République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la
Recherche Scientifique

Ecole Nationale Supérieure Polytechnique

Département de Génie Industriel

Mémoire du Projet de Fin d'Etudes d'Ingénieur

Thème

Contribution à la résolution d'un problème d'ordonnancement en temps-réel sur deux ressources identiques en parallèle par l'approche collaborative

Réalisé par:

M. Walid BEHIRI
M. Mohamed Rami LATRECHE

Dirigé par :

M. BAKALEM

Promotion : juin 2009

Dédicaces

A mes Parents qui n'ont cessé de me rappeler

qu'on ne peut se fier à la richesse quelle qu'elle

soit. Seule l'instruction fait exception.

A mon frère Abdelkader et ma sœur Amina.

A tous mes amis.

Walid

A ma famille,

A mes amis,

Et à mes enseignants.

Rami

Remerciements :

Nous remercions Monsieur BAKALEM, notre promoteur, pour ses précieux conseils et ses remarques, qui nous ont permis de réaliser et mettre en valeur notre travail.

Nous remercions, également, Mademoiselle Bougchiche pour ses remarques et le temps qu’elle a consacré pour nous.

Nous remercions tous les enseignants du département Génie Industriel pour leur soutien tout au long de notre formation.

ملخص:

في هذه الدراسة ، نقترح دراسة مشكلة جدولة الأشغال في زمن واقعي على وسيلتين متماثلتين على التوازي. نبدأ بتقييم المقارب المتواجدة لحل هذا المشكل. ثم نقترح مقاربة تعاونية كطريقة للقرار, التي نقارنها, بعد ذالك, مع المقاربات السابقة.

الغاية هي تخفيض مدة البرمجة الكلية و زيادة عدد الأشغال الغير متوقعة البرمجة.

نظرا للصعوبة الشديدة للمشكلة, استعمال طريقة دقيقة يصبح مستحيل, لهذا, اخترنا مقاربة تجريبية محضة مستعملين تقنيات التصنع لتقييم الفعالية, و ذالك, بفضل استعمال ARENA.

المفاتيح
مشاكل الجدولة, تقنيات التصنع, مقاربة تعاونية, زمن واقعي, وسائل متماثلة على التوازي.

Résumé :

Dans cette étude, nous nous proposons d'étudier un problème d'ordonnancement en temps-réel sur deux ressources identiques en parallèle. Nous commençons par l'évaluation des performances des approches de résolution existantes, puis, nous proposons une approche collaborative comme méthode de résolution, que nous comparerons aux approches précédentes.

L'objectif est de minimiser la durée totale d'ordonnancement, Makespan, ainsi que la maximisation du nombre de tâches aléatoires exécutées.

Du fait du caractère NP-difficile du problème, l'utilisation d'une approche de résolution exacte est impossible, c'est pourquoi nous avons opté pour une approche empirique moyennant la simulation afin d'évaluer la performance. Pour cela, le simulateur ARENA sera utilisé.

Mots clés : *Ordonnancement, Simulation, Approche collaborative, Système multi agents, Protocole de communication, Ordonnancement temps réel, Ressources Identiques en Parallèle, Makespan.*

Abstract:

In this study, we propose to study an on-line scheduling problem with two parallel identical resources. We begin by evaluating the performance of existing resolving approaches, and then we suggest a collaborative approach as a resolving method, that we compare with others approaches.

The purpose is to minimize the scheduling total duration "Makespan", and to maximize the executed random tasks number.

Because of the NP-hard problem feature, resolving with an exact method becomes impossible, and that is why we choose the simulation, as an empirical approach, to evaluate the performance. ARENA is used like a simulator.

Keywords: *Scheduling, Simulation, collaborative approach, multi agent systems, Communication Protocol, On-line Scheduling, Parallel Identical Resources, Makespan.*

Table des matières :

Liste des annexes :

Liste des figures :

Liste des tableaux :

Liste des abréviations :

SPT: Shortest Processing Time
LPT: Longest Processing Time
FIFO: First in First out
EDD: Earliest deadline
PSE : Procédures par Séparation et Évaluation
PL : La Programmation Linéaire
PLNE : La Programmation Linéaire en nombres entiers
PD : La Programmation Dynamique
NP: Non polynomial
IA : Intelligence Artificielle
IAD : Intelligence Artificielle Distribuée
SMA : Systèmes Multi-Agents
BDI: Beliefs, Desires, Intentions
AOP: Agent Oriented Programming
KQML: Knowledge Query Manipulation Language
FIPA: Foundation for Intelligent Physical Agents
SdP: Système de production
VBA: Visual Basic for Application
GNA : Générateur de nombres aléatoires
MaJ : Mise à Jour
BS: Borne supérieure
FATCI : File d'attente des tâches candidates à l'insertion

Liste des symboles :

Symboles	Description
IndexP	Indice de la tâche programmable
IndexA	Indice de la tâche aléatoire
r_i	La date de disponibilité de la tâche i
p_i	La durée opératoire de la tâche i
d_i	La date échue de la tâche i
ta_i	Le début au plus tôt de la tâche i
tb_i	Le début au plus tard de la tâche i
w_i	Le poids de la tâche i
C_{max}	La durée totale de l'ordonnancement
F_{max}	La plus grande durée de séjour
$\bar{F}$	La durée moyenne de séjour
$\bar{F}_w$	La durée de séjour moyenne pondérée
NQ(i)	L'état de la file d'attente numéro i
NR(i)	L'état de la ressource numéro i
TNOW	Run Current Time
mach	Le numéro de la machine ∈ {0, 1, 2, 3}
PA	La séquence admissible des tâches programmables
NTAE	Le nombre de tâches aléatoires exécutées

Introduction générale et problématique :

Pour rester compétitives, les entreprises manufacturières sont contraintes d'améliorer leur pilotage, tant au niveau stratégique, pour s'adapter aux progrès de la technologie ou suivre les évolutions du marché, qu'au niveau opérationnel, pour réagir face aux aléas. Au niveau stratégique, on retrouve, entre autres, les politiques de gestion de production qui sont traduites au niveau tactique par la planification de production. Pour qu'au niveau opérationnel, l'ordonnancement de production prenne la suite.

On retrouve aussi l'ordonnancement dans d'autres domaines d'application. On le trouve, ainsi, dans la gestion des projets, la gestion des lignes d'assemblage, les systèmes embarqués et les systèmes distribués, les systèmes d'exploitation des ordinateurs, etc.

L'ordonnancement en temps-réel, se distingue parmi les solutions pour la gestion de l'aléa au niveau opérationnel. Ceci, par le fait de sa capacité à s'adapter aux différentes architectures des systèmes de production, que cela soit : dans un environnement à une machine, plusieurs machines en parallèle, job shop, open shop,…

Notre problématique traite de l'ordonnancement en temps-réel sur deux ressources identiques en parallèle.

Il existe un nombre précis de tâches à exécuter sur ces ressources (les tâches programmables) selon l'ordre de leurs arrivées. Au cours de cette exécution, d'autres tâches apparaissent aléatoirement (les tâches aléatoires). L'exécution de ces tâches est possible si elles n'induisent pas de retard sur le traitement des tâches programmables.

L'objectif est de minimiser la durée totale d'ordonnancement, Makespan, ainsi que la maximisation du nombre de tâches aléatoires exécutées, sous contraintes de dates de disponibilité des tâches, contraintes disjonctives des ressources et dates échues dont il est impératif de respecter, ainsi que des durées opératoires.

La communauté scientifique considère le problème d'ordonnancement sur deux ressources identiques en parallèle, NP-Difficile. La résolution de ce problème consiste à développer et utiliser des approches heuristiques.

De la même façon, certains problèmes complexes n'ont pas d'algorithme de résolution précis et connu, permettant d'obtenir une solution convenable. Concevoir un programme permettant de résoudre ce genre de problèmes est alors difficile et fastidieux. Généralement, le concepteur explore l'espace des fonctions solutions possibles jusqu'à en trouver une, acceptable. Bien que certaines techniques permettent de réduire l'espace de recherche, la complexité des algorithmes standards pour ces problèmes augmente exponentiellement avec la taille des données. Ces algorithmes deviennent alors insuffisants.

Un travail a été effectué pour le traitement du problème d'ordonnancement temps-réel dans le cas d'une seule ressource [Our 01]. Les résultats obtenus ont été adaptés au cas de ressources identiques en parallèle dans un autre travail [Bou 07].

Notre travail consiste, dans un premier temps, à valider les résultats obtenus dans [Bou 07]. Dans un deuxième temps, nous développerons une approche collaborative, pour la résolution du problème d'ordonnancement temps-réel sur deux ressources identiques en parallèle, en vu de permettre l'ordonnancement d'un plus grand nombre de tâches aléatoires en respectant les délais d'exécution de toutes les tâches.

Afin de répondre à ces objectifs, nous avons utilisé les systèmes multi-agents (SMA). Inspirés des systèmes naturels complexes, tels que les fourmilières, ces systèmes sont composés d'une multitude d'agents logiciels plus ou moins simples interagissant afin d'accomplir une tâche commune. Les relations entre les différents agents du système sont dynamiques et aucun contrôle central ne supervise l'exécution globale. Ainsi, la fonctionnalité globale est distribuée et dépend essentiellement de l'organisation des agents au sein du système établi par les relations entre les agents. La possibilité de changer cette organisation de façon autonome permet alors d'obtenir un système pouvant changer sa fonctionnalité globale au cours du temps.

Notre mémoire se compose de quatre chapitres :

- Le premier chapitre est consacré à la description de la problématique étudiée dans le cadre de ce travail. Pour cela, la première partie de ce chapitre, a pour objectif de préciser les concepts fondamentaux, nécessaires à la bonne description de notre problématique, qui se fera dans la seconde partie de ce chapitre. Pour clore ce chapitre, on présentera les différentes méthodes de résolution.

- Pour le deuxième chapitre, dans un premier temps, nous allons présenter l'approche de résolution que nous avons retenue, pour la résolution du problème, à savoir, l'approche collaborative. Dans un deuxième temps, nous présenterons la méthode retenue pour l'évaluation des performances des différentes méthodes de résolution, à savoir, la simulation, telle que, la première partie de cette section, sera consacrée au développement d'un certain nombre de concepts de base essentiels pour la mise en place de la simulation. La seconde partie, portera sur le langage de simulation ARENA.

- Le troisième chapitre est réservé à la présentation des modèles développés pour la simulation. En première partie, nous présenterons les modèles qu'on a réalisés sur ARENA pour la simulation des différentes approches de résolution proposées dans [Bou 07]. Dans la seconde partie, on décrira l'approche collaborative qu'on a développée, puis présenter le modèle ARENA qui reprend son principe.

- Dans le quatrième chapitre, on mettra en place un plan d'expérimentation, afin de procéder à l'évaluation des performances de toutes les approches présentées dans ce mémoire et de sélectionner la meilleure méthode pour la résolution d'un problème d'ordonnancement en temps-réel sur deux ressources identiques en parallèle.

CHAPITRE I : Généralités sur l'ordonnancement et présentation de la problématique

I. Introduction

II. Position du problème d'ordonnancement

1. Description de la gestion de la production
2. Planification de production
3. Ordonnancement de production

III. Généralités sur l'ordonnancement

1. Définitions
2. Éléments d'un problème d'ordonnancement
3. Classification des problèmes d'ordonnancement
4. Les méthodes de résolution des problèmes d'ordonnancement
   A. Les méthodes exactes
   B. Les méthodes approchées

IV. L'ordonnancement sur des ressources en parallèle

1. Définition
2. Les algorithmes d'ordonnancement des tâches dans un environnement à ressources parallèles
   A. Ordonnancement local et ordonnancement global
   B. Ordonnancement statique et ordonnancement dynamique
   C. Ordonnancement centralisé et ordonnancement distribué
   D. Degré de coopération
   E. Régulation de charge
3. Ordonnancement sur machines identiques parallèles
4. Ordonnancement temps-réel
   A. Caractéristiques des systèmes temps réel
   B. Problèmes d'ordonnancement temps réel

V. Le problème d'ordonnancement sur machines parallèles identiques en temps-réel

1. Les hypothèses
2. Caractéristiques des tâches programmables
3. Caractéristiques des tâches aléatoires
4. Les différentes approches de résolution proposées dans [Bou 07]

VI. Conclusion

I. Introduction :

Nous avons consacré ce chapitre à la présentation de l'état de l'art sur l'ordonnancement dans sa globalité, puis, à la présentation de la problématique traitée dans le cadre de ce travail.

Pour cela, ce chapitre est structuré comme suit : la première section nous permet de localiser le problème d'ordonnancement dans un système de production. Dans la deuxième section, nous développerons les notions élémentaires relatives à l'ordonnancement, afin de pouvoir entamer la description du problème étudié. Dans la troisième section, nous présenterons les définitions et les notions de base de l'ordonnancement en parallèle en temps réel. Dans la dernière section, nous décrirons la problématique sujette à ce travail, puis, les différentes méthodes de résolution existantes pour le problème d'ordonnancement sur des ressources identiques parallèles.

II. Position du problème d'ordonnancement:

Il est nécessaire de localiser le problème d'ordonnancement, dans un système de production, ça sera l'objet de cette partie.

La production est une transformation de ressources appartenant à un système productif et conduisant à la création de biens ou de services. La gestion de production a pour objet la recherche d'une organisation efficace de la production de biens ou de services [Gia 03].

1. Description de la gestion de la production :

Un système de production est un processus d'addition de valeurs à des biens ou à des services répondant à des impératifs de qualité, de prix, de quantité et de délai. Dans un système de production, la partie la plus importante est le processus de production, qui est formé par un ensemble d'actions élémentaires telles que la production proprement dite, le transport, le stockage, etc.

En général, les principaux objectifs d'une gestion de production efficace peuvent être considérés de la manière suivante :

- Respecter les délais de commandes : il ne fait aucun doute que cet objectif est prioritaire vis à vis des pénalités de retard. Ainsi, il est nécessaire pour une entreprise de fournir un bien aux clients avec un retard le plus faible possible ;
- Minimiser les stocks intermédiaires pour réduire les coûts de fonctionnement de l'entreprise, surtout le coût de stockage ;
- Optimiser l'utilisation des moyens : il faut, en particulier, minimiser la charge des moyens les plus coûteux et détecter la présence de goulots d'étranglement limitant le débit de sortie des produits ;
- Améliorer l'efficacité du système logistique et ainsi accélérer le processus de production.

Parmi les différents sous-systèmes de la gestion de production, on s'intéresse plus particulièrement, à la fonction d'ordonnancement qui est la suite logique de la fonction planification de production.

2. Planification de production :

Avec une définition généralisée, la planification de production se compose de trois niveaux de décision : le niveau stratégique, le niveau tactique et le niveau opérationnel, pour décider «Qui, Quoi et Comment produire ?» (i.e. : quel type de produit il faut produire, déterminer la combinaison optimale des ressources en entrée, comment affecter et ordonnancer les tâches aux ateliers).

La définition de la planification de production, ne consiste qu'à trouver l'affectation optimale des tâches aux ateliers.

Ainsi, la planification de production vise à affecter un ensemble de tâches aux centres de production tout en respectant une ou plusieurs contraintes des ressources importantes. L'objectif de ce problème est normalement de minimiser le coût total de la fabrication sans se préoccuper comment et en particulier dans quel ordre ces tâches sont effectuées par ces centres de production. Cette dernière fonction est remplie par le niveau d'ordonnancement qui s'occupe d'un planning plus détaillé des produits [Hil 80] et [Ros 75].

3. Ordonnancement de production :

Dès que nous affectons les tâches aux ateliers pour résoudre le problème de planification, un problème d'ordonnancement de production devra aussi être résolu pour définir où et à quel moment précis un certain nombre de tâches, correspondant aux fabrications des objets ou aux fournitures des prestations de service, doivent être réalisées dans les centres de production. Chaque tâche se décompose en un certain nombre d'opérations dans un centre de production, étant aussi bien une usine, un département de production, un atelier, un groupe de machines ou une machine.

III. Généralités sur l'ordonnancement :

Cette section est consacrée à la fonction ordonnancement d'un système de production. Nous aborderons respectivement : la définition de l'ordonnancement, ses éléments, la classification des problèmes d'ordonnancement et les méthodes de leur résolution.

1. Définitions :

L'ordonnancement est la détermination conjointe des dates d'exécution d'un ensemble d'opérations et des ressources mobilisées dans cette exécution [Gia 03].

L'ordonnancement détermine d'abord les priorités de passage des travaux sur les ateliers (séquencement) et les différentes affectations temporelles des ressources. En fait, cette sous fonction est constituée d'une partie séquencement et d'une autre affectation. L'horizon peut être le court ou le moyen terme.

Les domaines d'application des techniques d'ordonnancement sont variés. Nous les trouvons dans :

- La gestion des projets ;
- La gestion des ateliers de production et les lignes d'assemblage ;

- Les systèmes embarqués et les systèmes distribués ;
- Les systèmes d'exploitation des ordinateurs, etc.

2. Éléments d'un problème d'ordonnancement :

Tous les problèmes d'ordonnancement partagent un ensemble d'éléments, dont l'état et les propriétés de ces derniers caractérisent le problème en question. Les éléments principaux sont :

a. Les tâches :

Une tâche ou « task » en anglais, est l'entité de base qui peut être une pièce mécanique, une personne (client), une requête en informatique, etc. Elle se caractérise par sa date de disponibilité (r : release date), sa durée opératoire (p : process time) et son délai (d : due date) et sa gamme[1]. Elle peut être préemptive, et a donc la possibilité de migrer d'une ressource à une autre.

En informatique, les tâches sont préemptives, leurs codes sont dupliqués sur tous les processeurs disponibles, alors que leurs contextes (données ou registre du processeur) peuvent être transférés d'un processeur à un autre selon l'ordonnancement arrêté.

L'ordonnancement d'atelier manufacturier quant à lui, relève en général du cas non préemptif.

b. Les ressources :

Une ressource est un moyen technique ou humain requis pour la réalisation d'une tâche. Une ressource a une capacité et un temps de préparation. Elle est renouvelable (ouvrier, engin…) ou consommable (pièces de rechange, carburant…).

Elle peut également être :

- Disjonctive (non partageable) : la machine ne peut traiter qu'une tâche à la fois. De même une tâche ne peut être exécutée que sur une seule machine à la fois. L'exécution des tâches d'un même travail (préemptif) est séquentielle et non parallèle.
- Cumulative (partageable) : la machine peut traiter plusieurs tâches à la fois en fonction de sa capacité. Et les tâches d'un même travail (préemptif) peuvent être exécutées en parallèle sur plusieurs ressources.

c. Les contraintes :

Des contraintes de différents types conditionnent l'admissibilité d'un ordonnancement :

- Contraintes temporelles : respect d'un début au plus tôt, d'une fin au plus tard, d'une date échue…;

[1] une tâche se caractérise par un ensemble d'opérations (ordonnancées ou pas), qu'elle doit subir

- Contraintes d'antériorité : respect de la gamme ou de la cohérence technologique (durées minimales ou maximales entre tâches) ;

- Contraintes de ressources :
  - Disjonctives : les tâches doivent être effectuées l'une après l'autre ;
  - Cumulatives : les besoins en ressources ne doivent pas dépasser les capacités des ressources.

d. Les objectifs et les critères d'un ordonnancement :

Chaque objectif d'un ordonnancement, a un ensemble de critères, permettant son atteinte. Soit, à titre d'exemple, à minimiser les encours de fabrication. Les critères permettant la réalisation de cet objectif sont la minimisation de:

- La plus grande durée de séjour ($F_{\max}$) ;
- La durée moyenne de séjour ($\bar{F}$) ;
- La durée de séjour moyenne pondérée ($\bar{F}_{w}$).

Les méthodes de résolution usuelles considèrent un seul critère à la fois, or les problèmes industriels sont plus complexes. Un responsable d'atelier est souvent intéressé par un ensemble d'objectifs : minimiser le temps total d'exécution de toutes les tâches, tout en minimisant l'oisiveté des ressources coûteuses. De ce fait, les problèmes d'ordonnancement industriels sont considérés comme des problèmes multicritères. Ces problèmes sont abordés au moyen de la notion de Pareto-dominance[1]. Ainsi, le décideur pourra choisir entre un ensemble de solutions dominantes (front de Pareto).

3. Classification des problèmes d'ordonnancement :

Les problèmes d'ordonnancement peuvent être classifiés selon plusieurs critères. Selon que la production soit unitaire (un projet), de masse ou continue. Les approches classiques de gestion de production dissocient l'ordonnancement de projet et l'ordonnancement des fabrications [Gaz 03].

Les problèmes d'ordonnancement sont classés en fonction du type de l'atelier, selon la disposition des ressources, leurs nombres et leurs fonctions.

On distingue principalement:

- Les ateliers à cheminement unique ou « Flow shop » (figure I.1) ;
- Les ateliers à cheminement multiple ou « Job shop » (figure I.2) ;
- Les ateliers à cheminement libre ou « Open shop » ;
- Les ateliers à une seule machine;
- Les ateliers à machines parallèles (figure I.3).

Ajoutez à cela, les ateliers flow shop hybrides, le problème flow shop de permutation (figure I.1), les ateliers dotés d'un système de transport critique, etc.

[1] Une solution est Pareto-dominante si aucune autre solution n'améliore la satisfaction d'un critère sans dégrader la satisfaction d'un autre [Gra 05].

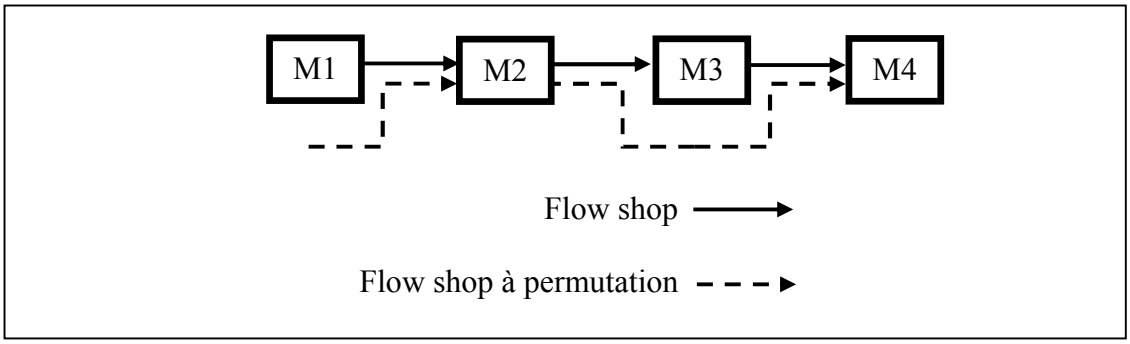


Figure I.1: flow shop

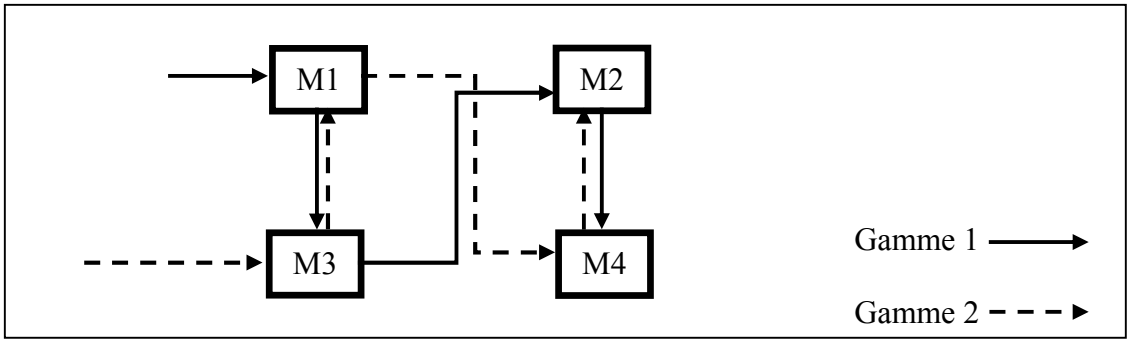


Figure I.2: job shop

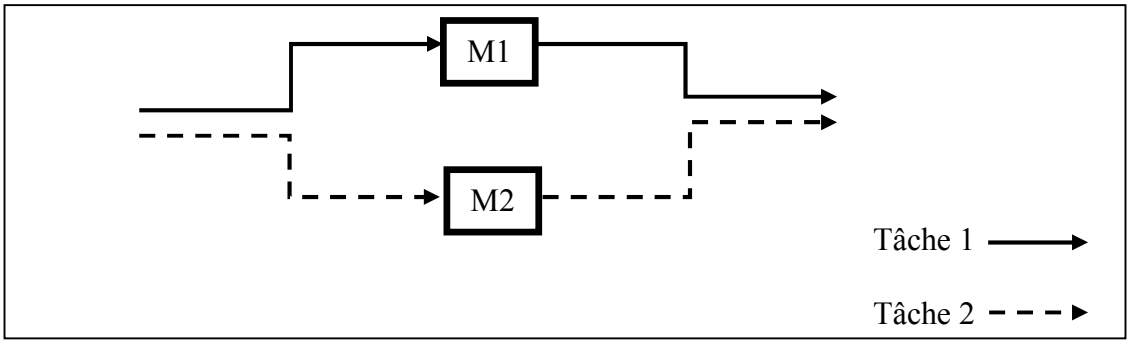


Figure I.3: machines parallèles

Les propriétés des tâches à exécuter (disponibilité, délais…), ainsi que les propriétés des ressources (capacité, disjonction…) ; sont aussi à prendre en compte dans la classification des problèmes d'ordonnancement.

Parmi les notations proposées par la communauté scientifique, et prenant en compte tous les éléments d'un ordonnancement donné, nous citons les classifications suivantes:

- Conway, Maxwell et Miller (1967) ;
- Graham & al. (1979) ;
- Lawler & al. étendue par Vignier & al. (1995).

Nous nous contentons de la notation de Graham & al. , du fait de sa prise en considération de tous les paramètres du problème étudié et de son applicabilité à tous les types de problèmes d'ordonnancement. Cette notation comprend trois paramètres $\alpha/\beta/\gamma$. Les valeurs que peut prendre chaque paramètre sont consignées dans le tableau en annexe I.

4. Les méthodes de résolution des problèmes d'ordonnancement :

Elles sont classées en deux grandes catégories : les méthodes exactes et les méthodes approchées.

A. Les méthodes exactes :

Ces méthodes sont dites exactes du fait qu'elles convergent vers une solution optimale du problème considéré. Par ailleurs, leur complexité NP-difficile[1] les rend très couteuses en matière de temps et d'allocation de mémoire. Ces derniers sont des fonctions exponentielles de la taille du problème. Ce qui les réserve aux problèmes de petite ou moyenne taille. En plus, et si nous choisissons mal les conditions initiales, ces algorithmes présentent un grand risque de convergence vers un optimum local.
Parmi ces méthodes, nous distinguons principalement :

i. La méthode de séparation et d'évaluation (Branch and Bound) :

La méthode procède par l'exploration de toutes les branches de l'arborescence des solutions, tout en éliminant les branches présentant à priori une borne inférieure supérieure à la borne supérieure déjà calculée. Etant une méthode exacte, le nombre de branches à explorer croit exponentiellement avec la taille du problème. Ce qui fait que les travaux concernant cette technique sont plutôt orientés vers la réduction du nombre de branches explorables par des heuristiques. L'heuristique de Jackson portant sur la priorité d'une tâche réalisant exhaustivement les deux règles de priorité : SPT (Shortest Processing Time) et EDD (Earliest due date).

ii. La programmation linéaire :

Cette méthode concerne la formalisation du problème sous un problème de programmation linéaire. La fonction objective porte sur le(s) critère(s) à optimiser, et les contraintes sur les contraintes temporelles de l'ordonnancement. Les variables seront les temps de début de chaque tâche. Le problème doit ensuite être linéarisé et résolu par les méthodes de la programmation linéaire (Simplexe[2], les points intérieurs[3], PLNE[4], etc.).

iii. La programmation dynamique :

La méthode consiste à décomposer le problème en sous problèmes, de résoudre les sous problèmes considérant les tailles de bas en haut, en partant à chaque fois de la solution optimale du problème précédent, jusqu'à arriver à la taille du problème en question.

[1] Le temps de convergence de l'algorithme est une fonction exponentielle de la taille du problème
[2,3] méthodes pour résoudre les problèmes d'optimisation d'une fonction linéaire avec des contraintes linéaires
[4] Programmation linéaire en nombres entiers : les variables du problème linéaire sont entières

B. Les méthodes approchées :

La NP-complétude des problèmes d'ordonnancement, a conduit au recours à des méthodes présentant des solutions non-optimales, mais plus économes en termes de temps et d'allocation de mémoire. Ces méthodes sont basées sur la déduction de complexité ou sur l'équivalence entre les problèmes. Cette déduction porte sur trois axes :

- Selon les instances figure I.4 ;
- Selon les configurations figure I.4 ;
- Selon les critères d'optimisation figure I.5.

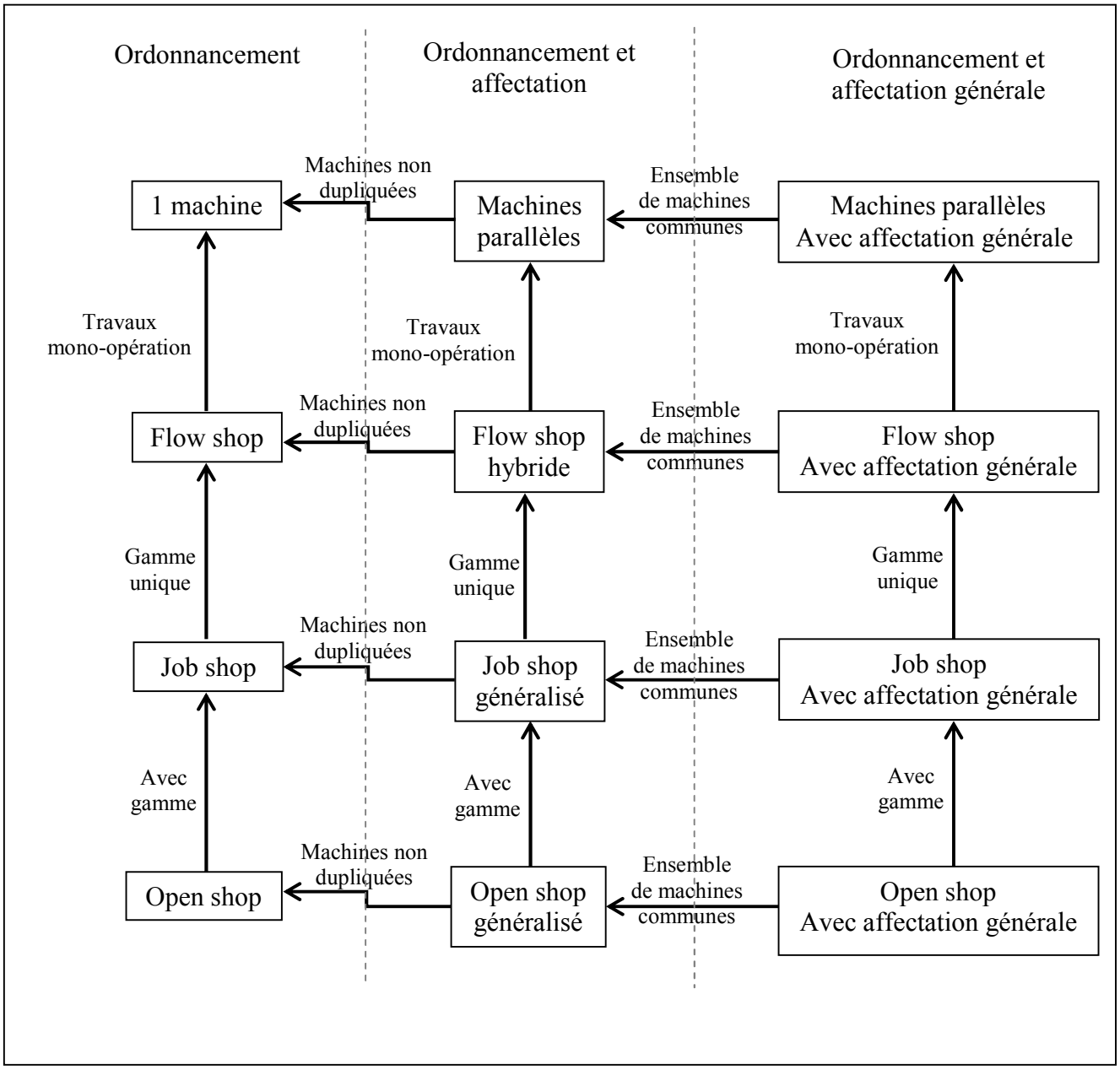


Figure I.4 : Typologie des problèmes d'ordonnancement [Bil 99]

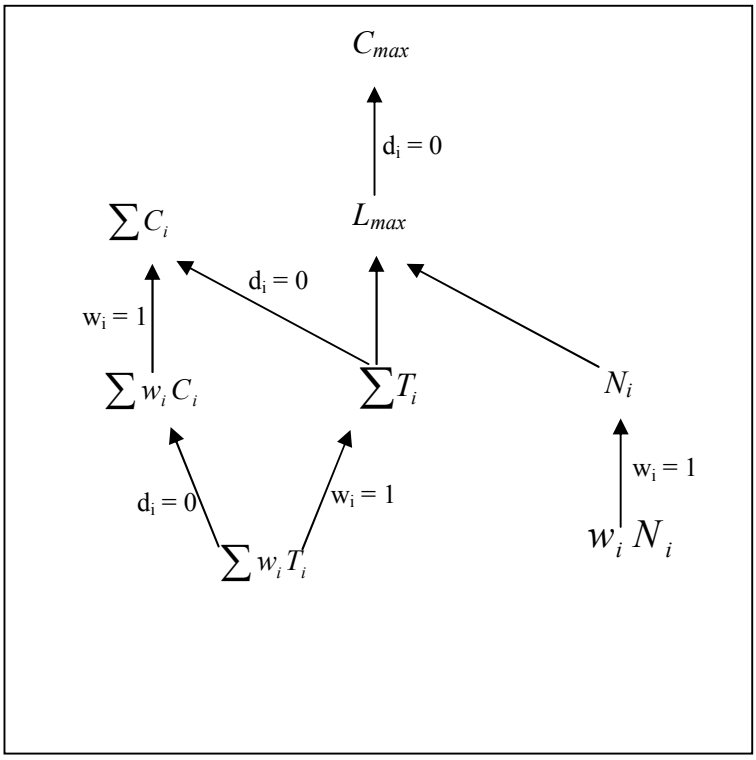


Figure I.5 : Déduction sur les critères d'optimisation

Les méthodes approchées sont classées selon [Car 88] en trois classes principales :

i. Les algorithmes gloutons :

Cette catégorie de méthodes procède à la construction d'une solution globale à partir d'une solution partielle complétée au fur et à mesure de la résolution. Ces algorithmes utilisent de simples règles de priorité, de type SPT (Shortest Processing Time), EDD (Earliest Due Date).

ii. Les méthodes de recherche locale (tabou, recuit simulé, algorithmes génétiques, etc.) :

Elles consistent à explorer l'espace des solutions d'ordonnancement en partant d'une solution initiale complète de laquelle on extrait une solution voisine. Si cette nouvelle solution est meilleure que la solution de référence, cette dernière devient la solution de référence. Le déplacement dans l'espace s'effectue à l'aide d'une fonction de voisinage.

iii. Les méthodes de recherche arborescente tronquée :

Ayant le même principe que les méthodes de séparation et d'évaluation (Branch and Bound), ces méthodes procèdent à la décomposition du problème, afin de restreindre l'espace de recherche des solutions.

IV. L'ordonnancement sur des ressources en parallèle :

Nous voulons dans cette section définir la vision du problème d'ordonnancement sur des ressources en parallèle, en commençant par les différentes définitions du problème en général, pour aboutir au final, à la description du problème d'ordonnancement en temps-réel sur des ressources identiques en parallèle que nous traitons. Ce problème, selon Graham, est noté : $P2 / r_j, \Delta_j / C_{max}$.

1. Définition :

Si nous disposons de plusieurs machines en parallèle pour exécuter tous les jobs, on parle de problème à machines parallèles.

Les machines peuvent être de différentes natures :

- Elles peuvent être interchangeables, on parle alors de machines parallèles identiques;
- Elles peuvent être corrélées entre elles, un seul paramètre de vitesse les faisant différer, elles sont dites machines uniformes;
- Elles peuvent être indépendantes les unes des autres. Cela signifie que l'exécution d'un job peut être plus longue que celle d'un autre job sur une machine, tandis que la situation est inversée sur une autre machine.

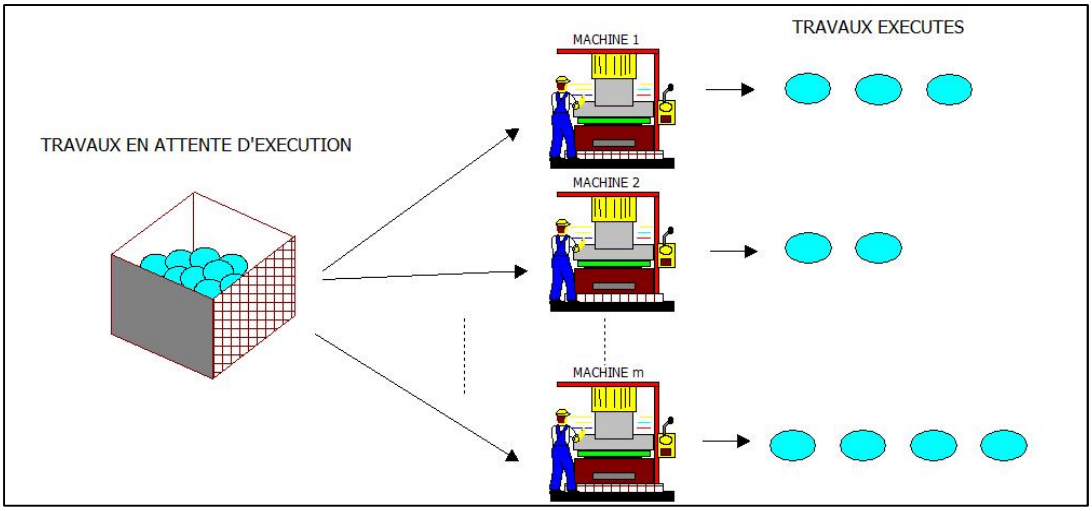


Figure I.6 : représentation d'une cellule d'assemblage à machines parallèles

2. Les algorithmes d'ordonnancement des tâches dans un environnement à ressources parallèles :

Pour la classification des algorithmes d'ordonnancement des tâches, dans un environnement à ressources parallèles, nous avons choisi la classification donnée dans [Cas 88] qui reprend un certain nombre de concepts que nous aurons à développer par la suite, mais aussi, peut-être que c'est celle qui est la plus souvent citée dans la littérature. Dans [Jac 96] une discussion plus approfondie sur cette classification est présentée et une proposition d'extension, pour cette dernière, a été faite. Nous pouvons citer d'autres classifications, notamment celles proposées par G. Bernard dans [Ber 91] et Talbi dans [Tal 97].

A. Ordonnancement local et ordonnancement global :

Une première distinction entre les politiques d'ordonnancement est basée sur l'identification de leurs champs d'action. Ainsi, un ordonnancement **local** est responsable de la gestion d'accès des tâches en attente d'exécution à la ressource.
L'attribution d'une tâche à une ressource est du ressort d'un ordonnancement **global**.

B. Ordonnancement statique et ordonnancement dynamique :

De nombreuses stratégies peuvent être employées par l'ordonnancement global. Selon le moment où elles sont appliquées, une stratégie est dite **statique** ou **dynamique** [Lis 91].
Les algorithmes statiques sont applicables dans les cas où il est possible d'obtenir la description de toutes les tâches à exécuter avant le démarrage. Une analyse de cette description permet de déterminer à l'avance la ressource sur laquelle et la date à laquelle, chaque tâche sera exécutée.
Dans le cas où la description des tâches n'est connue que lors de leur apparition, l'ordonnancement applique des stratégies dynamiques. Ce type d'algorithme prévoit une activation à chaque changement d'état de l'environnement d'exécution, par exemple au moment de la création ou de la terminaison d'une tâche. Bien que les algorithmes dynamiques ne puissent pas analyser la description complète du problème, ils peuvent suivre l'évolution des taux d'utilisation des ressources de la machine par un, voir plusieurs indices de charge. Dans les stratégies statiques ces indices de charge ne sont pas disponibles, cependant elles peuvent considérer, lors de l'ordonnancement, une description de l'évolution du système.

C. Ordonnancement centralisé et ordonnancement distribué :

Selon la nature de la politique d'ordonnancement global appliquée, les décisions sont implantées de façon **centralisée** ou **distribuée**. Dans une politique **centralisée** les décisions de toutes les opérations d'ordonnancement sont réalisées par un serveur central. Dans une politique **distribuée** au contraire, les opérations d'ordonnancement sont réalisées par toutes les ressources à leur niveau.

D. Degré de coopération :

Le niveau de **coopération** entre les ressources permet d'identifier le degré d'indépendance de chaque ressource dans la prise de décision d'une stratégie dynamique distribuée. Un faible degré de coopération indique que chaque ressource prend les décisions d'ordonnancement en tenant compte seulement de sa propre charge. Tandis que dans une stratégie coopérative, les décisions prises par une ressource prennent en compte l'impact de la réalisation d'une opération d'ordonnancement sur la charge globale de la machine.

E. Régulation de charge :

Réguler la charge d'un programme d'ordonnancement, implique de distribuer entre les ressources en parallèle le travail nécessaire pour l'exécution de toutes les tâches (i.e. : la répartition des tâches sur les ressources selon la charge de travail nécessaire). Nous avons deux approches légèrement différentes pour la régulation de charge : l'**équilibrage de charge** (line balancing) qui prévoit une distribution équitable du travail entre les ressources, et la **répartition de charge** qui cherche à éviter que des ressources soient inactives pendant l'exécution.

3. Ordonnancement sur machines identiques parallèles:

Dans le cas d'un atelier de m machines identiques parallèles, chaque tâche est constituée d'une seule opération qui peut être exécutée sur n'importe quelle machine. Ce type d'atelier a attiré l'attention de beaucoup de chercheurs en ordonnancement par son intérêt pratique.

Dans le cas de machines identiques parallèles sans contraintes de temps de préparation, le problème de minimisation du Makespan est largement étudié. Parmi les travaux qui ont étudié ce problème on peut citer, Garey et Johnson [Gar 78] qui ont démontré que le problème P‖Cmax est fortement *NP*-difficile et McNaughton [McN 59] qui a prouvé que la préemption rend ce problème facilement solvable. Grâce à cette relaxation, McNaughton a proposé une borne inférieure pour ce problème qui est :

$$C^*_{max} = max\left(\frac{1}{m}\sum_{i=1}^{n} p_i\,, max(p_1, \dots, p_n)\right)$$

m étant le nombre de machines, p_i la durée opératoire de la tâche *i* et *n* le nombre des tâches.

