F.S.D.……№ d'Ordre……..

Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

Université Saâd Dahlab de Blida

Faculté des Sciences

Département d'Informatique

Mémoire présenté par : **Mr General Helton Luis**
Mr Mahesh Edvaldo da Gloria

En vue d'obtenir un diplôme de Master

Domaine : Mathématique et Informatique

Filière : Informatique

Spécialité : Informatique

Option : Ingénierie de Logiciel

Sujet :

Conception et Réalisation d'un modèle de négociation multi-agents pour l'ordonnancement des Ateliers de Production

Promotrice : Mme Ouarahni Leila

Encadreur : Mr Djiar Sofiane

Organisme d'accueil :

Soutenue le : 27/01/2014 devant le jury composé de :

Melle	N. Boustia	Président
Melle	M. Azzouz	Examinateur
Mr	M. Hammouda	Examinateur

2012/2013

Remerciements

Avant tout, nous rendons grâce à DIEU tout puissant de nous avoir accordé la volonté et le courage pour réaliser ce mémoire.

Notre plus grande gratitude à Mme Ouharani Leila qui a accepté d'encadrer et a bien, avec patience et rigueur et sa finesse d'anlyse, su porter un regard critique et constructif et de nous avoir guidé, encore et très vivement mon grand merci.

Le travail présenté dans ce mémoire a été réalisé à CR-Metal Blida au sein de l'équipe d'informatique sous la direction du Mr Sofiane notre encadreur. Nous tenons à le remercier très vivement par son proposition sur le sujet et par ses précieux conseils et son soutien dont nous avons bénéficié tout au long de notre collaboration.

Pour l'honneur qu'ils nous font en acceptant d'examiner ce travail nous remercions à tous les membres du jury.

Nous sommes fiers d'exprimer notre gratitude à Manave Edmundo, compatriote et étudiant en Master GSI dans notre département, par son collaboration dans le plan de recherche, réalisation et matériel.

À nos familles même lointaines ont su par leur soutien illimité, notre grand merci du fond du cœur.

Nous adressons également une pensée particulière à tous les Professeurs et membres et personnel technique et administratif du Département d'informatique et de l'Université Saad Dahlab, par leur enseignement, gentillesse, aide et disponibilité.

Un grand merci à notre pays Mozambique et Algérie qui avec leur amitié nous ont permis de continuer avec nos études supérieures en Algérie.

RESUME

Dans ce travail, nous nous intéressants aux problèmes du type Job-Shop, on cherchant à la minimisation du makespan qui représente la fin d'achèvement de toutes les taches du problème. Le problème est du type NP-Difficile et pour sa résolution on propose une nouvelle méthode basée sur les systèmes multi agents, où on propose une heuristique de construction de la solution de départ et elle est améliorée en utilisant une recherche locale et la meta-heuristique de kangourou en prenant en compte les ressources (humaines et matériels) pour la génération du meilleur ordonnancement. On va utiliser la plateforme de développement JADE pour l'implémentation du modèle proposé.

Mots clés : Ordonnancement, Systèmes Multi Agents, Algorithmes génétiques, Ressources Humaines, JADE.

SUMMARY

In this work, we are interested with the problems of the Job-Shop type, by looking into the minimization of the makespan which represents the completion of all the tasks of the problem. The problem is of the Np-Difficult type and for its' resolution we propose a new method based on the multi agent systems, whereby a heuristic construction of the starting solution is proposed and it is improved by using a local research and the meta-heuristics of a kangaroo, by taking account ofthe resources (human and material) for the generation of best scheduling. We will use the development platform JADE for the implementation of the model suggested.

Key words: Scheduling, Systems Multi Agents, Algorithms genetic, Human Resources, JADE.

Liste des abréviations

ACC : Agent Communication Chanel.

ACL : Agent Communication Language.

AG : algorithmes génétiques.

AUML: Agent Unified Modeling Language.

BDD: Base de Données.

CFP: Call For Proposition.

CRMOSMA : CR Metal Ordonnancement par Systèmes Multi Agents.

CRUD: Create, Read, Update, Delete.

DF: Directory Facilitator.

FIPA: Foundation for Intelligent Physical Agents.

GLS : Guided Local Search (recherche locale guidée).

HJS : Heuristique Job Shop.

IAD : Intelligence Artificielle Distribué.

IHM : Interface Homme Machine.

JADE: Java Agents Development Framework.

JDBC: Java DataBase Connectivity.

KQML : Knowledge Query ManipulationLanguage.

MVC : Modèle Contrôleur Vue.

PSE : Procédures de Séparation Evaluation.

SMA : Systèmes multi agents.

TS: Tabu Search.

UML : Unified Modeling Language.

Notations

J : ensemble de jobs, $J = \{J_1, J_2, ..., J_n\}$;

M : ensemble de machines, $M = \{M_1, M_2, ... M_m\}$;

O : ensemble des opérations $i \in O$ tel que $\exists\, j \in J$, $\exists k \in [1, n_j]$, avec $i = O_{j,k}$ où $O_{j,k} \in O$ représente la $k^{ième}$ opération du job j;

P : ensemble des opérations qui n'ont pas de prédécesseurs dans O. $i \in P$ signifie que $\exists j \in J$, tel que $i = O_{j,1}$ est la première opération machine du job j.

U : ensemble d'opérations qui n'ont pas de successeurs dans O. $i \in U$ signifie que $\exists j \in J$, tel que $i = O_{j,nj}$ est la dernière opération du job j.

E : ensemble de toutes les opérations du problème;

p_i : temps opératoire de l'opération $i \in O$;

μ_i : la machine où s'exécute l'opération $i \in O$;

$(i$ -1) : le prédécesseur de l'opération i dans le même job

$(i$ +1) : le successeur de l'opération i dans le même job

H : Un nombre entier positif suffisamment grand

Liste des Tableaux

Liste des Figures

CHAP I : Ordonnancement des ateliers de production

CHAP II : Agent et Système Multi Agents

CHAP III : Modélisation de la solution

Liste des Figures

Liste des Figures

Table des Matières

CHAP II : Agent et Système Multi Agents

Introduction Générale

Dans des systèmes de production, l'ordonnancement de la production est une activité importante où les tâches et les dates limites doivent être assignées aux ressources de production tout en respectant des contraintes et optimisant des objectifs. Le problème d'ordonnancement a été largement étudié et beaucoup des méthodes ont été proposées. Parmi ces méthodes, nous nous intéressons aux Systèmes Multi Agents (SMA). Les Systèmes Multi Agents (SMA) présentent un très bon support de modélisation des ateliers de production. Les agents peuvent représenter toute entité autonome qui cherche à réaliser un but seule ou en coopérant avec les autres entités de son environnement. Les agents interagissent en communicant, négociant et coordonnant leurs tâches.

Les activités de planification et d'ordonnancement sont étroitement liées car le choix d'un bon plan influence sur l'optimalité de l'ordonnancement. De cette idée, des chercheurs ont pensé à intégrer planification et ordonnancement pour trouver des solutions plus optimales.

Ce projet de fin d'études consiste en proposer une nouvelle méthode basée sur les systèmes multi agents pour le problème d'ordonnancement d'atelier de production de type Job-Shop de la société de construction métallique CR. METAL BLIDA. Nous utilisons la plateforme JADE pour l'implémentation du modèle proposé. Des tests doivent être effectués pour mesurer les performances de ce dernier.

Problématique :

La société CR-METAL est une entreprise spécialisée dans la réalisation (fabrication et montage) de produits métalliques. Les technologies employées par CR-METAL dans la fabrication de la charpente métallique, chaudronnerie, serrurerie, sont conventionnels.

Elles se résument en :

- Débitage dans ses différentes formes ;
- Usinage mécanique et fabrication d'outillage ;
- Assemblage par boulonnage et/ou par soudure ;

- Formage de tôles ;
- Traitement de surface.

La plupart des travaux considèrent que les machines sont toujours disponibles et en bon état, alors que ce n'est pas toujours le cas dans la pratique.

L'entreprise dispose de certaines ressources critiques que ne sont pas toujours suffisante pour la réalisation de toutes les tâches dans un instant donnée.

Le choix de la tâche à effectuer ou bien le choix de la commande à réaliser est fait aléatoirement sans vérification du temps de réalisation ni même la disponibilité des ressources dans des étapes suivantes, ce que donne :

- L'indéterminisme dans le temps de réalisation ;
- Le non respect des délais de livraison ;
- L'accumulation de commandes.

Objectif :

Notre objectif est de proposer une nouvelle méthode basée sur les systèmes multi agents pour le problème d'ordonnancement d'atelier de production qu'intègrent les activités de planification des taches en prenant en compte les ressources, afin de :

- Optimiser le temps de production ;
- Optimiser l'organisation des taches.

Notre Proposition :

D'après l'atelier de production de la CR. Metal (figure 1), nous avons considère un atelier comme étant un agent et dans le système multi agent ils utilisent un protocole de négociation pour l'allocation dynamique des ressources, un agent a comme base de connaissance ces ressources et les plannings de production de son propre atelier.

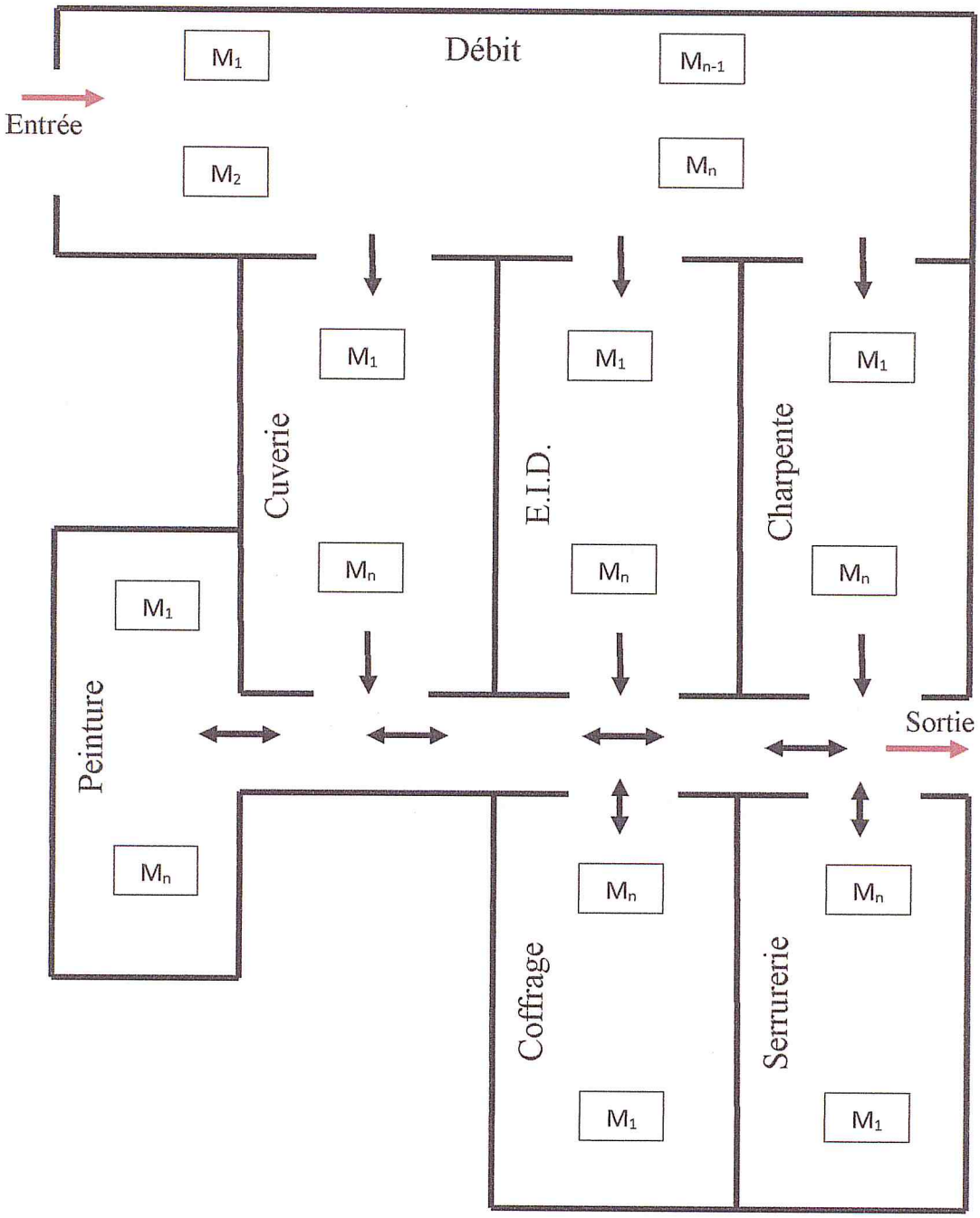
Les agents :

L'utilisateur du système (ordonnanceur) pour lancer un ordonnancement de la production réalisera une demande qu'est composée des ressources nécessaires, les opérations et temps d'exécution. Notre système sera composé de huit agents cognitifs

qui vont passer par des mécanismes de négociation pour l'allocation de ressources nécessaires à la réalisation de tâches parce que la base de connaissance est décentralisée et l'agent chargé d'ordonnancement réalise l'ordonnancement:

- L'agent ordonnanceur : c'est l'agent qui donne prise en charge à une demande de l'utilisateur (ordonnanceur) en déterminant les agents possédant les ressources nécessaires(machines) à la réalisation de la tâche et il réalise des négociations pour avoir la disponibilité des ressources, après avoir la disponibilité il réalise l'ordonnancement, pour améliorer le résultat d'ordonnancement initial il lance l'algorithme de recherche combiné avec l'algorithme du kangourou, du résultat initial il informe les agents concernées des tâches à effectuer et informe aussi l'utilisateur du résultat d'ordonnancement,
- Les autres agents représentent un atelier (débit, serrurerie, peinture, cuverie, charpente, coffrage et E.I.D-Equipement Industriel Divers) : chacun contient le planning des travaux des ressources de son propre atelier et à travers ce planning, dans une négociation avec l'agent ordonnanceur, il va donner un temps de disponibilité de la ressource demandé par l'agent ordonnanceur, après réception de nouveaux travaux à réaliser il va réaliser une mise à jour sur son planning.

Architecture des ateliers de production de la CR-METAL



Organisation du Mémoire :

Ce présent document est organisé en chapitres définis dans un ordre bien précis, afin de maitriser l'enchainement de notre sujet, en effet :

- Chapitre 1 défini le contexte d'étude qui est l'ordonnancement, ainsi qui les différentes notions qui lui sont associées, notamment l'Ordonnancement d'atelier ainsi qui les problèmes d'ordonnancement.

- Chapitre 2 nous abordons sur systèmes multi-agents dans lequel nous introduisons les aspects théoriques liés à ce concept.

- Chapitre 3 traite la modélisation de job shop et l'implémentation de l'heuristique pour la construction de la résolution et méta heuristiques pour l'amélioration de la solution.

- Le chapitre 4 sera consacré à la réalisation et nous l'achèverons par une suite d'expérimentations de notre approche.

- Finalement nous terminons par une conclusion générale qui récapitule les principales observations concernant l'évolution du travail et nous indiquons également comment le travail réalisé tout au long de ce mémoire pourrait être amélioré, en donnant la perspective de nouvelles approches de recherche.

CHAP 1
Ordonnancement

1. Introduction :

L'ordonnancement occupe une place particulière dans la gestion informatisée des flux de production au sein de l'entreprise. C'est le lieu de l'interface entre l'élaboration globale de la commande et la partie opérationnelle. La gestion de production crée les ordres de fabrication qui déterminent globalement ce qui doit être fait dans une période donnée. L'ordonnancement trouve sa place essentiellement dans tous les ateliers ayant une production diversifiée en flux poussé, gérée par des ordres de fabrication.

L'ordonnancement est en effet un élément crucial dans l'ensemble de tâches liées au pilotage de l'atelier, que l'on peut décrire par un cycle d'ordonnancement, de contrôle, de suivi, de réaction. Dans notre travail, nous nous limitons à l'aspect ordonnancement qui joue un rôle central, surtout lorsque les ateliers ne sont pas entièrement automatisés.

2. Ordonnancement :

Les différentes données d'un problème d'ordonnancement sont les tâches et leurs caractéristiques, les ressources, les contraintes potentielles, et le(s) critère(s) d'optimisation. Souvent une tâche ne peut commencer son exécution avant une date de disponibilité et doit être achevé avant une date échue. Les tâches sont souvent liées entre eux par des relations d'antériorité, si ce n'est pas le cas, on dit qu'ils sont indépendants.

2.1. Définition :

Le problème d'ordonnancement consiste à organiser dans le temps la réalisation d'un ensemble de tâches, compte tenu de contraintes temporelles et de contraintes portant sur l'utilisation et la disponibilité des ressources requises. **[ESQ, 99].**

Tâche est un travail élémentaire nécessitant un certain nombre d'unités de temps et ressources : elles peuvent passer sur des machines, utiliser de la main d'œuvre, etc. **[GOT, 93]**.

Une ressource est une moyenne technique ou humain requis pour la réalisation d'une tâche et disponible en quantité limité. Une ressource peut être renouvelable ou consommable. **[GOT, 93]**.

Une contrainte [GOT, 93] exprime des restrictions sur les valeurs que peuvent prendre conjointement les variables représentant les relations reliant les tâches et les ressources. On distingue les contraintes temporelles et les contraintes de ressources. Contrainte temporelle intègrent :

- Les contraintes de temps alloué, issues généralement d'impératifs de gestion et relatives aux dates limites des tâches ou à la durée total d'un projet ;
- Les contraintes d'antériorités et plus généralement les contraintes de cohérence technologique, qui décrivent le positionnement relatif de certaines tâches par rapport aux autres, etc. ;

Les contraintes de ressources traduisent le fait que les ressources sont disponibles en quantité limitée. On distingue deux types de contraintes de ressources:

- Disjonctive: ne peut être utilisée que par une tâche à la fois ;
- Cumulative : les tâches non réalisables sont de cardinalité quelconque.

3. Typologie des problèmes :

Les différents problèmes qui l'on rencontre en ordonnancement dépendent principalement des machines et de l'enchaînement des opérations. Implicitement, le premier type de problème rencontré est alors le problème à une machine.

3.1. Les problèmes à une machine [ALO, 06]:

Ce type de problème si caractérise par le fait qui l'on ne dispose que d'une ressource (figure 1) pour traiter un ensemble de tâche. Cette ressource ne peut effectuer le traitement qui d'une tâche à un instant donné.

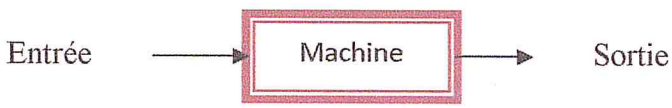


Figure 1 : Atelier à machine unique [ALO, 06].

3.2. Les problèmes à machines parallèles [ALO, 06]:

Ce type de problème consiste à ordonnancer un ensemble de tâches, sur un ensemble de ressources en parallèle (figure 2). On distingue cependant trois sous-classes de problèmes :

- **Machines identiques (P)** : il y a *m* machines identiques en parallèle. Une tâche *j* requiert une opération unique qui peut être réalisée sur n'importe laquelle des *m* machines
- **Machines uniformes (Q)** : ce type de problème est toujours basé sur *m* machines en parallèle. Une tâche *j* requiert toujours une opération unique qui peut être réalisée sur n'importe laquelle des *m* machines. Cependant, la durée $p_{i,j}$ de la tâche *j* sur la machine *i* sera différente suivant la machine employée.
- **Machines indépendantes (R)** : cet environnement est une généralisation du précédent. Il y a *m* machines différentes en parallèle. La durée $p_{i,j}$ de la tâche *j* sur la machine *i* sera différente suivant la machine employée.

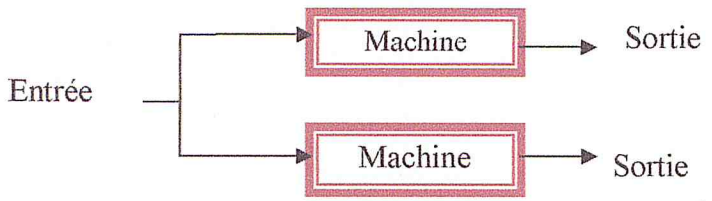


Figure 2 : Atelier à machines parallèles [ALO, 06].

Dans cette partie, les tâches ne sont composées que d'une opération. Or, dans la réalité, les tâches peuvent requérir une succession d'étapes de traitement sur différentes machines. On parle alors de problèmes d'atelier multi-machines.

3.3. Les problèmes d'atelier multi-machines [ALO, 06]:

Les ateliers considérés ici sont composés d'un ensemble de *m* machines. Dans ces ateliers doivent être réalisés des travaux (aussi appelé job). Chacun de ces travaux peut être décomposé en tâches. Ces sont les opérations réalisées successivement sur les différentes machines. En fonction de mode passage sur les différentes machines, il sera possible d'identifier trois types d'atelier :

- **Flow Shop (figure 3)** : dans c0e problème d'ordonnancement d'atelier qui correspond à un atelier à cheminement unique, *n* tâches utilisent les *m* machines qui

le composent. Chaque tâche passe tour à tour sur chaque machine dans le même ordre.

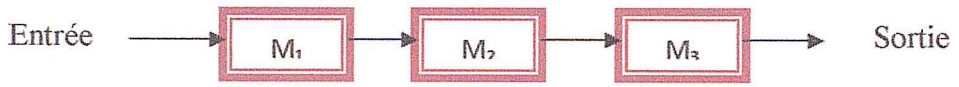


Figure 3 : Flow-shop à trois machines [ALO, 06].

- **Open Shop (figure 5)** : ce type d'atelier est à cheminement libre. L'ordre des opérations n'est pas fixé, à priori.

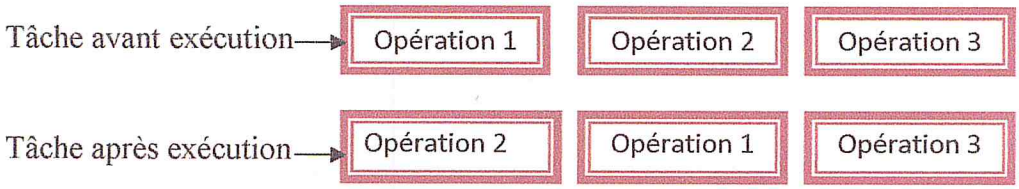


Figure 4 : Atelier de type Open Shop [ALO, 06].

- **Job Shop (figure 4)** : les *n* tâches utilisent les *m* machines qui composent l'atelier. Cependant, dans cet atelier, le cheminement est multiple et il est donc propre à chaque type de job. Les problèmes de type Job-Shop sont parmi les plus étudiés dans la littérature à cause de leurs ressemblances avec les problèmes réels, ce que sera modélisé en proposant des méthodes approchées pour sa résolution dans le chapitre 3 puisque notre atelier est de ce type.

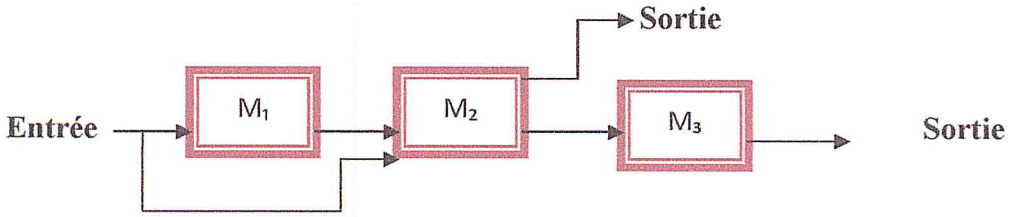


Figure 5 : Atelier de type job-shop à 3 machines [ALO, 06].

3.3.1. Définition formelle du Job-Shop [LAR, 10] :

Le problème du job-shop consiste en l'exécution d'un ensemble de *n* jobs $J=\{J_1,\ldots, J_n\}$ sur un ensemble de *m* machines $\{M=M_1,\ldots,M_m\}$. Chaque job J_i est composé d'un ensemble ordonné de n_i opérations notées $O_{i,1}$, $O_{i,2}$, …, $O_{i,ni}$ pour lesquelles des contraintes de précédence sont décrites. A chaque opération machine $O_{i,j}$ est associé un temps de traitement p_{ij} correspondant à sa durée d'exécution.

Chaque machine ne peut traiter qu'une opération à la fois et un job ne peut s'exécuter que sur une seule machine en même temps. Les jobs exécutés sur une machine M_i doivent attendre dans le stock de sortie. De la même manière chaque machine M_i dispose d'un stock d'entrée où les jobs attendent leur exécution sur cette machine. Le temps de transfert des jobs n'est pas pris en compte.

On suppose que les valeurs p_{ij} sont des entiers non-négatifs. L'objectif est de trouver un ordonnancement faisable qui minimise le makespan C_{max}.

$$C_{max} = \underset{J=1,n}{Max} \{C_j\}$$

Où C_j dénote la fin d'exécution de la dernière opération $O_{j,nj}$ du job j. Le résultat de l'ordonnancement est constitué des dates de début de toutes les opérations.

Les hypothèses suivantes sont communément retenues pour le job-shop.

Certaines variantes du problème de job-shop consistent à supprimer certaines de ces hypothèses :

- Tous les jobs sont disponibles dès le début de l'ordonnancement.
- Les machines sont disponibles sur tout l'horizon d'ordonnancement.
- Les temps de traitement p_i sont déterministes et connus à l'avance.
- Chaque machine ne peut réaliser à un instant donné qu'une seule opération.
- Un job peut être traité au plus par une machine à un instant donné.
- Les produits peuvent attendre dans les zones de stockage de capacité illimitée.

