وزارة التعليم العالي و البحث العلمي

BADJI MOKHTAR-ANNABA UNIVERSITY
UNIVERSITE BADJI MOKHTAR ANNABA

جامعة باجي مختار- عنابة

Faculté Des Sciences de l'Ingénieur
Département d'Informatique

Année 2009/2010

MÉMOIRE

Présenté en vue de l'obtention du Diplôme de **MAGISTER**

Partitionnement Matériel/Logiciel et Systèmes MultiAgents

Application aux systèmes embarqués

Option

Informatique industrielle

Par : ZENAKHRA Djamel

DIRECTEUR DE MEMOIRE : Rachid BOUDOUR, Maître de conférence A, Univ. Annaba

DEVANT LE JURY

PRESIDENT :	Mohammed Tahar KIMOUR	M.C.A	Université d'Annaba
EXAMINATEURS :	Habiba BELLEILI	M.C.A	Université d'Annaba
	Yamina TLILI	M.C.A	Université d'Annaba

Résumé

Le partitionnement logiciel/matériel demeure une étape critique dans le processus du Co-design. De nombreux algorithmes et techniques ont été proposés pour aider le concepteur dans cette fastidieuse tâche, mais chacune d'entre elles présente des carences.

Le but de ce mémoire est de proposer une méthode originale qui fournit une solution meilleure du point de vue qualité et temps d'exécution en utilisant les Systèmes Multi Agents ou SMA. Une implémentation du système *SPMA* a été faite dans l'environnement MadKit avec des résultats prometteurs.

Notre travail peut être étendu de plusieurs façons : une combinaison des heuristiques avec les SMA pour améliorer davantage la qualité de la solution.

Mots clés : *Co-design, Partitionnement automatiques M/L, SMA, MadKit, systèmes embarqués.*

Abstract

The hardware / software partitioning remains a critical step in the process of co-design. Many algorithms and techniques have been proposed to assist the designer in this tedious task, but each has shortcomings.

The purpose of this paper is to propose a novel method that provides a better solution in terms of quality and execution time using the Multi Agent Systems or SMA. An implementation of the AMP system was made in the environment MadKit with promising results.

Our work can be extended in several ways: a combination of heuristics with ADM to further improve the quality of the solution.

Keywords: *Co-design, Automatic H/S Partitioning, MAS, MadKit, embedded systems.*

ملخص

تقسيم / سوفت وير الأجهزة يبقى خطوة حاسمة في عملية التصميم المشترك .وقد اقترح العديد من الخوارزميات والتقنيات لمساعدة المصمم في هذه المهمة شاقة ، ولكن كل نقائص وعيوب .

والغرض من هذه الورقة هو اقتراح طريقة الرواية التي توفر حلا أفضل من حيث الوقت ونوعية التنفيذ باستخدام عامل متعدد النظم أو إس إم .تم إجراء تنفيذ نظام امبير في MadKit البيئة مع نتائج واعدة.

ويمكن تمديد العمل لدينا في عدة طرق: مزيج من الاستدلال مع شعبة الشؤون الإدارية لزيادة تحسين نوعية الحل .

كلمات: شارك في تصميم والتقسيم التلقائي م / ل MadKit MAS ، ونظم المضمنة.

Sommaire

Liste des figures

Liste des tableaux

Introduction générale

1 Motivation

La conception de système contenant des composants matériels et logiciels n'est pas un problème nouveau. Les approches traditionnelles de conception consistent à traiter le matériel avant de passer au développement des composants logiciels. Récemment encore, le développement des systèmes informatiques a été caractérisé par le fait que les ingénieurs matériels fournissent des systèmes à usage général, programmés ensuite par les ingénieurs du logiciel. Même si l'utilisation des microprocesseurs durant les années 80 a permis une mise en œuvre d'application à dominante logiciel plutôt que matériel, les idées n'ont pas beaucoup changé. Les mises en œuvre du logiciel et du matériel sont deux activités qui ont toujours été menées de manière relativement indépendante.

Les heuristiques modernes retournent des bons résultats pour des systèmes d'une certaine taille, mais pour des systèmes d'une grande taille où l'espace de solutions est trop grand, et même si le résultat retourné par les heuristiques classiques sont de bonne qualité, la possibilité d'avoir des solutions meilleures est toujours possible, vu que l'espace de solutions n'est pas suffisamment parcouru. Par ailleurs, la résolution coopérative de problèmes prend une ampleur importante en intelligence artificielle distribuée (IAD). Un domaine d'actualité de recherche relativement complexe, dérivé de l'IA est celui des systèmes muli-agents (SMA). Les SMA sont une meilleure réponse à ce problème, parce que le paradigme SMA offre la possibilité de faire des recherches parallèles dans l'espace de solutions.

2 Problématique

Le partitionnement logiciel/ matériel est une des phases clés de cette conception, c'est dans cette phase, que sont effectués les choix menant à une réalisation soit matérielle soit logicielle des différentes parties constituant le système. De nombreuses techniques et algorithmes ont été proposés pour approcher ce problème, car ce dernier est un problème NP-Complet.

Les choix effectués dans le partitionnement sont des choix définitifs qui ne sont plus remis en cause dans les phases suivantes de synthèse. Un mauvais choix à ce niveau impose alors de recommencer le cycle de conception du partitionnement jusqu'à la simulation. Il faut donc prêter une grande attention à cette étape. Vu que les systèmes modernes sont des systèmes en évolution constante alors en complexité constante.

Dans ce sillage, notre travail porte sur la modélisation du problème de partitionnement M/L à l'aide des SMA en vue de fournir une qualité meilleure de solution.

3 Contributions

Notre contribution peut être perçue dans les deux points suivants :

- Une modélisation nouvelle et originale de ce problème à l'aide des SMA, pour exploiter au mieux la puissance de ce paradigme dans la résolution des problèmes NP-Complets.
- Une implémentation de notre démarche en utilisant un environnement dédié Multi agents (*MadKit)*.

4 Structure du mémoire

Le mémoire et structuré en deux parties :

La première traite de l'état de l'art. Elle est composée de trois chapitres dont le premier aborde le problème de partitionnement matériel/logiciel dans tous ses états, le second décrit le paradigme des SMA, et le troisième présente les approches possibles de modélisation de ce problème.

La deuxième partie comporte deux chapitres. Elle met en avant dans le chapitre 1, l'approche choisie avec une description détaillée des différentes étapes et concepts, suivie en chapitre 2 de l'implémentation de notre système SPMA avec une illustration sur un exemple.

Le mémoire s'achève par une conclusion et de nouvelles directions de travail.

Première Partie : État de l’art

Chapitre 1 : Le partitionnement M/L, Méthodes et outils

Dans une stratégie de conception conjointe, la co-spécification a pour but de permettre au concepteur d'effectuer la spécification d'un système sans se soucier du découpage sur des architectures hétérogènes logicielles et matérielles. Cette spécification abstraite (sans choix d'implantation) est ensuite raffinée (choix d'algorithmes particuliers, choix des représentations des structures de données, ...) pour aboutir à une spécification dans laquelle il est possible d'identifier les fonctionnalités à réaliser. Ces fonctionnalités de base coïncident le plus souvent avec les objets de plus bas niveau spécifié dans la spécification la plus raffinée. L'étape suivante consiste à rechercher pour chaque fonctionnalité une réalisation soit logicielle soit matérielle: c'est l'étape de partitionnement logiciel/matériel. L'objectif du partitionnement est ici de trouver le meilleur compromis entre les parties logicielles et matérielles en fonction de leurs interactions et des contraintes dictées notamment par des critères de performances et de réutilisation.

La complexité croissante des systèmes pour lesquels la réalisation résulte de l'association d'une partie matérielle et d'une partie logicielle, la diversité des choix technologiques et les contraintes de coûts et délais de plus en plus sévères nécessitent l'utilisation de nouvelles méthodologies et outils logiciels associés pour diminuer leur durée de conception et accroître leur qualité.

Le développement des systèmes électroniques composés d'une partie matérielle et d'une

partie logicielle n'est pas un problème nouveau. Généralement, concevoir et réaliser de tels systèmes nécessite une compétence technique dans au moins 3 domaines: l'électronique analogique, l'électronique numérique et l'informatique. La nature spécifique du traitement à effectuer et le couplage du système avec son environnement nécessitent aussi des compétences complémentaires: en traitement de l'information (signal, image, parole...), en électronique de puissance lorsque l'environnement utilise des courants forts, en réseaux et télécommunications, etc.

1 Le partitionnement

Le problème du partitionnement matériel/logiciel est au cœur de l'activité de co-design. Le choix de l'architecture matérielle est un élément de décision essentiel et la démarche diffère selon que l'architecture se trouve imposée ou choisie d'emblée ou que l'architecture et les composants de celle-ci sont à déterminer. La première situation est la plus commune et la plus simple. L'architecture matérielle est généralement une architecture générique constituée d'un microprocesseur, d'un ensemble de circuits matériels programmables ou d'ASICs et d'une mémoire commune. Le problème du partitionnement se réduit alors à un problème de partitionnement binaire matériel/logiciel pour l'allocation des éléments fonctionnels sur les constituants de l'architecture et peut se résoudre de manière automatique. La deuxième situation plus complexe. Dans ce cas, face à la nature hétérogène de l'architecture cible et à la diversité des contraintes imposées, une démarche itérative et guidée par le concepteur s'impose. Il s'agit alors d'offrir au concepteur des moyens rapides d'estimation des propriétés de l'implantation résultant du choix de l'architecture, du partitionnement et de l'allocation pour vérifier si celles-ci répondent aux contraintes imposées. Pour les parties du système relevant du co-design, les contraintes imposées sont surtout des contraintes de performances. Les contraintes de performances sont de nature statique ou dynamique. L'estimation des performances statiques telles que la surface de silicium occupée, la puissance consommée repose sur des techniques de synthèse. La plupart des travaux de la communauté du co-design sur l'estimation des performances dynamiques d'un partitionnement, sont basés sur une analyse des contraintes temporelles et un calcul de la charge du processeur par des techniques proches de celles utilisées en ordonnancement de tâches pour des systèmes temps-réels.

2 Problématique du partitionnement M/L

Le problème du partitionnement se pose dès que nous disposons de plus d'un composant pour réaliser une tâche. Le problème est de déterminer « qui » réalise chaque tâche (ou

fonctionnalité). C'est un des problèmes classiques de l'allocation des ressources. Or, les contraintes liées au partage des ressources dépendent de l'ordonnancement des tâches. En effet, le parallélisme d'action dépend du nombre de ressources disponibles sur l'architecture. Autrement dit, pour répondre correctement au « qui » il faut répondre au « quand » et réciproquement. Il est donc nécessaire d'aborder la problématique dans son ensemble.

Précisément, pour des systèmes embarqués orientés traitement de données, la problématique est pour chaque tâche

- De déterminer sa date d'exécution (ordonnancement),
- De choisir une réalisation logicielle ou matérielle (partitionnement),
- De définir le nombre et le type de ressources pour implanter le système (allocation).

L'ordonnancement, *le partitionnement* et *l'allocation* sont trois problèmes interdépendants. L'idéal serait de les réaliser simultanément. Cependant, pour échapper à la trop grande complexité du problème général, il faut réaliser des approximations guidées par des choix locaux. Ainsi dans la majorité des cas, l'allocation, le partitionnement et l'ordonnancement sont réalisés séparément. Pour renseigner le concepteur sur la qualité des solutions trouvées, on utilise des estimateurs (estimations des performances, des coûts, .. .) afin de prédire les résultats de la conception sans aller jusqu'à la réalisation du système.

Le problème du partitionnement logiciel/matériel est approché de nombreuses façons suivant les modèles d'application et d'architecture considérés. Il ne semble pas envisageable actuellement de proposer une méthode de partitionnement générale, en particulier du fait de l'absence d'un modèle ou d'un environnement de modélisation de sémantique bien défini pour décrire les différents types de comportement d'une application (contrôle, traitements numériques et en temps continu par exemple) [De Kock, 2000]. Aussi, pour obtenir une approche efficace, il est nécessaire d'identifier une classe d'applications et un modèle d'architecture adapté qu'il s'agit de paramétrer de manière efficace en fonction des caractéristiques de l'application cible, pour obtenir une solution optimisée par rapport aux critères choisis.

3 Les types du partitionnement

Il y a principalement deux types de partitionnement automatique et interactif. Le partitionnement automatique (et c'est le type qui nous intéresse dans ce papier) repose sur une architecture cible imposée et mono-processeur, une heuristique et l'utilisation d'une fonction de coût dont les coefficients de pondération dépendent de critères tels que consommation, surface de silicium, coûts, taille du code, taille de la mémoire. Le partitionnement interactif

cible généralement une architecture hétérogène à définir et s'appuie sur des estimateurs de performances statiques et/ou une estimation des performances dynamiques du système pour guider le concepteur dans le choix d'une répartition.
Les techniques de partitionnement décrites dans la littérature peuvent être classées par:

- leur degré d'automatisation allant d'une démarche manuelle à une démarche entièrement automatique,
- Les critères influençant le choix d'un partitionnement (contraintes statiques ou dynamiques, sûreté de fonctionnement, flexibilité, testabilité, coûts),
- le choix de l'architecture cible figée ou libre,
- le degré d'abstraction du modèle représentant les éléments de la spécification du système à partitionner et de l'architecture matérielle allant d'une modélisation macroscopique à une modélisation architecturale détaillée.

Pour les spécifications d'entrée d'un partitionnement, trois niveaux de granularité du partitionnement sont habituellement utilisés: le niveau tâche, le niveau procédure et le niveau instruction. Pour le niveau tâche (coarse-grain partitioning), l'unité d'allocation est la fonction qui est considérée indivisible et dont le comportement n'est pas obligatoirement séquentiel.
Pour le niveau procédure, une fonction est décomposée en un ensemble de séquences d'instructions appelées procédures et qui peuvent être allouées sur des processeurs différents.
Pour le niveau instruction (fine-grain partitioning), l'unité d'allocation est la plus petit possible puisqu'il s'agit d'une instruction. L'utilisation d'un niveau de granularité fine concerne plutôt des systèmes de faible complexité ou une conception architecturale avancée qui se situe relativement tard dans le cycle de développement.

3.1 Le partitionnement automatique

Le problème du partitionnement est souvent présenté comme un problème NP complexe dépendant d'un grand nombre de paramètres. Pour résoudre ce problème, la plupart des méthodes automatiques réduit le nombre des paramètres (prise en compte d'un nombre limité de critères) et utilise une heuristique basée sur une fonction de coût pondérée par les critères retenus.

Actuellement de nombreuses heuristiques de partitionnement ont été développées. On peut citer:

- L'algorithme gourmand (Greedy algorithm) utilisé dans VULCAN [Gupta, 94] pour lequel toutes les fonctions sont initialement implantées en matériel et sont migrées

vers le processeur logiciel en vérifiant les contraintes temporelles. La fonction de coût utilisée est pondérée par la surface du matériel, la taille du programme de code (mémoire) et le taux d'occupation du processeur.

- l'approche de COSYMA [Ernst, 93] est opposée à celle utilisée dans VULCAN. Les fonctions sont au départ toutes implantées en logiciel puis migrées vers le matériel jusqu'au respect des contraintes de performances.
- Les algorithmes utilisés dans Co-Saw [ADAMS-95] et dans SpecSyn [Vahid, 95] sont basés sur la construction progressive de groupes de fonctions (clustering based algorithm) pouvant partager la même ressource matérielle ou logicielle.

Les algorithmes cités précédemment dépendent fortement (coefficient de pondération de la fonction de coût) des caractéristiques de l'architecture cible choisie. Souvent l'architecture cible est composée d'un seul processeur logiciel couplé à un ou plusieurs FPGA ou ASICs et éventuellement une mémoire commune. Peu de techniques de partitionnement ciblent vers une architecture hétérogène composée d'un ensemble de processeurs logiciels (microprocesseur, DSP, ASIP) et matériels (FPGA, ASIC). Hou [HOU-96] propose cependant une heuristique basée sur la construction progressive de groupes de process (clustering based algorithm) dont la fonction de coût dépend de la communication inter-processeurs, des temps de commutation de contexte (approche préemptive et non synchrone) et du taux d'utilisation des ressources. Les résultats obtenus sont très dépendants des coefficients de la fonction de coût: "the cost function plays an important role in our partitioning approach" [Ernest, 93].

3.2 Le partitionnement interactif

Ismail [Ismail, 94] propose une technique de partitionnement interactive (PARTIF) permettant de cibler une architecture hétérogène. Le concepteur peut aisément explorer plusieurs alternatives de partitionnement du système en manipulant une hiérarchie d'automates à états finis concurrents représentée selon le formalisme SOLAR. Un ensemble de primitives de transformations d'états (déplacement, regroupement, décomposition,...) est disponible. Cette approche est intéressante mais pour l'instant le concepteur dispose de peu de résultats quantitatifs en retour pour évaluer la partie logicielle (statistiques sur les interconnexions et les variables partagées) et la partie matérielle (statistiques sur le nombre d'états et sur le nombre d'opérateurs de la partie opérative) obtenues de manière à les comparer aux contraintes imposées.

4 Méthodes de partitionnement

Le partitionnement fait l'objet de nombreux travaux [IEEE, 1997], qui peuvent être classés en fonction des critères suivants :

- Les étapes de partitionnement, d'ordonnancement et d'allocation sont des étapes interdépendantes. L'approche idéale consiste à réaliser ces trois étapes simultanément. Cependant la taille des graphes est alors limitée par la complexité du partitionnement. Pour les graphes de grande taille (**de l'ordre d'une centaine de nœuds**), la méthodologie généralement utilisée se caractérise par l'utilisation de méthodes d'optimisation ainsi que par l'emploi, dans chacune des étapes, de termes prédicateurs sur les résultats des autres étapes. Ces termes permettent de traiter séparément chacune de ces étapes interdépendantes.
- Les niveaux de granularité (qui correspondent à la finesse de description) varient du niveau instruction où groupe d'instructions, au niveau bloc système. La problématique est ici de trouver un compromis entre la taille de la granularité et le nombre de grains à traiter pour éviter une explosion combinatoire. Une meilleure approche serait évidemment une granularité variable, hiérarchique qui permettrait de parcourir entièrement l'espace de conception. Cependant une telle approche entraînerait une complexité impossible à traiter. L'expérience du concepteur est donc pour l'instant grandement sollicitée, afin qu'il modélise le système au niveau de granularité qui lui semble le plus approprié en fonction des bibliothèques qu'il utilise.
- Les principaux objectifs du partitionnement sont la minimisation du coût (surface, consommation, ...) et la durée d'exécution du système. Un compromis entre ces deux minimisations est généralement recherché.
- Le type d'application (orienté contrôle ou données) est également une caractéristique importante, car les applications flot de données permettent un ordonnancement statique alors que des applications orientées contrôle nécessitent un ordonnancement dynamique (la date de début d'exécution de certaines tâches est en effet indéterminée).
- Les types de communication utilisés et le degré de prise en compte des contraintes des communications lors du partitionnement sont également des critères de différenciation des approches existantes. La conception des communications est composée de deux étapes. La première étape consiste à déterminer l'architecture de communication la plus appropriée pour le transfert des données. Pour définir cette architecture, il faut déterminer le canal de transmission, le protocole et l'unité réalisant le transfert.

- Le problème des communications se pose donc en termes de choix de ressources logicielles ou matériels. La seconde étape consiste à implanter l'architecture qui a été définie lors de la première étape. Cette étape est généralement appelée synthèse de l'interface.
- Dans le cadre de la conception conjointe, l'architecture cible définit le support du système. Elle peut être déduite après le partitionnement par l'analyse des besoins. Pour des applications spécifiques, notamment en traitement du signal, le choix du processeur fait partie intégrante du partitionnement et des choix du matériel à effectuer. L'architecture cible peut être imposée avant le partitionnement si le concepteur possède un modèle d'architecture qu'il pense être efficace. C'est une étape de pré partitionnement qui restreint les choix de réalisations possibles.

Le tableau 2.1 permet une comparaison d'un certain nombre de méthodes en fonction des critères utilisés. Il est intéressant de noter l'hétérogénéité de ces différentes approches, qui empêche une réelle comparaison qualitative.

Auteur	Modèle	Fonction du coût	Algorithme	Communication	Architecture cible
Gupta	CDFG	Temps	Heuristique	Bus, mémoire	Processus + ASIC (s)
Kumar	Set-Based	Profiling	Programmation linéaire	?	?
Chou	Diagramme de temps	Temps, surface	Min-cut	Synthèse des protocoles	Micro-contrôleurs
Kalavade	Acyclique DFG	Temps	Heuristique par liste	bus	DSP + ASIC (s)
Henkel	CDFG	Profiling, synthèse	Simulation annealing	Mémoire partagé	Processeur + coprocesseur
Vahid	Acyclique DFG	Profiling, communication	ILP	?	Processeurs + ASIC(s)
Barros	HDL (unity)	Affinité	Clustering	Mémoire partagé	Processeurs + ASIC(s)
Ben Ismail	Processus communicant	Résultat de simulation	Hand	Bus, FIFO	Processeurs + ASIC + FPGA + ASIP
Freund	Acyclique DFG	Surface de communication	Heuristiqe, exhaustif	Bus, FIFO, DMA	Processeur + ASI

Tableau 1.41 : Comparaison des méthodes de partitionnement

5 Principaux algorithmes de partitionnement

5.1 Les colonies de fourmis

5.1.1 Les colonies de fourmis dans la nature

En marchant du nid à la source de nourriture et vice-versa (ce qui dans un premier temps se fait essentiellement de façon aléatoire), les fourmis déposent au passage sur le sol une substance odorante appelée phéromones. Cette substance permet ainsi donc de créer une piste chimique, sur laquelle les fourmis s'y retrouvent. En effet, d'autres fourmis peuvent détecter les phéromones grâce à des capteurs sur leurs antennes [Constanzo, 2006].