Pour le même problème, l'heuristique basée sur la règle *LPT* (Longest Processing Time) est une heuristique simple et efficace. Sa performance au pire cas a été évaluée par Graham dans [Gra 69] égale à :

$$\frac{C_{max}(LPT)}{C^*_{max}} \leq \frac{4}{3} - \frac{1}{3m}$$

Avec C^*_{max} la valeur de la solution optimale, $C_{\max}(LPT)$ la solution obtenue avec l'heuristique *LPT* et *m* le nombre de machines.

Le problème $P2||\sum w_jC_j$ a été démontré NP-difficile par Bruno *et al*. dans [Bru 74] bien que le problème $P||\sum C_j$ soit solvable en *O(NlogN)* par un algorithme de liste basé sur la règle *SPT* (Shortest Processing Time).

Pour le problème $P||\sum w_jC_j$, plusieurs algorithmes de séparation et d'évaluation ont été proposés comme ceux d'Elmaghraby et Park [Elm 74], Belouadeh et Potts [Bel 94], Chen et Powell [Che 99] et Azizoglu et Kirca [Azi 99]. Des bornes inférieures ont été aussi développées par Webster [Web 95] et [Web 97]. Ces bornes sont basées sur l'idée de subdiviser les tâches en plusieurs mini-tâches et de les ordonnancer d'une manière optimale sous certaines conditions.

Concernant le problème de minimisation de Makespan avec prise en compte de temps de préparation dépendant de séquence, la plupart des méthodes sont des heuristiques. Le problème *P2|pmtn,si|Cmax* a été démontré *NP*-difficile par Monma et Potts [Mon 89].

Le problème de minimisation de la somme pondérée des dates d'achèvement n'a pas été beaucoup étudié. Webster et Azizoglu [Web 01] ont étudié le problème $P|s_i|\sum w_jC_j$ en élaborant deux algorithmes exacts dans le cas où *F* (le nombre de familles) et *m* (le nombre de machines) sont fixés, grâce à deux formulations dynamiques. Dans [Azi 03], Azizoglu et Webster ont examiné les différents algorithmes de la méthode de séparation et d'évaluation et ils ont proposé des bornes inférieures pour le problème $P|s_i|\sum w_jC_j$. Ces bornes sont inspirées de celles de Mason et Anderson dans [Mas 91] et Dunstall et *al*. [Dun 00] pour le problème $1|s_i|\sum w_jC_j$ et celles de Webster [Web 95] et Azizoglu et kirca [Azi 99] pour le problème

$P||\sum w_jC_j$. De même, Dunstall et Wirth [Dun 05a] ont fait une étude bibliographique des algorithmes de séparation et d'évaluation pour ce problème et ont proposé une nouvelle technique de séparation plus performante. Webster et Azizoglu [Web 01] ont proposé deux formulations dynamiques pour le problème $P|s_i|\sum w_jC_j$ et deux conditions d'optimalité impliquant la somme des poids et la date d'achèvement de la dernière tâche pour n'importe quelle machine. Dans [Dun 05b], Dunstall et Wirth proposent des heuristiques pour le même problème. Ces heuristiques commencent par l'affectation des groupes de tâches, constitués de l'ensemble des tâches de chaque famille. Ensuite, des opérations de division et de reséquencement de groupes de tâches sont répétées sur chaque machine.

4. Ordonnancement temps-réel :

Les systèmes temps-réels sont de plus en plus présents dans le quotidien : on les trouve dans l'aéronautique, le transport ferroviaire, les installations nucléaires, l'automobile, l'électroménager ou le multimédia. On désigne par temps-réel, toute application mettant en œuvre un système informatique dont le fonctionnement est conditionné par l'évolution dynamique de l'état d'un environnement qui lui est connecté et dont il doit contrôler le comportement [Ell 91]. Le rôle du système informatique est alors de suivre ou de piloter ce procédé en respectant des contraintes temporelles définies dans le cahier des charges de l'application. Également, dans [Sta 88], on qualifie de temps réel tout système informatique dont la correction du fonctionnement ne dépend pas seulement de l'exactitude logique des résultats qu'il fournit, mais surtout dépend de la date à laquelle ces résultats sont produits. Ainsi, le système ne traite plus uniquement des valeurs, mais des couples valeur et temps [Des 82]. Cette définition implique que la seule rapidité moyenne d'exécution ne conditionne pas la validité du système, mais des contraintes temporelles doivent être respectées (par exemples les échéances des traitements) sont relatives à un temps physique mesurable, et font partie de la spécification du système à implanter.

A. Caractéristiques des systèmes temps réel :

Dans le contexte des systèmes temps réel, les données ont une validité définie à la fois par le domaine de valeurs admises et par la durée de validité (temporelle) de celles-ci, qui dépend naturellement de l'échéance (date échue). Les données ont, ainsi, une durée d'existence limitée [Zaf 07]. Autrement dit, les systèmes de contrôle doivent respecter deux exigences, fonctionnelles et temporelles en parallèle.

Selon les contraintes temporelles, on distingue deux grandes catégories des systèmes temps réel :

- Systèmes temps réel à contraintes strictes (temps réel dur) : lorsque toutes les contraintes temporelles doivent être impérativement respectées. Le non respect des contraintes peut provoquer des conséquences catastrophiques [Par 91]. Les systèmes de contrôle de vol, systèmes de contrôle de station nucléaire, systèmes de contrôle de voies ferrées en sont des exemples.
- Systèmes temps réel à contraintes relatives (temps réel souple) : contrairement aux systèmes durs, le non respect des contraintes temporelles est toléré (acceptable) par le système, et sans que cela ait des conséquences catastrophiques [Che 99a], [Che 99b]. Par exemple des applications multimédias.
- Systèmes temps réel à contraintes mixtes : sont composés des tâches à contraintes strictes et des tâches à contraintes relatives [Gro 99].

B. Problèmes d'ordonnancement temps réel :

La théorie de l'ordonnancement des systèmes temps réel se focalise principalement sur deux problèmes :

- Le problème de faisabilité : étant donné les spécifications des tâches et les contraintes sur l'environnement d'ordonnancement (par exemple si les priorités sont fixes/dynamiques au niveau des tâches, si les préemptions sont admises, etc.), il s'agit de déterminer l'existence d'un ordonnancement qui satisfasse toutes les échéances ;
- Le problème d'ordonnançabilité en ligne : étant donné les spécifications des tâches (et des contraintes de l'environnement) supposées être faisables, il s'agit de déterminer un algorithme d'ordonnancement qui construit un ordonnancement qui satisfasse toutes les échéances.

Pour les problèmes en environnement parallèle, les tâches à exécuter sont distribuées sur les différentes ressources ; il faut alors, donner un ordre d'exécution des tâches (i.e. : il s'agit de résoudre des problèmes d'ordonnancement).

Dans le cadre des systèmes temps réel, le respect des contraintes de temps, généralement des contraintes de dates au plus tard (dates échues), est imposé par l'environnement. Le problème de l'ordonnancement temps réel est donc de donner un ordre aux tâches qui respectent ces contraintes de temps.

D'une autre façon, un mécanisme d'ordonnancement temps réel a pour rôle de mettre en œuvre l'exécution d'un ensemble de tâches sur un ensemble de ressources. Dans un modèle dynamique, ceci s'accompagne d'une gestion dans le temps où le mécanisme d'ordonnancement a pour charge de suivre cette exécution et de la modifier en fonction des précisions obtenues sur certains paramètres et des nouvelles tâches créées.

En univers distribué, un mécanisme d'ordonnancement a deux fonctions principales : une fonction de gestion locale et une de gestion globale. La première est la gestion temporelle. La seconde concerne les ressources et consiste à allouer les tâches créées dynamiquement aux ressources, c'est une gestion spatiale.

V. Le problème d'ordonnancement sur machines parallèles identiques en temps-réel :

Le problème posé consiste à ordonnancer sur plusieurs ressources identiques parallèles, un certain nombre de tâches arrivant aléatoirement, sachant, que le système dispose d'un certain nombre de tâches, dites programmables, à traiter. Le critère à optimiser est la durée totale de l'ordonnancement (Makespan) sous des contraintes liées aux ressources, aux intervalles temporels et aux tâches elles-mêmes, ainsi que le nombre de tâches aléatoires ordonnancées (exécutées).

Un travail a été effectué pour le traitement du problème d'ordonnancement temps-réel dans le cas d'une seule ressource [Our 01]. Les résultats obtenus ont été adaptés au cas de ressources identiques en parallèle dans un autre travail [Bou 07].

On s'intéresse dans cette section, au problème décrit dans [Bou 07]. Car ce problème, précisément, n'a pas encore été traité sauf dans ce dernier. Notre travail, aussi, traitera de ce problème en particulier. Ainsi, nous présentons les caractéristiques et les paramètres de ce problème, mais aussi, les différentes approches de résolution qui ont été proposées.

1. Les hypothèses :

Les hypothèses retenues sont les suivantes :

- Chaque tâche est exécutée sur une ressource seulement.
- Toutes les ressources sont identiques.
- Les tâches sont non-préemptives.
- Il y'a deux types de tâche : programmables et aléatoires.
- Il n'y a pas d'interdépendance entre les tâches.
- Toutes les ressources sont disponibles à l'instant zéro.
- Le temps de transport entre les ressources est négligeable.
- Les temps de préparation des ressources et des tâches sont négligeables.
- Les tâches ne sont pas nécessairement exécutées dés leurs apparitions.
- Les capacités de stockage des files d'attente sont illimitées.
- Les pannes ne sont pas considérées dans le cadre du problème posé.

2. Caractéristiques des tâches programmables:

A l'instant zéro, les données suivantes caractérisant chaque tâche i, sont supposées connues :

- r_i : date de disponibilité de la tâche i,
- p_i : durée opératoire de la tâche i,
- d_i : date échue de la tâche i,

Toutes les tâches programmables sont supposées exécutables dans leurs intervalles d'exécution $[r_i, d_i]$.

3. Caractéristiques des tâches aléatoires:

Chaque tâche aléatoire j est caractérisée par les éléments suivants :

- r_j : date de disponibilité de la tâche,
- p_j : durée opératoire de la tâche,
- d_j : date échue de la tâche.

Ceci étant, contrairement aux tâches programmables, ces informations ne sont connues qu'au moment de l'apparition de la tâche aléatoire (i.e. : au moment r_j pour la tâche j). Les différentes distributions régissant les dates de disponibilité, les durées opératoires et les dates échues ont été identifiées comme suit :

- Le processus d'arrivée des tâches aléatoires suit une loi de poisson avec un paramètre à déterminer (i.e. : la durée qui sépare l'arrivée de deux tâches suit une loi exponentielle), on note cette variable aléatoire : X ;
- On suppose que les durées opératoires suivent une loi triangulaire (Min, Mode, Max) ;
- Les dates échues sont aussi aléatoires et sont données par la variable aléatoire $D = X + C_t$, (C_t : est un paramètre du problème à déterminer).

Une tâche aléatoire est rejetée s'il n'est pas possible de l'exécuter dans son délai.

4. Les différentes approches de résolution proposées dans [Bou 07] :

Le problème d'ordonnancement temps-réel des tâches intervenant aléatoirement sur des ressources identiques en parallèle est supposé NP-Difficile au sens fort [Bou 07].

Dans le cas multi ressources, il a été démontré dans [Mok 83], que pour l'ordonnancement dynamique, aucun algorithme d'ordonnancement ne peut être optimal sans avoir connaissance complète à l'avance de toutes les échéances, de tous les temps d'exécution, de toutes les dates de démarrage des tâches. En pratique, des heuristiques sont utilisées pour résoudre ces problèmes.

La résolution de ce problème passe par deux phases. La première, concernant l'ordonnancement des tâches programmables, appelé "ordonnancement statique", puis la seconde, qui consiste à introduire les tâches aléatoires dans l'ordonnancement obtenu précédemment, appelé "ordonnancement dynamique".

1ère étape : Ordonnancement statique.

Il s'agit de répartir les tâches prévisionnelles sur les deux ressources selon une règle d'ordonnancement permettant d'exécuter les tâches dans leurs intervalles temporels à l'intérieur d'une plage de temps la plus faible possible. La limite supérieure de cette plage représente la borne supérieure qui doit être minimisée. Minimiser cette borne permet de minimiser la durée totale d'ordonnancement.

La règle d'ordonnancement choisie est basée sur l'ordre croissant des dates de disponibilité des tâches en respectant les délais.

Une fois la répartition achevée, deux problèmes d'ordonnancement à une ressource sont à résoudre. De là, considérer une ressource et trouver la séquence admissible PA pour l'exécution des tâches prévisionnelles qui **ne sera plus modifiée** (ultérieurement). La même procédure est à appliquer à l'autre ressource.

Toute tâche prévisionnelle disponible et non exécutée par une ressource est en attente d'exécution et stockée dans une file d'attente.

2ème étape : Ordonnancement dynamique.

Pour cette phase, on utilise le programme d'exécution de l'ordonnancement statique en insérant les tâches aléatoires parmi les tâches programmables en respectant les différentes contraintes (chaque tâche aléatoire doit être exécutée dans son intervalle temporel, l'ordonnancement d'une tâche aléatoire sur une ressource ne doit pas remettre en cause l'exécution de la tâche programmable, …).

Principalement, deux approches de résolutions ont été proposées par Bougchiche dans [Bou 07]. La première approche, consiste à stocker les tâches aléatoires dans une file d'attente commune aux deux ressources, pour cette approche, deux algorithmes ont été développés, à savoir, l'algorithme d'ordonnancement temps-réel au plus tard et l'algorithme

d'ordonnancement temps-réel au plus tôt. La deuxième approche, propose deux files d'attente pour les tâches aléatoires (pour chaque ressource, une file d'attente), pour cette approche, aussi, deux algorithmes ont été développés, à savoir, l'algorithme d'ordonnancement temps-réel au plus tard et l'algorithme de calcul de la marge disponible.

i. Algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente :

Cet algorithme donne la priorité à la tâche aléatoire dans l'utilisation des marges disponibles. La tâche aléatoire doit vérifier certaines conditions pour être ordonnancée et qui sont :

- L'intervalle temporel de la tâche aléatoire : une tâche dont la date échue n'est pas respectée (i.e. : $t_{now} + p_j \leq d_j$) est rejetée;
- L'intervalle d'exécution des tâches programmables : l'ordonnancement d'une tâche aléatoire induisant un retard sur l'exécution d'une tâche programmable, ordonnancée au plus tard (i.e. : $t_{now} + p_j \leq t_{bi}$) est rejetée.

L'algorithme proposé permet d'affecter les tâches programmables aux ressources selon un ordonnancement au plus tard.

Le fonctionnement de cet algorithme est décrit par un organigramme dans le chapitre III.

ii. Algorithme d'ordonnancement temps-réel au plus tôt dans le cas d'une file d'attente :

Cet algorithme donne la priorité à la tâche programmable dans l'utilisation des marges disponibles, ainsi, la tâche programmable est exécutée dès que l'instant courant est égal à la date de début au plus tôt de cette tâche $t_{ai} \geq t_{now}$.

Pour être exécutée, une tâche aléatoire doit vérifier les mêmes conditions que dans le premier cas, à savoir, l'intervalle temporel de son traitement et l'intervalle d'exécution des tâches programmables.

L'algorithme proposé permet d'affecter les tâches programmables aux ressources selon un ordonnancement au plus tôt.

Le fonctionnement de cet algorithme est décrit par un organigramme dans le chapitre III.

iii. Algorithme d'ordonnancement temps-réel au plus tard dans le cas de deux files d'attente :

Cet algorithme permet d'affecter les tâches aléatoires aux files d'attente des ressources en les insérant directement dans la position d'exécution. Par conséquent, à chaque apparition d'une tâche aléatoire, elle doit satisfaire des conditions pour pouvoir l'exécuter. Dès son affectation, elle a une position entre les tâches programmables.

Cet algorithme donne la priorité à la tâche aléatoire à chaque fois qu'un choix se présente (i.e. : lors de l'ordonnancement dès son arrivée ou chaque fois qu'une ressource est libre).

Pour être exécutée, une tâche aléatoire doit vérifier les mêmes conditions que dans les deux premiers cas, à savoir, l'intervalle temporel et l'intervalle d'exécution des tâches programmables.

Le fonctionnement de cet algorithme est décrit par un organigramme dans le chapitre III.

iv. Algorithme de calcul de la marge disponible dans le cas de deux files d'attente :

La démarche de résolution est constituée principalement de deux étapes :

- La première étape détermine le premier espace libre entre deux tâches programmables consécutives, pour insérer la tâche aléatoire intervenant A_j. D'où, une première position provisoire pour A_j est obtenue ;
- La seconde étape définit la position d'insertion de la tâche qui permet de maximiser sa chance d'acceptation et qui préserve au mieux l'avenir. L'idée, dans ce cas, consiste à avancer éventuellement certaines tâches programmées vers la droite pour insérer A_j plus tôt que sa position provisoire.

Cet algorithme, à l'instant courant, vérifie les contraintes sur l'intervalle temporel de la tâche aléatoire (i.e. : $t_{now} + p_j \leq d_j$) et sur la marge libre permettant d'insérer cette tâche qui doit être supérieure ou égale à sa durée opératoire.

Après étude de cet algorithme, son fonctionnement s'avère être le même que le premier (i.e. : l'algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente), car, ce qui est fait en temps réel dans le premier algorithme (i.e. : vérification de la tâche aléatoire optimale à exécuter sur la machine lors de sa libération), dans ce dernier algorithme, ce choix se fait au fur et à mesure que les tâches aléatoires apparaissent, mais à la libération de la ressource, on se retrouve avec la même tâche aléatoire optimale qui a été sélectionnée par le premier algorithme.

VI. Conclusion :

Dans ce chapitre, nous sommes partis des notions élémentaires pour arriver à la description des différentes approches de résolution proposées dans [Bou 07], en passant par la présentation de la problématique traitée par ce travail. Ce chapitre nous permettra d'appréhender la modélisation proposée dans le chapitre III.

Dans ce travail, nous nous proposons de donner une nouvelle approche pour sa résolution, et c'est l'approche collaborative qui a été retenue. Nous avons, par ailleurs, vu que ce problème est NP-difficile au sens fort, ce qui rend l'utilisation de toute méthode exact impossible, pour cette raison, nous avons opté pour une approche empirique moyennant la simulation afin d'évaluer la performance.

Ainsi, le prochain chapitre sera consacré à la présentation des généralités sur l'approche collaborative, mais aussi, des généralités sur la simulation.

CHAPITRE II : Etat de l'art de l'approche collaborative et de la simulation

I. Introduction
II. L'approche collaborative
  1. Systèmes d'ordonnancement distribués
  2. Systèmes multi agents
    A. L'agent
    C. L'interaction
    D. La communication
    E. L'organisation et les rôles
  3. Résolution par approche coopérative
    A. Point de vue général de la résolution coopérative
    B. Protocoles de rôles : point de vue de l'agent perturbé
    C. Protocoles de rôles : point de vue de l'agent coopérant
    D. Mise à jour des connaissances
    E. Les formes de coopération
III. La simulation
  1. Généralités sur la simulation
    A. Définition de la simulation
    B. Généralités sur la modélisation
  2. Présentation du logiciel ARENA 10
    A. Structure graphique
    B. Eléments d'un modèle de simulation
    C. Le générateur de nombres aléatoires (GNA)
    D. L'application Microsoft Visual Basic
IV. Conclusion

I. Introduction :

Dans la première section de ce chapitre, il est question de l'approche de distribution de l'ordonnancement sur les différents centres de décision (qui sont dans notre cas, les différentes machines). Ainsi, cette section est structurée comme suit : la première section présente les systèmes d'ordonnancement distribués, puis les systèmes multi agents, qui présentent une architecture particulièrement bien adaptée pour la mise en pratique des systèmes d'ordonnancement distribués. Par la suite, nous aborderons la mise en application d'une approche coopérative dans le cas général.

Nous avons vu que le problème étudié est NP-difficile au sens fort (dans [Bou 07]), ainsi, nous avons opté pour une approche empirique moyennant la simulation afin d'évaluer la performance des différentes approches de résolution. Donc la deuxième et dernière section de ce chapitre, est consacrée à l'approche empirique qu'est la simulation et qui sera utilisée, par la suite, pour évaluer la performance des différentes approches de résolution. Cette section débutera par, une description des généralités sur la simulation, pour l'acquisition des connaissances nécessaires à la compréhension du travail développé par la suite, puis, nous présenterons le langage de simulation retenu, à savoir, ARENA 10.

II. L'approche collaborative :

La complexité des procédés à commander ou à superviser, le nombre élevé de données et d'événements à traiter, la répartition géographique des procédés, d'une part, et l'arrivée depuis plusieurs années, sur le marché, de réseaux locaux industriels, d'autre part, sont tous des facteurs qui ont conduit à repenser les applications temps réel centralisées.

Aujourd'hui, la notion d'architecture temps réel et réparti est communément acceptée dans le milieu industriel. On peut citer par exemple, les domaines d'applications qui font couramment appel aux systèmes temps réel et répartis : le contrôle et la régulation de trafic en milieu urbain, les industries (contrôle/commande de procédés…), le domaine militaire (suivi de trajectoires de missiles…), le domaine aérospatial (suivi de satellites, pilotage automatique…), le multimédia (téléconférences, téléachat...)…

L'introduction de l'approche distribuée en ordonnancement résulte du rapprochement des domaines de l'Ordonnancement et de l'Intelligence Artificielle Distribuée (IAD). Au-delà des concepts, définitions et applications spécifiques qui les caractérisent, l'étude des problèmes auxquels ils s'attachent ainsi que les solutions qu'ils proposent, révèlent des points de convergence voire de possibles coopérations entre ordonnancement et IAD.

1. Systèmes d'ordonnancement distribués :

La distribution d'un problème revient à le décomposer (identifier dans un problème global un ensemble de sous problèmes partiels). Il y a en conséquence autant de modes de distribution qu'il y a de possibilités de décomposer un problème d'ordonnancement et d'interpréter cette décomposition. Ainsi, l'objet de la distribution peut être le problème lui-même ou le système réel qu'il modélise. Il convient ensuite de définir les unités de résolution

auxquelles vont être allouées les sous problèmes identifiés (agents artificiels et/ou humain), déterminer leur degré de liberté d'action (agent dirigé ou autonome) ainsi que leur organisation sociale (hiérarchique ou décentralisée). Enfin il reste alors à préciser leurs modes d'interaction dans la construction d'une solution d'ordonnancement finale (mode de communication, coordination ou coopération, etc.).

Face au grand nombre d'options conceptuelles, il est difficile de proposer une classification qui recouvre tous les cas possibles. On peut citer la classification de Tranvouez, proposée dans [Tra 01], cette dernière se base sur le critère de perspective selon lequel ces approches abordent le problème d'ordonnancement initial. Il a ainsi identifié quatre grandes classes de méthodes qui sont : l'aide à la décision, l'allocation de ressource, la coordination et la coopération.

La mise en place d'un système d'ordonnancement distribué, nécessite l'utilisation d'une architecture particulière, qui prend en charge les particularités d'un ordonnancement distribué (distribution des calculs d'ordonnancement sur les différentes ressources). Il s'avère que les systèmes multi agents (SMA) répondent à ce besoin.

Les principes de fonctionnement des systèmes multi agents sont radicalement différents des systèmes conventionnels. Pour bien saisir les mécanismes et leur dynamique, un certain nombre de concepts doivent être acquis.

2. Systèmes multi agents :

Issus de l'évolution des systèmes du domaine de l'IAD, les systèmes multi agents sont apparus avec la décentralisation du contrôle de l'exécution [Gas 87]. Les systèmes multi agents peuvent être observés selon deux points de vue :

- Le macro-niveau : qui correspond au point de vue d'un observateur extérieur. Il considère le système dans son ensemble et ignore tout mécanisme interne.
- Le micro-niveau : qui correspond au point de vue d'un observateur considérant le système au niveau des agents. À ce niveau, on peut observer le comportement des agents ainsi que leurs interactions.

La description d'un SMA commence par la définition des agents qui le composent et des modes d'interactions qui les relient.

A. L'agent :

L'agent est le composant principal des systèmes multi-agents. Selon la définition communément admise, il s'agit d'une entité physique ou virtuelle autonome, capable de percevoir partiellement son environnement et d'agir sur celui-ci de façon à atteindre un certain but [Fer 95].

Pour être autonome, un agent doit posséder les propriétés suivantes :

- Son existence ne dépend pas de la présence d'autres agents ;
- Il est capable de maintenir sa viabilité dans des environnements dynamiques, sans contrôle extérieur ;

- Son comportement est régi uniquement par un contrôle interne. Ce comportement est par conséquent uniquement fonction des perceptions de l'agent, de ses connaissances et de sa représentation du monde (l'agent est ainsi capable de refuser la requête d'un autre agent) ;
- Aucun contrôle extérieur ne peut accéder à ses représentations internes ;

Un agent fonctionne selon le cycle classique Perception, Décision, Action. L'agent perçoit son environnement. Il décide, par la suite, de l'action qu'il effectuera en fonction de ses perceptions, de ses connaissances et de ses buts. Enfin, il exécute l'action choisie tendant généralement à le rapprocher de son objectif s'il en possède un, voire à l'atteindre.

Deux systèmes d'agent existent, les agents réactifs et les agents cognitifs. Les systèmes d'agents réactifs s'appuient sur une conception émergente de l'intelligence, à l'instar des sociétés d'insectes, un grand nombre d'agents de faible Granularité (faible degré de détail des connaissances de l'agent) peut présenter un comportement global cohérent et efficace. Au contraire, les systèmes d'agents cognitifs sont constitués d'un petit nombre d'agents de forte granularité (chaque agent est apparenté à un système expert plus ou moins évolué). Les actions de ces agents seront ainsi *« réfléchies »* en ce sens qu'elles sont basées sur les connaissances de l'agent (sur lui-même, sur les autres et sur son environnement) et les objectifs qui le guident. La première étape pour la conception d'un SMA, est la définition des proportions des caractères cognitifs et réactifs constituant les agents de ce système.

Ce qui est central dans la définition et la modélisation d'un agent, c'est son comportement. En se basant sur le comportement, trois modèles nous semblent représentatifs, à savoir, l'éco-agent, l'agent BDI et l'agent AOP.

i. L'éco-agent :

L'éco-agent est un modèle d'agent réactif proposé par Ferber dans [Fer 95]. Il est basé sur trois modes de comportement : chercher la satisfaction, satisfaire et fuir. Un éco-agent cherche constamment à atteindre l'état de satisfaction guidé par sa fonction de satisfaction. S'il est gêné dans sa recherche par un autre agent, il l'agresse. Ce dernier est alors dans l'obligation de fuir. Un ensemble de règles simples prédéfinies permettront à un agent d'associer à un signal reçu un comportement particulier.

ii. L'agent BDI :

Les architectures BDI (Belief - Desire - Intention) expliquent le comportement d'un agent cognitif par la propension de ce dernier à satisfaire des désirs ou buts, se traduisant par des intentions (plans d'action intermédiaires) à partir de ses croyances (connaissances supposées vrai d'un agent sur lui-même et son environnement) [Had 96]. Le comportement d'un agent résulte alors de l'influence de plusieurs « attitudes » vis à vis du monde qui l'entoure.

iii. L'agent AOP :

La démarche de Programmation Orientée Agent (POA) a été proposée par Shoham dans [Sho 93]. Elle est, également, basée sur des postures intentionnelles. Un agent est perçu comme étant une entité dont l'état est considéré comme constitué de composantes mentales

telles que croyances, compétences, choix et engagements. Les compétences regroupent les différentes tâches qu'un agent peut accomplir et les choix correspondent aux capacités décisionnelles de l'agent. Les engagements sont les actions qu'un agent promet à un autre de réaliser et sont la base de toute coopération. Les croyances définissent, comme pour les architectures BDI, les connaissances d'un agent. Shoham formalise ces notions en un langage qui intègre également les notions d'actions (sur quoi portent les engagements) et de temps.

La plupart des SMA proposent des architectures d'agents particulières empruntant des concepts à l'une ou plusieurs de ces approches. Les architectures d'agents reflètent les besoins particuliers d'un domaine d'application donné, car il est très difficile, voire impossible d'élaborer une architecture générale et universelle [Huh 98].

B. L'environnement :

Le type d'environnement varie selon le point de vue que l'on adopte (SMA, agent, concepteur).

i. Point de vue du système multi-agent :

Du point de vue du système multi-agent, l'environnement correspond à l'ensemble des entités extérieures au système. Par exemple, si l'on considère une fourmilière, l'environnement est tout ce qui est présent dans la nature à l'exception des fourmis appartenant à la fourmilière : cailloux, végétaux, nourriture, lois physiques, etc.

ii. Point de vue de l'agent :

L'environnement est tout ce qui est extérieur à lui-même. Par exemple, l'environnement d'une fourmi est composé de l'environnement de la fourmilière et de toutes les autres fourmis.

iii. Point de vue du concepteur du système :

De ce point de vu, on peut considérer plusieurs sortes d'environnements :

- L'environnement d'exécution du programme : qui correspond à l'état du système informatique sur lequel s'exécute le SMA.
- L'environnement de simulation : qui représente l'ensemble des outils logiciels permettant de simuler, de visualiser et d'évaluer l'exécution du SMA.
- L'environnement de développement : qui est l'ensemble des outils logiciels facilitant le développement du système.

C. L'interaction :

Selon la définition communément admise, l'interaction est la mise en relation dynamique de deux ou plusieurs agents par le biais d'un ensemble d'actions réciproques. L'interaction peut, ainsi, être vue comme une sorte de communication, de moyen pour mettre en œuvre une activité commune entre plusieurs agents.

L'interaction peut être directe, via l'envoi d'un message à un ou plusieurs destinataires, ou être indirecte via un dépôt d'informations dans l'environnement. Par exemple, les fourmis

communiquent selon ces deux modes, de façon directe lorsqu'elles interagissent par contact antennaire (échange de messages) et de façon indirecte en déposant des phéromones dans l'environnent de façon à créer des chemins balisés vers la nourriture.

La représentation des interactions entre agents dans un SMA conduit à définir des modes d'interaction standards ou du moins génériques. Ainsi, la FIPA avec ACL (Agent Communication Language) préconise non seulement un langage de communication mais également des modèles d'échanges de messages formant des protocoles d'interaction. Ces modèles se limitent à des cas généraux (requête, appel d'offre, …) et surtout s'appuient sur une schématisation restrictive.

Dès lors, la spécification des modes d'interaction entre agents, qu'on appelle protocoles d'interaction ou protocoles de conversation, doit dépasser le stade normatif pour devenir un outil de conception SMA à part entière. Pour cela, un outil d'aide à la spécification de tels protocoles doit vérifier les propriétés suivantes [Cos 99], [Maz 99]:

- Spécification : avant de commencer l'implémentation d'une plate-forme multi agents, cet outil doit permettre de spécifier ce que devront faire les agents et dans quel ordre ;
- Communication : une spécification graphique ou semi-formelle, facilite la communication entre concepteurs dès la phase de spécification des comportements d'agents ;
- Vérification : il doit être possible de détecter les incohérences existant dans les spécifications des activités des agents, avant même la phase d'implémentation, ceci soit par une analyse syntaxique, soit par simulation ;
- Implémentation : de manière connexe, l'outil doit pouvoir guider l'implémentation de ces comportements. Notons qu'une approche BDI s'adapte facilement à une telle approche du fait que la notion de plan recouvre celle de description de comportement.

D. La communication :

La communication est la forme d'interaction consciente (interaction contrôlée, désirée et connue explicitement de(s) l'agent(s) initiateur(s) de l'interaction) la plus élémentaire dans un SMA et constitue l'ossature sur laquelle des modes d'interaction plus évolués s'appuient. La communication dans un SMA suppose l'existence d'un médium physique (un réseau informatique par exemple), d'un médium linguistique (langage d'expression) et un mode de diffusion. Le **médium physique** dépendant davantage de choix techniques d'implémentation (architecture du réseau, types de connections, protocoles de communication).

Le **médium linguistique** nécessite la spécification d'un langage commun à un ensemble d'agents leur permettant d'interpréter un message. Les recherches en communication inter-agents ont mis en valeur l'apport de la théorie des actes de langages (speech act theory [Sea 72]). Le langage KQML (Knowledge Query Manipulation Language), développé dans le cadre du projet DARPA [Fin 94], largement utilisé dans les SMA ainsi que le langage ACL (Agent Communication Language) proposé par la FIPA (Foundation for Intelligent Physical Agents [Fip 97]) (sachant qu'on va utiliser les primitives du langage ACL, elles sont données en annexe II), reposent sur cette théorie. Brièvement, chaque acte de discours (un envoi de message dans un SMA) est considéré comme une action convoyant des informations sur l'intention de son émetteur et le résultat attendu.

Les **modes de diffusion** se distinguent en deux grands types : les modes anonymes et les modes nominatifs. Le mode de diffusion anonyme peut s'effectuer au travers d'un tableau noir (ou Blackboard System) ou par des envois multiples (broadcasting). Dans le premier cas, les agents envoient leurs informations ou requêtes à une zone de connaissance (ou mémoire) partagée que chaque agent consulte [Dur 95]. Les agents lisent ainsi tous les messages inscrits dans le tableau noir mais ne réagissent qu'à ceux qui les concernent (spécifications de compétences ou d'informations qu'ils possèdent).

Le mode de diffusion anonyme par envoi multiple est assez similaire quant aux réactions des agents. Il s'appuie sur l'envoi d'un message à tous les autres agents sans destinataire spécifique (broadcasting).

A l'opposé, dans le mode de diffusion nominatif ou par message, un agent envoie "personnellement" un message à chaque agent identifié dans sa base d'accointances (agents connus par celui-ci).

Aussi est-il nécessaire de prévoir un mécanisme permettant à chaque agent de savoir comment contacter un agent spécifique (de connaître son "adresse"). Précisons que loin de s'opposer, les modes de communication par envoi multiples de message et par message sont souvent utilisés de paire : le premier permettant d'établir une liste de correspondants privilégiés, utilisée ensuite par le second.

E. L'organisation et les rôles :

L'organisation est la description, au macro-niveau, des interactions et des relations du micro-niveau, mises en place entre les agents d'un système. Grâce à l'observation de ces relations, il est possible d'identifier des rôles. Ainsi, par exemple, dans une fourmilière certains agents s'occupent de recruter d'autres fourmis (rôle de recruteurs), alors que d'autres s'adonnent à la recherche de nouvelles sources de nourriture (rôle de fourrageur).

Ainsi, l'activité des agents au sein du système fait apparaître une certaine organisation et, par conséquent, un certain nombre de rôles.

3. Résolution par approche coopérative :

Ce type de résolution met en jeu un agent perturbé (à l'origine de la coopération et responsable du bon déroulement de la coopération), et p agent(s) coopérant(s) ($p \geq 1$). L'étude des comportements des agents en coopération a révélé une grande similitude dans leur gestion du processus d'obtention d'une solution coopérative (recueil des propositions, sélection, etc.) [Tra 01].

Dans cette section, nous illustrons comment se déroule une résolution par approche coopérative, à travers la description des différentes étapes la constituant.

A. Point de vue général de la résolution coopérative :

Les grandes étapes par lesquelles passent les deux types d'agents, perturbé et coopérant sont décrites dans la figure II.1. On peut identifier à travers elle, les points d'entrée et de sortie des flux de communication entre les agents ainsi que les actions effectuées par ces mêmes agents pour traiter une perturbation (on appelle, dans notre cas, perturbation, la

libération, d'au moins une machine, ou, l'arrivée d'une tâche aléatoire et une machine au moins est libre).

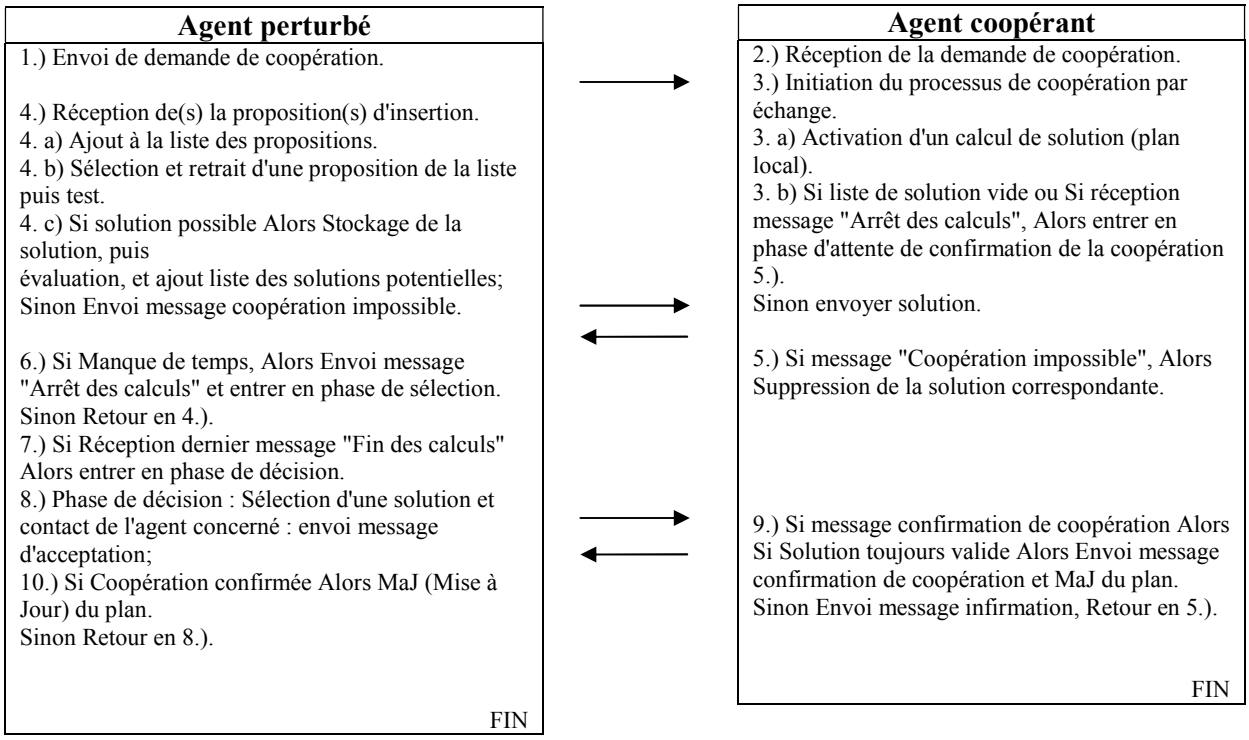


Figure II.1 : Point de vue général d'une résolution coopérative [Tra 01]

A partir de là, deux protocoles de rôles émergent : selon le point de vue de l'agent perturbé et selon le point de vue de l'agent coopérant.

B. Protocoles de rôles : point de vue de l'agent perturbé :

La gestion de la coopération est assurée par l'agent perturbé (initiateur de la résolution coopérative), ainsi, il envoie un appel d'offre aux agents adéquats et sélectionne la solution la plus efficace ou du moins la moins perturbatrice. C'est un rôle de coordinateur qu'on peut identifier clairement via la figure II.2.

Il s'agit pour cet agent, de sélectionner les destinataires des appels d'offre (sélection des contractants potentiels) en fonction de ses croyances sur les compétences des agents du système. Ainsi, Il ne s'adresse qu'aux agents possédant la fonctionnalité nécessaire à l'accomplissement de la tâche à effectuer, les autres ne pouvant pas physiquement le faire.
Pour accomplir cette tâche, l'agent perturbé fait appel à l'aide d'un R-agent (Le R-agent remplit le rôle d'un agent Pages Jaunes maintenant une liste d'agents et de leurs compétences, il connaît l'ensemble des agents ainsi que leurs compétences. Il a deux fonctions : la gestion de ces connaissances et l'envoi multiple de message).

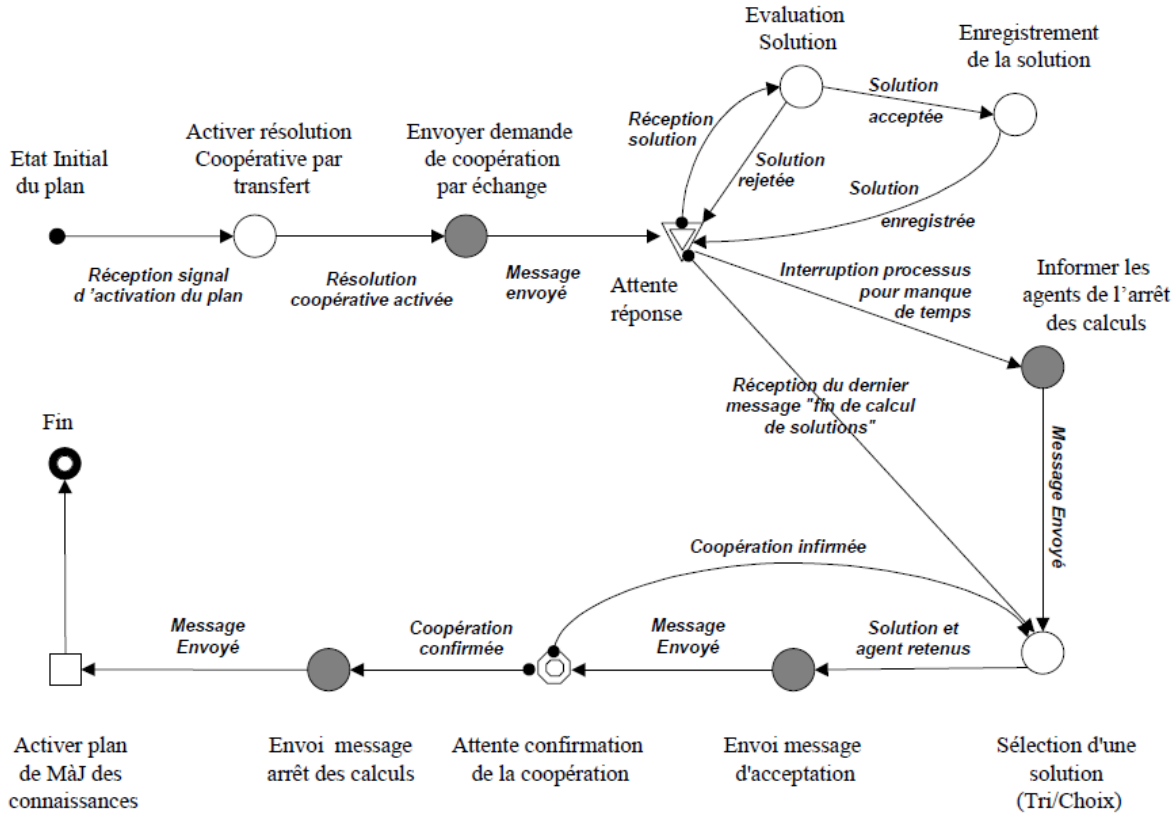


Figure II.2 : Protocole de rôle de résolution coopérative : point de vue de l'agent perturbé [Tra 01]

C. Protocoles de rôles : point de vue de l'agent coopérant :

On peut considérer l'agent coopérant comme étant le véritable moteur de la coopération, de par sa contribution à l'alimentation du processus de résolution coopérative en solutions potentielles permettant d'absorber la perturbation. La figure II.3, décrit son comportement.