3.3.2. Modélisation sous forme d'un graphe disjonctif-conjonctif :

Le graphe disjonctif a été introduit pour la première fois en **[ROY, 84]**. La particularité de ce graphe est qu'il utilise des arêtes non orientées qui peuvent prendre les deux sens, l'évaluation du graphe n'est possible qu'une fois toutes les arêtes sont orientées. D'une manière formelle le graphe conjonctif-disjonctif est un graphe dont les nœuds représentent des opérations, les arcs et des arêtes des contraintes temporelles de la forme $t_j \geq t_i + a_i$, entre les opérations (i, j) où t_i, t_j sont leurs dates respectives de début et a_i durée de l'opération i.

Pour orienter ce graphe il suffit donc de fixer l'ordre des opérations au préalable et ainsi obtenir un graphe conjonctif orienté qu'il suffit d'évaluer avec un algorithme

de plus long chemin de type Bellman. Le résultat obtenu est alors les dates de début au plus tôt des opérations.

Dans la suite, on détaille la construction des deux graphes : le graphe disjonctif (Figure 6) qui modélise le problème et dans lequel à priori aucune disjonction n'est arbitrée et le graphe conjonctif (Figure 7) qui modélise une solution.

On note $G = (V, A, E)$ un graphe disjonctif, où V est un ensemble de nœuds, A est un ensemble d'arcs et E est un ensemble d'arêtes. Les ensembles V, A et E sont construits de la manière suivante :

- Pour chaque opération du problème de job-shop, on crée un nœud dans V. On crée de plus deux nœuds *0* et * qui modélisent deux opérations fictives.
- On crée un arc dans A entre deux opérations dans la gamme du job.
- On crée une arête dans E entre deux opérations traitées par la même machine.

Le Tableau 1 propose une instance de job-shop et la Figure 6 présente le graphe disjonctif exemple d'une solution S associé.

	Op1	Op2	Op3
Job1	(M3, 5)	(M1, 3)	(M2 ,4)
Job2	(M3, 6)	(M4, 2)	(M2, 3)
Job3	(M3, 4)	(M1, 1)	(M4, 5)

Tableau 1 : Exemple de Job-Shop

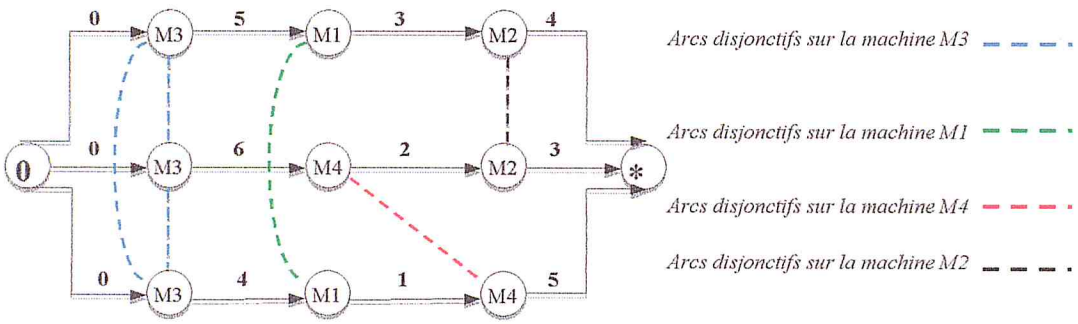


Figure 6 : Graphe disjonctif de l'exemple de Job-Shop.

L'étape suivante est l'orientation des arêtes du graphe disjonctif. Orienter une arête entre deux opérations consiste à remplacer cette arête par un arc dans le sens de l'orientation. La longueur de cet arc est la durée de traitement de l'opération de départ de l'arc.

Le modèle de graphe conjonctif-disjonctif permet de calculer l'ordonnancement semi-actif.

L'obtention de l'ordonnancement semi-actif passe donc par les étapes suivantes :

- analyse du problème de départ pour construire le graphe disjonctif ;
- un ordre d'opération est considéré. Cet ordre peut être fourni par un expert, peut être calculé ou obtenu par un algorithme d'optimisation.
- le graphe disjonctif et l'ordre des opérations permettent de construire le graphe conjonctif. Ensuite, un algorithme de plus longs chemins permet de calculer l'ordonnancement semi-actif associé.

S'aidant de l'ordre d'exécution des opérations sur les différentes machines donné par le tableau 2, nous allons orienter le graphe disjonctif.

Machine	Opération	Opération	Opération
M1	Op 2, Job 1	Op 2, Job 2	-
M2	Op 3, Job 1	Op 3, Job 2	-
M3	Op 1, Job 2	Op 1, Job 1	Op 1, Job 3
M4	Op 2, Job 2	Op 3, Job 3	-

Tableau 2 : Séquence des opérations

L'exemple de la solution *S* est évalué dans le graphe de la Figure 7. La date de début de l'opération * est la date de fin de la dernière opération. Le chemin le plus long donne le makespan de cette solution qui est donc 21.

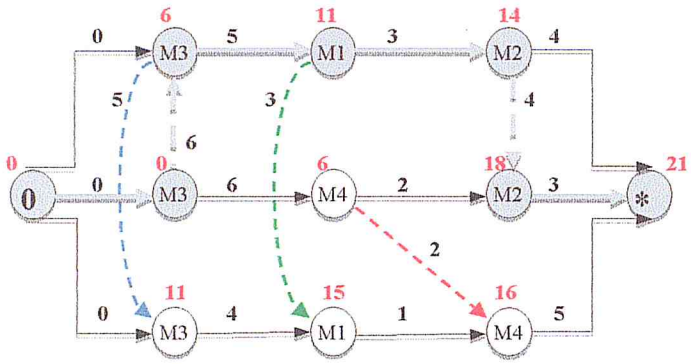


Figure 7 : Graphe conjonctif résultant de l'orientation des arcs.

Le chemin encadré en gris constitue le chemin critique et la succession des 3 opérations sur la machine M3 s'appelle le bloc critique « bloc machine ». Ce chemin

est remarquable car tout retard d'une opération le long de ce chemin retarde la date de fin de l'ordonnancement. Le chemin critique est le chemin le plus long du graphe conjonctif et c'est lui qui définit le makespan de l'ordonnancement correspondant à ce graphe.

3.3.3. Représentation sous forme de diagramme de Gantt:

La représentation des résultats du problème d'ordonnancement peut être faite par plusieurs méthodes, la plus utilisée est la représentation par le diagramme de Gantt.

Ce diagramme, créée par Henry Laurence Gantt et très utilisé en gestion de projets, a pour but de représenter de manière claire et rapide la solution à un problème d'ordonnancement. Il se représente par un ensemble de lignes horizontales tracées dans un graphe avec le temps en abscisses. Chaque ligne est composée de différents rectangles qui sont pour chacun la représentation du temps nécessaire à la réalisation d'un travail. Dans le cadre d'un problème d'ordonnancement, le diagramme de Gantt prend en ordonnée les différentes ressources utilisées.

En utilisant l'exemple donné par le tableau 1 et l'ordre d'exécution donnée par le tableau 2 on donne une illustration de la représentation de solution en utilisant le diagramme de Gantt.

Si on suppose que chaque opération est calée le plus à gauche (elle définit alors une solution semi-active), on obtient le diagramme de la Figure 8.

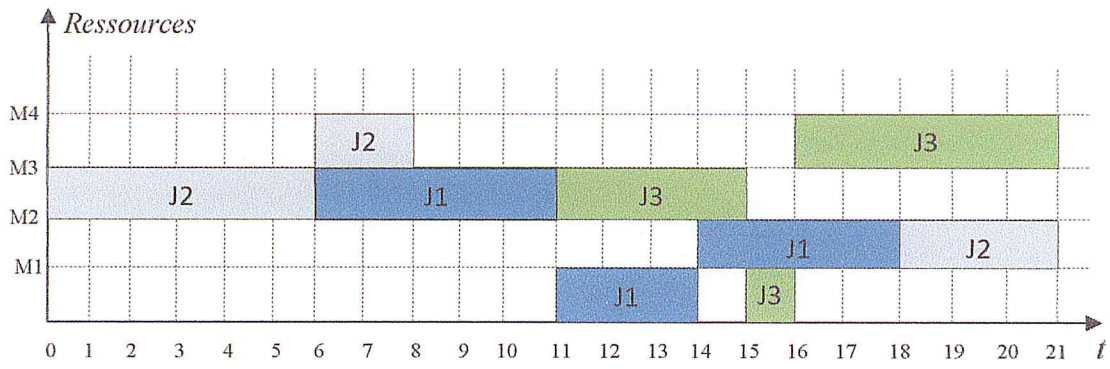


Figure 8: Représentation de la solution par le diagramme de Gantt

4. Les méthodes d'optimisation :

On peut séparer les méthodes d'optimisation en deux familles distinctes : les méthodes exactes et les méthodes approchées. Dans ce qui suit on donne une description des méthodes les plus employées en recherche opérationnelle.

4.1. Méthodes exactes [ADD, 11] :

Une manière intuitive de résoudre les problèmes d'optimisation discrets consiste à énumérer tout ou une partie des solutions. Néanmoins, pour un grand nombre de problèmes, cette énumération s'avère impossible à cause des temps de calcul prohibitifs entrainé par l'énumération.

4.1.1. La programmation linéaire :

Il y a trois grandes familles de programmes linéaires :

- **Les programmes linéaires simples** ont des variables dites "continues". Ces programmes linéaires peuvent être efficacement résolus par la méthode du simplexe ou la méthode du point intérieur.
- **Les programmes linéaires en nombres entiers** font intervenir des variables entières. Ces programmes peuvent paraître plus simples que les précédents dans la mesure où le nombre de solutions possibles est fini. Pourtant, c'est ce type de problème qui est le plus difficile à résoudre.

4.1.2. Les procédures de séparation/évaluation (PSE)

La technique par séparation/évaluation (également rencontrée sous la dénomination Branch&Bound) réalise une énumération implicite de toutes les solutions : elle consiste à trouver une solution optimale sans parcourir toutes les solutions du problème.

Pour y parvenir, deux fonctionnalités sont introduites : la séparation et l'évaluation. La séparation consiste à partitionner un problème en deux ou plusieurs sous-problèmes. L'évaluation fournit pour chaque nœud de l'arbre une valeur appelée « borne inférieure ».

Ce processus est réitéré sur chacun des sous problèmes jusqu'à leur résolution

ou leur rejet.

L'utilisation conjointe des méthodes approchées et des méthodes exactes existe. Plusieurs chercheurs essaient de combiner ces méthodes pour profiter de leurs avantages.

4.2. Méthodes approchées [CAR, 88] :

Le principe d'une méthode heuristique est de trouver en un temps polynomial une solution réalisable la meilleure possible.

La connaissance de l'espace de recherche des solutions d'un problème permet de guider le choix d'une méthode. Souvent ces espaces sont énormes et multidimensionnels et le processus de recherche est fastidieux, piégeant ainsi de nombreux algorithmes donnent des solutions qui ne sont pas optimales. Ces solutions sont appelées *minimums locaux*. Le but d'une méthode d'optimisation est donc d'éviter de rester bloqué dans ces endroits (*minimums locaux*).

4.2.1. Les heuristiques de construction :

Ce sont des méthodes très simples à mettre en œuvre et qui dépendent du domaine d'utilisation. Elles sont souvent utilisées pour avoir rapidement des solutions admissibles (réalisables) de qualité relativement bonne, bien que, en générale leur performance ne puisse être garantie. Les heuristiques de construction sont souvent couplées avec d'autres méthodes.

4.2.2. Les méthodes amélioratrices :

Le principe de ces méthodes consiste à modifier le résultat d'une solution admissible en vue d'améliorer la valeur de la fonction objectif.

Parmi les méthodes qui ont prouvé leur efficacité dans la résolution de problème d'optimisation combinatoire, nous pouvons citer le recuit simulé, la recherche tabou, le kangourou, la recherche dispersée et la colonie des fourmis.

4.2.3. Les méthodes de recherche locale :

Elles se basent sur le principe d'amélioration d'une solution de départ x à chaque

pas de leur progression, on définit alors un voisinage *N(x)* propre au problème étudié.

Dans la suite nous détaillons les principaux métas heuristiques.

- **La descente stochastique :** Elle part d'un principe qui est basé sur le voisinage d'une solution X. Le voisin Y obtenu est acceptée si et seulement si Y est meilleur que X. Le seul paramètre important d'un algorithme de descente stochastique est la méthode de sélection ou de génération d'un voisin. La descente a en revanche le principal inconvénient de rester bloquée dans un minimum local.
- **Les descentes stochastiques successives :** Cette méthode consiste à effectuer plusieurs descentes stochastiques en partant de plusieurs solutions de départs différentes.

La méthode des descentes stochastiques successives converge très lentement, en probabilité, vers l'optimum. Les descentes stochastiques effectuées sont indépendantes et sont réalisées à partir de solutions initiales différentes qui ne prennent pas en compte d'éventuelles informations sur les précédents minimas locaux trouvés.

4.2.4. La méthode de recuit simulé [YAN, 02]:

Cette méthode s'appuie sur un principe physique, qui est celui du recuit utilisé par les forgerons pour donner de la résistance et améliorer la structure du fer en alternant des périodes de chauffe (recuit) et de refroidissement. Cette méthode et sous certaines conditions peut atteindre l'optimum global avec une probabilité proche de 1.

4.2.5. La méthode TABOU [YAN, 02]:

La méthode taboue TS (Tabu Search) fonctionne avec une mémoire. A chaque mouvement on met à jour une liste de mouvements interdits dits tabous d'où le nom de la méthode.

Cette méthode élabore une liste des voisins avec n mouvements non tabous ; on sélectionne le meilleur voisin qui deviendra par la suite la solution courante, on insère alors le mouvement qui a donné le meilleur voisin dans la liste tabous, et on réitère ce processus jusqu'à la satisfaction d'un critère d'arrêt.

4.2.6. La méthode de Kangourou [FLE, 95] :

Cette méthode étend et améliore la descente stochastique en autorisant la détérioration de la solution en cours pour ainsi éviter des blocages prématurés dans des minimums locaux. Elle se comporte à la manière d'un kangourou. Le principe de cet algorithme est d'effectuer une descente stochastique, et lorsque l'état minimal actuel est de même coût depuis trop longtemps, d'accepter une transition défavorable dans un voisinage de l'état actuel (pas nécessairement le même que celui utilisé pendant les descentes) et ceci quel que soit le coût. Ceci est appelé un saut. Ensuite on débute une nouvelle descente stochastique.

Soit :

- V : le système de voisinage utilisé lors de la descente ;
- W : le système de voisinage utilisé lors des sauts ;
- A : le nombre maximum d'itérations à effectuer avant un saut.

Le deuxième système de voisinage W utilisé par la méthode n'est pas nécessairement le même que V. Le voisinage W doit être tel que tout élément puisse être joint à tout autre par une chaîne finie d'état deux à deux voisins au sens de W. Les Descentes Successives ne sont de fait qu'un cas particulier de l'algorithme du Kangourou, pour lequel W(x) pour tout x, représente l'espace de recherche complet. Le fait d'effectuer des 'sauts' permet à la méthode de sortir d'une vallée c'est-à-dire d'un minimum local.

Lorsque la méthode se trouve dans une vallée contenant un optimum local, il effectue après A itérations sans amélioration un « saut ».

4.2.7. Les Algorithmes Génétiques [YAN, 02] :

Les algorithmes génétiques (AG) sont certainement la branche des algorithmes évolutionnistes la plus connue et la plus utilisée. Dans ce cas, il s'agit de simuler l'évolution naturelle d'organismes (individus), génération après génération, en respectant des phénomènes d'hérédité et une loi de survie. Dans une population d'individus, ces sont en général les plus forts, c'est à dire les mieux adaptés au milieu, qui survivent et donnent une descendance. Par ailleurs, on suppose que les qualités et les défauts peuvent être hérités des parents de manière stochastique.

4.2.8. La méthode de colonie de fourmis [DRE, 03] :

Elle s'appuie sur le principe de l'auto-organisation : une suite de mouvements désordonnés localement qui contribuent à une organisation sans faille globalement. C'est une organisation où chaque individu a un rôle bien déterminé à jouer. Les colonies de fourmis, les abeilles sont des exemples pertinents de ces systèmes.

Dans les tableaux qui suivent on donne les caractéristiques principales des méthodes d'optimisation (tableau 3) et les caractéristiques des méthodes approchées les plus utilisées dans la littérature (tableau 4) ce qui va nos guider lors des choix qu'on va utiliser pour résoudre le problème traité dans ce mémoire.

Caractéristiques des méthodes d'optimisation	
Méthodes exactes	**Méthodes approchées**
Possèdent un ensemble fini de solutions	Possèdent une infinité de solutions
Elles sont de nature discrète	Elles sont de nature continue
Sont capable de donner une solution exacte pour un problème de petit taille mais elles ne sont pas capable de résoudre des problèmes de grand taille à cause des temps de calcul prohibitifs	Elles fournissent des solutions optimales dans un temps raisonnable pour des problèmes de grande taille

Tableau 3 : principales caractéristiques des méthodes d'optimisation

Caractéristiques de différentes méthodes approchées	
Les heuristiques de construction	- très simples à mettre en œuvre - obtention rapide de solutions amissibles de qualité relativement bonne - les performances de ces solutions ne sont pas garanties - détermination d'une solution selon une règle de construction donnée - pas de possibilité d'améliorer la solution
Les méthodes de recherche locale	- se basent sur le principe d'amélioration d'une solution de départ X - elles donnent des solutions minimales localement - simples à programmer - comportent très peut de paramètres - s'adaptent très facilement à de nombreux problèmes
La méthode du recuit simulé	- se basent sur le principe d'amélioration d'une solution de départ X - donne des solutions minimales globales - sous certaines conditions, elle peut atteindre l'optimum global avec une probabilité proche à 1 - exige un grand nombre de paramètres, ce que rend sa mise en œuvre complexe
La méthode tabou	- se basent sur le principe d'amélioration d'une solution de départ X

	- donne des solutions minimales globales - les implémentations de cette méthode varient avec la taille, la variété et l'adaptabilité du problème étudié
La méthode du kangourou	- se basent sur le principe d'amélioration d'une solution de départ X - donne des solutions minimales globales - s'adaptent facilement à plusieurs types de problèmes - explorent un espace de recherche de solutions complet -permettent de minimiser des fonctions pouvant prendre de valeurs infinies - facile à programmer
Les algorithmes génétiques	- elles procurent un jeu de solutions optimales - très bien adaptés au traitement d'un problème d'optimisation multi-objectif - plus récente et plus utilisée des méthodes ce qui permet de trouver une large documentation dans la littérature lui traitant - se basent sur le principe de la sélection naturelle, de l'évolution, et de l'hérédité, ce qui la permet d'explorer plusieurs solutions
La méthode de colonie de fourmis	- se basent sur le principe d'amélioration d'une solution de départ X - donne des solutions minimales globales - bien adaptée pour résoudre le problème du voyageur do commerce

Tableau 4 : caractéristiques des différentes méthodes approchées

5. Les principaux voisinages proposés pour le Job-Shop [CAR, 88] :

Comme nous l'avons mentionné le chemin le plus long du graphe conjonctif définit le makespan associé à ce graphe. Pour modifier le makespan d'une solution, il est nécessaire de modifier le chemin le plus long. Toute modification du chemin critique le casse et permet d'obtenir un nouvel ordonnancement, mais rien ne garanti que ce dernier soit égal, meilleur ou pire.

La notion de voisinage s'appuie sur des opérations ou des mouvements élémentaires pour passer d'une solution à une autre. Ces mouvements sont effectués sur des successions d'opérations s'exécutant sur une même machine (appelés blocs). On peut citer les voisinages : N1, N2, N3, N4 et N5. Ces voisinages permettent d'obtenir des solutions voisines en faisant des opérations simples sur le chemin critique. Dans le cas où on fait une seule opération le voisinage est dit en 1 OPT, il est en 2 OPT s'il requière 2 opérations et ainsi de suite. Une brève description de ces voisinages est donnée ci-après.

5.1. Voisinages N1 et N2 :

Le voisinage de type N1 consiste à permuter deux opérations consécutives traitées par la même machine au long du chemin critique. Le voisinage N2 dérive de N1. Ce voisinage, consiste à inverser les opérations qui se situent sur les extrémités des blocs. Les voisinages N1 et N2 sont des voisinages en 1 OPT.

5.2. Voisinages N3 et N4 :

Le voisinage N3 est en 3 OPT. Dans le bloc critique on fait trois inversions différentes pour obtenir le voisinage N3. Le voisinage N4 est une variante du voisinage N3.

5. 3. Voisinage N5 :

Le voisinage N5 essai de remédier aux défauts des voisinages précédents. Cela signifie que toute amélioration par un voisin des voisinages N1 et N2 est dans le voisinage N5.

Conclusion :

Dans notre étude de l'ordonnancement nous avons accordé notre attention aux ateliers flexibles de production, et plus particulièrement au problème de job-shop comme problème modélisant le type d'atelier sur lequel on va travailler dans la suite.

Dans ce chapitre, nous avons présenté les méthodes approchées et en particulier mis en évidence que le graphe disjonctif et les voisinages basés sur la notion de blocs et sur le chemin critique sont des approches très efficaces.

Nous avons aussi présenté quelques notions de base concernant l'ordonnancement et les problèmes d'ordonnancement dans les ateliers de production. De nombreuses méthodes et d'outils performants permettent de résoudre de manière efficace les problèmes d'ordonnancement. Ces méthodes peuvent se subdiviser en deux grandes catégories, les méthodes exactes et les méthodes approchées. Dans cette deuxième catégorie, on peut distinguer les heuristiques et les métas heuristiques.

Nous avons vu que les méta-heuristiques utilisent la notion de voisinage.

CHAP 2
Systèmes Multi-Agents

1. Introduction :

Les travaux sur les SMA sont poursuivis activement depuis le début des années 90. Des nouvelles tendances se sont concrétisées tel l'accent mis sur l'apprentissage collaboratif, les interfaces adaptifs et les systèmes multi-agents. Les recherches sur des agents sont guidées par la nécessité d'avoir des environnements interactifs qui tracent à la fois le comportement du système informatique et celui des usagers (tuteur ou apprenant). En effet, lorsqu'il s'agit de concevoir des systèmes informatiques distribué qui manipulent des connaissances hétérogènes, la technologie agent se révèle bien adapté. Les systèmes multi-agents permettent non seulement le partage ou la distribution de connaissance, mas aussi, de faire coopérer un ensemble d'agents et de coordonner leurs actions pour l'accomplissement d'un but commun.

Ce chapitre introduit tout d'abord, les notions d'agents et de systèmes multi-agents qui seront détaillé par la suite.

2. Agent :

Il existe en effet plusieurs définitions ou significations données à cette notion, le terme "agent" reste relativement vague, il n'y pas de consensus sur la définition.

2.1. Définition [CHA, 01] :

Un agent est un système informatique, situé dans un environnement, qui agit de façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu.

2.2. Caractéristiques des agents :

Un agent est une entité qui possède les caractéristiques suivantes :

- Autonomie : Un agent a un certain degré d'autonomie. Il peut prendre certaines décisions par rapport à ses états.
- Situé : Un agent est situé dans son environnement (physique ou virtuel). Il a une représentation de son environnement.
- Social : Un agent est capable d'interagir et de communiquer avec les autres agents. Il est capable de coopérer pour résoudre des problèmes ou effectuer des tâches.

- Proactif : un agent est capable de prendre d'initiative pour atteindre son but ou effectuer des tâches.
- Actif : un agent est toujours actif. Il s'exécute donc nécessairement dans un thread ou un processus indépendant.
- Capable de répondre à temps : l'agent doit être capable de percevoir son environnement et élaborer une réponse dans les temps requis;

2.3. Environnement [BEL, 11]:

Un agent ne peut exister sans environnement. L'environnement est une structure dans laquelle l'agent évolue. Un agent va agir sur son environnement et l'environnement va agir sur l'agent comme illustre la figure 1.

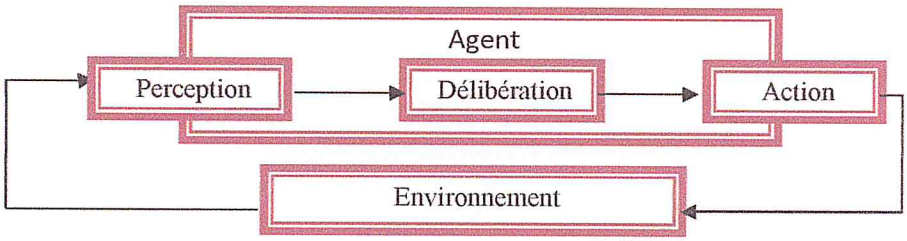


Figure 1 : Architecture d'un environnement.

2.4. Typologie des agents :

2.4.1 Agent réactif [ATT, 97] :

Agent réactif est un agent capable de réagir à des événements externes pour assurer soit la conduite et le pilotage d'un équipement matériel dont il est partie commande, soit la supervision et la coordination des activités d'un ensemble d'agents dont il est l'agent de contrôle.

2.4.2. Agent cognitif [ATT, 97] :

Agent cognitif est un agent qui ne contient pas uniquement des données et des méthodes procédurales mais aussi, une base de connaissances propres et sur celles des autres de même type.

