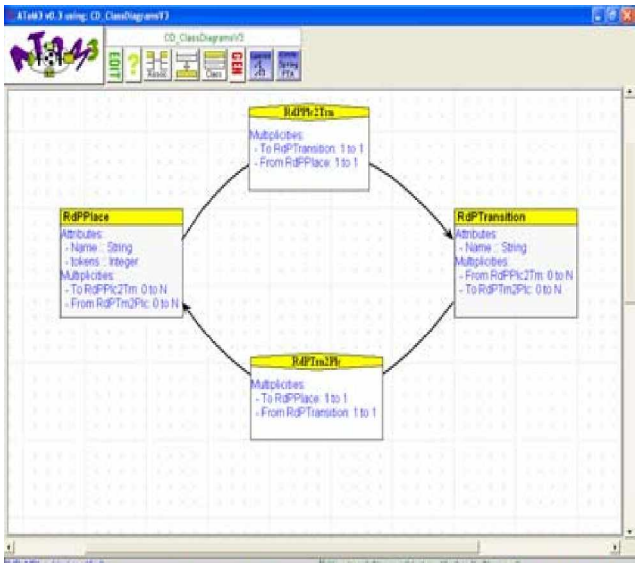

Figure 2.17 Méta-modèle pour les réseaux de Petri.

2. A partir de ce méta-modèle, ATOM^3 génère un outil de modélisation pour le formalisme réseau de Petri. Cet outil est illustré dans la Figure suivante :

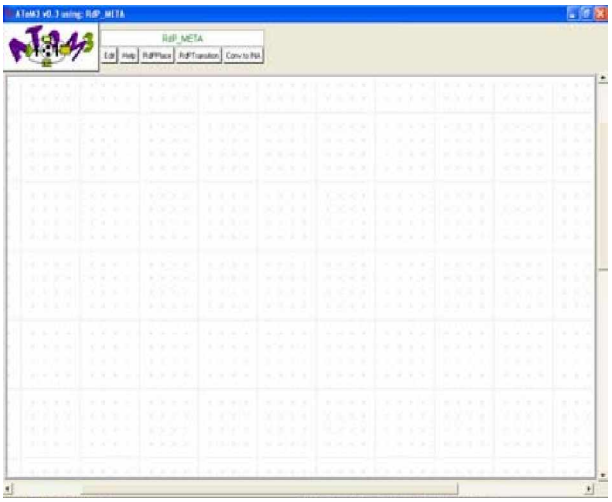

Figure 2.18 Outil de modélisation pour les RdPs généré par ATOM^3.

2.4.3.2 Définition d'une grammaire de graphes

Une grammaire de graphes est une grammaire constituée d'un ensemble de règles, permettant de transformer deux formalismes de même nature ou de nature différente. Chaque règle est composée de deux parties, la partie gauche (LHS) et la partie droite (RHS). Chaque partie peut être un sous graphe des formalismes considérés dans la transformation.

Dans notre travail, la partie gauche de chaque règle est un sous graphe du diagramme de classes pour le processus de production, et la partie droite peut être un sous graphe du réseau de Petri.

Cette grammaire est définie en utilisant l'outil ATOM^3. Pour la définir et comme notre but est de transformer un diagramme de classes vers les réseaux de Petri, il faut :

- Charger les deux méta-modèles définis précédemment qui sont le diagramme de classes et les réseaux de Petri.
- Définir les règles de la grammaire, et générer son fichier exécutable.

a. Règles de la grammaire de graphes

Notre grammaire de transformation du diagramme de classes pour un processus de production vers les réseaux de Petri est composée de trois parties:

1. **Action initiale**

La partie action initiale de la grammaire est une partie d'initialisation, où nous avons initialisé l'ensemble des variables globales utilisées le code Pyton dans la grammaire.

2. **Action finale**

La partie action finale de la grammaire est une partie**,** où la libération de l'espace mémoire occupé par l'ensemble des variables globales a eu lieu.

3. Ensemble de règles

Notre grammaire est composée de dix règles. Chaque règle est caractérisée par un nom et une priorité d'exécution. Nous citons ici quelques règles :

➢ **Règle 1 :*ClasstoRdpRule (priority 1)***

Cette règle est appliquée pour transformer chaque classe du diagramme de classes vers un RdP formé de quatre places et cinq transitions et les associer des noms. La nomination est faite selon le nom de la classe correspondante de la partie gauche de la règle. Les noms sont sous la forme suivante : **nom de la classe_nom de place /transition.**

Cette règle a la priorité égale à un, c à d c'est la première règle appliquée dans la grammaire.

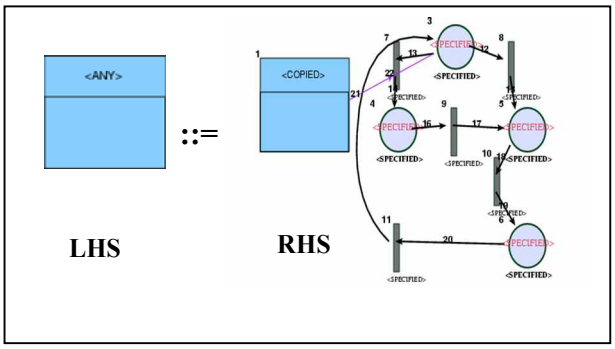

Figure 2.19 Partie gauche et droite de la règle une.

Pour l'application de cette règle, il faut vérifier que la classe de la partie gauche à transformer n'est pas traitée auparavant. Pour cela nous avons définis une condition pour cette règle :

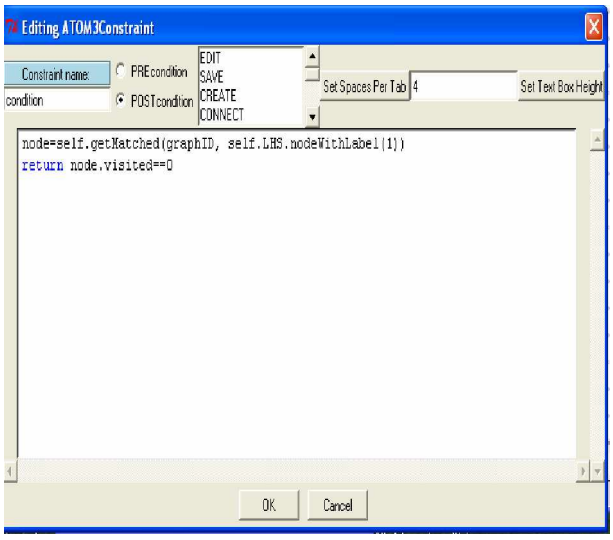

Figure 2.20 Condition d'application de la règle une.

Après l'application de la règle une action est exécutée permettant de marquer que l'objet classe de la partie gauche est traité.

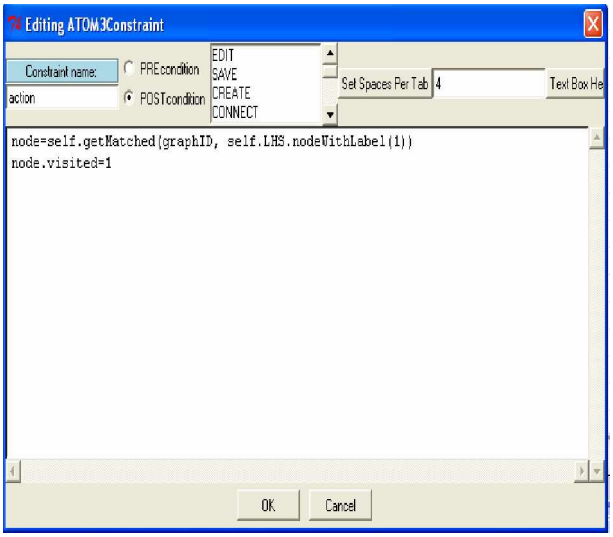

Figure 2.21 Action de la règle une.

➢ **Règle 2:** ***Identify Association Rule (priority 2)***

Cette règle est appliquée pour identifier s'il y a une association entre deux classes, et supprime cette association du diagramme de classes. Ainsi, elle sauvegarde les noms des deux classes membres de l'association dans des variables globales. Elle a la priorité égale à deux.

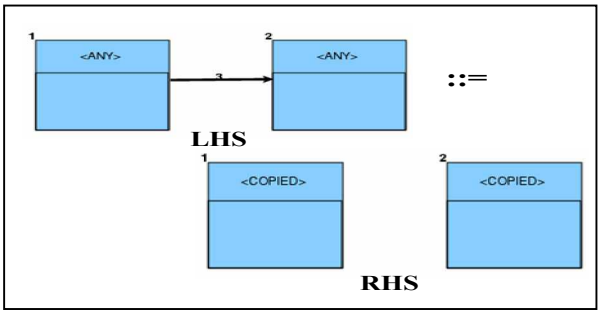

Figure 2.22 Parties gauche et droite de la deuxième règle.

La condition d'application de cette règle, est qu'il faut vérifier que les deux classes membres de l'association ne sont pas traitées auparavant. Cette condition est définie dans la contrainte de la règle, comme le montre la Figure 2.23.

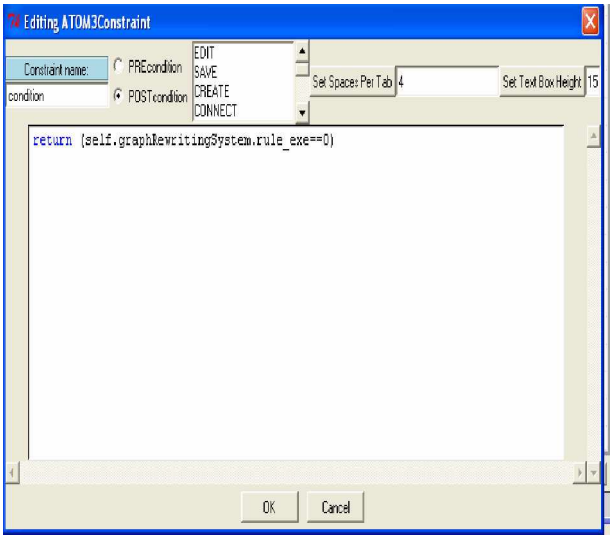

Figure 2.23 Condition d'application de la deuxième règle.

L'action correspondante à cette règle permet de :

- Sauvegarder les noms des deux classes membres de l'association dans des variables globales.
- Marquer que ces deux classes sont traitées.