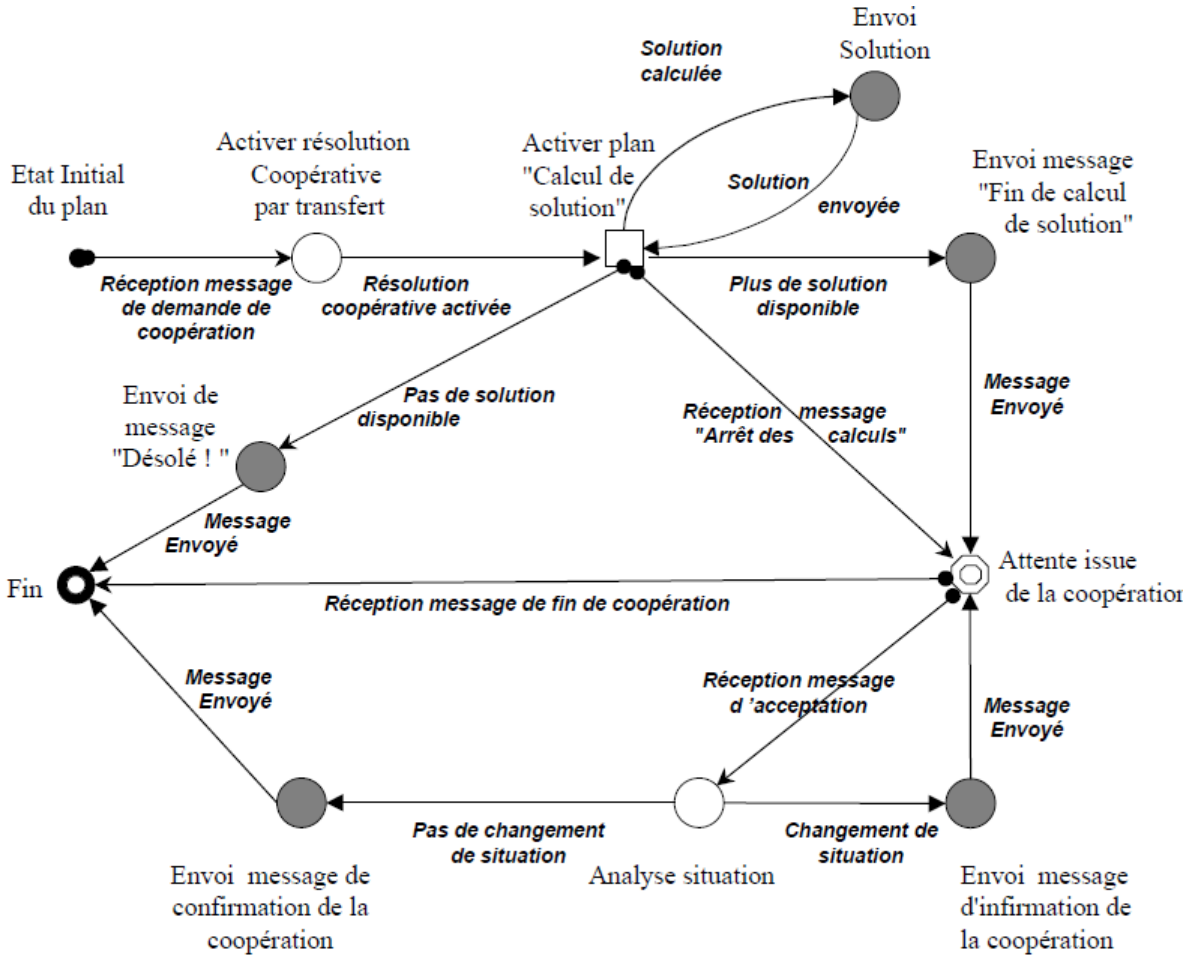


Figure II.3 : Protocole de rôle de résolution coopérative : point de vue de l'agent coopérant [Tra 01]

D. Mise à jour des connaissances :

Aussitôt une stratégie de résolution réussit, une coopération informationnelle est engagée pour évaluer si les modifications de connaissances chez les agents impliqués peuvent en concerner d'autres. Les figures II.4 et II.5, décrivent la mise en application de la mise à jour des connaissances du point de vu émetteur et destinataire respectivement.

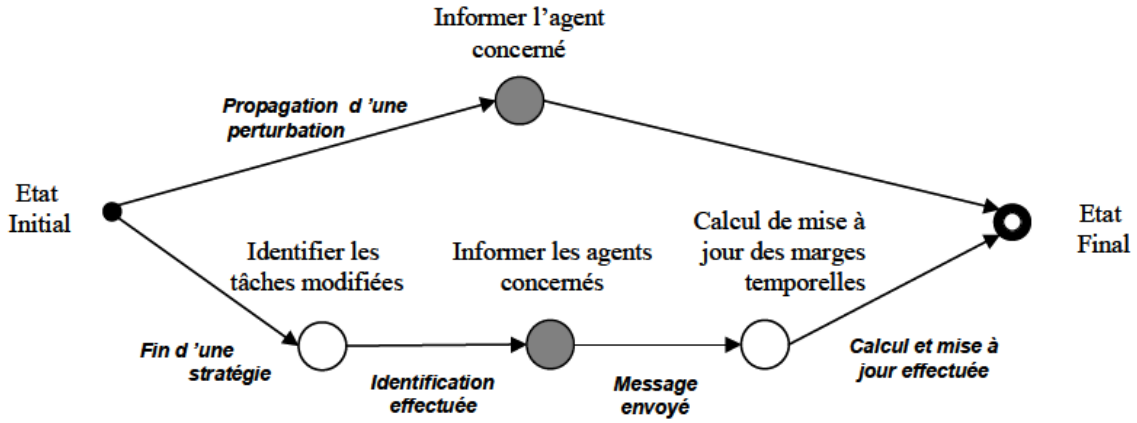


Figure II.4 : Protocole de rôle de mise à jour des connaissances : rôle émetteur [Tra 01]

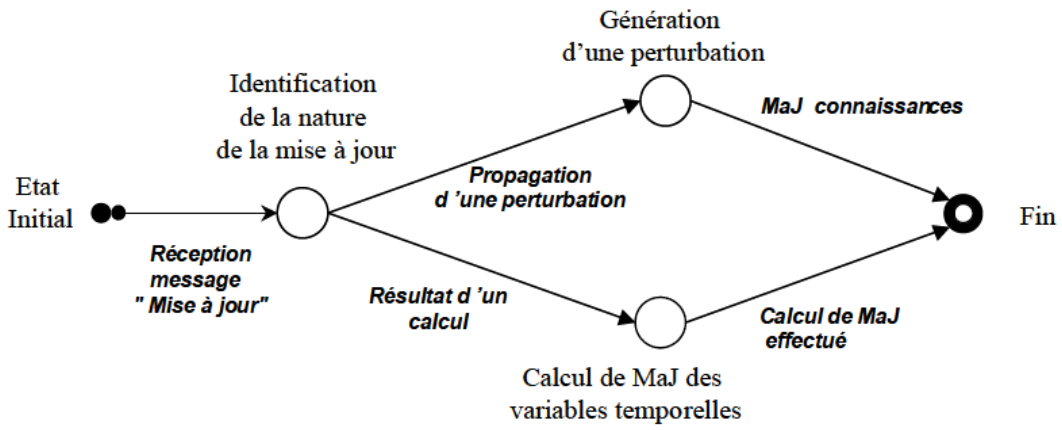


Figure II.5 : Protocole de rôle de mise à jour des connaissances : rôle destinataire [Tra 01]

E. Les formes de coopération :

Trois formes de coopération ont été recensées [Cam 00] : La coordination, la collaboration et la codécision.

- **La coordination :** vise à synchroniser les actions dans le temps en exploitant un référentiel temporel commun, et à gérer la cohérence des actions individuelles par rapport à l'ensemble des activités. La coopération a pour objet de faciliter la coordination d'activités étroitement complémentaires pour la réalisation des processus.
- **La collaboration :** signifie travailler ensemble à l'exécution d'une certaine action pour produire un résultat final. Selon [Bou 02], la collaboration implique le partage d'informations à l'intérieur d'un groupe donné, sans prise de décision collective. Le terme collaboration s'utilise à la place de coopération lorsque les actions individuelles ne sont pas différentiables.
- **La codécision** : signifie la collaboration de plusieurs acteurs en vu de prendre des décisions. Cette codécision peut être le résultat de mécanismes de négociation ou de renégociation. Ces mécanismes visent à trouver un compromis acceptable entre les objectifs locaux de chaque entité qui peuvent être contradictoires. Si aucune codécision n'a été prise au préalable, on parlera de négociation. Au contraire, on parlera de renégociation si l'objet de la collaboration entre partenaires est la remise en cause d'une décision passée.

III. La simulation :

Comme outil d'aide à la décision, la simulation joue un rôle très important pour l'industriel. Les principales raisons de son utilisation sont :

- Le manque de nécessité pour l'utilisation de mathématiques poussées ;
- La possibilité de réalisation de modèles réalistes ;
- Les résultats visuels sont un support pour l'aide au développement du modèle et à sa validation ;
- La simulation est aussi un moyen de compréhension et de confiance à disposition du client ;

- Différents scénarii d'exploitation peuvent être considérés sans expérimentation directe sur le système,
- Des systèmes non existants peuvent être modélisés,
- Les méthodes analytiques sont perçues comme des techniques peu utiles par le management de l'entreprise ou peuvent nécessiter beaucoup de simplifications,...

Cette technique peut aider à mieux cerner les conséquences de choix potentiels d'actions, afin de mieux les maîtriser et d'améliorer ainsi le pilotage de l'atelier, sachant qu'elle est appropriée à l'étude des systèmes complexes et de grande taille, composés de plusieurs éléments en interaction. Elle permet de répondre à certains problèmes à chaque fois qu'un modèle mathématique ne peut être trouvé ou que l'expérimentation en grandeur nature se révèle impossible et/ou trop coûteuse.

Si la simulation est une technique de plus en plus utilisée dans le milieu industriel, en devenant l'outil de base des ingénieurs, il n'en demeure pas moins qu'elle reste encore mal maîtrisée dans certaines entreprises. Pourtant, nombreuses sont celles qui sont amenées à concevoir ou à faire évoluer des systèmes de production et qui doivent se poser des questions décisives du type :

- ✓ la gestion appliquée sur cette ligne de production est-elle bien adaptée ?
- ✓ les paramètres de gestion choisis sont-ils les meilleurs ?
- ✓ le pilotage par l'aval, de type Kanban par exemple, est-il bien adapté ?
- ✓ comment convaincre le responsable du budget qu'il devient indispensable d'investir dans telle ou telle machine ou d'embaucher du personnel ?
- ✓ comment déterminer le rendement opérationnel ou le flux sortant d'une ligne comportant de nombreux équipements soumis à des pannes, des changements de série ou d'outil, des modifications d'allure des opérateurs ?

1. Généralités sur la simulation :

La simulation est l'expérimentation sur un modèle. C'est une procédure de recherche scientifique qui consiste à réaliser une reproduction artificielle (modèle) du phénomène que l'on désire étudier, à observer le comportement de cette reproduction lorsque l'on fait varier expérimentalement les actions que l'on peut exercer sur celle-ci, et à en induire ce qui se passerait dans la réalité sous l'influence d'actions analogues. On a recours aux techniques de simulation essentiellement dans deux types de situations. Le premier type se définit par l'impossibilité de recourir à l'expérimentation directe, en raison de considérations morales, d'impératifs temporels, de contraintes budgétaires, ou d'obstacles naturels. La simulation permet par exemple de déterminer empiriquement la configuration optimale d'un réseau routier, de prévoir les répercussions de diverses mesures économiques sur la consommation ou l'épargne, ou d'analyser les phases successives d'une panique provoquée par un incendie dans une salle close. Pour que l'expérimentation sur le modèle ait une valeur scientifique, il faut évidemment que le modèle constitue une reproduction satisfaisante de la réalité, c'est-à-dire qu'il repose sur des bases théoriques assurées. Le second type de situation où la simulation s'avère efficace est celui où l'on ne dispose pas de bases théoriques solides, et où l'on cherche précisément à élaborer une théorie qui permette de rendre compte des données d'observation grâce aux techniques de simulation ; on peut alors définir avec précision les conséquences concrètes des différents modèles théoriques possibles et déterminer lequel fournit l'approximation la plus correcte de la réalité.

A. Définition de la simulation :

On trouve dans la littérature plusieurs définitions de la simulation. Ces définitions sont parfois proches, équivalentes ou complémentaires. Une synthèse de différentes définitions de la simulation est proposée dans [Bak 96] : *« la simulation est une méthode de mesure et d'étude consistant à remplacer un phénomène ou un système à étudier par un modèle informatique plus simple mais ayant un comportement analogue »*. Ce modèle permet de réaliser des tests afin de cerner le comportement dynamique du système étudié. Il est implémenté sur ordinateur, et évolue d'un état vers un autre en fonction de certaines règles de changement d'état bien définies. La simulation est une technique expérimentale appropriée à l'étude des systèmes de grande taille composés de plusieurs éléments en interaction. Elle permet ainsi de répondre à certains problèmes à chaque fois que l'expérimentation en grandeur nature se révèle impossible et/ou trop coûteuse [Car 88], [Cer 88].

Simuler revient à créer un modèle informatique fidèle au système étudié de par son comportement. A l'aide du modèle défini, on peut mettre au point de véritables plans d'expérience pour faire des tests sur certaines décisions à prendre, certains paramétrages modifiés [Zei 76]. « *L'objectif de la simulation sur ordinateur est donc de reproduire et anticiper, prédire le comportement dynamique d'un atelier existant (ou futur), modélisé de façon virtuelle, donc sans aucun risque »* [Pri 89], [Hol 90].

La simulation est principalement utilisée pour étudier les flux dans l'atelier (pièces, matières, outils, informations, ...) et les disponibilités des ressources (opérateurs, machines, convoyeurs,...) [Pro 93]. Elle est utilisée en général pour évaluer et comparer des scénarii possibles. Ses capacités d'imitation et de prédiction permettent d'obtenir des renseignements sur les conséquences de changements ou de modifications dans l'atelier (niveau physique ou décisionnel), avant que ceux-ci ne soient effectués [Peg 95].

Ainsi, différents scénarii représentant plusieurs jours, mois ou années de fonctionnement d'un système réel, peuvent être testés rapidement, en quelques minutes ou quelques heures grâce à la simulation : c'est ce que Zeigler [Zei 89] appelle *« l'effet de compression »* dans la simulation. Et inversement, d'autres scénarii représentant une minute, une heure,…, d'un système réel, peuvent être détaillés très précisément : c'est *« l'effet de dilatation »*.

De manière relativement simplifiée, le processus de simulation peut être présenté selon la figure II.6. La modélisation fournit un modèle conceptuel (modèle de connaissance), la programmation fournit un modèle exécutable (modèle d'action), la simulation ou l'expérimentation fournit les résultats obtenus du modèle, l'analyse des résultats permet d'évaluer le processus à modéliser.

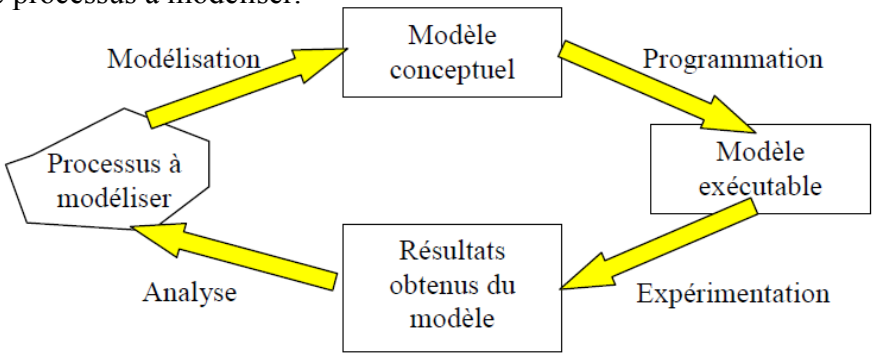


Figure II.6 : Processus simplifié de simulation [Hab 01]

De manière plus détaillée, ce processus peut être éclaté en dix étapes [Pri 86], figure II.7 : analyse et formulation du problème, identification et collecte des données, construction du modèle, transcription informatique du modèle, vérification du modèle, validation du modèle, planification stratégique et tactique de la simulation, exécution de la simulation, analyse et interprétation des résultats, recommandations et mise en place.

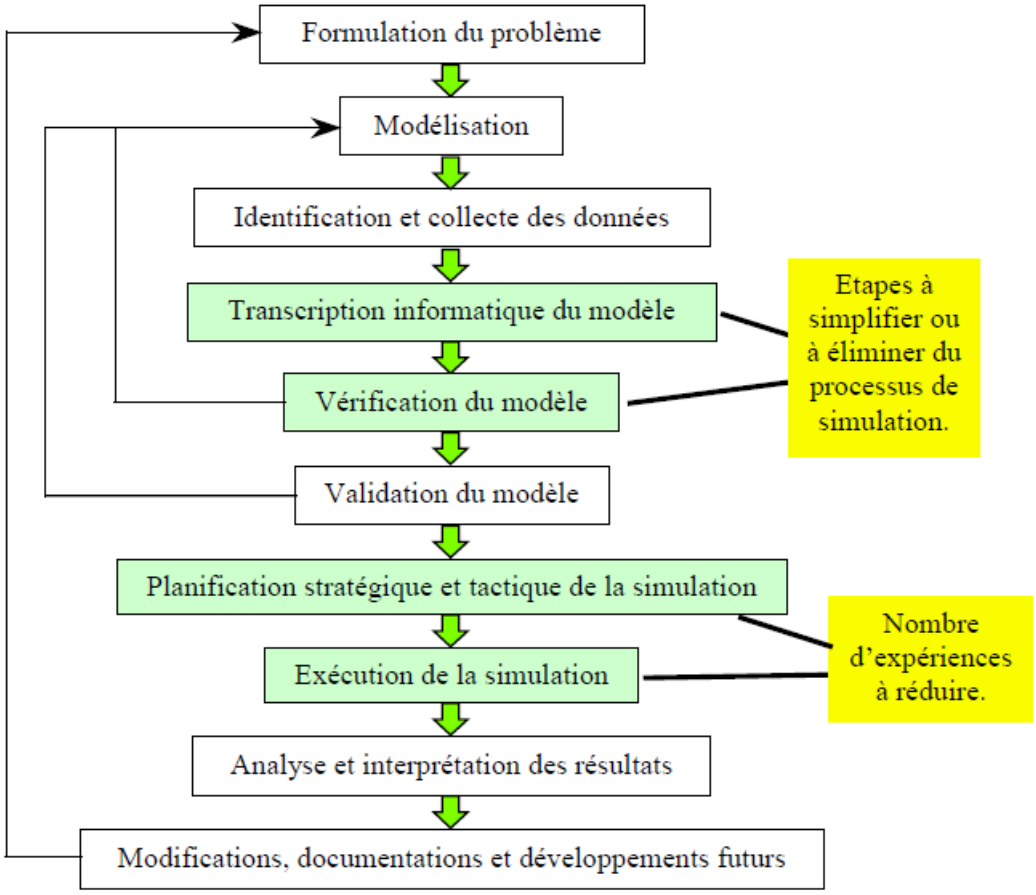


Figure II.7 : processus de simulation selon Pritsker [Pri 86]

La simulation informatique est un outil puissant d'imitation des systèmes de production [Fis 97]… encore faut-il que cette imitation soit fiable, et donc qu'elle ait des « bases saines ». La fiabilité d'une simulation dépend directement de la pertinence et de la validité de son modèle de base utilisé. La démarche de faire un modèle de simulation demande rigueur dans l'analyse de l'existant, compétence pour la création des modèles et savoir-faire pour l'analyse des résultats**.** La modélisation constitue ainsi, le **fondement** même de la simulation. La réciproque n'est d'ailleurs pas forcément vraie, car, la finalité de la modélisation n'est pas forcément la simulation. Il existe des techniques de modélisation qui ont pour objectif l'amélioration de la compréhension du système étudié, l'analyse de son organisation, sans pour autant effectuer de simulation ensuite. Mais la modélisation est la première phase essentielle dans la simulation.

Les potentiels de la simulation sont vastes. La figure II.8 résume les différents champs de représentation couverts par la simulation.

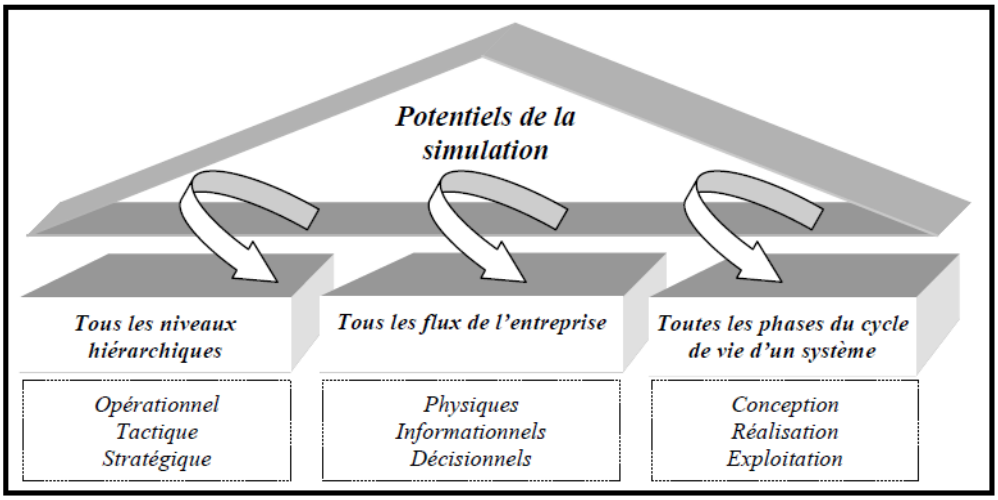


Figure II.8 : les potentiels de la simulation [Ber 00]

Les apports de la simulation dans le domaine des systèmes de production sont aussi vastes, comme le montre la figure II.9.

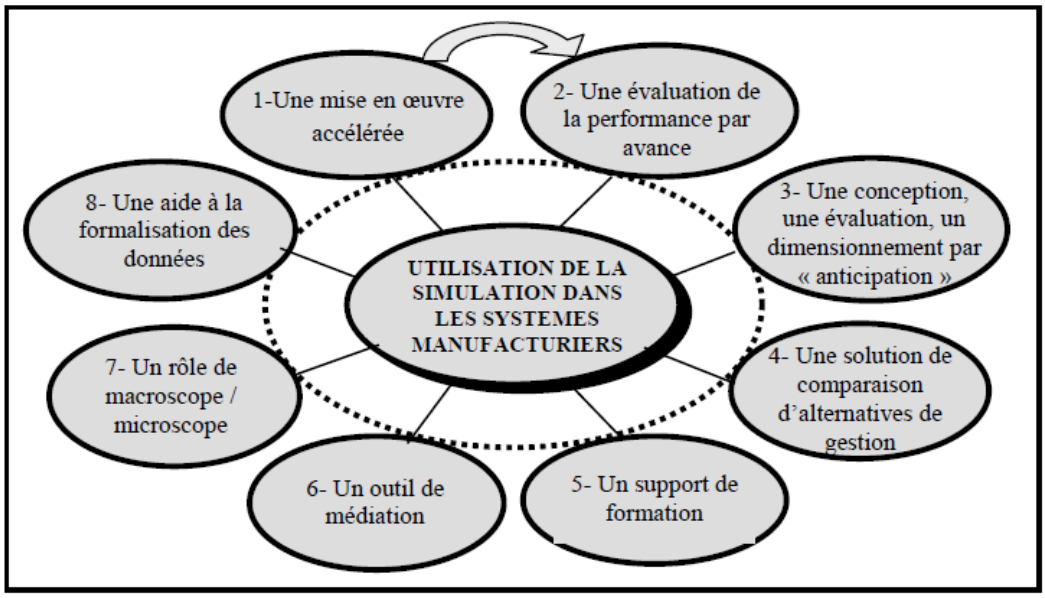


Figure II.9 : utilisation de la simulation dans les systèmes manufacturiers [Ber 00]

B. Généralités sur la modélisation :

Qui dit simulation, dit modèle. La modélisation en entreprise a pour objet la construction de modèles d'une partie déterminée d'une entreprise pour en expliquer la structure et le fonctionnement ou pour en analyser le comportement [Ver 99]. Ou encore, les activités de modélisation consistent à construire un modèle à partir d'un système réel et à définir la validité de ce modèle par rapport à ce système.

i. Définition de la modélisation :

Le modèle est par définition une représentation d'une abstraction d'une partie du monde réel, exprimée dans un langage de représentation. Ce langage peut être formel (i.e. : ayant une syntaxe et une sémantique bien définies comme un langage informatique), semi-formel (i.e. : notation graphique normalisée) ou informel (i.e. : description en langage naturel) [Ver 97] [Ver 96]. On dit que b est un modèle de B pour un observateur O, si b peut renseigner O sur le fonctionnement de B. Le modèle permet une description structurelle du système réel, d'une part, et représente ainsi un substitut du système réel afin d'en analyser le comportement dans des conditions variées, d'autre part. Le modèle n'est probablement pas la solution d'un problème, mais il donne quelques idées pour l'aborder plus intelligemment. Il peut concerner un système existant ou à concevoir : on parlera alors de modélisation *« a posteriori »* dans le premier cas (tel que : nouvelle reconfiguration, test d'ordonnancement, ...), et de modélisation *« a priori »* dans le second (tel que : dimensionnement, agencement, évaluation, ...).

Ce processus de modélisation est défini par J-L Le Moigne comme *« une élaboration et construction intentionnelle, par composition de symboles, de modèles susceptibles de rendre intelligible un phénomène perçu complexe et d'amplifier le raisonnement de l'acteur projetant une intention délibérée au sein de phénomène ; raisonnement visant notamment à anticiper les conséquences de ces projets d'actions possibles »* [Moi 90].

Les moyens de production devenant de plus en plus complexes, la réalisation des modèles le devient encore davantage. De fait, la modélisation des systèmes de production suscite depuis plusieurs années, un intérêt accru de la part de chercheurs qui y trouvent un terrain fertile pour concevoir des méthodes et des outils pratiques et efficaces, facilement utilisables au niveau industriel. On peut trouver dans [Ver 99], un aperçu des principales techniques essentielles de modélisation existantes.

ii. Les différents types de modèles pour la simulation :

On recense plusieurs types de modèles de simulation en ce qui concerne les systèmes de production, à savoir :

- Les modèles discrets (ou discontinus) : *« déroulent »* l'historique des événements qui surviennent lors d'une fabrication. Les changements d'état ne surviennent que lors d'événements tels le début ou la fin d'une opération, la mise en attente d'une pièce dans une file, la libération d'une ressource, l'occurrence d'une panne, ...
- Les modèles continus : plus adaptés aux industries dites de « process », utilisent des équations mathématiques pour prendre en compte les changements d'état qui s'effectuent de façon continue au cours du temps. Les valeurs des variables d'état sont recalculées régulièrement selon un pas d'horloge d'après ces équations.
- Les modèles combinés : sont capables d'intégrer les deux aspects. Ils sont utilisés, par exemple, pour les industries métallurgiques et les industries agro-alimentaires [Pie 90].

Ceci étant, les modèles discrets sont les plus courants. Les résultats qu'ils fournissent sont des statistiques portant sur des variables qui caractérisent les flux et les ressources, tel que:

- Le taux d'occupation d'une machine ;
- Le nombre moyen de pièces en attente d'un convoyeur ;
- La dispersion du retard de livraison d'un ensemble de pièces ;
- Le nombre de tâches traitées ;
- Une estimation de la durée nécessaire pour réaliser un plan donné…

iii. La démarche de réalisation d'un modèle de simulation :

La réalisation d'un modèle de simulation requière au préalable, la définition des objectifs de l'étude et des simplifications de la réalité qu'il est raisonnable de faire.

Réalisé en fonction des objectifs de l'étude, le modèle de simulation permet d'obtenir des résultats numériques (éventuellement statistiques) qui sont ensuite analysés pour permettre la prise de décisions concernant le système réel comme le montre la figure II.10.

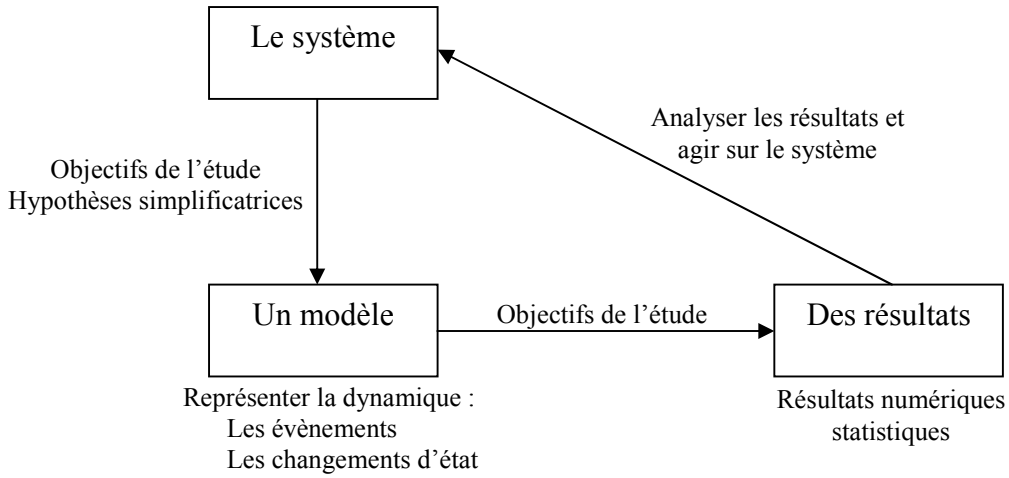


Figure II.10 : réalisation d'un modèle de simulation

La première étape de la démarche de réalisation d'un modèle de simulation consiste à décrire précisément les entités du modèle et ses règles de gestion. On décrit ensuite la dynamique du système. Dans le cas d'un système simple avec un faible nombre d'entités, une représentation graphique peut s'avérer suffisante. Dans le cas de systèmes plus complexes, on peut être amené à utiliser des outils de formalisation comme les réseaux de Petri. Cette étape de formalisation est très importante car elle évite de remettre en cause le modèle de simulation ultérieurement.

La deuxième étape concerne le codage ; elle s'appuie sur la description du système réalisée précédemment.

La troisième étape concerne la vérification du programme afin de s'assurer qu'il fait ce que l'on veut qu'il fasse. Il s'agit de vérifier le code et sa logique. La validation du modèle est une étape très importante et délicate car, dans le cas d'un système complexe, il n'existe pas de méthode permettant de s'assurer que le modèle représente correctement le système de départ. On s'appuie alors sur :

- ✓ Des techniques d'animation du modèle montrant les entités en déplacement conformément aux règles de fonctionnement ;
- ✓ Des comparaisons avec des résultats mathématiques théoriques lorsque ces derniers sont disponibles ;
- ✓ Des comparaisons avec les résultats fournis par des modèles conçus indépendamment ;
- ✓ Enfin, dans le cas d'un modèle de système existant, il faut souvent recourir à des spécialistes du domaine pour interpréter les résultats obtenus par simulation, et vérifier qu'ils sont en conformité avec le comportement du système réel.

Les étapes de cette démarche sont reprises dans la figure II.11.

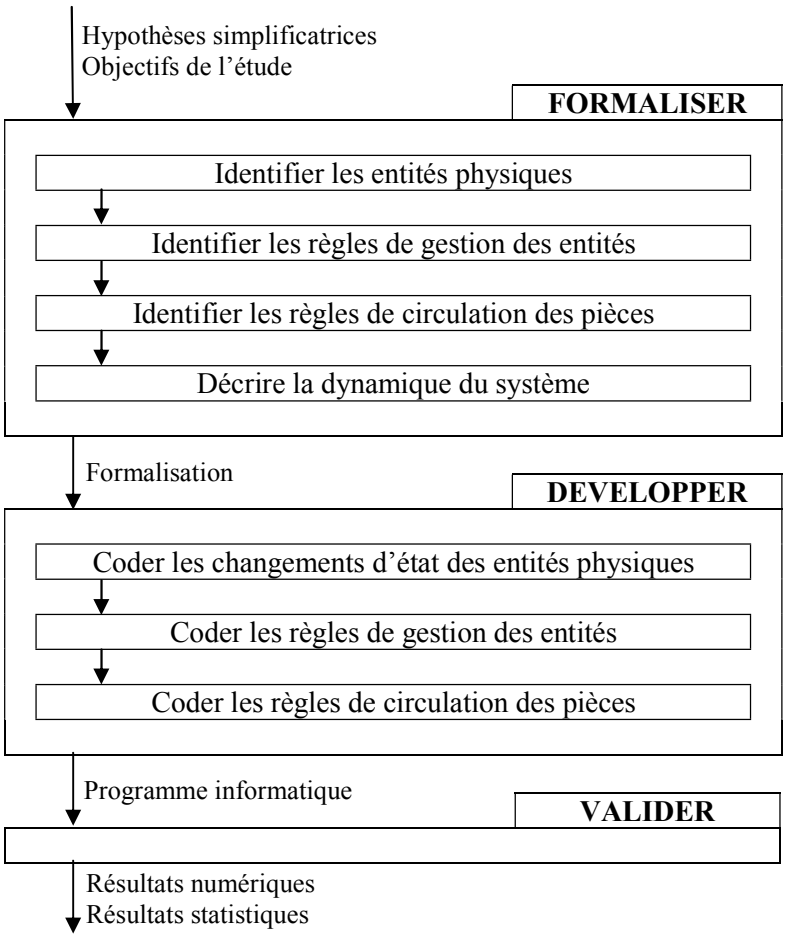


Figure II.11 : le détail de la démarche pour la réalisation d'un modèle de simulation

Une présentation des langages de simulation est donnée en annexe III. Dans le cadre de ce travail, nous avons utilisé la version étudiant numéro 10 de Rockwell ARENA (SIMAN).

2. Présentation du logiciel ARENA 10 :

ARENA est un logiciel dédié de modélisation et de simulation des systèmes de production, des procédés, des chaines logistiques, de la distribution, des stocks, etc. Il présente une facilité d'utilisation en étant un simulateur de haut niveau, et une flexibilité de la modélisation en utilisant le langage de simulation SIMAN et le script Microsoft Visual Basic. Les composants d'ARENA sont programmés en SIMAN, ce qui permet d'en créer ses propres blocs, et ce qui en rajoute à la flexibilité du simulateur. Ajoutez à cela, le simulateur comprend des animations graphiques, des courbes et de différents composants d'analyse des résultats.

A. Structure graphique :

La fenêtre d'ARENA se présente comme suit :

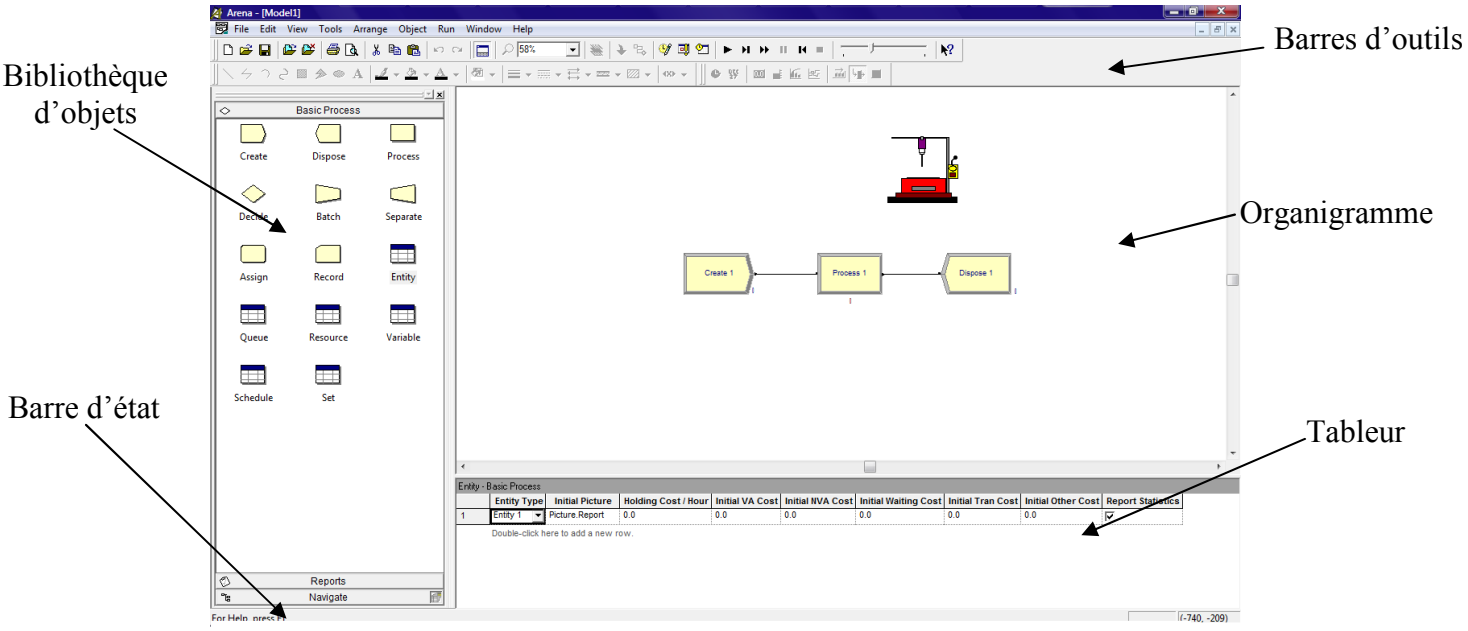


Figure II.12 : interface graphique d'ARENA

Comme le montre la figure II.12, l'interface graphique se compose de :

- **Une barre d'outils** comportant les fonctionnalités :
  - Standards : nouveau, ouvrir, enregistrer, copier, coller, revenir, répéter, imprimer, zoom, attacher de nouvelles *templates*[1], détacher des *templates*[1], les connections entre blocs ainsi que les boutons de commande de la simulation : début, pause, arrêt, avance rapide, retour, etc.
  - De dessin
  - D'animation
  - …
- **Une barre de projet** comportant la bibliothèque des objets préprogrammés ou « *templates* » parmi elles nous citons principalement :
  - Les processus de base ou « basic process » préprogrammés et prêts à l'emploi parmi lesquels nous distinguons :
    - Les modules organigrammes :
      - CREATE
      - PROCESS
      - ASSIGN
      - DECIDE
      - DISPOSE
    - Les modules de données :
      - Déclaration et caractérisation de variables
      - Déclaration et caractérisation de files d'attente
      - Déclaration et caractérisation de ressources
      - …

Les deux types de modules communiquent par les noms des objets. Chaque file d'attente, ressource ou variable a son propre nom.

[1] Un groupe d'objet appartenant à la même classe. Ex : les blocs « blocks », les éléments « elements », les processus de base « basic process »…

- Les blocs ou « blocks » plus flexibles que les « basic process » mais plus rudimentaires. Ils comprennent les mêmes fonctionnalités des blocs précédents mais en décomposant chaque bloc en plusieurs blocs plus élémentaires. Ex : le bloc PROCESS du *template* « Basic process » se décompose en plusieurs blocs du *template* « Blocks » et qui sont : Queue, Seize, Delay, Release. En procédant ainsi, nous devons déclarer les blocs Queue et Seize dans le *template* « elements ». Et pour chaque classe, les blocs doivent être identifiés manuellement.
- Les éléments ou « elements » Conservent les caractéristiques statiques du modèle. Ex : un bloc Queue est, statiquement, caractérisé par une étiquette, un nom et une règle de priorité.
- Les rapports de simulation : « reports » Ils donnent les informations recueillies après simulation auprès des ressources, des files d'attentes ainsi que les différentes statistiques.

- **Une barre d'état :** elle donne le temps écoulé de la simulation, la date de la simulation, l'état de la simulation (initialisation de la simulation, en cours de simulation, fin de la simulation, simulation interrompue), le numéro de la réplication et les coordonnées du pointeur de la souris sur l'espace de dessin.

- **Le tableur du modèle ou « model window spreadsheet view » :** Il contient les informations concernant les différents attributs et variables, des files d'attente et des ressources, des plannings et des ensembles de blocs.

- **La fenêtre de l'organigramme du modèle ou « model window flowchart view » :** C'est là que le modèle est « dessiné ». Avec les différents blocs, les connections, les animations et les graphiques.

B. Eléments d'un modèle de simulation :

Dans un modèle de simulation, nous trouvons :

i. Les entités :

Ce sont les premières choses à identifier dans un système pour pouvoir le modéliser. Les entités sont les objets dynamiques de la simulation. Elles sont créées et peuvent être dupliquées, regroupées ou disposées du système. Selon le système à modéliser, les entités représentent des choses réelles ou fictives. Elles peuvent modéliser les pannes d'une machine donnée. Comme elles peuvent être assimilées à des pièces mécaniques, des personnes, des encours de production, etc.

ii. Les attributs :

Ils servent à personnaliser les entités. Un attribut est généralement commun entre toutes les entités, mais sa valeur change d'une entité à une autre. De ce fait, les attributs sont considérés comme des variables locales pour chaque entité.

iii. Les variables (globales) :

Les variables sont des pièces contenant des informations sur le système. Nous distinguons deux types de variables globales :

- Les variables prédéfinies (états des files d'attente, états des machines, etc.) ;
- Les variables définies par l'utilisateur (variables spécifiques au modèle).