Les agents cognitifs se regroupent en plusieurs sous-types d'agents définis de la façon suivante :

- **Agents intelligents [BEL, 11] :** combinent les trois caractéristiques (autonomie, coopération, adaptabilité) (figure 2) à leur plus haut niveau.

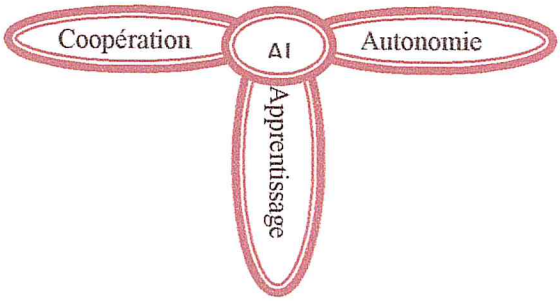


Figure 2 : Agent Intelligent (AI).

- Les **agents collaborant [NWA, 96]** : ce sont des agents cognitifs non apprenants. Ils sont donc à la fois fortement autonomes et coopérants.
- Les **agents interface [RHO, 00]:** ce sont des agents relativement autonomes qui utilisent différentes techniques d'apprentissage pour effectuer des tâches pour leur utilisateur. (Figure 3).

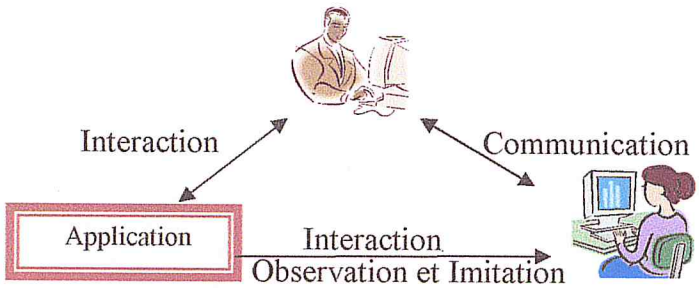


Figure 3 : Architecture d'un agent interface.

- Les **agents informations [RHO, 00]** sont dédiés à la recherche d'information, principalement sur l'Internet. Ces agents possèdent une grande autonomie.

Le tableau 1 résume les différentes propriétés des deux approches :

Système d'agents cognitifs	Système d'agents réactifs
Représentation explicite d'environnement	Pas de représentation explicite
L'agent peut tenir compte de son passé	Pas de mémoire de son historique
Agents complexes	Fonctionnement stimulus/réponse
Petit nombre d'agents	Grand nombre d'agent

Tableau 1 : les agents cognitifs et réactifs.

2.5. Architecture des agents :

Il existe plusieurs manières de concevoir des agents, mais peu importe l'architecture adoptée.

Nous présentons dans cette partie les différents modèles d'agents, afin de comprendre leur caractéristique et leur mode de fonctionnement. Nous distinguons deux grandes familles d'agents : les agents réactifs et agents cognitifs.

2.5.1. Agents réactifs [RUS, 95]:

Un agent réactif ne fait que réagir aux changements qui surviennent dans l'environnement. Autrement dit, un tel agent ne fait ni délibération ni planification, il se contente simplement d'acquérir des perceptions et de réagir à celles-ci en appliquant certaines règles prédéfinies. Étant donné qu'il n'y a pratiquement pas de raisonnement, ces agents peuvent agir et réagir très rapidement.

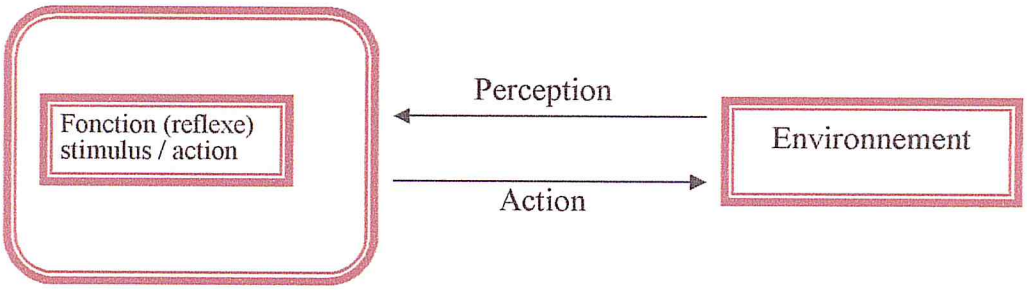


Figure 4 : modèle d'agent réactif ***[REA, 03].***

Dans la même catégorie que les agents réactifs, mais avec davantage de rationalité, se trouvent les agents hédoniques. Ces agents apprennent, par auto-renforcement, à modifier leur comportement afin d'augmenter leur "plaisir ou satisfaction". Ils sont capables d'anticipations "hédoniques" et d'adaptation lente à partir de leur expérience historique, ce qui suppose un niveau de rationalité plus élevé que l'agent purement réactif.

Les agents réactifs, traitent généralement des informations quantitatives tout en utilisant des calculs élémentaires ou d'optimisation. Ils peuvent être construits par des réseaux connexionnistes comme les Réseaux de Neurones (RN) ou en utilisant des simples algorithmes de calcul comme les Algorithmes Génétiques (AG). Leurs capacités répondent à la loi stimulus/action et apprentissage par auto-renforcement.

2.5.2. Agents cognitifs :

Les agents cognitifs, nommé aussi agents délibératifs, sont des agents qui effectuent une certaine délibération pour choisir leurs actions. En effet, les agents cognitifs sont capables à eux seuls de réaliser des opérations relativement complexes. Généralement, ils coopèrent les uns avec les autres pour atteindre un but commun (résolution d'un problème, une tâche complexe, etc.). Ils possèdent un ensemble de représentations explicites (sur l'environnement, sur les autres agents et sur eux-mêmes) décrits dans une base de connaissance sur laquelle ils peuvent raisonner. Ils réagissent en fonction de leurs connaissances, leurs buts, de leurs échanges d'informations avec les autres agents et de la perception de l'environnement. Ils sont dotés de moyens et mécanismes de communication pour gérer les interactions avec d'autres agents (coopération, coordination et négociation) **[REA, 03]**.

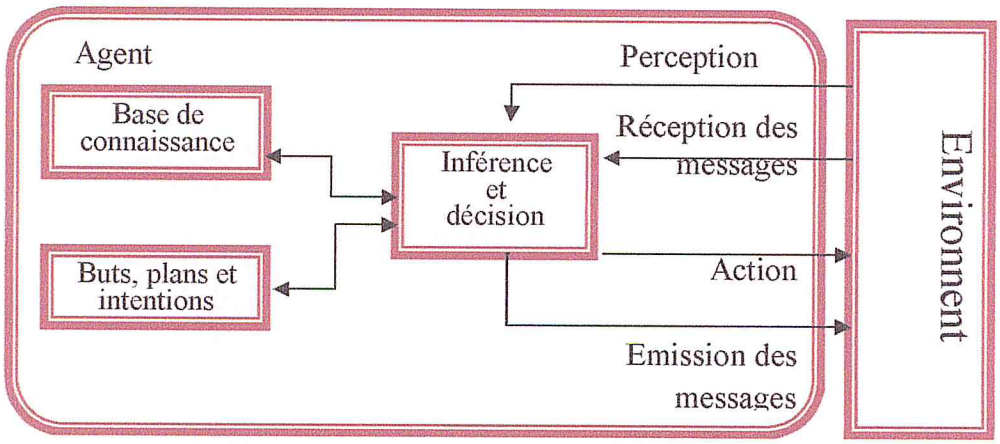


*Figure 5 : Modèle d'un agent cognitif **[REA, 03]**.*

2.5.3. Agents Hybrides :

L'architecture hybride est une architecture composée d'un ensemble de modules organisés dans une hiérarchie, chaque module étant soit une composante cognitive, soit une composante réactive **[SEC, 03]**. Dans l'architecture hybride on trouve généralement plusieurs couches logicielles, ces couches peuvent être arrangées verticalement ou horizontalement (figure 8).

La plupart des architectures considèrent que trois couches suffisent amplement **[CHA, 01]**. Ainsi, au plus bas niveau de l'architecture, on retrouve habituellement une couche purement réactive. La couche intermédiaire fait abstraction des données

brutes et travaille plutôt avec une vision qui se situe au niveau des connaissances de l'environnement. Finalement, la couche supérieure se charge des aspects sociaux de l'environnement, c'est à dire du raisonnement tenant compte des autres agents.

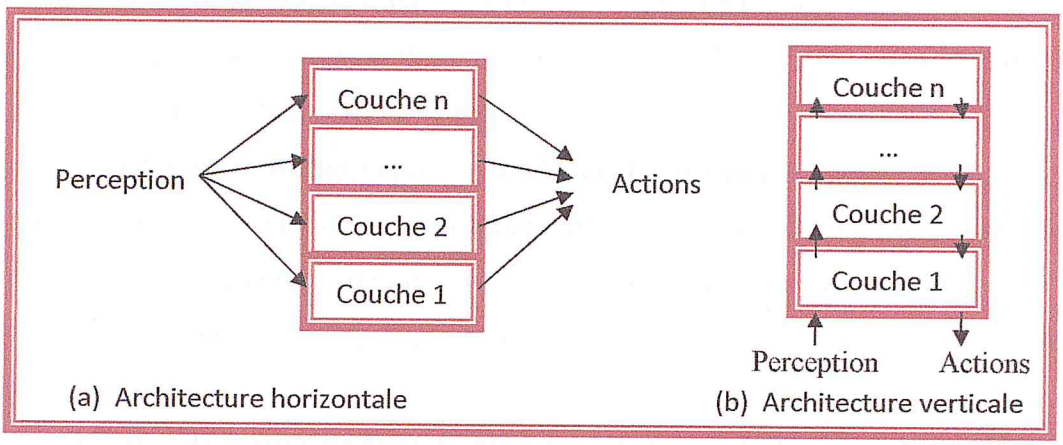


Figure 6 : architectures des agents en couche ***[JEN, 98b].***

2.6. Fonctionnement des agents :

Les agents sont immergés dans un environnement dans lequel et avec lequel ils interagissent. D'où leur structure autour de trois fonctions principales : percevoir, décider et agir. Parmi les sous-fonctions importantes d'un agent on peut citer : le raisonnement et l'apprentissage.

2.6.1. Le raisonnement des agents [BEL, 11] :

On dit d'un agent qu'il est un agent de raisonnement s'il intègre les éléments suivants :

- la base de faits : constitue l'état du programme; elle peut être vue comme la perception en temps réel, par l'agent, du monde à un instant T;
- la base de règles : est généralement la connaissance du travail fourni à l'agent par son concepteur ou par l'utilisateur;
- La structure de contrôle : qui est une structure procédurale du moteur raisonnement, et qui détermine, entre autres choses, quand réévaluer une règle donnée;
- les interfaces événement/action : qui sont des interfaces de programmation par lesquelles l'agent reçoit des événements et effectue des actions.

- Des opérateurs chargés de représenter l'application de ces opérations et la réaction de l'environnement envers les tentatives de modification.

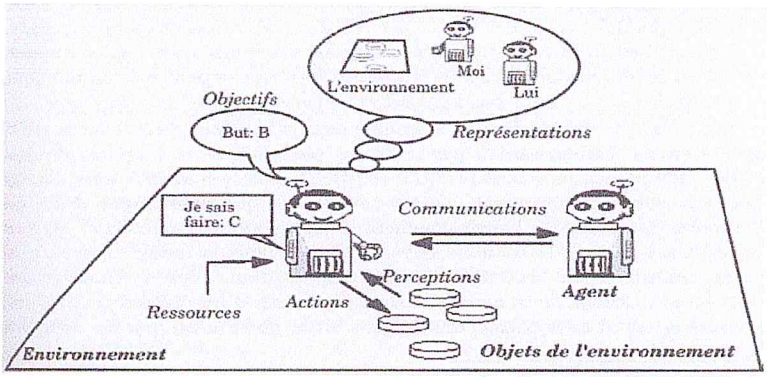


Figure 7 : Représentation imagée d'un agent en interaction avec son environnement et les autres agents ***[FER, 95].***

3.2. Caractéristiques d'un SMA [HUN, 06], [BEL, 11]:

Généralement, un SMA possède les caractéristiques suivantes :

- Chaque agent est une partie du système.
- Chaque agent travaille dans le but d'accomplir ses tâches.
- Chaque agent est capable de communiquer et d'interagir avec d'autres agents.
- Un agent coopère avec les autres agents lorsque nécessaire.
- Chaque agent a une vue partielle du SMA.
- Chaque agent a un restreint de capacité et de ressource, donc plusieurs agents doivent interagir pour résoudre le problème général.
- Il est distribué les ressources : chaque agent s'occupe d'une partie de travail et il possède aussi une partie de ressource du système.

3.3. Les type d'interactions dans les SMA :

Une interaction est une mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques **[FER, 95].**

3.3.1. La communication [BEL, 11] :

Elle permet de synchroniser les actions des agents et résoudre les conflits de ressources et de buts par la négociation. Dans les systèmes multi-agents, les agents ne disposent d'aucune mémoire commune. La communication entre les agents repose

explicitement sur des mécanismes d'envoi de message, de réception et de synchronisation.

3.3.1.1. Protocoles de communication [BEL, 11] :

Dans un système distribué, il faut que les entités disposent d'un langage commun pour pouvoir échanger des informations et coopérer pour la résolution d'un problème. Les primitives de base d'un protocole de communication sont les suivantes :

- établissement d'une connexion entre deux entités ;
- identification du nœud destinataire dans un réseau de communication ;
- envoi de données.

Contract Net [FIPA, 97]:

Le Contract Net est un protocole qui repose sur un mécanisme d'allocation de tâches régi par le protocole d'appel d'offres. Le modèle est basé sur une communication par envoi de messages. Dans les réseaux contractuels, un agent peut avoir deux rôles par rapport à une tâche : initiateur ou participant. Le protocole se compose de trois phases (voir figure 8):

- Annonce d'une tâche : l'initiateur fait un annonce de tâches (ou contrat) aux agents du système,
- Offre d'un service : les agents évaluent leur intérêt en fonction de leurs ressources. Si la tâche est intéressante ils soumettent une offre,
- Attribution d'une tâche : l'initiateur choisit un ou plusieurs agents candidats pour exécuter la tâche et informe les agents de ce choix.

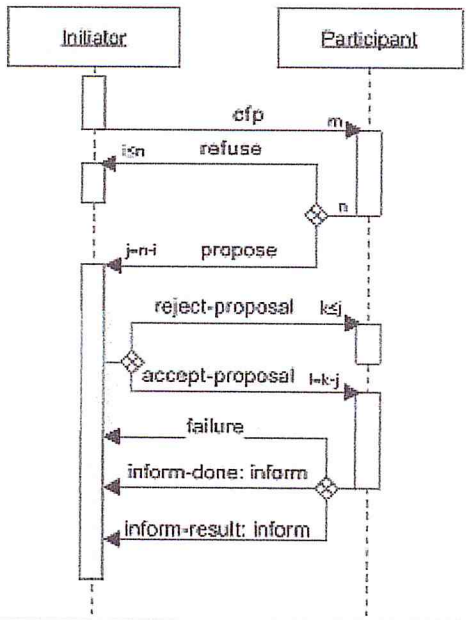


Figure 8 : Contract Net ***[FIPA, 97].***

3.3.1.2. Architecture de communication [BEL, 11] :

La communication entre les agents dans les SMA peut être organisée suivant trois schémas différents :

- **Réseaux en anneau** (figure 9)**:** Les interactions dans ce genre d'organisations sont trop lentes, un message destiné à un agent doit transiter par n-1 agents.

Figure 9 : réseau en anneau ***[BEL, 11].***

- **Réseaux en étoiles** (figure 10)**:** Ils présentent l'avantage de l'accès rapide entre le superviseur et les autres agents. La nature bidirectionnelle des connexions rend complexe la gestion des interactions inter-agents.

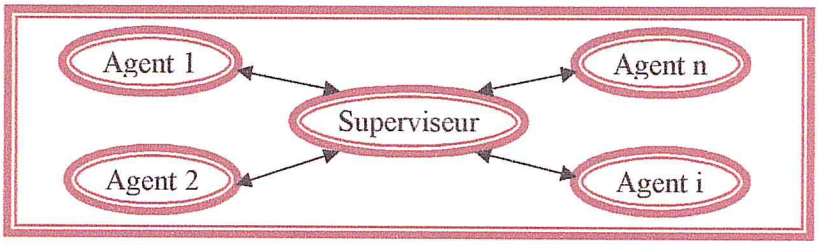


*Figure 10 : réseau en étoile **[BEL, 11]**.*

- **Réseaux hybrides** (figure 11)**:** Ce type d'organisation est une solution hybride reliant deux types d'organisation : un réseau en bus et un réseau en étoile.

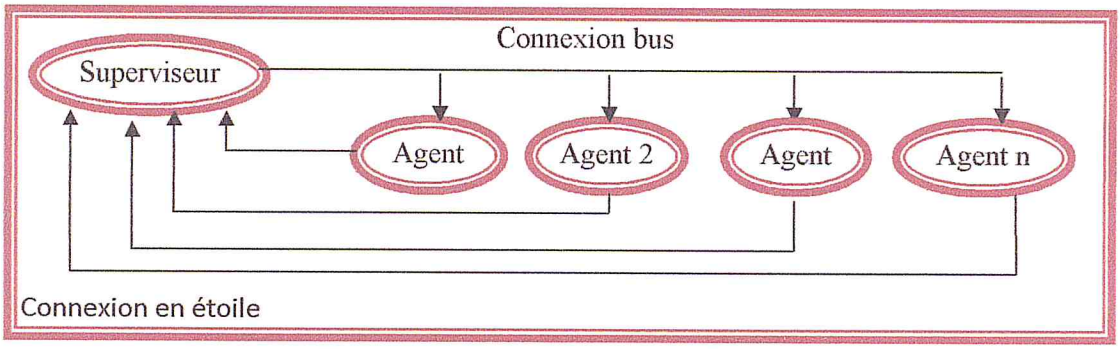


*Figure 11 : réseau hybride **[BEL, 11]**.*

3.3.1.3. Mode de communication [BEL, 11] :

Il existe deux principaux modes de communication :

- **Communication par envoi de messages** (figure 12)**:** Les agents sont en liaison directe et envoient leurs messages directement et explicitement au destinataire. La seule contrainte est la connaissance de l'agent destinataire.

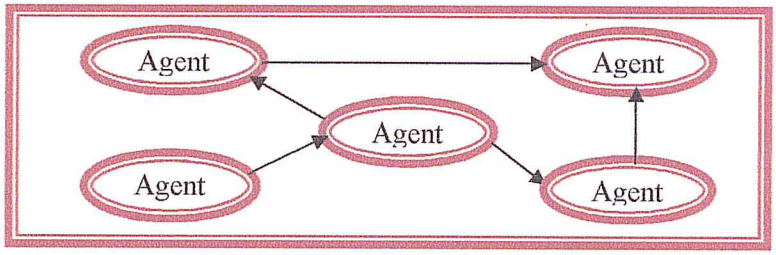


Figure 12 : Communication par envoi des messages.

- **Communication par partage d'informations** (figure 13)**:** Les composants ne sont pas en liaison directe mais communiquent via une structure de données

partagée, ou on trouve les connaissances relatives à la résolution (état courant du problème) qui évolue durant le processus d'exécution.

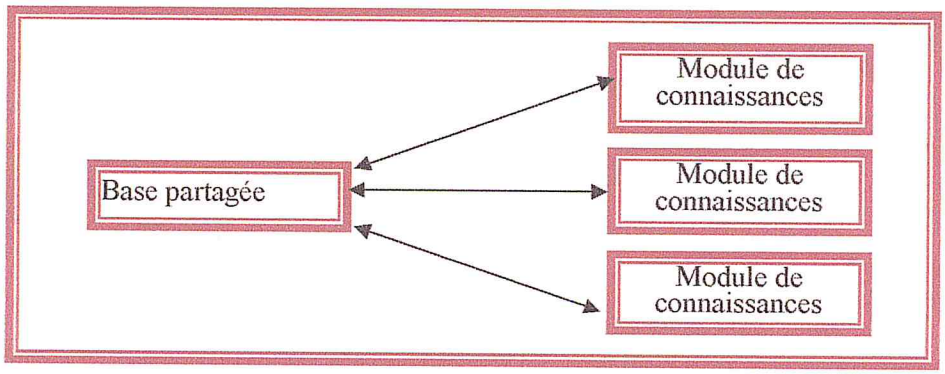


Figure 13 : Communication par partage d'information ***[BEL, 11].***

3.3.1.4. Langages de communication :

Les langages de communication et de représentation de l'information qui sont devenus des standards en SMA :

- **KQML** (Knowledge Query Manipulation Language) **[CHA, 01]:** ce langage définit un ensemble de type de messages (appelés " peformatifs ") et de règles qui définissent les comportements suggérés pour les agents qui reçoivent ces messages.

Le terme "performatif" a été utilisé pour nommer diverses commandes qui ont une certaine ressemblance avec des verbes utilisés de façon performative dans le langage naturel.

KQML détermine un format pour les messages et un protocole pour la réception et l'envoi de ces derniers. Le type de messages de KQML sont de natures diverses :

- Simples requêtes et assertions : ask, tell ;
- Instructions de routage de l'information : forward, broadcast ;
- Commandes persistantes : subscribe, monitor ;
- Commandes qui permettent aux agents consommateurs de demander à des agents intermédiaires de trouver les agents fournisseurs pertinents : advertise, recommend, recruit, broker.

- **FIPA ACL :** FIPA (Foundation For Intelligent Physical Agent) ACL (Agent Communication Language) **[FIPA, 97]**: Le langage (FIPA ACL) est un langage semblable à KQML auquel des protocoles d'interaction comme le *contractnet* et

autre protocole populaire ont été ajoutés. FIPA-ACL possède 21 actes communicatifs qui peuvent être groupés selon leurs fonctionnalités de la façon suivante :

- Passage de information: inform*, inform-if, inform-ref, confirm*, disconfirm*;
- Réquisition d'information : query-if, query-ref, subscribe;
- Négociation : accept-proposal, cfp, propose, reject-proposal;
- Distribution de tâches (ou exécution d'une action) : request*, request-when, request whenever, agree, cancel, refuse;
- Manipulation des erreurs : failure, not-understood.

Les actes communicatifs peuvent être :

- Les actes communicatifs primitifs, définis de façon élémentaire (mentionnés ci-dessus par une étoile '*') ;
- Les actes communicatifs composés, définis à partir d'autres actes par l'une des opérations suivantes :
  - un acte fait partie du contenu d'un autre acte; à travers l'opérateur de composition " ; " pour indiquer une séquence d'actions;
  - à travers l'opérateur de composition " | " pour indiquer un choix non déterministe de l'action.

3.3.2. La coopération :

La coopération peut être considérée comme une attitude adoptée par les agents qui décident de travailler ensemble. Durfee et ses collègues **[DUR, 90]** ont proposé quatre buts génériques pour établir la coopération dans ce groupe d'agents :

- Augmenter le taux de finalisation des tâches grâce au parallélisme,
- Augmenter le nombre de tâches réalisables grâce au partage de ressources (information, expertise, dispositifs physiques, etc.),
- Augmenter les chances de finaliser des tâches en les dupliquant et en utilisant éventuellement des modes de réalisation différents,
- Diminuer les interférences entre tâches en évitant les interactions négatives.

3.3.3. La négociation [BEL, 11]:

La technique de la négociation pour la résolution de conflits ne fixe aucun plan préalable, mais elle résout le problème du conflit au moment où il se pose. Elle considère qu'il est souvent plus facile de mettre en œuvre des mécanismes de résolution de conflits fondés sur des agents que de planifier l'ensemble des actions et leurs interactions.

3.3.3.1. Composantes de la négociation dans les SMA [BEL, 11] :

Pour modéliser la négociation dans un logiciel multi-agents, il faut prendre en compte les aspects suivants :

- Le langage de négociation est utilisé par les agents pour échanger des informations pendant la négociation; il est composé d'un ensemble de primitives de communication ;
- Le protocole de négociation est l'ensemble des règles concernant : les participants possibles dans la négociation, les propositions légales que les participants peuvent faire, les états de la négociation (par exemple l'état initial où commence la négociation) et une règle pour déterminer quand on est arrivé à un accord ou quand il faut s'arrêter parce qu'aucun accord n'a pu être trouvé ;
- L'objet de négociation représente le sujet autour duquel tourne la négociation ;
- Le processus de décision, c'est à dire le modèle que l'agent utilise pour prendre des décisions pendant la négociation. La partie la plus importante de la prise des décisions dans ce cas est la stratégie de négociation qui permet de déterminer quelle primitive de négociation l'agent doit choisir à un certain moment ;
- Les participants sont les agents qui prennent parti dans la négociation.

3.4. Modélisation des interactions [BEL, 11]:

Les interactions entre agents sont souvent modélisées avec des approches logiques, mais aussi avec des outils comme des graphes de transition d'automates à états finis. Les figures 14 et 15 représentent respectivement un exemple d'interaction modélisé par un automate et à l'aide d'un réseau de Pétri. Dans cet exemple, un état de l'automate représente l'état d'un système à deux agents A et B et une transition

représente une interaction. Avec les réseaux de Pétri, les places du réseau représentent les états internes de l'agent et les messages en cours d'acheminement, alors que les transitions représentent des synchronisations sur les messages et des conditions d'application d'actions.