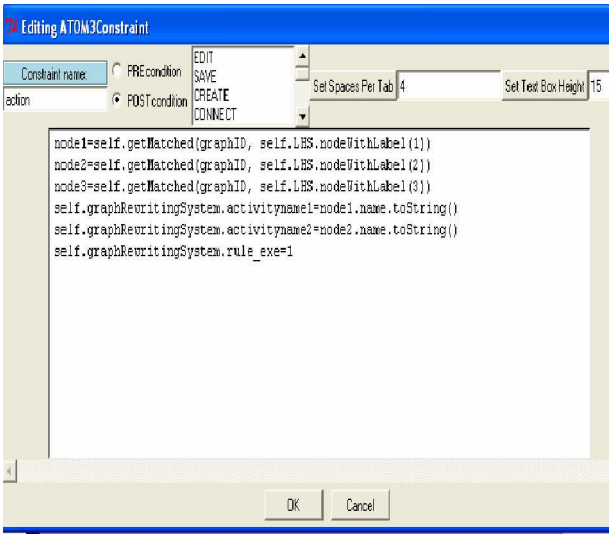

Figure 2.24 Action de la deuxième règle.

➢ **Règle 4:** ***AddPlacesSp1Rule (priority 4)***

Cette règle est une règle de synchronisation entre les composants d'un processus de production. Elle est appliquée pour chercher dans le RdP généré par les règles précédentes, des transitions par ses noms et d'ajouter une place nommée **SP1** comme une place d'entrée pour certaines transitions et une place de sortie pour d'autres. Elle a la priorité égale à quatre.

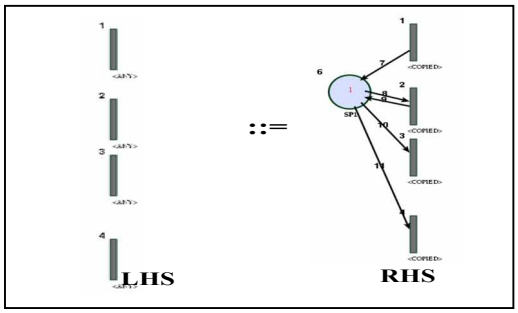

Figure 2.25 Parties gauche et droite de la quatrième règle.

Pour appliquer cette règle, il faut vérifier qu'elle n'est pas appliquée auparavant et que les noms des transitions cherchées correspondent bien à des noms spécifiques.

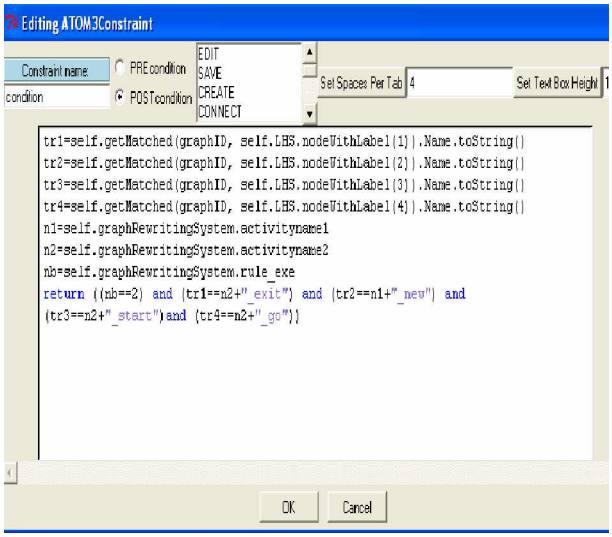

Figure 2.26 Condition d'application de la quatrième règle.

Une fois que la règle est appliquée, son action permet de marquer qu'elle est exécutée.

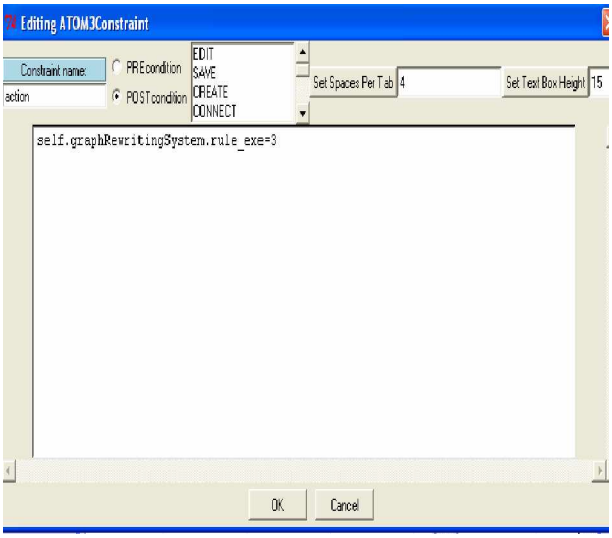

Figure 2.27 Action de la quatrième règle.

➢ **Règle 10:** ***ComponentDeleteRule (priority 10)***

Cette règle est appliquée pour chercher des classes non connectées c'est-à-dire sans associations, pour les supprimer. La partie droite de la règle est vide. Elle a la priorité égale à dix, c'est la dernière règle qui permet de supprimer toutes les classes restantes du diagramme de classes après la fin d'exécution des autres règles. Son action et condition sont vides.

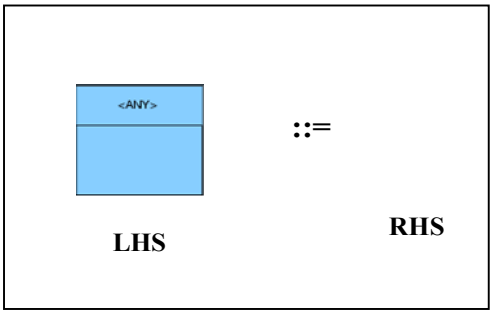

Figure 2.28 Parties gauche et droite de la dixième règle.

2.4.3.3 Vérification automatique des propriétés comportementales du processus de production

Cette étape consiste à vérifier certaines propriétés comportementales du modèle réseau de Petri généré par la grammaire de graphes. Ces propriétés sont relatives à l'aspect dynamique du processus de production et concernent son bon fonctionnement. Elles consistent en : la bornitude, la vivacité et l'absence de transitions bloquantes.

Pour obtenir une vérification automatique après la génération du modèle réseau de Petri, nous avons intégré l'outil INA dans notre travail de telle sorte que l'utilisateur aura la possibilité de faire cette vérification a partir de l'outil ATOM^3.

Cette possibilité est effectuée en appuyant sur un bouton **Convert To INA**, défini dans l'outil de modélisation avec les réseaux de Petri comme le montre la Figure 2.29. Un fichier texte d'extension ''**.pnt**''sera généré, il contient une description du modèle réseau de Petri généré. Ensuite ce fichier sera une entrée à l'outil INA, qui effectue l'analyse du modèle.

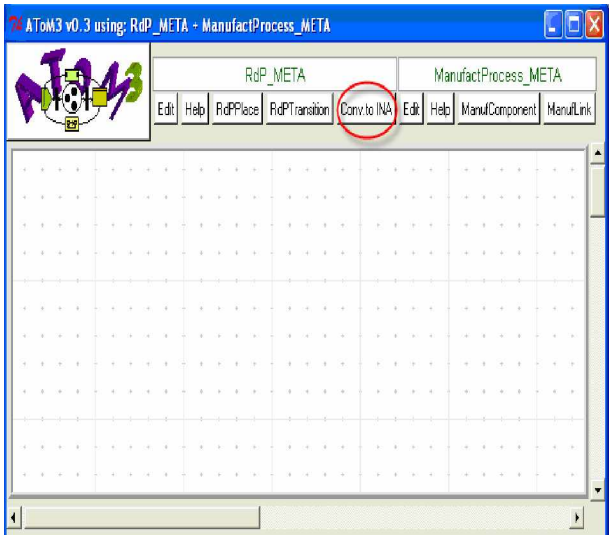

Figure 2.29 Analyse automatique du modèle RdP.

3.4.3.4 Génération automatique d'un graphe des états désirables

Le graphe des états désirables (GDS :Graph of Desirable States) est un graphe qui énumère tous les états désirables et leurs relations. Les états désirables sont des marquages et les relations entre ces marquages sont des transitions franchissables, depuis un marquage et amenant à un autre.

Le GDS utilise les informations du diagramme des cas d'utilisation dans le domaine de RdP, c'est-à-dire les phrases de ce diagramme sont traduites à un ensemble de règles en terme de places et de transitions. Le mot "désirable" reflète le fait que le graphe englobe tout ce que nous attendons du système et n'importe quel comportement indésirable est inhibé.

Le GDS est équivalent à un graphe de marquages, donc pour obtenir ce graphe de manière automatique, nous avons profité de l'outil INA qui calcule ce graphe a partir du fichier texte présentant le RdP généré. Le graphe résultat de cet outil est sous forme d'un fichier texte en terme de places et de transitions, comme le montre la Figure suivante :

```
Etat i
Liste des places : Pi ,Pj ,……..PN
Les jetons :       toki, tokj,……tokN
Lest transitions franchissables Ti=>Sj
                                Tk=>SN
                                  .
                                  .
Etat j
Liste des places : Pi ,Pj ,……..PN
Les jetons :       toki, tokj,……tokN
Lest transitions franchissables Ti=>Sj
   .                            Tk=>SN
   .                              .
                                  .
```

Figure 2.30 Structure du fichier du graphe de marquages.

2.5 Conclusion

La méthode de conception d'un superviseur de contrôle pour un processus de production distribué, que nous avons proposé permet de disposer d'un outil de modélisation et de vérification de ce processus.

Cette méthode est basée sur deux modèles, un pour la structure du processus de production, c'est le diagramme de classes et l'autre pour modéliser la dynamique de ce processus, c'est le formalismes réseaux de Petri .En empruntant une technique de transformation de graphes offerte par l'outil ATOM^3 sous forme de grammaire de graphes, nous avons la possibilité d'effectuer une transformation du diagramme de classes vers son équivalent réseau de Petri , ainsi d'utiliser des outils existants de vérification des propriétés adaptés pour le modèle réseau de Petri.

La méthode de conception proposée est validée par des exemples de processus de production déjà traités d'une façon manuelle. Pour voir son application, ainsi pour bien comprendre les étapes présentées dans ce chapitre, nous allons étudier deux exemples de processus de production , chacun avec deux variantes dans le chapitre suivant.

ETUDE DE CAS :Exemples d'application de la méthode de conception d'un superviseur de contrôle pour un processus de production distribué

Dans le chapitre précédent, une méthode de modélisation et de vérification des processus de production en vue de concevoir un superviseur de contrôle a été présentée. Pour montrer comment cette méthode est utilisée, nous proposons dans ce chapitre une application de cette méthode sur deux exemples de processus de production .Le premier est une chaîne d'emballage des produits et le deuxième est une chaîne d'embouteillage. La structure physique de chaque chaîne, ainsi que les caractéristiques des éléments physiques, sont présentées. Ensuite, leur modélisation et vérification sont réalisées à l'aide des outils nécessaires présentés dans le chapitre précédent.