Contrairement aux attributs, les variables n'appartiennent pas à une entité spécifique, mais à tout le système.

iv. Les ressources :

Une ressource est affectée à une entité, afin de la transformer, de lui rajouter de la valeur ou juste pour la faire attendre. Une ressource peut être une personne, une machine, etc. Elle peut contenir un ensemble de serveurs identiques (*units*) en parallèle.

v. Les files d'attente :

Lorsqu'une entité doit attendre la libération d'une unité appartenant à une ressource, elle est placée dans une file d'attente. Cette dernière se caractérise par une capacité et une règle de priorité.

vi. Les accumulateurs des statistiques :

Ils servent à mesurer les différentes grandeurs et indices de performance. Parmi eux, nous distinguons :

- Le temps moyen de séjour des pièces dans le système ;
- Le temps de séjour des pièces dans une file d'attente ;
- Le nombre moyen de pièces dans une file d'attente ;
- Le taux d'occupation d'une ressource ;
- …

Les accumulateurs calculent par rapport au temps actualisé (TNOW).

vii. Les évènements :

Un évènement est une action de durée nulle, suite à laquelle, les valeurs des attributs, des variables ou des accumulateurs sont changées. Nous citons comme exemples d'évènements :

- L'entrée de nouvelles entités dans le système ;
- La finition de traitement d'une tâche ;
- …

Dans ARENA, les évènements programmés pour une simulation sont stockés dans un calendrier d'évènements *« event calendar »*. Pour chaque évènement futur, on enregistre dans le calendrier des évènements :

- Une identification de l'entité concernée ;
- Le temps d'exécution de l'évènement ;
- Le type d'évènement.

Les évènements dans le calendrier sont triés selon leurs dates d'occurrence, de manière à ce que les prochains évènements soient les premiers dans la liste. Lorsque la date d'un évènement arrive, son enregistrement est effacé de la liste du calendrier des évènements, et les informations que contient son enregistrement sont utilisées pour produire l'évènement voulu.

viii. L'horloge de la simulation :

Le temps actualisé de la simulation est contenu dans la variable « horloge de la simulation ». Cette variable ne balaie pas toutes les valeurs du temps. Elle interagit avec le calendrier des évènements et elle actualise sa valeur uniquement lorsqu'un évènement se produise. Ceci pour éviter un comptage intégral en vain, puisque généralement nous n'avons pas besoin des valeurs du temps entre évènements. Cette manière de faire présente l'inconvénient de « sauter » les conditions temporelles (voir figure II.13).

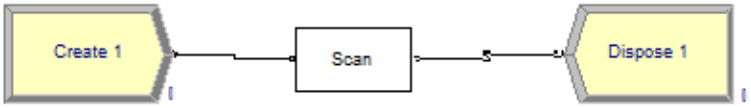


Figure II.13 : exemple d'un saut d'une condition

Dans la figure II.13, le bloc CREATE[1] crée deux entités à un intervalle de 2 unités de temps, dont la première est créée à l'instant 0. La condition du bloc SCAN[2] est comme suit : **TNOW==1.** Dans ce cas, la première entité stockée dans le bloc SCAN[2] doit être libérée à l'instant 1, mais ce n'est pas le cas. Le calendrier des évènements contient, par ordre des dates d'occurrence, les évènements suivant :

- L'évènement déclencheur de la simulation à l'instant 0

- La création de la première entité à l'instant 0
- La disposition de la première tâche (TNOW ≥ 0)

- La création de la deuxième entité à l'instant 2
- La disposition de la deuxième tâche (TNOW ≥ 2)

- La fin de la simulation à l'instant 3 (prédéfinie par l'utilisateur)

Le fait qu'il n'y a pas d'évènements entre l'instant 0 (création de la première entité) et l'instant 2 (création de la seconde entité), l'horloge ne prend que les valeurs : 0 et 2. D'où, la condition ne sera jamais vérifiée. Et par conséquent, l'entité ne sera pas libérée.

Si le cas précédent se présente, nous proposons d'effectuer une combinaison de blocs, créant des évènements aux instants voulus. Le schéma qui suit (figure II.14), permet la création d'événements à un intervalle en fonction de l'unité des temps utilisés. Ainsi, à chaque unité de temps, nous aurons un évènement.

Figure II.14 : création d'événements à intervalle régulier

[1] le bloc crée des entités
[2] le bloc garde les entités arrivées, et les libère si sa condition est vérifiée

C. Le générateur de nombres aléatoires (GNA) :

L'efficacité du simulateur à donner des résultats valides et proche de la réalité, vient de sa capacité à générer l'aléa. La génération de l'aléa avec un ordinateur n'est rien d'autre qu'un algorithme récursif répétant la même séquence à chaque fois. Et c'est pourquoi, il est appelé générateur de nombres pseudo-aléatoires. Le principe consiste à générer des nombres, d'une distribution continue et uniforme entre 0 et 1. En fait les nombres générés sont des observations. Par la suite, et selon la loi choisie par l'utilisateur, les valeurs correspondantes des variables aléatoires seront calculées par la fonction inverse de la loi dans le cas continu, et par une décomposition en intervalles, des probabilités d'occurrence de chaque valeurs que peut prendre la variable aléatoire, dans le cas discret.

Exemple :

- Le cas discret :

Soit à générer une variable aléatoire discrète X prenant les valeurs 0,1 et 2 avec des probabilités telles que :

$$P(X = x) = \begin{cases} 0.2 \ pour \ x = 0 \\ 0.3 \ pour \ x = 1 \\ 0.5 \ pour \ x = 2 \end{cases}$$

L'intervalle [0,1] sera décomposé en trois intervalles : [0, 0.2], [0.2, 0.5] et [0.5, 1]. Ainsi le nombre aléatoire généré N aura une distribution de probabilités telle que :

$$P(N = n) = \begin{cases} 0.2 \quad pour \ n \in [0, 0.2] \\ 0.3 \ pour \ n \in [0.2, 0.5] \\ 0.5 \quad pour \ n \in [0.5, 1] \end{cases}$$

La variable X, quant à elle, prend les valeurs :

$$X = \begin{cases} 0 \quad si \ N \in [0, 0.2] \\ 1 \ si \ N \in [0.2, 0.5] \\ 2 \quad si \ N \in [0.5, 1] \end{cases}$$

- Le cas continu :

Dans le cas continu, nous traduisons la probabilité générée par celle de l'appartenance de la variable à un intervalle donné. La borne inférieure de cet intervalle est égale à 0.

Prenons le cas d'une loi exponentielle de moyenne β. Sa fonction de répartition est donnée par :

$$F(x) = \begin{cases} 1 - e^{-\frac{x}{\beta}} & si \ x \geq 0 \\ 0 & sinon \end{cases}$$

De la probabilité générée N, nous tirons la variable X, la borne supérieure de l'intervalle. Et ceci pour $X \geq 0$:

$$N = 1 - e^{-\frac{X}{\beta}} \quad \Rightarrow \quad \boldsymbol{X = -\beta * \ln (1 - N)}$$

La forme commune des algorithmes de GNA est le générateur linéaire congruent (linear congruential generator LCG). Il génère une séquence Z_1, Z_2, Z_3 … suivant la suite récursive :

$$Z_i = (aZ_{i-1} + c) mod(m)$$

a, c et m sont des constantes, dont leur choix est basé sur des règles empiriques ou théoriques.

La séquence des Z_i est une séquence de nombres entiers. Or, nous avons besoin de générer des nombres compris entre 0 et 1. De ce fait, nous divisons les Z_i par m, ce qui donne toujours un nombre compris entre 0 et 1.

Et notons ce nombre :

$$U_i = Z_i/m.$$

L'inconvénient de cette méthode est que la taille de la séquence générée, sans répétition, est majorée par m. Ce qui conduit à choisir un m assez grand pour éviter la répétition de séquences.

Les dernières versions d'ARENA utilisent un nouveau code de génération de nombre aléatoires, dit CMRG (combined multiple recursive generator). Il consiste à effectuer des récursivités en parallèle de type :

$$A_i = (1403580 * A_{i-2} - 810728 * A_{i-3}) mod(4294967087)$$

$$B_i = (527612 * B_{i-1} - 1370589 * B_{i-3}) mod(4294944443)$$

Et pour chaque Z_i:

$$Z_i = (A_i - B_i)\, mod(4294967087)$$

Et enfin les U_i :

$$U_i = \begin{cases} Z_i / 4294967088 & si\ Z_i \neq 0 \\ 4294967087/4294967088 & si\ Z_i = 0 \end{cases}$$

Cette manière de procéder augmente la longueur du cycle à $\mathbf{3.1 * 10^{57}}$. Celle de l'ancien générateur est de seulement $\mathbf{2.1 * 10^{9}}$, avec une vitesse d'exécution légèrement supérieure à celle du nouveau générateur. A titre illustratif, l'ancien code de génération occupe pendant une demi-heure un PC 600MHz, en ne faisant que générer les nombres aléatoires avant qu'il ne cycle. Le nouveau code occupe le même PC pendant $\mathbf{8.4 * 10^{40}}$ millénaires avant de cycler.

Le cycle complet du nouveau générateur est séparé en $\mathbf{1.8 * 10^{19}}$ flots *« streams »*, chacun d'une longueur $\mathbf{1.9 * 10^{38}}$. Chaque flot est divisé en $\mathbf{2.3 * 10^{15}}$ sous-flots *« substreams »*, chacun de longueur $\mathbf{7.6 * 10^{22}}$. Nous pouvons choisir le flot à utiliser pour générer les nombres aléatoires, en rajoutant son numéro avec les paramètres de la loi de probabilité. Par exemple : nous voulons générer des nombres aléatoires selon la loi de Poisson de moyenne 4, et nous choisissons pour cela le flot numéro **3**. La syntaxe sera donc : POIS(4,**3**).

Contrairement à l'ancien, le nouveau générateur ne stocke pas les germes de chaque flot, mais il les calcule. Et afin d'éviter à ARENA de calculer des germes de flots que nous n'allons pas utiliser, il vaut mieux que les flots choisis soient ordonnés de la manière suivante : 1, 2, 3…

Lorsque nous aurons à effectuer plusieurs réplications, ARENA bascule automatiquement au sous-flot suivant du flot lors de la nouvelle réplication. Ce qui est important, lorsque nous avons à comparer plusieurs modèles et que les modèles à comparer seront simulés avec les mêmes flots de nombres aléatoires.

D. L'application Microsoft Visual Basic :

La flexibilité d'ARENA se consolide par son utilisation du langage de programmation Microsoft Visual Basic (VBA). Ce dernier est intégré dans ARENA, et peut automatiser les applications d'ARENA, comme les autres applications comme MS Excel, AutoCAD…

Le VBA peut être utilisé selon deux manières :

- En insérant un bloc VBA dans le modèle logique, et à chaque fois que l'entité passe par le bloc, le code correspondant au bloc s'exécute ;
- En l'ouvrant avec l'éditeur de VBA.

Les primitives du VBA sont reliées aux différents évènements apparaissant lors d'une simulation ou même avant le début de celle là. Les évènements sont classés en trois grandes familles (toutes les fonctions commencent par *ModelLogic_*) :

- Les évènements d'avant exécution de la simulation : DocumentOpen, DocumentSave…
- Les évènements d'initialisation de la simulation : RunBegin, RunBeginSimulation…
- Les évènements lors de la simulation : VBA_Block_Fire, OnKeyStroke…

La figure II.15 montre les moments opportuns d'implémenter le code.

1. RunBegin
2. ARENA teste et initialise le modèle
3. RunBeginSimulation
4. RunBeginReplication
5. ARENA exécute la réplication
OnKeyStroke
UserFunction
6. RunEndReplication
7. RunEndSimulation
8. ARENA termine la simulation
9. RunEnd

Figure II.15 : les évènements du VBA lors de l'exécution de la simulation

La fonction ModelLogic_VBA_Block_Fire s'exécute à chaque fois qu'une entité passe par le bloc VBA concerné.

Le transfert de données du modèle vers le code est vice-versa s'effectue en déclarant un objet modèle. Après lui avoir affecté les variables d'état du modèle, nous pouvons lire et écrire sur les variables, attributs et accumulateurs du modèle.

Exemples :

- ***oSIMAN.ResourceState(oSIMAN.SymbolNumber("nom de la ressource"))*** : contient l'état d'une ressource.

- ***oSIMAN.QueueNumberOfEntities(oSIMAN.SymbolNumber("nom de la file d'attente"))*** : contient le nombre d'entités dans une file d'attente.
- ***oSIMAN.VariableArrayValue(oSIMAN.SymbolNumber("nom de la variable globale"))*** : contient la valeur d'une variable globale.
- ***oSIMAN.EntityAttribute(oSIMAN.ActiveEntity, oSIMAN.SymbolNumber("nom de l'attribut"))***: contient la valeur d'un attribut.

Exemples d'utilisation du langage VBA :

```
" Déclarer le modèle oSIMAN comme un objet modèle
Dim oSIMAN As Arena.SIMAN
Private Sub ModelLogic_RunBegin()
  " Affichage d'un message « Initialisation du modèle »
  MsgBox " Initialisation du modèle "
End Sub
Private Sub ModelLogic_RunBeginSimulation()
  " Ajouter les variables d'état au modèle oSIMAN à chaque début de simulation
  Set oSIMAN = ThisDocument.Model.SIMAN
End Sub
Private Sub ModelLogic_RunBeginReplication()
  " À chaque début de replication, un message « début de la réplication » est affiché
  MsgBox " début de la réplication "
End Sub
Private Sub VBA_Block_1_Fire()
  " Déclarer c comme un entier statique pendant toute la simulation
  Static c as Integer
  c = c + 1
 If c5 Then
    Msgbox "5 tâches sont déjà passées par ce bloc"
  End If
  " Afficher un message donnant le temps de passage d'une entité à travers le bloc VBA
  Msgbox "TNOW= " & oSIMAN.RunCurrentTime
End Sub
```

Pour plus d'informations sur les fonctionnalités d'ARENA, se référer à [Kel 03].

IV. Conclusion :

Considérer chaque machine comme une unité autonome capable de résoudre un problème, réduit considérablement la complexité du problème d'ordonnancement posé. Nous avons vu que l'apport des systèmes Multi Agents, issus des travaux en Intelligence Artificielle Distribuée, ne se limite, cependant, pas à un ensemble de concepts attractifs pour appréhender la nature foncièrement distribuée d'un problème d'ordonnancement. Les SMA mettent également à disposition des outils de modélisation et de conception éprouvés, très avantageux dans les problèmes d'ordonnancement.

Nous avons vu pour la simulation, que la modélisation constitue le fondement même de cette dernière ce qui nous a menés à la description des différentes étapes de construction d'un modèle. Puis, nous avons présenté ARENA, le langage de simulation utilisé dans ce travail.

CHAPITRE III : Modélisation des approches de résolution

I. Introduction

II. Présentation de la problématique

III. Modélisation des approches de résolution proposées dans [Bou 07]

1. Modèle Arena décrivant le fonctionnement de l'algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente

A. Partie tâches programmables

B. Partie tâches aléatoires

C. Partie traitement des tâches

D. Remarque

2. Modèle Arena décrivant le fonctionnement de l'algorithme d'ordonnancement temps-réel au plus tôt dans le cas d'une file d'attente

3. Modèle Arena décrivant le fonctionnement de l'algorithme d'ordonnancement temps-réel au plus tard dans le cas de deux files d'attente

IV. Modélisation de l'approche collaborative

1. Pourquoi une approche collaborative

2. Présentation des primitives retenues dans le cadre de la communication inter-agent

3. Proposition d'une approche collaborative pour la résolution du problème étudié

4. Définition des règles de priorités arrêtées

5. Modélisation de l'approche proposée sous ARENA

V. Vérification et validation des modèles obtenus

VI. Conclusion

I. Introduction :

Ce chapitre est consacré à la présentation des différents modèles que nous avons développés, pour les différentes approches de résolution présentées dans ce mémoire.

Ainsi, ce chapitre se décompose comme suit : la première section portera sur la présentation du problème étudié. La deuxième section sera consacrée aux approches de résolution proposées dans [Bou 07]. Dans la troisième section, nous présenterons l'approche collaborative que nous avons développée et le modèle ARENA correspondant.

II. Présentation de la problématique :

Le problème traité dans notre travail, consiste en l'ordonnancement sur deux machines identiques en parallèle de deux types de tâches :

- Programmables : dont les paramètres (dates de disponibilité, durées opératoires et dates échues) sont connus à l'avance. Ce qui fait que leur ordonnancement peut être planifié à l'avance.
- Aléatoires : arrivant en temps réel, leur ordonnancement se fait en les plaçant dans les marges de l'ordonnancement des tâches programmables.

Les règles à respecter sont :

- Les machines ne peuvent traiter qu'une tâche à la fois (machines disjonctives).
- Les tâches ne sont pas préemptives.
- L'ordonnancement des tâches aléatoires ne doit pas retarder les tâches programmables.
- Le nombre de tâches aléatoires exécutées est à maximiser
- Le C_{max} de l'ordonnancement est à minimiser.

L'heuristique de résolution de ce problème consiste à ordonnancer d'abord les tâches programmables, et d'insérer par la suite les tâches aléatoires dans les marges disponibles dans l'ordonnancement des tâches programmables. Le second ordonnancement se fait en temps-réel et se base sur des règles de priorité.

L'ordonnancement statique des tâches programmables est établi de la manière suivante : Toutes les tâches sont ordonnancées selon leurs dates de disponibilité (de la plus petite à la plus grande). Ensuite les tâches sont affectées aux machines « en escalier » comme le montre la figure III.1. Enfin, et pour chaque machine, les tâches présentant les mêmes dates de disponibilité sont triées selon leurs dates échues (de la plus petite à la plus grande).

La plus grande *date de fin au plus tôt*[1] des tâches représente la borne supérieure. Cette dernière peut augmenter jusqu'à la plus grande *date de fin au plus tard*[2].

[1] Date avant laquelle une tâche ne peut être terminée.
[2] Date limite d'achèvement d'une tâche.

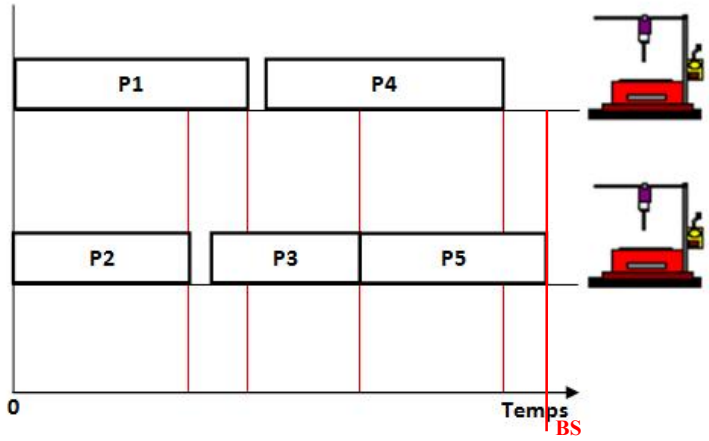


Figure III.1 : algorithme de l'ordonnancement statique

L'ordonnancement des tâches aléatoires s'effectue en temps réel, lorsqu'une ou plusieurs machines sont libres à un instant donné (TNOW), et qu'il y a des tâches aléatoires dans la file d'attente.

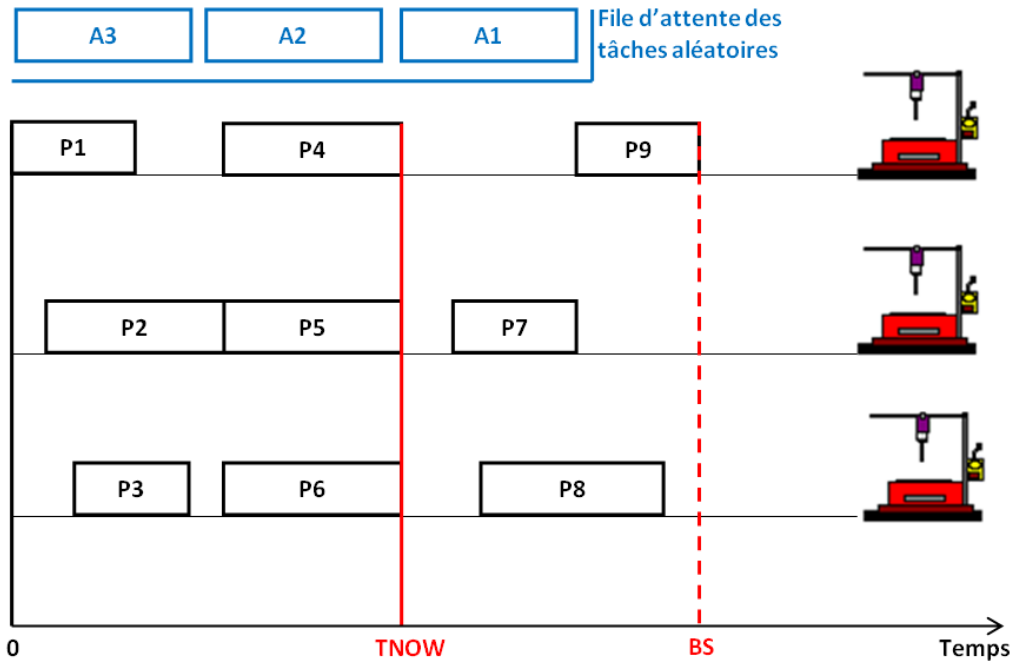


Figure III.2 : Le cas de placement des tâches aléatoires

L'insertion d'une tâche sur une marge d'une machine donnée, peut ne pas décaler la tâche programmable suivante, la décaler sans décaler la borne supérieure ou décaler la borne supérieure. La figure III.3 montre les possibilités d'insertion d'une tâche aléatoire.

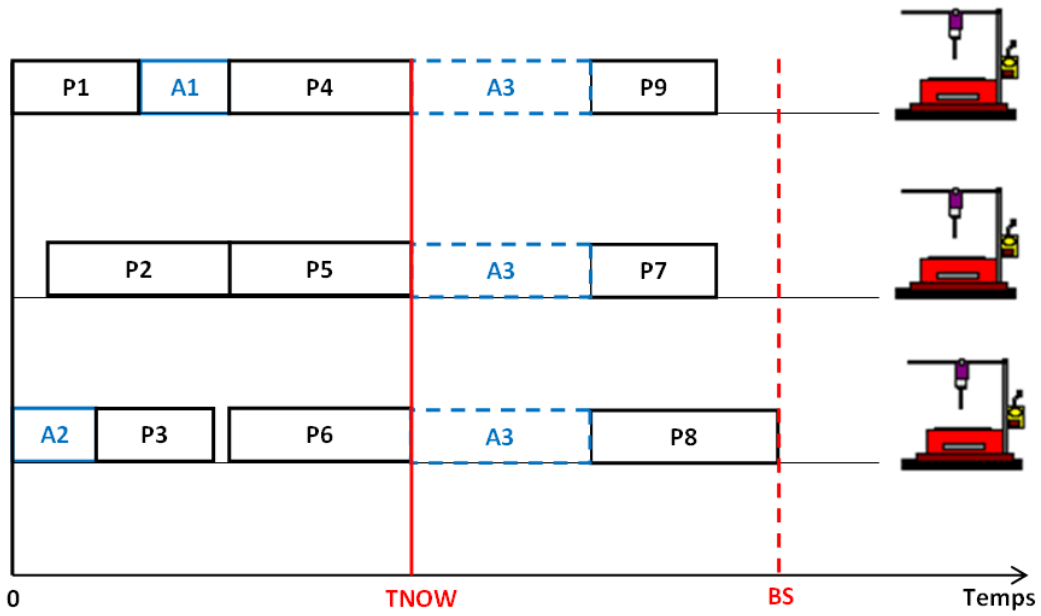


Figure III.3 : Les possibilités d'insertion d'une tâche aléatoire

Il peut s'agir aussi de choisir entre plusieurs tâches aléatoires à ordonnancer, ses tâches sont alors affectées aux machines selon des règles de priorité caractérisant chaque approche que nous allons présenter après.

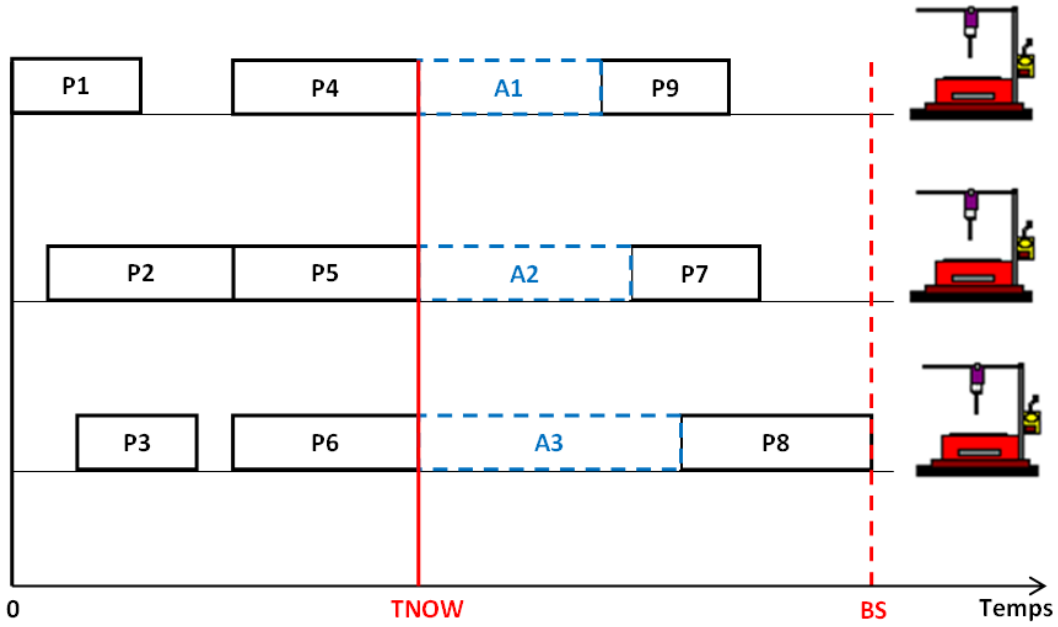


Figure III.4: Le cas de plusieurs tâches aléatoires

On passe, dans ce qui suit, à la présentation des différents modèles ARENA construit.

III. Modélisation des approches de résolution proposées dans [Bou 07] :

Dans cette section, on s'intéresse à la modélisation des différentes approches de résolution proposées dans [Bou 07]. Le fonctionnement de ces dernières est repris dans la figure III.5.

Les éléments principaux à retenir pour la construction de notre modèle sont :

- Chaque tâche est exécutée sur une ressource seulement.
- Toutes les ressources sont identiques.
- Les tâches sont non-préemptives.
- Il y'a deux types de tâche : programmables et aléatoires.
- Il n'y a pas d'interdépendance entre les tâches.
- Toutes les ressources sont disponibles à l'instant zéro.
- Le temps de transport entre les ressources est négligeable.
- Les temps de préparation des ressources et des tâches sont négligeables.
- Les tâches ne sont pas nécessairement exécutées dés leurs apparitions.
- Les capacités de stockage des files d'attente sont illimitées.
- Les pannes ne sont pas considérées dans le cadre du problème posé.
- Il y a trois files d'attente. Une file d'attente pour les tâches aléatoires et une file d'attente des tâches programmables pour chaque machine.

Les différents attributs et variables définis dans le cadre de cette étude, sont :

- IndexP : cet attribut est alloué à chaque tâche programmable afin d'identifier l'ordre de son apparition.
- IndexA : cet attribut est alloué à chaque tâche aléatoire afin d'identifier l'ordre de son apparition.
- r : attribut spécifiant la date de disponibilité de la tâche.
- pr : attribut spécifiant la durée opératoire de la tâche.
- dl : attribut spécifiant la date échue de la tâche.
- ta_i : attribut spécifiant le début au plus tôt de la tâche programmable i.
- tb_i : attribut spécifiant le début au plus tard de la tâche programmable i.
- mach : $\in \{0, 1, 2, 3\}$. Cet attribut indique sur quelle machine sera traitée la tâche (1 pour la machine 1, 2 pour la machine 2, 0 la tâche est stockée en attendant une prochaine vérification et 3 la tâche est rejetée).
- type : $\in \{1,2\}$. Cet attribut nous permet de distinguer entre les tâches programmables et aléatoires (1 pour les tâches programmables et 2 pour les tâches aléatoires).
- X6 : cette variable nous indique le nombre de tâches aléatoires traitées.
- Makespan : variable indiquant le C_{max}.

Dans ce qui suit, on présentera les différents modèles que nous avons développés avec ARENA et qui reprennent le fonctionnement des différents algorithmes proposés dans [Bou 07] (les trois premiers algorithmes seulement, puisqu'on a relevé dans le premier chapitre que le dernier algorithme était une autre formulation du premier) afin de pouvoir procéder à la simulation dans le dernier chapitre.

Il y a des parties dans les modèles développés qui sont les mêmes, donc elles ne seront pas reprises à chaque fois, on se contentera d'en faire allusion pour pouvoir suivre le mécanisme du modèle.

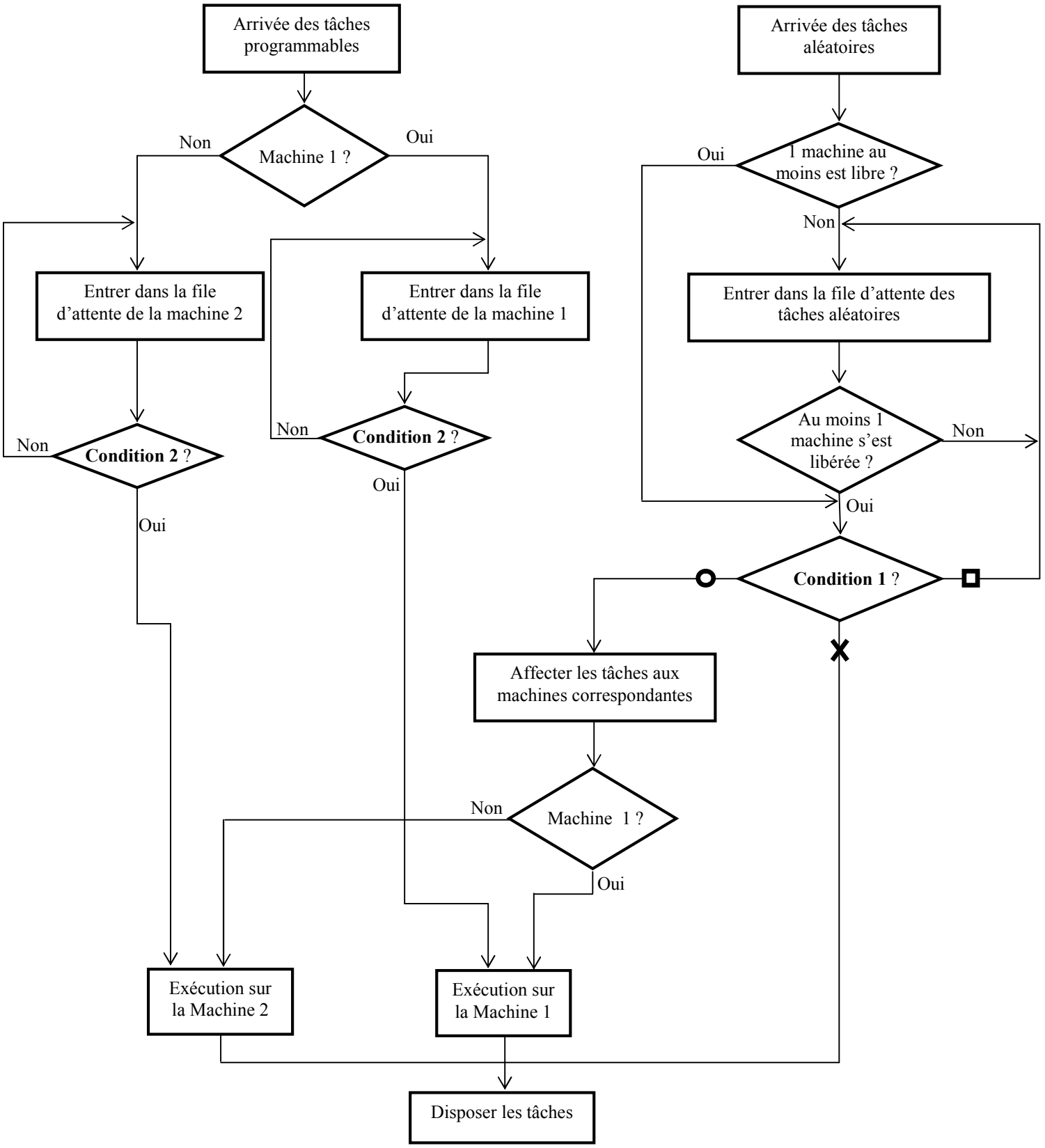
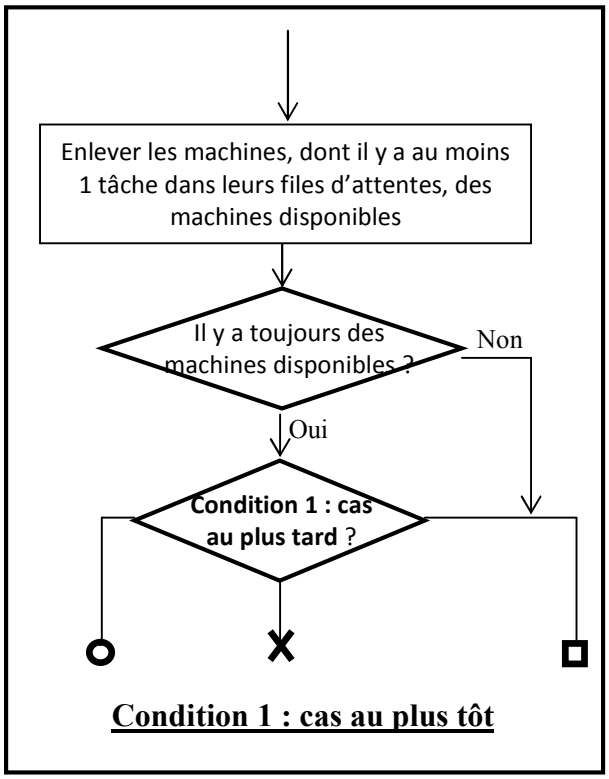
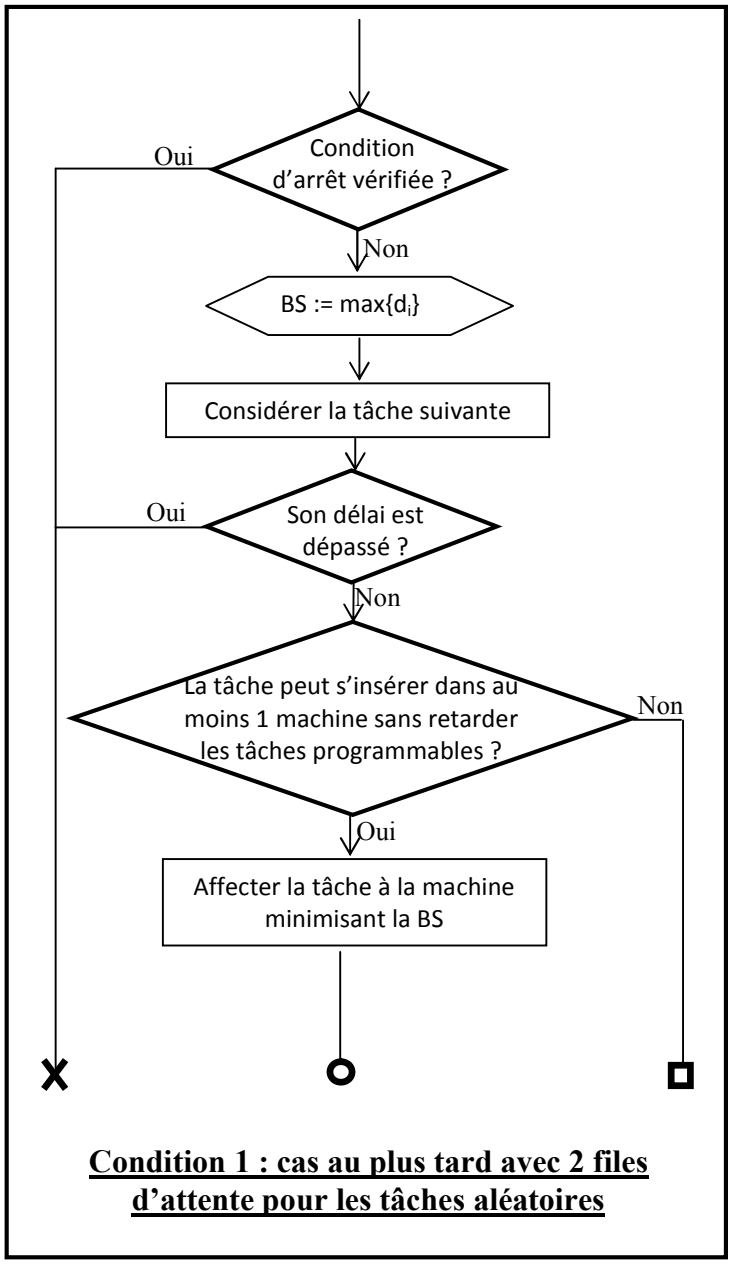


Figure III.5 : La forme commune aux trois algorithmes proposés par [Bou 07]

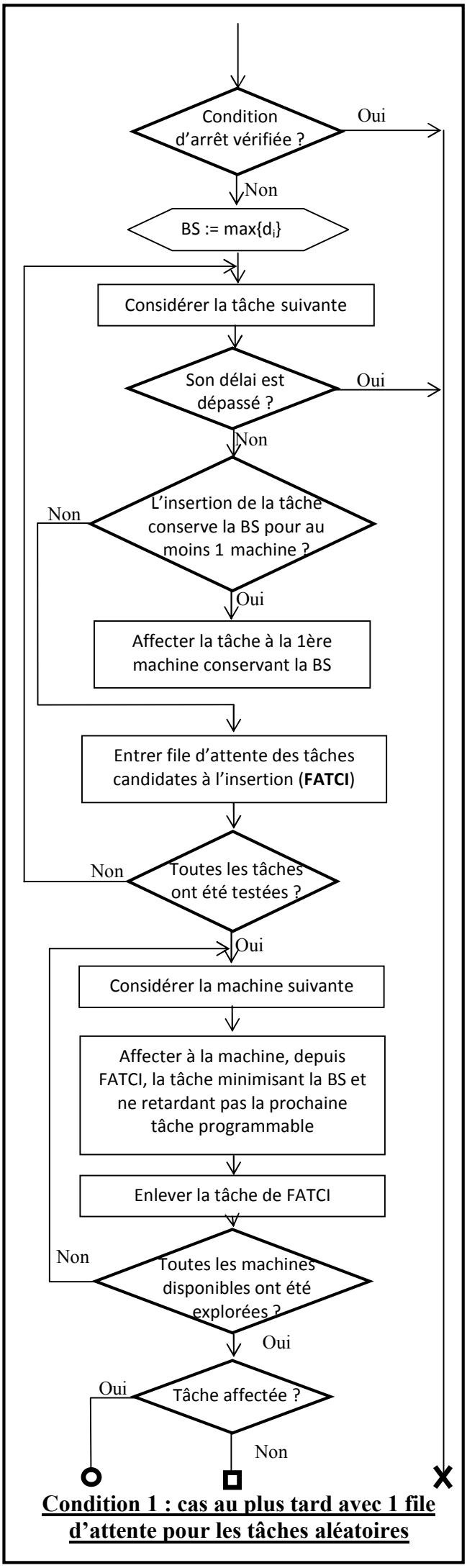
Cas au plus tard avec 1 file d'attente et au plus tard avec 2 files d'attente :
Condition 2 = il n'y a pas de tâches aléatoires acceptées sur cette machine.
Cas au plutôt:
Condition 2 = la file d'attente de la machine concernée n'est pas vide et la machine est libre.



Condition 1 : cas au plus tôt



Condition 1 : cas au plus tard avec 2 files d'attente pour les tâches aléatoires



Condition 1 : cas au plus tard avec 1 file d'attente pour les tâches aléatoires

1. Modèle Arena décrivant le fonctionnement de l'algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente :

Le modèle Arena proposé comporte trois parties principales, l'une qui concerne le parcours des tâches programmables, l'autre pour les tâches aléatoires et enfin une partie commune entre les deux types de tâches pour leur traitement.

A. Partie tâches programmables :

Cette partie comporte deux étapes, à savoir, création des entités puis l'attente de l'exécution.

- En ce qui concerne la première étape :

Sachant que les tâches programmables sont connues à priori, on a mis en place une combinaison entre le module Create et Assign, tel que, toutes les entités qui vont représenter les tâches programmables, par la suite, sont créées au même instant (t=0). Les caractéristiques de chaque tâche (le type de la tâche, IndexP, r, pr, dl) lui sont assignées via le module Assign.

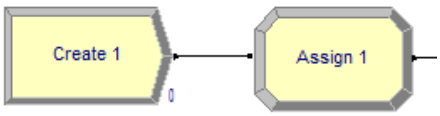


Figure III.6 : modélisation de la création des entités qui vont représenter les tâches programmables par la suite

Le module Assign a aussi pour fonction, l'initialisation des différents attributs : mach, ta_i, tb_i…

Ensuite, on a la combinaison des deux blocs VBA et Scan.

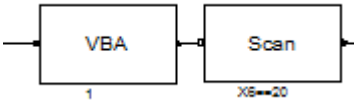


Figure III.7 : stockage des données relatives aux tâches programmables

Le Scan a pour fonction d'attendre que toutes les tâches soient passées par le bloc VBA avant de les relâcher. Le bloc VBA, quant à lui, a pour fonction de stocker les différentes données relatives aux tâches programmables dans une matrice « d » afin de faciliter leur traitement tout au long du modèle.

Dès que la dernière tâche programmable passe par ce bloc, une séquence d'opérations est exécutée comme suit :

- Affectation des tâches aux machines (1 ou 2) ;
- Calcul des ta_i ;
- Création de deux matrices « m1 » et « m2 », contenant les caractéristiques des tâches qui s'exécutent sur la machine 1 et 2 respectivement ;
- Calcul des tb_i, le calcul des durées entre création qui va nous servir plus tard à simuler la création des tâches programmables ;
- Calcul de la borne supérieure (le C_{max} initial).

On clôt cette phase de création des tâches programmables par la séquence de blocs suivante :

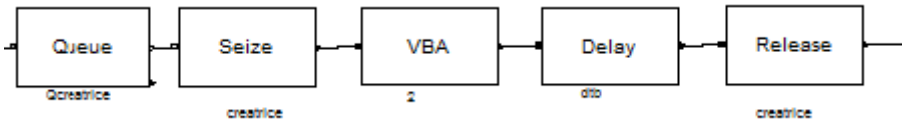


Figure III.8 : modélisation de la création des tâches programmables

Le bloc VBA présent dans cette séquence, nous permet d'attribuer les différentes valeurs calculées aux attributs.
Cette séquence a pour mission la simulation de la création des tâches programmables grâce au calcul des durées entre création qu'on a calculé auparavant et qu'on met dans le Delay.