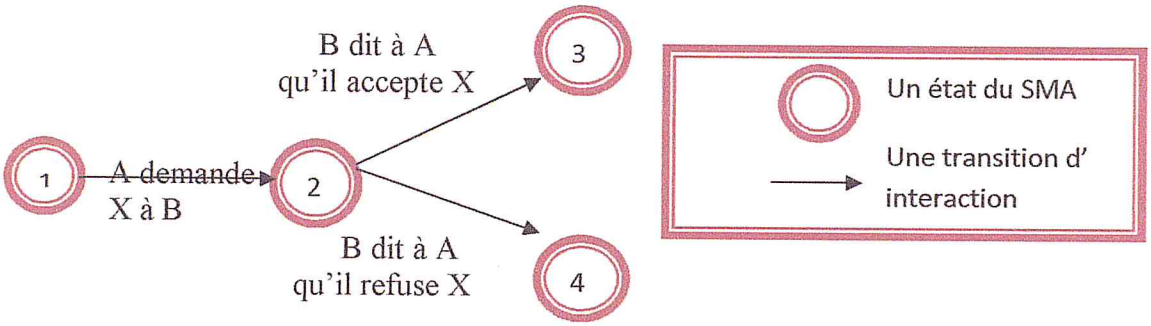


Figure 14 : un exemple d'automate à états modélisant une interaction ***[BEL, 11].***

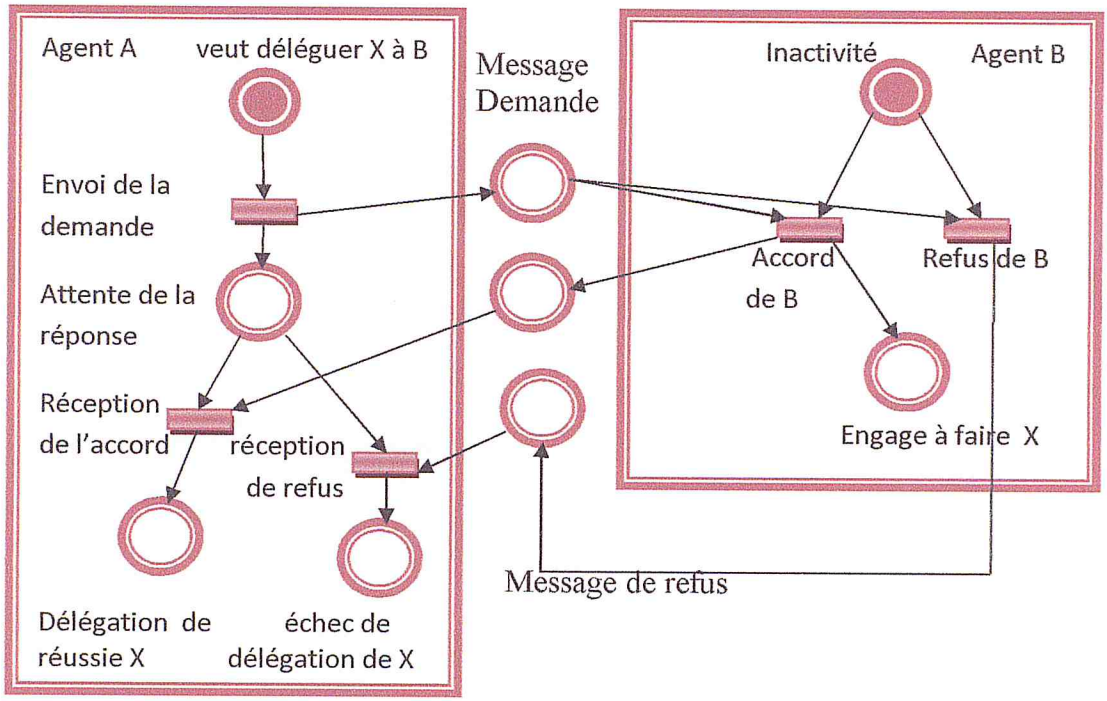


*Figure 15 : Un exemple de réseau de Pétri modélisant la même interaction que la figure 14****[BEL, 11].***

Modéliser les interactions permet d'une part à un observateur de comprendre la façon dont elles se déroulent et d'autre part de concevoir le comportement d'un agent qui respecte, par exemple, un protocole de communication.

3.5. Les situations d'interaction :

La façon dont se réalisent les interactions permet d'obtenir diverses situations au niveau du système multi-agents. Le Tableau 2 montre la classification des situations d'interaction donnée par Ferber **[FER, 95]** en fonction de la compatibilité des buts des intervenants, de la disponibilité des ressources et de la capacité des agents à atteindre leur but en termes de compétences individuelles.

Buts	Ressources	Compétences	Type de situation	Catégorie
Compatibles	Suffisantes	Suffisantes	Indépendance	Indifférence
Compatibles	Suffisantes	Insuffisantes	Collaboration simple	Coopération
Compatibles	Insuffisantes	Suffisantes	Encombrement	
Compatibles	Insuffisantes	Insuffisantes	Collaboration coordonnée	
Incompatibles	Suffisantes	Suffisantes	Compétition individuel pure	Antagonisme
Incompatibles	Suffisantes	Insuffisantes	Compétition collective pure	
Incompatibles	Insuffisantes	Suffisantes	Conflits individuels pour des ressources	
Incompatibles	Insuffisantes	Insuffisantes	Conflits collectives pour des ressources	

*Tableau 2 : Classification des situations d'interaction **[FER, 95]**.*

3.6. L'organisation dans les SMA [BEL, 11]:

L'organisation sociale d'un SMA est la manière dont le groupe est constitué, à un moment donné, pour pouvoir fonctionner. Elle décrit l'ensemble des composants fonctionnels du système, leur nature, leur responsabilité et leur besoin en ressources ainsi que les liens de communications entre agents. Il existe plusieurs modèles d'organisation, nous citons parmi eux :

- l'organisation à membre unique : où le seul agent présent dans l'organisation accomplit toutes les tâches;

- l'organisation en groupe simple: dès qu'un groupe existe, on peut avoir une coordination ou une coopération entre les agents afin d'atteindre un but commun;

- l'organisation hiérarchique simple : lorsqu'un agent ne peut pas réaliser une tâche complexe, cette dernière est divisée en sous tâches qui sont ensuite distribuées à un ensemble d'agents.

3.7. Les méthodes de conception des SMA [HUN, 06]:

Il y a plusieurs recherches qui s'intéressent à la méthodologie d'analyse et de conception orientées agent. Maintenant, il n'y a pas encore une méthode standard car chaque groupe de recherche propose une méthode différente. Chaque méthode convient à un groupe particulier des applications.

Cette section va présenter, en point de vue de la modélisation système multi agent, quelques méthodes typiques qui sont nées dans les dernières années :

3.7.1. La méthode MAS-CommonKADS [HUN, 06]:

MAS-CommonKADS (Multiagent System – Knowledge Analysis and Development System) est étendu de la méthodologie de génie connaissance en prenant les techniques orientées objet et celles de la méthodologie de génie protocole. Cette méthode modélise un système par des étapes suivantes :

- Modélisation des agents : crée les instances originales pour identifier et décrire les agents en utilisant le modèle d'agent. Modèle d'agent détermine les caractéristiques de l'agent : Les capacités de raisonnement, les comportements (percevoir/s'agir), les services, les groupes et hiérarchie des agents.
- Modélisation des tâches : découvrit et décrit toutes les tâches possibles dans le système en utilisant le modèle des tâches. Modèle des tâches déterminent les tâches dont l'agent peut s'occuper : les buts, les partitions des composants et les méthodes de résoudre le problème qui lie à la tâche.
- Modélisation des coordinations : développe le modèle des coordinations. Modèle des coordinations déterminent les conversations entre les agents : les interactions, les protocoles et les capacités nécessaires.
- Modélisation de connaissance : modélise la connaissance du domaine en utilisant le modèle d'expert. Modèle d'expert détermine les connaissances dont les agents ont besoin pour obtenir leurs objectifs.
- Modélisation d'organisation : développe le modèle d'organisation en basant sur le modèle d'agent. Ce modèle présente les relations statiques ou structurées entre les agents.

Cette méthodologie intègre la méthodologie de génie connaissance et les techniques de la méthodologie orientée objet et génie protocole. Elle propose un

nouveau modèle, c'est modèle de coordination, qui peut représenter les relations dynamiques entre les agents ou entre les agents et les gens dans le système, et donc, elle nous aide à concevoir le système plus dynamique qu'autrefois.

3.7.2. La méthode Gaia [HUN, 06]:

Gaia exploite l'abstraction organisationnelle pour fournir une méthode d'analyse et conception le système de logiciel ouvert et complexe. Elle contient des modèles :

- Modèle d'organisation : divise le système par rapport plusieurs sous-systèmes. Soit selon l'identification des sous-systèmes qui existent déjà dans le système, soit selon la structuré des composants. Des composants appartiennent au même sous-système quand : soit ils ont les objectifs communs ; soit ils interagissent avec une haute fréquence ; soit leurs capacités sont proches et chaque composant a des capacités dont les autres ont vraiment besoin.
- Modèle d'environnement : considère l'environnement en termes de ressources calculées abstraites. Il est considéré comme une liste des ressources. Chaque ressource est associée avec un nom caractérisé par l'action que l'agent peut s'y agir.
- Modèle préliminaire de rôle : ce n'est pas encore l'organisation actuelle. Il est une définition préliminaire des rôles et des protocoles de l'organisation. Gaia propose deux termes pour représenter de la façon demi-formelle le rôle. *Permission*, définit la relation de l'agent avec son environnement si l'agent a le droit d'accéder les ressources, de les changer ou de les consommer. *Responsabilité*, détermine les comportements d'un agent selon deux types : la propriété vivant qui décrit les états auxquels un agent doit arriver sur quelques conditions, et propriété sécurité qui assure qu'un agent vient aux états acceptables. Puis, on crée un ensemble des schémas de rôle, un rôle en a une correspondante.
- Modèle préliminaire d'interaction : capte les indépendances et les relations entre les rôles dans le système, en termes d'une définition protocole pour un type de l'interaction entre eux. Un protocole est défini préliminairement par les attributs : le nom, l'initiateur, le partenaire, les entrées, les sorties et la description.
- Modèle de règle organisationnelle : la règle organisationnelle est considérée comme la responsabilité de l'organisation. Elle contient deux types : la règle vivante concernant l'invariant que l'organisation doit respecter, la règle sécurité concernant

l'express dynamique de l'organisation. La règle vivante assure l'ordre de réalisation des rôles ou des protocoles, d'un après d'autre. La règle sécurité assure qu'un rôle est joué par au moins d'un agent et qu'un agent peut jouer au plus un rôle à la fois.

Gaia se base fortement sur le rôle dans le système. Elle propose aussi des règles d'environnement et d'organisation. Cependant, elle manque la façon de construire l'ontologie du système, qui supporte l'interaction entre les agents.

3.7.3. AMAS/Adelfe [HUN, 06]:

De façon similaire à MESSAGE/UML, la méthode ADELFE (Atelier pour le Développement de Logiciels à Fonctionnalité Émergente) est une extension d'UML qui tente de prendre en compte les notions associées au paradigme multi-agent. La méthode ADELFE a en fait été développée afin de fournir un outil applicatif de la théorie des AMAS (Adaptive Multi-Agent System).

La méthode ADELFE comprend les phases suivantes :

- Une phase d'analyse : Dans cette phase on vérifie la pertinence d'une approche multi-agent. Cela entend l'étude de critères tels que :
  - complexité grande,
  - absence d'autres méthodes,
  - frontières mal définies,
  - distribution physique ou fonctionnelle,
  - environnement évolutif,
  - besoin d'interactions-coopération entre entités.
  - Puis, la suite de l'analyse est similaire à celle de UML;

- **Une phase de conception** : Pour la conception d'un agent, ADELFE propose une description d'agent générique composée de sept modules :
  - communication avec les autres agents,
  - communication avec l'environnement,
  - croyances sur lui-même,
  - croyances sur les autres agents,
  - croyances sur son environnement,

- compétences,
- attitude sociale coopérative.

3.8. Les Plateformes de développement des SMA :

La notion de plate-forme est liée à l'implémentation des systèmes multi-agents : elle correspond à l'environnement qui permet de gérer le cycle de vie des agents et dans lequel les agents ont accès à certains services. Nous en présentons dans ce qui suit trois dont le choix a dépend des méthodologies de conception précédemment citées.

3.8.1. Agent Tool [QUI, 02]:

Outil et méthodologie qui mettent l'accent sur les premières phases du développement (analyse et développement).

- Méthodologie MaSE : extension au modèle OO (7 phases)
- Trouver les buts, appliquer les cas d'utilisation, raffiner les rôles, créer les classes d'agents, construire les conversations, assembler les classes d'agents et l'implémentation.
- L'outil permet la validation des conversations.
- Génération du code des conversations (en Java).

3.8.2. Jade [QUI, 02]:

- Outil qui répond aux normes FIPA.
- Trois modules principaux (nécessaires aux normes FIPA):
- Le DF (directory facilitator) fourni un service de pages jaunes à la plate-forme.
- L'ACC (agent communication chanel) gère la communication entre les agents.
- L'AMS (agent management system) supervise l'enregistrement des agents, leur authentification, leur accès et utilisation du système.
- Langage de communication entre les agents : FIPA ACL.
- Éditeur pour l'enregistrement et la gestion des agents.

3.8.3. Madkit [QUI, 02] :

- Environnement basé sur la méthodologie Aalaadin ou AGR (agent, groupe, rôle).
- Un agent joue des rôles à l'intérieur de groupes.
- L'outil fourni un éditeur permettant le déploiement et la gestion des SMA (G-box).
- L'outil offre aussi un utilitaire pour effectuer des simulations à grande échelle.

3.8.4. Zeus [QUI, 02] :

- Environnement de développement complet ;
- Méthodologie (role modeling) ;
- Les agents possèdent trois couches :
  - La définition : l'agent est vu comme une entité autonome capable de raisonner grâce à ses croyances, ressources et préférences.
  - L'organisation : relations entre les agents.
  - La coordination : modes de communication entre les agents, protocoles, coordination et autres mécanismes d'interaction.

Conclusion :

Les agents peuvent être utilisés comme modules d'encapsulation selon deux approches : l'approche de décomposition fonctionnelle et l'approche de décomposition physique. Dans la première approche les agents sont assignés ou affectés à des fonctions dans le système (la planification, l'ordonnancement, etc.). Tandis que dans la seconde approche les agents sont utilisés pour représenter des entités physiques (les travailleurs, les ressources, les produits, les opérations, etc.). Pour cela, l'ordonnancement du système peut être réalisé selon deux types.

Dans le premier cas, les agents sont responsables de l'ordonnancement des ordres de fabrication. Cette approche est liée aux savoir-faire en ordonnancement, l'ordonnancement global du système est obtenu à travers l'émergence d'ordonnancements locaux.

Le second cas est lié au savoir-faire ayant trait à l'ordonnancement local des ressources du système opérant (machines, cellules, etc.). Un agent représente une ressource ; il est responsable de son propre ordonnancement. Pour ce faire, il utilise des mécanismes de négociation avec les autres agents dans le système.

Finalement, nous pouvons mentionner un autre type d'ordonnancement local, ou plutôt d'allocation dynamique des ressources locales, pour le pilotage décentralisé et auto-organisé d'un système de production. Dans ce type l'ordonnancement est effectué en temps réel, totalement distribué et organisé au niveau du produit à fabriquer. En effet, l'allocation dynamique des ressources est effectuée par un agent représentant le produit physique dans le système. Il est responsable de l'accomplissement de l'ensemble de ses tâches sur des ressources appropriées durant son cycle de vie.

CHAP 3
Modélisation de la solution

1. Introduction :

La réalisation d'un projet passe souvent par plusieurs étapes dès l'analyse à la réalisation. Il faut bien comprendre le sujet sur lequel on travail pour meilleur choisir les instruments à utiliser.

Après une documentation exhaustive on a pu se situer par rapport à notre sujet mais les idées restaient encore vagues puisque on ne savait pas comment l'entreprise menait la production, ce qui nous a enduits à effectuer plusieurs visites aux ateliers de production et à discuter avec plusieurs personnes spécialistes dans le domaine.

Après quelques visites on était capables de remarquer que pour la fabrication d'un certains produit il fallait passer par plusieurs étapes, dès le débitage à la peinture et par conséquent par plusieurs machines et plusieurs ateliers.

L'entreprise fabrique des produits de natures variées et son cheminement varie selon la nature et la complexité, ce qui nous a conduits à déduire qu'on était devant un problème de type job-shop, comme on peut le voir dans l'exemple de la figure 1.

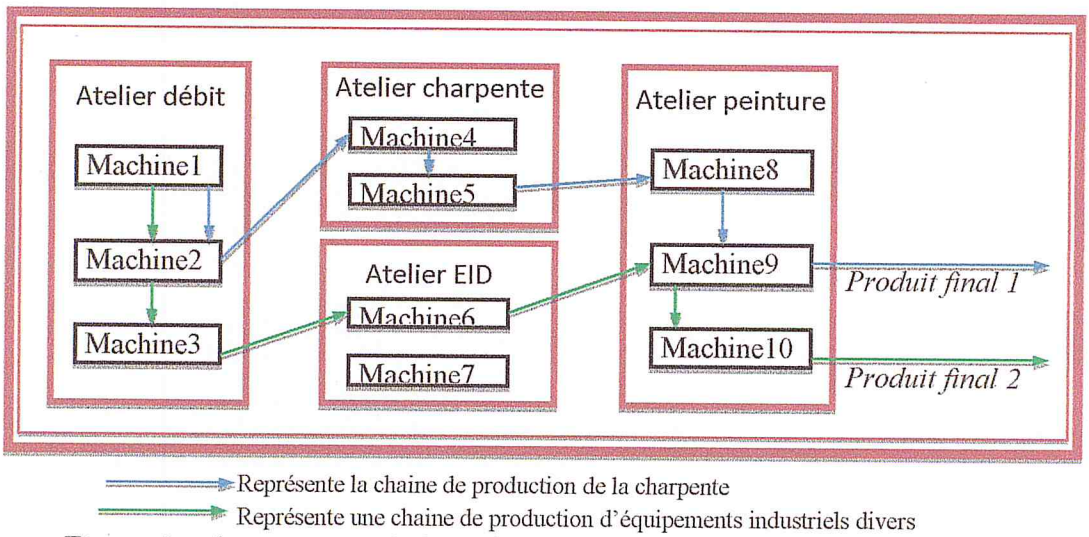


Figure 1 : cheminement de la production de produits de natures différentes.

La figure 1 donne l'illustration du cheminement de la production montrant qu'on peut avoir des chemins différents pour des produits de natures variées ce que nous donne un problème du type job-shop qui sera modélisé dans la suite de ce chapitre.

2. Job-shop :

Dans ce chapitre, nous nous intéressons au problème du job-shop. En appliquant une démarche de modélisation au problème d'ordonnancement des systèmes flexibles de production, le problème de job-shop est central en ordonnancement car c'est un problème d'atelier général et qu'il est largement étudié dans la littérature. Ainsi, de nombreuses méthodes efficaces ont été proposées pour sa résolution. Nous proposons donc d'étendre les méthodes existantes pour pouvoir minimiser le temps production (*makespan*).

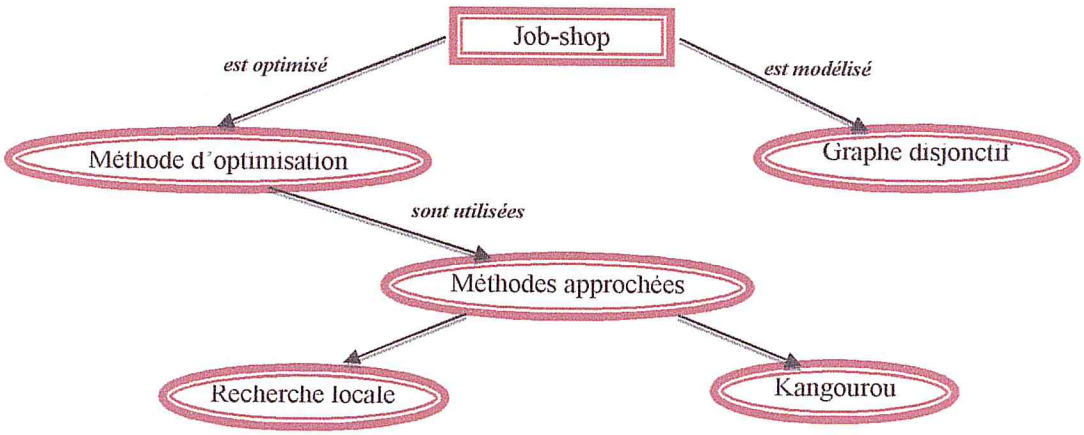


Figure 2 : Synoptique de nos propositions pour le job-shop.

La figure 2 représente nos propositions pour le problème de job-shop. Dans la suite de ce chapitre on donne une illustration de nos propositions pour résoudre le problème du job-shop, tels que le graphe disjonctif, le chemin critique. Nous introduisons aussi un schéma de résolution basé sur l'utilisation des heuristiques et des méta-heuristiques.

2.1. Définition :

On considère $\wp$ un problème de job-shop. On dispose de m machines pour réaliser le traitement de n jobs. Le job-shop consiste à ordonnancer l'ensemble des n jobs sur l'ensemble des m machines. Chaque job J_i est composé de n_i opérations notées $O_{i,1}$, $O_{i,2}$,…, O_{i,n_i} , à réaliser dans un ordre donné (contraintes de précédence), appelé la gamme du job : $O_{i,1} \rightarrow O_{i,2} \rightarrow \ldots \rightarrow O_{i,n_i}$. Chaque opération $O_{i,j}$ du job J_i doit s'exécuter sur la machine $\mu_{i,j}$ sans préemption pendant une durée

$p_{i,j} > 0$. Chaque machine ne peut exécuter qu'une seule opération à la fois. On considère que, chaque machine dispose de stocks (buffers) illimités d'entrée/sortie.

Ainsi, chaque job J_i a un instant donné peut être soit :

- traité par une machine ;
- dans le buffer d'entrée d'une machine ;

Les jobs déposés sur les machines doivent attendre dans les buffers d'entrée la disponibilité de la machine pour être traités. On note que l'ordre d'arrivée des jobs peut différer de l'ordre d'exécution par les machines. Toutes les variables $p_{i,j}$, $C_{max} = \max_{j=1,\ldots,n} \{C_j\}$, $C_j, j = 1 \ldots n$ étant la date fin de la dernière opération, sont supposées être des variables entières et non négatives.

D'après la notation $\alpha / \beta / \gamma$ le problème étudié peut être noté J | |Cmax. Dans cette notation :

- J est utilisé pour spécifier qu'il s'agit d'un problème de job-shop,
- le critère à optimiser est la minimisation du temps de fin de la dernière opération dans le système appelé makespan (C_{max}).

2.2. Variables :

c_i : date de fin d'exécution de l'opération $i \in O$;

t_i : date de début de l'opération $i \in O$;

$$b_{i,j} = \begin{cases} 1 \text{ si l'opération } i \text{ est exécuté avant l'opération } j \text{ sur la même machine } m \\ 0 \quad \text{sinon} \end{cases}$$

2.3. Contraintes :

On recense deux ensembles de contraintes : les contraintes de précédence entre les opérations (1), les contraintes de disjonction machine (Contraintes (2 – 4)) :

- **Date de fin des opérations** : ces contraintes garantissent que la date de fin c_i de l'opération i soit supérieure ou égale à sa date de début t_i plus sa durée opératoire p_i.

$$c_i \geq t_i + p_i \qquad \forall i \in O \quad (1) \quad \textbf{[LAR, 10]}$$

- **Les contraintes de disjonctions sur les machines :** cette contrainte impose que chaque machine ne traite qu'une opération à la fois. Etant donné deux opérations

$i \in O$ et $j \in O$ traitées consécutivement sur une même machine $\mu_i = \mu_j$, les contraintes sont données par les équations (2-4). Les contraintes (2) et (3) sont arbitrées par la variable binaire $b_{i,j} \in \{0,1\}$ qui rend active une seule contrainte à la fois correspondant à un seul ordre d'exécution. Dans ce cas, la contrainte (2), peut être écrite en $t_i \geq c_j$, ce qui signifie que l'opération i ne peut commencer son traitement que si la l'opération j est terminé le sien. Par contre si $b_{i,j} = 1$, c'est la contrainte (3) qui est active impliquant que l'opération j ne commence qu'une fois l'opération i terminé. La contrainte (4) garanti que soit l'opération $i \in O$ est lieu en premier ou soit l'opération $j \in O$ est lieu en premier.

$$t_i \geq c_j - H.b_{i,j} \qquad \forall (i,j) \in O / \mu_i = \mu_j \qquad (2)\ \textbf{[LAR, 10]}$$

$$t_j \geq c_i - H.(1 - b_{i,j}) \qquad \forall (i,j) \in O / \mu_i = \mu_j \qquad (3)\ \textbf{[LAR, 10]}$$

$$b_{i,j} + b_{j,i} = 1 \qquad \forall (i,j) \in O / \mu_i = \mu_j \qquad (4)\ \textbf{[LAR, 10]}$$

2.4. Fonction objectif :

Le critère à minimiser est la date de fin de la dernière opération. "Makespan" donnée par la variable $C_{max} = \max(c_i)$, $\forall i \in U$.

$$Min\ C_{max} \qquad \textbf{[LAR, 10]}$$

$$C_{max} \geq c_i \qquad \forall i \in U$$

Toutes les variables sont supposées être positives ou nulles. Cette formalisation génère un grand nombre de variables binaires qui rendent difficile la résolution dans des temps raisonnables. Cependant, des méthodes approchées donnant des solutions de bonne qualité sont proposés pour la résolution de ce problème dans la suite du chapitre.