Pour chaque exemple, nous essayons de donner deux variantes pour comparer les résultats obtenus.

3.1 Exemple1 : Chaîne d'emballage de produits

3.1.1 Présentation

La Figure suivante (Figure 3.1) illustre une ligne d'emballage de produits simple, composée de quatre composants : une ceinture (belt), un dévidoir de feuilles d'emballage (Film), un appareil de collage de feuilles (welder) et un coupoir (cutter). Un produit à emballer (ou un JOB) et la feuille d'emballage qui a une étiquette imprimée (ou TAG) sont identifiés par leurs supports c'est-à-dire la ceinture et le dévidoir de feuilles, respectivement. Le produit (JOB) et l'étiquette (TAG) sont disposés suivant la ceinture et le dévidoir Chacun de ces modules peut être contrôlé séparément. Par exemple, la ceinture est mise en marche par un moteur [Bordbar, 2000].

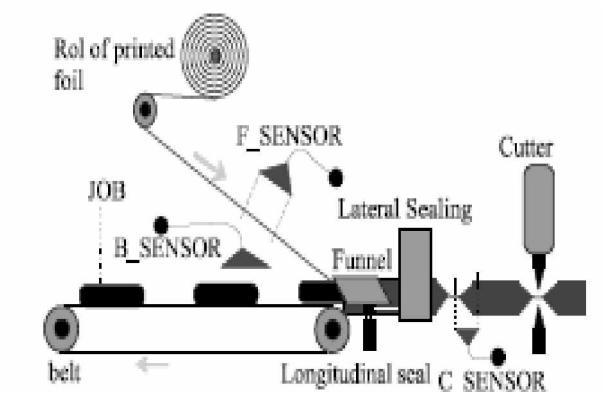

Figure 3.1 Exemple d'une chaîne d'emballage [Bordbar, 2000].

3.1.2 Modélisation avec UML

3.1.2.1 Diagramme de cas d'utilisation

Le fonctionnement de la chaîne d'emballage de la Figure 3.1 est décrit comme suit :

Quand un produit arrive (new JOB) sur son support (belt), l'état de l'étiquette (TAG) est évalué. Si l'étiquette est sur son support (film), l'opération d'emballage peut avoir lieu. Cependant, si l'étiquette est hors la zone d'emballage, le JOB s'arrêtera et prend l'état d'attente, en attendant l'étiquette de parvenir à son support. Quand l'étiquette arrive (new TAG), le JOB se repris pour être dans la zone d'emballage. Quand le produit et l'étiquette sont tout les deux dans la zone d'emballage, la feuille d'emballage prend la forme d'un tube, via un entonnoir et un appareil de soudure colle les deux bords de la feuille. Un fer à souder latéral soude le tube entre les différents paquets et le produit emballé sort de la zone d'emballage. Les produits scellés sont ensuite séparés par le coupoir pour produire des produits individuellement emballés [Bordbar, 2000]. Et tout le cycle recommence.

A partir de ce texte qui présente le scénario de fonctionnement de cette ligne d'emballage, nous pouvons distinguer les acteurs de ce diagramme qui sont : la ceinture (belt), le dévidoir de film d'emballage (Film), l'appareil de soudure (welder) et le coupoir (cutter).

La synchronisation entre ces acteurs s'effectue deux à deux, entre la ceinture et le dévidoir de film d'emballage, entre le dévidoir de film d'emballage et l'appareil de soudure et entre le dévidoir de film d'emballage et le coupoir.

3.1.2.1 Diagramme de classes

Dés que le diagramme de cas d'utilisation pour la ligne d'emballage a quatre acteurs, alors nous avons quatre classes dans le diagramme de classes comme le montre la Figure 3.2.

Les relations entre ces classes sont des associations entre chaque deux classes, présentant

la synchronisation entre chaque deux acteurs du diagramme de cas d'utilisation.

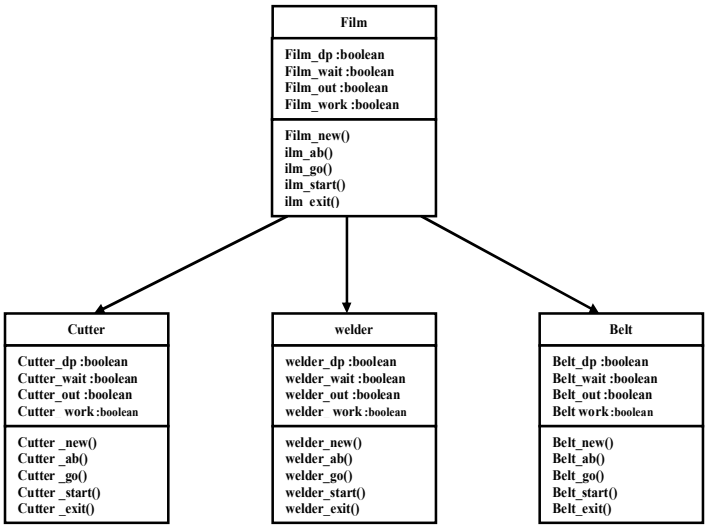

Figure 3.2 Diagramme de classes pour la chaîne d'emballage.

3.1.2.3 Modélisation automatique de l'exemple

Suivant le méta-modèle présenté dans la Figure 2.11 du chapitre précédent et avec l'outil de modélisation généré par ATOM^3, le diagramme de classes de la Figure 3.2 est modélisé dans cet outil comme suit :

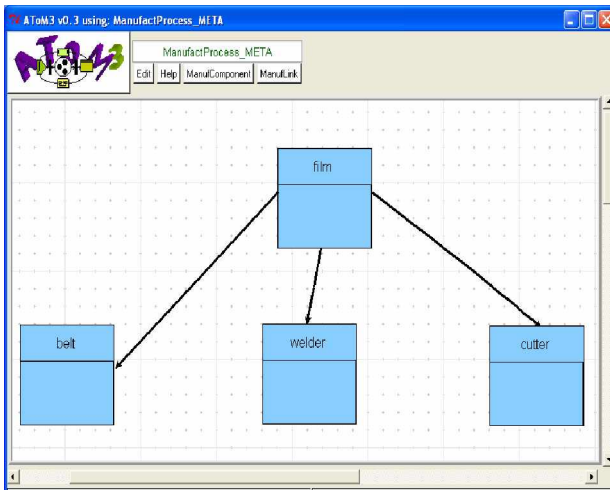

Figure 3.3 Diagramme de classes pour la chaîne d'emballage dans ATOM^3.

Pour chaque classe, nous pouvons saisir les valeurs de ses attributs, chaque attribut coché indique qu'elle est égale à vrai, par conséquent dans le réseau de Petri équivalant, la place correspondante à cet attribut est marquée par un jeton.

Les méthodes des classes sont sans code puisque ce qui nous intéresse c'est la synchronisation entre leurs exécutions et pas comment elles sont implémentées.

Pour cet exemple, nous allons étudier deux variantes. Dans la première, nous laissons l'exemple tel qu il est, et dans le deuxième nous modifions l'état initial de la ligne d'emballage.

a. Variante n°1

Dans cette variante nous cochons l'attribut **out** pour toutes les classes pour indiquer que dans l'état initial, toutes les machines sont hors la zone d'emballage.

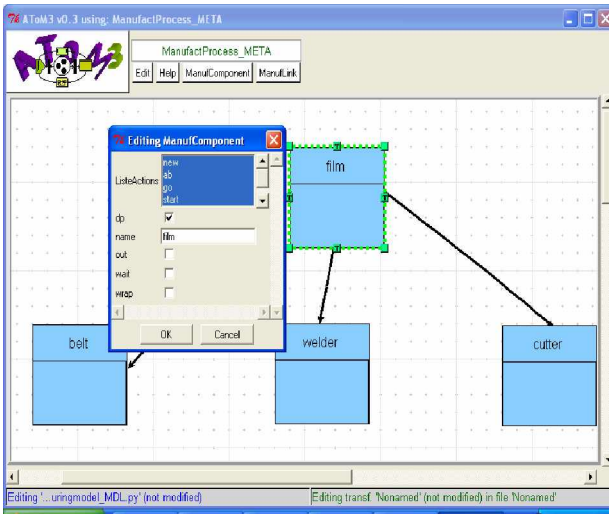

Figure 3.4 Saisie des valeurs des attributs des classes pour la variante 1.

b. Variante n°2

Dans cette variante nous cochons un attribut différent pour chaque classe, par exemple l'attribut **dp** pour la classe Film et **out** pour la classe Belt, pour indiquer que dans l'état initial, chaque machine est dans un état différent de l'autre.

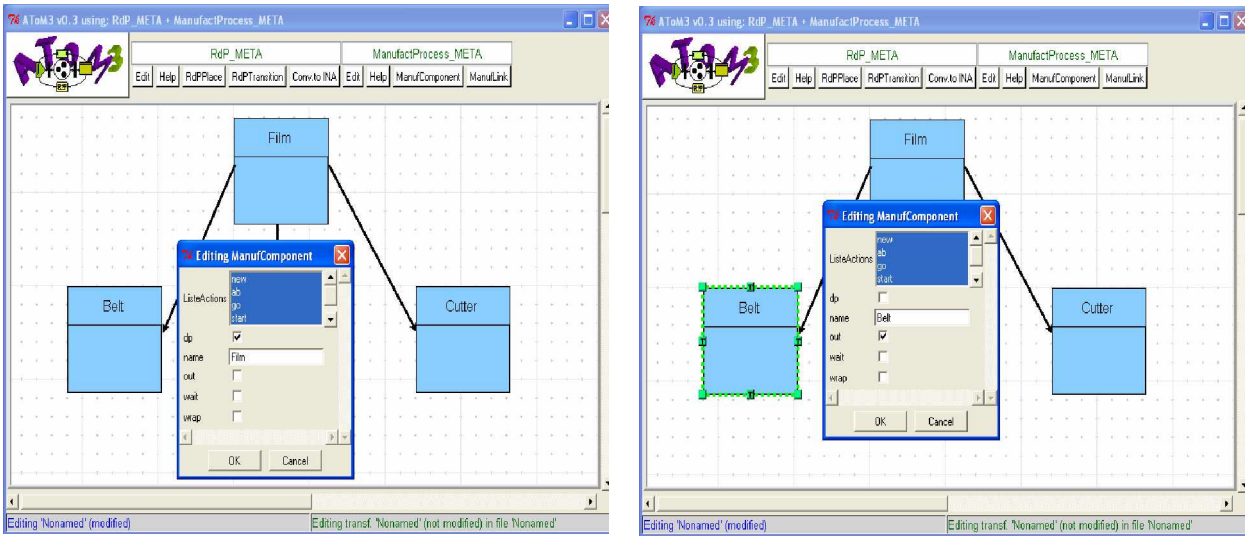

Figure 3.5 Saisie des valeurs des attributs des classes pour la variante 2.