5.1.2 Meta-heuristique d'optimisation par colonie de fourmis

Aussi connu sous le nom de ACO (Ant Colony Optimization), est proposé par Dorigo and al. [Dorigo, 1992], pour résoudre le problème de voyage de commerce, et il est inspiré par le comportement naturel des fourmis pour trouver le chemin entre leur nid et la nourriture. En se basant sur ce principe plusieurs variantes de résolution de problème d'optimisation ont été développées [Costa, 1997] [Solnon, 2000] [Benatchba, 2004] [Admane, 2004] .

Le ACO utilise trois mécanismes qui sont : activité de fourmis, évaporation de phéromone et des actions d'un superviseur en arrière plan.

Activité de fourmis : chaque fourmi construit un chemin en parcourant l'espace de recherche, qui est souvent représenté par un graphe. Au début, les fourmis sont placées aléatoirement dans les extrémités du graphe (état initial). Puis elles se déplacent dans leur voisinage pour construire leur solution. Le choix du sommet suivant à visiter parmi un certain nombre de voisins est fait suivant une recherche locale. Le choix est basé sur des informations locales de la fourmi, ainsi que les traces de phéromone, et autres constantes liées au problème à résoudre.

Évaporation des phéromones : La méta-heuristique ACO comprend aussi la possibilité d'évaporation des phéromones. Ce mécanisme permet d'oublier lentement ce qui s'est passé avant. C'est ainsi qu'elle peut diriger sa recherche vers de nouvelles directions, sans être trop contrainte par ses anciennes décisions.

Le superviseur : pour améliorer les performances du ACO, un composant superviseur est souvent ajouté (appelé aussi Damon). Il a une vision globale de l'état de la recherche. Il peut influencer la recherche par jeter plus de phéromone sur un chemin, ou le mettre sur des chemins nouveaux.

Ce qui suit représente la procédure de ACO [Koudil, 2005].

```
PROCEDURE ACO Meta heuristic()
 While (not stopping criterion) do
  Program activities:
   Ant activity;
   Pheromone Evaporation;
   Demon actions optional;
  End Program activities
 End do
End.
```

5.2 Algorithme Génétique

Les algorithmes génétiques, qui rentrent dans le cadre des Algorithmes Evolutifs (AE), ont été développés au départ par John Holland en 1975 [Holland, 1975], et ont été depuis utilisés avec succès pour résoudre plusieurs problèmes dans le domaine des systèmes VLSI comme le placement/routage, l'optimisation de code pour les DSP ou la génération de tests fonctionnels. Ces algorithmes ont montré de très bonnes performances dans la résolution de problèmes sur lesquels peu d'informations sont disponibles ou pour lesquels on doit considérer de multiples critères d'optimisation. Les concepteurs doivent souvent optimiser des critères différents comme le temps d'exécution, la surface, le débit des données, le coût, la consommation, etc. Les Algorithmes Evolutifs en général sont une alternative intéressante par rapport aux autres approches d'optimisation puisqu'ils peuvent être adaptés rapidement à la taille du problème et intégrer de nouveaux critères en changeant seulement leurs fonctions de coût.

Certains principes de l'évolution ont été soulignés par Darwin sous la maxime «les plus adaptés survivent». Le terme d'adaptation correspond à l'aptitude d'un individu à vivre (et se reproduire) dans le milieu où il se trouve. C'est le fondement du calcul évolutif.

Mais au delà de la survie d'un individu se pose le problème de la survie de son espèce. Là encore les progrès de la biologie nous ont permis de comprendre comment une espèce évoluait au fil des générations : à la sélection Darwinienne s'ajoute un autre phénomène : l'évolution génétique. La génétique biologique est une science trop vaste pour que nous entrions dans ses détails, mais elle indique que le patrimoine génétique (i.e l'architecture matérielle d'un individu) évolue au cours du temps, menant à des représentants de l'espèce toujours plus adaptés à leur milieu. S'inspirant de ce phénomène, J. Holland a imaginé au début des années 70 le principe des algorithmes génétiques. Tout comme l'ADN représente un codage de la construction d'un individu chez les êtres vivants, dans les algorithmes génétiques les solutions possibles sont codées dans l'espace de recherche, et l'évolution des individus (uniquement représentés par leur génotype) dans un milieu artificiel mène à

l'amélioration des performances de ces individus. Ainsi, il devient possible d'optimiser tout problème pour lequel un codage des solutions et une fonction d'évaluation sont constructibles.

Un algorithme génétique se base sur une procédure itérative durant laquelle un ensemble de générations sont créées, une génération par itération. L'entière population évolue simultanément de façon à ce que la probabilité de convergence vers un minimum local soit réduite.

5.3 Recuit simulé

La méthode du recuit simulé tire son nom et son inspiration de la physique des matériaux et plus spécialement de la métallurgie. Le recuit est une opération consistant à laisser refroidir lentement un métal pour améliorer ses qualités. L'idée physique est qu'un refroidissement trop brutal peut bloquer le métal dans un état peu favorable (alors qu'un refroidissement lent permettra aux molécules de s'agencer au mieux dans une configuration stable). C'est cette même idée qui est à la base du recuit simulé. Pour éviter que l'algorithme ne reste piégé dans des minima locaux, on fait en sorte que la température T = T(n) décroisse lentement en fonction du temps.

Le processus physique décrit ci-dessus a été associé à des problèmes d'optimisation combinatoire dans [Kirkpatrick, 1983].

Partitionnement avec le recuit simulé :

Dans ce qui suit nous donnons une brève description de l'algorithme. Avec X nous noterons une solution composée de deux ensembles HW et SW. x^{now} représente la solution en cours et N (x^{now}) désigne le quartier de x^{now} dans l'espace des solutions.

Step 1: Construct initial configuration $x^{now} := (Hw_0, Sw_0)$

Step 2: Initialize Temperature $T := TI$

Step 3: 3.1. *for i:=1 to TL do*

Generate randomly a neighboring solution $x' \in N(x^{now})$

Compute change of cost function $\Delta C := C(x') - C(x^{now})$

If $\Delta C \leq 0$ *then* $x^{now} := x'$

else

Generate *q:=random(0,1)*

if $q < e^{-\Delta C/T}$ *then* $x^{now} :=$

3.2. Set new temperature $T := \alpha * T$

Step 4: *if* stopping criterium not met *then goto* Step 3

Step 5: *return* solution corresponding to the minimum cost function

Recuit simulé sélectionne la solution voisine de manière aléatoire et accepte toujours une solution améliorée. Il accepte également des mauvaises solutions avec une certaine probabilité, qui dépend de la détérioration de la fonction de coût et sur un paramètre de contrôle appelé la température [Kirkpatrick, 1983]. L'algorithme qui suit représente le déplacement dans le recuit simulé :

Algorithm SIMULATED-ANNEALING
Begin
temp = INIT-TEMP;
place = INIT-PLACEMENT;
while (*temp* > FINAL-TEMP) **do**
while (*inner_loop_criterion* = FALSE) **do**
new_place = PERTURB(*place*);
ΔC = COST(*new_place*) - COST(*place*);
if (ΔC < 0) **then**
place = *new_place*;
else if (RANDOM(0,1) > $e^{-(\Delta C/temp)}$) **then**
place = *new_place*;
temp = SCHEDULE(*temp*);
End.

6 L'analyse des propriétés d'un partitionnement

Pour analyser les propriétés d'un partitionnement, la plupart des techniques de partitionnement utilisent des estimateurs basés sur:

- une analyse des contraintes temporelles, ce qui nécessite de représenter le comportement des fonctions à un niveau interprété et très détaillé. Généralement, le comportement d'une fonction est représenté sous la forme d'un flot de données et d'un flot de contrôle (CDFG). L'analyse des graphes [Gupta, 94] permet d'extraire une approximation du temps d'exécution de chaque fonction et de vérifier le respect des contraintes temporelles. Puis pour une implantation logicielle de la fonction, on

applique les algorithmes utilisés dans les problèmes d'ordonnancement des systèmes temps réels pour calculer la charge du processeur. Formulé sous sa forme la plus simple (mono-processeur et tâches périodiques), le problème est résolu souvent par l'algorithme de base RMS "Rate Monotonic Scheduling". Le problème se complique lorsque les tâches peuvent être sporadiques ou lorsque le système est distribué.

- une analyse statique basée sur des résultats de techniques de synthèse qui nécessite une description des fonctions au moins au niveau algorithmique (synthèse haut niveau) pour extraire des caractéristiques telles que la surface de silicium occupée, la puissance dissipée, le nombre de broches, la taille du programme code, la taille de la mémoire nécessaire, etc.

Très peu de techniques de partitionnement sont basées sur un modèle plus abstrait et non-interprété d'un système. Ambrosio propose une technique de partitionnement basée sur un modèle "système". Les processeurs sont représentés par des ressources caractérisées par une taille mémoire et le temps d'exécution d'une instruction. La partie logicielle est représentée par des tâches utilisant une quantité de mémoire et un nombre d'instructions donné. Romdhani propose une méthode de partitionnement matériel/logiciel où l'évaluation des performances dynamiques du système est effectuée avec l'outil SES/Workbench.

Chapitre 2 : Les systèmes multi agents (SMA)

Ce chapitre a pour objectif d'introduire les systèmes multi-agents qui constituent un des piliers de notre travail. Nous nous intéressons tout d'abord aux entités qui composent cette catégorie de système: *les agents* et *les systèmes multi-agents.*

Depuis une dizaine d'années, les systèmes multi-agents ont connu un grand essor et sont appliqués à des domaines très variés comme, par exemple, le domaine de la simulation et de la vie artificielle, la robotique, le traitement d'images.

Les systèmes multi-agents sont issus de l'intelligence artificielle distribuée (IAD), une branche de l'intelligence artificielle qui s'articule autour de trois axes :

- La résolution distribuée des problèmes qui s'intéresse à la manière de diviser un problème en un ensemble d'entités distribuées et coopérantes et à la manière de partager la connaissance du problème afin d'en obtenir la solution.
- L'intelligence artificielle parallèle qui développe des langages et des algorithmes parallèles pour l'intelligence artificielle (IA) visant ainsi l'amélioration des performances des systèmes d'IA.
- Les systèmes multi-agents qui privilégient une approche décentralisée de la modélisation et mettent l'accent sur les aspects collectifs des systèmes.

1 Les agents

Durant ces dernières années, nous avons assisté à un fort et rapide développement des recherches sur les agents et les systèmes multi-agents. Le terme agent est un terme générique qui se rapporte à différentes entités [Franklin and Graesser, 1996]:

- Des entités biologiques : les agents associés sont appelés agents biologiques,
- Des robots autonomes,
- Des logiciels informatiques et leurs composants qu'ils soient intégrés dans des systèmes d'exploitation ou des systèmes informatiques complexes.

1.1 Définitions

L'agent : Il n'existe pas, actuellement, une définition de l'agent qui fasse foi dans le monde de l'intelligence artificielle distribuée. Il est donc nécessaire, pour avoir une bonne vision de ce concept, de confronter plusieurs de ces définitions. Nous allons présenter trois d'entre-elles.

1. Jacques Ferber [Ferber, 1995] définit un agent comme étant une entité physique ou virtuelle évoluant dans un environnement dont il n'a qu'une représentation partielle et sur lequel il peut agir (voir figure 3.1). Il est capable de communiquer avec d'autres agents et est doté d'un comportement autonome.

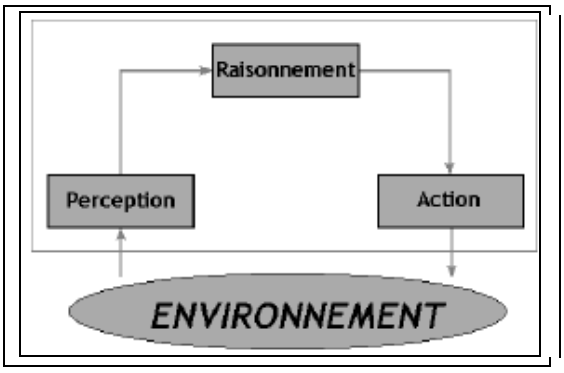

Figure 2.1 : Fonctionnement d'un agent

Cette définition aborde une notion essentielle : l'autonomie. En effet, ce concept est au centre de la problématique des agents. L'autonomie est la faculté d'avoir ou non le contrôle de son comportement sans l'intervention d'autres agents ou d'êtres humains. Une autre notion importante abordée par cette définition concerne la capacité d'un agent à communiquer avec d'autres.

2. Selon Yves Demazeau [Demazeau and Costa, 1996], un agent est une entité réelle ou virtuelle dont le comportement est autonome, évoluant dans un environnement qu'il est capable de percevoir et sur lequel il est capable d'agir, et d'interagir avec

les autres agents. Cette définition introduit l'interaction [1] qui, comme nous le verrons par la suite, est le moteur des systèmes multi-agents. En effet, l'interaction suppose la présence d'agents capables de se rencontrer, de communiquer, de collaborer et d'agir.

3. Pour Mickael Wooldridge [Wooldridge, 1999], un agent est un système informatique capable d'agir de manière autonome et flexible dans un environnement. Par flexibilité on entend :
   - Réactivité : un système réactif maintient un lien constant avec son environnement et répond aux changements qui y surviennent.
   - Pro-activité : un système pro-actif (aussi appelé téléonomique) génère et satisfait des buts. Son comportement n'est donc pas uniquement dirigé par des événements.
   - Capacités sociales : un système social est capable d'interagir ou coopérer avec d'autres systèmes.

Ces définitions sont le résultat de différentes approches. Il existe en fait plusieurs types d'agents, qui selon les capacités possédées, seront qualifiés de réactifs, cognitifs ou hybrides.

Capacités cognitives. Ces agents possèdent une représentation explicite de leur environnement, des autres agents et d'eux-mêmes. Ils sont aussi dotés de capacités de raisonnement et de planification ainsi que de communication. Ces agents sont structurés en société où il règne donc une véritable organisation sociale. Le travail le plus représentatif de cette famille d'agent porte sur le modèle BDI (Believe Desir Intention) [Rao and Georgeff, 1995]. Les sources de ces travaux sont les sciences humaines et sociales.

Capacités réactives. Ces ne possèdent pas de moyen de mémorisation et n'ont pas de représentation explicite de leur environnement : ils fonctionnent selon un modèle stimuli/ réponse. En effet, dès qu'ils perçoivent une modification de leur environnement, ils répondent par une action programmée. L'exemple le plus célèbre est celui de la fourmilière étudié par Alexis Drogoul [Drogoul, 1993].

Modèle d'agent hybride. Ces agents sont des agents ayant des capacités cognitives et réactives. Ils conjuguent en effet la rapidité de réponse des agents réactifs ainsi que les capacités de raisonnement des agents cognitifs. Cette famille regroupe donc des agents dont le modèle est un compromis autonomie/coopération et efficacité/complexité. Pour illustrer cette famille, nous pouvons citer l'architecture ASIC [Boissier, 1996] utilisée pour le traitement

[1] Dans les systèmes multi-agents, ce terme englobe la communication (l'échange d'informations) et l'action sur le monde.

numérique d'images, l'architecture ARCO [Rodriguez, 1994] créée dans le cadre de la robotique collective et l'architecture ASTRO [Occello, 1998] développée pour être utilisée dans les systèmes multi-agents soumis à des contraintes de type temporel.

1.2 Caractéristiques d'un agent

En partant de l'ouvrage de Wooldrige et Jennings, 1995, et des définitions citées, on peut identifier les caractéristiques suivantes pour la notion d'agent:

- **situé :** l'agent est capable d'agir sur son environnement à partir des entrées sensorielles qu'il reçoit de ce même environnement.
- **autonome:** l'agent est capable d'agir sans l'intervention d'un tiers (humain ou agent) et contrôle ses propres actions ainsi que son état interne.
- **proactif :** l'agent doit exhiber un comportement proactif et opportuniste, tout en étant capable de prendre l'initiative au bon moment.
- **capable de répondre à temps :** l'agent doit être capable de percevoir son environnement et d'élaborer une réponse dans le temps requis.
- **social :** l'agent doit être capable d'interagir avec des autres agents (logiciels ou humains) afin d'accomplir des tâches ou aider ces agents à accomplir les leurs.

1.3 Catégories d'agents

Il existe une gradation dans l'intelligence d'un agent que l'on classe habituellement en Il existe trois catégories :

- **réactivité :** l'agent est uniquement dirigé par les événements perçus dans l'environnement. Il réagit de manière opportuniste à ces changements.
- **pro-activité :** l'agent n'agit pas simplement en réponse à des changements de l'environnement, mais est aussi capable de s'assigner des buts et de prendre des initiatives pour les atteindre ;
- **socialité :** l'agent interagit avec les autres agents et les humains.

1.4 Niveaux de description d'un agent

Lorsque nous définirons les systèmes multi-agents nous parlerons de modèle d'agents, d'architecture d'agents et d'implémentation d'agents. Il est donc nécessaire d'expliciter ces termes qui constituent les différents niveaux de description d'un agent qui sont [Wooldridge and Jennings, 1995]:

- Le *modèle* qui décrit comment l'agent est compris, ses propriétés et comment on peut les représenter.
- L'*architecture* qui est un niveau intermédiaire entre le modèle et le contrôle et l'implémentation. Elle précise la création du système c'est-à-dire les propriétés qu'il doit posséder conformément au modèle et les liaisons avec les autres agents.
- L'*implémentation* qui s'occupe de la réalisation pratique de l'architecture des agents à l'aide de langages de programmation.

1.5 Architectures d'agents

1.5.1 Architecture informatique parallèle

Il existe dans cette architecture deux modèles, **Parallélisme** : modèles à contrôle centralisé (SIMD) à grain fin et **Distribution** : modèles à contrôle décentralisé (MIMD) à grain fort.

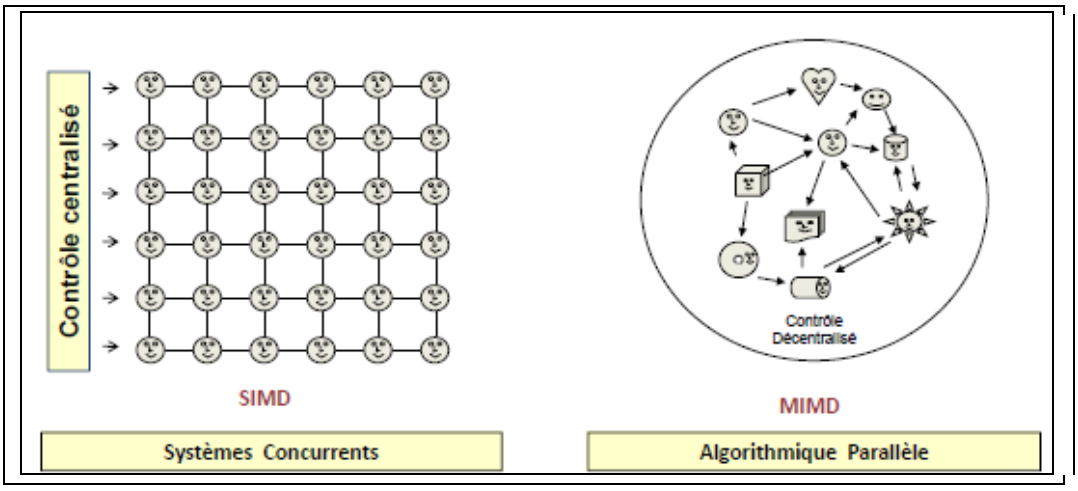

Figure 2.2 : Les deux classes d'architectures parallèles

Dans le modèle **SIMD**, les agents ont tous la même structure : comme les instances d'une même classe, ils possèdent les mêmes variables d'état (slots) et le même comportement (mêmes méthodes). Au niveau de l'interaction avec les autres agents, on retrouve cette uniformité : le réseau interactif est régulier, homogène, symétrique et le protocole d'interaction est unique. Au cours du fonctionnement, seules les valeurs d'état des agents sont différentes et ce sont elles qui dessinent la forme du résultat attendu. On parle alors d'Emergence ou d'Intelligence Collective.

Dans le modèle **MIMD**, les agents ont tous une structure différente : ils ont des variables d'états différentes et ils exécutent des programmes différents. Au niveau de l'interaction, il en est de même ; les échanges sont hétérogènes : en temps, en vitesse, en quantité, en topologie ... Chaque agent possède en quelque sorte sa propre personnalité : il a

sa propre spécialité, il peut résoudre un type de problème particulier. On parle alors de Sociétés d'Experts ou de Système Multi Agents (SMA).