3. Modélisation du job-shop :

Pour résoudre le problème de job-shop, les méthodes exactes ne constituent pas, du moins pour l'instant le meilleur outil, c'est pourquoi que nous avons orienté nos efforts vers les méthodes approchées. Elles s'appuient sur une modélisation sous la forme de graphe conjonctif-disjonctif du problème.

Soit l'exemple suivant (JS3), donné par les tableaux 1 et 2 que sera utilisé pour

donner une illustration sous forme d'un graphe disjonctif-conjonctif.

	Op1	Op2	Op3
Job1	(M3, 6)	(M1, 4)	(M2 ,2)
Job2	(M3, 6)	(M1, 2)	(M2, 6)
Job3	(M3, 4)	(M1, 5)	(M2, 4)

Tableau 1 : Temps opératoires.

Machine	Opération	Opération	Opération
M1	Op 2, Job 1	Op 2, Job 2	Op 2, Job 3
M2	Op 3, Job 1	Op 3, Job 2	Op 3, Job 3
M3	Op 1, Job 1	Op 1, Job 2	Op 1, Job 3

Tableau 2 : Ordre d'exécution des opérations.

3.1. Modélisation sous forme d'un graphe conjonctif-disjonctif :

Dans cette modélisation le problème du job-shop est représenté par un graphe $G = (V_m, C \cup D_m)$ où :

- V_m est l'ensemble de sommets représentant les opérations plus deux nœuds fictifs {0,*} tel que {0} est le prédécesseur de toutes les opérations du problème, sa date de début est donc (0) et {*} est le successeur de toutes les dernières opérations de tous les jobs et sa date de début sera le makespan;
- C représente l'ensemble des arêtes conjonctives à l'intérieur du même job. Les arrêtes sont de type : $O_{i,k} \rightarrow O_{i,k+1}$ plus deux arrêtes reliant les opérations de début et de fin des jobs aux nœuds fictifs {0} et {*} i.e. : $\forall j \in J$ on ajoute les deux arêtes $0 \rightarrow O_{j,1}$ et $O_{j,nj} \rightarrow *$;
- D_m est l'ensemble contenant les arrêtes représentant les disjonctions machine.

S'aidant de l'exemple (JS3), on donnera une illustration du graphe disjonctif associé. Soit P un problème de job-shop: les temps opératoires sont donnés par le tableau 1.

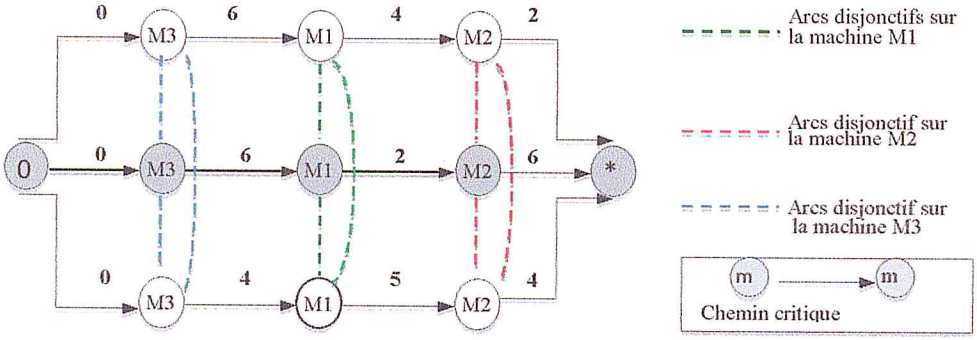


Figure 3: Le graphe disjonctif non orienté du problème JS3.

La figure 4 donne une illustration d'un graphe disjonctif-conjonctif représentant le problème de job-shop. Cela permet d'obtenir un graphe disjonctif dans lequel on fait apparaître que les opérations affectées à la même machine sont en disjonction.

Une solution du problème de JS consiste à choisir une solution au problème de disjonction pour chaque machine. Par exemple, on peut utiliser l'ordre d'exécution donnée par le tableau 2. Le graphe disjonctif est alors complètement orienté et représente une solution du problème dont on peut calculer le makespan par une simple application d'un algorithme de plus long chemin.

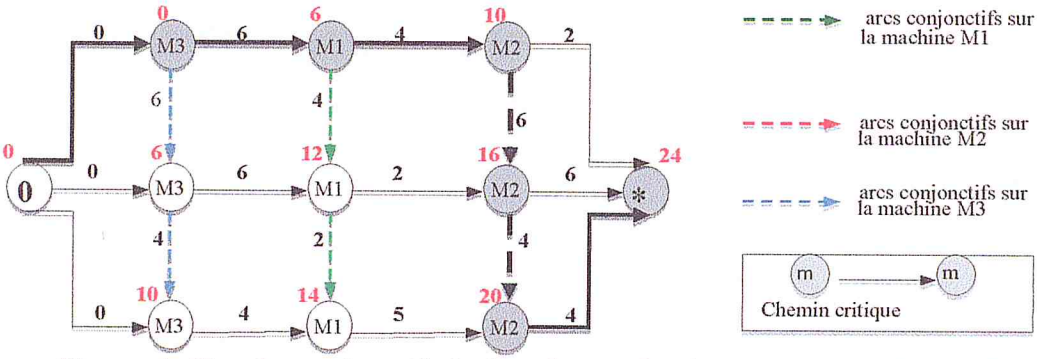


Figure 4: Graphe conjonctif résultat du graphe disjonctif du problème JS3.

Ce graphe constitue la première étape pour une résolution utilisant des méthodes approchées, il offre un moyen visuel représentant les différentes contraintes du problème. Basés sur le graphe ci-dessus, nous proposons d'utiliser le principe de vecteur à valeurs dupliquées proposé par **[BIE, 95]**. Il s'agit d'une suite ordonnée de numéro de jobs. La $j^{ème}$ occurrence du nombre i dans le vecteur par répétition désigne la $j^{ème}$ opération du job i. L'avantage de cette représentation est de ne représenter que des ordres valides d'opérations. Pour le job-shop, ces vecteurs par répétition permettent d'obtenir uniquement des graphes acycliques.

4. Représentation R du problème du job-shop :

Plusieurs représentations existent pour les problèmes d'ordonnancement. Ces dernières constituent des codages adoptés pour représenter des solutions d'un problème d'ordonnancement. On a choisi de représenter une solution du problème du job-shop par l'intermédiaire d'un vecteur. Dans cette représentation nous utilisons un vecteur par répétition noté *SM*.

SM constitue une sélection machine. Il indique un ordre de passage des jobs sur les différentes machines. Ce vecteur contient autant d'éléments que d'opérations sur l'ensemble du problème. Chaque job *j* est représenté par un numéro apparaissant n_j fois, où n_j est le nombre d'opérations du job j.

Ainsi, le vecteur *SM* permet de décrire un ordre d'opérations, cet ordre est utilisé pour orienter le graphe disjonctif. Pour l'exemple JS3 donné par le Tableau 1 et le tableau 2, et la Figure 5 donnent une illustration de cette représentation.

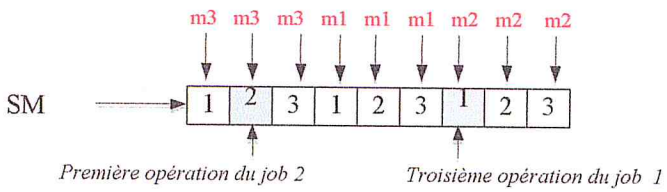


Figure 5: Représentation par vecteur SM du graphe de la figure 4.

5. Heuristique de construction de solution :

Pour une résolution approchée du problème, un effort important a été réalisé pour le modéliser d'abord par le graphe disjonctif puis par des vecteurs avec répétitions. Nous proposons l'adaptation de l'heuristique H2 utilisée dans **[LAR, 10]** pour le problème de job-shop avec transport. Le but de celle présentée dans cette partie est de permettre la construction de solutions sans cycles. S'aidant de l'exemple JS3, on présente une heuristique de construction de solutions complètes.
L'heuristique est appelée HJS.

5.1. Heuristique HJS :

L'heuristique HJS est une heuristique de construction, elle fournit une solution complète à un problème. A chaque itération de l'heuristique, se fait un choix d'une opération de l'ensemble *U* (ensemble des opérations non ordonnancées) ne possédant

pas de prédécesseurs dans U, ensuite on soustrait l'opération de l'ensemble U pour la mettre dans l'ensemble S contenant les opérations ordonnancées. L'algorithme 1 donne le pseudo code du fonctionnement général de cette heuristique. Chaque itération consiste à ordonnancer une nouvelle opération choisie par une règle de priorité. Les procédures *Queue, Elire et Update* sont appelées.

La sélection va se faire comme suit : une opération qui n'a pas de prédécesseur dans U est choisie à chaque itération. Si plusieurs choix existent, on fait intervenir le calcul de la queue *Queue* qui trie les jobs dans un ordre donné.

Pour effectuer le tri on applique la règle suivante: si plusieurs choix existent on choisit l'opération ayant la plus grande somme des processing times (appelée *Queue*) des opérations non encore ordonnancées.

Algorithme 1: *Génération du vecteur SM par l'heuristique HJS*

Nom de procédure générer_SM
Données : donnée du problème (gammes, temps opératoires)
Sortie
SM : sélection machine
Début
U := O *// ensemble de toutes les opérations non ordonnées*
S := ∅ *// ensemble de toutes les opérations ordonnées*
Tant que *U ≠∅* **faire Pour** *I ∈J* **faire**
Calculer *Queue[i]* pour job *i* ***Fin pour*** *// calculer la priorité du job i*
Pour i:=1 à n faire
j := *Elire(U, J)* // élire un job j
S := *Update(S, {Oj,sj })* *// mettre à jour l'ensemble S avec l'opération*
U := U- {Oj,sj } *// mettre à jour l'ensemble U en soustrayant l'opération*
Fin pour
Fin Tant que
Fin

S'aidant de l'exemple JS3 on donnera une illustration du fonctionnement de l'heuristique HJS. Soit la figure 6 dans laquelle le vecteur *SM* est vide au début:

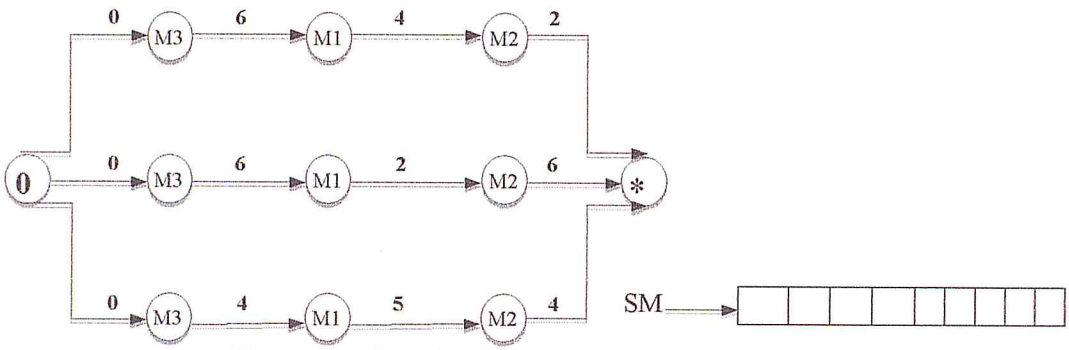


Figure 6: Graphe de départ, le vecteur SM est vide.

Au départ l'ensemble $U=\{O_{1,1},O_{1,2},O_{1,3},O_{2,1},O_{2,2},O_{2,3},O_{3,1},O_{3,2},O_{3,3}\}$ et

l'ensemble S=Φ. Les seules opérations ordonnables sont : l'opération 1 du job 1, l'opération 1 du job 2 et l'opération 1 du job 3. Nous avons *Queue (1)=12, Queue (2)=14 et Queue (3)=13*. La règle consiste à choisir l'opération appartenant au job ayant donné la queue maximal i.e. l'opération 1 du job 2.

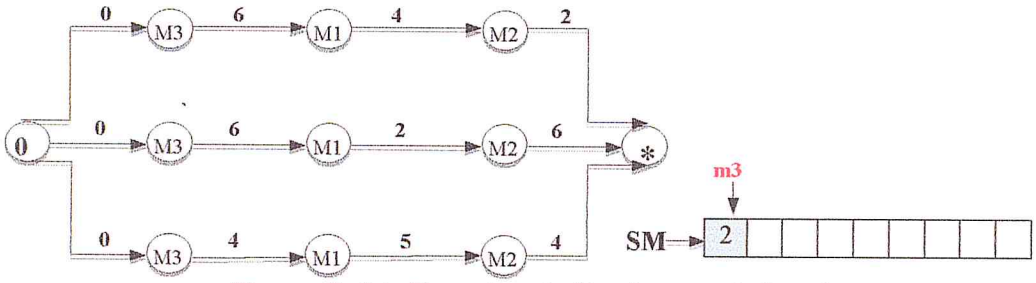


Figure 7: L'affectation de l'opération 1 du job 2

Les opérations ordonnables sont l'opération 1 du job 1, l'opération 2 du job 2 et l'opération 1 du job 3. Nous avons *Queue (1)=12, Queue (2)=8 et Queue (3)=13*. La règle consiste à choisir l'opération appartenant au job ayant donné la queue maximal i.e. l'opération 1 du job 3.

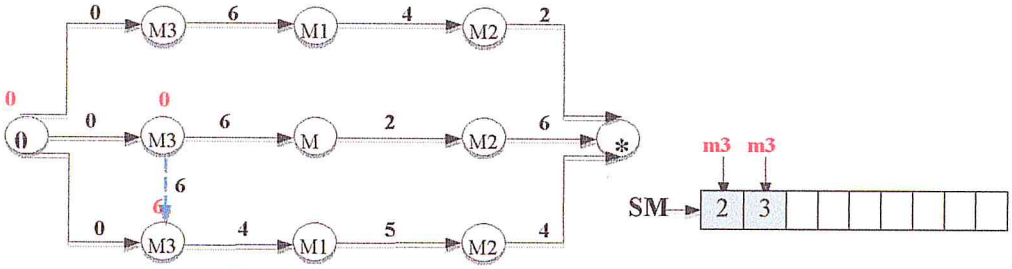


Figure 8: L'affectation de l'opération 1 du job 3.

Les opérations ordonnables sont l'opération 1 du job 1, l'opération 2 du job 2 et l'opération 2 du job 3. Nous avons *Queue (1)=12, Queue (2)=8 et Queue (3)=9*. La règle consiste à choisir l'opération appartenant au job ayant donné la queue maximale. Par exemple l'opération 1 du job 1.

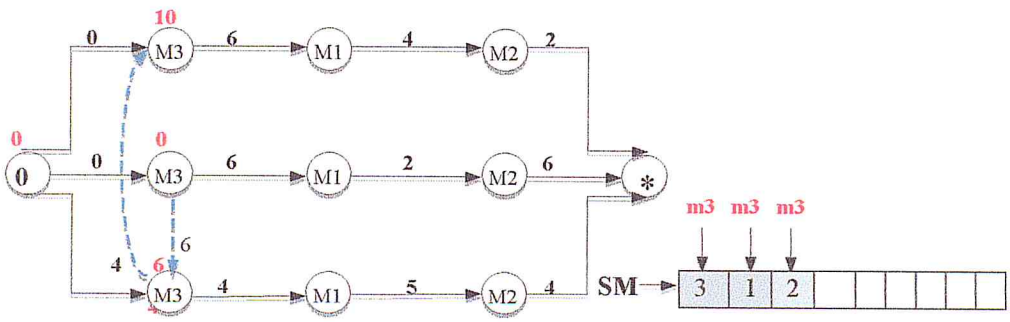


Figure 9: L'affectation de l'opération 1 du job 1.

A la fin de l'exécution de l'heuristique on obtient une solution initiale du problème JS3.

Dans cette partie nous avons montré comment construire une solution complète à un problème de job-shop. Le point traité dans la partie suivante est la possibilité d'améliorer une solution donnée, s'appuyant sur des propriétés de voisinage. Nous proposons une recherche locale efficace qui permet d'améliorer une solution, basée sur l'algorithme *Local_Research* utilisé en **[LAR, 10]** pour le problème du Job-Shop avec transport qui est adapté au notre cas.

6. Recherche Locale :

Cette partie est consacrée à la recherche locale, qui constitue une autre brique de base des méta-heuristiques que nous utilisons pour résoudre le problème du JS.

6.1. Présentation générale :

Cette recherche local consiste à chercher toute succession d'opérations ayant lieu sur la même machine et appartenant à des jobs différents, cette succession est recherché au long du chemin critique, donc la recherche de ce chemin est nécessaire.

Pour casser le chemin critique on procède par inversion des arcs qui sont aux extrémités de ces successions, dans notre cas, on inverse les arcs à l'extrémité la plus à gauche, car on analyse le chemin critique à l'envers partant du nœud {*} vers {0}. Chaque fois qu'on rencontre une succession on inverse l'arc et on réévalue la nouvelle configuration du graphe. Ces opérations sont effectuées sur le vecteur *SM*.

Dans ce qui suit, on donne une illustration en s'appuyant sur la solution fournie par le graphe disjonctif et sa représentation sous forme de vecteur illustrée par la même Figure 10.

Nous avons choisi ce graphe parce que le chemin critique qu'il contient comporte un certain nombre de successions d'opérations de jobs différents sur la même machine, ce que va nous aider à bien démontrer le fonctionnement de cet algorithme en utilisant un voisinage de type N2. La Figure 10 présente une solution de départ qu'on va améliorer au moyen de la recherche locale.

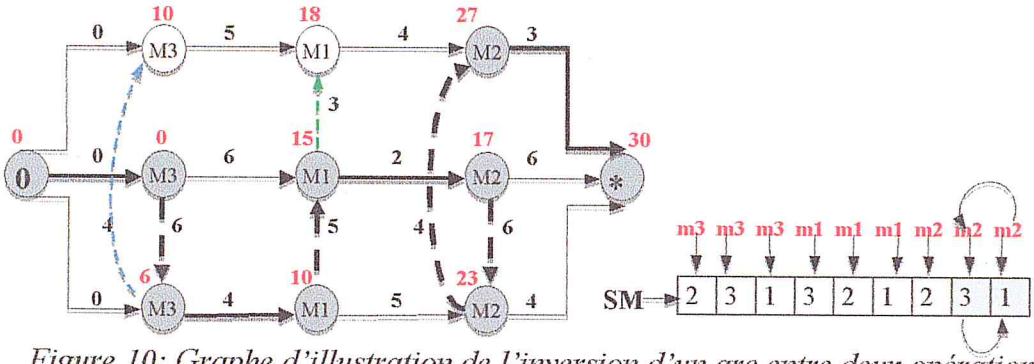


Figure 10: Graphe d'illustration de l'inversion d'un arc entre deux opérations.

Dans l'analyse de la solution donnée par le graphe de la Figure 10, on tente d'améliorer la solution, pour ce là il suffit d'inverser l'arc ($O_{3,3} \rightarrow O_{3,1}$) (le résultat de cette opération est donné par la Figure 12. Cette opération nécessite d'inverser l'ordre relatif de ces deux opérations dans le vecteur *SM*.

On signale la disparition des arcs ($O_{3,3} \rightarrow O_{3,1}$) et ($O_{3,2} \rightarrow O_{3,3}$), et l'apparition de nouveaux arcs ($O_{3,1} \rightarrow O_{3,3}$), ($O_{3,3} \rightarrow O_{3,2}$) réalisées par la simple permutation des cases concernées dans le vecteur *SM* (voir Figure 11). Après cette opération, la nouvelle solution obtenue est meilleure que la solution précédente puisqu'elle vaut 29 alors que l'ancienne valait 30.

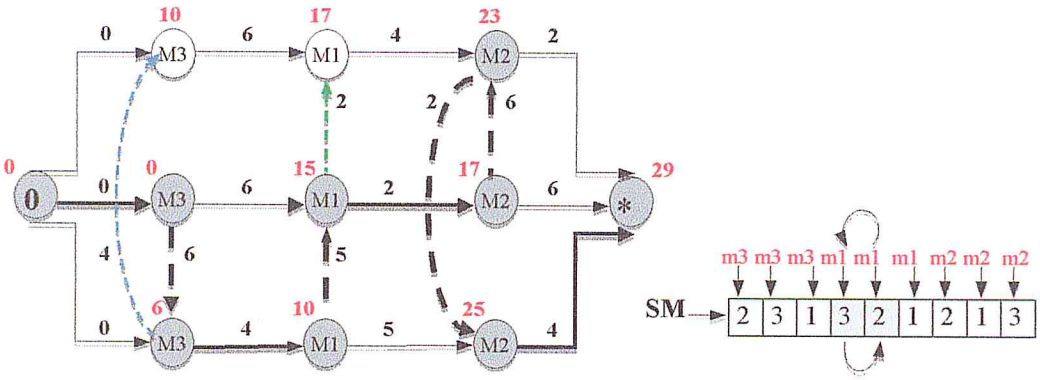


Figure 11: Résultat de l'inversion d'un arc.

On obtient un nouveau graphe dans lequel figure un arc $O_{3,1} \rightarrow O_{3,3}$.

Ce graphe représente une solution de coût 29 qui est strictement inférieur au coût de 30. Ce graphe est donc le graphe courant sur lequel la recherche locale va continuer. Ce graphe possède un nouveau chemin critique.

On parcourt le nouveau chemin critique en partant du sommet final * et on recherche une nouvelle succession. Dans notre cas, le premier, est celui des

opérations $O_{3,1}$ et $O_{3,3}$ sur la machine M2, on est conduit dans un premier temps à inverser l'ordre de ces opérations, ce qui donnerait le graphe initial de coût 30. Cette modification donnant une solution de coût supérieur, le parcours du chemin critique continue jusqu'à la succession des opérations $O_{3,2}$ et $O_{3,1}$ sur la machine M2, mais l'inversion de l'arc formé par ces deux opération donnerait elle aussi un graphe de coût supérieur au coût du graphe courant, donc on continue le parcours du chemin critique jusqu'à la succession formée par les opérations $O_{2,3}$ et $O_{2,2}$ sur la machine M1.

Pour améliorer la solution, il suffit d'inverser l'arc $(O_{2,3} \rightarrow O_{2,2})$ (le résultat de cette opération est donné par la Figure 12. Cette opération nécessite d'inverser l'ordre relatif de ces deux opérations dans le vecteur *SM*.

On signale la disparition des arcs $(O_{2,3} \rightarrow O_{2,2})$ et $(O_{2,2} \rightarrow O_{2,1})$, et l'apparition de nouveaux arcs $(O_{2,2} \rightarrow O_{2,3})$, $(O_{2,3} \rightarrow O_{2,1})$ réalisées par la simple permutation des cases concernées dans le vecteur *SM* (voir Figure 12). Après cette opération, la nouvelle solution obtenue est meilleure que la solution précédente puisqu'elle vaut 26 alors que l'ancienne valait 29, donc l'inversion de cet arc sera validée.

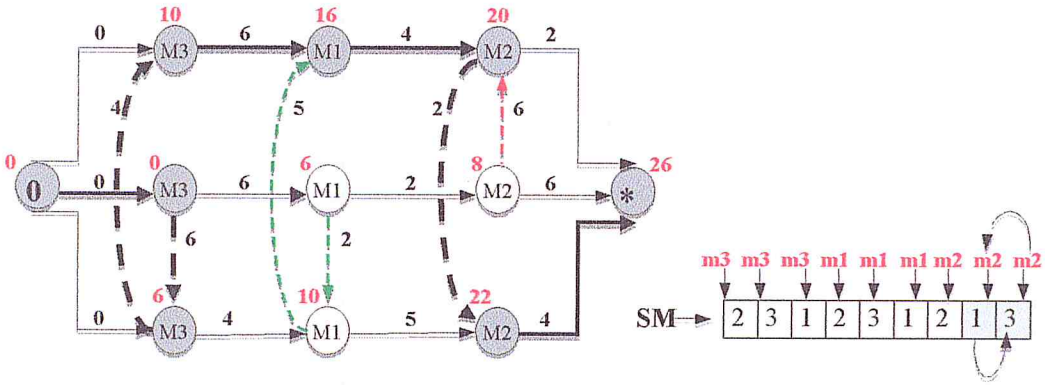


Figure 12: graphe illustrant l'amélioration de la solution.

Le processus de la recherche locale continue avec l'inversion de l'arc formé par les opérations $O_{3,1}$ et $O_{3,3}$.

Cette procédure se répète jusqu'à :

- l'obtention d'un chemin critique ne possédant pas des successions d'opérations appartenant à des jobs différents sur la même machine ;
- ou bien, quand on atteint le nombre maximal d'itérations.

Ces mécanismes sont implantés dans la procédure Rec_Loc dont le pseudo code

est donné par l'Algorithme 2.