3.1.3 Transformation du diagramme de classes vers les réseaux de Petri

Pour obtenir Un modèle réseau de Petri équivalent au diagramme de classes de la Figure 3.3, nous avons appliqué notre grammaire qui est composée de dix règles .L'application de la grammaire s'effectue en trois étapes :

- Chargement des deux méta-modèles pour les réseaux de Petri et le diagramme de classes.

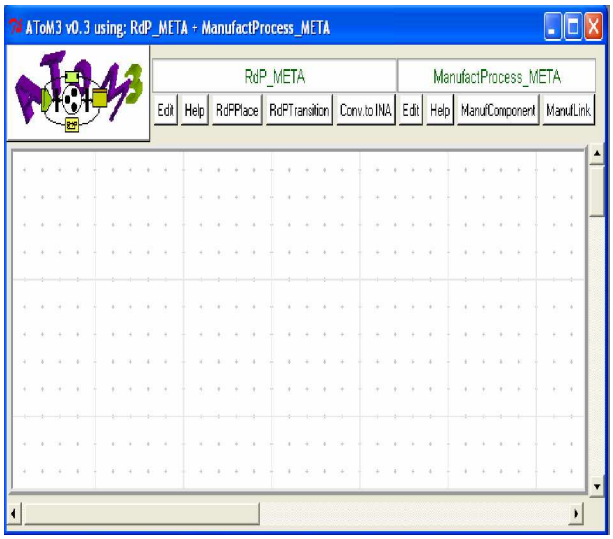

Figure 3.6 Chargement des méta-modèles.

- Ouverture du modèle de diagramme de classes de la figure 3.3 qui a été sauvegardé.

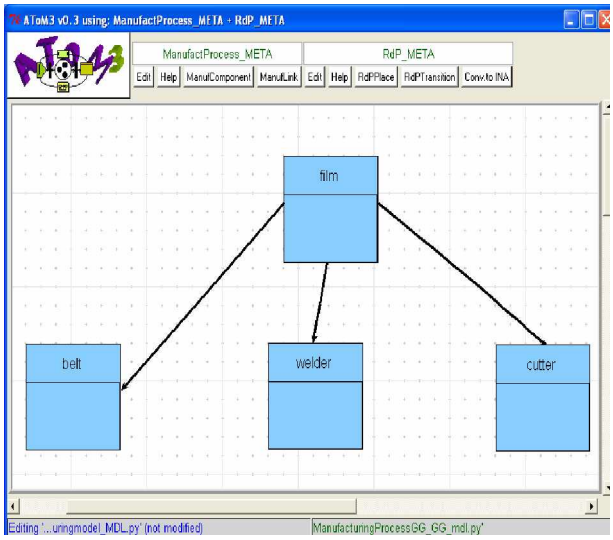

Figure 3.7 Ouverture du diagramme de classes.

- Chargement de la grammaire de transformation.

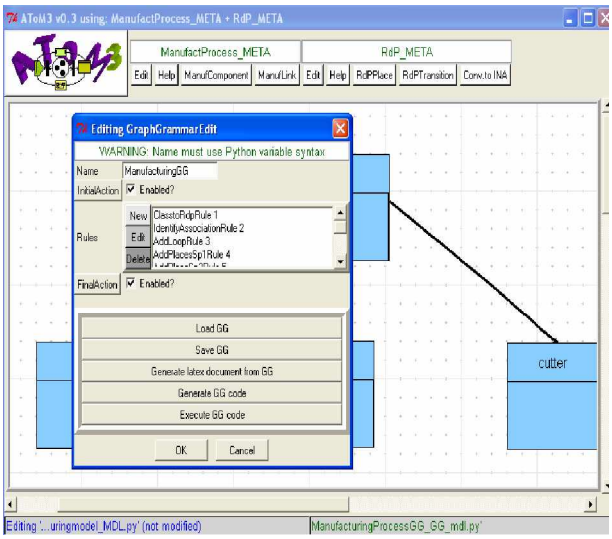

Figure 3.8 Chargement de la grammaire de transformation.

- Exécution de la grammaire en lançant son fichier exécutable.

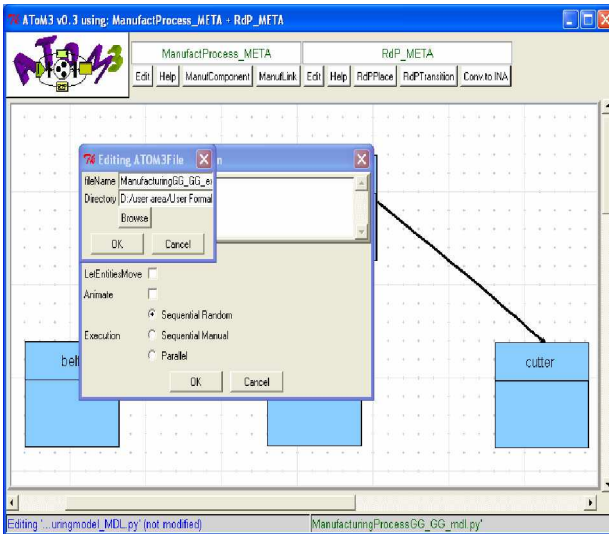

Figure 3.9 Ouverture du fichier exécutable de la grammaire.

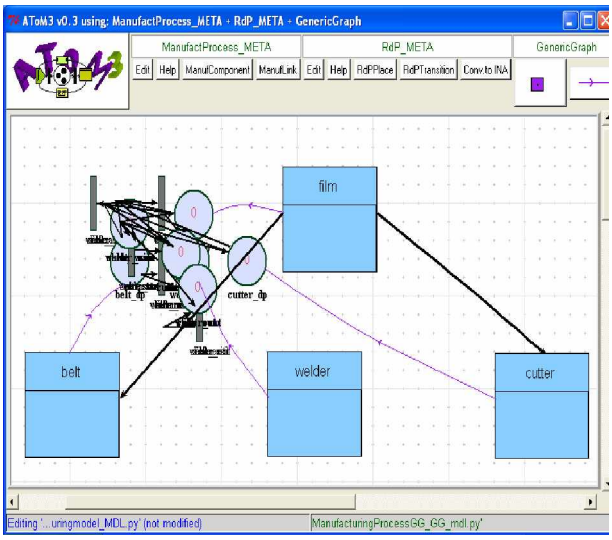

Figure 3.10 Lancement de la grammaire.

Pour une raison de la taille du modèle réseau de Petri généré, nous présentons ici le réseau de Petri qui concerne la synchronisation entre le Belt et le Film pour l'exemple du processus d'emballage. Les résultats de la transformation sont illustrés dans les Figures suivantes :

a. Variante n°1

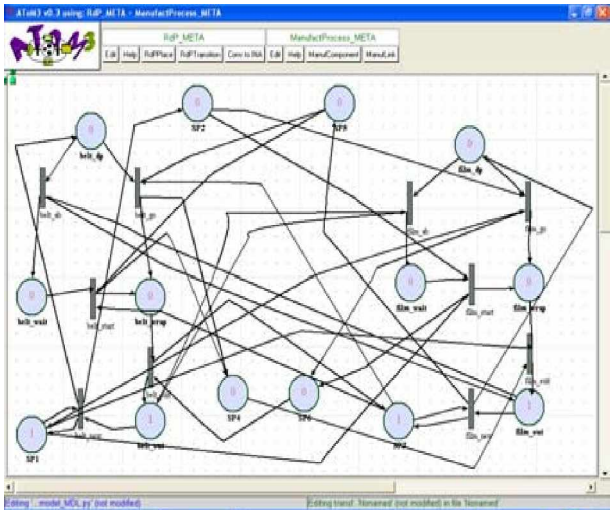

Figure 3.11 Réseau de Petri généré pour la variante n°2.

b. Variante n°2

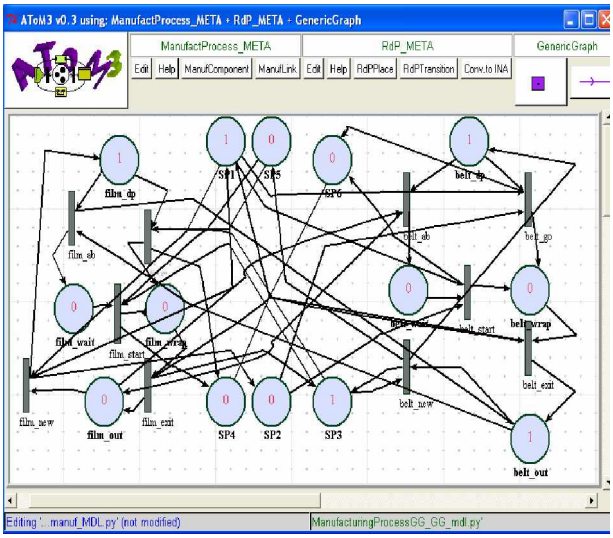

Figure 3.12 Réseau de Petri généré pour la variante n°2.

3.1.4 Vérification des propriétés du réseau de Petri généré

Comme nous avons mentionné dans le chapitre précèdent, que la vérification des propriétés comportementales du processus de production est réalisée en utilisant l'outil de vérification INA, adapté pour les réseaux de Petri.

Nous avons intégré l'outil INA dans ATOM^3 de telle sorte que nous pouvons obtenir un fichier texte qui est une description du modèle réseau de Petri généré. Les étapes permettant la

réalisation de cette procédure sont :

- Génération du fichier texte d'extension **.pnt** décrivant le réseau de Petri généré a partir de ATOM[3], en cliquant sur le bouton **Conv to INA**, comme le montre la Figure suivante :

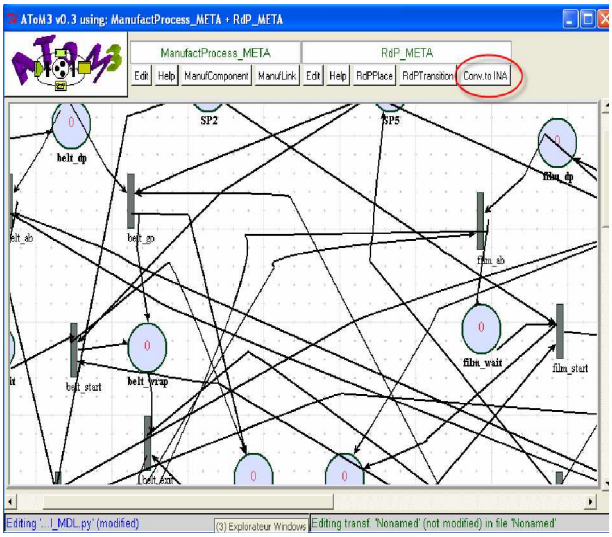

Figure 3.13 Obtention du fichier d'entrée pour l'outil INA.