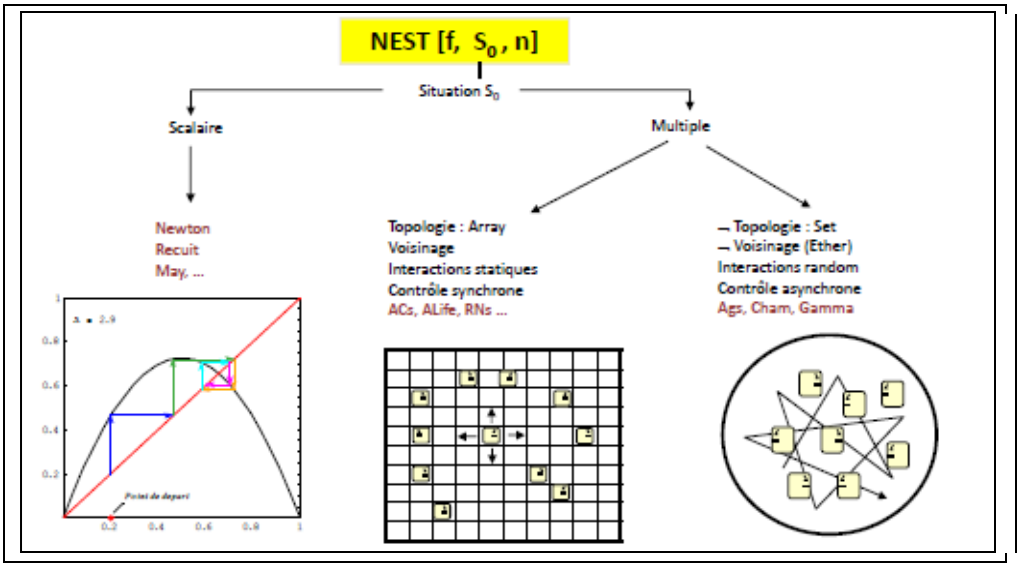

Figure 2.3 : Architectures à contrôle centralisé

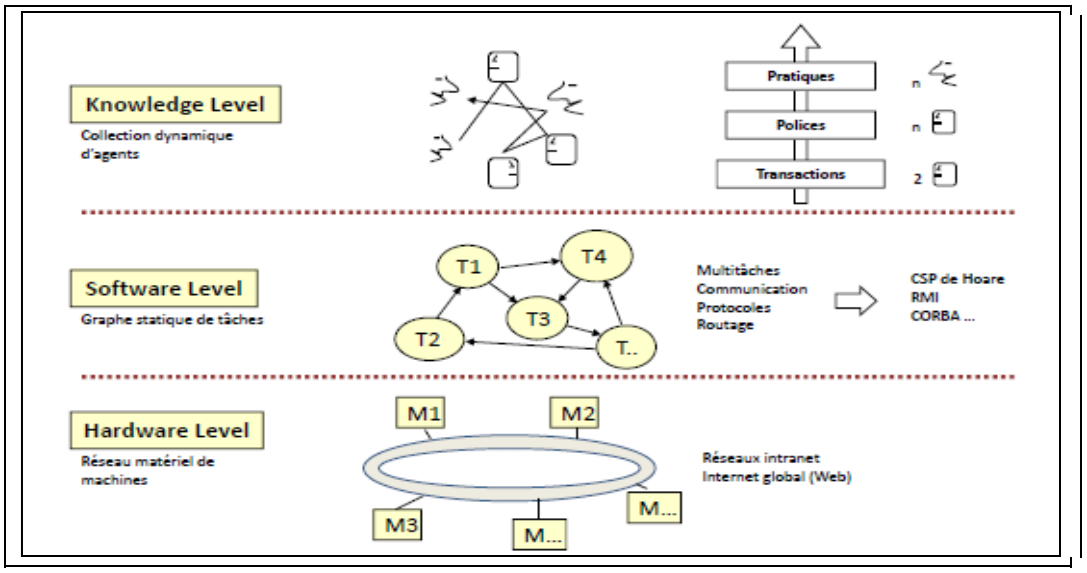

Figure 12.4 : Architectures à contrôle distribué

1.5.2 Architectures cognitives et réactives

La distinction que l'on peut faire entre cognitif et réactif tient de la représentation du monde dont dispose l'agent. Si l'individu est doté d'une "représentation symbolique" du monde à partir de laquelle il est capable de formuler des raisonnements, on parlera d'agent cognitif tandis que s'il ne dispose que d'une "représentation sub-symbolique", c'est-à-dire limitée à ses perceptions, on parlera d'agent réactif. La seconde distinction entre comportement téléonomique ou réflexe sépare les comportements intentionnels (poursuite de buts explicites) des comportements liés à des perceptions. Les tendances des agents peuvent ainsi être exprimées explicitement dans les agents ou au contraire provenir de

l'environnement. On peut construire un tableau regroupant les différents types d'agents [Ferber, 1995] :

	Agents cognitifs	**Agents réactifs**
Comportement téléonomique	Agents intentionnels	Agents pulsionnels
Comportement réflexe	Agents "modules"	Agents tropiques

Tableau 2.1 : Les types d'agent

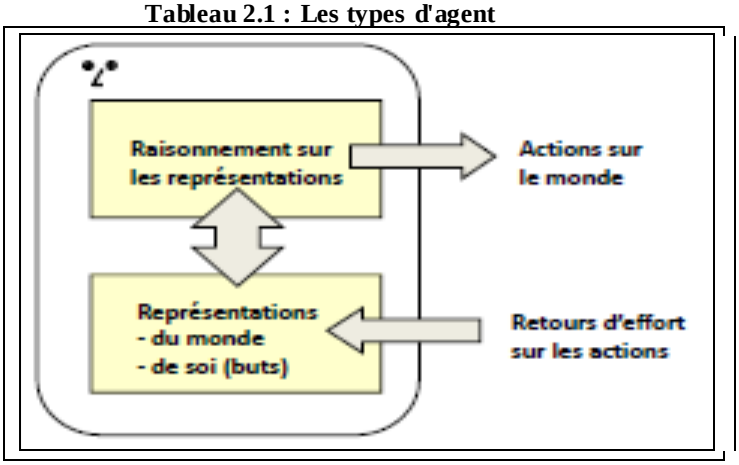

Figure 2.5 : Les agents cognitifs

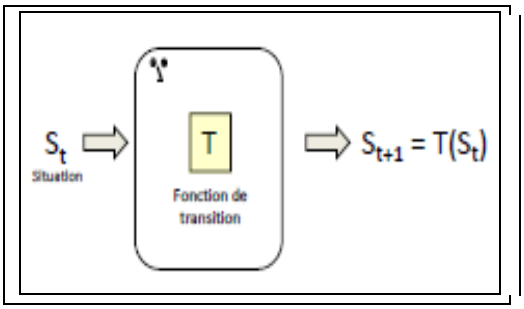

Figure 2.6 : Les agents réactifs

2 Les systèmes multi-agents

Nous avons vu précédemment que les agents fonctionnaient dans des environnements qui peuvent être réels ou virtuels. Ces environnements peuvent permettre à des agents de se rencontrer et donner ainsi naissance à des communications ou encore des interactions. La communication consiste à envoyer des messages ; l'interaction, quant à elle, permet d'aller plus loin et d'établir des conversations structurées entre agents. C'est donc, comme le souligne Jacques Ferber [Ferber, 1995], une mise en relation dynamique de plusieurs agents par le biais d'actions réciproques qui influenceront le comportement futur des différents intervenants.

2.1 Définitions

Un système multi-agents est un ensemble d'agents qui évoluent dans un environnement commun. Dans [Weiss, 1999], Gerhard Weiss définit l'intelligence artificielle distribuée comme étant l'étude, la conception et la réalisation de systèmes multi-agents qu'il présente comme étant des systèmes dans lesquels des agents intelligents interagissent et poursuivent un ensemble de buts ou réalisent un ensemble d'actions.

Alors comment, à partir d'un ensemble d'agents pas forcément intelligents, on arrive à un système complexe où semble se dégager de l'intelligence ? Dans un système multi-agents l'intelligence provient de l'émergence d'un comportement global [Gasser, 1992].

Les systèmes multi-agents, depuis leur création, font l'objet d'un véritable engouement. Les raisons exprimées dans [Bourdon, 2001] indiquent qu'il ne s'agit pas simplement d'un phénomène passager. En effet, l'importance de l'approche *système multi-agents* croît considérablement et s'explique essentiellement par :

- Le fait que les systèmes d'information soient de plus en plus distribués, ouverts, à grande échelle et hétérogènes. Cela rend les interconnexions tellement compliquées et croissantes qu'elles dépassent la compréhension globale que peut en avoir un être humain.
- Le fait que cette approche soit utile pour expérimenter des réflexions sociologiques et psychologiques pour ce qui concerne par exemple les relations entre personnes dans les sociétés modernes.

Un Système multi-agents ouvert [Vercouter, 2000] partage les caractéristiques des systèmes ouverts. Pour ce qui concerne les entités élémentaires du système que sont les agents, ils n'ont pas la possibilité d'avoir une représentation complète de l'environnement. Pour ce qui est du système dans sa globalité, il doit être modulaire et extensible. La modularité concerne le fait que le système multi-agents est composé de plusieurs sous-systèmes mis en relation. Ces sous-systèmes ont chacun leur propre mode de fonctionnement. L'extensibilité se traduit par le fait que le système multiagents supporte l'ajout et le retrait dynamique d'éléments.
A l'inverse, le qualificatif *fermé* signifie que l'ensemble des agents qui compose le système reste le même.

Un système multi-agents homogène/hétérogène est composé d'agents homogènes. Deux agents ont cette particularité s'ils sont identiques du point de vue de leur modèle et de leur architecture. Le qualificatif *hétérogène* est utilisé pour préciser que le système multi agents

est composé d'agents différents du point de vue de leurs modèles et de leurs architectures.

2.2 Les cinq problématiques des SMA

2.2.1 La problématique de l'action

Comment un ensemble d'agents peuvent agir simultanément dans un environnement partagé, et comment cet environnement interagit avec les agents ? Les questions sous-jacentes sont celles de la représentation de l'environnement par les agents, de la collaboration entre agents, de la planification multi-agents.

2.2.2 L'agent et de sa relation au monde

C'est représenté par le modèle cognitif dont dispose l'agent. L'individu d'une société multi-agents doit être capable de mettre en œuvre les actions qui répondent au mieux à ses objectifs. Cette capacité à la décision est liée à un "état mental" qui reflète les perceptions, les représentations, les croyances et un certain nombre de paramètres "psychiques" (désirs, tendances, ...) de l'agent. La problématique de l'individu et de sa relation au monde couvre aussi la notion d'engagement de l'agent vis-à-vis d'un agent tiers.

2.2.3 La problématique de l'interaction

Qui s'intéresse aux moyens de l'interaction (quel langage ? quel support ?), et à l'analyse et la conception des formes d'interactions entre agents. Les notions de collaboration et coopération (en prenant coopération comme collaboration + coordination d'actions + résolution de conflits) sont ici centrales.

2.2.4 La problématique de l'adaptation

En termes d'adaptation individuelle ou apprentissage d'une part et d'adaptation collective ou évolution d'autre part.

2.2.5 La réalisation effective et de l'implémentation des SMA

En structurant notamment les langages de programmation en plusieurs types allant du langage de type L5, ou langage de formalisation et de spécification, au langage de type L1 qui est le langage d'implémentation effective. Entre les deux, on retrouve le langage de communication entre agents, de description des lois de l'environnement et de représentation des connaissances.

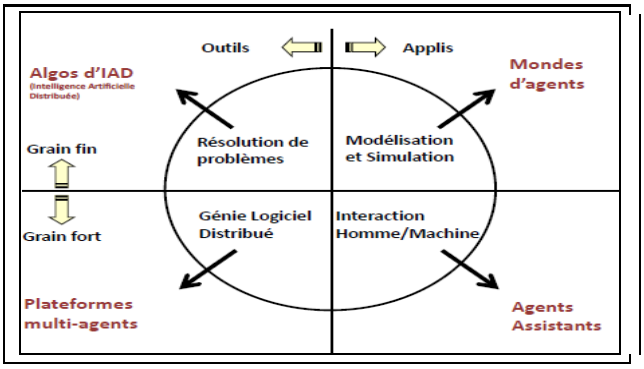

Figure 2.7 : Les problématiques multi-agents

2.3 Architecture des systèmes multi-agents

En reprenant les cinq problématiques précédentes, on peut décrire quelques éléments de l'architecture d'un système multi-agent.

- Les agents doivent être dotés de systèmes de décisions et de planification. Dans la catégorie des interactions avec l'environnement.
- Les agents doivent être dotés d'un modèle cognitif : plusieurs modèles existent, l'un des plus classiques étant le modèle BDI (Beliefs-Desires-Intentions). Il considère d'une part l'ensemble de croyances (Beliefs) de l'agent sur son environnement, un ensemble d'objectifs (Desires). Et un ensemble d'intentions (Intentions) qui peuvent ensuite se traduire directement en actions.
- Les agents doivent être dotés d'un système de communication. Plusieurs langages spécialisés existent : le Knowledge Query and Manipulation Language (KQML), le standard FIPA-ACL (ACL pour Agent Communication Language) de la Foundation for Intelligent Physical Agents (FIPA) . qui se repose sur la théorie des actes de langage, chère à John Searle.
- La problématique de l'adaptation est un sujet difficile .On pourrait toutefois citer l'exemple de certains virus, aussi bien biologiques qu'informatiques, capable de s'adapter à leur environnement en mutant.
- l'implémentation effective du système multi-agent

2.4 La décomposition multi-agents

Nous adopterons la vision d'un système multi-agents introduite par Yves Demazeau et communément admise aujourd'hui [Demazeau, 1995]. La décomposition Voyelles identifie un axe **A**gent, un axe **E**nvironnement, des **I**nteractions et une structure **O**rganisationnelle explicite ou non.

Équation 1

$$SMA = Agent + Environnement + Interaction + Organisation$$

De plus l'intelligence d'un système multi-agents étant plus que la somme de l'intelligence des agents qui la compose, un principe a été énoncé pour rendre compte du surcroît d'intelligence observé, de l'intelligente émergente.

Équation 2

$$Fonction\ (SMA) = \sum\ fonction\ (Agent) + fonction\ collective$$

Enfin, un système multi-agents peut être vu comme un agent d'un système multi-agents global. Il s'agit du principe de la récursivité.

Équation 3

$$Recursion : A = A_{élémentaire} / SMA$$

2.4.1 Les agents (A)

Cette partie de l'analyse AEIO regroupe les éléments nécessaires à la construction des agents à savoir leur modèle, leur architecture, leur implémentation...

2.4.2 L'environnement (E)

Dans un système multi-agents, on appelle environnement l'espace commun aux agents du système.

Un environnement peut être [Russell and Norvig, 1995, Wooldridge et al., 2000]:

- *Accessible* si un agent peut, à l'aide des primitives de perception, déterminer l'état de l'environnement et ainsi procéder, par exemple, à une action. Si l'environnement est *inaccessible* alors il faut que l'agent soit doté de moyens de mémorisation afin d'enregistrer les modifications qui sont intervenues.
- *Déterministe*, ou non, selon que l'état futur de l'environnement ne soit, ou non, fixé que par son état courant et les actions de l'agent.
- *Episodique* si le prochain état de l'environnement ne dépend pas des actions réalisées par les agents.
- *Statique* si l'état de l'environnement est stable (ne change pas) pendant que l'agent réfléchit. Dans le cas contraire, il sera qualifié de *dynamique*.
- *Discret* si le nombre des actions faisables et des états de l'environnement est fini.

2.4.3 Les interactions (I)

Les interactions proviennent de la mise en relation dynamique de plusieurs agents par le

biais d'un ensemble d'actions réciproques. Il existe plusieurs types d'interactions, qui dépendent de trois paramètres que sont les buts, les ressources et les compétences comme l'illustre le tableau référencé 2.1 tiré de l'ouvrage de Jacques Ferber [Ferber, 1995].

- Illustrons ces différentes situations d'interactions par des exemples [Bourdon, 2001]: L'*indépendance* est une situation d'interaction similaire à celle de deux personnes qui se rencontrent dans une rue assez large pour qu'elles puissent passer en même temps.
- La collaboration simple

BUTS	RESSOURCES	COMPETENCES	TYPE DE SITUATION	CATEGORIE
Compatibles	Suffisantes	Suffisantes	Indépendance	Indifférence
		Insuffisantes	Collaboration simple	Coopération
	Insuffisantes	Suffisantes	Encombrement	
		Insuffisantes	Collaboration coordonnée	
Incompatibles	Suffisantes	Suffisantes	Compétition individuelle pure	Antagonisme
		Insuffisantes	Compétition collective pure	
	Insuffisantes	Suffisantes	Conflits individuels pour des ressources	
		Insuffisantes	Conflits collectifs pour des ressources	

Tableau 2.2 : Types d'interactions d'après [Ferber, 1995]

- Pour illustrer l'*encombrement* on peut prendre l'exemple du trafic aérien.
- Le cas d'un système intégrant un mécanisme de régulation distribuée ou le cas d'une société de robots autonomes peuvent illustrer ce qu'on appelle la *collaboration ordonnée*.
- La course à pied illustre bien ce qu'est une *compétition individuelle pure*. _Une épreuve tel que le relais 4*100m illustre bien la *compétition collective pure* car les équipes ne se gênent pas.
- Les *conflits individuels pour des ressources* sont représentables par le cas d'animaux se battant pour leur territoire.
- Les coalitions sont des exemples de *conflits collectifs pour des ressources*.

2.4.4 L'organisation (O)

Le concept d'organisation peut être définit comme étant une structure décrivant les interactions et autres relations qui existent (dans le but d'assouvir un objectif commun) entre les membres de la dite organisation [Fox, 1981]. L'organisation peut donc apparaître comme une structure de coordination et de communication [Malone, 1987]. Jacques Ferber [Ferber,

1995] va dans ce sens en affirmant que les organisations constituent à la fois le support et la manière dont se passent les inter-relations entre les agents, c'est-à-dire dont sont réparties les tâches, les informations, les ressources et la coordination d'actions. Il précise que ce qui rend l'organisation si difficile à cerner est qu'elle est à la fois le processus d'élaboration d'une structure et le résultat de ce processus. Cette citation met en avant la dualité du terme organisation : en effet, il y a l'aspect statique et l'aspect dynamique. Dans [Hübner et al., 2002] l'organisation est décrite via une spécification structurelle, une spécification fonctionnelle et une spécification déontique.

L'aspect dynamique, appelé aussi plus simplement organisation, est relatif aux relations entre les éléments du système, qui sont soumis à des changements dynamiques. L'aspect statique, appelé structure organisationnelle provient de la raison d'être des organisations qui est la nécessité pour des individus de se regrouper pour repousser leurs propres limites, en terme de capacités [March and Simon, 1958]. Il existe de nombreuses structures organisationnelles [Baeijs and Demazeau, 1996]. On en dénombre essentiellement trois grandes familles (les groupes, les hiérarchies et les marchés) basées sur les trois processus de base de coordination [Le Strugeon, 1995, Agimont, 1996] étudiés par Mintzberg [Mintzberg, 1982] et qui sont :

- l'ajustement mutuel lorsqu'un ou plusieurs agents se mettent d'accord entre eux pour partager une ressource afin de réaliser un but qui leur est commun,
- la supervision directe qui fait intervenir une relation hiérarchique entre les agents : le gérant grâce à son contrôle sur les autres agents peut réguler la consommation faite par les agents qui sont soumis à son autorité,
- La standardisation où l'agent gérant, qui a autorité sur les autres, met en place des procédures à réaliser par les autres agents dans des cas concrets.

2.5 Population d'agents

- **Dans un SMA rien n'est complètement global**

L'environnement étant vaste (sans compter le problème de la représentation récursive :je sais que je sais que je sais...) et ouvert, il n'est pas possible en un lieu donné (exemple, un agent) de stocker toute la représentation du monde. Mais, un agent peut se déplacer ou encore interagir avec d'autres agents qui sont dans son voisinage pour explorer l'environnement.

- **Dans un SMA rien n'est complètement local**

Pour un agent donné, toutes ses entités (informations, processus, buts, …) sont locales mais elles restent accessibles à l'introspection par d'autres agents. Le moyen d'accéder à cette information passe par les interactions entre les agents.

2.6 Typologie des interactions

2.6.1 Interaction Directe

i. Action directe (interdite)

Un agent peut agir directement sur l'état physique d'une chose de son environnement (objet, autre agent, humain). Cela sera interdit dans un SMA :

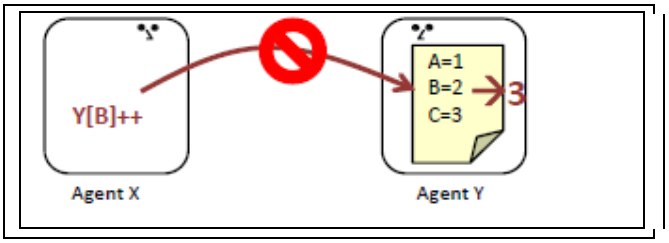

Figure 2.8 : Action directe (interdite)

ii. Requête (formelle ou langagière)

L'agent envoie une requête à un interlocuteur qui est un autre agent ou à un humain de son environnement (pas à un objet !). L'interlocuteur interprète cette requête et la satisfait ou non en fonction de sa propre subjectivité (état physique et mental) :

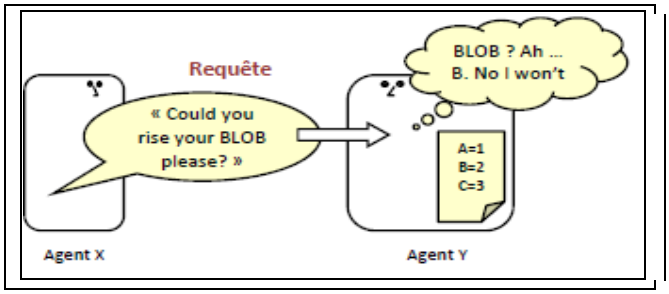

Figure 2.9 : Requête (formelle ou langagière)

2.6.2 Interaction Indirecte

i. Blackboard (base de connaissances)

Plusieurs agents déposent et recueillent des objets ou des informations dans une partie de l'environnement prévue à cet effet. Cette partie commune est appelée « Blackboard ».

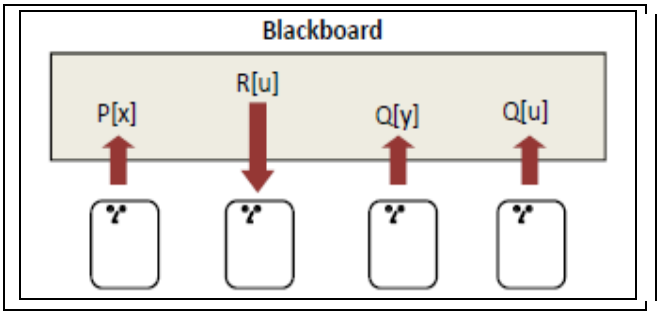

Figure 2.10 : Blackboard (base de connaissances)

ii. Partage de ressources (Stygmergie)

Les modèles de population animales par exemple sont fondés sur une compétition pour une quantité de ressources à partager qui est fixée : la ressource sert alors de médiateur entre les agents.