Pour simplifier la lecture de ce pseudo code, nous utilisons l'opération *i* à la place de *f* (i) et l'opération j à la place de *f* (Pred[i] = *j*)= *f* (*j*) . Une solution *S* est représentée par les 2 vecteurs *SM* et PRED augmentée d'une variable *Cmax* qui sert à stocker le coût de cette solution. Evaluer (S.SM, S.PRED, S.Cmax), est une fonction d'évaluation d'une solution *S* donnée par le vecteur *SM* et qui retourne le coût de cette évaluation dans la variable *Cmax* et stocke le chemin critique dans le vecteur PRED.

***Algorithme 2**: Recherche locale*

```
Nom de la procédure : Rec_Loc
Données
S: solution donnée par (SM, PRED, Cmax)
Sorties
S: solution donnée par (SM, PRED, Cmax)
Début
i:=N+1   // commencer l'analyse à partir du nœud {*}
iter:=1    // initialiser le compteur d'amélioration à 1
Cbest := S.Cmax  // Initialiser le meilleur coût au coût de la solution d'entrée
   Tant que (i ≠0) and (iter<=nm) faire
        j:=Pere(i)    // j le prédécesseur immédiat de l'opération i dans le chemin critique
        Si (job[i] ≠job[j]) alors
                Save_S :=S
                Permuter les opérations i et j dans DM
                Evaluer(S.SM, S.PRED, S.Cmax)
                Si (S.Cmax <Cbest ) alors
                        Cbest :=S.Cmax ;    // amélioration
                        i:= PRED (N+1) ;    // retour au nœud {*} du nouveau chemin critique
                         iter :=iter+1 ;
                Sinon
                        S:=Save_S ;     // annuler le changement
                        i:=PRED[i] ;     // continuer l'analyse d'ancien chemin critique
        Fin Si
   Fin Tant que
    Retourner S
Fin
```

L'algorithme *Rec-Loc* nous permet d'améliorer notre solution mais on risque de rester bloqués dans une solution minimale locale.

Pour palier ce problème on a décidé de combiner cette recherche locale avec la méthode du Kangourou, ce que nous permettra d'atteindre des endroits plus

prometteurs afin de trouver des solutions optimales. Le fonctionnement de cette méthode sera décrit dans ce qui suit.

7. La méthode du Kangourou :

Elle se comporte à la manière d'un kangourou. Le principe est d'effectuer une descente stochastique, et lorsque l'état minimal actuel est de même coût depuis trop longtemps, accepter une transition défavorable dans un voisinage de l'état actuel et ceci quel que soit le coût. Ceci est appelé un saut. Ensuite on débute une nouvelle descente stochastique.

L'avantage principal d'un algorithme du Kangourou est qu'il permet de minimiser des fonctions pouvant prendre des valeurs infinies.

L'algorithme suivant montre le fonctionnement de cette méthode.

Algorithme 3 : *Algorithme du Kangourou*

```
Nom de procédure : KANGOUROU
  X : Solution initiale
Début
    c :=0
    Choisir un état X0
    X :=X0
      Tant que nécessaire faire
        Si (c<A) Alors
           Choisir Y dans le V(X)
              Si H(Y)≤H(X) Alors
                 Si H(Y)<H(X) Alors
                    c:=0
                 Fin Si
                X := Y;
              Fin Si
              c := c + 1
        Sinon (* Faire un saut*)
             Choisir Y dans W(X)
             Si H(Y)≠H(X) Alors
                    c:=0
            Fin Si
            X:=Y;
        Fin Si
     Fin Tant que
    Fin
```

Nous avons choisit cette méthode car elle donne des solutions de bonne qualité et elle est facile à implémenter.

Conclusion :

Dans ce chapitre nous avons proposé la combinaison de certaines méthodes parmi les méthodes existantes pour résoudre le problème du job-shop. Nous avons utilisé le graphe conjonctif-disjonctif, par la suite nous avons utilisé une heuristique de construction de solutions du problème du job-shop. On a utilisé un algorithme de recherche locale pour améliorer la solution construite par l'heuristique de construction et on a combiné cette recherche locale avec la méthode du kangourou pour éviter de se bloquer dans un minimum local.

Ces méthodes seront implémentées par l'agent ordonnanceur lors de réalisation décrite dans le chapitre suivant.

Ces modélisations et les approches de résolution constituent un outil d'aide à la décision pour établir une stratégie de conception ou d'évolutions des ateliers de production.

CHAP 4
Méthodologie et réalisation

1. Introduction :

Un projet informatique requiert la mise en œuvre de différentes technologies pour sa phase de développement. Cette partie présente en premier lieu les choix effectués pour la réalisation de notre système en termes de langages, d'un système de communication à travers une plateforme multi-agent et de bien d'autres composants indispensables. Ensuite, elle décrit son fonctionnement ainsi que la réalisation du Système Multi-Agent, et en dernier lieu les interfaces et résultats de l'application.

2. Environnement de développement :

2.1. Plateforme de développement des agents:

Comme plateforme pour le développement de notre SMA on a choisi Jade.

2.1.1. JADE

Jade (Java Agent Développement Framework **[WEB 1] :** est une plate-forme de développement d'agents gratuite et Open Source développée par le laboratoire TILAB (Telecom Italie Lab) et qui résulte principalement des activités de recherche. Ces principales caractéristiques sont :

- JADE simplifie l'implémentation d'un SMA à travers un Middleware répondant aux spécifications de la FIPA une librairie de classes que les utilisateurs peuvent utiliser et étendre ainsi un ensemble d'outils graphiques qui permettent le débogage et l'administration du SMA à concevoir;
- JADE assure une communication transparente par l'échange de messages dans l0e langage normalisé FIPA-ACL;
- JADE diminue l'effort de programmation car elle implémente deux agents (DF et AMS) dont les fonctionnalités sont utiles à notre application;
- JADE a comme but la construction des SMA et la réalisation d'applications conformes à la norme FIPA (FIPA, 1997).
- JADE comprend deux composantes de base : une plate-forme agents compatible FIPA et un paquet logiciel pour le développement des agents Java
- JADE est entièrement implémentée en Java.

2.1.2. Architecture de Jade (figure 1) :

- Une application Jade est une plateforme déployée sur une ou plusieurs machines.
- La plateforme héberge un ensemble d'agents, identifiées de manière unique, pouvant communiquer de manière bidirectionnelle avec les autres agents.
- Chaque agent s'exécute dans un conteneur (container) qui lui fournit son environnement d'exécution, il peut migrer à l'intérieur de la plateforme.
- Toute plateforme doit avoir un conteneur principal qui enregistre les autres conteneurs.
- Une plateforme est un ensemble des conteneurs actifs.

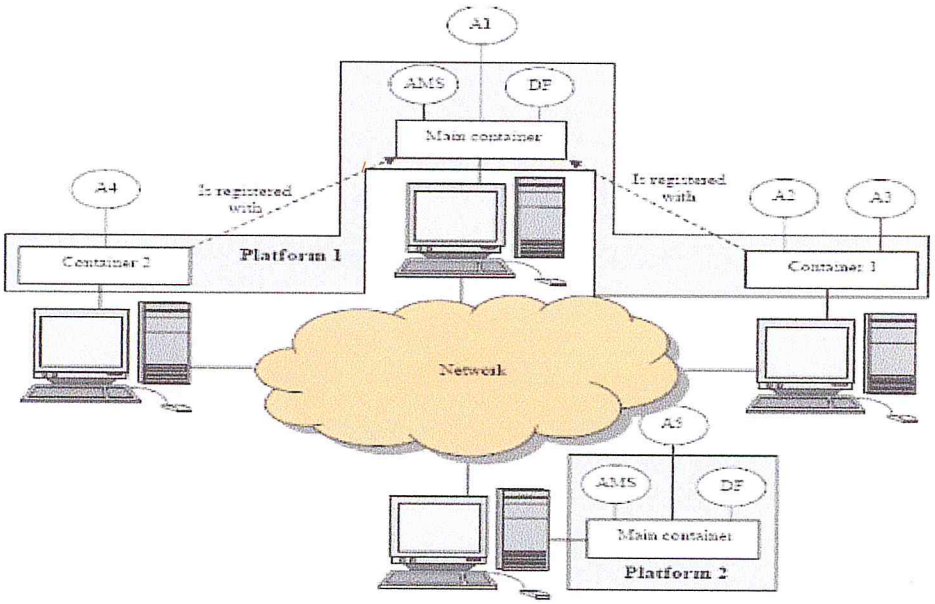


Figure 1 : Architecture de Jade.

2.1.3. La norme FIPA pour les systèmes multi-agents :

Les premiers documents de spécification de la norme FIPA (FIPA 1997), appelés spécifications FIPA97, établissent les règles normatives qui permettent à une société d'agents d'inter-opérer. Tout d'abord, les documents FIPA décrivent le modèle de référence d'une plate-forme multi-agents (figure 2) où ils identifient les rôles de quelques agents clés nécessaires pour la gestion de la plate-forme, et spécifient le contenu du langage de gestion des agents et l'ontologie du langage.

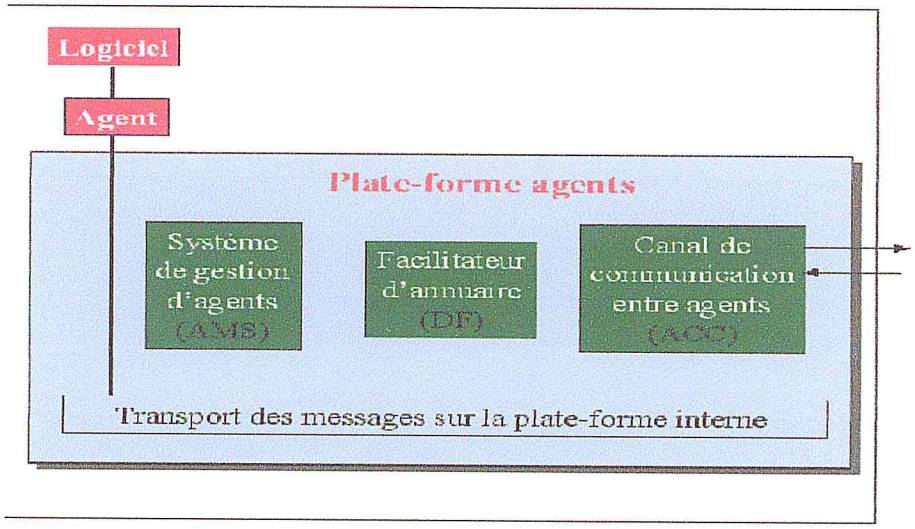


Figure 2 : Plateforme multi-agent FIPA.

Dans la figure 2 nous voyons trois rôles principaux dans une plate-forme multi agents FIPA :

- Le **Système de Gestion d'Agents** (Agent Management System - AMS) est l'agent qui exerce le contrôle de supervision sur l'accès à et l'usage de la plate-forme ; il est responsable de l'authentification des agents résidents et du contrôle d'enregistrements.
- Le **Canal de Communication entre Agents** (Agent Communication Channel - ACC) est l'agent qui fournit la route pour les interactions de base entre les agents dans et hors de la plate-forme ; c'est la méthode de communication implicite qui offre un service fiable et précis pour le routage des messages ; il doit aussi être compatible avec le protocole IIOP pour l'interopérabilité entre les différentes plates-formes multi-agents.
- Le **Facilitateur d'Annuaire** (Directory Facilitator - DF) est l'agent qui fournit un service de pages jaunes à la plate-forme multi-agents.

2.2. Choix du langage de programmation :

Afin de réaliser ce prototype, nous avons choisi le langage Java. Ce choix a été motivé par les raisons suivantes :

- Les agents développés sous la plate-forme JADE, sont entièrement écrits en Java;

- Java assure une totale indépendance des applications vis-à-vis de l'environnement d'exécution : c'est à dire que toute machine supportant Java est en mesure d'exécuter un programme sans aucune adaptation (ni recompilation, ni paramétrage de variables d'environnement);
- Une programmation orientée objet et une bibliothèque immense d'objets prêts à l'emploi;
- Un accès simplifié aux bases de données ;

En ce qui concerne l'environnement de développement, nous avons choisi Eclipse IDE **[WEB 3]**, pour Java EE Développeurs, pour le développement d'applications orientées objet.

2.3. La base de données :

Afin de respecter les choix technologiques de la société en assurant une cohérence autour de la base de données utilisée par la société on a choisi l'Oracle 11g **[WEB 2]** comme notre serveur de données et l'Oracle SQL Développer qu'est un fort client sql pour les requêtes et administration des BDD.

2.4. Langage de modélisation :

Jade est une plateforme qui aucune méthodologie est spécifié, donc pour cela on va choisir AUML pour l'analyse et conception.

AUML (Agent-based Unified Modeling Language) **[HUN, 06]** : est une extension d'UML réservé pour l'analyse et la conception orientées agent.

L'utilisation d'AUML a des avantages :

- Elle utilise les notions d'UML dans les diagrammes.
- Elle supporte tous les diagrammes de phases d'analyse et de conception.
- Elle modifie les diagrammes d'UML et elles deviennent les diagrammes qui peuvent représenter l'interaction entre les agents et l'état interne de l'agent.
- Elle introduit le diagramme de protocole.

3. L'architecture d'application (figure 3):

On va concevoir un système qui contient trois parties principales : le coté de l'utilisateur, celui d'agent ordonnanceur et celui des agents atelier :

- Le côté de l'utilisateur : fournit l'interface entre l'utilisateur et système pour envoyer la demande d'ordonnancement et recevoir le résultat.
- Le côté d'agent ordonnanceur : reçoit la demande d'ordonnancement et envoi le résultat ; analyse la demande pour les ressources correspondantes, fait appel à une négociation avec les agents possédant les ressources en utilisant le port 1099 pour la communication par échange de messages, intègre les réponses reçues des agents et réalise l'ordonnancement et améliore les résultats des algorithmes présentées au chapitre 3, envoi les résultats au utilisateur et divise les planning pour après les envoyer aux agents concernés.
- Le côté des agents atelier (débit, serrurerie, etc…) : reçoit l'appel à proposition sur la disponibilité de ses ressources et envoi des propositions de disponibilités dans la négociation avec l'agent ordonnanceur; récolte la disponibilité de ses ressources à partir de planning de la ressource concerné en accédant sa base de connaissance par l'intermédiaire du fichier connectBD qui réalise la connexion avec la bdd par l'intermédiaire du JDBC et le port utilisé est le 1521, sauvegarde les nouveaux tâches à réaliser dans le planning de ces ressources.

L'architecture de l'application CRMOSMA est montrée dans la figure 3 :

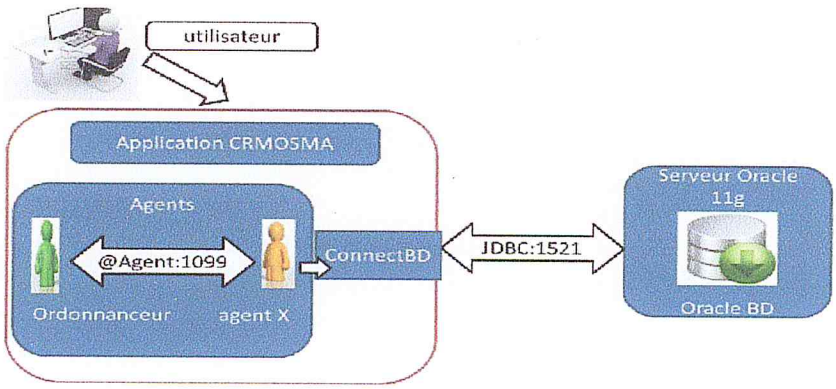


Figure 3 : Architecture d'application.

4. Modélisation du système multi agent :

Pour le fonctionnement de ce système il faut que :

- Les entités représentant chaque atelier extraient automatiquement des informations de la ressource de son atelier ;

- Une entité centrale qui lance l'algorithme d'ordonnancement après avoir communiqué avec les entités de chaque atelier : la communication a lieu plusieurs fois, les informations échangées sont sous plusieurs types différentes;
- Les entités dans le système doivent fonctionner de la façon active : ils doivent écouter activement la demande de l'entité centrale pour la servir tout suite.

Si on applique une méthode d'analyse et conception orientées objet, on aura des difficultés :

- Comment peut-on modéliser un objet qui est capable de fonctionner automatiquement sans control?
- Comment peut-on modéliser un système distribué dans lequel on ne sait pas combien des composants qu'il va contenir?
- Comment peu-ton modéliser la communication entre l'entité de lancement d'ordonnancement et les entités représentant les ateliers en utilisant les appels passifs des fonctions des objets? Surtout dans le cas où ces communications se passent plusieurs fois et qu'il y a plusieurs types d'informations échangées!
- Est-ce que les objets (entités atelier) sont actifs pour réagir activement quand l'entité de lancement demande?

Cependant, si on applique une méthode d'analyse et conception orientées agent, on aura des facilités :

- On n'a plus besoin de s'occuper du fonctionnement automatique des entités atelier et de l'entité de lancement car la capacité de fonctionnement automatique est une caractéristique implicite de l'agent ;
- On peut modéliser les communications entre l'entité de lancement et les entités atelier grâce aux techniques de modélisation des interactions dans le système multi agent : modéliser le protocole ;
- Un agent est toujours actif pour réagir activement quand l'entité de lancement demande.

4.1. Méthode :

Puisque Jade ne contient pas une méthode standard pour modéliser les agents, on ne suit pas une méthode concrète. On va passer par deux phases :

- Dans la phase d'analyse, on a besoin de cinq modèles :
  - Modèle de buts : représenté par une hiérarchie des buts;
  - Modèle de cas d'utilisation : représente par le diagramme de cas d'utilisation;
  - Modèle des rôles et de tâches : représenté par le diagramme des classes, chaque classe représente un rôle ou un tâche;
  - Modèle de domaine : représenté par le diagramme des classes;
  - Modèle d'agents : représenté par le diagramme des classes, chaque classe représente un agent.

- Dans la phase de conception on a besoin de trois modèles:
  - Modèle de classe d'agent : représenté par le diagramme des classes, chaque classe représente un agent;
  - Modèle de protocole : représenté par le diagramme de séquence;
  - Modèle de structure d'agent : représenté par le diagramme des états, chaque état représente un état interne d'un agent;

4.1.1. Analyse :

4.1.1.1. Modèle de but :

L'objective de cette étape est de déterminer les buts du système et les relations entre eux. Pour déterminer les buts, on considère trois aspects :

- Afin d'avoir l'ordonnancement de la production, il faut : lancer l'ordonnancement et fournir l'ordonnancement à l'ordonnanceur;
- Afin de fournir l'ordonnancement à l'ordonnanceur, il faut une interface pour: recevoir la demande d'ordonnanceur et lui donner les résultats d'ordonnancement;
- Afin de lancer l'ordonnancement, il faut : extraire des données (machines, matière premier, temps d'exécution des opérations, temps de disponibilité des machines) nécessaires et sauvegarde des travaux issues d'ordonnancement.

Donc, on a un ensemble des buts du système :

- B0 : ordonnancement de la production ;
- B1 : fournir ordonnancement à l'ordonnanceur ;
- B2 : lancement d'ordonnancement ;

- B3 : réception de la demande d'ordonnancement ;
- B4 : envoi du résultat d'ordonnancement ;
- B5 : Extraction de données nécessaires ;
- B6 : Sauvegarde des travaux à réaliser.

En se basant sur les relations entre les buts, on construit l'arbre des buts suivant:

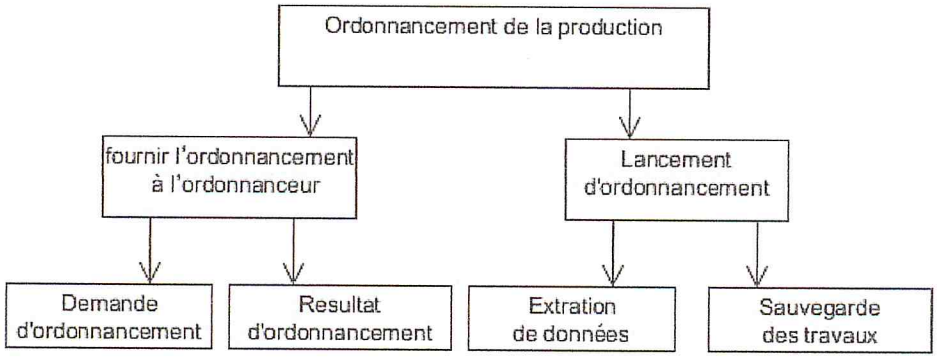


Figure 4 : modèle de but.

4.1.1.2. Modèle de cas d'utilisation :

L'objective de cette étape c'est de décrire le comportement du système en interagissant avec l'utilisateur (l'ordonnanceur) et l'agent ordonnanceur.

Le système a deux acteurs :

- L'utilisateur (ordonnanceur) : ce qu'utilise le système d'ordonnancement avec l'objective d'avoir un ordonnancement plus optimisé des plannings des travaux à réaliser;
- Agent ordonnanceur : entité qui récolte des données pour l'ordonnancement et realisé une négociation sur la disponibilité des ressources et le lancement d'ordonnancement en améliorant le résultat d'ordonnancement avec les algorithmes d'ordonnancement.

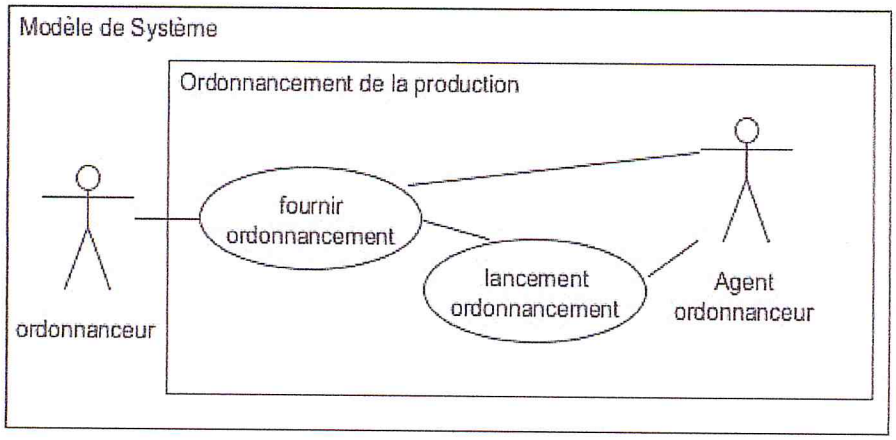


Figure 5 : Modèle de cas d'utilisation.

Donc, il y a deux cas d'utilisation principaux :

- Fournir ordonnancement : il contient la demande d'ordonnanceur de réalisation d'un ordonnancement et le résultat d'ordonnancement qui sera présenté au utilisateur;
- Lancement d'ordonnancement : après la collecte d'informations nécessaires à l'ordonnancement, l'ordonnancement est lancé en exécutant les algorithmes d'ordonnancement et d'optimisation des résultats, après les résultats sont diffusés aux agents concernés pour les enregistrer dans son planning.

4.1.1.3. Modèle des rôles :

L'objective de cette étape est de déterminer les rôles dans le système, les relations entre eux et les relations entre les rôles et les buts (tableau 1).

Selon l'ensemble des buts du système, on propose des rôles :

- R1 : Récepteur: c'est que reçoit la demande d'utilisateur;
- R2 : Négociateur : c'est qui lance la négociation après analyse de la demande;
- R3 : Lanceur : c'est qui exécute l'ordonnancement après négociation;
- R4 : Envoyeur : c'est qui envoie le résultat final à l'ordonnanceur;
- R5 : Participant : c'est qui envoie des propositions au négociateur et sauvegarde les travaux pour à la fin pouvoir les exécuter.

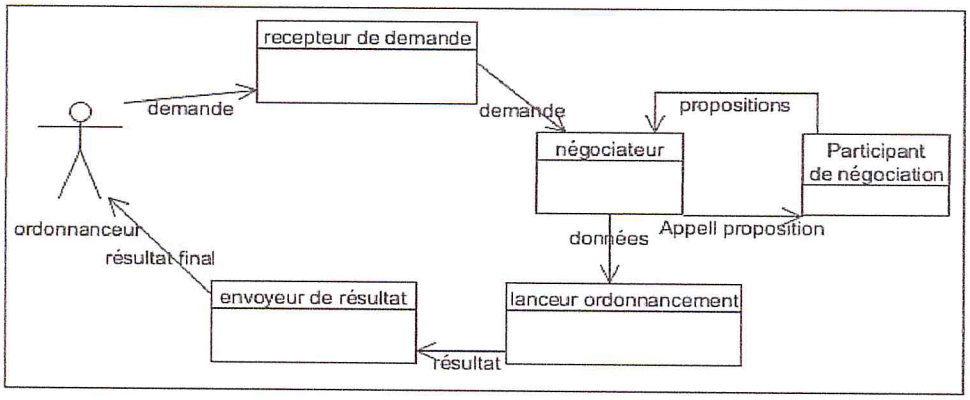


Figure 6 : Modèle des rôles.

Les relations entre les rôles et buts:

- Récepteur envoi demande à Négociateur;
- Négociateur demande des propositions au Participant de négociation ;
- Le Participant répond avec des propositions au Négociateur ;
- Négociateur fournit les données au Lanceur ;
- Lanceur lance un ordonnancement et envoi le résultat au Envoyeur ;

	B0	B1	B2	B3	B4	B5	B6
R1				X			
R2						X	
R3			X				
R4					X		
R5						X	X

Tableau 1 : Relations entre les rôles et les buts.