On a par la suite un Branch, qui sépare les tâches programmables selon qu'elles doivent s'exécuter sur la machine 1 ou 2.

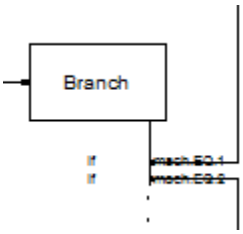


Figure III.9 : modélisation de la séparation des tâches selon la machine sur laquelle elles seront exécutées

Les séquences qui suivent le Branch (selon la première ou la seconde condition) sont les mêmes (i.e. : elles ont la même fonction). Ainsi, on se limite à la description de l'une d'elles seulement.

- On passe à la seconde étape « attente » :

La séquence suivante représente la file d'attente pour la machine 1(2) des tâches programmables :

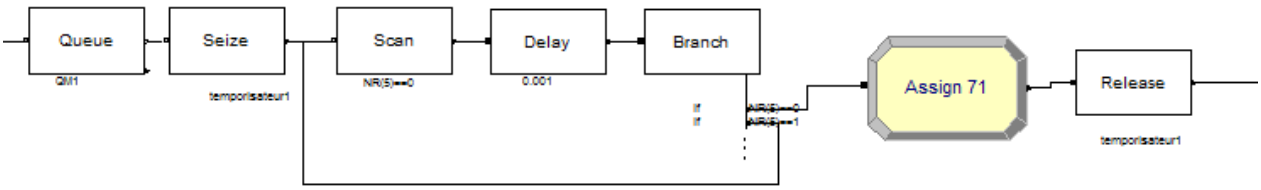


Figure III.10 : modélisation de la file d'attente de la machine 1 (2)

Elle est développée comme suit :

Une fois la tâche créée et assignée à une machine pour son exécution, on la stocke dans le Queue QM1(2).

Puis on a une combinaison de blocs qui ont pour fonction de palier à un problème d'instantanéité (car si on conditionne la libération d'une entité par le passage d'une autre entité dans un certain bloc du modèle, le résultat est vrai une fois sur deux, puisque deux entités qui peuvent prétendre à être exécutées par une ressource au même moment, ARENA dans ce cas, donne la priorité aux entités qui passent par la même série de blocs à chaque fois,

alors que, dans notre cas, ceci dépend d'un certain nombre de facteur. Cette particularité se présente dans certaines situations mais dans d'autres pas. Tout en précisant qu'il s'agit d'événements qui se produisent au même moment), c'est ce qu'on a appelé un temporisateur (il donne le temps aux tâches aléatoires de vérifier si elles peuvent être exécutées, ou pas, par une machine (machine 1(2)) est libre, d'où la présence du Scan.

Le Branch quant à lui, nous permet dans le cas où une tâche aléatoire a été acceptée (i.e. : la ressource est occupée et NR ()=1) de remettre la tâche programmable en attente dans le Scan en attendant la prochaine libération de la ressource. Dans le cas contraire, si aucune tâche aléatoire n'a été acceptée, alors, la tâche programmable est libérée, sa durée opératoire est corrigée en lui soustrayant le temps qui a servi à la temporisation grâce au module Assign. Puis on libère le temporisateur pour la prochaine tâche programmable.

B. Partie tâches aléatoires :

Pour suivre le cheminement logique de l'idée globale développée dans ce travail, on va présenter la partie tâches aléatoires.

Cette partie comporte trois étapes, à savoir, création des entités puis l'attente de la vérification et enfin la vérification.

- Etape de création des tâches aléatoires :

A la différence des tâches programmables, nous n'avons aucune donnée à priori sur les tâches aléatoires, nous avons mis en place une combinaison entre le module Create et Assign, tel que, les tâches aléatoires sont créées selon une loi de probabilité (EXPO) dont les paramètres sont à déterminer (le r dans ce cas constitue la date de création de la tâche). Les autres caractéristiques de chaque tâche (le type de la tâche, IndexA, pr, dl) lui sont assignées via le module Assign.

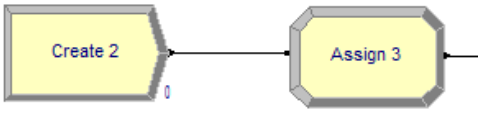


Figure III.11 : modélisation de la création des tâches aléatoires

Le module Assign a aussi pour fonction, l'initialisation de l'attribut mach.

On a ensuite, un Branch dont la fonction est de permettre, dans le cas où une ressource a été libérée, en plus aucune tâche aléatoire ne satisfait les conditions de son admission et où il n'y a aucune tâche programmable à exécuter (i.e. : la ressource est libre depuis un certain moment), de faire des tests de vérification sur une nouvelle tâche aléatoire qui vient d'apparaitre, sans passer par l'attente.

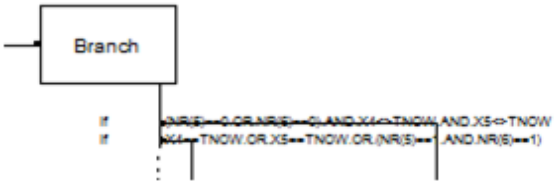


Figure III.12 : séparation des tâches aléatoires qui interviennent alors que les machines sont occupées ou pas

- Attente de la vérification :

Dans l'autre cas, la tâche aléatoire est admise dans une file d'attente, en attendant la libération d'une ressource.

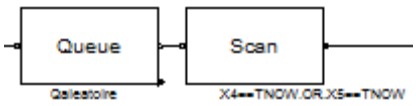


Figure III.13 : modélisation de l'attente de tâches aléatoires

Le Scan relâche les tâches aléatoires dès que l'une des ressources se libère (NR (machine 1 ou 2)=0).

- Vérification des tâches aléatoires :

Cette étape nous permet dans un premier temps de déterminer les tâches aléatoires susceptibles d'être exécutées sur la (les) ressource (s) libre (s), dans un second temps, on sélectionne la tâche aléatoire à exécuter selon les critères retenus.

Pour sélectionner les tâches candidates, on a utilisé le même procédé défini précédemment pour la temporisation, afin de permettre à toutes les tâches aléatoires présentes dans le Queue d'être vérifiées.

Figure II.14 : modélisation des tests d'insertion des tâches aléatoires

Le bloc VBA est chargé de vérifier quelle machine est libre (on peut avoir, une machine libre ou les deux), puis d'exécuter une série d'opérations selon le cas identifié. A la sortie de ce bloc on aura les tâches candidates et celles qui n'ont pas été retenues.

Par la suite, on a un autre bloc VBA qui sélectionne la tâche optimale (selon les critères) à exécuter sur la ressource.

Figure III.15 : sélection de la tâche optimale

On a après, un Branch, qui oriente les différentes tâches aléatoires selon le résultat de la vérification.

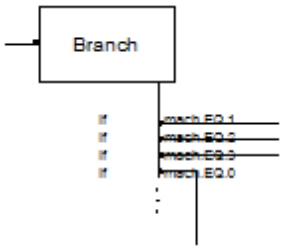


Figure III.16 : modélisation du transfert des tâches aléatoires à exécuter vers la machine cible

Dans le cas où l'attribut mach, de la tâche aléatoire est égal à 1, cette dernière est affectée à la machine 1.
S'il est égal à 2, on l'affecte à la machine 2.
S'il est égal à 3, cela signifie que cette tâche aléatoire ne peut plus être exécutée dans son intervalle d'exécution, ainsi, on l'élimine.
S'il est égal à 0, cela signifie que la tâche aléatoire n'a pas été acceptée, mais qu'elle peut encore prétendre à être exécutée. Dans ce cas, elle est redirigée vers la fille d'attente.

C. Partie traitement des tâches :

La dernière partie, c'est la partie commune qui s'occupe du traitement des tâches (i.e. : nos deux machines).

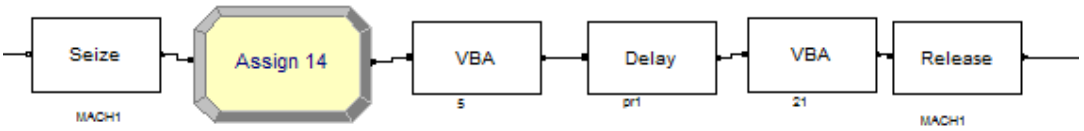


Figure III.17 : modélisation du traitement des tâches

Dans cette partie, on affecte une tâche à une ressource, ensuite, avec le module Assign, on enregistre le TNOW+pr1 (i.e. : à quel moment, la machine sera libre).
Les deux blocs VBA qui suivent, calculent le nombre de tâches programmables et aléatoires déjà exécutées, et nous permettent de représenter graphiquement l'exécution des tâches.

Une fois la tâche exécutée, on l'oriente vers un module Dispose.

Figure III.18 : élimination des tâches traitées du système

D. Remarque :

La combinaison des modules Create-Dispose suivante, nous permet d'avoir un TNOW actualisé à tous les moments où il y a possibilité d'apparition d'un événement, puisqu'on simule des cas aléatoire. Ceci, dans le but de remédier au fait que Arena ne calcule le TNOW que si un événement s'est produit.

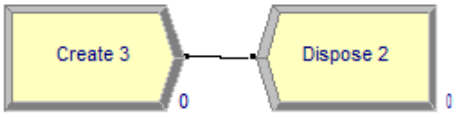


Figure III.19 : calcul du TNOW

Pour cela, on crée des entités avec ce Create avec un intervalle le plus petit qu'on aura à utiliser.

2. Modèle Arena décrivant le fonctionnement de l'algorithme d'ordonnancement temps-réel au plus tôt dans le cas d'une file d'attente :

A part quelques modifications dans la partie tâches programmables et quelques changements au niveau du lancement des tests sur les tâches aléatoires dans la partie tâches aléatoires, ce modèle est en tout autre point similaire au premier.

Pour la partie tâches programmables, les modifications consistent en la suppression de l'attente qui consistait à attendre le test des tâches aléatoires, dans ce cas la tâche programmable était exécutée seulement si aucune tâche aléatoire n'était admise, grâce au changement apporté. Dès l'arrivée de la tâche programmable, elle est exécutée sur la machine si cette dernière est libre, sinon elle est stockée dans une file d'attente et elle sera exécutée juste après l'exécution de toutes les tâches programmables apparues avant elle. C'est précisément le principe de l'algorithme au plus tôt.

Pour la partie tâches aléatoires, le changement consiste en l'introduction d'une temporisation supplémentaire ayant pour but de donner la priorité aux tâches programmables.

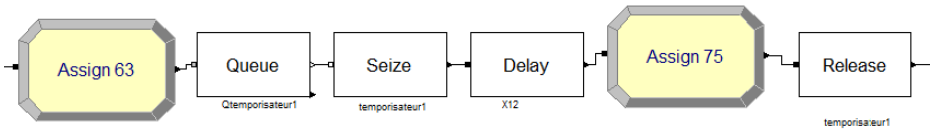


Figure III.20 : modélisation de l'attente des tâches aléatoires

Cette série de blocs (figure III.20 est introduite juste avant les tests d'insertion des tâches aléatoires. Elle sert à vérifier si aucune tâche programmable ne se présente, pour exécution, avant de pouvoir lancer les tests.

3. Modèle Arena décrivant le fonctionnement de l'algorithme d'ordonnancement temps-réel au plus tard dans le cas de deux files d'attente :

Cet algorithme permet d'affecter les tâches aléatoires aux files d'attente des ressources en les insérant directement dans la position d'exécution. A partir de là, on peut encore une fois déduire, après une analyse détaillée du principe de fonctionnement de cet algorithme, qu'il suffit juste d'apporter quelques modifications au premier modèle, pour obtenir un modèle qui reprend, fidèlement, le fonctionnement de ce dernier. Pour cela, on introduit quelques changements dans le modèle précédent au niveau du bloc VBA chargé des tests d'insertions des tâches aléatoires. Quant à la partie tâches programmables aucune modification ne s'avère nécessaire, puisque, dans cet algorithme, aussi, on a un ordonnancement au plus tard.

Comme nous venons d'expliquer, si les tâches aléatoires vérifient les conditions d'insertions à leur **apparition**, elles sont **directement insérées** dans la position d'exécution, cela signifie, par rapport au premier modèle, qu'on ne sélectionne plus la tâche aléatoire à exécuter parmi les différentes tâches candidates, mais c'est la première tâche à être acceptée qui est admise pour exécution (i.e. : on remplace la sélection de la tâche optimale parmi toutes les tâches candidates, par la règle FIFO).

IV. Modélisation de l'approche collaborative :

L'objectif principal attendu de l'approche collaborative, est de distribuer les calculs tout en coordonnant les actions des différents agents afin d'atteindre un but commun. Ceci est rendu possible par l'autonomie des centres de décision qui peuvent, à leur niveau et selon leurs moyens, effectuer les différents calculs.

1. Pourquoi une approche collaborative :

Même si le modèle au plus tard dans le cas d'une file d'attente développé dans le chapitre I, contient les prémices d'une approche collaborative, au moment de son application (la modélisation de l'atelier sous cette approche), la difficulté combinatoire pour deux machines en parallèle (seulement) s'est affirmée. L'utilisation de la même approche pour 3 machines ou plus s'avère être très difficile, voir impossible.

Pour remédier à ce problème, l'approche collaborative est plus que recommandée, du fait que la difficulté combinatoire sera absorbée par la collaboration (nous allons décrire cela par la suite).

On peut résumer les avantages apportés par cette approche comme suit :

- ✓ Réduction de la difficulté combinatoire au niveau de la conception ;
- ✓ Possibilité de généraliser l'approche aux cas de plusieurs machines ;
- ✓ Convenance des SMA avec les ateliers de production.

2. Présentation des primitives retenues dans le cadre de la communication inter-agent :

L'envoi et la réception des messages à destination ou en provenance des autres agents, sont gérés par le module "Communication". Cette gestion concerne le médium linguistique de la transmission des messages, évoqué dans la section 3 de ce chapitre (l'expression et l'interprétation d'un message). Nous avons retenu, comme langage d'expression des messages le langage de communication ACL de la FIPA. ACL présente sur KQML l'avantage d'une sémantique plus rigoureuse ainsi qu'un effort important de standardisation.

Dans le cadre de ce travail et vis-à-vis des besoins exprimés par notre modèle, nous avons retenu les primitives suivantes :

- ➢ Primitives de l'agent perturbé :
  - ✓ Query : demande d'information à un autre agent. La réponse de l'agent coopérant se fait par un "Inform".
  - ✓ Request : demande envers les agents coopérant de s'engager pour l'exécution de certaines actions.
  - ✓ Confirm : cette primitive est réservée à la confirmation d'un engagement entrepris par un agent coopérant.

- ➢ Primitives de l'agent coopérant :
  - ✓ Inform : réponse au Query reçu par un ensemble d'information.
  - ✓ Agree : cette primitive, émise par l'agent coopérant, indique qu'il accepte de prendre en charge une requête qu'il a reçue à travers le Request.

- ✓ Refuse : cette primitive, émise par l'agent coopérant, indique qu'il refuse de prendre en charge une requête qu'il a reçue à travers le Request.

3. Proposition d'une approche collaborative pour la résolution du problème étudié :

Dans ce qui suit, nous proposons une formalisation de la négociation collaborative entre machines, afin de choisir la meilleure combinaison de tâches aléatoires.

Le processus de négociation est enclenché lorsqu'il y a des tâches à tester (libération de machines ou arrivées de tâches avec au moins une machine libre). L'initialisation du processus consiste en une analyse de la file d'attente des tâches aléatoires afin de déterminer **pour chaque machine**, les tâches de la file respectant les conditions d'insertion[1]. Ces tâches seront ensuite triées selon leurs priorités. Par exemple : sur une machine donnée et parmi les tâches vérifiant la condition d'insertion, l'insertion d'une tâche n'augmentant pas la borne supérieure du Makespan est prioritaire à l'insertion d'une autre tâche qui l'augmenterait.

Ces calculs sont répartis sur les machines. Chaque machine a son propre code de calcul, des tâches aléatoires pouvant s'insérer sur cette dernière, et celui du tri des tâches selon leurs priorités. A la fin du calcul, chaque machine aura une liste de tâches aléatoires candidates triées selon leurs priorités.

L'agent décideur est la dernière machine (dans le cas de quatre machines, c'est la machine 4 qui décide). Le processus de négociation s'effectue comme suit :

Etape 1 :

- *L'agent décideur envoie des messages de demande d'information « Query » aux autres agents. Le contenu du message est : quelle est la nouvelle combinaison de tâches aléatoires ?*

Etape 2 :

- *Tous les autres agents renvoient un message d'information « Inform » à l'agent décideur. Le message contient la combinaison actuelle des tâches aléatoires.*
- ***Aller à l'étape 3***

Etape 3 :

- *L'agent décideur effectue un test[2] sur la combinaison.*

Si** la combinaison est acceptée, **Alors

- *L'agent décideur s'engage pour effectuer la tâche qu'il a proposée ;*
- *L'agent décideur envoie aux autres agents une demande « Request », leurs demandant de s'engager pour l'exécution de la combinaison.*
- *Les autres agents répondent avec un message d'acceptation « Agree » à l'agent décideur.*
- ***Aller à l'étape 4.***

[1] Elles ne dépassent pas leurs délais, et leurs insertions sur les machines, ne retardent pas les tâches programmables suivantes.

[2] Le test commence par vérifier si les tâches affectées sont différentes. Ensuite il entame une série de test que nous détaillerons dans (**Chap.III** :V.4).

Sinon

- ***Aller à l'étape 4***

Etape 4 :

Si *l'agent décideur n'a pas encore puisé toutes ses tâches,* ***alors***

- *L'agent décideur passe à la prochaine tâche dans sa liste ;*
- ***Aller à l'étape 3****.*

Sinon *:* ***aller à l'étape 5***

Etape 5 :

- *L'agent actuel envoie une demande « Request » à l'agent qui le précède, lui demandant de passer à sa prochaine tâche.*

Si *ce dernier a encore des tâches à proposer,* ***alors***

- *Il passe à la tâche suivante dans sa liste en envoyant un message d'acceptation « Agree » à l'agent décideur.*
- ***Aller à l'étape 6.***

Sinon

Si *tous les agents ont été interrogés,* ***alors***

- *Envoie un message de refus « Refuse » à l'agent décideur*
- ***Aller à l'étape 6.***

Sinon

- *Aller à l'****étape 5***

Etape 6 :

Si *le message est « Agree »,* ***alors***

- *Aller à l'étape 1.*

Sinon

- *L'agent décideur envoie un message de confirmation « Confirm » aux autres agents, afin de confirmer la dernière demande acceptée.*
- ***Fin du processus de négociation***

Les schémas suivants illustrent le processus de négociation, en pas-à-pas. Pour des raisons illustratives, nous nous sommes contentées de quatre machines :

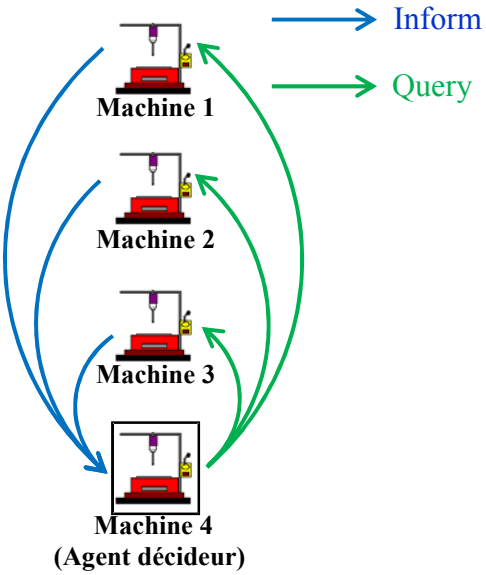


Figure III.21 : Etapes 1 et 2

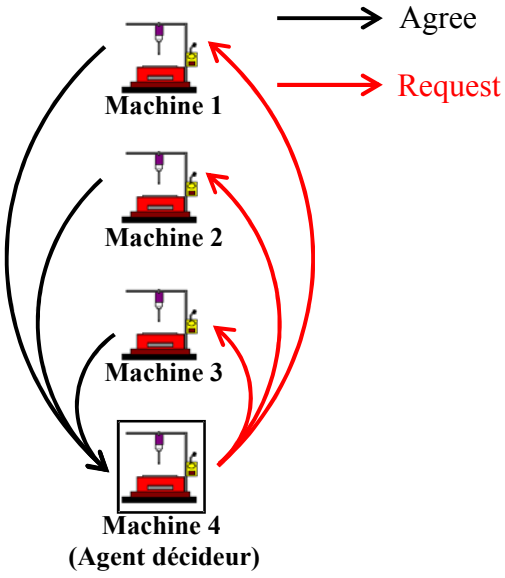


Figure III.22 : Etape 3. Le cas d'acceptation de la combinaison

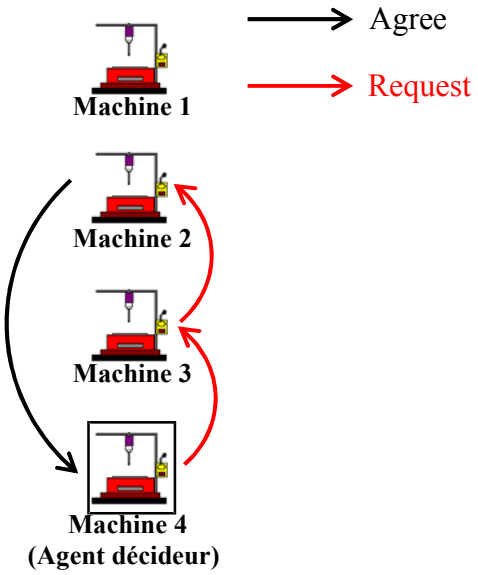


Figure III.23 : Etape 5. Le cas où il y a encore des combinaisons à tester

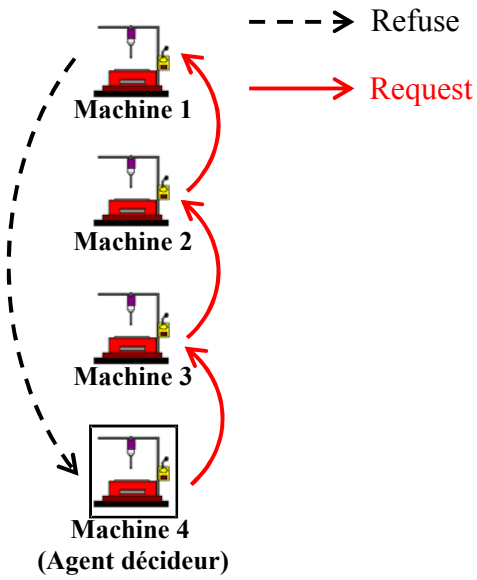


Figure III.24 : Etape 5. Le cas où il n'y a plus de combinaisons à tester

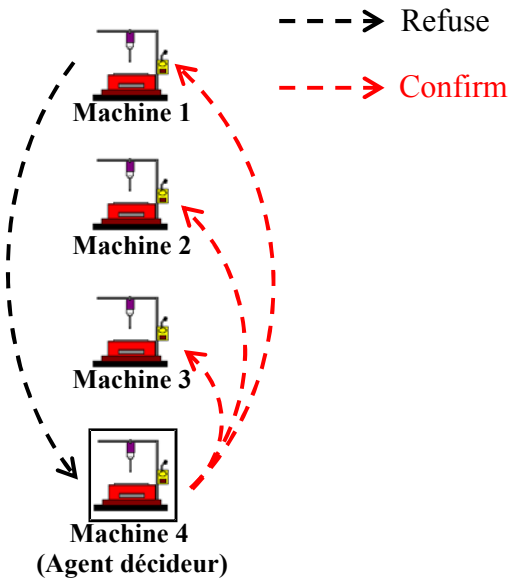


Figure III.25 : Etape 6. La confirmation de la demande

Cet ordonnancement a été obtenu après sélection de la tâche aléatoire la plus adaptée à être exécutée sur chacune de ces machines selon les critères arrêtés (qui œuvrent pour l'atteinte d'un objectif local, qui, grâce à la collaboration, concorde avec l'objectif global).

4. Définition des règles de priorités arrêtées :

Pour la sélection des tâches aléatoires à traiter sur les machines, on a défini une séquence de règles consécutives, dont la première règle satisfaite sur cette séquence, arrête le test par rapport aux autres (restantes).

Les règles définies sont :

1) Le nombre de tâches aléatoires à traiter sur les machines libres : on cherche à maximiser ce nombre (qui fait partie de nos objectifs). Partant de là, on choisit, les tâches aléatoires de sorte qu'un maximum de machines puissent traiter des tâches aléatoires (dans la mesure du possible). Dans le cas où on a plusieurs possibilités, on passe à la règle 2 ;
2) La borne supérieure : on sélectionne entre les tâches candidates pour être exécutées sur une machine, celle qui minimise cette borne. Dans le cas où on a plusieurs tâches qui ont la même action (optimale) sur la borne supérieure, on passe à la règle 3 ;
3) Décalage de tâche programmable : cette règle nous permet de sélectionner la tâche aléatoire qui ne décale pas la tâche programmable suivante en étant ordonnancée sur la machine considérée. Dans le cas où plusieurs tâches aléatoires ne décalent pas cette tâche programmable, on passe à la règle 4 ;
4) SPT : l'application de cette règle, nous permet de sélectionner la tâche aléatoire de plus courte durée opératoire (parmi les tâches qui ont déjà satisfait toutes les règles précédentes), ceci, dans l'objectif d'augmenter les chances d'insérer plus de tâches aléatoires par la suite (effectivement, notre objectif est de maximiser le nombre de tâches aléatoires à traiter. L'insertion de tâches dont la durée opératoire n'est pas importante, nous permettra par la suite d'avoir, encore, une marge assez conséquente pour l'insertion de nouvelles tâches aléatoires). Dans le cas où plusieurs tâches aléatoires ont la même durée opératoire (qui est minimale), on passe à la règle 5 ;
5) EDD : on sélectionne parmi les tâches aléatoires dont la durée opératoire est la même (et qui est la durée opératoire minimale), celle qui a la date échue la plus proche afin de ne pas diminuer ses chances pour une éventuelle exécution. Dans le cas où plusieurs tâches ont la même date échue, on utilisera la règle FIFO.

5. Modélisation de l'approche proposée sous ARENA :

Globalement, la configuration de base de notre atelier, reste la même que celle présentée pour le modèle au plus tard à une seule file d'attente pour les tâches aléatoires dans le chapitre II (autrement dit, même l'ordonnancement des tâches programmables reste le même). La différence est que notre modèle peut être généralisé à plusieurs machines en parallèle sans grand changement dans la configuration de base, comme on le verra dans ce qui suit.

La figure III.26 représente la configuration de base de l'atelier étudié dans le cas de trois machines en parallèle.

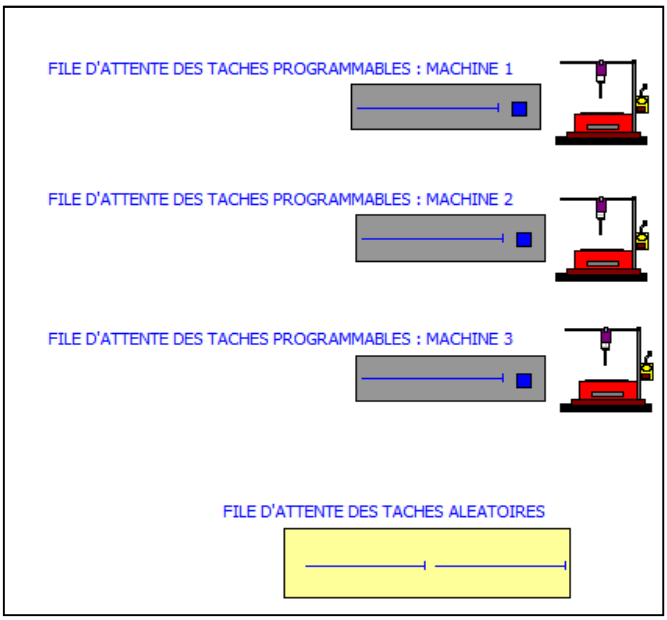


Figure III.26 : configuration de base de l'atelier étudié dans le cas de 3 machines en parallèle

Le passage d'une configuration à deux machines à une configuration à trois machines voir plus se fait grâce à la boite de dialogue (figure III.27) qui offre à l'utilisateur du modèle la possibilité d'indiquer le nombre de machine composant la configuration de son atelier.

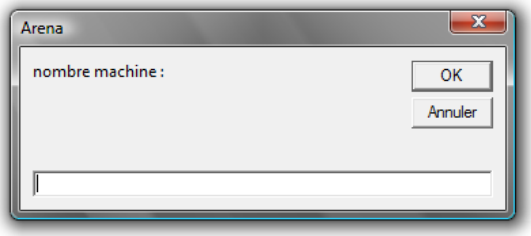


Figure III.27 : boite de dialogue pour l'indication du nombre de machine

Ainsi, pour la partie tâche programmable et la partie traitement des tâches, décrites en détail dans la deuxième section de ce chapitre, pour le modèle au plus tard à une seule file d'attente pour les tâches aléatoires aucun changement n'a été apporté (sauf pour ce qui est du nombre de séquences après le Branch dans la partie tâche programmable et la partie traitement des tâches, qui, dans le modèle précédent étaient de deux et désormais, selon l'attente du concepteur, ils peuvent être de trois, quatre, voir plus).

Le changement important apporté se situe dans la partie tâche aléatoire, plus précisément, dans les deux blocs VBA présents au niveau de la partie vérification des tâches aléatoires (figure III.28), dont la fonction est de sélectionner les tâches aléatoires à exécuter sur les machines libres.

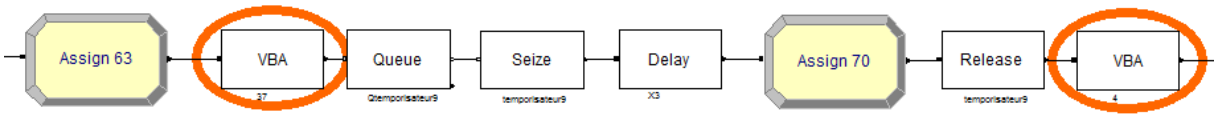


Figure III.28 : la branche du modèle concernée par le changement

- Rôle du premier bloc VBA :

Le premier bloc VBA a pour fonction le remplissage d'une matrice "M" (figure III.29) avec un certain nombre d'informations sur les tâches aléatoires, nécessaires lors de la collaboration, ces informations sont les suivantes.

Comme informations communes à toutes les machines, la durée opératoire de la tâche aléatoire et sa date échue (ces données sont stockées dans les 2 premières lignes de la matrice "M").

Pour ce qui est de la partie relative à chaque machine (3 lignes consécutives pour chaque machine), on a, l'impact sur la borne supérieure dans le cas de l'exécution de la tâche aléatoire sur la machine concernée. Puis, une ligne est consacrée à la priorité de la tâche concernée quant à son exécution sur cette machine (car, chaque machine libre, classe les tâches aléatoires présentes dans la file d'attente selon un ordre de priorité tel que :

- Les tâches qui peuvent être exécutées sans augmenter la borne supérieure, ni décaler une tâche programmable représentent les tâches de priorité 1.
- Les tâches qui peuvent être exécutées sans augmenter la borne supérieure mais décalent les tâches programmables représentent les tâches de priorité 2.
- Les tâches qui peuvent être exécutées mais augmentent la borne supérieure, font partie des tâches de priorité 3.
- Pour les tâches qui ne peuvent être exécutées, on leur affecte dans cette ligne la valeur 0).

La troisième ligne de la partie relative à chaque machine est réservée pour le décalage induit par l'éventuelle exécution de cette tâche sur la machine concernée.

Pour chaque machine une liste de tâches aléatoires candidates est dressée. Ces tâches sont triées dans la matrice "ma" (figure III.30) selon leurs priorités.

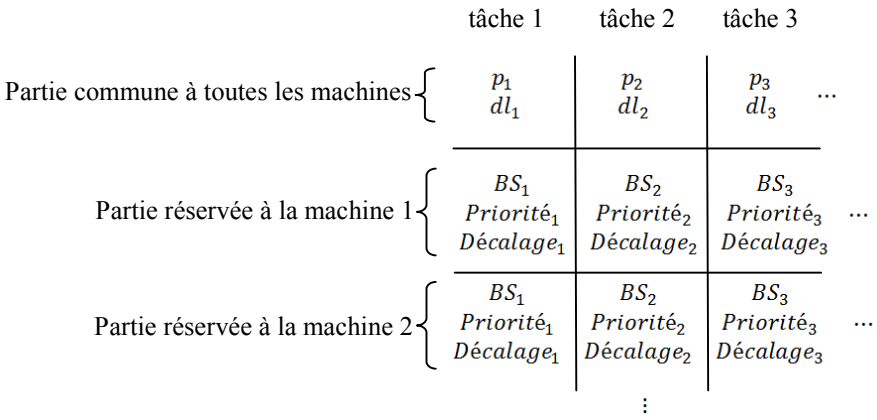


Figure III.29 : la forme de la matrice M

Les tâches aléatoires de la matrice "ma" sont triées selon leur ordre dans la file d'attente.

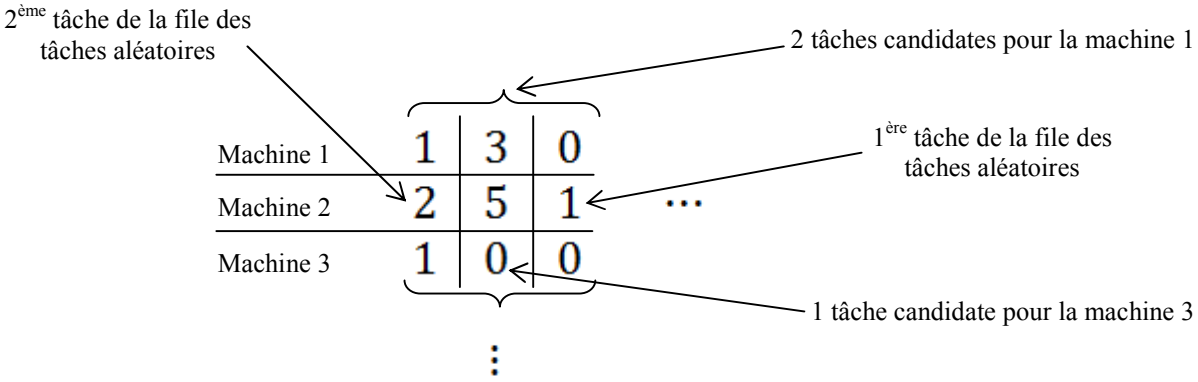


Figure III.30 : la forme de la matrice de négociation

- Rôle du deuxième bloc VBA :

Le deuxième bloc VBA a pour fonction, la mise en œuvre de la collaboration entre les machines.

Ce bloc est structuré d'une façon telle que chaque machine libre commence par remplir sa ligne de tâches aléatoires candidates et ceci dans la matrice "ma". Le remplissage s'effectue depuis la matrice "M". Ensuite, chaque machine trie ses tâches selon leurs priorités. Puis chaque machine calcule le nombre de tâches aléatoires candidates pour une éventuelle exécution sur cette dernière. Au bout de cette étape, on aura pour chaque machine une tâche aléatoire optimale à exécuter sur cette dernière selon des critères locaux (relatives à la machine).

Ces informations sont, par la suite, transmises à la dernière machine (agent décideur) qui testera, la combinaison de tâches obtenues pour l'exécution sur chaque machine libre, si cette dernière est compatible avec l'objectif global. Si c'est le cas, la machine initiatrice de la collaboration (agent décideur) confirmera pour chaque autre machine l'exécution de la tâche qu'elle a sélectionnée. Dans le cas contraire, il sera demandé aux différentes machines, une à une, de proposer une autre tâche aléatoire pour exécution (qui est la prochaine tâche aléatoire candidate sur la liste). La machine initiatrice aura à confirmer la meilleure combinaison.

Au bout de ce processus, on obtient une combinaison, qui, si elle n'est pas optimale, tend très sensiblement vers cette dernière, grâce aux différents tests auxquels la machine initiatrice fait subir à la combinaison obtenue.

Les codes VBA sont présentés en annexe IV.

V. Vérification et validation des modèles obtenus :

La vérification, c'est avoir le modèle *« bon »*, par conséquent, fiable dans ses données (par exemple : 1+1=2 !).

La validation du modèle, c'est avoir *« LE bon »* modèle, reflétant au mieux la réalité.

On a soumis les modèles obtenus à une série de tests, qui nous ont permis de vérifier leur bon fonctionnement et de les valider par la suite.

Pour la vérification, on a introduit des données en entrée dont les résultats en sortie sont connus, puis on a procédé à une vérification pas à pas grâce à un affichage des résultats obtenus à chaque étape, ce qui nous a permis de vérifier la cohérence de ces derniers mais aussi de confirmer que chaque bloc, variable,… remplissait bien sa fonction.

Pour la validation, on a usé des différentes techniques, dont l'application aux modèles obtenus est possible :

- ✓ Des techniques d'animation du modèle montrant les entités en déplacement conformément aux règles de fonctionnement : ainsi, grâce à l'animation mise en place, on a pu confirmer que la circulation des entités est cohérente avec les règles mises en place ;
- ✓ Des comparaisons avec les résultats obtenus dans [Bou 07] : ainsi, en appliquant les modèles obtenus aux exemples présentés dans [Bou 07], les résultats sont les mêmes ;
- ✓ Des tests avec un nombre réduit d'entités : ainsi, on peut prévoir le résultat, qui, pour chaque modèle est confirmé après application.

VI. Conclusion :

Les différents concepts de base exposés dans les deux chapitres précédents, nous ont permis de développer des modèles sous ARENA fidèles à la réalité. Ainsi, dans ce chapitre, nous avons présenté, étape par étape, la structure des différents modèles ARENA, ce qui permet de les appréhender.

Ces modèles seront utilisés pour la simulation dans le chapitre suivant. Cette simulation servira à répondre aux objectifs de notre travail.

CHAPITRE IV : Simulation et comparaison entre les différentes approches de résolution

I. Introduction

II. Définition du plan d'expérimentation

1. Objectifs de la simulation
2. Configuration de base de l'atelier
3. Plan d'expérience
4. Indicateurs de performances

III. Simulations et enregistrement des résultats

1. Paramètres retenus
2. Résultats des expériences menées

A. Algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente

B. Algorithme d'ordonnancement temps-réel au plus tôt dans le cas d'une file d'attente

C. Algorithme d'ordonnancement temps-réel au plus tard dans le cas de deux files d'attente

D. Algorithme d'ordonnancement par approche collaborative

IV. Evaluation des performances et interprétation des résultats

V. Validation des résultats et recommandations

VI. Conclusion

I. Introduction :

Nous avons commencé ce travail par présenter et décrire les différentes approches de résolution proposées dans [Bou 07], puis, nous sommes passés à la modélisation de ces dernières sous ARENA, dans le but de pouvoir procéder à la simulation. Dans le chapitre III, nous avons proposé une approche de résolution collaborative que nous avons modélisée, aussi, sous ARENA.

Ce présent chapitre a pour objectif, en utilisant la simulation, la validation des résultats obtenus dans [Bou 07] dans un premier temps ; dans un second temps, la comparaison entre la performance de l'approche collaborative proposée dans le cadre de ce travail et les autres approches.

Pour cela, ce chapitre se décompose comme suit : dans la première section, nous définirons les expériences à réaliser. Dans la deuxième section, nous effectuerons, à titre illustratif, une centaine d'expériences afin d'avoir une première tendance sur la performance des différentes approches de résolution, nous enregistrerons, ainsi, les résultats des différentes expériences. La troisième section sera consacrée à l'évaluation des performances et l'interprétation des résultats. La quatrième section concernera la validation des résultats relatifs à la performance des différentes approches déduites dans la troisième section, ceci, grâce à la variation de certains paramètres et la comparaison entre les résultats obtenus pour chaque approche, et la proposition de recommandations fondées sur les résultats obtenus.

II. Définition du plan d'expérimentation :

On définit les conditions d'utilisation du modèle, la stratégie (nombre d'expériences, durée, plans d'expérience, ...) et la tactique (conditions initiales, types de résultats, ...).

La simulation exige au préalable :

- ✓ La détermination de la configuration de base de l'atelier, à savoir, le nombre de machines, de files d'attente,… ;
- ✓ L'identification des paramètres physiques et des différents modèles développés pour l'expérimentation ;
- ✓ La présentation des indicateurs de performance du système.

1. Objectifs de la simulation :

Les objectifs principaux à atteindre par la simulation dans le cadre de ce travail sont :

- ➢ La validation des résultats obtenus dans [Bou 07] ;
- ➢ Evaluer la performance de l'approche collaborative développée dans le cadre de ce travail.

2. Configuration de base de l'atelier :

L'atelier sujet à la simulation est constitué de :

- Deux types de tâches : programmables et aléatoires.
- Deux machines identiques en parallèles.
- Trois files d'attente. Une file d'attente pour les tâches aléatoires et une file d'attente des tâches programmables pour chaque machine.

3. Plan d'expérience :

La simulation est l'expérimentation sur un modèle. C'est une procédure de recherche scientifique qui consiste à :

- Réaliser une reproduction artificielle (modèle) du phénomène que l'on désire étudier,
- Observer le comportement de cette reproduction lorsque l'on fait varier expérimentalement les actions que l'on peut exercer sur celle-ci,
- En induire ce qui se passerait dans la réalité sous l'influence d'actions analogues.

Vis-à-vis des objectifs attendus de ce travail, on doit avoir un nombre suffisant d'expériences et des valeurs aléatoires générées pour chaque expérimentation, tel que :

- Le nombre de tâches programmables est de 75, celui des tâches aléatoires est de 60 (ce choix a été fait dans le but d'avoir des tâches aléatoires qui arrivent tout au long de l'exécution des tâches programmables et pas au delà. Effectivement, ne pas avoir de nouvelles tâches aléatoires qui arrivent alors que les tâches programmables n'ont pas encore été exécutées totalement, ou, avoir des tâches aléatoires qui arrivent après l'exécution de toutes les tâches programmables n'a pas d'intérêt pour notre étude. C'est pourquoi, nous avons procédé à des tests avec les modèles de simulation afin de dimensionner le nombre de tâches qui répond à nos attentes) ;
- Les différents paramètres de ce problème (r, p, d) seront générés selon les distributions suivantes :
  - ✓ Pour les tâches programmables : l'arrivée des tâches est poissonnienne, leurs durées opératoires suivent une loi triangulaire et leurs dates échues sont données par la somme de la date de disponibilité, la durée opératoire et une variable aléatoire triangulaire ;
  - ✓ Pour les tâches aléatoires : même chose que pour les tâches programmables (l'arrivée des tâches est poissonnienne, leurs durées opératoires suivent une loi triangulaire et leurs dates échues sont données par la somme de la date de disponibilité, la durée opératoire et une variable aléatoire triangulaire) ;
- La même série de données aléatoires sera utilisée pour chaque modèle afin de garantir le résultat de la comparaison entre ces derniers ;
- Les modèles testés sont au nombre de quatre : l'algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente, l'algorithme d'ordonnancement temps-réel au plus tôt dans le cas d'une file d'attente, l'algorithme d'ordonnancement temps-réel au plus tard dans le cas de deux files d'attente (ces trois modèles, sont ceux développés dans [Bou 07]) et l'approche collaborative développée dans le cadre de ce travail.