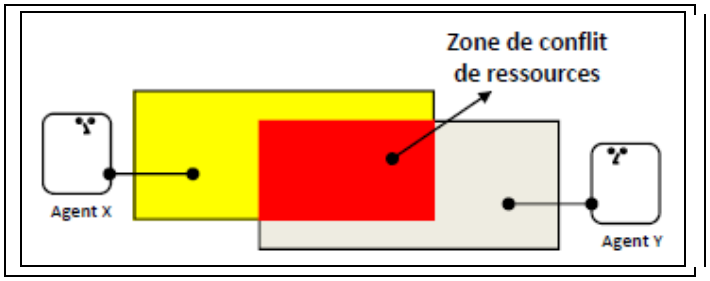

Figure 2.11 : Partage de ressources (Stygmergie)

2.7 La communication entre les agents

Les agents sont des éléments actifs car ils intègrent systématiquement des senseurs (Pour observer d'autres agents ou l'extérieur du système) et sont capables d'agir dans leur environnement. L'interaction entre les agents passe par une communication et conditionne l'organisation du système.

L'interaction entre agents a été étudiée dans [Ferber, 1995] selon deux axes : Le partage de ressources et le partage de compétences.

Les agents logiciels mettent en œuvre leurs communications de deux manières. D'une part, les agents peuvent communiquer de manière indirecte en laissant des traces dans leur environnement comme les fourmis laissent de la phéromone pour indiquer aux autres fourmis un chemin vers une source de nourriture. D'autre part, les agents peuvent communiquer par envoi de messages.

Les communications, dans les SMA comme chez les humains, sont à la base des interactions et de l'organisation. Une communication peut être définie comme une forme d'action locale d'un agent vers d'autres agents. Les questions abordées par un modèle de communication peuvent être résumées par l'interrogation suivante :

1. *Pourquoi les agents communiquent-ils ?* La communication doit permettre la mise en œuvre de l'interaction et par conséquent la coopération et la coordination d'actions.
2. *Quand les agents communiquent-ils ?* Les agents sont souvent confrontés à des situations où ils ont besoin d'interagir avec d'autres agents pour atteindre leurs buts locaux ou globaux. La difficulté réside dans l'identification de ces situations.

3. *Avec qui les agents communiquent-ils ?* les communications peuvent être sélectives sur un nombre restreint d'agents ou diffusées à l'ensemble des agents. Le choix de l'interlocuteur dépend essentiellement des accointances de l'agent.
4. *Comment les agents communiquent-ils ?* La mise en œuvre de la communication nécessite un langage de communication compréhensible et commun à tous les agents. Il faut identifier les différents types de communication et définir les moyens permettant non seulement l'envoi et la réception de données mais aussi le transfert de connaissances avec une sémantique appropriée à chaque type de message.

On distingue deux modèles de communication :

Communication par partage d'information	Communication par envoi de messages
Mode adopté dans les systèmes à tableau noir. Communication via le tableau noir.	Agents en liaison directe. Envoi directement et explicitement au destinataire.

Tableau 2.3 : Les deux modèles de communication

On peut distinguer deux types de communication :

Communication synchrone	Communication asynchrone
- le message est reçu en même temps qu'il est émis (dialogue oral ou gestuel, téléphone. - il peut ou non y avoir coprésence des interlocuteurs dans l'espace. -le locuteur et l'allocutaire sont en présence (gestes ou paroles)	- le message est émis hors de la présence de l'allocutaire. -le monde sera partagé par la représentation que l'un des interlocuteurs s'en fait. -les traces qui peuvent être observées hors du lieu et du moment de leur production (dessin ou écriture).

Tableau 2.4 : Les deux types de communication

2.8 Application des SMA

Dans le monde de la recherche

On distingue généralement trois types d'utilisation : la simulation de phénomènes complexes, la résolution de problème, et la conception de programmes.

On utilise les systèmes multi-agents pour simuler des interactions existants entre agents autonomes. On cherche à déterminer l'évolution de ce système afin de prévoir l'organisation qui en résulte. Par exemple, en sociologie, on peut paramétrer les différents agents composant une communauté. En ajoutant des contraintes, on peut essayer de comprendre quelle sera la composante la plus efficace pour parvenir à un résultat attendu (construction d'un pont). Ce qui importe c'est le comportement d'ensemble et non pas le comportement individuel. Des applications existent en physique des particules (agent =

particule élémentaire), en chimie (agent = molécule), en biologie cellulaire (agent = cellule), en éthologie (agent = animal), en sociologie et en ethnologie (agent = être humain). L'autonomie permet ici de simuler le comportement exact d'une entité.

L'intelligence artificielle distribuée est née pour résoudre les problèmes de complexité des gros programmes monolithiques de l'intelligence artificielle : l'exécution est alors distribuée, mais le contrôle reste centralisé. Au contraire, dans les SMA, chaque agent possède un contrôle total sur son comportement. Pour résoudre un problème complexe, il est en effet parfois plus simple de concevoir des programmes relativement petits (les agents) en interaction qu'un seul gros programme monolithique. L'autonomie permet au système de s'adapter dynamiquement aux changements imprévus qui interviennent dans l'environnement.

Dans le même temps, le génie logiciel a évolué vers des composants de plus en plus autonomes. Les SMA peuvent être vus comme la rencontre du génie logiciel et de l'intelligence artificielle distribuée, avec un apport très important des systèmes distribués. Par rapport à un objet, un agent peut prendre des initiatives, peut refuser d'obéir à une requête, peut se déplacer ... L'autonomie permet au concepteur de se concentrer sur une partie humainement appréhendable du logiciel.

2.9 Environnement et Plate-formes de développement de SMA

Les plates-formes multi-agents fournissent des services qui simplifient le développement d'une application multi-agent.

Une plate-forme peut prendre en charge la recherche d'un agent ayant une compétence particulière, ce qui est le cas dans la plate-forme Magique. Magique se base sur une communauté de hiérarchie d'agents. Un agent est une coquille vide dans laquelle le concepteur code les compétences de l'agent.

1. JADE (Java Agent DEvelopment), est un framework de développement de systèmes multi-agents, open-source et basé sur le langage Java. Il offre en particulier un support avancé de la norme FIPA-ACL, ainsi que des outils de validation syntaxique des messages entre agents basés sur les ontologies.
2. MadKit, est une plate-forme multi-agents modulaire écrite en Java et construite autour du modèle organisationnel Agent/Groupe/Rôle. C'est une plate-forme libre basée sur la licence GPL/LGPL développée au sein du LIRMM.
3. JAgent, est un framework open source réalisé en Java dont l'objectif est de faciliter le développement et le test de systèmes multi-agents.

4. JACK, est un langage de programmation et un environnement de développement pour agents cognitifs, développé par la société Agent Oriented Software comme une extension orientée agent du langage Java.
5. SemanticAgent, est basé sur JADE et permet le développement d'agents dont le comportement est représenté en SWRL. SemanticAgent est développé au sein du LIRIS, il est open-source et sous licence GPL V3.

- **Langages de base les plus utilisés :** Java, LISP, C++, Prolog, SMALTALK, ainsi que les langages d'Acteurs, qui supportent les mécanismes d'exécution parallèles.

3 Conclusion

Nous avons défini, dans un premier chapitre, la notion de systèmes complexes physiques ouverts. Cette classe de systèmes se caractérise, d'une part, par la nature matérielle de ses composants, d'autre part, par la richesse des interactions mises en jeu. Les composants sont en effet en interaction entre eux, mais aussi avec l'environnement matériel du système. Ils peuvent le percevoir et agir sur lui, tout comme l'environnement agit sur eux. Nous avons identifié en particulier quatre catégories de systèmes complexes physiques : les systèmes de traitement (d'information et de signaux), les systèmes de communication, les systèmes de contrôle/commande et les systèmes interactifs.

Le paradigme multi-agents de part son inhérente décentralisation, sa prise en compte du global au niveau local, la richesse des interactions qu'elle peut opérationnaliser et la prise en compte du niveau social des composants, est efficace pour s'intéresser aux systèmes complexes. En effet, l'approche et la richesse des modèles développés dans ce domaine permet de s'intéresser aux systèmes complexes. Ces modèles sont, par exemple, liés à l'environnement, aux diverses interactions qui permettent entre autres la communication et la coopération, les organisations et aux agents.

Les systèmes multi-agents embarqués se traduisent généralement par des systèmes d'entités physiques autonomes utilisant des communications sans fil. Il est donc indispensable, au delà des aspects méthodologiques, d'étudier ces besoins architecturaux spécifiques (quelle architecture pour assurer une communication fiable entre les agents?) afin de les intégrer à l'approche et de proposer les outils nécessaires.

Chapitre 3 : Approches SMA dans le partitionnement M/L.

Le problème de partitionnement est un problème Np-complet qui dans des cas ne peut pas être résolu dans un temps raisonnable, sachant que des méthodes existent déjà pour donner des résultats satisfaisants. Les problèmes de type NP-complet sont une classe de problèmes difficiles qui, jusqu'à maintenant, ne peuvent pas être résolus dans un temps polynomial. Ces méthodes incluent :

- Approximation (une solution proche de l'optimal peut être trouvée),
- Aléatoire (une solution optimale peut être trouvée avec une certaine probabilité)
- Heuristique (une bonne solution peut être trouvée, mais sans garanties).

Pour résoudre un problème avec les SMA il faut le diviser, mais pour beaucoup de problèmes il est difficile de diviser un algorithme qui garantit une solution optimale. Mais des méta-heuristiques peuvent être utilisées pour combiner entre : des heuristiques et autres stratégies pour obtenir des solutions meilleures [Agerbeck, 1998].

1 Les problèmes NP-complet

1.1 Définition

1.1.1 Les classes des problèmes NP-complet

Il existe plusieurs types de problèmes NP-complet on cite :

1. **Les problèmes de satisfaction de contraintes (*Constraint Satisfaction Problems* (CSP))** : ce sont des problèmes mathématiques, où quelques contraintes sur un ensemble de variables sont données, le but est de trouver la valeur pour toutes les variables qui satisfaissent les contraintes données. Le problème de partitionnement peut être vu comme un CSP.

2. **Satisfiability (SAT)** : c'est un problème qui consiste à déterminer, si on peut assigner des valeurs à des variables dans une formule booléenne, pour que la formule prenne la valeur vraie.

3. **Le problème du voyageur de commerce (*Travelling Salesman* problem (TSP))** : c'est le problème qui consiste à trouver le chemin de coût le plus bas, qui visite un nombre de villes une seule fois et ensuite retourne au début. Le résultat est les villes visitées et le coût de passer de n'importe quelle ville à une autre. Il y a aussi le M-TSP les m-voyageurs doivent couvrir les villes données et chaque ville doit être visitée par un et un seul voyageur. Chaque voyageur doit commencer de la même ville, appelée dépôt, et doit retourner à la fin à cette même ville.

4. **Le partitionnement des graphes** : on peut le définir comme suit : soit un graphe G= (V, E), où V représente les sommets, et les E : l'ensemble des arcs qui connectent les sommets. Les sommets et les arcs peuvent avoir un poids. Le problème de partitionnement de graphe consiste à diviser le graphe G en K parties.

2 Résolution des problèmes NP-complet

Il existe plusieurs techniques pouvant être utilisées pour résoudre les problèmes NP-complet. Les techniques les plus connues sont :

2.1 Approximation

Dans quelques problèmes NP-complet il peut être suffisant de trouver une solution proche de l'optimal pour avoir un résultat satisfaisant [Cormen, 2001]. La raison pour trouver une solution proche de l'optimal plus tôt que la solution optimale est dûe aux coûts de calcul dans

la mesure du possible pour trouver une solution dans un temps polynomial (Ex : la voyageur de commerce).

2.2 Aléatoire (Randomisation)

Un algorithme aléatoire est un algorithme qui peut faire appel à un générateur de nombres aléatoires pendant son exécution.

2.3 Heuristiques

Les heuristiques peuvent donner des solutions proches de l'optimal ou fournir une solution pour un certain nombre d'instances (pas toutes) de ce problème, En d'autres mots, l'heuristique cesse de chercher une solution optimale pour améliorer dans le temps d'exécution. Il y a une classe d'heuristiques générales nommées méta-heuristiques. Elles peuvent être appliquées sur une grande variété de problèmes mais sans vraiment garantir une solution optimale. Recherche tabou, recuit simulé, algorithme génétique, les colonies de fourmis sont des exemples de différentes méta-heuristiques.

3 SMA et les problèmes NP-complets

3.1 L'idée générale d'utiliser les SMA pour résoudre les problèmes NP-complet

Il y a plusieurs approches possibles pour résoudre un problème NP-complet avec les SMA on cite :

1. Les SMA peuvent être utilisés comme une liaison entre les heuristiques [Hammami, 2005] [Zhou, 2005]. La première idée est de laisser de différents agents avoir la responsabilité d'une meta-heuristique (ou une heuristique), puis dans la mesure du possible combiner et utiliser les solutions données par les autres agents pour avoir de meilleurs résultats et des méta-heuristiques.

2. Une autre approche consiste à utiliser les SMA pour résoudre les problèmes de satisfaction de contraintes (Ex : partitionnement matériel/logiciel dans le co-design). L'idée est de diviser les contraintes et les variables entre les agents, et de manière adéquate, laisser chaque agent individuellement optimiser ses contraintes locales pour obtenir une solution globale commun optimale [Xiaolong, 2002] [Liu, 1994].

3. La troisième approche est d'analyser les instances réelles du problème et de cette manière déterminer quelles entités vont être utilisées, et comment elles vont interagir pour résoudre le problème, ensuite utiliser les interactions pour construire un SMA.

4 SMA et les méta-heuristiques

Comme il est décrit précédemment, les méta-heuristiques sont utilisées souvent quand il s'agit de problème NP-complet de type « problème d'optimisation de contraintes ». Bien qu'elles fournissent la possibilité de trouver une solution acceptable, elles ne peuvent pas être dans un temps raisonnable. La raison est que les algorithmes de recherche de ces heuristiques -sur des grands problèmes d'optimisation- souvent perdent du temps dans l'optimisation locale (ou minimisation locale), qu'ils ne peuvent pas finir dans un temps raisonnable. Ces temps là ne sont pas les mêmes pour tous les algorithmes de recherche des heuristiques [Emin Aydin, 2004] [Zhou, 2005].

Une approche pour améliorer le comportement des algorithmes de recherche de ces heuristiques est d'utiliser le paradigme multi-agents pour combiner des méta-heuristiques différentes, de manière coopérative, ils peuvent se compléter et s'entre-aider les uns les autres, on évitant la perte de temps d'exécution dans l'optimisation locale, [Emin Aydin, 2004] [Zhou, 2005] [Hammami, 2005]. La raison pour laquelle on combine les méta-heuristiques est d'utiliser les différents points forts de chaque méta-heuristique pour minimiser les points faibles des autres. D'autre raison pour l'hybridation est d'assurer ***l'exploration et l'exploitation*** [Talbi, 2002]:

Deux buts compétitifs gouvernent la conception d'une méta-heuristique : exploration et exploitation.

L'exploration a pour objectif d'assurer que chaque partie de l'espace de recherche est parcourue suffisamment pour fournir une estimation fiable de l'optimisation globale.

L'exploitation est importante parce que le raffinement la solution courante va souvent donner une meilleure solution.

Les méta-heuristiques basées sur les populations, comme les algorithmes génétiques et les colonies de fourmis, sont très puissants dans l'exploration de l'espace de recherche, mais elles sont faibles dans l'exploitation de la solution trouvée. Cependant les heuristiques de recherche locale comme le recuit simulé et la recherche taboue sont très puissantes dans l'exploitation de solutions. Comme les deux types de méta-heuristiques sont complémentaires dans les points forts et points faibles, les combiner sera très bénéfique et donnera une meilleure solution.

4.1 Est-il bénéfique d'hybrider les méta-heuristiques et les SMA ?

Les méta-heuristiques hybrides n'ont pas vraiment besoin d'être implémentées comme des SMA, mais un bénéfice évident de l'utilisation des SMA est la possibilité de changer rapidement les méta-heuristiques utilisées par cette hybridation. Cependant la chose importante à remarquer est que même si toutes les implémentations montrent une meilleure performance, une performance similaire peut être probablement obtenue par l'implémentation d'un seul agent. Il est évident que les systèmes séquentiels peuvent être réalisés par un seul agent, qui traite des tâches différentes pendant la recherche de la solution [Zhou, 2005]. Il est aussi possible qu'un seul agent qui exécute le recuit simulé puis la recherche tabou de manière successive, donne les mêmes résultats que le recuit simulé et la recherche tabou exécutés séparément [Hammami, 2005]. Ou encore on peut les implémenter comme un agent unique qui exécute les différents algorithmes dans une certain séquence, et probablement il va mener aussi à des bons résultats [Emin Aydin, 2004].

4.1.1 Alors pourquoi utiliser les SMA ?

La chose la plus évidente est qu'on peut avoir un gain de performance de la recherche si on exécute les algorithmes en parallèle [Hammami, 2005] [Emin Aydin, 2004]. Plus encore le système sera plus facile à maintenir, parce que l'agent est indépendant des autres, et peut être remplacé par un autre agent qui est simplement une autre heuristique sans perturber la stabilité du système. Cela sera plus difficile si les heuristiques étaient toutes combinées dans un seul agent hybride.

4.1.2 L'utilisation de cette approche pour le partitionnement M/L

Si on prend les *colonies de fourmis* [Koudil, 2005] ou les *algorithmes génétiques* [Koudil, 2003] comme méta-heuristiques d'exploration, on aura besoin d'une méta-heuristique d'exploitation, comme le recuit simulé.

4.1.2.1 L'architecture de la solution

L'architecture des fourmis

Les fourmis dans cette solution auront presque la même architecture qu'une fourmi classique, mais à l'exception de :

Dans le choix de voisinage, les fourmis ne le choisissent pas seulement avec la phéromone, mais aussi à l'aide de l'algorithme de recuit simulé, car chaque fourmi exécutera cette méta-heuristique localement, pour calculer une fonction de coût local (avec des

métriques locales, taille, connectivité, etc). Et en suivant l'algorithme de recuit simulé elles choisissent un voisinage possible. Dans ce cas, les fourmis auront deux voisinages possibles (l'un choisi avec l'évaporateur de phéromone, l'autre avec le recuit simulé), pour choisir entre ces deux solutions on choisit l' une des techniques suivantes :

1- en utilisant une fonction *val* qui a pour but de faciliter la comparaison entre l'évaporation phéromone et la fonction de coût du recuit simulé de la façon suivante :

val(evaporisation de phéromene) > val(ΔC)
où ΔC= COST(*new_place*) - COST(*place*);

On peut aussi ajouter une pondération dans la comparaison pour privilégier un voisinage sur l'autre exemple :

val(evaporisation de phéromene) > A×val(ΔC)

2- ajouter l'évaporation de phéromone dans la fonction de coût, c'est-à-dire au lieu d'avoir la fonction de coût de la forme suivante :

$$F(X, Y, Z, evap) = (X+Y+EXP(Z)) \times Evap$$

Dans cette fonction, les trois paramètres sont utilisés (X : espace, Y : temps et Z : communication). Le paramètre de communication est le plus important, et le paramètre espace et le moins important. Le tout est multiplié par l'évaporation de phéromone Evap.

Les agents dans cette solution sont tout simplement des fourmis, c'est-à-dire au lieu d'appliquer la méta-heuristique d'une façon itérative, ils seront simultanés. Les fourmis dans ce cas seront classifiés comme des agents réactifs parce que, et selon la définition, « un agent réactif, est un agent qui n'a pas besoin d'être intelligent individuellement pour que le système ait un comportement global que l'on puisse qualifier d'intelligent ».

Dans cette solution, les fourmis construisent la solution et le recuit simulé choisit le voisinage de la fourmi [Meignan, 2008].

Algorithme de la solution

Initialiser le graphe, (phéromone 0 et les fourmis affectées aléatoirement aux extrémités de graphe)

Répéter

- Chaque fourmi construit une solution en utilisant la méta-heuristique recuit simulé comme décrit précédemment

- Evaporer les traces de phéromone en les multipliant par un facteur de persistance (agent superviseur)
- Récompenser les meilleures solutions en ajoutant de la phéromone sur leurs composants phéromone (agent superviseur)

Jusqu'à ce qu'un nombre maximal de cycles soit atteint **ou** une solution de qualité acceptable ait été trouvée

Retourner la meilleure solution trouvée

Dans cette approche nous pouvons explorer tout l'espace de solution du problème de partitionnement matériel/logiciel mais d'une façon optimisée à l'aide du recuit simulé. En pourra construire plusieurs solutions et choisir la meilleure solution, de cette façon on est sûr de l'avoir dans un temps d'exécution acceptable.

5 Les SMA et les contraintes distribuées

La résolution des problèmes distribués ou multi-agnts étend des techniques de résolution des problèmes classiques, où plusieurs agents peuvent planifier et agir ensemble. Il existe de nombreuses évolutions récentes dans ce domaine qu'on peut classer sous différentes approches pour les algorithmes de résolution distribuée et la planification distribuée d'exécution des processus. L'une des raisons pour l'utilisation de la résolution distribuée, c'est qu'elle est la manière la plus appropriée pour résoudre un certain type de problèmes. Spécialement ceux où un système de résolution centralisé est irréalisable. Un domaine où cette approche a été utilisée est les problèmes NP-complet CSP. Si un problème CSP est réparti entre un certain nombre d'agents, il est appelé un problème de satisfaction de contraintes distribuées (*distributed constraint satisfaction problem* DCSP) [Vidal, 2006]. Dans un DCSP chaque agent a la responsabilité de gérer la valeur des variables qui ont été attribuées. Les agents ne connaissent pas les valeurs des autres variables, mais ils peuvent communiquer entre-eux pour déterminer la valeur directe de ces variables.

Les problèmes d'optimisation de contraintes distribuées (*distributed constraint optimization problem* (DCOP)) est similaire au problème DSCP sauf que le but est de minimiser la valeur de violation des contraintes. La définition du problème est la suivante :

Soit un ensemble de variables : $x1$, $x2,\ldots,$ xn avec les domaines $D1,$ $D2,\ldots Dn$ un ensemble de contraintes, puis donner des valeurs pour toutes les variables pour que la somme des contraintes soit minimisée.

5.1 L'utilisation des SMA dans la résolution des CSP

L'algorithme décrit dans [Hirayama, 2002] est bien adapté pour la résolution des problèmes de type SAT. L'idée est de distribuer les variables et les contraintes sur plusieurs agents pour transformer le SAT original en un DCSP.