4.1.1.4. Modèle de domaine :

L'objective de cette étape est de déterminer les informations qu'on a besoin d'utiliser dans les communications entre les agents. Il y a deux types des informations auxquelles on s'intéresse :

- Les informations nécessaires pour échanger entre les agents. Elles sont sous forme des messages échangés entre eux.
- Les informations nécessaires pour comprendre le contenu de message. Elles sont sous forme d'une base de connaissance du système.

Type de message :

Il y a trois types de message :

- Adresse : il représente l'adresse d'un agent. Il contient un hôte auquel l'agent se situe et un port (port=1099) par laquelle l'agent se communique avec les autres.
- Enquête : il représente un message envoyé par l'agent ordonnanceur aux autres agents pour le début d'une négociation. L'attribut type représente l'acte communicatif soit :
  - un cfp(call for proposition) : appel à proposition sur la disponibilité d'une ressource, elle contient les ressources demandés,
  - accept-proposal : message acceptant la proposition d'un agent, elle contient aussi les travaux qui l'agent doit réaliser.

L'attribut protocole correspond au protocole de communication qui dans notre cas c'est le ContractNet. L'attribut contenu est contenu du message soit une ressource soit la ressource accompagnée des tâches à réaliser et leurs temps début de réalisation.

- Résultat : il représente le résultat obtenu d'une négociation. L'attribut type représente l'acte communicatif soit :
  - Propose : pour le message possédant la proposition d'un agent atelier,
  - Inform : pour le message informant l'agent ordonnanceur du succès de prise en charge des travaux;

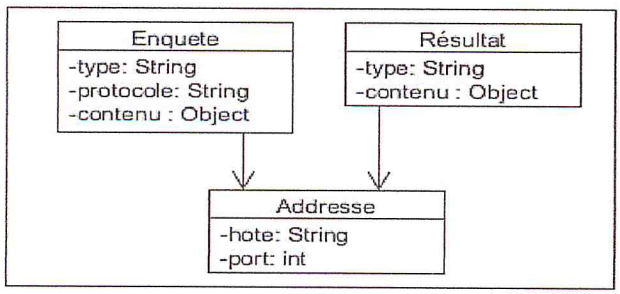


Figure 7 : Modèle de domaine.

Base de connaissance du système :

On doit représenter de la connaissance grâce laquelle les agents peuvent comprendre ce qui est demandé et extraire des informations nécessaires pour répondre à la demande.

Notre système possède une base de données décentralisé qui servira de base de connaissance aux agents. Chaque agent possède sa propre base de connaissance comme source d'information et passent par une phase de récolte d'information sur ces plannings pour la négociation et coopération :

- Ord (ordonnanceur) (figure 9): base de connaissance de l'agent ordonnanceur,
- Débit, Serrurerie, Charpente, Cuverie, E.I.D, Coffrage et Peinture (figure 8): sont des base de connaissance de chaque agent atelier concerné et tous ont la même caractéristique d'information et tables.

Les informations sur la structure des tables sont représentées par les tableaux 2 et 3.

Table	Description
ATELIER	Table des ateliers
RH	Table des ressources humaines
MACHINE	Table des machines
OPERATION	Table des opérations (Débit, Peinture, …)
OUVRAGE	Tables des ouvrages
PAQUET	Table des paquets
COMMANDE	Table des commandes
DOSSIER	Table des dossiers techniques
S_FAMILLE	Table des sous familles des articles
FAMILLE	Table des familles des articles
UTILISATEUR	Table des utilisateurs du système
OP_SFAM	Table associative (OPERATION et S_FAMILLE)
PAQ_SFAM	Table associative (PAQUET et S_FAMILLE)
ELEMENT	Table des éléments constituant une pièce
PIECE	Table des pièces graphiques
PAQ_OP_MAC	Table associative (PAQUET, OPERATION et MACHINE)
TRAVAUX	Table des travaux affectés à une machine

Tableau 2 : Table des fichiers.

Table	COLONNE	DESCRIPTION	TYPE	TAILLE
ATELIER	C_atelier	Code de l'atelier	C	2
	Lib	Libellé de l'atelier	C	50
RH	C_RH	Code de ressource humaine	C	3
	C_Mac	Code de la machine opéré	C	6
	Nom	Nom du technicien	C	40
	Prénom	Prénom du technicien	C	80
	H_entree	Heure d'entrée	N	5
	H_Sortie	Heure de sortie	N	5
MACHINE	C_mac	Code de la machine	C	6
	Lib	Libellé de la machine	C	100
	C_atelier	Code d'atelier affecté	C	2
	C_op	Code de l'opération	C	2
OPERATION	C_op	Code de l'opération	C	2
	Lib	Libellé de l'opération	C	30
	Type	Type	C	1
	C_mac	Code de la machine	C	6
OUVRAGE	C_ouv	Code de l'ouvrage	C	8
	N_cmd	Numéro de la commande	C	6
	D_ouv	Date de l'ouvrage	D	
	Lib_ouv	Libellé de l'ouvrage	C	50
	P_ouv	Poids de l'ouvrage	N	7
PAQUET	C_paquet	Code du paquet	C	8
	C_ouv	Code de l'ouvrage	C	8
	D_paquet	Date du paquet	D	
	P_paquet	Poids du paquet	N	7
COMMANDE	N_cmd	Numéro de la commande	C	6
	C_dos	Code du dossier	C	6
	D_cmd	Date de la commande	D	
DOSSIER	C_dos	Code dossier	C	6
	C_ouv	Code ouvrage	C	8
	N_plan	Numéro du plan	C	8
	Lib_dos	Libellé dossier	C	50
	P_dos	Poids du dossier	N	7
	D_dos	Date de dossier	D	
	C_sprod	Code sous-produit	C	2
	D_debit	Date débitage	D	
	D_ass	Date assemblage	D	
	D_solde	Date du solde	D	
	T_al_ass	Temps alloué en assemblage	N	7
	T_pass_ass	Temps passé en assemblage	N	7
S_FAMILLE	C_sfamille	Code sous-famille	C	5
	Lib	Libellé	C	30
	Poids	Poids théorique	N	7
	C_famille	Code de la famille	C	3

Tableau 3 : Description de la structure des tables.

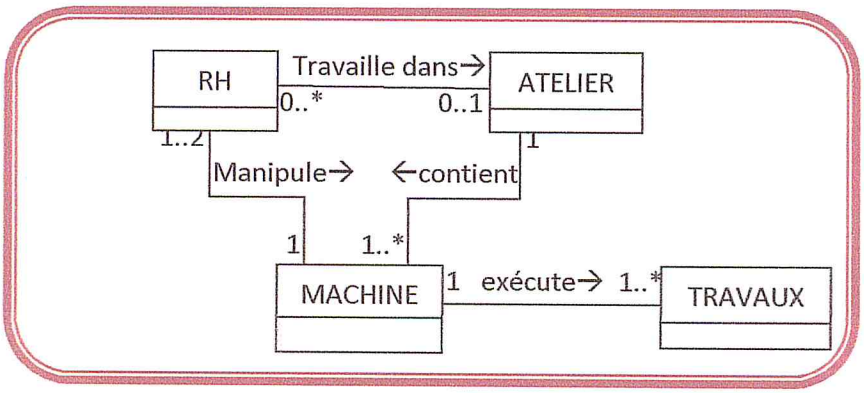


Figure 8 : diagramme de classe de la base de connaissance d'agent atelier.

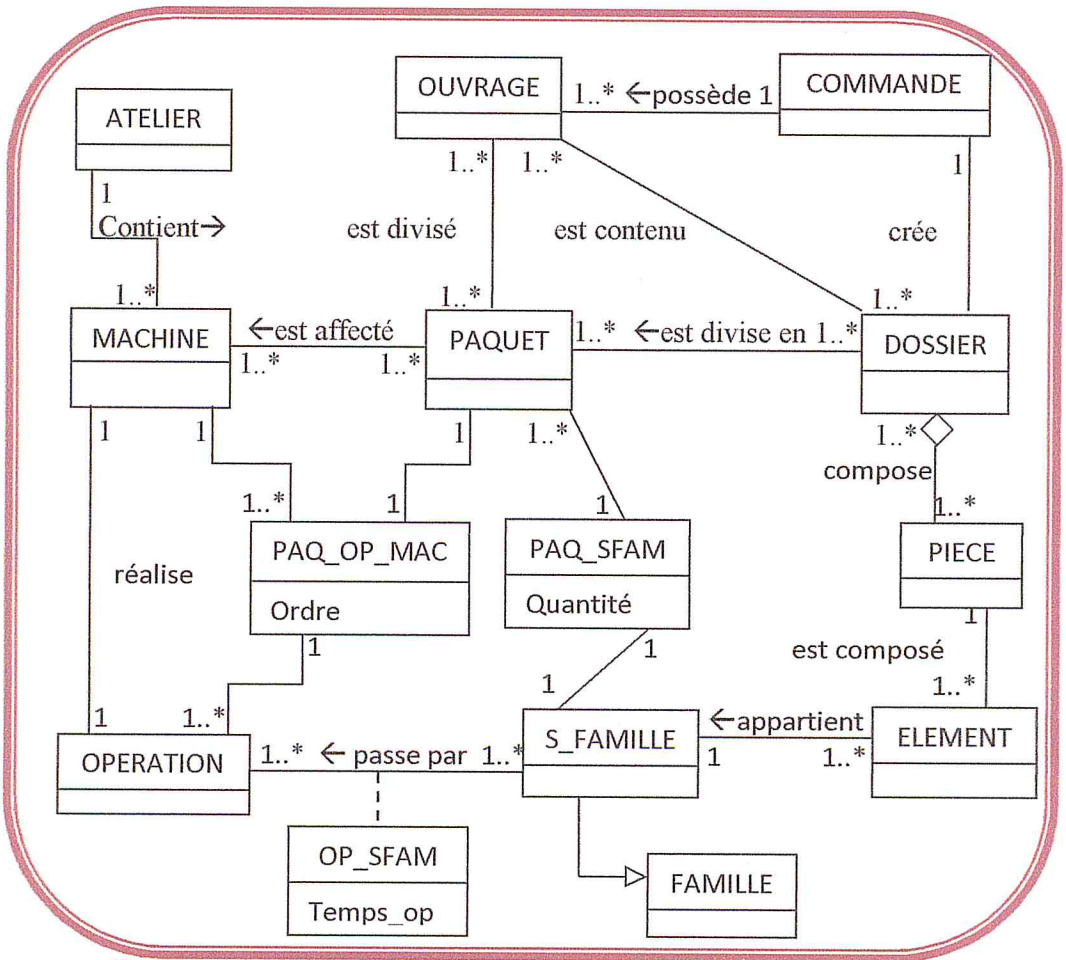


Figure 9 : diagramme de classe de la base de connaissance d'agent ordonnanceur.

4.1.1.5. Modèle d'agents :

L'objective de cette étape est de déterminer les agents dans le système, les relations (communication) entre eux, les rôles et les buts.

Pour déterminer les agents du système, on propose huit agents, l'agent ordonnanceur et autres sept qui représentent un atelier de production:

- L'agent ordonnanceur : ce qui communique avec l'utilisateur, gère la manipulation générale du système d'ordonnancement, c'est-à-dire, il gère l'ordonnancement et les négociations avec les autres agents;
- L'agent débit : ce qui répond la demande de proposition concernant les disponibilités machines de débitage à l'agent ordonnanceur ;
- L'agent charpente : suivant son planning de production, il donne au agent ordonnanceur les disponibilités des ressources demandés ;
- L'agent cuverie : donne les disponibilités de ces ressources suivant le planning de production de son atelier ;
- L'agent coffrage : donne la disponibilité de ces ressources suivant le planning de production de son atelier ;
- L'agent EID : donne la disponibilité de ces ressources suivant le planning de production de son atelier ;
- L'agent serrurerie : donne la disponibilité de ces ressources suivant le planning de production de son atelier ;
- L'agent peinture : donne la disponibilité de ces ressources suivant le planning de production de son atelier ;

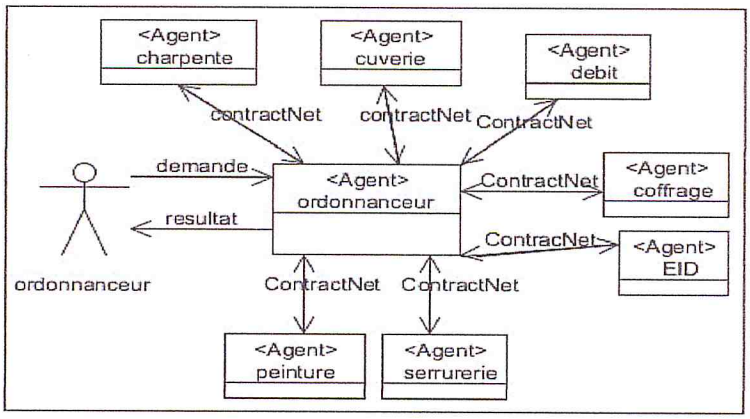


Figure 10 : Modèle d'agent.

4.1.2. Conception :

4.1.2.1. Modèle de classes d'agent :

Le système a huit classes agents et un protocole entre eux et un utilisateur :

- Les huit classes d'agent sont : ordonnanceur, débit, serrurerie, coffrage, charpente, E.I.D, cuverie et peinture.
- Le protocole est le ContractNet (voir modèle de protocole);
- L'utilisateur est l'ordonnanceur.

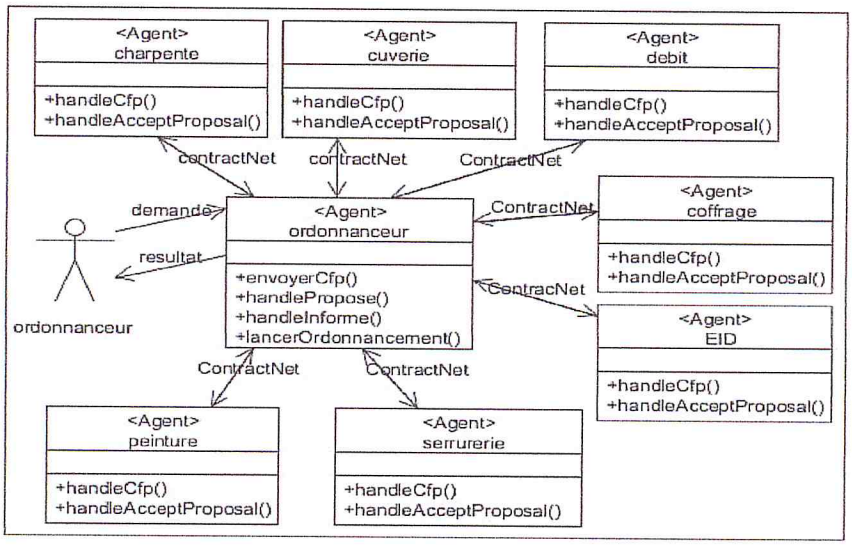


Figure 11 : Modèle de classe d'agent.

Les méthodes des classes on comme fonctionnalités:

- envoyerCfp() : c'est le méthode qu'initialisé la négociation entre l'agent ordonnanceur et les autres agents par envoi de messages, comme message elle envoie une appel à proposition (cfp) contenant les ressources dont il a besoin de leur disponibilité,
- handleCfp() : c'est le méthode de prise en charge d'appel de proposition(cfp) venant de l'agent ordonnanceur, l'agent extrait les disponibilités des ressources à partir de sa base de connaissance des plannings de ses ressources demandés et envoi une message de proposition de disponibilité,
- handlePropose() : ce méthode traite la proposition (propose) venant des agents, extrait les disponibilités des ressources et fait appel au méthode lancerOrdonnancement(), après les résultats venant de ce dernier il organise les

travaux et leur temps de début ainsi que les ressources concernés et envoi par une message d'accept-proposal ,

- handleAcceptProposal() : c'est le méthode de prise en charge de l'acte communicatif accept-proposition, l'agent reçoit les travaux et temps du début ainsi que les ressources concernés, enregistre le planning et envoi une message d'inform,
- handleInform() : c'est une simple méthode qui reçoit une message infom de confirmation des travaux ont été prise en charge par les agents concernées,
- lancerOrdonnancement() : c'est le méthode qui lance les algorithmes d'ordonnancement :

- algorithme 1 : l'algorithme heuristique HJS qui construit une solution d'ordonnancement initial,
- algorithme 2 : l'algorithme de recherche local pour améliorer la solution d'ordonnancement initial,
- algorithme 3 : l'algorithme du kangourou pour combiner avec la recherche locale afin de trouver des solutions d'ordonnancement plus optimales.

4.1.2.2. Modèle de structure d'agent :

Le système contient huit classes d'agents : l'ordonnanceur, débit, serrurerie, coffrage, charpente, cuverie, E.I.D, peinture. On donne le modèle du ordonnanceur (figure 12) et débit (figure 13), les autres la présentation est la même que le débit.

- **Agent Ordonnanceur** :

L'agent ordonnanceur a comme comportement le cycle suivant :

- Tout d'abord, après être créé, il entre en l'état de réception de demande d'ordonnancement ;
- Quand une demande d'ordonnanceur arrive, il entre en état de création d'appel des propositions (cfp), après cella il possède un ensemble de cfp qu'il envoi aux agents;
- Après avoir envoyé des cfp aux Participants de la négociation, il entre en état d'attente pour attendre toutes les propositions des Participants ;
- Après avoir reçu les réponses, il entre en état de teste de type de réponse reçu qui peut être une proposition ou une information ;

- Si la réponse est une proposition, il passe à l'état de charge de réponse à la proposition, et il envoie une acceptation de proposition ;
- Après envoi de acceptation de proposition il passe à l'état d'attente ;
- Si la réponse est une information, il passe à l'état de charge d'information, il envoie le résultat à l'ordonnanceur et entre en état de réception d'une nouvelle demande.

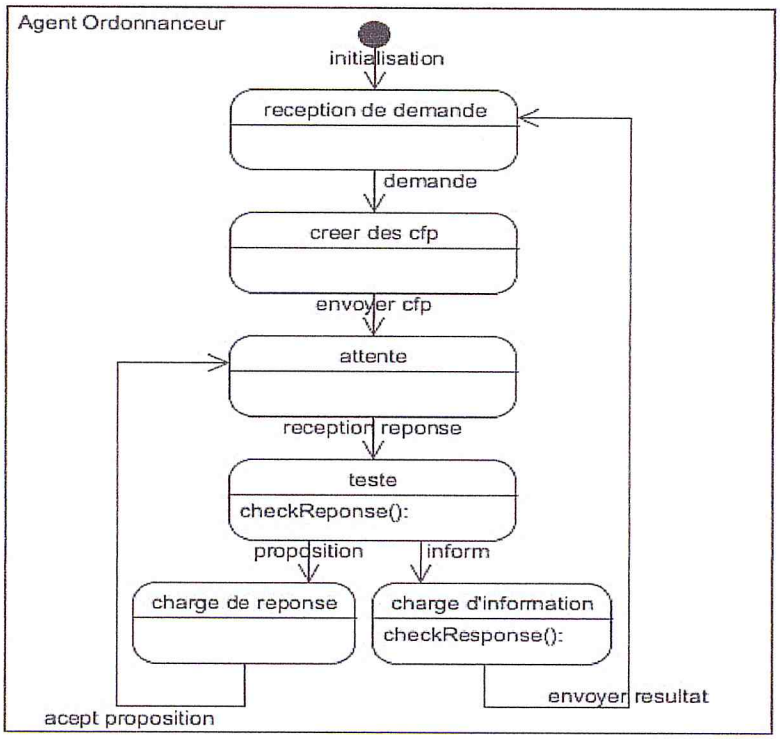


Figure 12 : Modèle d'état d'agent ordonnanceur.

- **Agent Débit :**

L'agent débit comme les autres six agents ont comme comportement le cycle suivant :

- Tout d'abord, après être créé, il entre en état de réception d'appel de proposition d'agent ordonnanceur ;
- Quand l'appel arrive il est en état de création de proposition ;
- Après avoir envoyé la proposition à l'agent ordonnanceur il entre en état d'attente pour recevoir une acceptation de proposition ;
- Après avoir reçu l'acceptation, il entre en état de créer une information sur l'exécution de la tâche. Puis il envoi à l'agent ordonnanceur et rentre en état de réception des appels de propositions d'agent ordonnanceur.

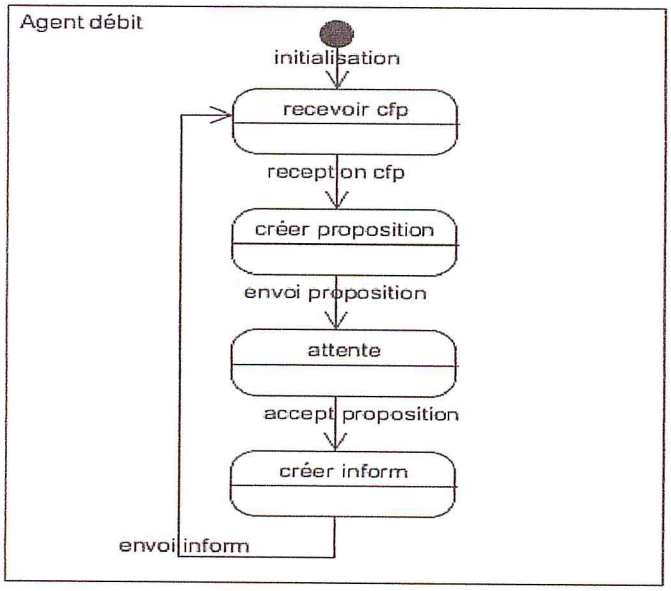


Figure 13 : Modèle d'état d'agent débit.

4.1.2.3. Modèle de protocole :

Il y a un seul protocole, le ContractNet, il permette à l'agent ordonnanceur de faire des demandes des propositions, sur la disponibilité des machines des différents ateliers, aux agents ateliers (débit, serrurerie, coffrage, …).

- Les participants : l'agent ordonnanceur (transmetteur) et les agents ateliers (débit, serrurerie, coffrage, …).
- Les informations échangées : une cfp (call for proposition), des propositions, acceptation d'une proposition et information.

Scénario :

- L'agent ordonnanceur envoie un appel à proposition (cfp) qui contient les machines dont il a besoin savoir sur le temps de disponibilité;
- L'agent atelier (débit, coffrage, …) concerné envoi propose (proposition) sur le temps de disponibilité de la ressource demandé;
- L'agent ordonnanceur envoi accept-proposal, après lancement d'ordonnancement, avec des tâches à réaliser;
- L'agent atelier envoi un inform, informant de la prise en charge des travaux.

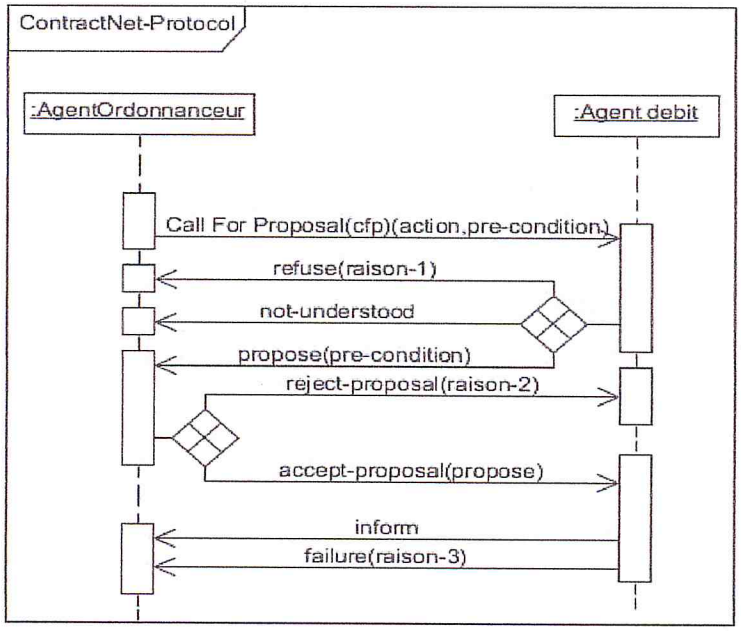


Figure 14 : Modèle de protocole.

5. Les résultats :

En termes de réalisation, le lancement d'ordonnancement est fait par l'intermédiaire d'une interface utilisateur (Ordonnanceur). À travers cette interface (figure 15) l'utilisateur (ordonnanceur) lance un dossier et les paquets constituant les jobs (travaux).Chaque paquet contient une seule sous famille de produit et pour chaque sous famille subi à d'opérations suivant un ordre comme montre le tableau 4.

Dossier	Paquet	Sous Famille	Operations (avec ordre d'exécution)
110044	11004401	10504	4, 7, 9
	11004402	10504	4, 7, 8, 9
	11004403	10102	4, 7, 9

Tableau 4 : Job à ordonnancer.

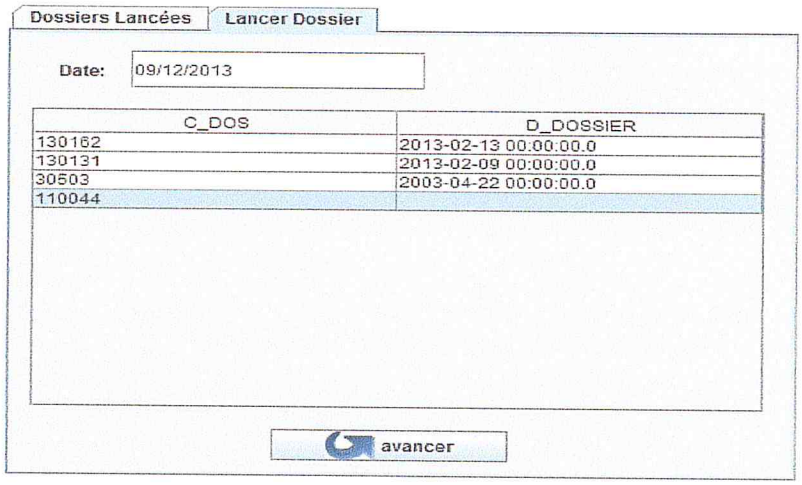


Figure 15 : Interface ordonnanceur pour lancement de dossier.