4. Indicateurs de performances :

Les indicateurs de performance retenus dans le cadre de ce travail sont :

- Le Makespan (C_{max}) : on prendra comme référence pour le C_{max} obtenu à chaque expérience, celui obtenu dans le cas où l'heuristique LPT est utilisée (sachant que

l'application de cette heuristique, implique la relaxation d'un certain nombre d'hypothèses telles que les dates d'arrivées des tâches, qui se fera dans ce cas à l'instant zéro, le respect des dates échues n'est plus considéré… la performance de l'application de cette règle a été évaluée par Graham dans [Gra 69] comme étant égale à :

$$\frac{C_{max}(LPT)}{C^*_{max}} \leq \frac{4}{3} - \frac{1}{3m}$$

(Avec C^*_{max} la valeur de la solution optimale, $C_{max}(LPT)$ la solution obtenue avec l'heuristique *LPT* et *m* le nombre de machines.)

Ainsi, c'est une bonne référence, pour l'évaluation de la performance des différentes approches expérimentées) ;

- Le nombre de tâches aléatoires traitées (exécutées).

III. Simulations et enregistrement des résultats :

On effectue, à titre illustratif, une centaine d'expériences, afin d'avoir une première tendance sur la performance des différentes approches de résolution, (on a choisi une centaine d'expériences pour pouvoir observer le comportement des modèles développés pour un grand nombre d'expériences. Ceci étant, un nombre de 40 expériences, comme ça sera le cas dans la dernière section, est suffisant, puisque, dépasser les 30 expériences, on peut appliquer le théorème central limite qui nous permet d'approximer toute loi par une loi gaussienne). Après chaque simulation, les résultats sont enregistrés avant leur analyse dans la prochaine section.

1. Paramètres retenus :

Les paramètres retenus pour l'expérimentation, ainsi que les flots utilisés (nouveau code de génération de nombres aléatoires ARENA 10.0), sont :

- Pour les tâches programmables :
  - ✓ Les arrivées : $r_i = r_{i-1} + EXPO(2{,}8)$
  - ✓ Les durées opératoires : $p_i = TRIA(1{,}2{,}4{,}2)$
  - ✓ Les dates échues : $d_i = r_i + p_i + TRIA(1{,}11{,}37{,}9)$

- Pour les tâches aléatoires :
  - ✓ Les arrivées : $r_j = r_{j-1} + EXPO(2{,}4)$
  - ✓ Les durées opératoires : $p_j = TRIA(1{,}2{,}5{,}6)$
  - ✓ Les dates échues : $d_j = r_j + p_j + TRIA(1{,}3{,}6{,}7)$

Il faut noter que le dernier paramètre représente le flot.

2. Résultats des expériences menées :

On résume dans ce qui suit les résultats obtenus à l'issu de la simulation pour chacun des quatre modèles développés.

A. Algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente :

Conformément au plan d'expérience présenté plus haut, le nombre d'expériences réalisées est de 100, les résultats obtenus se présentent comme suit :

i. Le C_{max} :

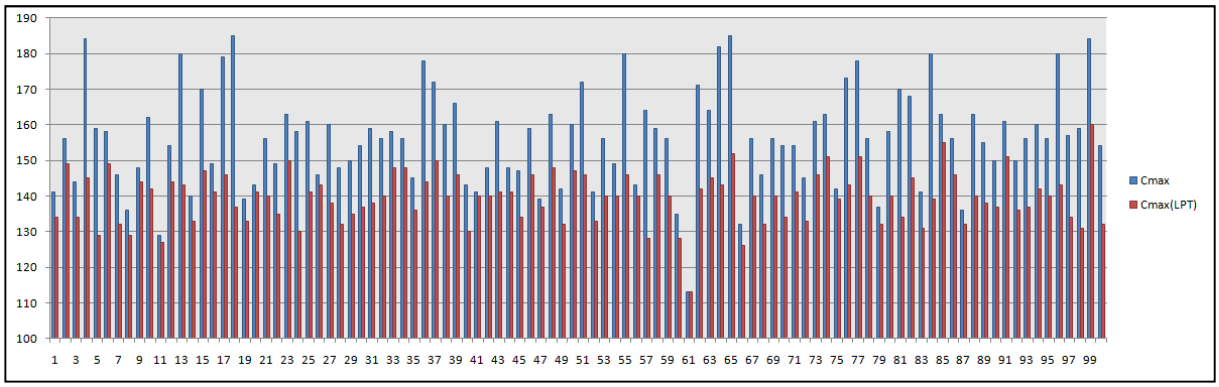


Figure IV.1 : Observations du C_{max} obtenu et du C_{max} selon la règle LPT

Le C_{max} obtenu avec l'application de l'algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente est, pour chaque expérimentation, supérieur à celui obtenu en ordonnançant les tâches exécutées selon la règle LPT, sauf pour la 61ème expérience, où, on a obtenu le même C_{max}.

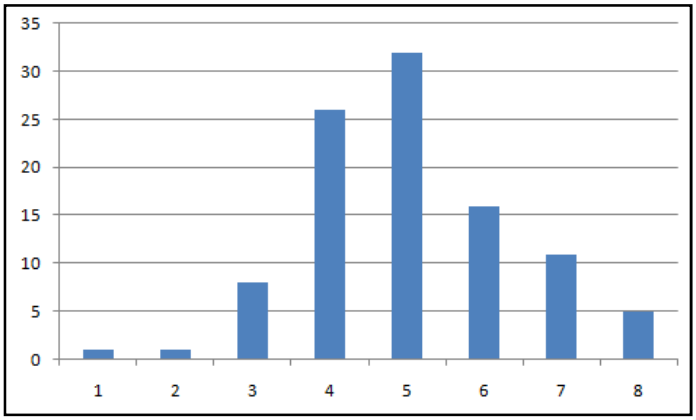


Figure IV.2 : Distribution des C_{max}

En regroupant les C_{max} par classe (figure IV.2) telle que les classes sont :
1 : [110, 120] ; 2 :]120, 130] ; 3 :]130, 140] ; 4 :]140, 150] ; 5 :]150, 160] ; 6 :]160, 170] ; 7 :]170, 180] ; 8 :]180, 190].

On obtient une distribution Gaussienne.

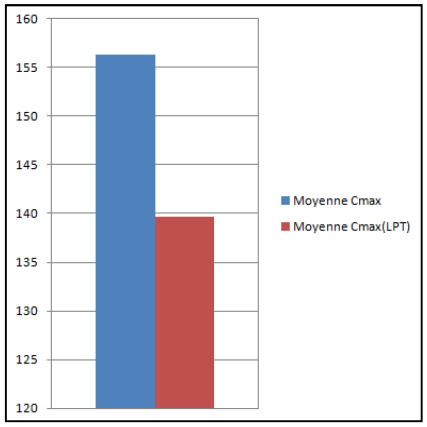


Figure IV. 3 : Moyenne du C_{max} obtenu et du C_{max} selon la règle LPT

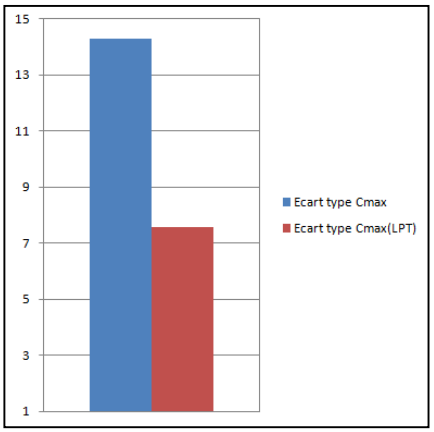


Figure IV.4 : Ecart type du Cmax obtenu et du Cmax selon la règle LPT

Dans la figure IV.3, on remarque que le C_{max} moyen obtenu selon la règle LPT est inférieur. Connaissant la performance de la règle LPT (**Chap. IV** : 2.4), on compare la moyenne du C_{max} obtenu par l'application de l'algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente à celui obtenu selon la règle LPT. Ce qui nous donne un taux de :

$$\tau_{tard1} = 11{,}91\%$$

On remarque, dans la figure IV.4, que l'écart type de la règle LPT est deux fois moins important que l'autre. De là, et sachant que cette règle est considérée comme référence, l'algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente donne des résultats deux fois plus dispersés autour de la moyenne que la référence.

ii. Le nombre de tâches aléatoires traitées :

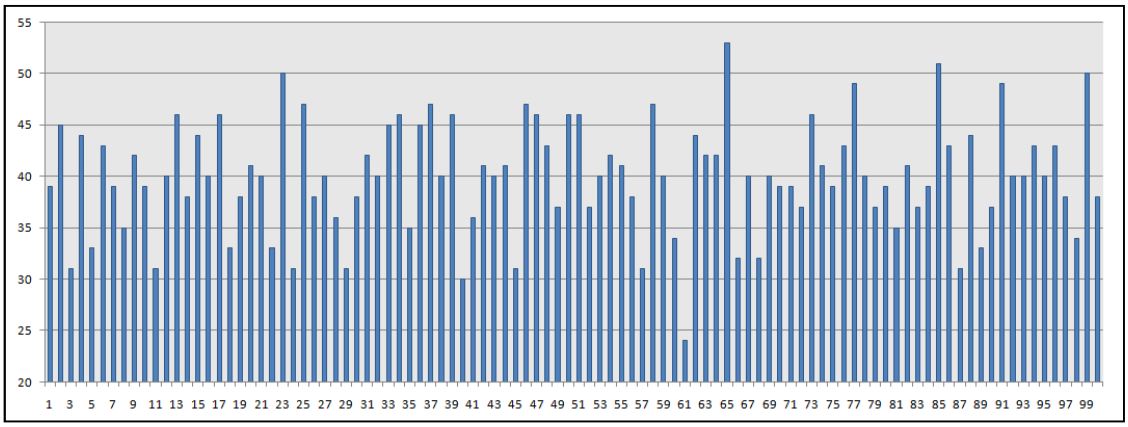


Figure IV.5 : Observations du nombre de tâches aléatoires traitées

La figure IV.5 représente le nombre de tâches aléatoires exécutées lors de chaque expérience. Le nombre maximum de tâches aléatoires exécutées a été obtenu lors de la 65ème expérience avec 53 tâches exécutées, quant au minimum, il est de 24, il a été obtenu lors de la 61ème expérience. La moyenne de ce nombre est de : 39,95 tâches aléatoires. Pour un écart type de : 5,18 tâches aléatoires.

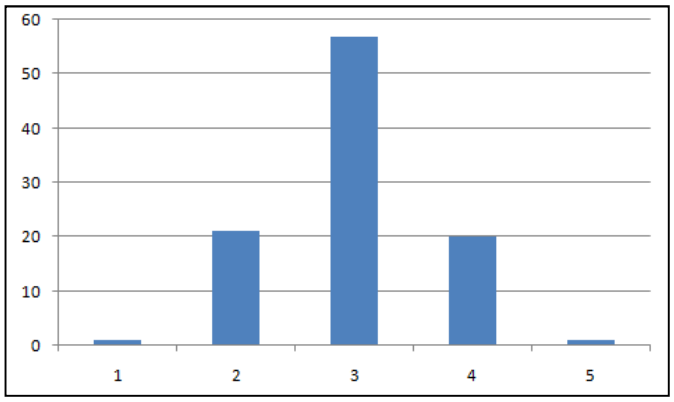


Figure IV.6 : Distribution des nombres de tâches aléatoires exécutées

En regroupant le nombre de tâches aléatoire exécutées par classe (figure IV.6) telle que les classes sont :
1 : [20, 28] ; 2 : [29, 36] ; 3 :[37, 44] ; 4 : [45, 52] ; 5 : [53, 60].

On obtient une distribution Gaussienne.

B. Algorithme d'ordonnancement temps-réel au plus tôt dans le cas d'une file d'attente :

Conformément au plan d'expérience présenté plus haut, le nombre d'expériences réalisées est de 100, les résultats obtenus se présentent comme suit :

i. Le C_{max} :

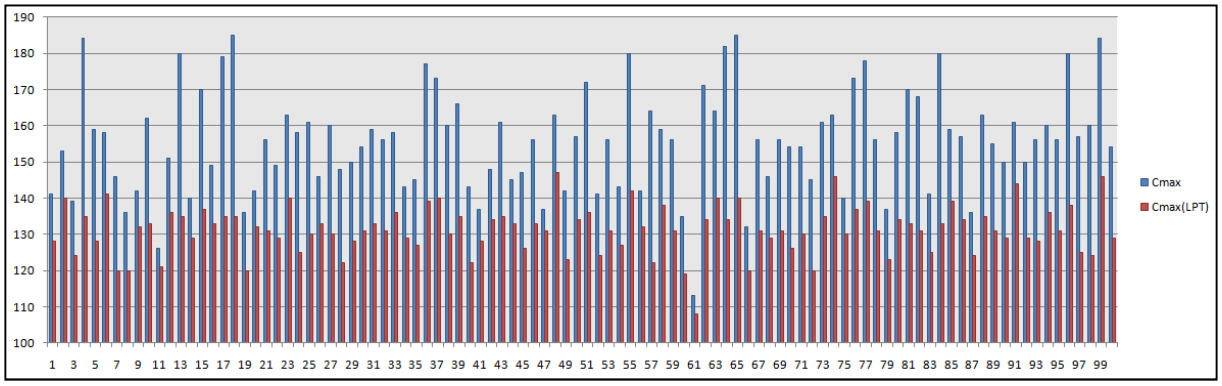


Figure IV.7 : Observations du Cmax obtenu et du Cmax selon la règle LPT

Le C_{max} obtenu avec l'application de l'algorithme d'ordonnancement temps-réel au plus tôt dans le cas d'une file d'attente est, pour chaque expérimentation, supérieur à celui obtenu en ordonnançant les tâches exécutées selon la règle LPT, sans exception.

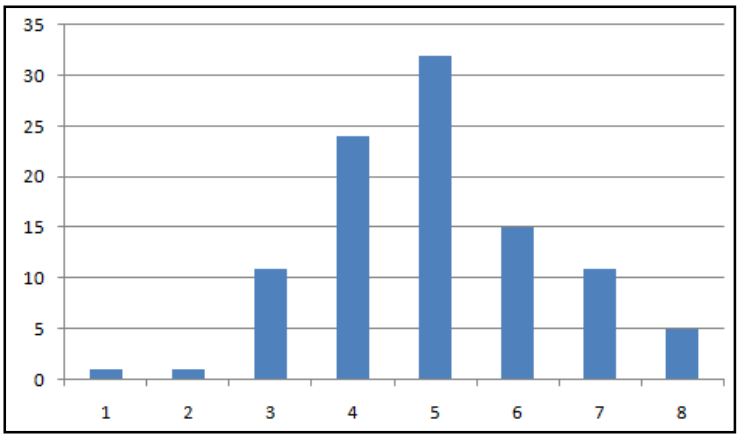


Figure IV.8 : Distribution des C_{max}

En regroupant les C_{max} par classe (figure IV.8) telle que les classes sont :
1 : [110, 120] ; 2 :]120, 130] ; 3 :]130, 140] ; 4 :]140, 150] ; 5 :]150, 160] ; 6 :]160, 170] ; 7 :]170, 180] ; 8 :]180, 190].

On obtient la même distribution que dans le cas précédent. Il s'agit du même phénomène, alors, on peut comparer les résultats des deux modèles en termes de C_{max}.

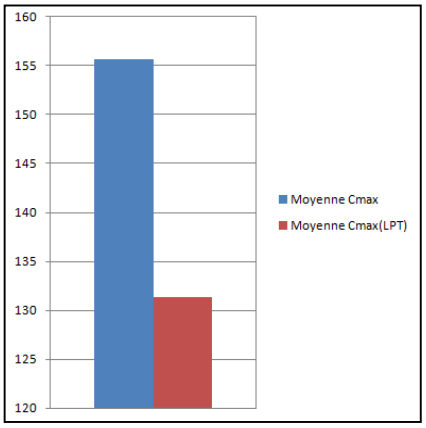


Figure IV.9 : Moyenne du Cmax obtenu et du Cmax selon la règle LPT

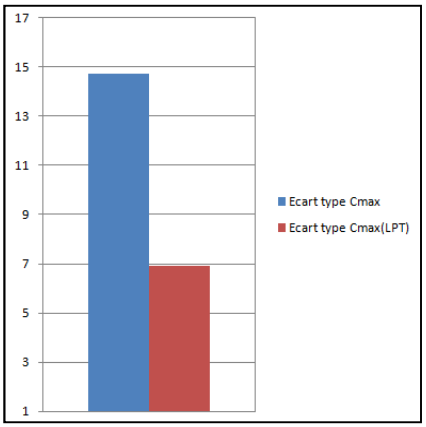


Figure IV.10 : Ecart type du Cmax obtenu et du Cmax selon la règle LPT

Dans la figure IV.9, on remarque que le C_{max} moyen obtenu selon la règle LPT est inférieur. On compare la moyenne du C_{max} obtenu par l'application de l'algorithme d'ordonnancement temps-réel au plus tôt dans le cas d'une file d'attente à celui obtenu selon la règle LPT grâce au taux suivant :

$$\tau_{tôt} = 18{,}52\%$$

On remarque, dans la figure IV.10, que l'écart type de l'algorithme d'ordonnancement temps-réel au plus tôt dans le cas d'une file d'attente est plus, de deux fois plus important que l'autre.

ii. Le nombre de tâches aléatoires traitées :

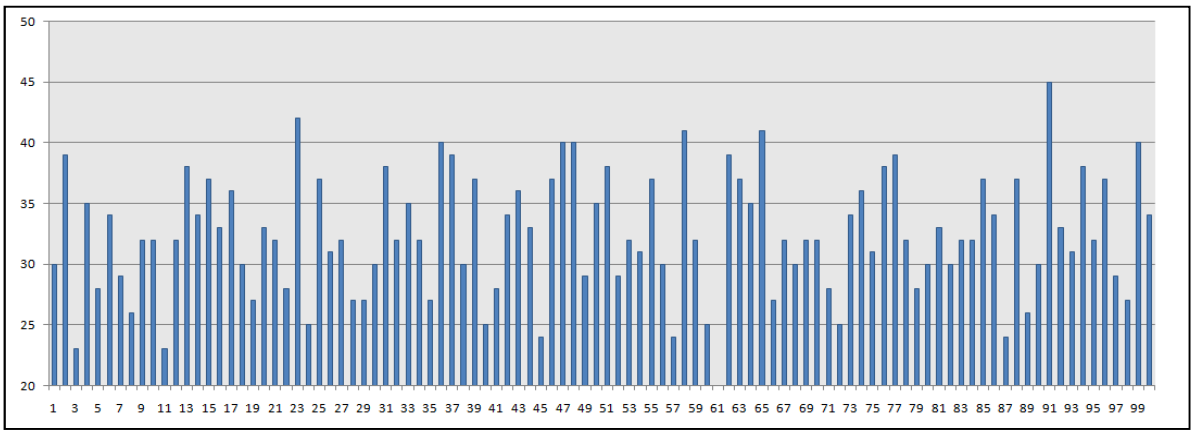


Figure IV.11 : Observations du nombre de tâches aléatoires traitées

La figure IV.11 représente le nombre de tâches aléatoires exécutées lors de chaque expérience. Le nombre maximum de tâches aléatoires exécutées a été obtenu lors de la 91ème expérience avec 45 tâches exécutées, quant au minimum, il est de 19, il a été obtenu lors de la 61ème expérience. La moyenne de ce nombre est de : 32,41 tâches aléatoires. Pour un écart type de : 5,19 tâches aléatoires.

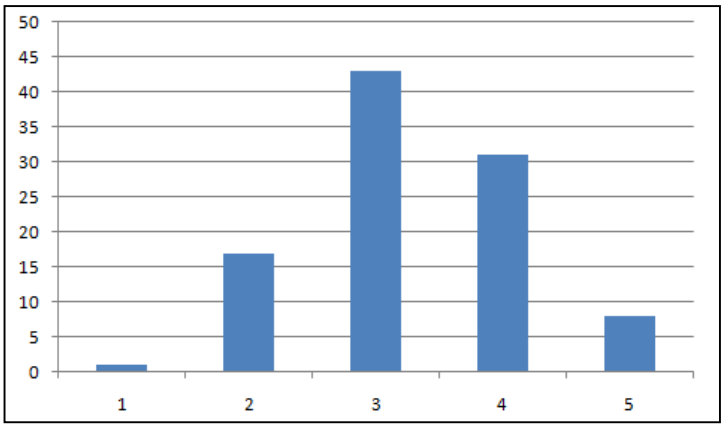


Figure IV.12 : Distribution des nombres de tâches aléatoires exécutées

En regroupant le nombre de tâches aléatoire exécutées par classe (figure IV.12) telle que les classes sont :
1 : [15, 21] ; 2 : [22, 27] ; 3 : [28, 33] ; 4 : [34, 39] ; 5 : [40, 45].

On obtient la même distribution que dans le cas précédent. Il s'agit du même phénomène, ainsi, on peut comparer les résultats des deux modèles en termes de nombres de tâches aléatoires exécutées.

C. Algorithme d'ordonnancement temps-réel au plus tard dans le cas de deux files d'attente :

Conformément au plan d'expérience présenté plus haut, le nombre d'expériences réalisées est de 100, les résultats obtenus se présentent comme suit :

i. Le C_{max} :

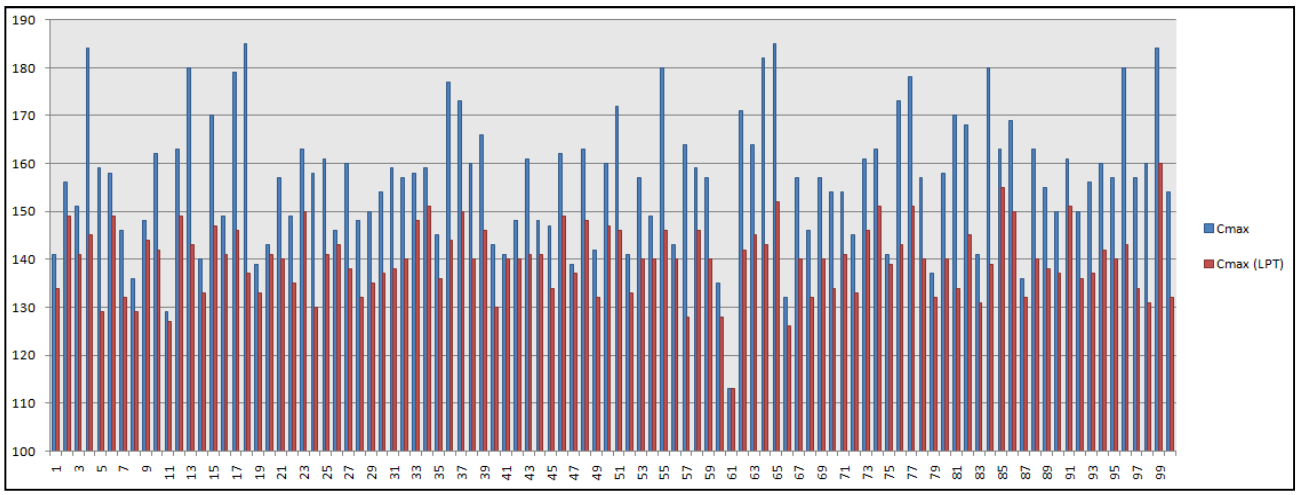


Figure IV. 13: Observations du Cmax obtenu et du Cmax selon la règle LPT

Le C_{max} obtenu avec l'application de l'algorithme d'ordonnancement temps-réel au plus tôt dans le cas d'une file d'attente est, pour chaque expérimentation, supérieur à celui obtenu en ordonnançant les tâches exécutées selon la règle LPT, sauf pour la 64ème expérience, où, on a obtenu le même C_{max}.

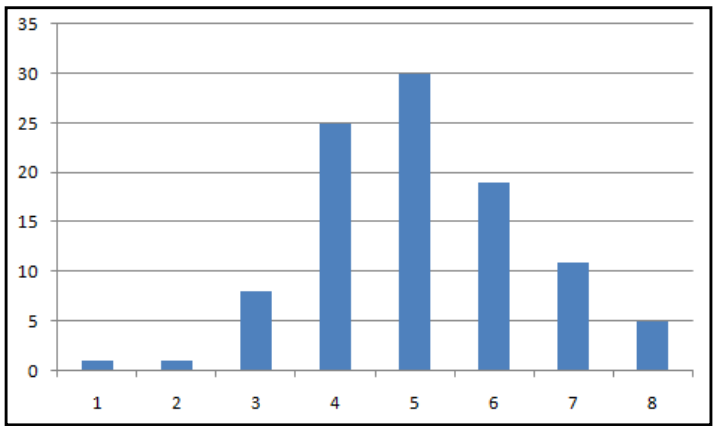


Figure IV.14 : Distribution des C_{max}

En regroupant les C_{max} par classe (figure IV.14) telle que les classes sont :
1 : [110, 120] ; 2 :]120, 130] ; 3 :]130, 140] ; 4 :]140, 150] ; 5 :]150, 160] ; 6 :]160, 170] ; 7 :]170, 180] ; 8 :]180, 190].

On obtient la même distribution que dans les deux cas précédents. Il s'agit du même phénomène, ainsi, on peut comparer les résultats des trois modèles en termes de C_{max}.

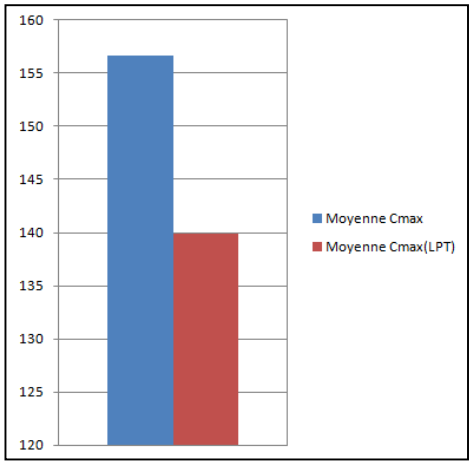


Figure IV.15 : Moyenne du Cmax et du Cmax selon la règle LPT

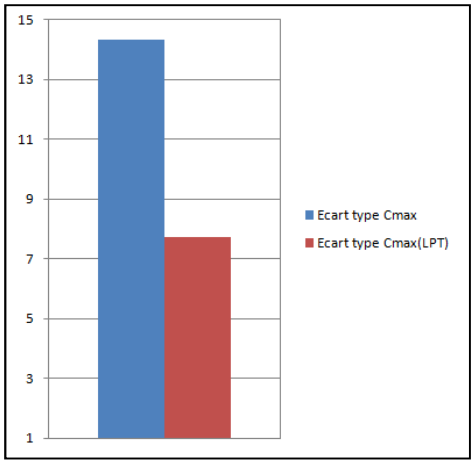


Figure IV.16 : Ecart type du Cmax et du Cmax selon la règle LPT

Dans la figure IV.15, on remarque que le C_{max} moyen obtenu selon la règle LPT est inférieur. On compare la moyenne du C_{max} obtenu par l'application de l'algorithme d'ordonnancement temps-réel au plus tard dans le cas de deux files d'attente à celui obtenu selon la règle LPT. Ceci nous donne un taux de :

$$\tau_{tard2} = 11{,}99\%$$

L'écart type de la règle LPT est, encore une fois, deux fois moins important que l'autre.

ii. Le nombre de tâches aléatoires traitées :

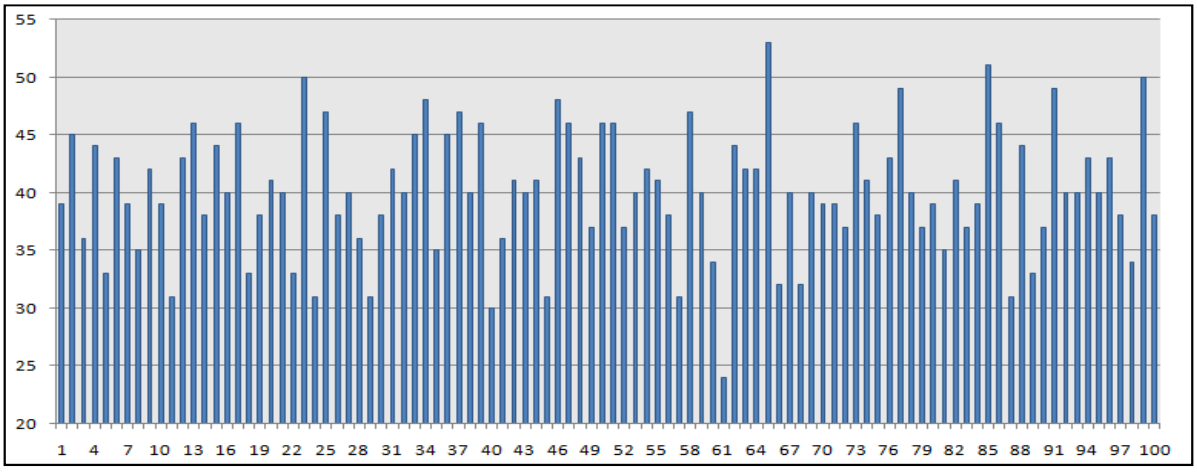


Figure IV.17 : Observations du nombre de tâches aléatoires traitées

La figure IV.17 représente le nombre de tâches aléatoires exécutées lors de chaque expérience. Le nombre maximum de tâches aléatoires exécutées a été obtenu lors de la 65ème expérience avec 53 tâches exécutées, quant au minimum, il est de 24, il a été obtenu lors de la 61ème expérience ; nous remarquons que les valeurs extrêmes, sont les mêmes que ceux obtenues par l'algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente. La moyenne du nombre de tâches aléatoires exécutées est de : 40,09 tâches aléatoires. Pour un écart type de : 5,60 tâches aléatoires.

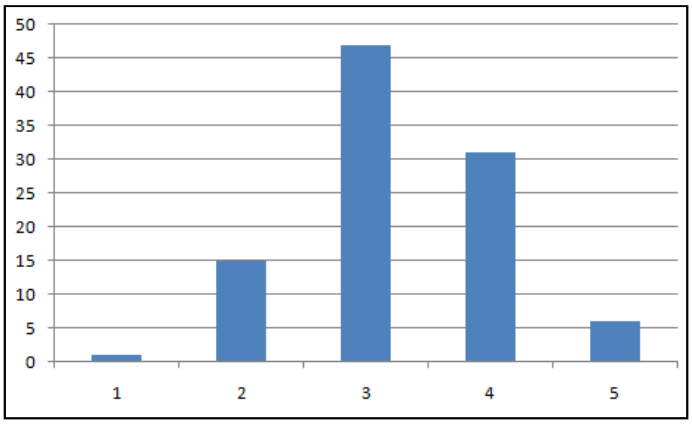


Figure IV.18 : Distribution des nombres de tâches aléatoires exécutées

En regroupant le nombre de tâches aléatoire exécutées par classe (figure IV.18) telle que les classes sont :
1 : [20, 27] ; 2 : [28, 34] ; 3 :[35, 41] ; 4[42, 48] ; 5 :[49, 55].

On obtient la même distribution que dans les deux cas précédents. Il s'agit du même phénomène. Il s'agit du même phénomène, ainsi, on peut comparer les résultats des trois modèles en termes de nombres de tâches aléatoires exécutées.

D. Algorithme d'ordonnancement par approche collaborative :

Conformément au plan d'expérience présenté plus haut, le nombre d'expériences réalisées est de 100, les résultats obtenus se présentent comme suit :

i. Le C_{max} :

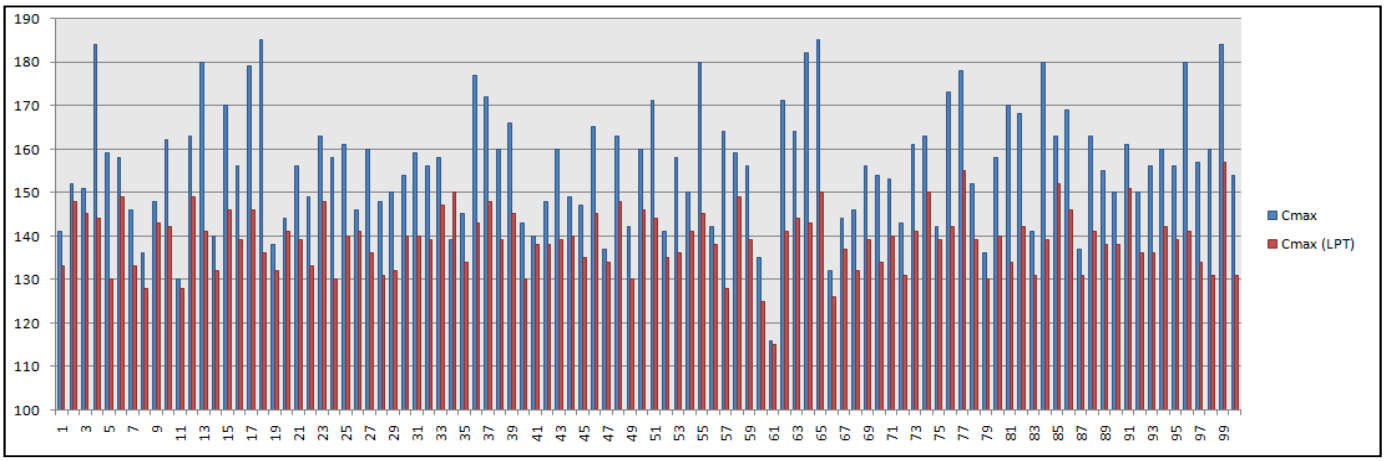


Figure IV.19 : Observations du Cmax obtenu et du Cmax selon la règle LPT

Le C_{max} obtenu avec l'application de l'approche collaborative est, pour chaque expérimentation, supérieur à celui obtenu en ordonnançant les tâches exécutées selon la règle LPT, sauf pour la 34ème expérience, où, on a obtenu un C_{max} inférieur à celui obtenu par la règle LPT (avec 11 unités de temps en moins).

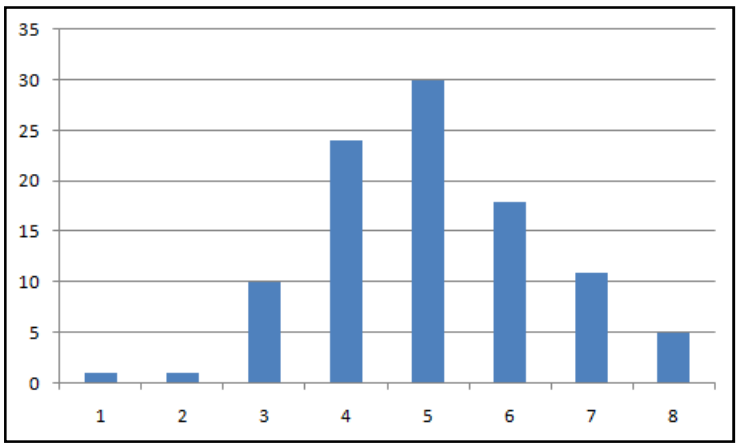


Figure IV.20 : Distribution des C_{max}

En regroupant les C_{max} par classe (figure IV.20) telle que les classes sont :
1 : [110, 120] ; 2 :]120, 130] ; 3 :]130, 140] ; 4 :]140, 150] ; 5 :]150, 160] ; 6 :]160, 170] ; 7 :]170, 180] ; 8 :]180, 190].

On obtient la même distribution que dans les trois cas précédents. Il s'agit du même phénomène, ainsi, on peut comparer les résultats des quatre modèles en termes de C_{max}.

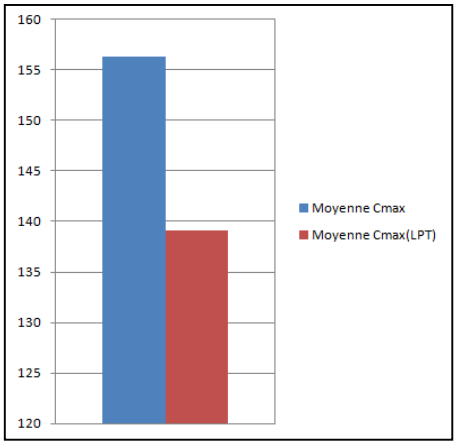


Figure IV.21 : Moyenne du Cmax et du Cmax selon la règle LPT

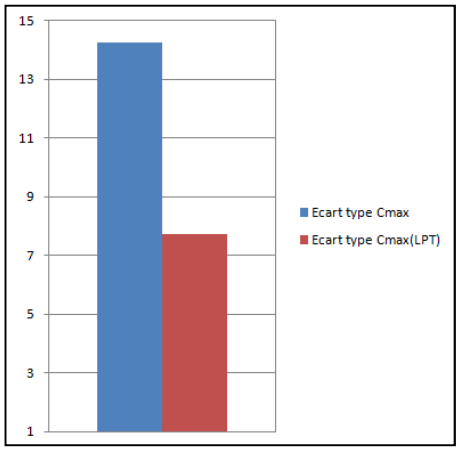


Figure IV.22 : Ecart type du Cmax et du Cmax selon la règle LPT

Dans la figure IV.21, on remarque que le C_{max} moyen obtenu selon la règle LPT est, encore une fois, inférieur. On compare la moyenne du C_{max} obtenu par l'application de l'approche collaborative à celui obtenu selon la règle LPT. Ce qui nous donne un taux de :

$$\tau_{tard2} = 12{,}37\%$$

L'écart type de la règle LPT est moins, de deux fois moins important que celui obtenu lors de l'application de l'approche collaborative (94% plus élevé).

ii. Le nombre de tâches aléatoires traitées :

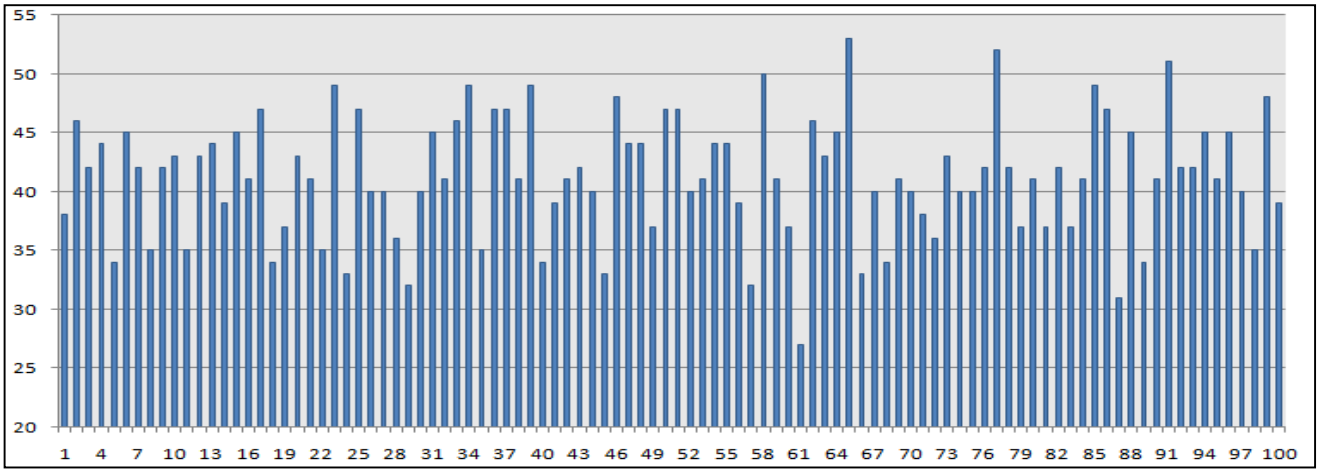


Figure IV.23 : Observations du nombre de tâches aléatoires traitées

La figure IV.23 représente le nombre de tâches aléatoires exécutées lors de chaque expérience. Le nombre maximum de tâches aléatoires exécutées a été obtenu lors de la 65ème expérience avec 53 tâches exécutées, quant au minimum, il est de 27, il a été obtenu lors de la 61ème expérience. Nous remarquons que la valeur maximale, est la même que celle obtenue par l'application de l'algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente et dans le cas de deux files d'attente, ceci étant, le nombre minimum de tâches aléatoires exécutées a été amélioré. La moyenne du nombre de tâches aléatoires exécutées est de : 41,22 tâches aléatoires. Pour un écart type de : 5,24 tâches aléatoires.

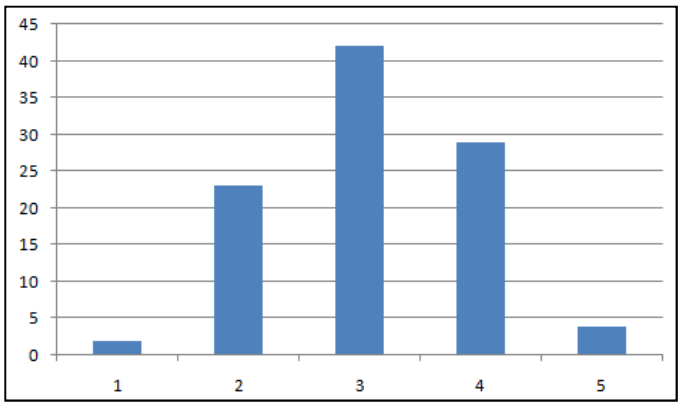


Figure IV.24 : Distribution des nombres de tâches aléatoires exécutées

En regroupant le nombre de tâches aléatoire exécutées par classe (figure IV.24) telle que les classes sont :
1 : [25, 31] ; 2 : [32, 37] ; 3 :[38, 43] ; 4[44, 49] ; 5 :[50, 55].

On obtient la même distribution que dans les trois cas précédents. Il s'agit du même phénomène. Il s'agit du même phénomène, ainsi, on peut comparer les résultats des quatre modèles en termes de nombres de tâches aléatoires exécutées.