L'algorithme marche comme suit :

- Initialement chaque agent y est assigné à plusieurs variables et clauses de ces variables.
- Puis une procédure de recherche locale est mise au point pour déterminer les valeurs qui donnent une amélioration possible pour la somme des contraintes violées.
- Puis les agents s'échangent ces valeurs pour résoudre n'importe quel conflit rencontré. Cela est fait par réévaluer n'importe quelle valeur qui augmente la somme totale des clauses qui ont été violées par les agents. Le reste des valeurs sera gardé.
- Cela est répété jusqu'à ce qu'une solution soit trouvée.

Cette approche est similaire dans [Hirayama, 2005] et il est différent de l'idée de backtracking utilisée dans ABT [Zivan, 2007] et AWC[Yokoo, 1995], dans ce sens c'est une stratégie « Recherche ascendante » (*hill-claming*) [Zhang, 2002] . Tous les algorithmes de « Recherche ascendante » ont le problème des minima (maxima) locaux, mais ils surmonte ce problème par trouver ce qu'on appelle un quasi minimum local. La définition du quasi-minimum local est la suivante :

Un agent x_i est dans un quasi-minimum local si : il est en violation de quelques contraintes et ni cet agent ni autre agent peut faire un changement qui puisse minimiser le coût total du système [Vidal, 2006].

Autres travaux ont plus tard amélioré cet algorithme pour faire un algorithme de « Recherche ascendante », en appliquant un random walk [Hirayama, 2005]. Cela diminue la probabilité pour l'algorithme de se coincer dans un minimum local (maximum).

Les résultats expérimentaux montrent que les deux algorithmes donnent des résultats au moins aussi bons (et souvent meilleur) que les algorithmes donnés. Les algorithmes améliorés trouvent la solution de 3-SAT, pendant que les anciens ne peuvent pas résoudre les 3-SAT.

L'algorithme décrit dans [Xiaolong, 2002] a beaucoup de similarité avec le précédent. On essaie de résoudre un problème de type SAT par la décomposition des variables en groupes, et

chaque groupe doit être représenté par un agent. Un agent système choisit leurs mouvements dans leur espace de recherche locale en leur assignant l'une des trois stratégies de recherche : déplacement aléatoire, le meilleur déplacement, le plus mauvais déplacement. L'agent système continue la distribution des stratégies sur les agents jusqu'à ce qu'une solution soit trouvée, ou certaine valeur de satisfaction soit atteinte. La différence entre cette approche et la précédente est la façon dont les agents font leur recherche dans l'espace local.

La différence majeure entre les deux algorithmes décrits précédemment et la stratégie de résolution des DCSP est qu'au lieu d'utiliser un agent par variable, on utilise plusieurs (plusieurs variables par agent). Cela mènera à avoir un coût de communication entre agents moins coûteux que les stratégies de résolution classique des DCSP.

La conclusion qu'on peut tirer de ces algorithmes est que tous ces algorithmes font une recherche distribuée, où chaque agent a quelque information locale. Le but est d'avoir tous les agents dans un certain état, pour que l'ensemble des états sera optimal dans le système. L'agent peut à n'importe quel moment prendre une de ses actions disponibles, mais l'utilité de ces actions dépend des actions des autres agents.

Un problème est facile à représenter par un SMA, si le problème a une structure qui facilite la transformation d'un système en un DCSP ou un DCOP. Imaginer un dîner où les invités doivent s'asseoir devant une personne du sexe opposé. Chaque personne doit trouver une place autour de la table, mais la place dépend de l'identité de la personne opposée. Dans cet exemple chaque invité peut être vu comme un agent avec des informations locales, où ses actions dépendent des actions des autres agents. C'est un problème qui peut être facilement transformé en un DCSP.

Appliquer ce genre de SMA peut d'une autre façon être un choix, si le problème est difficile à résoudre. Décomposer les problèmes en des sous-problèmes plus petits peut aider à résoudre le problème rapidement si chaque agent va s'exécuter parallèlement. Peut être que ce n'est pas toujours un avantage, dès que le coût de la communication entre les agents peut alourdir l'exécution des programmes en parallèle. Dans [Hirayama, 2002] ils ont essayé plusieurs agents par variable pour décrémenter le coût de la communication entre agents.

5.2 Les SMA, les contraintes distribuées et le partitionnement

Pour résoudre le problème de partitionnement matériel logiciel en utilisant les DCSP, on doit premièrement définir les métriques à utiliser dans le partitionnement, par exemple on

utilise les métriques suivantes :

Connectivité : cette métrique mesure le nombre des liens entre deux entités.

Communication : mesure la somme des données entre deux entités.

Le partage du matériel : mesure l'ensemble du matériel que deux entités peuvent partager.

Accesseurs communs : quand deux entités (ou variables) sont obtenus via un appel d'une fonction, (ou lecture écriture des variables), par les mêmes autres entités, grouper ces derniers augmentera la performance par la réduction de la communication inter-entité.

Espace : l'espace mémoire pris par cette entité.

Et les clauses :

Les contraintes de communication,

Les contraintes de l'espace.

On peut créer trois agents,

- Un agent pour gérer les communications (calcule les variables de communications et contient aussi les clauses de ces variables)
- Un agent pour gérer l'espace (calcule les variables de l'espace et contient aussi les clauses de ces variables)
- Un agent superviseur pour calculer la fonction de coût et vérifier la violation de ces contraintes, et déterminer si cette entité est matérielle ou logicielle (s'il y a une violation cette entité est logicielle, sinon matérielle).

Dans cette solution nous avons diminué le coût de communication, en diminuant le nombre d'agents (3 agents), où chaque agent a plusieurs variables.

Les performances de cette solution peuvent être vues comme une diminution dans le temps d'exécution du partitionnement mais la qualité reste presque la même. Cette solution s'appuie sur le choix des métriques pour représenter le problème.

6 SMA avec des types différents d'agents

Une autre approche pour utiliser les SMA pour résoudre les problèmes NP-complet, est d'analyser les instances réelles du problème (ou ce que on appelle les éléments du monde réel du problème). De cette manière, déterminer les entités du problème et comment ils interagissent pour résoudre ce problème, pour que ensuite on fasse un SMA inspiré par ces interactions. Cette méthode est très liée à l'abstraction des DCSP et DCOP, dans le sens où le problème est distribué autour de plusieurs agents, avec des responsabilités pour des parties

locales du problème.

Dans la plupart des problèmes NP-complet il n'est pas difficile de déterminer ces éléments là. Un exemple dans [Leong, 2006] « *Vehicle Routing Problem with Time Window* (VRPTW)» est un exemple d'un problème NP-complet, qui a des éléments dans le monde réel. Cela est utilisé pour décrire un SMA pour résoudre le VRPTW, qui sera analysé dans ce qui suit.

6.1 SMA avec des types différents d'agents et le partitionnement

Pour résoudre le problème de partitionnement via des agents de type différent on a besoin de déterminer les entités qui composent ce problème et on cite :

1. Une modélisation graphique des entités à partitionner
2. La représentation de la solution de partitionnement
3. La fonction de coût

6.1.1 Une modélisation graphique des entités à partitionner

Le partitionnement choisi, est le partitionnement fonctionnel.

6.1.2 La représentation de la solution de partitionnement

Le partitionnement peut être considéré comme le mapping de toutes les entités de l'application sur les différents processeurs de l'architecture cible. Pour résoudre ce problème d'affectation qui consiste à tenter de trouver une solution qui minimise la fonction de coût définie par l'utilisateur. La représentation proposée pour résoudre ce problème de mapping est la suivante:

soit *Nbe* le nombre d'entités de l'application sous conception, *Nbp* Le nombre de processeurs de l'architecture cible. La technique de codage consiste à créer un vecteur de cellules *Nbe*, dans lequel chaque entrée correspond à un numéro d'entité. Les entités sont triées dans l'ordre croissant. Chaque cellule de vecteur contient le numéro du processeur auquel l'entité correspondante est affectée au cours du partitionnement.

Exemple: La représentation suivante correspond à une solution de partitionnement problème avec 4 entités (NBE = 4) mappées sur 2 processeurs (NBP = 2).

entités	0	1	2	3
processeurs	0	1	0	1

Tableau 3.6.1 : Exemple de partitionnement

6.1.3 La fonction de coût

Une fonction de coût est utilisée pour évaluer la qualité des solutions générées. Pour la fonction de coût de cette approche, on tient compte des contraintes de coût différentes (L'espace matériel et logiciel), des contraintes de performance (notamment les temps d'exécution d'objet, le temps de l'application global) et de la communication (parce que l'échange d'informations entre les entités différentes d'application est souvent un goulot d'étranglement). Les caractéristiques prises en compte dans cette approche sont donc: le temps d'exécution d'espace, et de la communication.

Les poids sont associés à chaque caractéristique afin de permettre une distinction selon l'importance relative de chaque paramètre.

La fonction de coût aura la forme suivante :

$$F(X, Y, Z)= A\times X+B\times Y+C\times EXP\ (Z)$$

Où :

X : espace, Y : temps et Z : communication

A, B et C sont des valeurs de pondération.

EXP : la fonction exponentielle.

6.2 Résoudre le partitionnement M/L avec les types différents d'agents

Dans cette approche nous avons deux types d'agents :

6.2.1 Les agents matériels

ces agents représentent l'architecture cible, c'est-à-dire que chaque agent représente soit un FPGA, un ASIC, ou autres entités de l'architecture cible, et continent aussi toutes les informations concernant cette entité.

6.2.2 L'agent superviseur

Cet agent contient la fonction de coût, et calcule (et recalcule) la fonction de coût en fonction des déplacements des agents matériels dans le graphe G. Et selon la valeur de cette fonction, il ordonne un agent matériel de changer son choix.

6.2.3 L'algorithme du supervision

- Premièrement on démarre avec la supposition que tous les composants sont logiciels et les agents matériels sont affectés aléatoirement au graphe G (en général à l'extrémité pour construire une solution initiale).

- Chaque agent commence ensuite à chercher une entité logicielle qui correspond à la taille demandée par cet agent (espace logiciel ≥ espace Matériel et $\frac{\text{espace logiciel}}{\text{espace Matériel}} \geq \alpha$

  où :

  - espace logiciel est la taille de l'entité choisie.
  - espace matériel est la taille de matériel représenté par l'agent.
  - α est un nombre réel inclus dans l'intervalle [0 .. 1] introduit par l'utilisateur.

- Quand un agent matériel a choisi une entité, l'agent superviseur calcule la fonction de coût en se basant sur le choix des agents matériels (solution intermédiaire), et calcule aussi le temps d'exécution, et le coût de communication qui résulte de cette solution (cette opération sera faite après chaque choix d'un agent matériel).

- L'agent superviseur ordonne les agents matériels afin de chercher une autre entité possible pour construire une nouvelle solution intermédiaire dans un des cas suivants :

  - Un agent matériel a causé une violation d'une contrainte ou la fonction de coût.
  - Pour commencer la recherche d'une nouvelle solution.

- L'agent matériel applique un algorithme de choix d'une nouvelle entité.

- Si un agent matériel parcourt tout le graphe, et ne trouve pas une nouvelle entité, il envoie un message de fin à l'agent superviseur

- L'agent superviseur stoppe la recherche si :

  - les agents matériels on parcouru toutes les entités logicielles possibles.
  - Au moins un agent ne trouve pas une nouvelle entité dans tout le graphe.

- La fonction de coût a atteint une valeur satisfaisante, cette valeur est introduite par l'utilisateur.

Après la fin de la recherche de solution, l'agent superviseur ne retourne pas la solution avec une meilleure fonction de coût.

6.2.4 L'algorithme de choix de nouvelle entité.

- Chaque agent commence avec une entité initiale (une partie de la solution initiale).

- On cherche ensuite une nouvelle entité où :

$$\frac{\text{espace occupé}}{\text{espace Matériel}} >= \frac{\text{espace partagé}}{\text{espace Matériel}} >= \alpha$$

Et

Cette entité n'est pas choisie par un autre agent matériel

- Envoie cette solution à l'agent superviseur pour qu'il recalcule la fonction de coût, l'agent superviseur à son tour retourne la réponse (Ok si cette solution est acceptée, Non Ok si cette solution n'est pas acceptée).

- Si la réponse reçue est Ok, cet agent entre dans un état inactif jusqu'à ce que l'agent superviseur lui envoie un ordre de construction de nouvelle solution, ou fin de la recherche.

- Si la réponse reçue est Non Ok, cet agent cherchera une nouvelle entité à la prochaine fois.

- Chaque fois qu'un agent matériel trouve une entité acceptable, l'agent superviseur le rend inactif jusqu'à ce que tous les agents matériels deviennent inactifs (construite avec la fonction de coût calculée par l'agent superviseur).

- L'agent superviseur enregistre cette solution (partitionnement matériel/logiciel et la valeur de la fonction de coût)

- L'agent superviseur envoie ensuite un ordre de recherche de nouvelle solution aux agents matériels.

6.2.5 La structure simplifiée de l'agent Matériel

- **Une mémoire** : elle contient la liste des entités logicielles parcourues avec leurs tailles.
- **Propriété** : c'est la propriété du composant matériel que cet agent représente.

6.2.6 La structure de l'agent superviseur

- **Une mémoire :** enregistre la solution trouvée au cours de la recherche de solution avec la valeur de la fonction de coût. Les solutions sont ordonnées par qualité, cette qualité est calculée comme suit :
- La valeur de la fonction de coût pour la solution / la valeur de la fonction de coût idéal
- Calculateur de fonction de coût : pour calcul de la fonction de coût.
- Plus : les deux types d'agents ont un mécanisme de gestion de messages.

Cette approche sera détaillée dans le chapitre qui suit « Conception».

7 Conclusion

Il est évident que les SMA peuvent être utilisés dans la résolution de problème de partitionnement matériel/logiciel. Les différentes applications des SMA présentes dans ce chapitre montrent que les SMA restent un terme abstrait très large.

Dans les SMA et les méta-heuristiques il est important de remarquer, que les avantages de créer une heuristique hybride existe seulement si les différentes parties des méta-heuristiques peuvent être choisies de manière à se complétert les uns les autres. Il est seulement intéressant de créer une hybridation si les effets peuvent être vus, sinon les meilleures parties des heuristiques peuvent être utilisées seules.

Souvent il est évident de voir un problème de partitionnement matériel/logiciel comme un DCSP ou un DCOP, que les avantages d'un algorithme totalement distribué peuvent être utilisés. On affecte les différentes parties des problèmes entre les multiples agents et les laisser s'exécuter d'une façon asynchrone pour s'entre-aider à résoudre le problème. Le dernier type des approches SMA montre, qu'il est aussi possible d'avoir une solution satisfaisante d'un problème de partitionnement matériel/logiciel, par l'utilisation d'une abstraction différant des SMA quand on divise le problème. Cette approche contient un aspect des deux autres approches. Elle combine les fonctionnalités des SMA avec les heuristiques des recherches locales, qui en quelque sorte similaire à la première approche, où on utilise les heuristiques pour améliorer la solution quand l'environnement a changé.

L'utilisation du paradigme SMA lors de la conception des Problèmes NP-Complet (et particulièrement le partitionnement matériel/logiciel) est un domaine d'étude nouveau et en cours d'évolution.

Deuxième partie : Contributions

Chapitre 4 : Conception de SPMA

Dans ce chapitre nous allons présenter la conception de l'application qui vérifie la troisième approche du chapitre précédent (c'est l'approche jugée la plus intéressante). Dans ce chapitre, on va aborder :

- Une brève explication de l'approche.
- L'outil choisi pour réaliser cette application.
- L'architecture de l'application.
- Les entrées de l'application
- Le noyau de l'application (les algorithmes de partitionnement)
- La sortie de l'application.
- Et une conclusion

Pour résoudre le problème de partition matériel/logiciel via les SMA, on opte pour l'utilisation des agents de type différent (voir chapitre précédent). Cette approche consiste à analyser les instances réelles du problème (ou ce que l'on appelle les éléments du monde réel du problème) et les représenter par des agents. Les éléments du partitionnement matériel/logiciel sont :

1. Le système à partitionner : représenté par un graphe.
2. L'architecture matérielle cible : représentée par des composants matériels, ASIC, FPGA, etc.
3. La fonction de coût.

On suppose que tout le système est implémenté en logiciel au départ.

Les agents ici sont : les composants matériels de l'architecture cible, et la fonction de coût.

1 L'architecture de l'application

L'architecture de l'application se compose des éléments suivants :

a. **Les entrées :** sont un ensemble de fichiers XML qui représente :
   - le graphe (le résultat de l'étape précédente) : représente un partitionnement fonctionnel
   - La fonction de coût et les contraintes de partitionnement
   - l'architecture matérielle cible

b. **Le noyau de l'application :** il utilise les fichiers d'entrée pour la configuration du partitionnement, le partitionnement est fait par la collaboration de deux types d'agents (agent superviseur, agents matériel) comme suit :

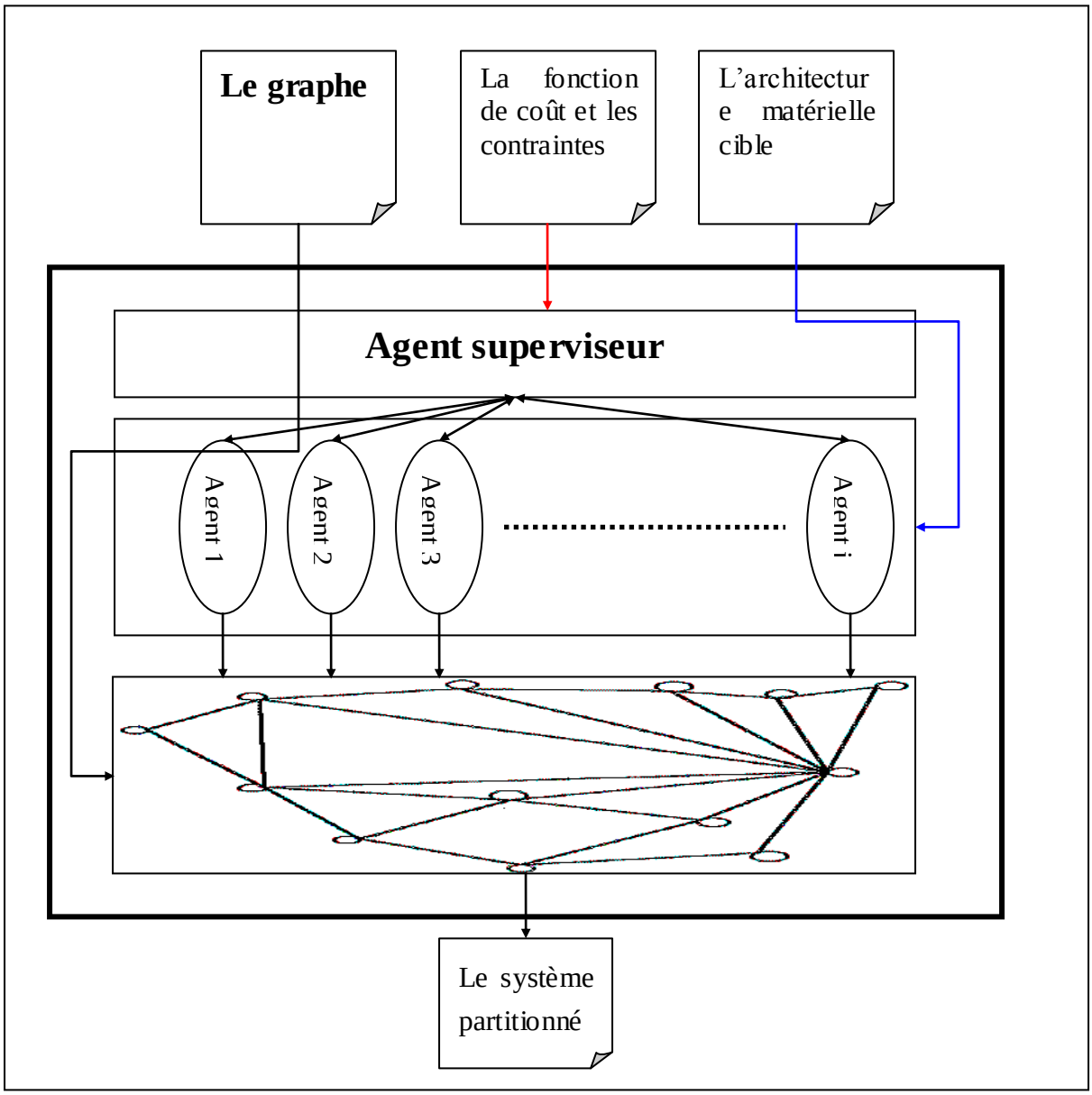

Figure 4.1 : Architecture de l'application

La sortie : de l'application est un fichier XML, qui représente le partitionnement.

2 Les SMA (les agents de l'application)

2.1 Les agents matériels

Ces agents représentent l'architecture cible, c'est-à-dire que chaque agent représente est, soit un FPGA, un ASIC, ou autres entités de l'architecture cible, et contient aussi toutes les informations concernant cette entité. Ce sont des agents simples de type réactif, c'est-à-dire qu'ils réagissent à des données d'entrées. Les agents matériels sont des agents anonymes (un agent matériel ne connait pas les états des autres agents matériels). L'architecture d'un agent matériel est la suivante :

a. Parseur de graphe de spécification : c'est le module responsable du parcours du graphe, (comment visiter un nœud ?)

b. Lecteur des données du nœud : ce module lit les données du nœud visité.

c. Les informations de matériel : contient les informations du matériel représenté par cet agent.

d. Calculateur de compatibilité : calcule la compatibilité du nœud vis-à-vis du matériel représenté par l'agent matériel de la façon suivante :

$E_L >= E_M$ et $\frac{\text{espace logiciel}}{\text{espace Matériel}} >= \alpha$ où :

- E_L : la taille de l'entité choisie.
- E_M : la taille de matériel représenté par l'agent.
- α : le nombre réel inclus dans l'intervalle [0 .. 1] introduit par l'utilisateur.

e. Mémoire de l'agent : elle contient la liste des entités logicielles parcourues avec leurs tailles, c'est-à-dire la trace de cet agent.

f. Boite aux lettres de l'agent : c'est le module responsable des échanges des données avec l'agent superviseur.

g. Moteur de déplacement : c'est le module responsable de la visite du nœud, (quel nœud visité ?)