Donc, à partir de l'interface d'utilisateur, l'utilisateur donne à l'agent ordonnanceur les informations nécessaires à l'ordonnancement comme le temps d'exécution d'opérations pour la sous-famille de produit (figure 16).

Temps d'operation par sous famille

C_OP	C_SFAMILLE	TEMPS_OP
4	30123	2
7	10102	2
7	10311	3
7	10315	3
7	10503	2
7	10504	5
7	20107	4
7	30112	6
7	30123	2
8	10504	4
8	20107	4
8	30112	6
9	10102	5

Figure 16 : Interface ordonnanceur pour acquisition du temps d'operation.

La désignation des opérations du tableau 4 est donnée par le tableau 5.

Opération	Désignation
4	Scie
7	Repro
8	Poinç
9	Grugeag

Tableau 5 : Désignation des opérations.

Lorsque l'ordonnancement est lancé par l'agent ordonnanceur, l'agent va construire une solution d'ordonnancement de départ en exécutant l'algorithme 1 de l'heuristique HJS expliqué au chapitre 3 et on aboutit à la solution de départ de cout 67 donnée par le tableau 6.

Paquet	Opération	Temps Opération (/minute)	Temps début opération (/minute)
11004403	Scie	9	32
11004402	Scie	3	41
11004402	Repro	5	44
11004401	Scie	3	44
11004401	Repro	5	49
11004402	Poinç	4	49
11004403	Repro	2	54
11004403	Grugeag	5	56
11004402	Grugeag	3	61
11004401	Grugeag	3	64
Cmax	67		

Tableau 6 : Résultat d'ordonnancement avant amélioration.

La solution présentée par le tableau 6 n'est pas automatiquement la meilleure, donc elle va passer pour une phase d'amélioration où l'agent ordonnanceur fait appel à l'algorithme de recherche locale. Après un certain nombre d'itérations il aboutit à une solution plus optimale donnée par le tableau 7 de coût 60 qu'est strictement inférieur au coût de la solution initiale.

Paquet	Opération	Temps Opération (/minute)	Temps début opération (/minute)
11004402	Scie	3	32
11004401	Scie	3	35
11004402	Repro	5	35
11004403	Scie	9	38
11004401	Repro	5	40
11004402	Poinç	4	40
11004403	Repro	2	47
11004403	Grugeag	5	49
11004402	Grugeag	3	54
11004401	Grugeag	3	57

Cmax	60

Tableau 7 : Résultat d'ordonnancement après amélioration initial.

La solution présentée par le tableau 7 c'est une solution approché qui peut être encore amélioré par l'algorithme du kangourou, on aboutit à une solution plus optimale encore donnée par le tableau 8 de coût 55 qu'est encore meilleur qui le coût de 60.

Paquet	Opération	Temps Opération (/minute)	Temps début opération (/minute)
11004402	Scie	3	32
11004402	Repro	5	35
11004401	Scie	3	35
11004403	Scie	9	38
11004402	Poinç	4	40
11004401	Repro	5	40
11004402	Grugeag	3	44
11004401	Grugeag	3	47
11004403	Repro	2	47
11004403	Grugeag	5	50
Cmax	55		

Tableau 8 : Résultat d'ordonnancement après amélioration final.

Conclusion :

Dans ce chapitre, nous avons présenté la méthodologie (analyse et conception) et une application baptisé CRMOSMA (CR Métal Ordonnancement par Systèmes Multi Agents) exploitant la technologie multi agent, pour une élaboration d'ordonnancement de production d'atelier du type Job Shop.

Dans notre solution nous avons implémenté dans l'agent ordonnanceur des méthodes approchées (recherche locale et kangourou) qui nous ont apporté des solutions optimales car avec des agents qui possèdent des bases de connaissance distribué dont les tâches sont distribuées entre les agents rendant plus rapide la récolte de disponibilité des ressources passant par des négociations pour les réunir et lancer l'ordonnancement et les algorithmes de recherche locale et kangourou vient pour optimiser le résultat d'ordonnancement.

Annexe A

Annexe A : Analyse et Conception du système

1. Introduction :

L'application contient une partie d'administration et une partie utilisateur, la partie utilisateur a été modélisé sur la base des agents, et la partie administration de la base de donné on applique une méthode d'analyse et conception orienté objet qui sera présentée dans ce qui suit.

2. Méthodologie du système :

2.1 Analyse et Conception :

Pour mener l'analyse du système d'administration nous utilisons UML qui est un langage graphique de modélisation des systèmes d'information, autrement dit, de notation ou de représentation graphique du domaine du monde réel que l'on désire modéliser.

Dans la phase d'analyse, il y a :

- Maquettes Prototypes (interfaces) ;
- Diagramme de cas d'utilisation ;
- Diagramme de séquence ;
- Diagramme de collaboration ;
- Diagramme de classes (voir l'annexe A) ;

2.1.1. Maquettes Prototypes :

Dans ce qui suit nous illustrons les maquettes qui vont constituer l'application :

Tout utilisateur doit passer par une connexion dans le système qui est représenté par la première interface d'accueil (figure 1).

Figure 1 : Interface de connexion de CRMOSMA.

Figure 2 : L'interface d'interaction avec l'administrateur.

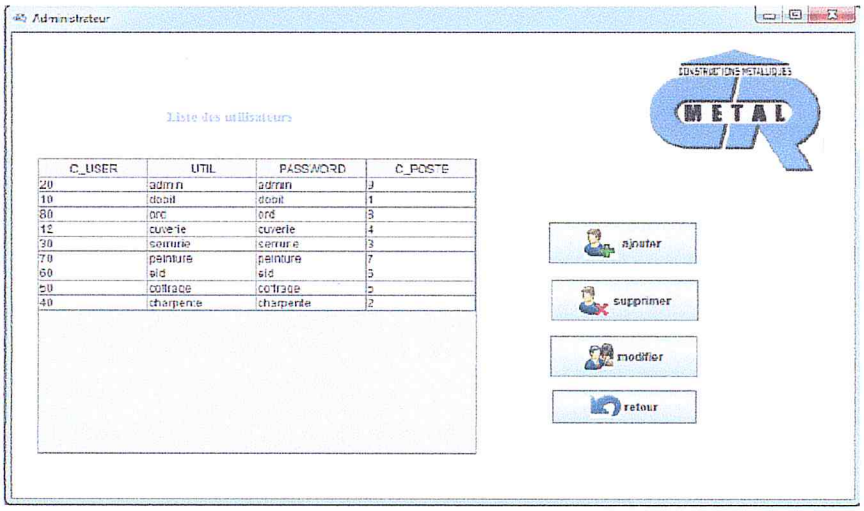


Figure 3 : Interface Gestion Utilisateurs (partie Admin).

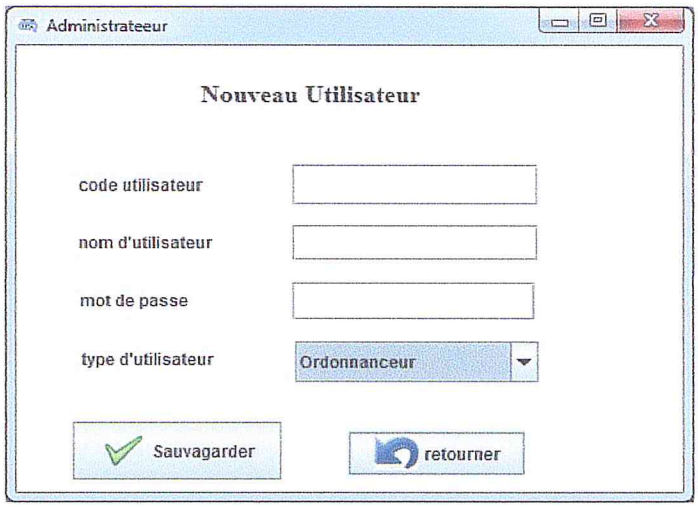


Figure 4 : Interface d'ajout d'un nouvel utilisateur.

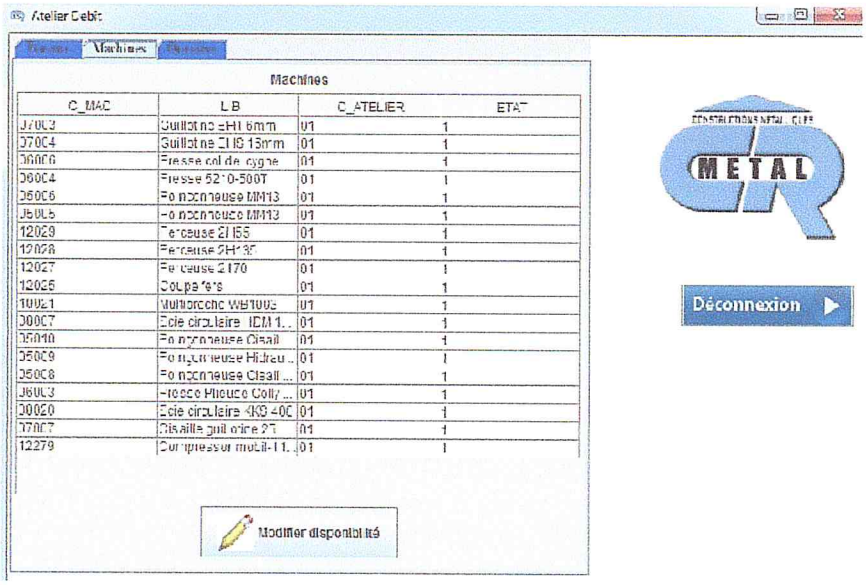


Figure 5 : Interface Atelier Débit.

Afin de faciliter la maintenance corrective et évolutive du code nous veillons ainsi à déterminer les classes en fonction de trois types :

- Dialogue (Interface/Vue) : regroupe les classes qui constituent l'interface homme machine (IHM). Ce sont en majorité des classes graphiques issues des maquettes. Elles n'effectuent aucun traitement hormis ceux qui permettent des interactions avec l'utilisateur et sont responsables de l'affichage des données.
- Contrôle : les classes de ce type établissent la jonction entre les classes des deux autres types. Par exemple, la demande de création d'un client est prise en compte par une classe contrôle qui la transmet à la classe entité possédant cette méthode. On peut aussi leur confier le contrôle du respect des règles métiers. Les classes techniques sont également de ce type.
- Entité (modèle): il correspond aux classes propres au domaine modélisé. Ces classes sont le plus souvent persistantes et sont implémentées pour notre projet dans une base de données Oracle. Elles disposent des méthodes dites CRUD (Create, Read, Update, Delete).

Nous aboutissons à une architecture MVC.

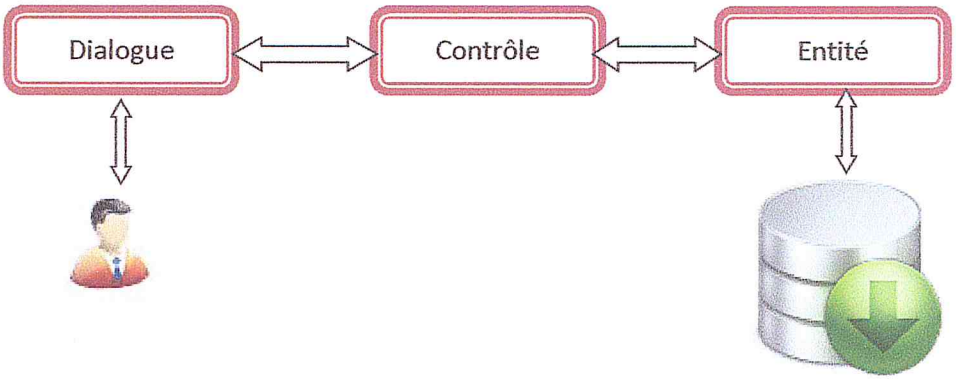


Figure 6: Architecture MVC.

2.1.2. Diagramme de cas d'utilisation :

Il s'agit avec le diagramme de cas d'utilisation de définir le comportement de la future application vis-à-vis de l'utilisateur ou de son environnement (autres systèmes informatiques, automates…).

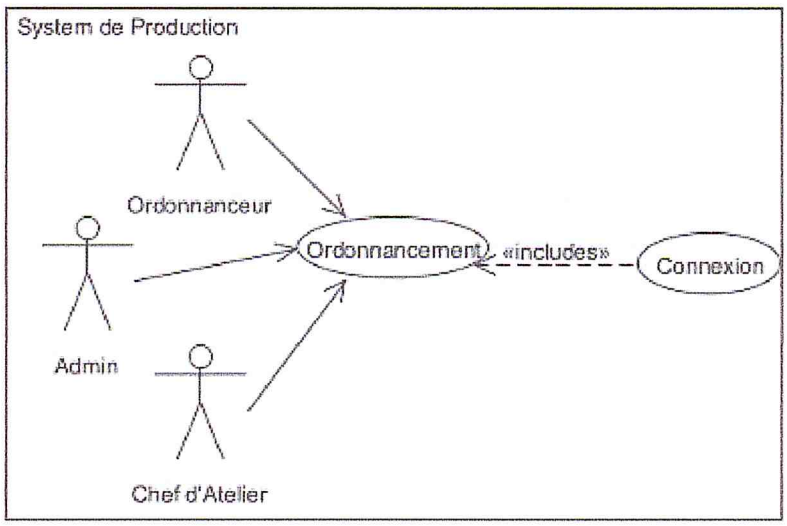


Figure 7 : Diagramme de cas d'utilisation.

2.1.3. Diagramme de Séquence :

Le diagramme de séquence s'attache à modéliser l'aspect dynamique du système. Il peut être comparé à un storyboard mettant en évidence les interactions existantes entre des objets du système.

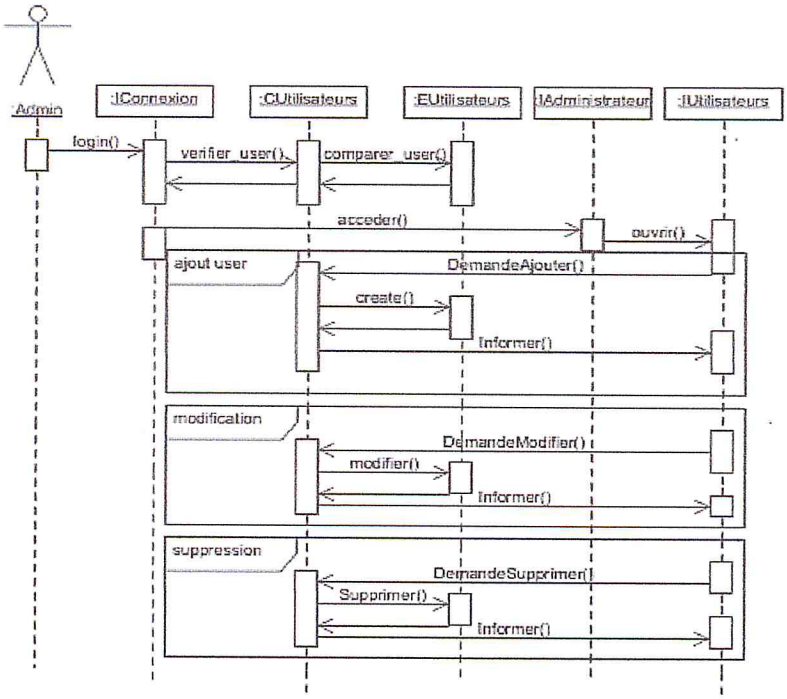


Figure 8: Diagramme Sequence (Gestion Utilisateurs).

2.1.4. Diagramme de Collaboration :

Le diagramme de collaboration est équivalent du point de vue sémantique au diagramme de séquence. Il présente par contre l'organisation structurelle des objets tout en mettant comme précédemment en évidence les messages émis ou reçus.

La figure 9 représente le diagramme de collaboration de gestion d'utilisateurs.

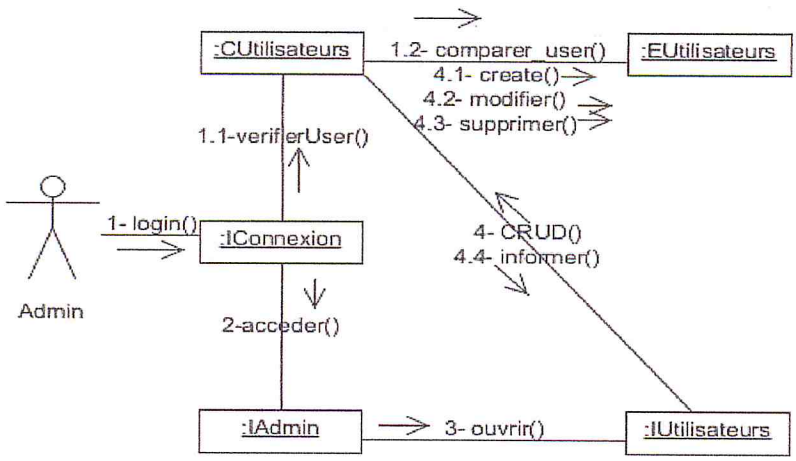


Figure 9 : Diagramme de collaboration (Gestion utilisateur).

Conclusion Générale

Nous nous sommes intéressés dans ce mémoire à la modélisation et à la résolution de problèmes difficiles venant du job-shop. Ceci nous a conduits à définir une modélisation et une méthode d'optimisation.

Ce travail commence avec la présentation d'un état d'art où on donne une description des différents problèmes d'ordonnancement et on fait un parcours des méthodes les plus utilisés dans la littérature pour résoudre des problèmes d'ordonnancement des ateliers de production.

Un état d'art concernant les agents et les systèmes multi agents a aussi été effectué où on a pu recenser des notions à ce sujet, tel que les différents types d'agents, les architectures d'agents, les langages de communication inter agent, les protocoles de communication et les plateformes de développement.

Notre démarche d'optimisation nous a permis d'adapter le modèle de graphe disjonctif-conjonctif et les outils associés de manière à les mettre en œuvre sur des algorithmes itératifs.

Pour effectuer l'ordonnancement, une heuristique de construction a été proposée pour nous donner une solution de départ que par la suite serait améliorée. Pour améliorer la solution nous avons proposé une recherche locale efficace permettant de nous donner une bonne solution, mais avec cette méthode on risquait de rester bloqués dans une solution minimale localement et pour pallier ce problème nous avons proposé la combinaison de cette méthode avec la méthode du kangourou que nous permet de sortir d'une vallée.

L'étude des problèmes de job-shop nous a permis de mettre en pratique nombre de compétences qui ont été développées dans ce mémoire. Ces compétences ont surtout été appliquées au travers du choix des outils de modélisation et des méthodes utilisées. Ce choix dépend de nombreux facteurs dont principalement la complexité des problèmes, des objectifs visés et du temps dont on dispose pour résoudre les problèmes.

De plus, afin d'apporter une solution efficace (collaborative), nous proposons dans ce travail une approche intelligente et distribuée basée sur les systèmes multi-agents. Il s'agit de la modélisation du système multi agent où les agents sont cognitifs possédant sa propre base de connaissance (base de connaissance décentralisée). Les agents du système sont divisés en deux : agent ordonnanceur celui

qui lance l'ordonnancement et distribue les travaux aux agents atelier (débit, serrurerie, peinture, coffrage, etc) représentant les ateliers de production.

Pour lancer l'ordonnancement, l'agent ordonnanceur passe d'abord par une phase de récolte d'informations nécessaires à l'ordonnancement après il passe par la phase de négociation avec les agents atelier que par son tour gèrent un planning et un ensemble de ressources, et à partir de ce planning, les agents atelier doivent être capable de donner à l'ordonnanceur la disponibilité de leurs ressources. Ce sont ces dates de disponibilité que l'ordonnanceur va utiliser pour lancer un nouvel ordonnancement et cet ordonnancement a été optimisé par l'utilisation des algorithmes de recherche locale combinée avec le kangourou donnant ainsi une solution plus optimisé.

Les agents ont été développés sur la plateforme Jade qui est basé sur le protocole de communication inter-agents FIPA, le Contract Net a été utilisé comme protocole de communication entre les agents pour faciliter la négociation.

Les perspectives qui nous souhaitons donner à ce travail concernent :

- L'optimisation des systèmes de type job-shop avec transport ;

BIBLIOGRAPHIE

[ADD, 11]: ADDAD Nora, SAIDANI Monira, Approche méta-heuristique basée sur la méthode de recherche d'harmonique pour la résolution des problèmes d'ordonnancement industriels, mémoire pour l'obtention d'un diplôme Master en Informatique. Université Saad Dahlab, 2011.

[ALO, 06] : ALOULOU Mohamed Ali, Modélisation et résolution des problèmes d'ordonnancement, Université de Paris Dauphine, 2006.

[ATT, 97]: ATTOUI, AMAR, Les Systèmes Multi-Agents et le Temps Réel, Paris, Editions Eyrolles, Mars 1997, 510p.

[BEL, 11] : BELLIFA Imane, Approche Multi-Agent pour la reconnaissance de diabète, thèse Master en Informatique Université Abou Bakr Belkaid – Tlemcen, Algerie.

[BIE, 95]: BIERWIRTH C. A generalized permutation approach to job-shop scheduling with genetic algorithms: OR Spektrum; 17:87-92, 1995.

[CAR, 88] : CARLIER, JACQUES Aniar, Problèmes d'ordonnancement : modélisation, complexité et algorithme, 1988.

[CHA, 01]: CHAIB-DRAA B., JARRAS I., MOULIN B., Systèmes multiagents : Principes généraux et applications, BRIOT J. P., DEMAZEAU Y., Eds., Agent et systèmes multiagents, Hermès, 2001.

[DRE, 03] : DREO Johann, PETROWSKI Alain, SIARRY Patrick et TAILLARD Eric, Méta heuristique pour l'optimisation difficile, Edition Eyrolles, 2003.

[DUR, 90]: DURFEE E. H. and MONTGOMERY T. A.. A Hierarchical Protocol for Coordinating Multiagent Behaviors. In Proceedings of the 8th National Conference on Artificial Intelligence, pages 86-93, Cambridge, July- August 1990. Volume One.

[ESQ, 99]: ESQUIROL Patrick, LOPEZ Pierre, Concepts et méthodes de base en ordonnancement de la production, in Ordonnancement de la production, Information, Commande, Communication, 2001, pp. 25-53.

[FER, 95]: FERBER J., Les systèmes multi-agents vers une intelligence collective, Inter Editions, 1995.

[FIPA, 97]: FIPA, ''Agent Communication Language'' Specification FIPA, 1997.

[FLE, 95] : FLEURY, G. Application de méthodes stochastiques inspirées du recuit simulé à des problèmes d'ordonnancement. RAIRO A.P.I.I. (Automatique Productique - Informatique industrielle) ; 29(4–5):445–470, 1995.

BIBLIOGRAPHIE

[GOT, 93] : GOTHA, Les problèmes d'ordonnancement. RAIRO- Recherche opérationnelle/ Operation research, 27(1):77-150, 1993.

[HUN, 06]: NGUYEN Manh Hung, HO Tuong Vinh, Génie Logiciel Orienté Agent, Hanoi, 2006.

[JEN, 98b]: JENNINGS N., SYCARA K., WOOLDRIDGE M., A Roadmap of Agent Research and Development, Autonomous Agents and Multi-Agent Systems, vol. 1, n°1, p. 7 - 38, July 1998.

[LAR, 10]: Mohamed LARABI, Le problème de job shop avec transport, modélisation et optimisation, thèse de doctorat en informatique Université Blaise Pascal - Clermont Ferrand II, 2010.

[NWA, 96]: NWANA H. S., Software Agents: An Overview. In Knowledge Engineering Review, Vol. 11(3), pp.205-244, 1996.

[QUI, 02]: Joel QUINQUETON, Outils logiciels des Systèmes Multi-Agents, LIRMM, Montpellier, France, 2002.

[REA, 03] : Jihad READIY, Etude et mise en œuvre d'une architecture d'agents en réseau dans les systèmes dynamiques situés : Pilotage de systèmes de production complexes, thèse Ecole Doctorale de l'Université de Savoie, 2003.

[RHO, 00]: RHODES J., Just-In-Time Information Retrieval. Ph.D. Thesis, MIT Media Lab, May 2000.

[ROY, 64]: ROY B. et SUSSMAN B. Les problèmes d'ordonnancement avec contraintes disjonctives. In : Note DS N°9 bis, SEMA, Paris, 1964.

[RUS, 95] : RUSSELL S. J., NORVIG P., Artificial Intelligence. A Modern Approach, Prentice-Hall, Englewood Cliffs, 1995.

[SEC, 03] : SECQ Y., RIO : Rôles, Interactions et Organisation, une méthodologie pour les systèmes multi-agents ouverts, Thèse Doctorat, Université des Sciences et Technologies de Lille, France, 2003.

[YAN, 02] : YANN Collette, PATRICK Siarry, Optimisation multi objectif, Edition Eyrolles, 2002.

[WEB 1] http://www.jade.tilab.com/

[WEB 2] http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index.html

[WEB 3] http://www.eclipse.org/downloads/