IV. Evaluation des performances et interprétation des résultats :

L'analyse des résultats est possible à l'aide des indicateurs de performance sous forme de tableaux, courbes, graphiques, histogrammes, ... Le recours à des outils d'analyse statistique est parfois nécessaire (analyse de la variance). Cette interprétation permet soit :

- Dans le cas de résultats non satisfaisants, d'émettre des recommandations, comme la remise en question des paramètres ou le cas échéant, des objectifs [Habchi *et al.* 96]. Dans ce cas, ces recommandations (hypothèses de plan d'action) sont encore à tester et, par conséquent, à réintroduire dans le modèle de simulation. Il faut ensuite à nouveau effectuer une simulation avec ce nouveau paramétrage, et cela, jusqu'à satisfaction des résultats.
- Dans le cas de résultats satisfaisants, de mettre en œuvre les plans d'action sur le système réel.

i. Le C_{max} :

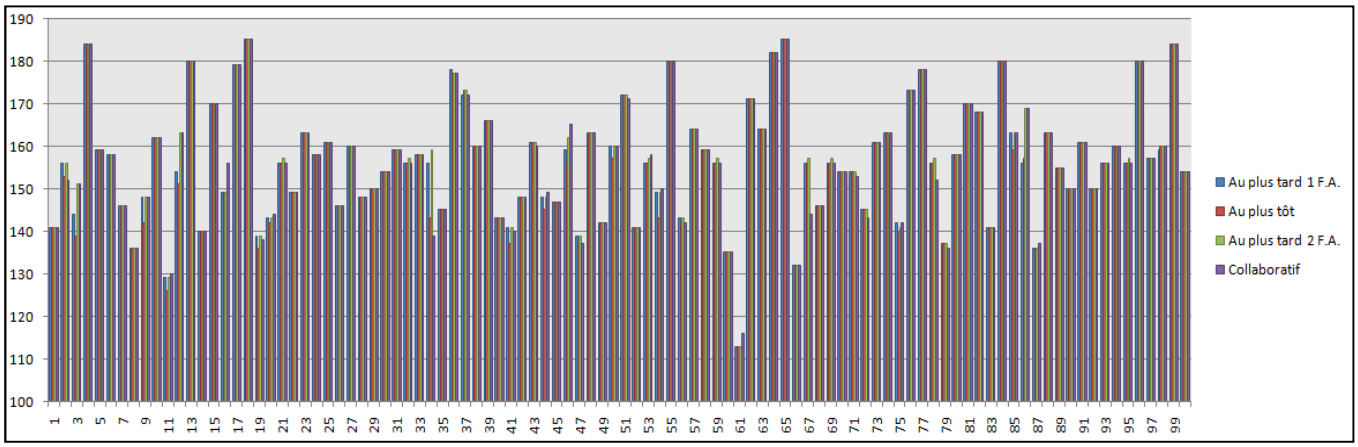


Figure IV.25 : Observations du Cmax des quatre modèles

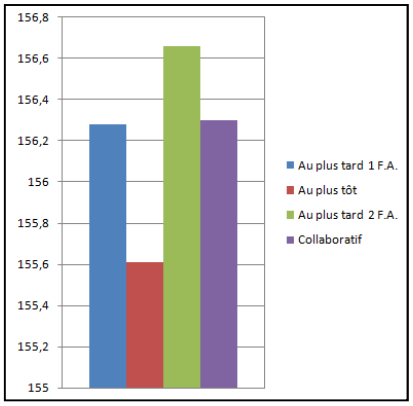


Figure IV.26 : Les moyennes des Cmax des quatre modèles

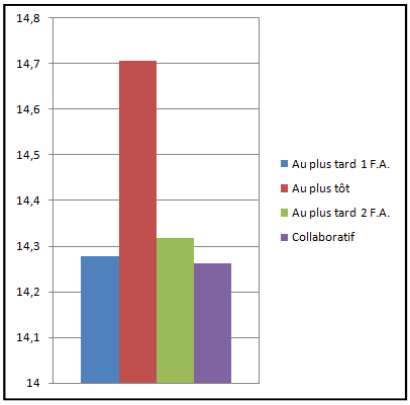


Figure IV.27 : Les écarts type des Cmax des quatre modèles

Nous remarquons que le modèle au plus tôt, favorisant l'ordonnancement des tâches programmables, présente le C_{max} le moins important. Le modèle au plus tard avec deux files d'attente, quant à lui, présente le plus grand C_{max}. Ce qui est attendu car ce dernier ne cherche pas à minimiser la borne supérieure et donne la priorité aux tâches aléatoires. En rajoutant des tâches aléatoires, le nombre de tâches à ordonnancer augmente. Ceci nécessite un autre critère

de comparaison. D'où, l'utilisation d'un taux du C_{max} rapporté à l'ordonnancement selon la règle LPT.

Le modèle optimisant le critère du C_{max} est alors, le modèle au plus tôt, avec une moyenne de **155,6.** Les autres modèles présentent des moyennes telles que : **156,3 ; 156,3** et **156,7** pour, respectivement, les modèles : au plus tard avec une file d'attente, collaboratif et au plus tard avec deux files d'attente.

Les variances des quatre modèles sont approximativement les mêmes, en plus, les sources d'aléa des différents modèles utilisent les mêmes flots de génération des nombres aléatoires, ce qui nous permet de conclure sur la cohérence des résultats.

ii. τ_{LPT} :

Afin de mieux comparer les C_{max}, en tenant compte du nombre de tâches aléatoires exécutées, nous rapportons le C_{max} à un taux, exprimant, en pourcentage, la différence entre le C_{max} de l'ordonnancement et celui obtenu avec la règle LPT.

$$\tau_{LPT} = (C_{max} - C_{max}(LPT)) \times 100 / C_{max}(LPT)$$

Ce paramètre mesure le rapprochement de l'ordonnancement avec les contraintes des dates de disponibilité et échues, de celui obtenu avec la règle LPT en relaxant ces contraintes.

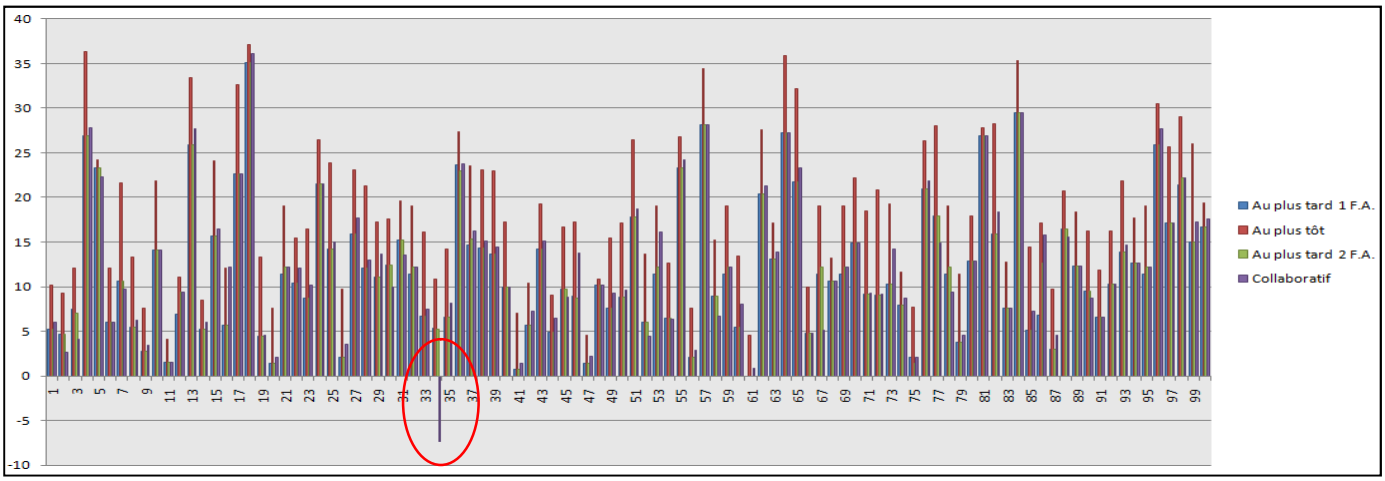


Figure IV.28 : Observations des τ_{LPT} des différents modèles

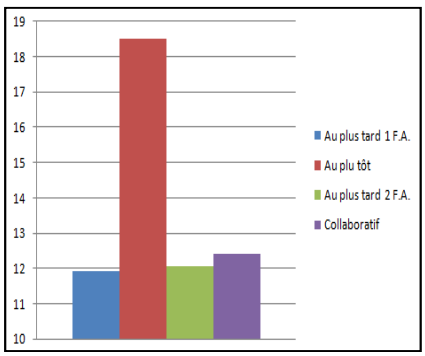


Figure IV.29 : Les moyennes des τ_{LPT} des différents modèles

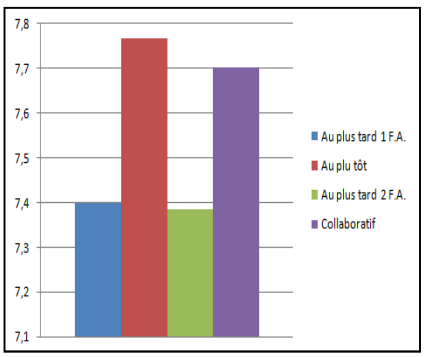


Figure IV.30 : Les écarts type des τ_{LPT} des différents modèles

Nous constatons, qu'en tenant compte du nombre de tâches exécutées, le modèle au plus tôt devient le plus médiocre. Ceci s'explique par le remplissage des vides (dus à la contrainte des dates d'apparition des tâches programmables) entre les tâches programmables par les tâches aléatoires. Et comme le modèle au plus tôt exécute moins de tâches aléatoires, alors, il y aura moins de vide rempli.

Pour ce critère, le meilleur algorithme est celui de l'ordonnancement au plus tard avec une file d'attente, avec une moyenne de : **11,9112832**. Les moyennes des autres modèles sont, respectivement : **12,0** ; **12,4** et **18,5** pour les modèles : au plus tard avec deux file d'attente, collaboratif et au plus tôt.

La valeur négative encerclée en rouge sur la figure IV.28, veut dire que l'ordonnancement obtenu est meilleur que celui obtenu avec la règle LPT.

iii. Le nombre de tâches aléatoires traitées :

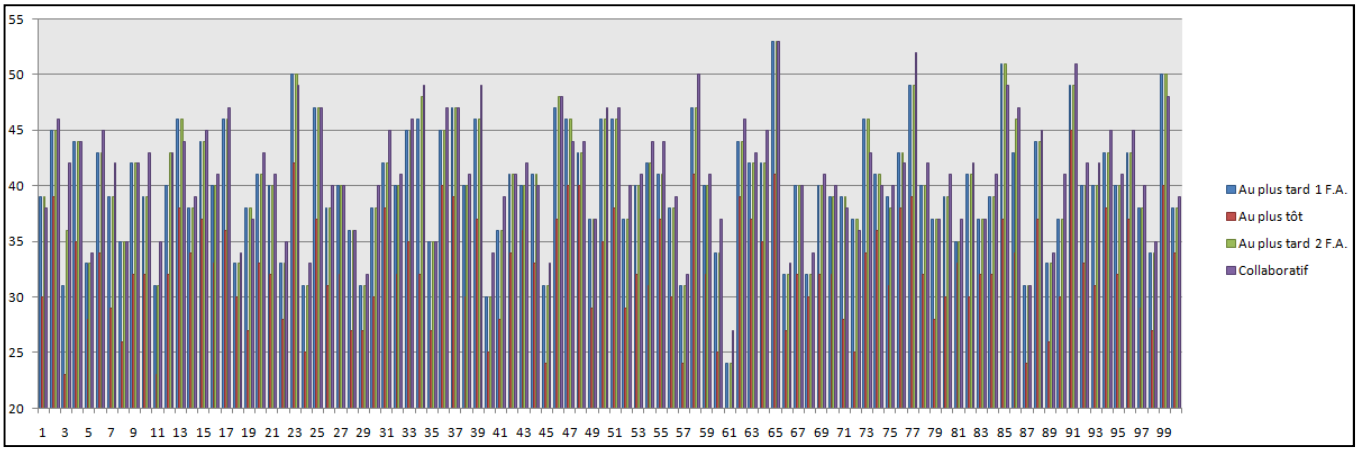


Figure IV.31 : Observations du nombre de tâches aléatoires pour les quatre méthodes

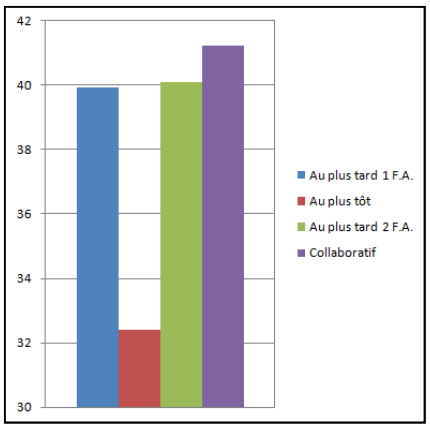


Figure IV.32 : Les moyennes des nombres de tâches aléatoires traitées des quatre modèles

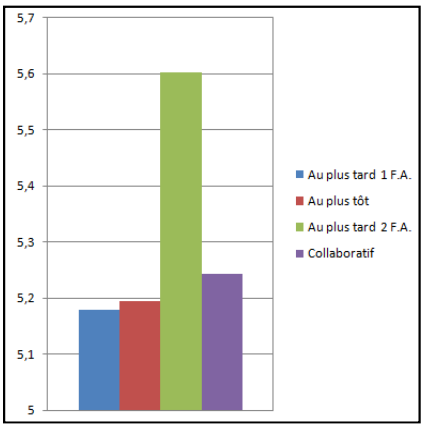


Figure IV.33 : Les écarts types des nombres de tâches aléatoires traitées des quatre modèles

L'histogramme des moyennes montre que le modèle au plus tôt traite le plus faible nombre de tâches aléatoires. Ce qui est expliqué par le fait que le modèle priorise les tâches programmables. En contre partie, le modèle collaboratif présente un léger avantage, quant à l'exécution des tâches aléatoires. Du fait que dans l'algorithme de ce dernier, l'exécution des

tâches aléatoires est avantagée. Ceci en maximisant à chaque fois le nombre de tâches aléatoires qui peuvent être exécutées et en choisissant parmi les tâches aléatoires candidates, celles qui présentent les plus faibles durées opératoires et les plus faibles dates échues.

Ainsi, pour le critère maximisant le nombre de tâches aléatoires exécutées, l'algorithme collaboratif est le meilleur. En présentant une moyenne de **41,2** contre des moyennes : **40,1** ; **40,0** et **32,4** pour les modèles, respectivement : au plus tard avec deux files d'attente, au plus tard avec une file d'attente et au plus tôt.

Les variances sont, une fois de plus, proches les unes des autres, ce qui rend les résultats des différents modèles comparables.

iv. La cadence :

La cadence d'un modèle mesure le nombre de tâches aléatoires exécutées par unité de temps. Sa formule est comme suit :

$$Cadence = Nombre\ de\ tâches\ aléatoires\ exécutées\ /\ C_{max}$$

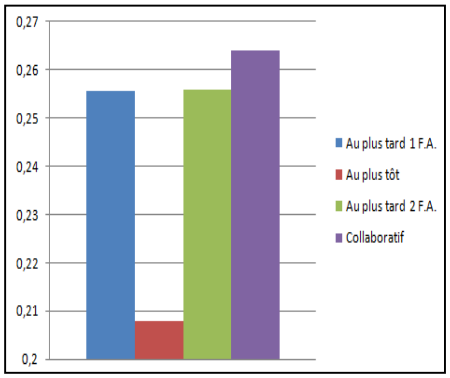


Figure IV.34 Moyennes des cadences des quatre modèles

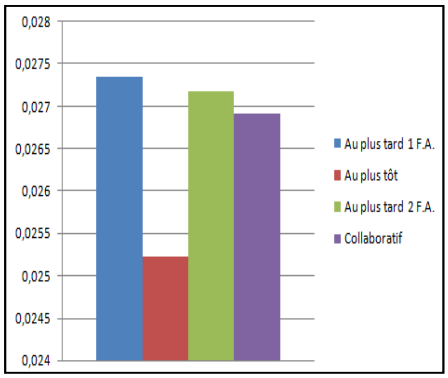


Figure IV.35 : Ecarts type des cadences des quatre modèles

Nous constatons que la cadence est légèrement supérieure pour le modèle collaboratif, suivi des deux algorithmes au plus tard (à une file et à deux files d'attente). Elle est relativement médiocre pour le modèle au plus tôt.

Le modèle collaboratif présente une moyenne de **0,26399**. Alors que les modèles au plus tard à une file d'attente, au plus tard à deux files d'attente et au plus tôt présentent des moyenne : **0,25573 ; 0,25585 ; 0,20785** respectivement.

Ceci veut dire que pour un même C_{max}=**500** unités de temps (par exemple), si le modèle collaboratif exécute à peu près **132** tâches, le modèle au plus tard à une file d'attente exécute **128** tâches, celui à deux files d'attente exécute **129** tâches et le modèle au plus tôt exécute **101** tâches. Et également, pour un même nombre de tâches exécutées égal à **200** (à titre d'exemple), le collaboratif le fait en **758 u.t.** (unités de temps), le modèle au plus tard à une file d'attente l'effectue en **782 u.t.** , celui à deux files d'attente en **782 u.t.** et le modèle au plus tôt en **862 u.t.**.

Les variances des quatre modèles sont proches relativement aux moyennes, ce qui permet d'effectuer une bonne comparaison.

V. Validation des résultats et recommandations:

Afin de valider les résultats pour un cas général, nous avons procédé à tester le comportement des différents modèles, face à des paramètres différemment proportionnés, voir même extrêmes. Pour ce faire, nous avons fait varier les durées opératoires des tâches programmables et celles des tâches aléatoires. Pour les mêmes dates de disponibilité et échues, la variation des durées opératoires nous permet de tester la capacité des modèles à « absorber » les cas extrêmes et par conséquent, l'aléa. Cette caractéristique des algorithmes est appelée la robustesse.

Dans ce qui suit nous allons tester les quatre modèles avec des paramètres (durées opératoires) différents (voir le tableau IV.1). Cette manière de faire nous permet de valider notre jugement quant à la performance des quatre modèles, et ceci pour les deux critères (C_{max} et le nombre de tâches aléatoires exécutées).

Numéro de l'expérience	$p_{tâches\ programmables}$	$p_{tâches\ aléatoires}$
1	Triangulaire (0.5, 1, 1.5)	Triangulaire (1, 2, 3)
2	Triangulaire (0.5, 1, 1.5)	Triangulaire (3, 4, 5)
3	Triangulaire (0.5, 1, 1.5)	Triangulaire (5, 6, 7)
4	Triangulaire (0.5, 1, 1.5)	Triangulaire (7, 8, 9)
5	Triangulaire (0.5, 1, 1.5)	Triangulaire (9, 10, 11)
6	Triangulaire (1, 1.5, 2.5)	Triangulaire (1, 2, 3)
7	Triangulaire (1, 1.5, 2.5)	Triangulaire (3, 4, 5)
8	Triangulaire (1, 1.5, 2.5)	Triangulaire (5, 6, 7)
9	Triangulaire (1, 1.5, 2.5)	Triangulaire (7, 8, 9)
10	Triangulaire (1, 1.5, 2.5)	Triangulaire (9, 10, 11)
11	Triangulaire (1.5, 2, 2.5)	Triangulaire (1, 2, 3)
12	Triangulaire (1.5, 2, 2.5)	Triangulaire (3, 4, 5)
13	Triangulaire (1.5, 2, 2.5)	Triangulaire (5, 6, 7)
14	Triangulaire (1.5, 2, 2.5)	Triangulaire (7, 8, 9)
15	Triangulaire (1.5, 2, 2.5)	Triangulaire (9, 10, 11)
16	Triangulaire (2, 2.5, 3)	Triangulaire (1, 2, 3)
17	Triangulaire (2, 2.5, 3)	Triangulaire (3, 4, 5)
18	Triangulaire (2, 2.5, 3)	Triangulaire (5, 6, 7)
19	Triangulaire (2, 2.5, 3)	Triangulaire (7, 8, 9)
20	Triangulaire (2, 2.5, 3)	Triangulaire (9, 10, 11)

Tableau IV.1 : paramètres des durées opératoires pour chaque expérimentation

Nous nous sommes contentés d'augmenter les durées des tâches programmables dans une plage allant de **0.5** jusqu'à **3**, afin de conserver l'ordonnançabilité des tâches programmables.

Quant aux durées opératoires (p) des tâches aléatoires, nous les avons faites varier de **2** jusqu'à **10**.
De cette manière nous obtiendrons des rapports allant de **0.05** jusqu'à **1.5**. Ces rapports sont obtenus par la formule :

$$\frac{\bar{p}_{tâches\ programmables}}{\bar{p}_{tâches\ aléatoires}}$$

i. Le C_{max} moyen :

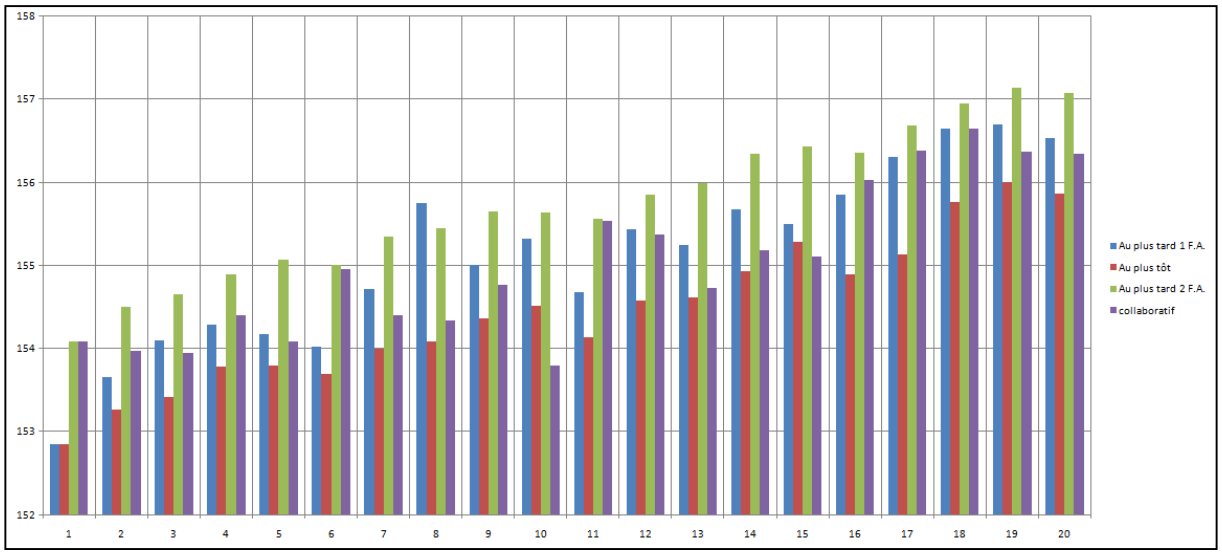


Figure IV.36 : Observations du Cmax moyen des quatre modèles

Nous remarquons dans la figure IV.36, que le C_{max} moyen pour les quatre méthodes et cela, pour chaque expérience, varie autour de la même valeur (i.e. : il n'y a pas beaucoup d'écart pour le C_{max}, entre les différents modèles), d'où, les quatre modèles sont stables quant au C_{max} (peu importe le changement de paramètres effectué).

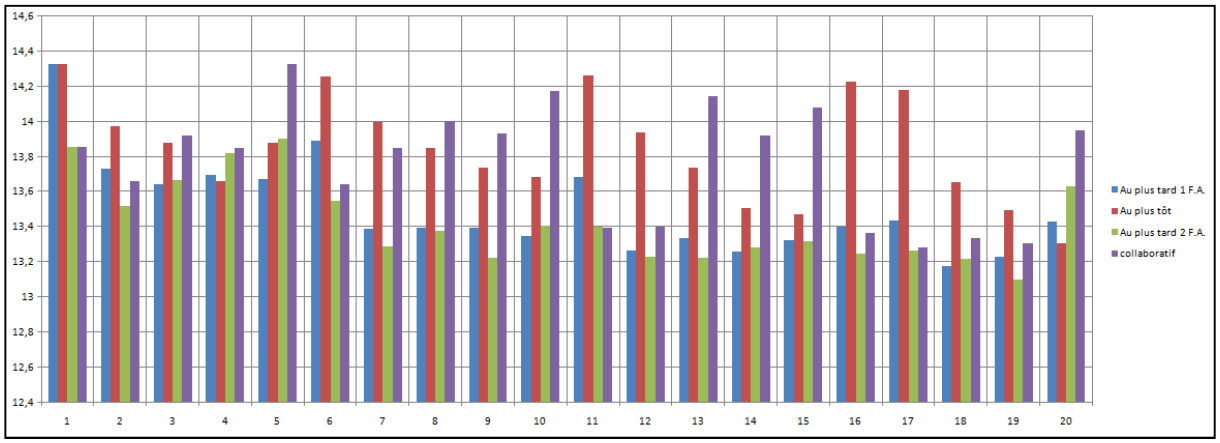


Figure IV.37 : Observations de l'écart type du C_{max} moyen des quatre modèles

Même remarque au sujet de l'écart type du C_{max} moyen.

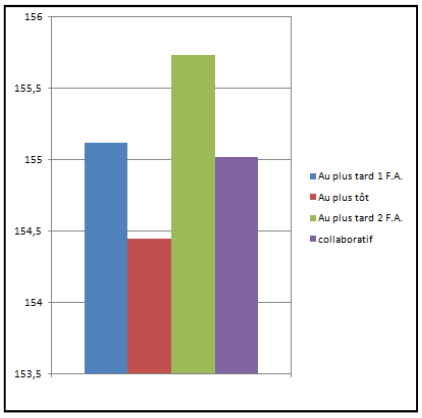


Figure IV.38 : Les moyennes des Cmax moyens des quatre modèles

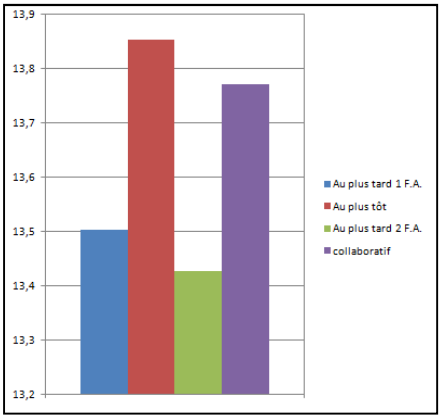


Figure IV.39 : Les écarts type des Cmax moyens des quatre modèles

On peut observer dans la figure IV.38 que la moyenne du C_{max} moyen est la moins importante pour le cas au plus tôt, l'approche collaborative a un C_{max} moyen un peu plus important, juste après le cas au plus tard avec une file d'attente et enfin, le C_{max} le plus important a été obtenu par l'approche au plus tard avec deux files d'attente.

On peut noter que c'est le même résultat que celui obtenu avec l'exemple à 100 expérimentations.

Pour les écarts types (figure IV.39), ils varient tous autour de 13,5.

ii. Le nombre de tâches aléatoires exécutées :

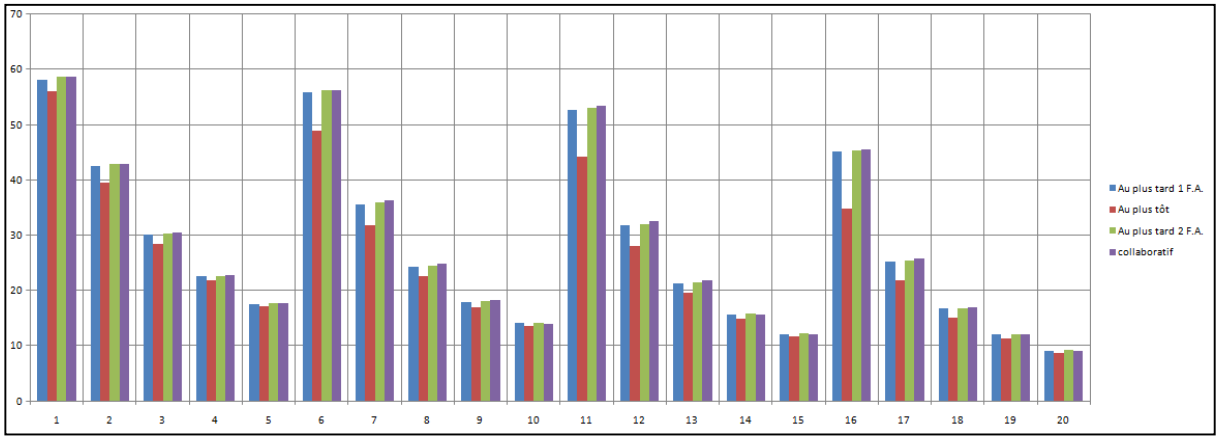


Figure IV.40 : Observations du nombre de tâches aléatoires exécutées des quatre modèles

Pour le nombre de tâches aléatoires exécutées (figure IV.40), le comportement des modèles est le même peu importe les proportions entre les durées opératoires et leurs valeurs. Ainsi, on a dans tous les cas, le modèle collaboratif qui exécute le plus de tâches aléatoires puis, au plus tard avec 2 files d'attente, au plus tard avec 1 file d'attente et au plus tôt, dans cet ordre là.

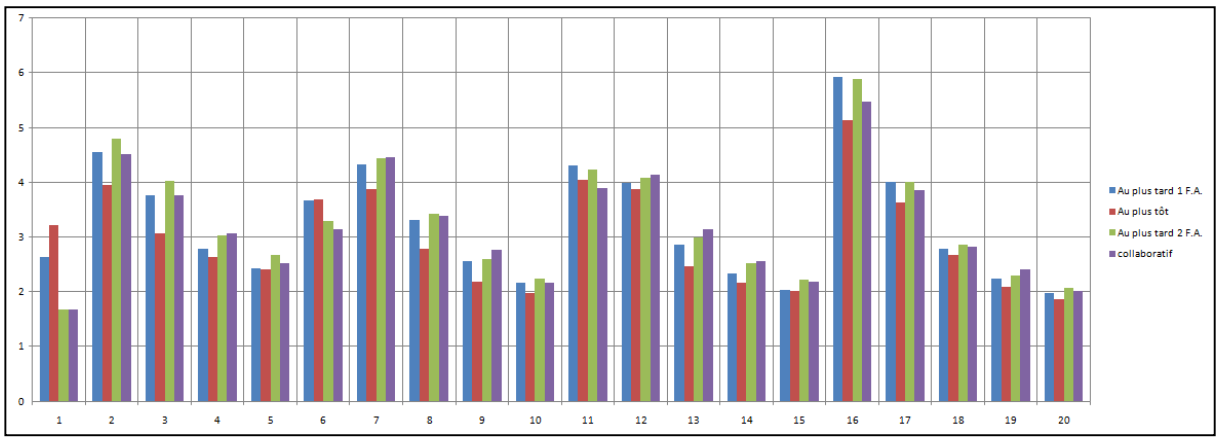


Figure IV.41 : Observations des écarts types du nombre de tâches aléatoires exécutées des quatre modèles

Nous remarquons sur la figure IV.41, que l'écart type est stable pour toutes les expériences et qu'il varie très peu d'un modèle à l'autre.

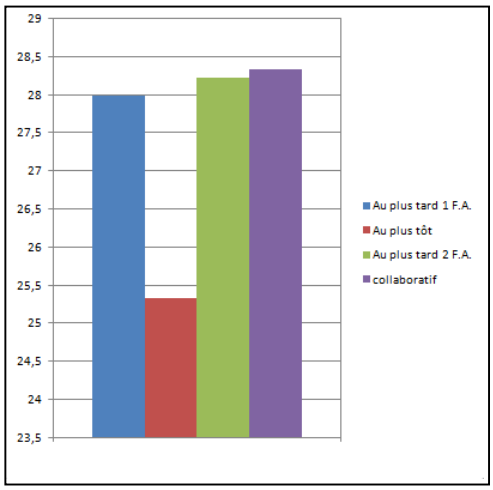


Figure IV.42 : Les moyennes des nombres de tâches aléatoires exécutées des quatre modèles

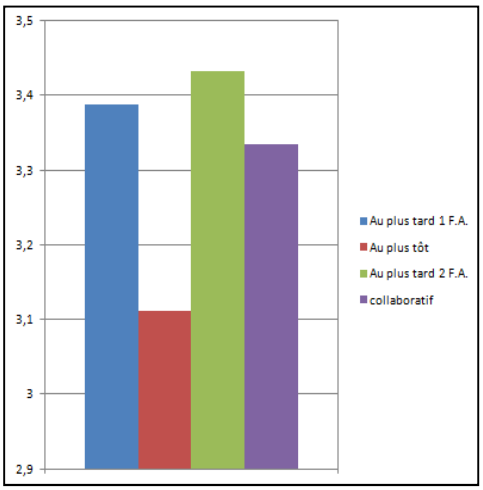


Figure IV.43 : Les écarts types des nombres de tâches aléatoires traitées des quatre modèles

Les résultats sur le classement des modèles, selon le nombre de tâches aléatoires exécutées, présenté dans la section précédente sont confirmés avec la figure IV.42. Quant aux écarts types, ils varient très peu autour de 3 (figure IV.43).

iii. Les recommandations relatives aux choix du meilleur modèle :

Grâce aux expériences effectuées dans cette section avec des durées opératoires qui suivent des distributions de paramètres différents, nous avons confirmé les résultats. Ainsi, le modèle collaboratif a présenté un C_{max} inférieur au modèle au plus tard avec deux files d'attente, légèrement inférieur à celui du modèle au plus tard avec une file d'attente mais supérieur au C_{max} du modèle au plus tôt. Et concernant le nombre de tâches aléatoires exécutées, le modèle collaboratif reste toujours en tête, avec un résultat loin de celui du modèle au plus tôt.

De ce fait, si on veut maximiser le nombre de tâches aléatoires exécutées, le modèle collaboratif est le plus adapté suivi du modèle au plus tard avec deux files d'attente. Sinon, si

on ne veut optimiser que le C_{max}, le modèle au plus tôt donne de meilleurs résultats. Enfin, si on veut optimiser la combinaison des deux critères, il est préférable d'utiliser le modèle collaboratif ou le modèle au plus tard avec une file d'attente.

VI. Conclusion :

L'objectif de ce dernier chapitre est de, en utilisant la simulation, valider les résultats obtenus dans [Bou 07] dans un premier temps, et, dans un second temps, la comparaison entre la performance de l'approche collaborative proposée dans le cadre de ce travail et les autres approches.

Pour le premier objectif, l'expérimentation a conduit à un résultat très intéressant, puisque l'approche citée comme étant optimale dans [Bou 07], à savoir, l'algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente s'avère être la deuxième en performance après l'algorithme d'ordonnancement temps-réel au plus tard dans le cas de deux files d'attente (ceci par rapport au nombre de tâches aléatoires exécutées, étant donné que tous les modèles présentent un C_{max} moyen équivalent autour de 156 unités de temps).

Pour le second objectif concernant la performance de l'approche collaborative, le résultat est encore plus intéressant, puisque, nous avons réussi à améliorer le résultat, en ordonnançant (exécutant) un nombre moyen de tâches aléatoires plus important que toutes les autres approches, pour un C_{max} équivalent.

Pour la confirmation du résultat dans le cas d'un grand nombre d'expériences (ce qui peut représenter le fonctionnement d'un atelier de production durant plusieurs années si on considère chaque expérience comme 1 jour de production), nous avons lancé la simulation, une dernière fois, pour 1000 expériences différentes avec des paramètres différents des 100 expériences présentées précédemment (avec 75 tâches programmables et 50 tâches aléatoires dont les dates d'arrivées suivent une EXPO(4). Les flots utilisés ont été changés aussi). Les résultats obtenus sont :

- L'approche collaborative : $\overline{\mathrm{NTAT}} = 39{,}30$, $\overline{C_{max}} = 157{,}1$.
- Au plus tard avec 2 files d'attente : $\overline{\mathrm{NTAT}} = 38{,}60$, $\overline{C_{max}} = 170{,}4$.
- Au plus tard avec 1 file d'attente : $\overline{\mathrm{NTAT}} = 35{,}34$, $\overline{C_{max}} = 158{,}4$.
- Au plus tôt avec 1 file d'attente : $\overline{\mathrm{NTAT}} = 28{,}00$, $\overline{C_{max}} = 155{,}8$.

Nous remarquons dans ce cas, que le C_{max} dans le cas au plus tard avec 2 files d'attente est le plus important (sachant que les algorithmes avec deux files d'attente ne sont pas construits avec comme objectif principal la réduction du C_{max}. Leur objectif principal étant, la maximisation du nombre de tâches aléatoires à exécuter). De là, et puisque nous avons deux paramètres à optimiser (C_{max} et NTAT), la décision quant à l'approche la plus performante entre celle au plus tard avec 1 file d'attente et celle avec 2 files d'attente reste en suspens (selon l'objectif recherché). Ceci étant, la performance de l'approche collaborative est confirmée (avec le nombre le plus important de tâches aléatoires exécutées et le C_{max} le plus faible si on ignore celui de l'approche au plus tôt dont le nombre de tâches aléatoires exécutées est médiocre en comparaison).

Conclusion générale et perspectives

L'ordonnancement dans un atelier de production est une problématique d'actualité, dont l'étude et le développement d'arsenal méthodologique pour sa prise en charge semble indispensable. Dans le contexte économique actuel, aucune entreprise se voulant performante, voir même, voulant pérenniser ses activités, ne peut se permettre une gestion approximative de sa production.

Nous avons vu dans ce mémoire, que l'ordonnancement a des règles et des bases bien établies, mais aussi, il est caractérisé par une capacité d'adaptation à différentes problématiques, ce qui rend les études menées dans ce domaine très captivantes pour les chercheurs. Le fait d'adapter à l'ordonnancement, les outils développés dans d'autres domaines d'activités, tel que ce que nous avons vu pour les SMA, n'est pas en marge de cet intérêt croissant que lui portent les chercheurs.

Le problème étudié dans ce travail, consiste à ordonnancer sur plusieurs ressources un certain nombre de tâches arrivant aléatoirement, sachant, que le système dispose d'un certain nombre de tâches, dites programmables, à traiter. Le critère à optimiser est la durée totale de l'ordonnancement (Makespan) sous des contraintes liées aux ressources, aux intervalles temporels et aux tâches elles-mêmes, ainsi que le nombre de tâches aléatoires ordonnancées (exécutées).

Notre travail a consisté, à valider les résultats obtenus dans [Bou 07], puis à développer une approche collaborative, pour la résolution du problème d'ordonnancement temps-réel sur deux ressources identiques en parallèle, en vu de permettre l'ordonnancement d'un plus grand nombre de tâches aléatoires en respectant les délais d'exécution de toutes les tâches.

L'approche collaborative proposée, nous a permis d'alléger la complexité du problème original, en distribuant l'ordonnancement sur les différents agents (machines) constituant notre atelier de production. Ce qui nous a conduit à proposer une solution qui peut être généralisée au cas de plusieurs machines en parallèle sans augmenter, pour autant, la difficulté du problème initial, ce qui n'était pas le cas avec les approches déjà existantes, qui voient la complexité de leur application augmenter en fonction du nombre de machines constituant l'atelier, jusqu'à rendre la solution inapplicable.

Pour réaliser ces objectifs, nous avons procédé par deux phases. En premier lieu, nous avons modélisé les différentes approches proposées dans [Bou 07], ainsi que, proposé une approche collaborative que nous avons modélisée aussi. En second lieu, nous avons réalisé une expérimentation des différents modèles. Les résultats obtenus peuvent être résumés comme suit :

- L'approche collaborative est, dans tous les cas étudiés, plus performante que les autres approches de résolution ;
- L'algorithme d'ordonnancement temps-réel au plus tard dans le cas de deux files d'attente, présente un nombre de tâches aléatoires exécutées, toujours en seconde position après l'approche collaborative. Ceci étant le C_{max} obtenu est, parfois, supérieur à ceux obtenus par les autres méthodes de résolution, quand il n'est pas voisin de ces derniers ;

- L'algorithme d'ordonnancement temps-réel au plus tard dans le cas d'une file d'attente, présente un nombre de tâches aléatoires exécutées, en troisième position. Cependant, le C_{max} obtenu est, dans tous les cas, inférieur à celui obtenu par L'algorithme d'ordonnancement temps-réel au plus tard dans le cas de deux files d'attente (nous le trouvons souvent voisin de ce dernier, sauf dans des cas, où il est nettement inférieur) ;
- L'algorithme d'ordonnancement temps-réel au plus tôt dans le cas d'une file d'attente, présente un nombre de tâches aléatoires exécutées, très inférieur par rapport aux méthodes de résolution précédentes. Ceci, même si le C_{max} obtenu est, dans tous les cas, inférieur à celui obtenu par ces dernières, sans pour autant être plus intéressant (puisque, dans le meilleur des cas, ce C_{max} est inférieur avec deux unités de temps en moyenne pour un nombre de tâches aléatoires exécutées nettement inférieur, en moyenne 8 tâches de moins que la moins performante des autres méthodes).

Ce travail nous a permis d'enrichir nos connaissances dans divers domaines. Que ce soit dans l'ordonnancement d'atelier, la simulation, l'Intelligence Artificielle Distribuée grâce à l'utilisation des systèmes multi agents (SMA) pour la résolution de notre problème via l'approche collaborative, mais aussi, en algorithmique.

La simulation a pris une grande part dans la crédibilisation de notre travail, puisque, elle nous a permis de trancher entre les différentes approches de résolution et de montrer que l'approche collaborative proposée, présente un meilleur résultat que les approches existantes. Sans oublier que la simulation est un outil d'aide à la décision, et qu'elle joue un rôle très important pour l'industriel, par conséquent, sa maîtrise est un atout non négligeable.

L'outil que nous avons utilisé pour réaliser la simulation est le simulateur ARENA Rockwell 10, son utilisation a requis l'acquisition d'un certain nombre de connaissances, et non des moindres. Ainsi, nous nous sommes familiarisés avec le langage de simulation SIMAN et le langage visuel basic (VBA) qui nous a permis d'intégrer des programmes pour transcrire le principe exact des méthodes à simuler quand il n'était pas possible de le faire avec les blocs prédéfinis, ce langage est utilisé dans la plupart des logiciels comme MS Excel, AutoCAD, …

Considérant ce travail comme une suite logique de celui réalisé par Mlle Bougchiche dans [Bou 07], dont, les perspectives étaient résumées en :

- La sélection de la tâche aléatoire à ordonnancer était basée sur des règles de priorité statiques. Il serait intéressant d'analyser l'impact d'un choix dynamique de ces règles. Dans ce cas, chaque sélection de la tâche aléatoire est basée sur une règle de priorité différente d'un ordonnancement à un autre.
- Etudier le cas où les durées opératoires (les tâches programmables et les tâches aléatoires) sont variables est intéressant étant donné que l'exemple qui a été étudié, considérait dans un premier temps les durées opératoires égales pour les deux types de tâches et les durées opératoires des tâches aléatoires variables.
- Comme une autre perspective, il serait intéressant, voire même nécessaire, de trouver une approche d'évaluation de la performance d'une heuristique développée pour un ordonnancement temps-réel. Une approche empirique ou analytique est envisageable.
- Le respect des intervalles temporels des tâches aléatoires était primordial. L'acceptation d'un retard sur les délais de ces tâches modifierait complètement

l'ordonnancement. Il est intéressant de considérer, comme objectif de production à optimiser, la somme des retards ou le retard maximal.