Remarque :

Les agents matériels visent principalement les gros grains.

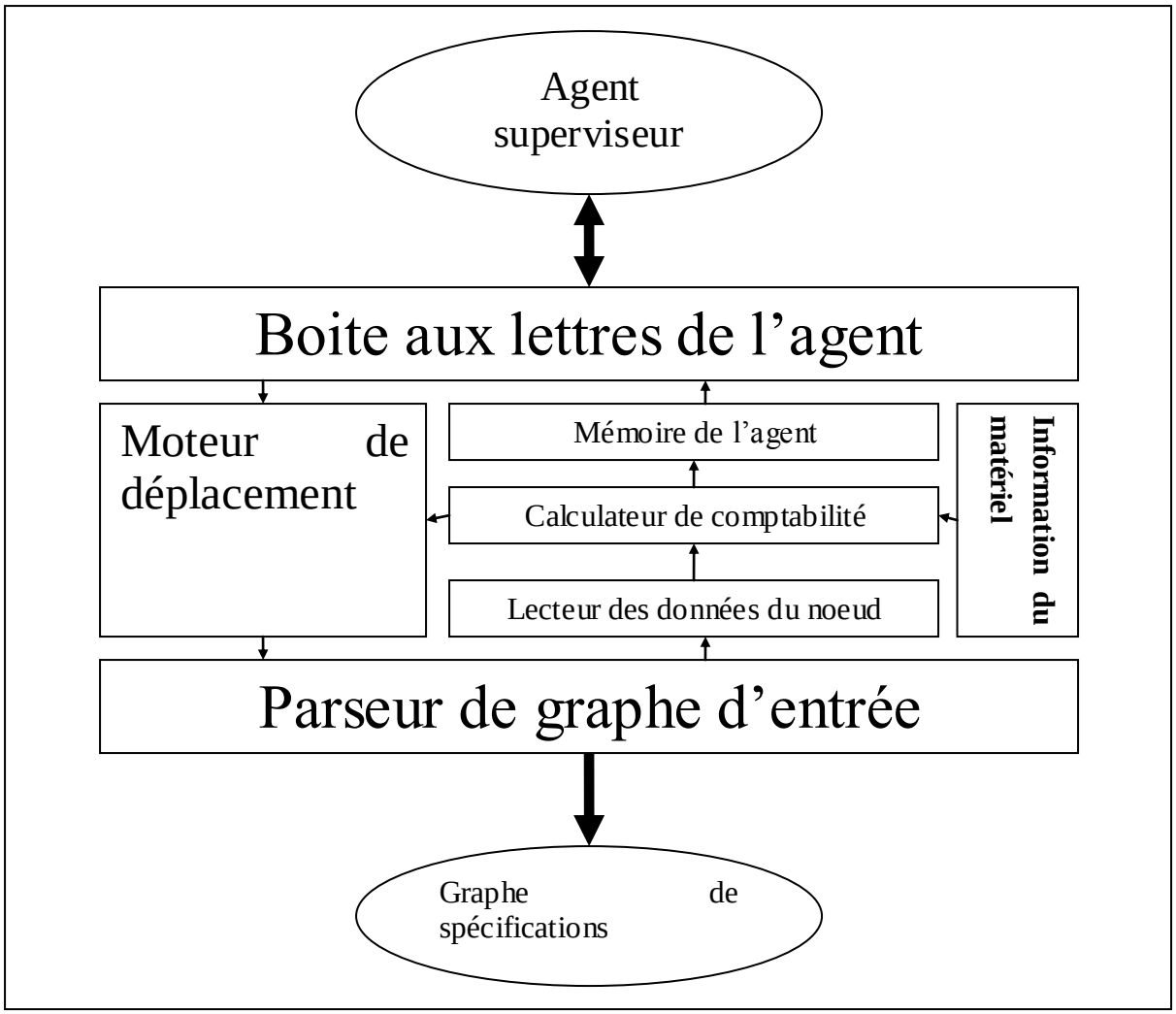

Figure 4.2 : Architecture de l'agent matériel

L'agent matériel suit l'algorithme suivant :

Initialisation :

Les agents matériels sont créés au nombre des composants matériels de l'architecture cible, chaque agent matériel contient toutes les informations du matériel en question.

On démarre avec une partition initiale totalement logicielle.

Un graphe G=(E, C),

où E : est l'ensemble des sommets du graphe qui représente les entités logicielles du système à mettre en œuvre,

et C : c'est l'ensemble des arcs du graphe et ils représentent le coût de la communication entre les entités logicielles.

Les agents matériels seront posés aléatoirement sur le graphe.

Répéter

Lecture des données de l'entité en cours

Si (l'entité est compatible avec le matériel) alors

Envoi d'un message à l'agent superviseur contenant :

- L'identité de l'agent émetteur de message.
- La taille de cette entité.
- Les coûts de communication à cette entité : c'est la valeur associée à l'arc qui relie l'entité précédente à celle-ci.
- Le temps d'exécution probable de cette entité par le matériel : il est calculé on utilisant la taille de l'entité et la configuration matérielle de l'agent.

Attente de la réponse de l'agent superviseur

Réception de la réponse de l'agent superviseur

Lecture de message

Finsi

Sinon

Choisir une autre entité pas encore visitée avec un coût de communication minimum.

Aller à cette entité

Vérifier si l'entité est déjà visitée.

Fin sinon

Jusqu'à ce que : l'agent ait visité toutes les entités ou l'agent superviseur ordonne l'arrêt.

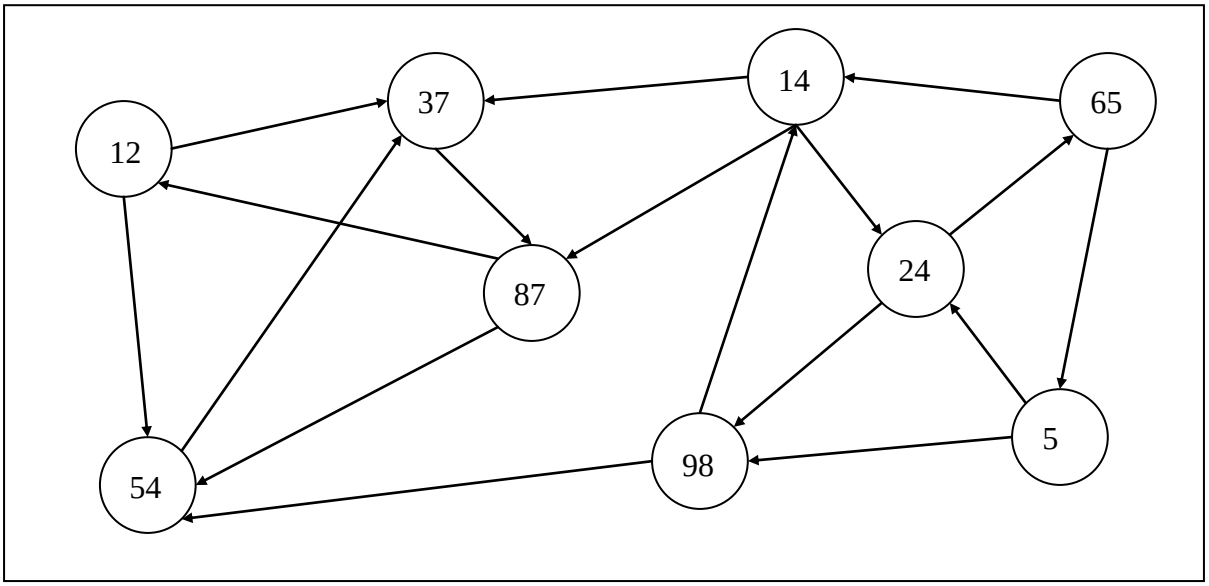

Figure 4.3 : Le graphe

2.2 L'agent superviseur

C'est l'agent qui supervise les choix des agents matériels, et il calcule la fonction de

coût global. Cet agent décide si l'entité choisie par un agent matériel est bonne pour construire une solution ou pas. Contrairement à l'agent précédent, cet agent est de type « *agent intelligent* » (on peut le nommer aussi agent interface parce qu'il joue le rôle de l'interface entre le système de partitionnement est l'utilisateur de ce système).

L'agent superviseur se compose de :

a. **Une boite aux lettres** : pour l'échange des messages entre lui et les agents matériels, lors de la lecture des messages, l'agent superviseur ne sait pas qui a envoyé le message (dés le début il va le savoir après la lecture du message).

b. **Lecteur et Analyseur de message** : c'est le module qui lit les messages des agents matériels et les analyse, et selon le contenu soit il les envoie pour les constructions d'une nouvelle solution (un message d'un nouveau déplacement), soit il envoie le message au module contrôle pour qu'il prenne la décision de continuer la recherche ou d'arrêter.

c. **Un constructeur de solution** : il a pour but d'attendre jusqu'à ce que tous les agents matériels aient choisi une entité logicielle pour construire une solution intermédiaire.

d. **Le calculateur de la fonction de coût** : il calcule la fonction de coût de la solution construite

e. **Mémoire des solutions** : un module pour la mémorisation de solutions intermédiaires.

f. **Analyseur de fonction de coût et de la solution intermédiaire** : il analyse la valeur de la fonction du coût, et la solution intermédiaire, et vérifie s'il y a respect des contraintes prédéfinies.

g. **Le contrôleur** : C'est le module qui décide si un agent matériel doit accepter une entité logicielle ou pas, et il est responsable d'accepter ou de refuser la solution intermédiaire. Et c'est ce module là qui décide si la solution intermédiaire et satisfaisante que la recherche peut être terminé.

h. Le sélecteur de meilleure solution : c'est le module qui enregistre la meilleure solution parmi les solutions intermédiaires enregistrées en mémoire.

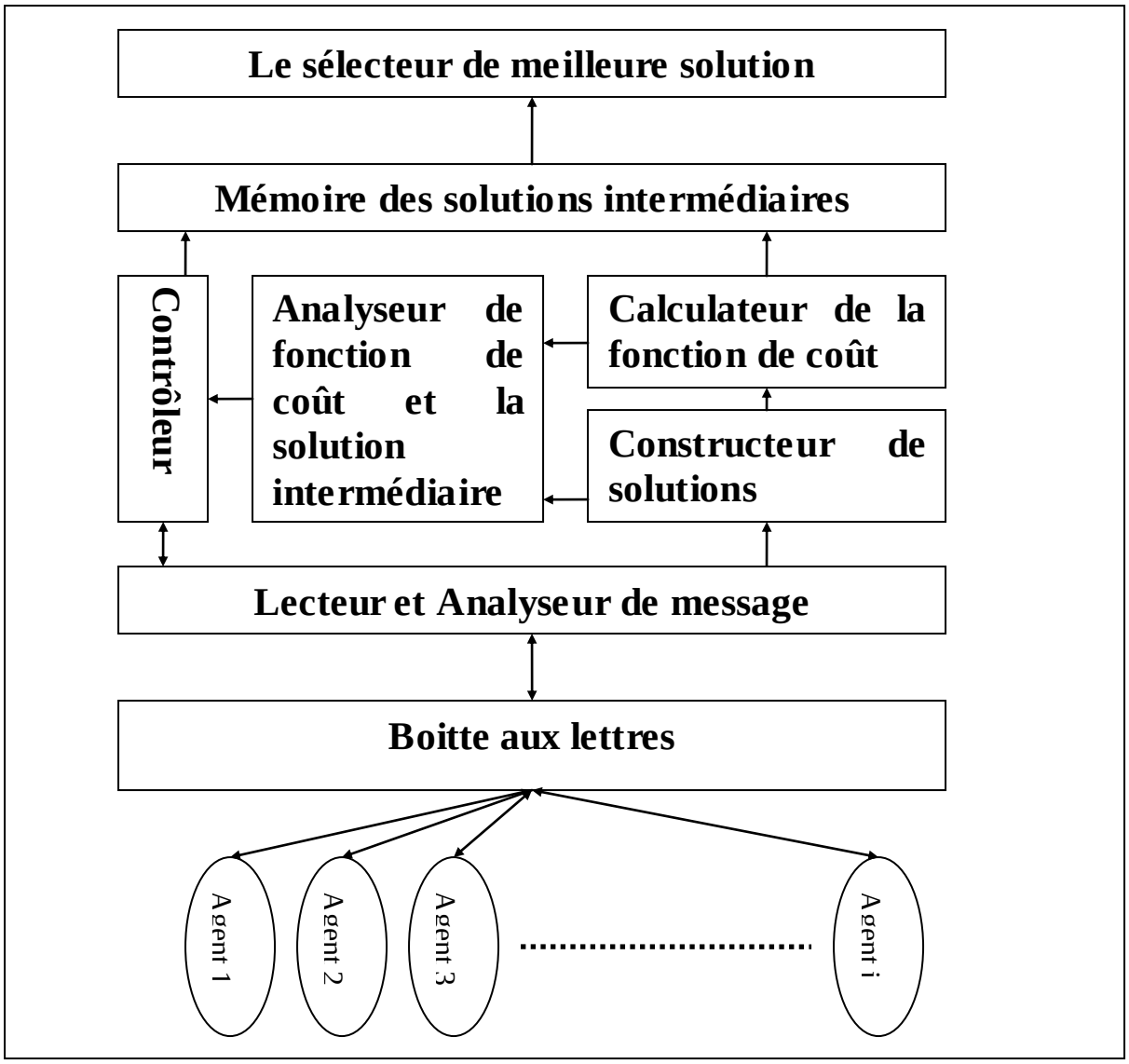

Figure 4.4 : Agent superviseur

L'agent superviseur suit l'algorithme suivant :

Initialisation :

On donne la fonction de coût et les contraintes à respecter à l'agent superviseur.

L'agent superviseur démarre les agents matériels (envoie un message de démarrage aux agents).

Répéter

Lecture du message de la boite aux lettres.

Analyser le message ;

Si (message est un nouveau déplacement pour un agent matériel)

Construire la nouvelle solution

Calculer la nouvelle fonction coût

Analyser la solution et la fonction de coût

Si (la solution est acceptée)

Enregistrer la solution

Fin Si

Sinon

Envoyer un message à l'agent pour choisir une entre entité logicielle

FinSinon

FinSi

Sinon

/* un agent n'a pas trouvé une entité logicielle adéquate */

Envoyer un message à cet agent pour lui demander de prendre la meilleure entité logicielle parcourue

Arrêter cet agent.

Fin sinon

Jusqu'à obtenir une solution est satisfaisante **ou** un nombre d'agents matériels atteint (ce nombre est défini par l'utilisateur par défaut 1) ne trouvent pas une meilleure entité logicielle **ou** arrêt par l'utilisateur.

3 Les entrées

On a choisi le langage XML pour représenter les entrées de l'application. Parce que le XML est un langage puissant dans la représentation des données, des métadonnées, et la configuration. Et nous allons utiliser ce langage pour représenter les données d'entrée : les contraintes générales et le système à partitionner (le graphe lui-même). La fonction de coût est définie dans l'agent superviseur.

3.1 Le système à partitionner

Le système à partitionner est représenté par un graphe orienté qui représente un partitionnement fonctionnel ***pré-partitionné***. Le pré-partitionnement a pour but de réduire le nombre de grains (en général des procédures), par la fusion des grains dont la séparation (leur implémentation dans des processeurs matériels différents) nous donnera une mauvaise solution [Vahid, 1998]. Cette étape se distingue de la sélection de granularité par le fait que les procédures regroupées ici peuvent ne pas être tels qu'ils auraient pu être en une seule procédure nouvelle.

Ci-joint le XSD du XML du graphe d'entrée:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="graphe">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="nodes">
                              <xs:complexType>
                                    <xs:sequence>
<xs:element name="node" minOccurs="1" maxOccurs="unbounded">
                                                <xs:complexType>
      <xs:attribute name="id" type="xs:int" use="required" />
      <xs:attribute name="size" type="xs:int" use="required" />
                                                </xs:complexType>
                                          </xs:element>
                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="neighborhood">
                              <xs:complexType>
                                    <xs:sequence>
<xs:element name="communication" minOccurs="1" maxOccurs="unbounded">
                                                <xs:complexType>
      <xs:attribute name="id" type="xs:int" use="required" />
      <xs:attribute name="node1ID" type="xs:int" use="required" />
      <xs:attribute name="node2ID" type="xs:int" use="required" />
      <xs:attribute name="com" type="xs:int" use="required" />
      <xs:attribute name="frequency" type="xs:int" use="required" />
                                                </xs:complexType>
                                          </xs:element>
                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

3.2 La fonction de coût et les contraints matérielles et logicielles

Ci-joint le fichier XSD du XML de la fonction de coût et les contraintes matérielles et logicielles :

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="constraints">
            <xs:complexType>
                  <xs:sequence>
<xs:element name="costFunction" minOccurs="1"  maxOccurs="1">
                              <xs:complexType>
                                    <xs:sequence>
                                          <xs:element name="time">
                                                <xs:complexType>
<xs:attribute name="weight" type="xs:float" use="required" />
                                                </xs:complexType>
                                          </xs:element>
                                          <xs:element name="communication">
                                                <xs:complexType>
```

```
                                                      <xs:attribute
name="weight" type="xs:float" use="required" />
                                                </xs:complexType>
                                          </xs:element>
                                          <xs:element name="space">
                                                <xs:complexType>
                                                      <xs:attribute
name="weight" type="xs:float" use="required" />
                                                </xs:complexType>
                                          </xs:element>
                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="constraint" minOccurs="1"
maxOccurs="1">
                              <xs:complexType>
                                    <xs:sequence>
                                          <xs:element
name="communicationConstraint">
                                                <xs:complexType>
                                                      <xs:attribute
name="maxValue" type="xs:int" use="required" />
                                                </xs:complexType>
                                          </xs:element>
                                          <xs:element
name="spaceConstraint">
                                                <xs:complexType>
                                                      <xs:attribute
name="maxValue" type="xs:int" use="required" />
                                                </xs:complexType>
                                          </xs:element>
                                          <xs:element name="timeConstraint">
                                                <xs:complexType>
                                                      <xs:attribute
name="maxValue" type="xs:int" use="required" />
                                                </xs:complexType>
                                          </xs:element>
                                          <xs:element name="alpha">
                                                <xs:complexType>
                                                      <xs:attribute
name="value" type="xs:float" use="required" />
                                                </xs:complexType>
                                          </xs:element>
                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:schema>
```

La fonction de coût sera calculée au niveau de l'agent superviseur de la façon suivante :

$$CostFct = \sum_{i=1}^{N} CostAgent(i)$$

Où :

N : le nombre d'agents matériels

CostAgent : c'est la valeur de la fonction de coût local pour chaque agent :

$CostAgent = A \times Time + B \times Space + C \times Communication$

Où

A,B et C : des constantes de pondération

Time, space et communication : les critères de partitionnement.

$$Communication = \sum_{I=1}^{N} val \times freq$$

Où :

N : le nombre de voisins

Val : le nombre les bits échangés à la fois entre deux voisins

Freq : la fréquence d'échange

Les contraintes à respecter

Les contraintes matérielles sont vérifiées comme suit :

$$TimeConstraint = valMin < \sum_{i=1}^{N} Time_i < valMax$$

Où : $Time_i$ est le temps d'exécution de l'entité choisie par l'agent i, sur le matériel représenté par cet agent.

$$SpaceConstraint = valMin < \sum_{i=1}^{N} Space_i < valMax$$

Où : $Space_i$ est la taille de l'entité choisie par l'$agent_i$

$$CommunicationConstraint = valMin < \sum_{i=1}^{N} Communication < valMax$$

Où : $Comunication_i$ est le coût de la communication de l'$entité_i$ choisie par l'$agent_i$

3.3 L'architecture cible

Ce sont les processeurs de l'architecture matérielle cible.

Ci-joint le XSD du XML qui va représenter les composants matériels :

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="hardeWare">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element name="prcessor">
                              <xs:complexType>
                                    <xs:attribute name="id" type="xs:int"
use="required" />
                                    <xs:attribute name="type"
type="xs:string" use="required" />
                                    <xs:attribute name="clockSpeed"
type="xs:float" use="required" />
```

```
                                        <xs:attribute name="space"
type="xs:float" use="required" />
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                    </xs:complexType>
            </xs:element>
</xs:schema>
```

4 Les sorties

En sortie et après le partitionnement, on a seulement le système partitionné. La représentation XML de la sortie est presque la même que celle du graphe d'entrée mais l'élément *node* est modifié comme suit :

```
<xs:element name="nodes" minOccurs="1"  maxOccurs="unbounded" >
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name="node">
                                                        <xs:complexType>
<xs:attribute name="id" type="xs:int" use="required" />
<xs:attribute name="size" type="xs:float" use="required" />
<xs:attribute name="implementation" type="xs:string" use="required" />
<xs:attribute name="hardwareID" type="xs:int" use="required" />
                                                        </xs:complexType>
                                                </xs:element>
                                        </xs:sequence>
                                </xs:complexType>
</xs:element>
```

5 Conclusion

Si on examine bien l'algorithme, l'architecture et les composants de cette approche, on voit qu'elle satisfait bien les deux points qu'on a voulu : L'exploration et l'exploitation

L'exploration est assurée par les agents matériels, qui parcourent tout le graphe ou au moins une bonne partie de ce graphe, ce parcours et optimisé parce que l'agent matériel ne visite qu'un nœud de coût de communication minimale. Et il ne choisit qu'un nœud qui respecte certaines contraintes (pour le moment c'est l'espace mais on peut ajouter autre contraintes)

L'exploitation est assurée par l'agent superviseur. Cet agent a pour but d'exploiter les données envoyées par les agents matériels pour construire une solution, il *exploite* les recherches locales de chaque agent matériel.