Après, le fichier texte correspondant à ce modèle est généré, sous la forme présentée dans le chapitre précédent. Les Figures suivantes présentent le contenu de ce fichier :

a. Variante n°1

```
P  M  PRE,POST  NETZ 1:
0   0   0,   1 2
1   0   2,   3
2   0   1 3,  4
3   1   4 5,  0 5
4   0   6,   7 5
5   0   5,   8
6   0   7 8,  9
7   1   9 2,  6 2
8   1   9 0,  0 8 7
9   1   4 6,  6 3 1
10  0   8 7,  4
11  0   1 3,  9
12  0   0,   7 8
13  0   6,   1 3
@
place nr. Name     capacity  time
     0:  belt_dp      00        0
     1:  belt_wait    00        0
     2:  belt_wrap    00        0
     3:  belt_out     00        0
     4:  film_dp      00        0
     5:  film_wait    00        0
     6:  film_wrap    00        0
     7:  film_out     00        0
     8:  SP1          00        0
     9:  SP3          00        0
    10:  SP6          00        0
    11:  SP4          00        0
    12:  SP2          00        0
    13:  SP5          00        0
@
trans nr.  name          priority  time
     0:  belt_new          0        0
     1:  belt_go           0        0
     2:  belt_ab           0        0
     3:  belt_start        0        0
     4:  belt_exit         0        0
     5:  film_ab           0        0
     6:  film_new          0        0
     7:  film_go           0        0
     8:  film_start        0        0
     9:  film_exit         0        0
@
```

Figure 3.14 Fichier d'entrée pour l'outil INA pour la variante n°1.

b. Variante n°2

```
 P  M  PRE ,POST  NETZ 1:
 0  1   0,   12
 1  0   2,   3
 2  0   13,  4
 3  0   45,  05
 4  1   6,   75
 5  0   5,   8
 6  0   78,  9
 7  1   92,  62
 8  1   90,  087
 9  1   46,  631
10  0   87,  4
11  0   13,  9
12  0   0,   78
13  0   6,   13
@
place nr.    Name        capacity time
     0:      film_dp       00     0
     1:      film_wait     00     0
     2:      film_wrap     00     0
     3:      film_out      00     0
     4:      belt_dp       00     0
     5:      belt_wait     00     0
     6:      belt_wrap     00     0
     7:      belt_out      00     0
     8:      SP1           00     0
     9:      SP3           00     0
    10:      SP6           00     0
    11:      SP4           00     0
    12:      SP2           00     0
    13:      SP5           00     0
@
trans nr.   name        priority  time
     0:     film_new       0     0
     1:     film_go        0     0
     2:     film_ab        0     0
     3:     film_start     0     0
     4:     film_exit      0     0
     5:     belt_ab        0     0
     6:     belt_new       0     0
     7:     belt_go        0     0
     8:     belt_start     0     0
     9:     belt_exit      0     0
@
```

Figure 3.15 Fichier d'entrée pour l'outil INA pour la variante n°2.

- Lancement de l'outil INA et l'opération de vérification, en tapant le caractère **A** et en spécifiant le nom du fichier d'entrée :

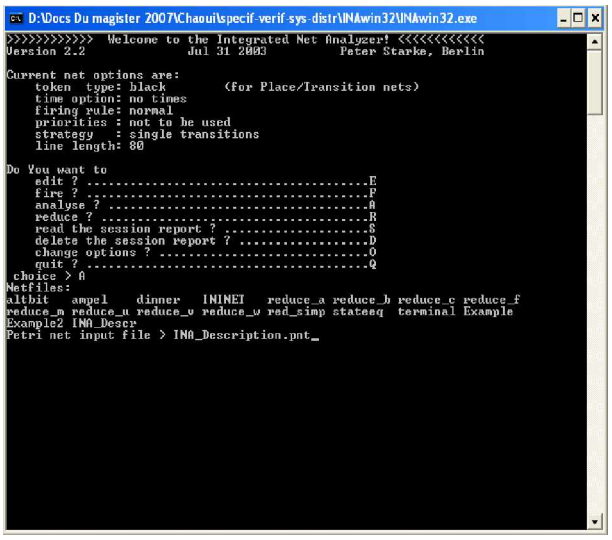

Figure 3.16 Lancement de l'outil INA.

Un menu d'analyse est affiché ensuite, nous devons spécifier le type d'analyse que nous voulons faire et comme, nous nous sommes intéressés aux propriétés comportementales du modèle, nous avons choisis les caractères **B**,**R** ensuite **L**.

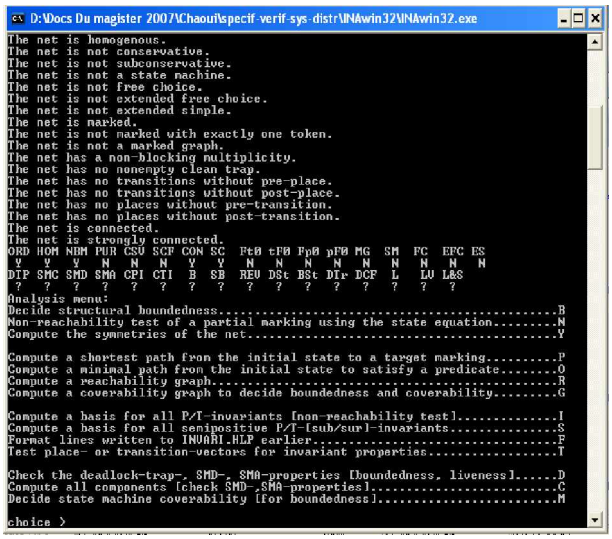

Figure 3.17 Choix du type d'analyse.

a. Variante n°1

Les résultats obtenus pour la variante 1 sont illustrés dans les Figures suivantes

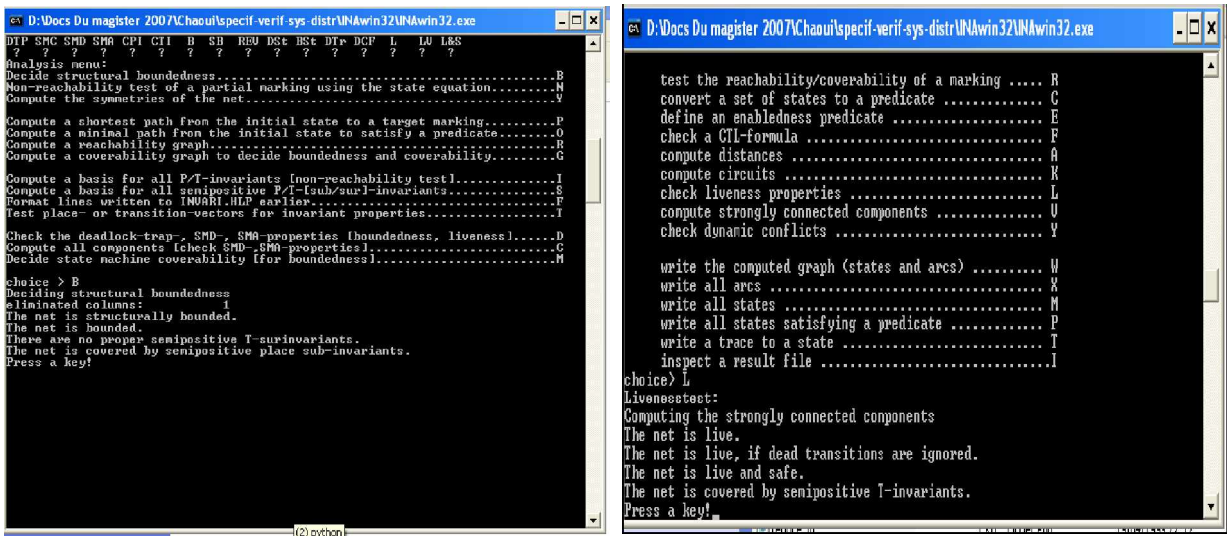

Figure 3.18 Analyse comportementale du modèle pour la variante n°1.

D'après le résultat d'analyse nous observons que le modèle est vivant et borné et n'a aucune transition bloquante.

b. Variante n°2

Les résultats obtenus pour la variante 2 sont les suivants :

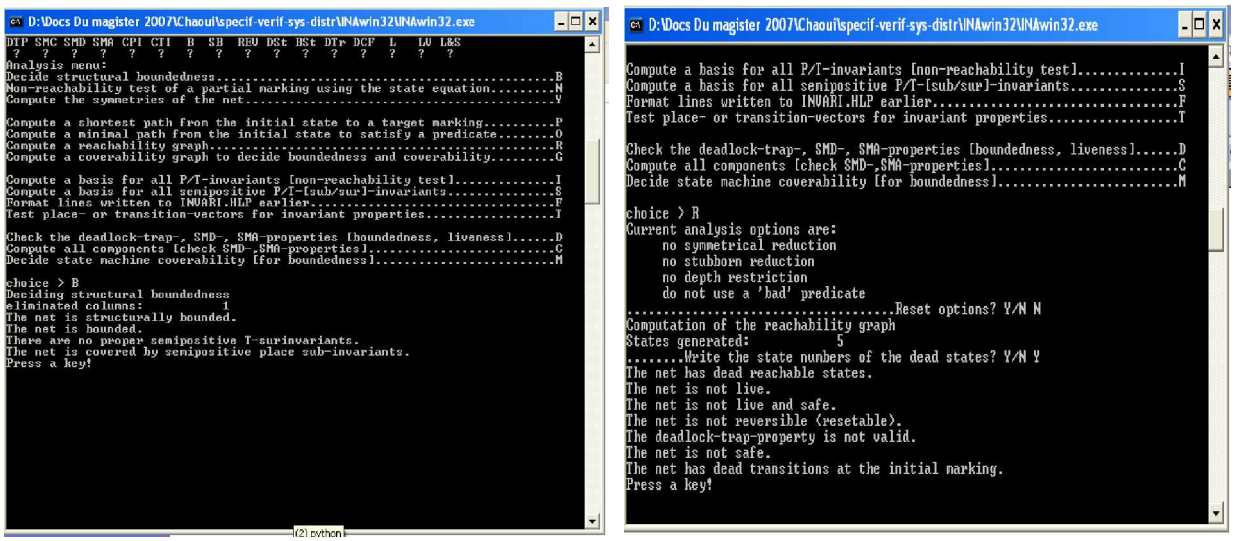

Figure 3.19 Analyse comportementale du modèle pour la variante n°2.