Dans notre travail, nous pouvons considérer que les trois premières perspectives qui ont été proposées, ont été prises en charge. L'étude de la quatrième perspective, reste très intéressante, pour l'étude du type d'atelier sujet de ce mémoire.

Nous rajoutons à cela, une perspective, qui nous est apparue comme prometteuse dans le cadre du type d'atelier à ressources identiques en parallèle en temps-réel. Celle ci, peut donner lieu à d'autres travaux futurs :

« Le réordonnancement des tâches programmables est une perspective très intéressante pour l'amélioration du résultat, même si une tentative, infructueuse, dans ce sens, a été effectuée lors de l'élaboration de ce travail. »

Ouvrages, thèses, mémoires et revues

[Azi 99]: Azizoglu M., Kirca O., 1999, "On the minimization of total weithed flow time with identical and uniform parallel machines", European Journal of operational Research, 113, p. 91-100.

[Azi 03]: Azizoglu M., Webster S., 2003, "Scheduling parallel machines to minimize flowtime with family set-up times", International Journal of Production Research, 41, p. 1199-1215.

[Bak 96]: Bakalem M., 1996, "Modélisation et simulation orientées objet des systèmes manufacturiers", Thèse de doctorat en Électronique –Électrotechnique- Automatique, Université de Savoie, France.

[Bel 94]: Belouadeh H., Potts C. N., 1994, "Scheduling identical parallel machines to minimize total weighted completion time", Discrete Appl. Math., 48, p. 201-218

[Ber 00]: Berchet C., 2000, "Modélisation pour la simulation d'un système d'aide au pilotage industriel", Thèse de doctorat en génie industriel, Institut National Polytechnique de Grenoble, France.

[Ber 91]: Bernard G., Stève D., Simatic M., 1991, "Placement et migration de processus dans les systèmes réparties faiblement couplés", Technique et Sciences Informatiques, vol. 10, n5, pp. 375–392.

[Bil 99] : billaut J-C, 1999, "recherche opérationnelle et aide à la décision pour les problèmes d'ordonnancement", Habilitation à diriger des travaux de recherche, laboratoire d'informatique, Université François Rabelais, Tours, France.

[Bou 02]: Boujut J.F., Cavaillé J.B., Jeantet A., 2002, "Instrumentation de la coopération", Coopération et connaissance dans les systèmes industriels, pages 91-109. Lavoisier, Hermes Science.

[Bou 07] : Bougchiche F., 2007, "Contribution à l'ordonnancement temps-réel sur des ressources identiques en parallèle", mémoire de magistère en Automatique, Ecole Nationale Polytechnique, Algérie.

[Bru 74] : Bruno J.W., Coffman E.G., Sethi R., 1974, "Scheduling independent tasks to reduce mean finishing time", AIIE Transactions, 17, p. 382-387.

[Cam 00]: Camalot J.P., 2000, "Aide à la décision et à la coopération en gestion du temps et des ressources", Thèse de Doctorat, Institut National des Sciences Appliquées de Toulouse, Toulouse, France.

[Car 88]: Carrie A., 1988, "Simulation of manufacturing systems", John Wiley & Sons.

[Cas 88]: Casavant T. L., Kuhl J. G., 1988, "A taxonomy of scheduling generalpurpose distributed computing systems", IEEE Trans. on Software Engineering, vol. 14, n2, pp. 141–154.

[Cer 88]: Cernault A., 1988, "La simulation des systèmes de production", Cepadues Editions, Toulouse.

[Che 99]: Chen Z. L., Powell W. B., 1999, "Solving parallel machine scheduling problems by column generation", INFORMS Journal on Computing, 11, p. 87-94.

[Che 99a]: Chevochot P., Puaut I., 1999, "Tolérance aux fautes dans les systèmes répartis temps-réel strict", Techniques et Sciences Informatiques (TSI), 18(8) :837-870.

[Che 99b]: Chevochot P., Puaut I., 1999, "An approach for fault-tolerance in hard-time distributed systems", Proc. 18th IEEE Symposium on Reliable Distributed Systems, Lausanne, Switzerland, pages 292–293.

[Cos 99]: Cost R. S., Chen Y., Finin T., Labrou Y., Peng Y., 1999, "Modeling Agent Conversations with Colored Petri Colored", Working Notes of the Workshop on Specifying Conversation Policies, Autonomous Agents' 99, Seattle, Washington.

[Dav 94]: Davis L., Williams G., 1994, "Evaluating and selecting simulation software using the analytic hierarchy process", Integrated Manufacturing Systems, Vol. 5, No. 1, pp. 23-32.

[Des 82]: Deschizeaux P., 1982, "Temps et évènement dans les systèmes distribués de contrôle de procédé", R.A.I.R.O. Automatique/Systems Analysis and Control. 16(3) :259-74.

[Dun 00]: Dunstall S., Wirth A., Baker K., 2000, "Lower bounds and algorithms for flowtime minimization on a single machine with set-up times", Journal of Scheduling, 3, p. 51-69.

[Dun 05a]: Dunstall S., Wirth A., 2005a, "A comparaison of branch-and-bound algorithms for a family scheduling problem with identical parallel machines", European Journal of Operational Research, 167.

[Dun 05b]: Dunstall S., Wirth A., 2005b, "Heuristic methods for the identical parallel machine flowtime problem with set-up times", Computers & Operations Research, 32, p. 2479-2491.

[Dur 95]: Durfee E.H., 1995, "Distributed Artificial Intelligence", The Handbook of Brain Theory and Neural Networks, M. Arbib (eds), MIT Press, pp. 232- 339.

[Ell 91]: Elloy J., 1991, "Les contraintes du temps réel dans les systèmes industriels répartis", RGE N2/91, pages 26–34.

[Elm 74]: Elmaghraby S., Park S.H., 1974, "Scheduling jobs on a number of identical machines", AIIE Transactions, 6 p. 1-13.

[Fer 95]: FERBER J., 1995, "Les systèmes multi-agents : vers une intelligence collective", Informatique, Intelligence Artificielle. InterÉditions.

[Fin 94]: Finin T., Fritzson R., McKay D., McEntire R., 1994, "KQML as an Agent Communication Language", Proceedings of the Third International Conference on Information Management (CIKM'94), ACM Press.

[Fip 97]: Foundation for Intelligent Physical Agents (FIPA), 1997, "FIPA 07 Specification part 2 – Agent Communication Language", disponible sur :www.fipa.org.

[Fis 97]: Fishwick P.A., 1997, "Computer Simulation : Growth Through Extension", Transactions of the Society for Computer Simulation International, n° 14, pp. 13- 23.

[Gar 78]: Garey M. R., Johnson D.S., 1978, " 'Strong' NPcompletness results: Motivation, examples and implications", Journal of the Association for Compting Machinery, 25, p. 499-508.

[Gas 87]: GASSER L., BRAGANZA C., HERMAN N., 1987, "MACE : A Flexible Testbed for Distributed AI Research", chapitre 5, pages 119–152. Pitman, San Francisco.

[Gaz 03] : S. H. Gazoby, 2003, "Planification des tâches dans un environnement d'ateliers à flux tirés". Mémoire de Magister, EMP, Algérie.

[Gia 03]: Giard V., 2003, "Gestion de la production et des flux", 3ème édition, Collection Gestion, Série : Production et techniques quantitatives appliquées à la gestion, Economica, 1229p.

[Gra 69]: Graham R. L., 1969, "Bounds on multiprocessing timing anomalies", SIAM Journal on Applied mathematics, 17, p. 416-429.

[Gra 05] : B. GRABOT, "Ordonnancement d'ateliers manufacturiers". Techniques de l'ingénieur, AG 3 015.

[Gro 99]: Grolleau E., 1999, "Ordonnancement temps réel hors-ligne optimal à l'aide de réseaux de Petri en environnement monoprocesseur et multiprocesseur", Thèse, ENSMA-Université Poitiers.

[Hab 01]: Habchi G., 2001, "conceptualisation et modélisation pour des systèmes de production", pour l'obtention de l'habilitation à diriger des recherches, université de Savoie.

[Had 96]: Haddadi A., Sundermeyer K., 1996, "Belief-Desire-Intention Agent Architectures", Foundations of Distributed Artificial Intelligence, Chap. 5, G.M.P. O'Hare et N.R. Jennings eds., J. Wiley and sons Inc., pp. 169-185.

[Hil 80]: Hillier F.S., Lieberman G.J., 1980, "Operations Research", 3rd ed., Holden-Day, San Francisco.

[Hol 90]: Hollocks B., 1990, "Introducing simulation successfully", Proceedings of the CIM'90 Conference, NEC, Burmingham, UK.

[Huh 98]: Huns M.N., Singh M.P., 1998, "Agents and Multiagent Systems", Themes: Approaches, and Challanges, Readings in Agents M.N. Huhns and M.P. Singh Eds., Morgan Kaufmann Pbs, pp. 1-22.

[Jac 96] : Jacqmot C., 1996, "Load Management in Distributed Computing Systems : Towards Adapti ve Strategies", Thèse de doctorat, Université Catholique de Louvain, FSA–DII, Louvain, Belgium.

[Kel 03]: Kelton W.D., 2003,"Simulation with Arena", 3éme Edition, International Edition, New York.

[Lis 91]: Lisper B., 1991, "Detecting static algorithms by partial evaluation", Partial Evaluation and Semantics, Based Program Manipulation, New Haven, Connecticut, Sigplan Notices, vol. 26, no. 9, September 1991, pp. 31–42, New York.

[McN 59]: McNaughton R., 1959, "Scheduling with deadlines and loss functions", Management Science, 6, p. 1-12.

[Mas 91]: Mason A.J., Anderson E.J., 1991, "Minimizing flow time on a single machine with job classes and setup times", Naval Research Logistics, 38.

[Maz 99]: Mazouzi H., El Fallah-Seghrouchni A., 1999, "Une démarche méthodologique pour l'ingénierie des protocoles d'interaction", Actes Ingénierie des systèmes multi-agents, JFIADSMA'99, Saint-Gilles, Ile de la Réunion.

[Moi 90]: Le Moigne J.L., 1990, "La modélisation des systèmes complexes", Dunod, France.

[Mok 83]: Mok A.K. , 1983, "Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environnement", PhD Thesis, Departement of electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge.

[Mon 89]: Monma C.L., Potts C.N., 1989, "On the complexity of scheduling with batch setup times", Operations research, 37.

[Our 01] : Ourari S., 2001, "Ordonnancement temps-réel en présence d'évènements aléatoires", Mémoire de Magister, EMP, Algérie.

[Par 91]: Parnas D., Xu J., 1991, "On satifying timing constraints in hard-real-time systems", Software Engineering Notes, 16(4) : 132-46.

[Peg 95]: Pegden C., Shannon R., Sadwski R., 1995, "Introduction to simulation using SIMAN", 2ème Edition, Mac Graw-Hill, New-York.

[Pie 90]: Pierreval H., 1990, " Les méthodes d'analyse et de conception des systèmes de production", Hermès, France.

[Pri 86]: Pritsker A.A.B., 1986, "Introduction to simulation and SLAM II", Halsted Press, New York, NY, 3rd edition.

[Pri 89]: Pritsker A., Sigal C., Hammesfahr R., 1989, "SLAM II : Network Models for decision Support" , Prentice-Hall, Englewood Cliffs, NJ.

[Pro 93]: Proth J., Xie X., 1993, "Performance evaluation and optimization of stochastic timed event graphs", IEEE Transactions on Automatic Control, 38.

[Rob 98]: Roberts C.A., Dessouky Y.M., 1998, "An overview of object-oriented simulation", Simulation Journal, The Society for Computer Simulation International, Vol. 70, No. 6, pp. 359-368.

[Ros 75]: Ross G.T., Soland R.M., 1975, "A branch and bound algorithm for the generalized assignment problem", Mathematical Programming, 8, 91-103.

[Sea 72]: Searle J.R., 1972, "Les actes de langages", Paris, Herman.

[Sho 93]: Shoham Y., 1993, "Agent Oriented Programming", in.Artificial Intelligence , Vol. 60, n°1, pp. 51-92.

[Sta 88]: Stankovic J., 1988, "Misconceptins about real-time computing. IEEE Computer, pages 10–19.

[Tal 97]: Talbi E.G., 1997, "Une taxonomie des algoritmes d'allocation dynamique de processus dans les systèmes parallèles et distribués", ISYPAR'97 2ème Ecole d'Informatique des Systèmes Parallèles et Répartis, pp. 137–164. – Toulouse, France.

[Tra 01]: Tranvouez E., 2001, "IAD et ordonnancement : une approche coopérative du réordonnancement par systèmes multi-agents", Thèse de doctorat,Université de Droit, dEconomie et des Sciences dAix-Marseille III, Faculté des Sciences et Techniques de Saint-Jérôme,France.

[Ver 96]: Vernadat F., 1996, "Entreprise modelling and integration : principles and applications", Chapman & Hall, London.

[Ver 97]: Vernadat F., 1997, "Entreprise modelling languages ", ICEIMT'97 Entreprise Integration – International Consensus, EI-IC ESPRIT Project 21.859, Hermès, Paris.

[Ver 99]: Vernadat F., 1999, "Techniques de modélisation en entreprise : application aux processus opérationnels", Economica.

[Web 95]: Webster S., 1995, "Weighted flow time bounds for scheduling identical processors", European Journal of Operational Research, 80, p. 103-111.

[Web 97]: Webster S., 1997, "The complexity of scheduling job families about common due date", Operations Research Letters, 20 p. 65-74.

[Web 01]: Webster S., Azizoglu M., 2001, "Dynamic programming for scheduling parallel machines with family setup times", Computers & Operations Research, 28.

[Zaf 07]: Zaffalon L., 2007, "Programmation concurrente et temps réel", Presses Polytechniques et Universitaires Romandes(PPUR).

[Zei 76]: Zeigler, 1976, "Theory of Modeling and simulation", New-York, John Wiley and Sons, 543 pages.

[Zei 89]: Zeigler, 1989, "Hierarchical modular discrete-event modeling in an object oriented environment", Simulation, vol. 49, n°5, pp. 219-230.

La notation en α/β/γ des problèmes d'ordonnancement :

α	Système	α_1	Ø	une seule machine
			P	machines identiques en parallèle
			U	machines à vitesse proportionnelle en parallèle
			R	machines non reliées en parallèle
			O	open shop
			F	flow shop
			J	job shop
		α_2	Ø	le nombre de machines est fixé à une valeur donnée précise
			1,2…	le nombre de machines est fixé dans la définition de l'énoncé
			m	le nombre de machines varie avec chaque instance considérée
β	Contraintes	β_1	Ø	les opérations ne peuvent pas être interrompues
			pmtn	les opérations peuvent être interrompues
		β_2	Ø	il n'y a pas de contraintes de précédence
			tree	il existe des contraintes de précédence additionnelles sous forme d'arborescence ou d'anti-arborescence
			prec	il existe des contraintes de précédence additionnelles sous forme d'un graphe orienté sans circuit
		β_3	Ø	$\forall i\ r_i = 0$
			r_j	les dates de disponibilité sont différentes selon les travaux
		β_4	Ø	$\forall i\ p_i \in \mathbb{N}^*$
			p_j=1	chaque opération a une durée unitaire
			(p_j=p) & (β_3= r_j)	La durée des opérations est égale à p≠1 et au moins un des r_j n'est pas divisible par p
		β_5	Ø	les temps de réglage sont constants et inclus dans la durée des opérations
			Snsd[1]	des temps de réglage sont affectés à chaque opération, ils sont indépendants de la séquence mais non inclus dans la durée opératoire car ils peuvent être exécutés avant que l'opération n'arrive dans la file d'attente de la machine
			Ssd[1]	les temps de réglage ne sont pas inclus dans la durée des opérations et sont dépendants de la séquence des opérations sur la ressource
		β_6	Ø	les dates de fin souhaitées si elles existent sont des nombres entiers arbitraires non négatifs
			dj=d	toutes les dates de fin souhaitée sont identiques (« common due date »)
			Δj	les dates de fin souhaitée sont remplacées par des dates de fin impérative
γ	Critères	γ_1	C	Completion time
			L	Le retard « Lateness »
			T	Le retard vrai « Tardiness »
			E	L'avance vraie « earliness »
			G	une fonction générale non décroissante du « completion time »
			NT	le nombre de travaux en retard
		γ_2	K_{max}	Minimisation de la plus grande valeur d'une fonction donnée de la date de fin pour tous les travaux (K≡C, L, T, E, G)
			$\overline{K}$	Minimisation de la valeur moyenne d'une fonction donnée de la date de fin pour tous les travaux (K≡C, L, T, E)
			$\overline{K}_w$	Minimisation de la somme pondérée de la fonction considérée de la date de fin pour tous les travaux (K≡C, L, T, E)

[1] S est remplacé par R pour définir le même type de contraintes pour le démontage des outils après l'opération.

Le langage ACL :

Les différentes primitives constituant le langage ACL, sont reprises dans le tableau suivant :

Primitives	Description
accept-proposal	L'action d'accepter une proposition précédemment soumise pour effectuer une action.
agree	L'action d'accepter d'effectuer une certaine action, probablement à l'avenir.
cancel	L'action de décommander une certaine action précédemment requis qui a l'ampleur temporelle (i.e. : n'est pas instantanée).
cfp	L'action de la réclamation des propositions pour effectuer une action donnée.
confirm	L'expéditeur informe le récepteur qu'une proposition donnée est vraie, où le récepteur est connu pour être incertain au sujet de la proposition.
disconfirm	L'expéditeur informe le récepteur qu'une proposition donnée soit fausse, où le récepteur est connu pour croire, ou le croient probablement, que la proposition est vraie.
failure	L'action de dire à un autre agent qu'une action a été essayée mais la tentative a échoué.
inform	L'expéditeur informe le récepteur qu'une proposition donnée est vraie.
inform-if	Une macro action pour l'agent de l'action pour informer le destinataire si une proposition est vraie.
inform-ref	Une macro action pour que l'expéditeur informe le récepteur qu'un objet correspond à un descripteur défini (par exemple un nom).
not-understood	L'expéditeur de l'acte (par exemple i) informe le récepteur (par exemple j) qu'il a perçu que j a effectué une certaine action, mais que je n'ai pas compris quel j a juste fait.
propose	L'action de soumettre une proposition pour effectuer une certaine action, donnée certaines conditions préalables.
query-if	L'action demandant à un autre agent si une proposition donnée est vraie.
query-ref	L'action de demander à un autre agent, avec l'objet qui est rapportée par une expression.
refuse	L'action de refuser d'effectuer une action donnée, et d'expliquer la raison du refus.
reject-proposal	L'action de rejeter une proposition pour effectuer une certaine action pendant une négociation.
request	L'expéditeur invite le récepteur à effectuer une certaine action. Une classe importante des utilisations de l'acte de demande est d'inviter le récepteur à exécuter un autre acte communicatif.
request-when	L'expéditeur veut que le récepteur effectue une certaine action quand une certaine proposition donnée devient vraie.
request-whenever	L'expéditeur veut que le récepteur effectue une certaine action dès qu'une certaine proposition deviendra vraie et ensuite chaque fois que la proposition devient vraie.
request-whomever	L'expéditeur veut une action effectuée par un certain agent autre que lui-même. L'agent de réception devrait effectuer l'action ou la transmettre à un autre agent.
subscribe	L'acte de demander une intention persistante d'informer l'expéditeur de la valeur d'une référence, et de l'annoncer encore, toutes les fois que l'objet identifié par la référence change.

Les langages de simulation :

Il existe plus d'une centaine de logiciels de simulation sur le marché. Certains auteurs [Dav 94] classent ces logiciels en deux catégories : les logiciels pour la simulation à événements discrets et les logiciels pour la simulation à variables continues. Si le second type est applicable pour un flux continu de produit et d'information, le premier est beaucoup plus applicable dans les systèmes manufacturiers (composants, assemblages...). Plus récemment, Habchi dans [Hab 01] propose deux classifications, l'une basée sur la spécialisation du logiciel et l'autre basée sur l'approche de modélisation. Dans ce qui suit, nous présenterons ces deux classifications qui nous semblent les plus adaptées dans le cadre de ce travail, étant donné que certains des concepts qui y sont développés, sont repris dans notre travail.

1. Classification des langages en fonction de la spécialisation :

On dénombre actuellement un nombre assez important de langages utilisés pour la simulation. Si on s'intéresse à la spécialisation de ces langages, on peut les classer en quatre familles : les langages universels ou évolués, les langages généraux de simulation, les langages spécialisés de simulation et les langages dédiés.

a. Les langages universels ou évolués :

Se sont des langages de programmation répandus tels que *VB*, *FORTRAN*, *PASCAL*, *C*, *ADA*, *C++*, *JAVA*... Ils ont l'avantage de permettre une simulation précise et fiable de tout type de système, quel que soit son niveau de complexité. Ils permettent de développer des modèles fonctionnels ou logico-mathématiques, grâce à la richesse de programmation qu'ils offrent. Cependant, ils nécessitent des compétences poussées et affirmées en simulation, en informatique et en génie industriel. Ils présentent l'inconvénient du temps de développement et par conséquent du coût de réalisation. Les modèles sont dédiés et généralement inexploitables dans des extensions futures.

b. Les langages généraux de simulation :

Se sont des langages enrichis par des primitives dédiées à la simulation en général, comme les générateurs de nombres aléatoires, les processeurs de base pour la programmation de certaines entités... Ils ont ainsi l'avantage de présenter une couche de base permettant de développer d'autres primitives pour la réalisation de modèles de simulation dédiés. Nous pouvons citer, entre autres, *SIM++* et *MODSIM II*.

c. Les langages spécialisés de simulation :

Ils présentent un compromis entre la rapidité de développement du modèle et la rigidité liée aux concepts utilisés. Les concepts utilisés sont généralement basés sur la théorie des files d'attente et des processus stochastiques. La modélisation se fait à l'aide d'éléments symboliques. Les modèles résultants sont de type réseau *(SLAM)*, des blocs représentatifs de processus *(ARENA, GPSS, WITNESS, SIMNET II)*, ou des diagrammes de cycle d'activités

(HOCUS). Ces logiciels nécessitent une formation et de l'expérience, et sont souvent utilisés par les consultants et concepteurs qui ont souvent recours à la simulation pour des systèmes différents. L'utilisation de ces langages est conseillée quand une évolution du modèle est nécessaire.

d. Les langages dédiés :

Ils sont adaptés aux systèmes ayant des caractéristiques identiques, et sont dédiés à un type de système au fonctionnement bien défini ou à une classe de problèmes. La modélisation est basée sur l'aspect topologique (i.e. : représentation du système physique) à travers des éléments représentatifs du système réel. Ces outils très conviviaux sont limités à un usage répétitif dans un domaine restreint. L'avantage principal de ce type d'outil est la facilité d'utilisation, alors que l'inconvénient majeur est la rigidité des modèles. Parmi les langages dédiés, citons à titre d'exemple *MAP/1* et *SIMFACTORY*.

		Famille des langages			
		Universels	Généraux	Spécialisés	Dédiés
critères	Rapidité de modélisation	Très faible	Faible	Bonne	Très bonne
	Convivialité	Très faible	Faible	Bonne	Très bonne
	Flexibilité	Très bonne	Bonne	Faible	Très faible
	Fidélité du modèle	Très bonne	Bonne	Faible	Dépend du système étudié
	Niveau de compétence	Analyste classique	Analyste orienté simulation	Spécialiste simulation	Utilisateur occasionnel

Tableau II.1 : Tableau comparatif des langages de simulation selon la spécialisation [Hab 01]

2. Classification des langages en fonction de l'approche de modélisation :

Parmi les autres classifications des langages de simulation, Roberts *et al.* **[Rob 98]** proposent une classification basée sur l'approche de modélisation et la logique de changement d'état. Ils distinguent trois types de langages : les langages orientés processus, les langages pilotés par les données et les langages orientés objets.

a. Les langages orientés processus :

On peut citer, SLAM II et SIMAN qui ont respectivement les deux environnements graphiques : AweSim et ARENA. Ces langages ont réduit avec succès les frontières de programmation et ont probablement fait que la modélisation en simulation soit accessible à plusieurs utilisateurs non informaticiens. Généralement, les langages orientés processus appartiennent à la catégorie des langages spécialisés présentés précédemment.

b. Les langages pilotés par les données :

Ces langages représentent une abstraction de haut niveau du système. SIMFACTRY et PROMODEL sont des exemples de langages où le modèle de fond est difficilement

modifiable. Ces langages comprennent des objets prédéfinis capables de réaliser une série d'opérations. La modélisation consiste à sélectionner, interconnecter et renseigner ces objets (station, convoyeur...) qui représentent au mieux le système réel. Ce type de langages nécessite peu de programmation de la part de l'utilisateur, mais en revanche, ils sont spécifiques et ont une flexibilité réduite. La plupart des langages pilotés par les données, utilisent un calendrier d'événements qui stocke l'état du système à travers les valeurs associées aux attributs.

c. Les langages orientés objets :

Dans ces langages, les objets physiques et logiques du système réel (produits, ressources, gammes...) sont représentés par des objets informatiques. L'approche objet cherche à combler l'écart qui existe entre un modèle de simulation classique et le système à étudier. Parmi les langages orientés objets, on peut citer, SIMULA, CLOS, MODSIM.

Les figures II.8 et II.9 présentent un exemple qui montre la différence de modélisation entre l'approche orientée processus et l'approche orientée objet sur la base d'un système simple représenté par la figure II.7.

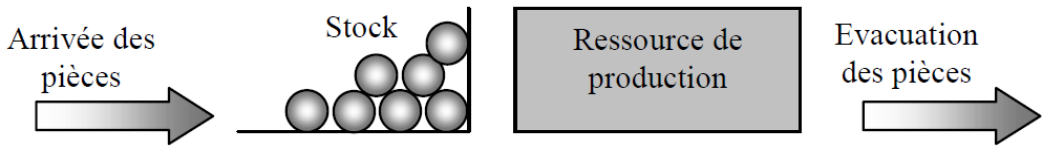


Figure II.7 : système à modéliser

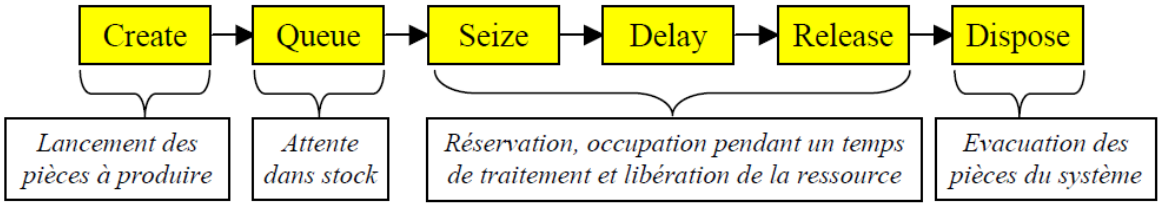


Figure II.8 : Modèle orienté processus (SIMAN)

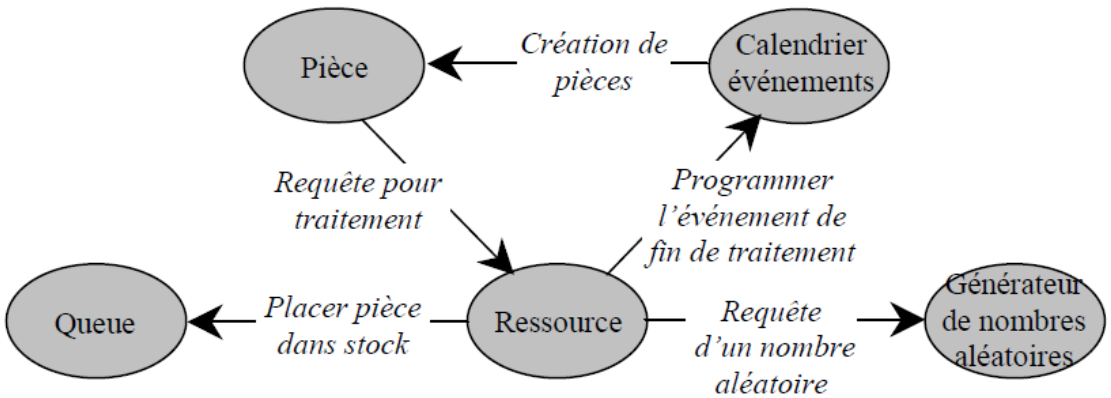


Figure II.8 : Modèle orienté objet inspiré de [Rob 98]

```
Dim oSIMAN As Arena.SIMAN
Dim  d(7, 75), m1(7, 75), m2(7, 75), m3(7, 75),
ma(10, 75), M(15, 75) As Double
Dim v(75), v1(75), v2(75), v3(75),  ve(75),
VBS(75)  As Double
Dim BS, SPT1, TNOW, x4, x5, x12, dn, BSP1,
BSP3, decalageglobal1As Double
Dim c, f, taille1, taille2,  taille3, compteur1,
compteur2, n, e, g, p, i, j, o, ntp, nbremach, As
Double

Private Sub ModelLogic_RunBeginSimulation()
Set oSIMAN = ThisDocument.Model.SIMAN
End Sub

Sub affectmachine(am, vv, ad, antp)
For R = 1 To antp
k = 1
For l = 1 To am
If vv(l) < vv(k) Then
 k = l
End If
 Next l
If ad(1, R) > vv(k) Then
 ad(4, R) = ad(1, R)
 vv(k) = ad(1, R) + ad(2, R)
 ad(5, R) = k
Else
 ad(4, R) = vv(k)
 vv(k) = vv(k) + ad(2, R)
 ad(5, R) = k
End If
Next R
End Sub

Sub matmach(ma, mi, mm, mee, md, mntp)
mi = mee
For l = 1 To mntp
If md(5, l) = ma Then
mi = mi + 1
mm(6, mi) = l
For k = 1 To 4
mm(k, mi) = md(k, l)
Next k
End If
Next l
For l = 1 To mi
For k = l To mi
If mm(1, l) = mm(1, k) And mm(3, l) > mm(3, k)
Then
For n = 1 To 6
yy = mm(n, l)
mm(n, l) = mm(n, k)
mm(n, k) = yy
Next n
End If
Next k
Next l
End Sub

Sub calcultbi(mi, mm, ce)
mm(7, mi) = mm(3, mi) - mm(2, mi)
For l = 1 To mi - ce - 1
n = mi - l
d1 = mm(3, n) - mm(2, n)
d2 = mm(7, n + 1) - mm(2, n)
If d1 <= d2 Then
mm(7, n) = d1
Else
mm(7, n) = d2
End If
Next l
End Sub

Function faisabilite(mi, mm, fe)
faisabilite = 0
For l = fe + 1 To mi
If mm(4, l) > mm(7, l) Then
faisabilite = 1
End If
Next l
End Function

Function calculBS(cm)
BS = VBS(1)
For l = 1 To cm
If VBS(l) > BS Then
BS = VBS(l)
End If
Next l
calculBS = BS
End Function

Sub affect(am)
For l = 1 To am
If v(l) = compteur2 Then
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("mach")) = l
End If
Next l
End Sub

Sub tachesuivante(mach, taille9, matrix)
If taille9 <> 0 Then
s = matrix(mach, 1)
For n = 1 To taille9
matrix(mach, n) = matrix(mach, n + 1)
Next n
matrix(mach, taille9 + 1) = s
End If
End Sub

Function decalageglobal(dnm)
dg = 0
For dgn = 1 To dnm
dg = M(dgn * 3 + 2, ma(dgn, 1)) + dg
Next dgn
decalageglobal = dg
End Function
```

```
Function taille(mac, matrix, AR)
For tl = 1 To AR + 1
If matrix(mac, tl) = 0 Then
taille9 = tl - 1
GoTo 20
End If
Next tl
20 taille = taille9
End Function

Sub tri(tnm)
BSP1 = 0
For tm = 1 To tnm
For tn = 1 To compteur1
If M(tm * 3 + 1, tn) <> 0 Then
ma(tm, tn) = tn
End If
Next tn
Next tm

For tm = 1 To tnm
For tn = 1 To compteur1
If ma(tm, tn) = 0 Then
For tk = tn To compteur1
ma(tm, tk) = ma(tm, tk + 1)
Next tk
End If
Next tn
Next tm

For n = 1 To nbremach
For l = 1 To compteur1
For k = l To compteur1
If M(1, ma(n, l)) > M(1, ma(n, k)) And M(1, ma(n,
k)) <> 0 And M(1, ma(n, l)) Then
yy = ma(n, l)
ma(n, l) = ma(n, k)
ma(n, k) = yy
Else
If M(1, ma(n, l)) = M(1, ma(n, k)) And M(2, ma(n,
l)) > M(2, ma(n, k)) And M(1, ma(n, k)) <> 0 And
M(1, ma(n, l)) Then
yy = ma(n, l)
ma(n, l) = ma(n, k)
ma(n, k) = yy
End If
End If
Next k
Next l
Next n
End Sub

Function verification(vnm, matrix)
verification = 0
For vl = 1 To vnm - 1
If matrix(vl, 1) <> 0 Then
For vk = vl + 1 To vnm
If matrix(vk, 1) = matrix(vl, 1) Then
verification = 1
End If
Next vk
End If
Next vl
End Function

Function nombreTAT(nnm)
nombreTAT = 0
For nn = 1 To nnm
If ma(nn, 1) <> 0 Then
nombreTAT = nombreTAT + 1
End If
Next nn
End Function

Function BSP(bnm)
BSP2 = 0
For bn = 1 To bnm
If ma(bn, 1) <> 0 Then
If BSP2 < M(bn * 3, ma(bn, 1)) Then
BSP2 = M(bn * 3, ma(bn, 1))
End If
End If
Next bn
BSP = BSP2
End Function

Function SPT(snm)
SPT = 0
For sn = 1 To snm
If ma(sn, 1) <> 0 Then
SPT = SPT + M(1, ma(sn, 1))
End If
Next sn
End Function

Function EDD(enm)
EDD = 0
For en = 1 To enm
If ma(en, 1) <> 0 Then
EDD = EDD + M(2, ma(en, 1))
End If
Next en
End Function

Sub affectcombine(anm, vv, maa)
For an = 1 To anm
vv(an) = maa(an, 1)
Next an
End Sub

Sub mat(mt, et, vt, xt, b, it)
pr =
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("pr"))
If xt <= TNOW Then
If et = it Then
If TNOW + pr <= BS Then
M(3 * b, compteur1) = BS
M(3 * b + 1, compteur1) = 1
M(3 * b + 2, compteur1) = 0
```

```
Else
M(3 * b, compteur1) = TNOW + pr
M(3 * b + 1, compteur1) = 2
M(3 * b + 2, compteur1) = 0
End If
Else
If TNOW + pr <= mt(4, et + 1) Then
M(3 * b, compteur1) = BS
M(3 * b + 1, compteur1) = 1
M(3 * b + 2, compteur1) = 0
Else
mt(2, et) = pr
vt(et) = TNOW
For l = et + 1 To it
vt(l) = mt(1, l)
If mt(1, l) < vt(l - 1) + mt(2, l - 1) Then
vt(l) = vt(l - 1) + mt(2, l - 1)
End If
Next l
If vt(it) + mt(2, it) <= BS And TNOW + pr <=
mt(7, et + 1) Then
M(3 * b, compteur1) = BS
M(3 * b + 1, compteur1) = 2
M(3 * b + 2, compteur1) = TNOW + pr - mt(4, et +
1)
Else
If TNOW + pr <= mt(7, et + 1) Then
M(3 * b, compteur1) = vt(it) + mt(2, it)
M(3 * b + 1, compteur1) = 3
M(3 * b + 2, compteur1) = TNOW + pr - mt(4, et +
1)
End If
End If
End If
End If
End If
End Sub

Sub test(tm)
If verification(tm, ma) = 0 Then
If BSP1 = 0 Then
NTAT = nombreTAT(tm)
BSP1 = BSP(tm)
SPT1 = SPT(tm)
EDD1 = EDD(tm)
decalageglobal1 = decalageglobal(tm)
 affectcombine tm, v, ma
Else
If NTAT < nombreTAT(tm) Then
NTAT = nombreTAT(tm)
BSP1 = BSP(tm)
SPT1 = SPT(tm)
decalageglobal1 = decalageglobal(tm)
 EDD1 = EDD(tm)
 affectcombine tm, v, ma
Else
If NTAT = nombreTAT(tm) Then
If BSP1 > BSP(tm) Then
NTAT = nombreTAT(tm)
BSP1 = BSP(tm)
SPT1 = SPT(tm)
EDD1 = EDD(tm)
decalageglobal1 = decalageglobal(tm)
 affectcombine tm, v, ma
Else
If BSP1 = BSP(tm) Then
If decalageglobal(tm) < decalageglobal1 Then
SPT1 = SPT(tm)
EDD1 = EDD(tm)
decalageglobal1 = decalageglobal(tm)
affectcombine tm, v, ma
Else
If decalageglobal(tm) = decalageglobal1 Then
If SPT(tm) < SPT1 Then
SPT1 = SPT(tm)
EDD1 = EDD(tm)
decalageglobal1 = decalageglobal(tm)
affectcombine tm, v, ma
Else
If SPT1 = SPT(tm) Then
If EDD(tm) < EDD1 Then
EDD1 = EDD(tm)
affectcombine tm, v, ma
decalageglobal1 = decalageglobal(tm)
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End If
End Sub

'Ordonnancement des tâches programmables

Private Sub VBA_Block_1_Fire()
ntp = 75
nbremach = 2

'Lecture des parametres
c = c + 1
d(1, c) =
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("r"))
d(2, c) =
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("pr"))
d(3, c) =
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("dl"))

'Affectation et ordonnancement des taches
programmables
If c = ntp Then
```

```
'calcul des ri et des di
d(3, 1) = d(1, 1) + d(2, 1) + d(3, 1) 'd1
For l = 2 To ntp
d(1, l) = d(1, l - 1) + d(1, l) 'ri
d(3, l) = d(1, l) + d(2, l) + d(3, l) 'di
 Next l

'Affectation aux machines
affectmachine nbremach, ve, d, ntp

'Séparation de la matrice
matmach 1, i, m1, 0, d, ntp
matmach 2, j, m2, 0, d, ntp
matmach 3, o, m3, 0, d, ntp

'calcul des Tbi
calcultbi i, m1, e
calcultbi j, m2, g
calcultbi o, m3, p

 'Vérification de la faisabilité de
l'ordonnancement
fsa = 0
If faisabilite(i, m1, e) = 1 Or faisabilite(j, m2, g) = 1
Or faisabilite(o, m3, p) = 1 Then
fsa = 1
End If
If fsa = 1 Then
MsgBox "L'ordonnancement des tâches
programmables ne respecte pas les délais"
End If

'Calcul de la borne supérieure de
l'ordonnancement statique
VBS(1) = m1(4, i) + m1(2, i)
VBS(2) = m2(4, j) + m2(2, j)
VBS(3) = m3(4, o) + m3(2, o)

oSIMAN.VariableArrayValue(oSIMAN.SymbolNu
mber("Makespan")) = calculBS(nbremach)

  'Calcul des durées entre créations selon les Ri
d(7, 1) = d(1, 1)
For l = 2 To ntp
d(7, l) = d(1, l) - d(1, l - 1)
Next l

  'Calcul du plus grand délai
dn = d(3, 1)
For l = 1 To ntp
If d(3, l) > dn Then
dn = d(3, l)
End If
Next l
End If
End Sub

‘Ordonnancement des tâches aléatoires

Private Sub VBA_Block_37_Fire()
compteur1 = compteur1 + 1
TNOW =
oSIMAN.VariableArrayValue(oSIMAN.SymbolNu
mber("X7"))
pr =
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("pr"))
dl =
oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("dl"))
x4 =
oSIMAN.VariableArrayValue(oSIMAN.SymbolNu
mber("X4"))
x5 =
oSIMAN.VariableArrayValue(oSIMAN.SymbolNu
mber("X5"))
x12 =
oSIMAN.VariableArrayValue(oSIMAN.SymbolNu
mber("X12"))

'Condition d'arrêt
If TNOW >= dn Then
  GoTo 1
End If

'Tâches dépassant leurs délais
If TNOW + pr > dl Then
  oSIMAN.EntityAttribute(oSIMAN.ActiveEntity,
oSIMAN.SymbolNumber("mach")) = 9
  GoTo 1
End If

'Remplissage de la matrice de données
M(1, compteur1) = pr
M(2, compteur1) = dl

 'machine1
 mat m1, e, v1, x4, 1, i

 'machine2
 mat m2, g, v2, x5, 2, j

 'machine3
 mat m3, p, v3, x12, 3, o

1  End Sub

Private Sub VBA_Block_4_Fire()
compteur2 = compteur2 + 1
If compteur2 = 1 Then

  'Initialisation
tri (nbremach)
taille1 = taille(1, ma, compteur1)
taille2 = taille(2, ma, compteur1)
taille3 = taille(3, ma, compteur1)
For l = 1 To nbremach
v(l) = 0
Next l
```

```
'Négociation inter machines de la meilleure
combinaison des tâches aléatoires à traitées

2 'MACHINE1
GoTo 8 'machine2
3 'mach1
If ma(1, 1) = 0 Then
GoTo 16 'affectation des tâches aux machines
End If
tachesuivante 1, taille1, ma

4 'MACHINE2
GoTo 8 'TEST
5 'mach2
If ma(2, 1) = 0 Then
tachesuivante 2, taille2, ma
GoTo 3 'mach1
End If
tachesuivante 2, taille2, ma

6  'MACHINE3
GoTo 8 'TEST
7 'mach3
If ma(3, 1) = 0 Then
tachesuivante 3, taille3, ma
GoTo 5 'mach2
End If
tachesuivante 3, taille3, ma
8  'TEST
test (nbremach)
GoTo 7 'mach3
End If
16 'Affectation
affect (nbremach)
VBS(1) = m1(4, i) + m1(2, i)
VBS(2) = m2(4, j) + m2(2, j)
VBS(3) = m3(4, o) + m3(2, o)

oSIMAN.VariableArrayValue(oSIMAN.SymbolNu
mber("Makespan")) = calculBS(nbremach)

'Réinitialisation
If compteur1 = compteur2 Then
For k = 1 To nbremach
For l = 1 To 20
ma(k, l) = 0
M(1, l) = 0
M(2, l) = 0
M(k * 3, l) = 0
M(k * 3 + 1, l) = 0
M(k * 3 + 2, l) = 0
Next l
Next k
compteur1 = 0
compteur2 = 0
End If
End Sub
```