Chapitre 5 : Implémentation et expérimentation de SPMA

Pour l'implémentation de notre application appelée S.P.M.A (**S**ystème de **P**artitionnement M /L **M**ulti **A**gent) , nous avons utilisé les outils suivants :

1 MadKit[2]

MadKit est une plateforme de développement de systèmes multi agents destinée au développement et à l'exécution de systèmes multi agents et plus particulièrement à des systèmes multi agents fondés sur des critères organisationnels ***agent, groupes et rôles*** ou ***AGR***.

MadKit n'impose aucune architecture particulière aux agents. Il est ainsi possible de développer aussi bien des applications avec des agents réactifs, que des agents cognitifs et communicationnels, et même de faire interagir aisément tous ces types d'agents.

MadKit est construit autour du concept de « **micronoyau** » et « **d'agentification de services** ». Le *micronoyau* de MadKit est très petit moins de 100 Ko de code, car il ne gère que les organisations groupes et rôles et les communications à l'intérieur de la plateforme. Le mécanisme de distribution, les outils de développement et de surveillance des agents, les outils de visualisation et de présentation sont des outils supplémentaires qui viennent sous la forme d'agents que l'on peut ajouter au noyau de base.

De ce fait, MadKit peut être utilisé aussi bien comme outil de développement d'applications que comme un noyau d'exécution de système multi agent qui peut être embarqué dans des

[2] Téléchargeable depuis le site http://www.madkit.org/ ,

applications quelconques.

MadKit est un logiciel libre de type « Open Source » avec une licence mixte GPL/LGPL. LGPL pour le micronoyau et les outils de communication, et GPL pour les outils de développement.

On reconnaît généralement à MadKit les qualités suivantes:

- Simplicité de mise en œuvre et de prise en main,
- Intégration très facile de la distribution au sein d'un réseau.
- L'aspect pratique et efficace des concepts organisationnels pour créer différents types d'applications
- Hétérogénéité des applications et des types d'agents utilisables: on peut faire tourner sur MadKit aussi bien des applications utilisant des agents réactifs simples de type fourmis (plus de 50 000 agents ayant un comportement simple ont tourné en MadKik), que des applications disposant d'agents cognitifs sophistiqués.

2 Eclipse IDE

Eclipse est un environnement de développement intégré libre extensible, universel et polyvalent, permettant de créer des projets de développement mettant en œuvre n'importe quel langage de programmation. Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT d'IBM), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions.

La spécificité d'Eclipse IDE vient du fait de son architecture totalement développée autour de la notion de plug-in (en conformité avec la norme OSGi) : toutes les fonctionnalités de cet atelier logiciel sont développées en tant que plug-in.

Il est utilisé ici pour la compilation et le développement vu que Madkit et limilté dans ce aspect.

3 NetBeans

NetBeans est un projet open source ayant un succès et une base d'utilisateur très large, une communauté en croissance constante, et près de 100 partenaires mondiaux et des centaines de milliers d'utilisateurs à travers le monde. Sun Microsystems a fondé le projet open source

NetBeans en Juin 2000 et continue d'être le sponsor principal du projet.

Il est un environnement de développement intégré (IDE) open source. Il est développé par Sun et se trouve sous licence CDDL (Common Development and Distribution License). En plus de Java, NetBeans permet également de développer avec d'autres langages tels que : Python, C, C++, Ruby, XML, PHP et HTML.

Il comprend toutes les caractéristiques d'un IDE moderne (coloration syntaxique, projets multi-langages, refactoring, éditeur graphique d'interfaces et de pages web, etc).

NetBeans est utilisé pour la création des Interfaces graphiques, à cause de son éditeur graphique puissant.

4 JUNG (the Java Universal Network/Graph Framework)

JUNG est une librairie open source permettant la modélisation, l'analyse et la visualisation de données pouvant être représentées sous forme de graphe ou de réseau.

Jung est un Framework qui fournit toutes les classes nécessaires à la représentation, la manipulation, l'analyse et la visualisation de données représentées sous la forme de graphes et de réseaux. De nombreux algorithmes de graphes et réseaux sont également implémentés et fournis dans le framework.
Elle est utilisée pour le dessin des graphes (pré partitionné et partitionné).

5 L'architecture de l'application

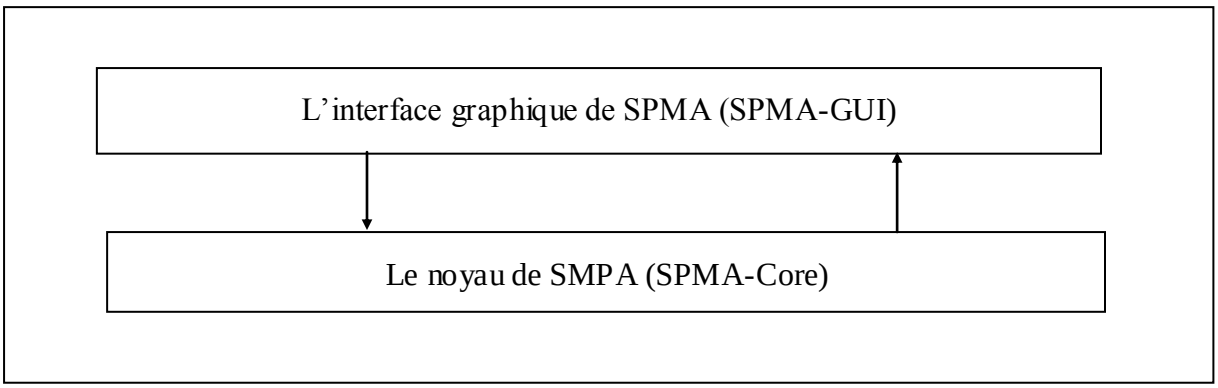

Figure 5.1 : L'architecture de SPMA

L'architecture de SPMA-Core a été décrite dans le chapitre précèdent.

5.1 L'architecture de SPMA-GUI

SPMA-GUI contient les modules suivants :

1. Lecture des fichiers : il lit les fichiers XML d'un emplacement dans le disque, et l'envoie vers l'affichage textuel.
2. Affichage textuel : il affiche les fichiers XML, et il assure : un syntaxe colorée, un code XML formaté et indenté, les numéros de ligne, etc.
3. Edition des fichiers : il permet les opérations primitifs d'édition de fichier : Copier, Couper, Coller, Effacer, Sélectionner tout, etc. Et il y a aussi la fonction de recherche dans le texte.[2]
4. Impression des fichiers : Impression des fichiers XML introduite à l'aide d'un outil de prévisualisation de l'impression.
5. Affichage graphique : il convertit les fichiers XML en un graphe.

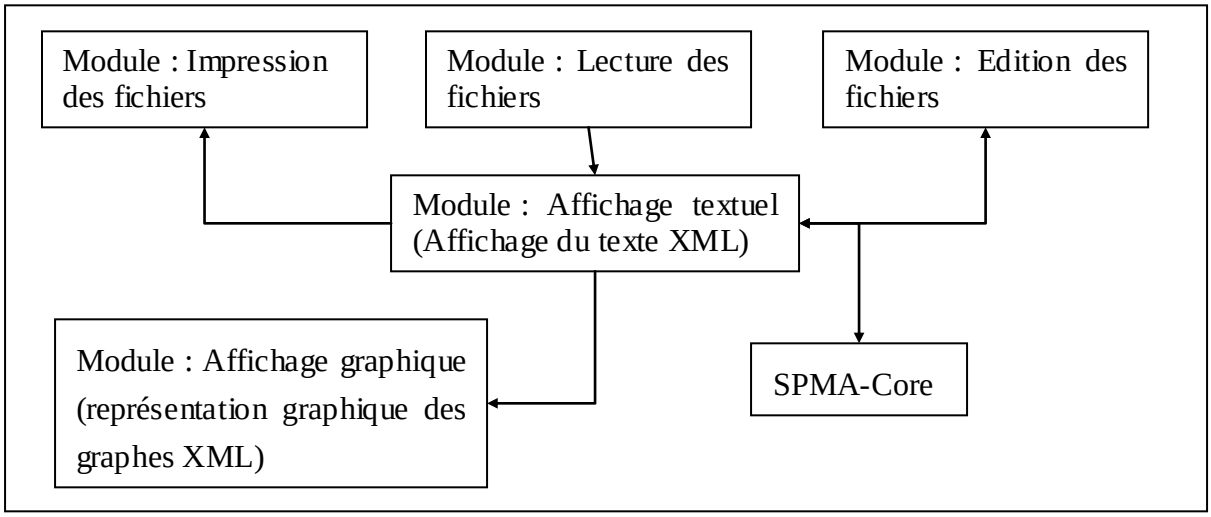

Figure 5.2 : Architecture de SPMA-GUI

6 Construction de l'application

Vue que l'application est composée de deux parties (SPMA-Core et SPMA-GUI), et chacune de ces deux parties est réalisée avec différents outils : SPMA-Core avec MadKit et Eclipse, SPAM-GUI avec Netbeans et JUNG. Alors pour avoir une version qui intègre les deux nous devons suivre les étapes suivantes.

6.1 Etape 1 : la création du standealone de SPMA-Core

Un Standalone un seul fichier archive (en général un zip) qui contient toutes les librairies (des

jar), en théorie il peut être exécuté seul sans avoir besoin de autre applications, mais on la crée parce qu'il importe les jar nécessaires pour l'intégrer à l'application hôte, dans notre cas c'est SPMA-GUI. Pour créer le standalone, on ouvre MadKit, et on applique l'action « Ant target : standalone dist » de menu contextuel du fichier de build Ant de SPMA-Core, cela créera un zib (SPMA-1.0.0-appli.zip) qui contient, un fichier batch (SPMA.bat) pour windows, un fichier shell (SPMA.sh) pour les systèmes à base de UNIX. Et un dossier lib qui contient les librairies de l'application.

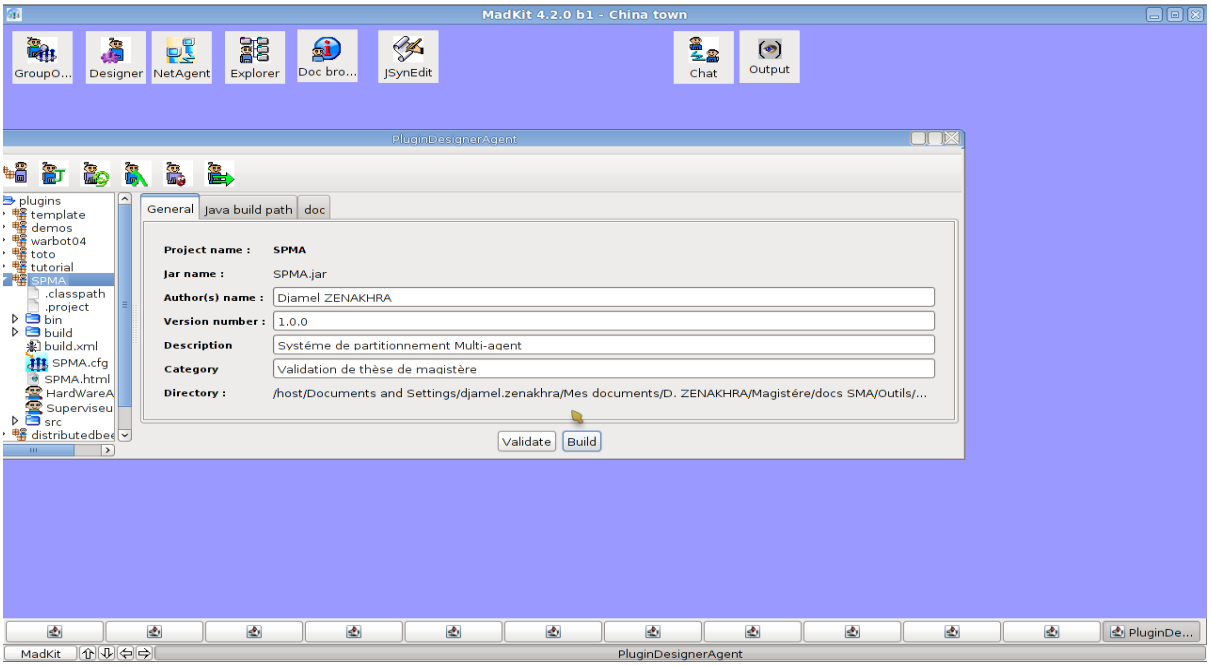

Figure 5.3 : Madkit

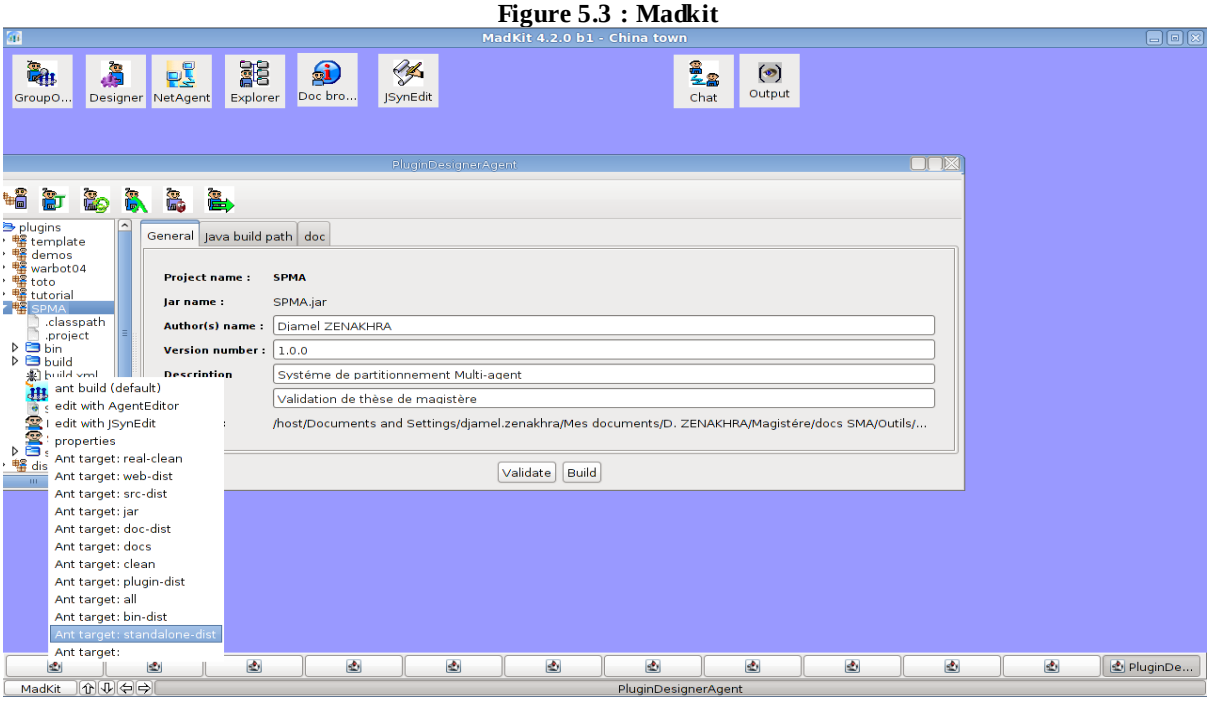

Figure 5.4 : Le menu de création de standalone de SPMA

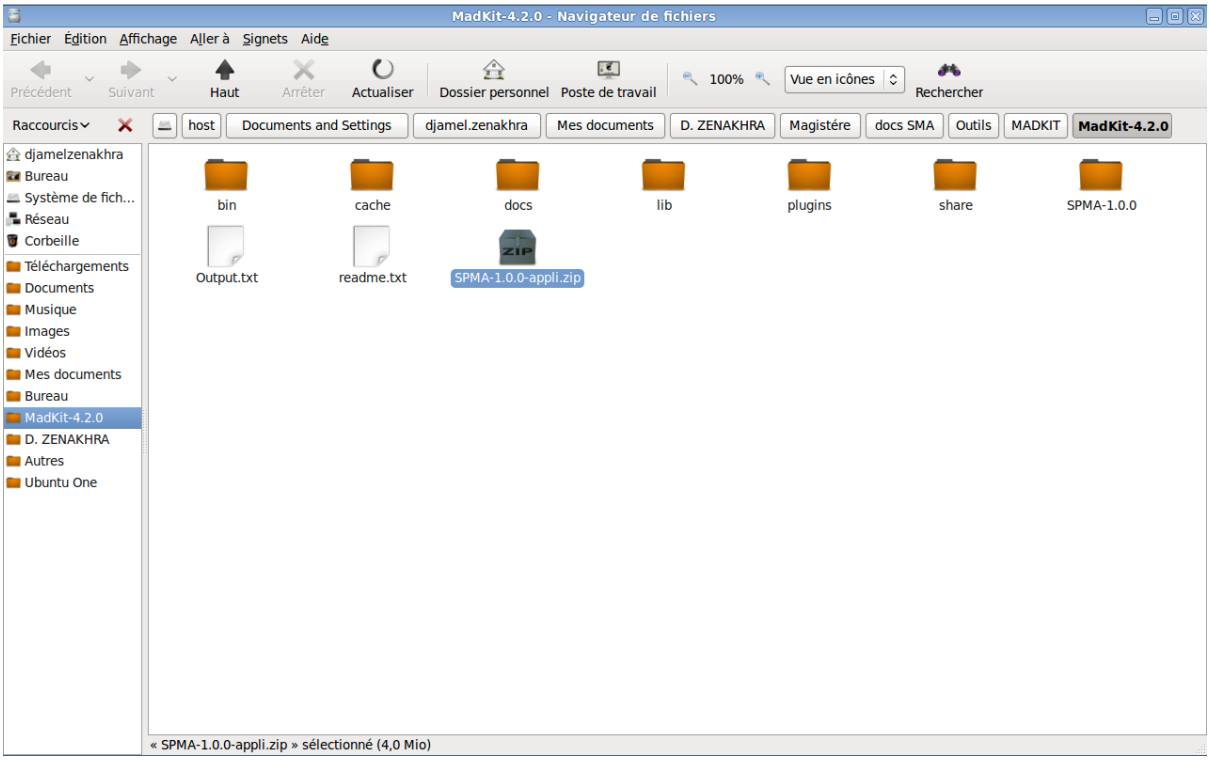

Figure 5.5 : Le fichier Standalone

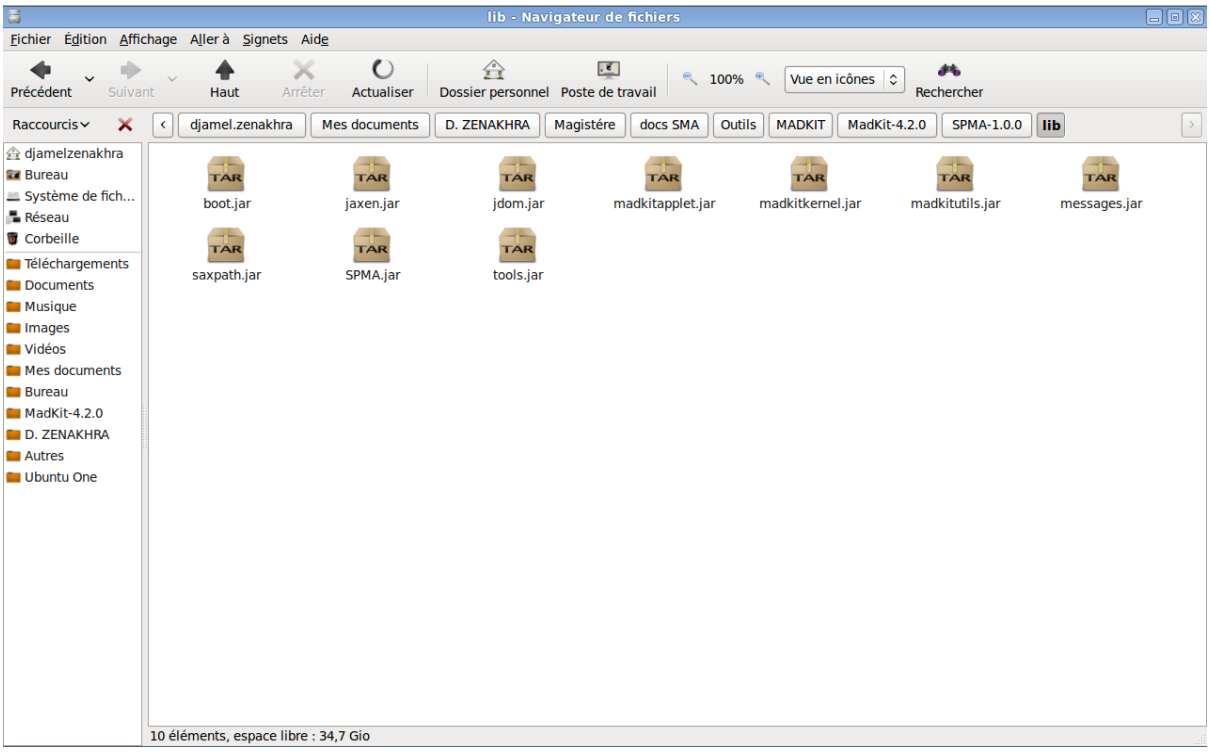

Figure 5.6 : Les lib de l'application standalone

6.2 Etape 2 : l'intégration des librairies dans SPMA-GUI

Ensuite, on intègre les librairies dans les SPMA-GUI, et on fait les appels nécessaires dans l'appel des événements adéquats.