D'après le résultat d'analyse nous observons que le modèle est borné mais non vivant et a des transitions bloquantes.

- Génération du graphe des états désirables. Comme nous avons mentionné dans le chapitre précédent que ce graphe est équivalent au graphe de marquages, ce graphe permet de produire un enchaînement des activités du processus de production en évitant les cas du fonctionnement anormal du processus.

L'outil INA nous a permis de générer ce graphe a partir du fichier texte d'entrés. Le résultat est un autre fichier texte d'extension ‘‘**.gra’’.** La procédure est réalisée en tapant dans INA le caractère **R** ensuite **W.** Après, le fichier texte correspondant à ce graphe est généré.

a. Variante n°1

Le GDS pour la variante 1 est calculé par l'outil INA car d'après l'analyse nous avons trouvé que le modèle réseau de Petri de cette variante est borné et vivant. Les résultats sont illustrés dans les Figures 3.20 et 3.21

```
State nr.   1
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  0  0  0  1  0  0  0  1  1  1  0  0  0  0
==t0=> s2
==t6=> s12
State nr.   2
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  1  0  0  0  0  0  0  1  1  1  0  0  1  0
==t2=> s3
==t6=> s10
State nr.   3
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  0  1  0  0  0  0  0  1  1  1  0  0  1  0
==t6=> s4
State nr.   4
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  0  1  0  0  1  0  0  0  1  1  0  0  1  1
==t3=> s5
==t7=> s9
State nr.   5
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  0  0  1  0  1  0  0  0  1  0  0  1  1  0
==t7=> s6
State nr.   6
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  0  0  1  0  0  0  1  0  0  0  1  1  0  0
==t4=> s7
==t9=> s8
State nr.   7
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  0  0  0  1  0  0  1  0  0  1  0  1  0  0
==t9=> s1
State nr.   8
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  0  0  1  0  0  0  0  1  1  0  1  0  0  0
==t4=> s1
State nr.   9
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  0  1  0  0  0  0  1  0  0  1  1  0  0  1
==t3=> s6
State nr.   10
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  1  0  0  0  1  0  0  0  1  1  0  0  1  1
==t1=> s5
==t7=> s11
State nr.   11
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  1  0  0  0  0  0  1  0  0  1  1  0  0  1
==t1=> s6
State nr.   12
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  0  0  0  1  1  0  0  0  1  1  0  0  0  1
==t0=> s10
==t5=> s13
State nr.   13
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  0  0  0  1  0  1  0  0  1  1  0  0  0  1
==t0=> s14
State nr.   14
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  1  0  0  0  0  1  0  0  1  1  0  0  1  1
==t1=> s15
==t8=> s11
State nr.   15
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  0  0  1  0  0  1  0  0  1  0  0  1  1  0
==t8=> s6
```

Figure 3.20 Fichier du graphe de marquage généré par l'outil INA pour la variante n°1.

Pour mieux comprendre le GDS généré, nous avons illustré le contenu du fichier par un graphe de marquages, comme le montre la Figure suivante :

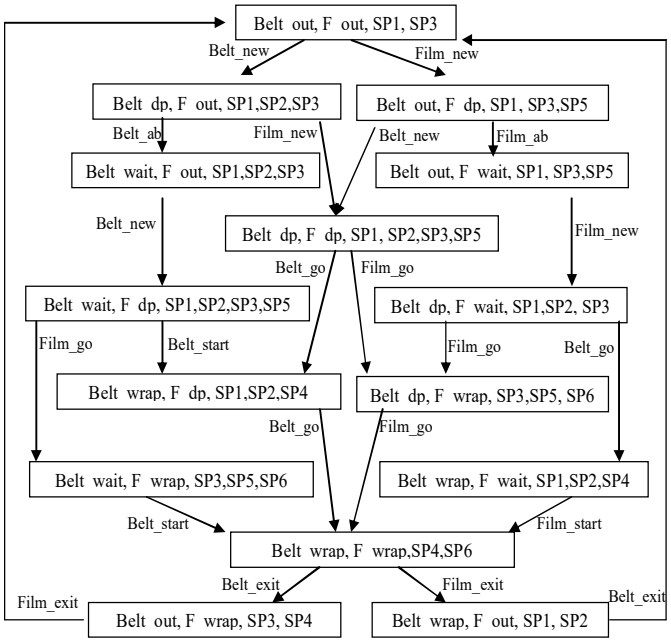

Figure 3.21 GDS généré par l'outil INA pour la variante n°1.

b. Variante n°2

Le GDS pour la variante 2 est non calculé par l'outil INA car le réseau de Petri généré pour cette variante est borné mais n'est pas vivant. Le résultat obtenu est illustré dans la Figure suivante

```
State nr.   1
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  1  0  0  0  1  0  0  1  1  1  0  0  0  0
==t2=> s2
==t6=> s5
State nr.   2
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  0  1  0  0  1  0  0  1  1  1  0  0  0  0
==t6=> s3
State nr.   3
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  0  1  0  0  2  0  0  0  1  1  0  0  0  1
==t3=> s4
State nr.   4
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  0  0  1  0  2  0  0  0  1  0  0  1  0  0
dead state
State nr.   5
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13
toks:  1  0  0  0  2  0  0  0  1  1  0  0  0  1
==t1=> s4
```

Figure 3.22 Fichier du graphe de marquage généré par l'outil INA pour la variante n°2.

3.2 Exemple2 : Chaîne d'embouteillage

3.2.1 Présentation

Notre deuxième exemple concerne une chaîne d'embouteillage d'eau. Une chaîne d'embouteillage est constituée de machines en série (Figure 3.23). Ces machines sont : un dépalettiseur, un convoyeur, une rinceuse, et un remplisseur.

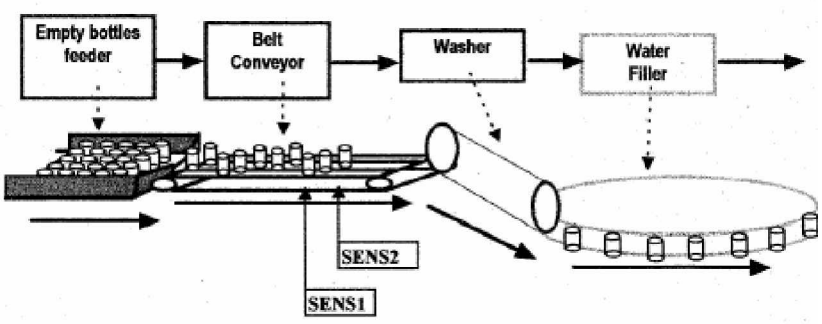

Figure 3.23 Exemple d'une ligne d'embouteillage [Adry, 1995].

- **Le dépalettiseur (Feeder) : s**a fonction est de déverser des couches de 360 bouteilles vides sur le convoyeur à accumulation situé en aval. Il a un fonctionnement asynchrone (il dépose une couche de 360 bouteilles uniquement s'il y a la place suffisante sur le convoyeur en aval).
- **Le convoyeur (conveyor):** son rôle est de transporter les bouteilles vides.
- **La rinceuse (washer) :** Pour assurer une parfaite hygiène avant le remplissage, les bouteilles doivent être nettoyées. C'est la fonction de la rinceuse, constituée d'un tunnel incliné et fermé à la poussière extérieure. Sa capacité est de 80 boîtes.
- **Le remplisseur (Filler):** La fonction du remplisseur est à la fois le remplissage d'eau et le sertissage (pose des couvercles). Sa cadence maximale est de 48000 boîtes par heure [Adry, 1995].

3.2.2 Modélisation avec UML

3.2.2.1 Diagramme de cas d'utilisation

Quand le dépalettiseur commence à fournir les bouteilles vides, si le convoyeur a des places vides, les bouteilles sont évacuées au convoyeur et le transport des bouteilles peut avoir lieu aux vitesses de 1 ou 2 mètres/seconde. La quantité de bouteilles sur le convoyeur reste entre les limites détectées par des capteurs d'accumulation. Cependant, si le convoyeur est inactif, le dépalettiseur s'arrêtera, attendant le convoyeur soit actif. Quand le convoyeur reprend son fonctionnement, le dépalettiseur est remis en marche. Quand les deux machines sont dans l'état

de fonctionnement, les bouteilles peuvent être accumulées vers la sortie du convoyeur. Si la sortie est fermée, la rinceuse commence à laver. Si le remplisseur est prêt alors il met une quantité constante de l'eau dans chaque bouteille. Pour faire cela, les bouteilles doivent être toujours remplies à la même vitesse. Ainsi, quand le remplisseur est de s'arrêter, les bouteilles qui sont intérieur doivent être évacuées et ne peuvent pas s'accumuler. Ceci est fait par la fermeture de l'entrée du remplisseur. Les bouteilles remplies sortent de la zone d'embouteillage. Ensuite elles seront stockées. Et tout le cycle d'embouteillage recommence.

Depuis le texte, les acteurs de ce diagramme sont : le dépalettiseur (Feeder), le convoyeur (conveyor), la rinceur (washer) et le remplisseur (filler). La synchronisation entre ces acteurs s'effectue deux à deux.

3.2.2.2 Diagramme de classes

Le diagramme de classes pour cet exemple est composé de quatre classes comme le montre la Figure 3.24. Les relations entre ces classes sont des associations entre chaque deux classes, présentant la synchronisation entre chaque deux acteurs du diagramme de cas d'utilisation.

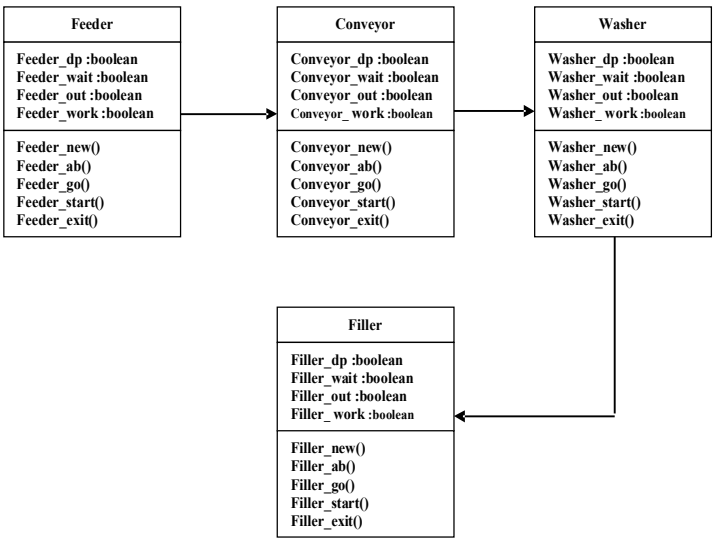

Figure 3.24 Diagramme de classes pour la chaîne d'embouteillage.

3.2.2.3 Modélisation automatique de l'exemple

a. Variante n° 1

Dans cette variante de l'exemple, le diagramme de classes est modélisé de telle sorte que le washer prend en charge aussi le rôle du Filler. Donc il y aura une synchronisation entre le Feeder et le conveyor et entre ce dernier et le washer.

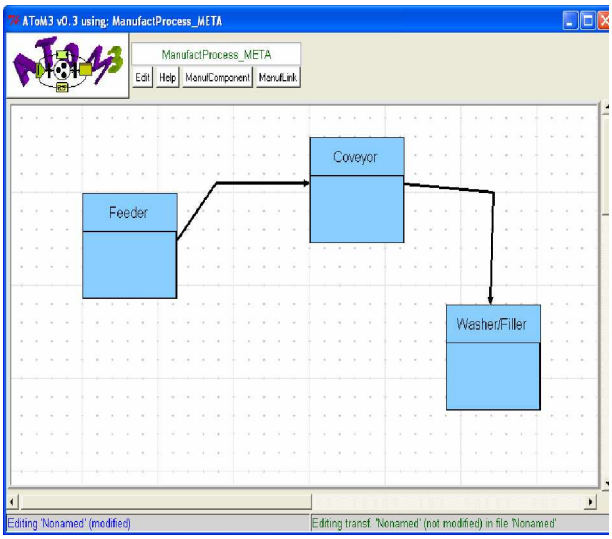

Figure 3.25 Diagramme de classes pour la variante n°1.

b. Variante n° 2

Dans La variante 2, le diagramme de classes modélise la chaîne d'embouteillage telle qu'elle est en mettant quatre classes comme le montre la Figure 3.26

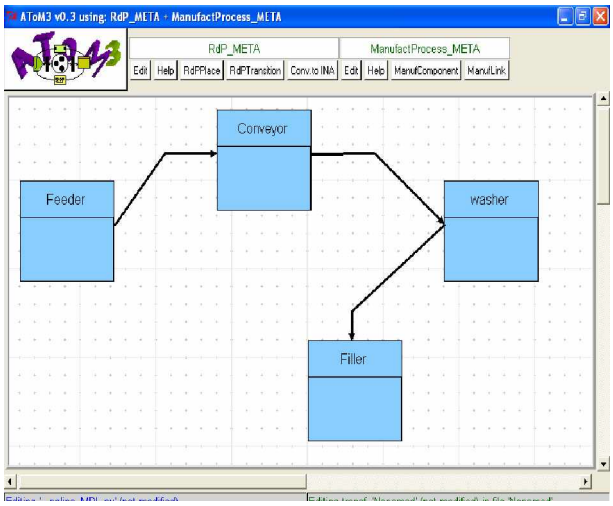

Figure 3.26 Diagramme de classes pour la variante n°2.

3.2.3 Transformation du diagramme de classes vers les réseaux de Petri

a. Variante n° 1

Le réseau de Petri équivalent au diagramme de classes de la Figure 3.25, après l'application de notre grammaire de transformation est illustré dans la Figure 3.27. Ainsi, nous présentons le réseau de Petri qui concerne la synchronisation entre le conveyor et le Feeder et entre le conveyor et le washer.

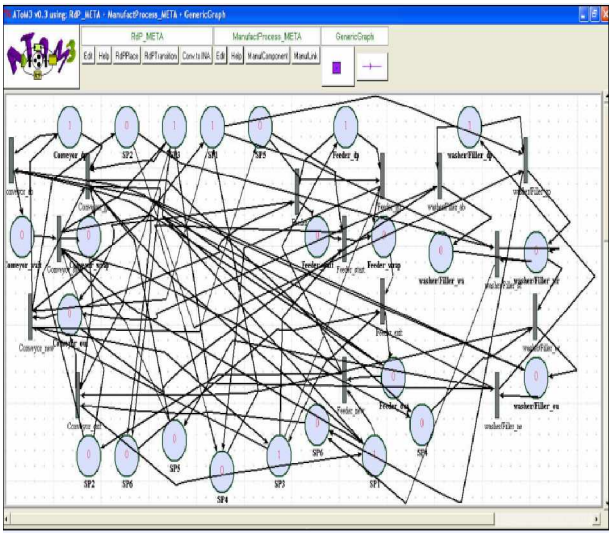

Figure 3.27 Réseau de Petri généré pour la variante n°1.

b. Variante n° 2

Pour la variante 2 le réseau de Petri équivalent au diagramme de classe de la Figure 3.26 est présenté dans la Figure suivante, et toujours à cause de la taille de ce réseau de Petri généré, nous présentons une partie qui concerne la synchronisation entre le conveyor et le feeder :

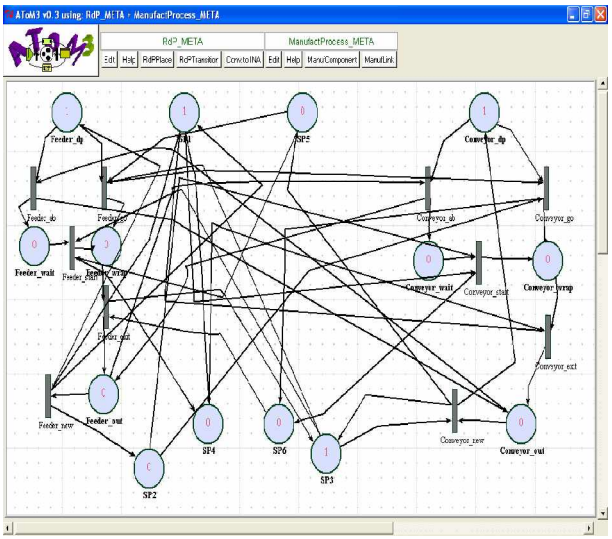

Figure 3.28 Réseau de Petri généré pour la variante n°2.

3.2.4 Vérification des propriétés du réseau de Petri généré

a. Variante n°1

Le fichier d'entrée pour l'outil INA est généré après l'application de la grammaire, comme le montre la Figure suivante :

```
 P  M  PRE ,     POST NETZ 1:
 0  1  0 ,       1 2
 1  0  2 ,       3
 2  0  1 3 ,     4
 3  0  4 5 ,     0 5
 4  1  6 ,       7 5
 5  0  5 ,       8
 6  0  7 8 ,     9
 7  0  9 2 10 ,  6 2 10
 8  1  11 ,      12 10
 9  0  10 ,      13
10  0  12 13 ,   14
11  0  14 5 ,    11 5
12  1  9 0 ,     0 8 7
13  1  4 6 ,     6 3 1
14  0  8 7 ,     4
15  0  1 3 ,     9
16  0  0 ,       7 8
17  0  6 ,       1 3
18  1  14 6 ,    6 13 12
19  1  9 11 ,    11 8 7
20  0  13 12 ,   9
21  0  7 8 ,     14
22  0  6 ,       12 13
23  0  11 ,      7 8
@
place nr.        name          capacity  time
     0:        Feeder_dp          00    0
     1:        Feeder_wait        00    0
     2:        Feeder_wrap        00    0
     3:        Feeder_out         00    0
     4:        Conveyor_dp        00    0
     5:        Conveyor_wait      00    0
     6:        Conveyor_wrap      00    0
     7:        Conveyor_out       00    0
     8:        washer/Filler_dp   00    0
     9:        washer/Filler_wa   00    0
    10:        washer/Filler_wr   00    0
    11:        washer/Filler_ou   00    0
    12:        SP1                00    0
    13:        SP3                00    0
    14:        SP6                00    0
    15:        SP4                00    0
    16:        SP2                00    0
    17:        SP5                00    0
    18:        SP1                00    0
    19:        SP3                00    0
    20:        SP6                00    0
    21:        SP4                00    0
    22:        SP2                00    0
    23:        SP5                00    0
 @
trans nr.        name          priority time
     0:        Feeder_new          0     0
     1:        Feeder_go           0     0
     2:        Feeder_ab           0     0
     3:        Feeder_start        0     0
     4:        Feeder_exit         0     0
     5:        Conveyor_ab         0     0
     6:        Conveyor_new        0     0
     7:      Conveyor_go           0     0
     8:        Conveyor_start      0     0
     9:        Conveyor_exit       0     0
    10:        washer/Filler_ab    0     0
    11:        washer/Filler_ne    0     0
    12:        washer/Filler_go    0     0
    13:        washer/Filler_st    0     0
    14:        washer/Filler_ex    0     0
@
```

Figure 3.29 Fichier d'entrée de l'outil INA généré pour la variante n°1.

L'analyse avec l'outil a donné les résultats suivants :

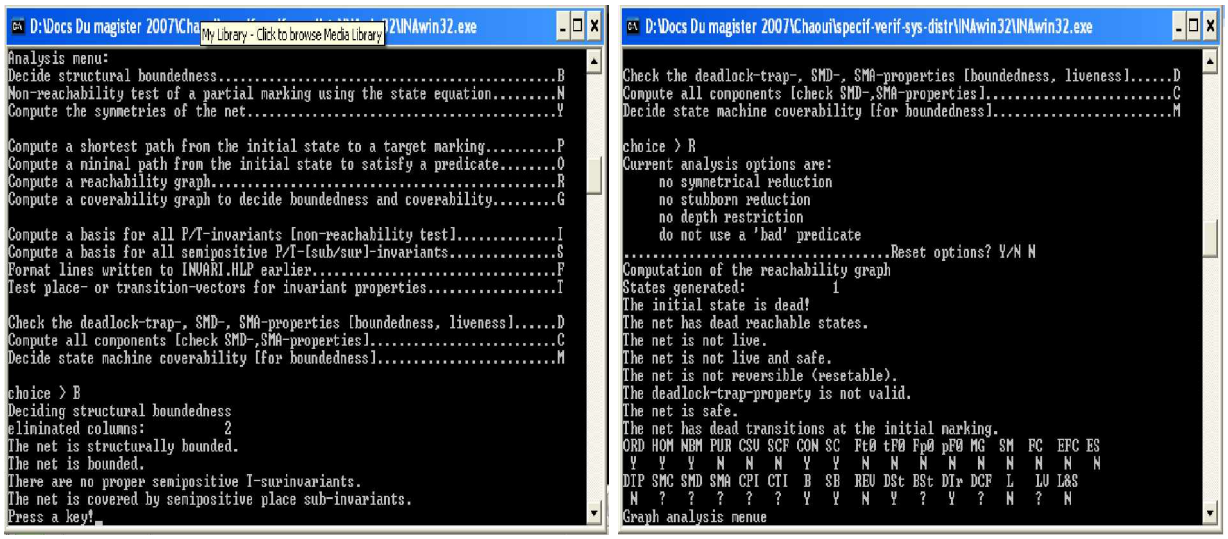

Figure 3.30 Analyse comportementale du modèle pour la variante n°1.

D'après les deux écrans d'INA, nous observons que le modèle est non vivant et borné et Il a des transitions mortes dés le marquage initial et le graphe des états désirables ne peut pas être calculé. Le fichier d'extension ''**.gra**'' est illustré dans la Figure suivante :

```
State nr.   1
P.nr:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
   : 23
toks:  1  0  0  0  1  0  0  0  1  0  0  0  1  1  0  0  0  0  1  1  0  0  0
   :  0
dead state
```

Figure 3.31 Fichier du graphe de marquage généré par l'outil INA pour la variante n°1.

b. Variante n°2

Le fichier d'entrée pour l'outil INA est généré, comme le montre la Figure suivante

```
P   M    PRE,POST   NETZ 1:
  0 1      0 , 1 2
  1 0      2 , 3
  2 0      1 3 , 4
  3 0      4 5 , 0 5
  4 1      6 , 7 5
  5 0      5 , 8
  6 0      7 8 , 9
  7 0      9 2 , 6 2
  8 1      9 0 , 0 8 7
  9 1      4 6 , 6 3 1
 10 0      8 7 , 4
 11 0      1 3 , 9
 12 0      0 , 7 8
 13 0      6 , 1 3
@
place nr.                name capacity time
        0: Feeder_dp              00     0
        1: Feeder_wait            00     0
        2: Feeder_wrap            00     0
        3: Feeder_out             00     0
        4: Conveyor_dp            00     0
        5: Conveyor_wait          00     0
        6: Conveyor_wrap          00     0
        7: Conveyor_out           00     0
        8: SP1                    00     0
        9: SP3                    00     0
       10: SP6                    00     0
       11: SP4                    00     0
       12: SP2                    00     0
       13: SP5                    00     0
@
trans nr.                name priority time
        0: Feeder_new              0     0
        1: Feeder_go               0     0
        2: Feeder_ab               0     0
        3: Feeder_start            0     0
        4: Feeder_exit             0     0
        5: Conveyor_ab             0     0
        6: Conveyor_new            0     0
        7: Conveyor_go             0     0
        8: Conveyor_start          0     0
        9: Conveyor_exit           0     0
@
```

Figure 3.32 Fichier d'entrée de l'outil INA généré pour la variante n°2.

L'analyse avec l'outil a donné les résultats suivants :

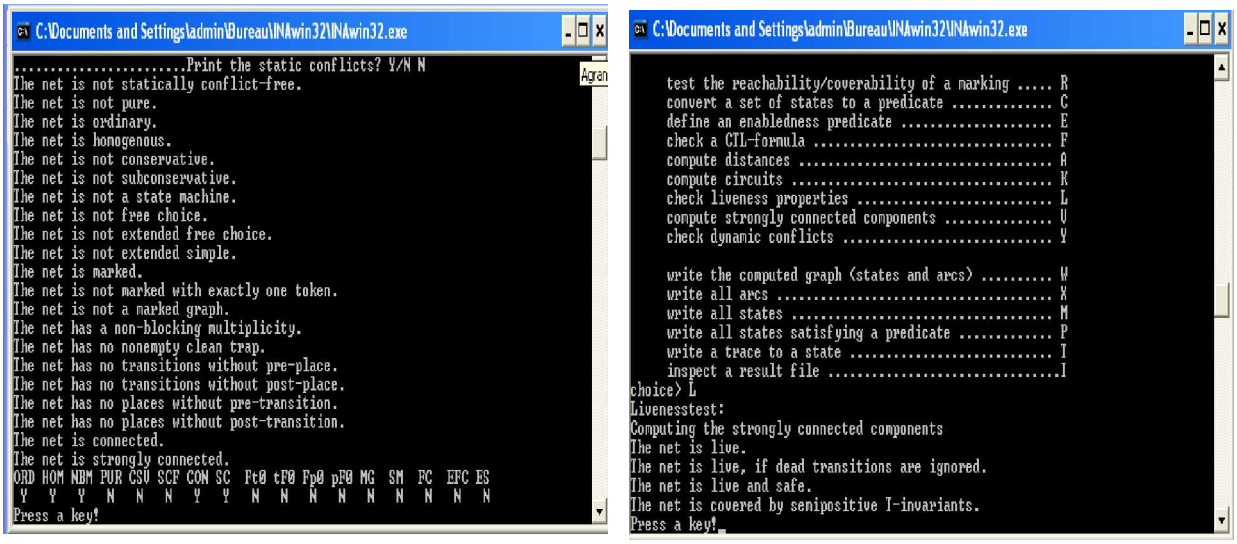

Figure 3.33 Analyse comportementale du modèle pour la variante n°2.

D'après les résultats obtenus, nous observons que le modèle est vivant et borné, donc le graphe des états désirables peut être calculé.

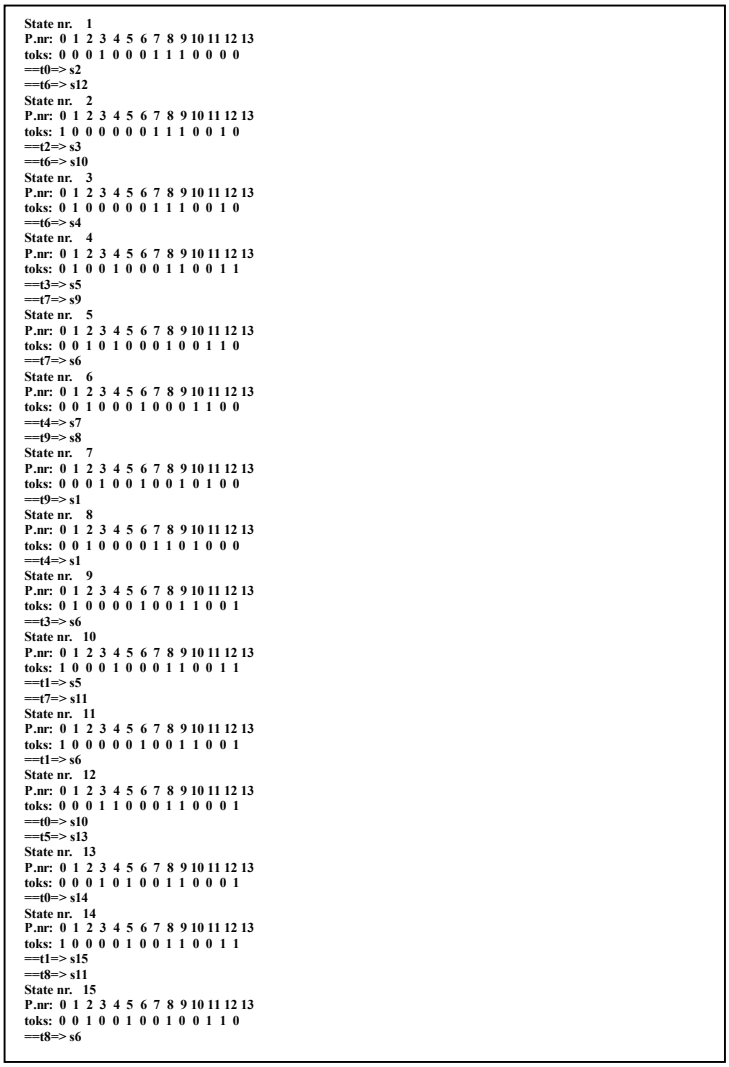

Figure 3.34 Fichier du graphe de marquage généré par l'outil INA pour la variante n°2.

Le format graphique correspondante à ce fichier texte est la suivante :

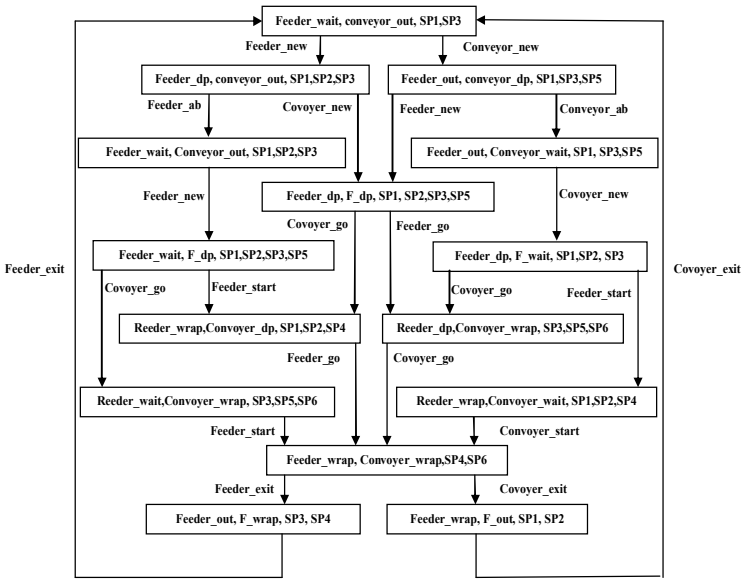

Figure 3.35 GDS généré par l'outil INA.

3.3 Conclusion

A partir d'une chaîne d'embouteillage et une autre d'emballage de produits, deux exemples ont été proposés pour la raison qu'ils sont traités de manière manuelle dans un travail précédent [Bordbar, 2000], donnant la possibilité de valider nos résultats. Ces deux exemples nous ont permis d'appliquer notre méthode de conception d'un superviseur de contrôle. Les deux exemples ont suivi les différentes étapes menant à la modélisation d'un processus de production distribué, et la vérification de ses propriétés en vue de sa supervision.

Nous avons étudié deux variantes pour chaque exemple. Dans le premier exemple concernant la chaîne d'emballage nous avons modélisé le processus tel qu il est dans la première variante, ensuite nous avons modifié l'état initial des machines composant la chaîne et nous avons trouvé que le processus risque d'être dans un état de blocage. .

Dans le deuxième exemple, Nous avons essayé de modifier la synchronisation entre les machines composant la chaîne d'embouteillage, après la vérification des propriétés, nous avons obtenu que le modèle est non vivant et le graphe des états désirables ne peut pas être calculé, indiquant qu' il y a un problème de blocage dans le processus d'embouteillage.