7 Le fonctionnement de SPMA-GUI

L'interface de SPMA (SPMA_GUI) est réalisée d'une façon ergonomique et moderne, comme le montre l'interface suivante :

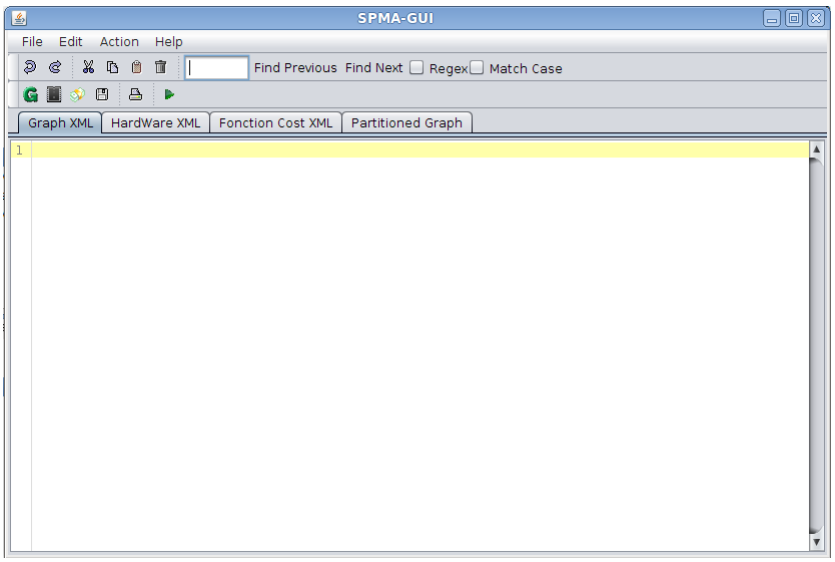

Figure 5.7 : SPMA-GUI

Elle est composée principalement des menus suivants :

7.1 Le menu File

Il sert au chargement des fichiers d'entrée (chargement du fichier graphe « Load Graph », chargement de la configuration hardware cible « Load HardWare » et chargement de la fonction de coût « Load Cost Function »), et l'enregistrement de graphe après le partitionnement, l'impression des documents XML (entrées et sorties).

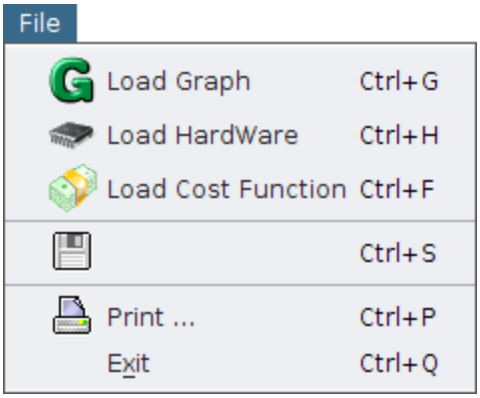

Figure 5.8 : Le menu File

7.2 Le menu edit

Il englobe les opérations d'édition (annuler action (UnDO), refaire action (Redo), couper

(Cut), Copy (Copier), Past (Coller), Erase (Effacer), Erase All (Effacer Tout), Sélectionner Tout(Select All))

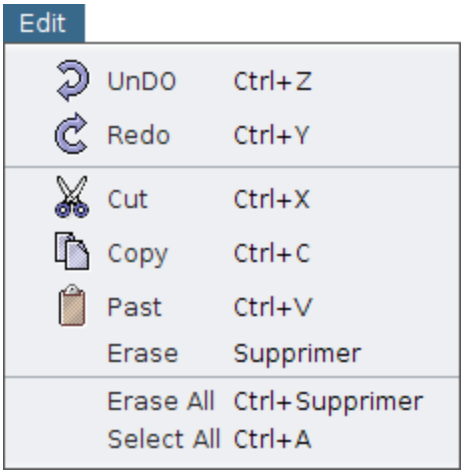

Figure 5.9 : Le menu Edit

7.3 Le menu action

Les Actions principales de l'application sont :

Start Partitioning : pour commencer le partitionnement dans cette action (et seulement dans cette action) SPMA-GUI appelle SPMA-Core avec les entrées , le graphe, les composants cibles et la fonction de coût, et retourne le graphe partitionné.

Visualize Presq-Partitioned Graph : affiche le graphe d'entrée sous forme graphique.

Visualize Partitioned Graph : affiche le graphe de sortie sous forme graphique.

Show Statistics : affiche des statistiques de l'opération de partitionnement.

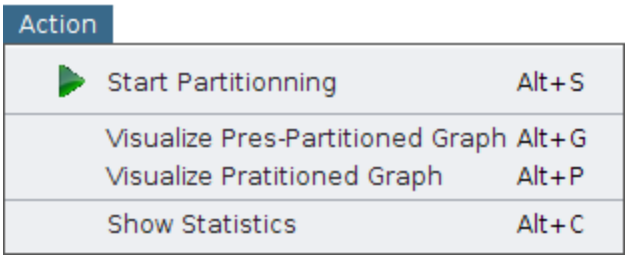

Figure 5.10 : Le menu Action

7.4 Le menu help

Help : affiche une fenêtre d'aide (pas encore implémentée)

About SPMA : affiche une fenêtre de présentation de l'application.

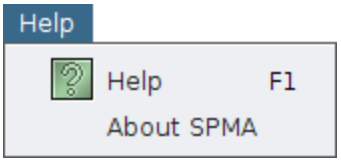

Figure 5.11 : Le menu Help

7.5 Le chargement des fichiers d'entrée

Le chargement des fichiers d'entrée sera fait par une fenêtre de dialogue comme celle décrite dans la figure suivante :

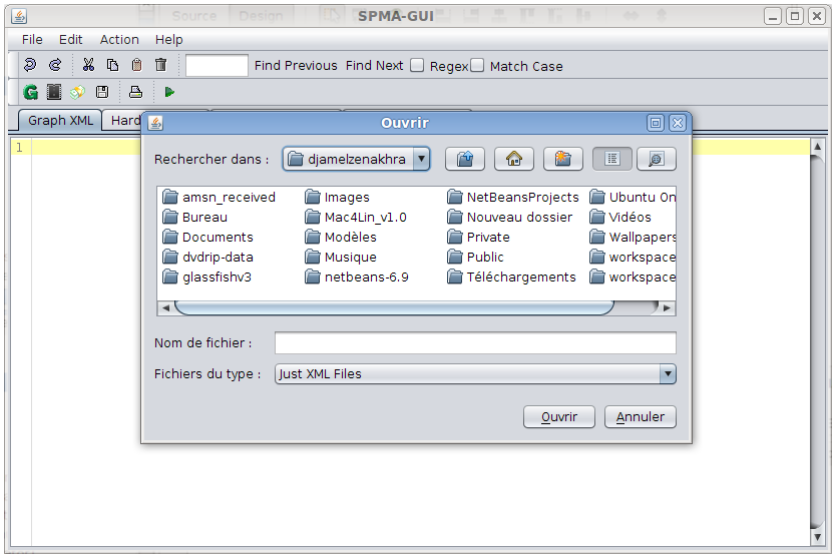

Figure 5.12 : La fenêtre de dialogue de chargement des fichiers d'entree

Le graphe d'entrée sera affiché dans le premier onglet.

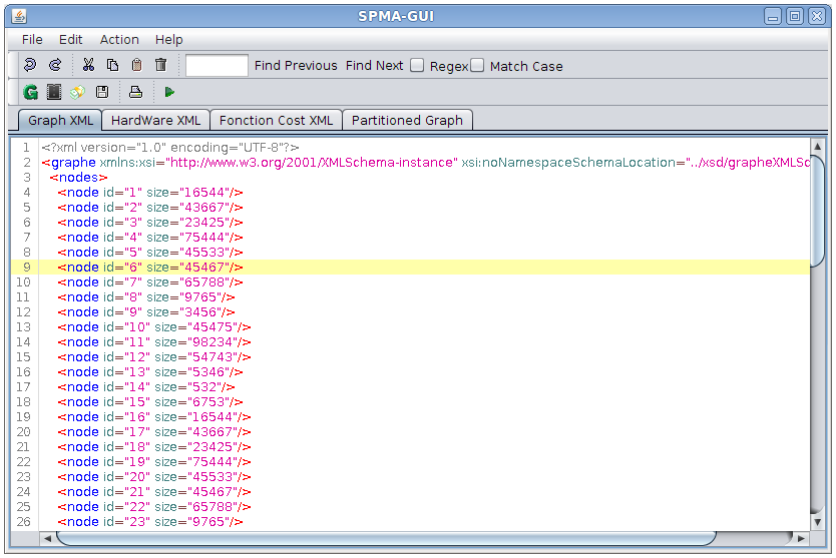

Figure 5.13 : Le graphe d'entrée

La configuration matérielle d'entrée sera affichée dans le deuxième onglet.

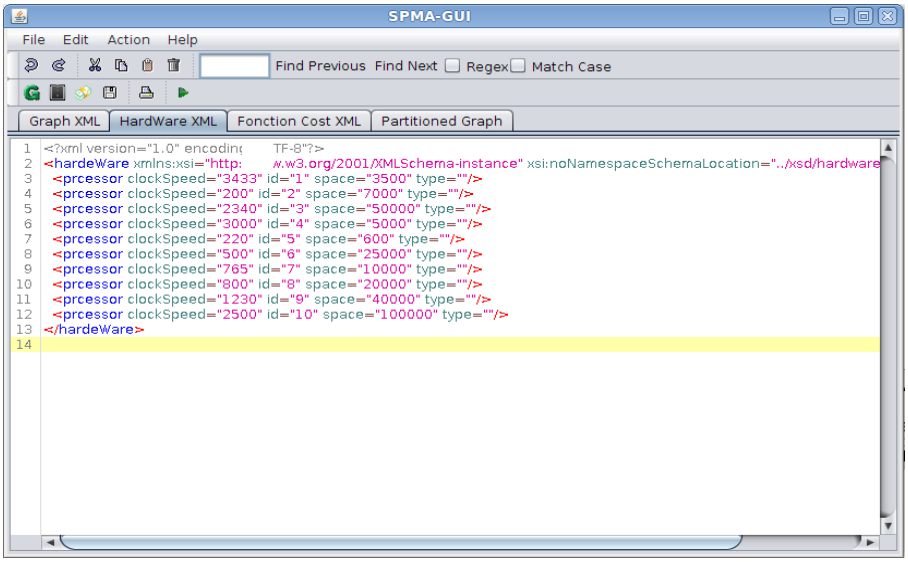

Figure 5.14 : La configuration matérielle

La fonction de coût d'entrée sera affichée dans le troisième onglet.

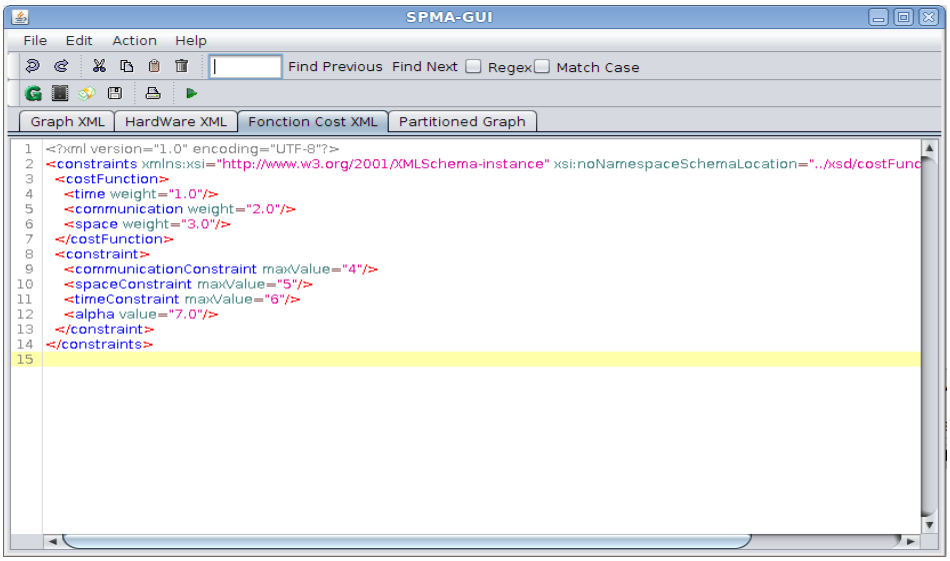

Figure 5.15 : La fonction de coût

7.6 Le partitionnement

Le partitionnement débutera après le lancement de, soit le menu de Action > Start Partitionning soit on appuie sur le bouton qui porte la même icône, dans la barre d'outils. Un écran d'attente masquera l'application pendant le partitionnement, le temps de partitionnement. En plus, on trouve aussi :

- La conversion de document XML d'entrée (le graphe) à des objets JAVA.
- La création des Agents Matériels en fonction de fichier de Hardware
- La création de l'agent Superviseur en fonction de fichier de fonction de coût.
- Le lancement des agents et commencement de partitionnement

- La conversion du résultat en un document XML, et le retourner.

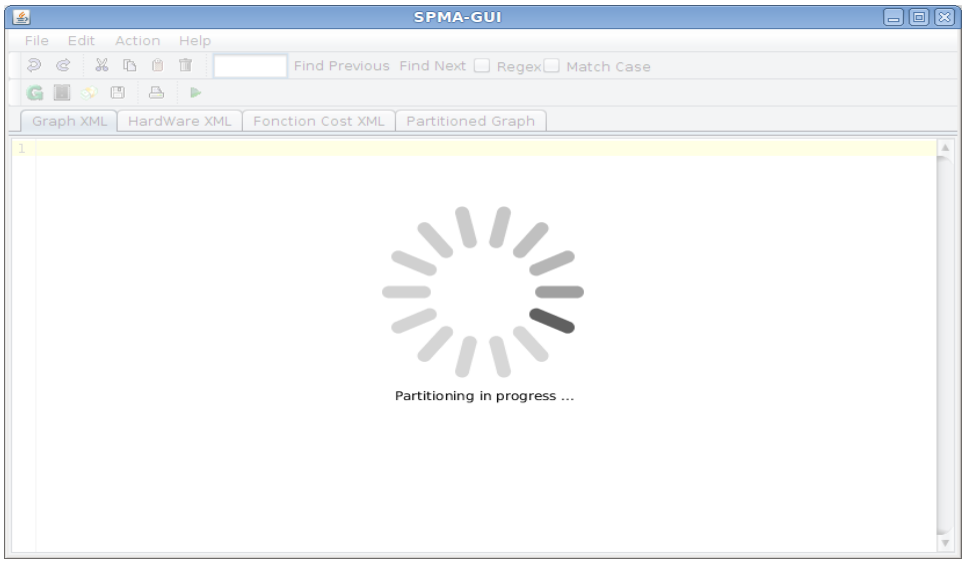

Figure 5.16 : L'écran d'attente

Apres le partitionnement, on obtient le fichier XML de partitionnement et une fenêtre de statistique et le résultat de partitionnement.

On peut visualiser les graphes d'entrée et de sortie avec les menus

Action > Visualize Presq-Partitioned Graph : pour visualiser le graphe avant partitionnement.

Action > Visualize Partitioned Graph: pour visualiser le graphe après partitionnement.

8 Expérimentation

Pour l'expérimentation de notre approche et vu le manque de BenchMark, on a choisi un exemple simple.

8.1 Exemple applicatif

Cet exemple traite de l'ascenseur d'un bâtiment. Nous supposons que ce système est décrit sous la forme suivante :

```
poss=1 ;
i=1 ;

write(presse entrée) ;
write(donner la direction) ;
read(direction) ;

if (poss <direction)
                begin
                write(Ascenseur montant) ;
                i:=poss ;
                while (i<n) do
```

```
                i:=i+1;
                end
else
        if (poss>direction)
                begin
                write(Ascenseur descendant) ;
                while (i<n) do
                i:=i+1 ;
                i:=poss ;
                end
else
        write(erreur...changer la direction) ;
```

Les modifications ajoutées à notre approche pour exécuter cet exemple

a. Le graphe d'entrée

On a ajouté à l'élément *node* l'attribut :

```
<xs:attribute name="type" type="xs:String" use="required" />
```

On a ajouté cet attribut pour spécifier le type du nœud, il y a les types suivants :

valeur de l'attribut type	Description
ADD	Le nœud représente une opération d'adition
SUB	Le nœud représente une opération de soustraction
CMP	Le nœud représente une opération de comparaison
MUL	Le nœud représente une opération de multiplication
REG	Le nœud représente une opération d'incrémentation
NOP	Autres que les opérations précédentes

Tableau 5.1 : les composants de l'architecture cible

Le graphe sera sous la forme suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<graphe                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../xsd/grapheXMLSchema2.xsd">
  <nodes>
    <node id="1" size="2" type="nop"/>
    <node id="2" size="6" type="nop"/>
    <node id="3" size="3" type="cmp"/>
    <node id="4" size="4" type="nop"/>
    <node id="5" size="3" type="cmp"/>
    <node id="6" size="3" type="cmp"/>
    <node id="7" size="3" type="add"/>
    <node id="8" size="4" type="nop"/>
    <node id="9" size="2" type="nop"/>
    <node id="10" size="3" type="cmp"/>
    <node id="11" size="4" type="sub"/>
  </nodes>
  <neighborhood>
    <communication id="1" node1ID="1" node2ID="2" com="2" frequency="1"/>
    <communication id="2" node1ID="2" node2ID="3" com="2" frequency="1"/>
    <communication id="3" node1ID="3" node2ID="4" com="1" frequency="1"/>
```

```
    <communication id="4" node1ID="3" node2ID="5" com="1" frequency="1"/>
    <communication id="5" node1ID="4" node2ID="6" com="2" frequency="1"/>
    <communication id="6" node1ID="6" node2ID="7" com="1" frequency="1"/>
    <communication id="7" node1ID="7" node2ID="6" com="1" frequency="1"/>
    <communication id="8" node1ID="5" node2ID="8" com="2" frequency="1"/>
    <communication id="9" node1ID="5" node2ID="9" com="0" frequency="1"/>
    <communication id="10" node1ID="8" node2ID="10" com="1" frequency="1"/>
    <communication     id="11"     node1ID="10"     node2ID="11"     com="2"
frequency="1"/>
    <communication     id="12"     node1ID="11"     node2ID="10"     com="1"
frequency="1"/>

  </neighborhood>
</graphe>
```

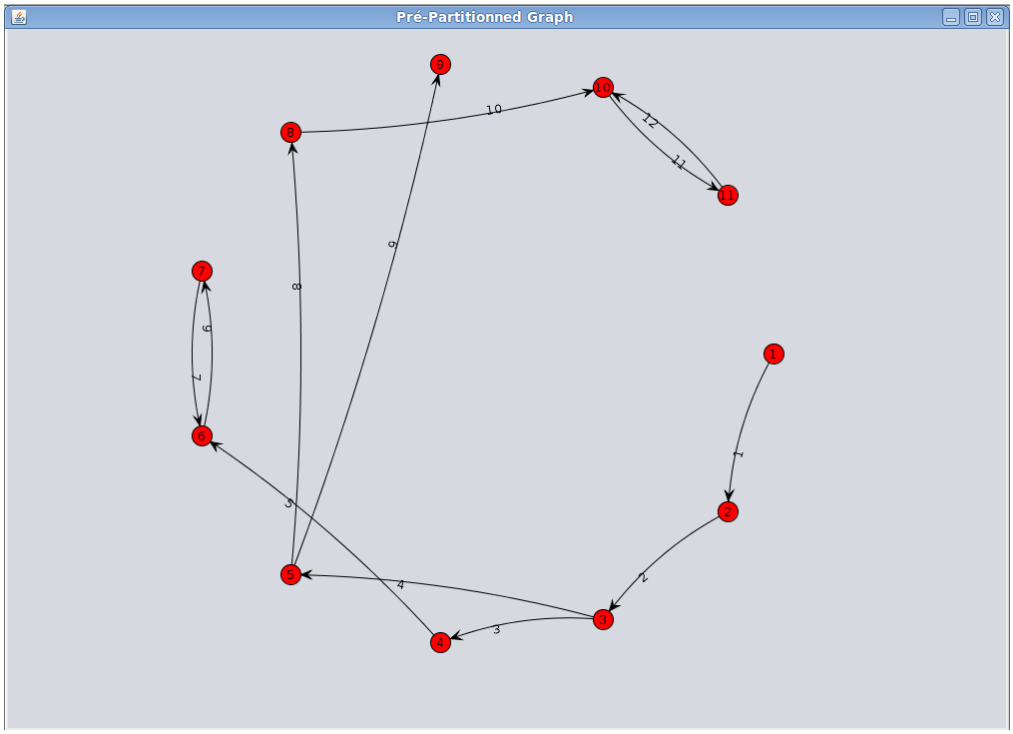

Figure 5.17 : Le graphe avant le partitionnement

b. L'architecture cible

Dans le XSD de L'architecture cible, l'attribut *type* fonction comme l'attribut *type* de le XSD du graphe d'entrée (est-ce que le composant matériel est additionneur soustracteur, etc).

```
<?xml version="1.0" encoding="UTF-8"?>
<hardeWare            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../xsd/hardwareSpecification2.xsd">
  <prcessor  id="1" clockSpeed="200" space="3" type="add"/>
  <prcessor  id="2" clockSpeed="200" space="3" type="sup"/>
  <prcessor  id="3" clockSpeed="200" space="3" type="mul"/>
  <prcessor  id="4" clockSpeed="200" space="3" type="cmp"/>
</hardeWare>
```

c. Les algorithmes

La seule modification sera dans l'algorithme des agents partitionneurs. On ajoute la condition suivante :

Si NoeudEncours.type<>AgentPartitioneur.type alors {pas de traitement} Sinon {Traitement}

C'est-à-dire on traite seulement les nœuds du même type que le matériel (un additionneur traite une édition par exemple).

d. Le résultat de partitionnement sera le suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<graphe                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../xsd/partitionedGrapheXMLSchema.xsd">
  <nodes>
    <node id="1" implementation="softWare" hardwareID="-1" />
    <node id="2" implementation="softWare" hardwareID="-1" />
    <node id="3" implementation="hardWare" hardwareID="4" />
    <node id="4" implementation="softWare" hardwareID="-1" />
    <node id="5" implementation="hardWare" hardwareID="4" />
    <node id="6" implementation="hardWare" hardwareID="4" />
    <node id="7" implementation="hardWare" hardwareID="1" />
    <node id="8" implementation="softWare" hardwareID="-1" />
    <node id="9" implementation="softWare" hardwareID="-1" />
    <node id="10" implementation="softWare" hardwareID="-1" />
    <node id="11" implementation="softWare" hardwareID="-1" />
  </nodes>

</graphe>
```

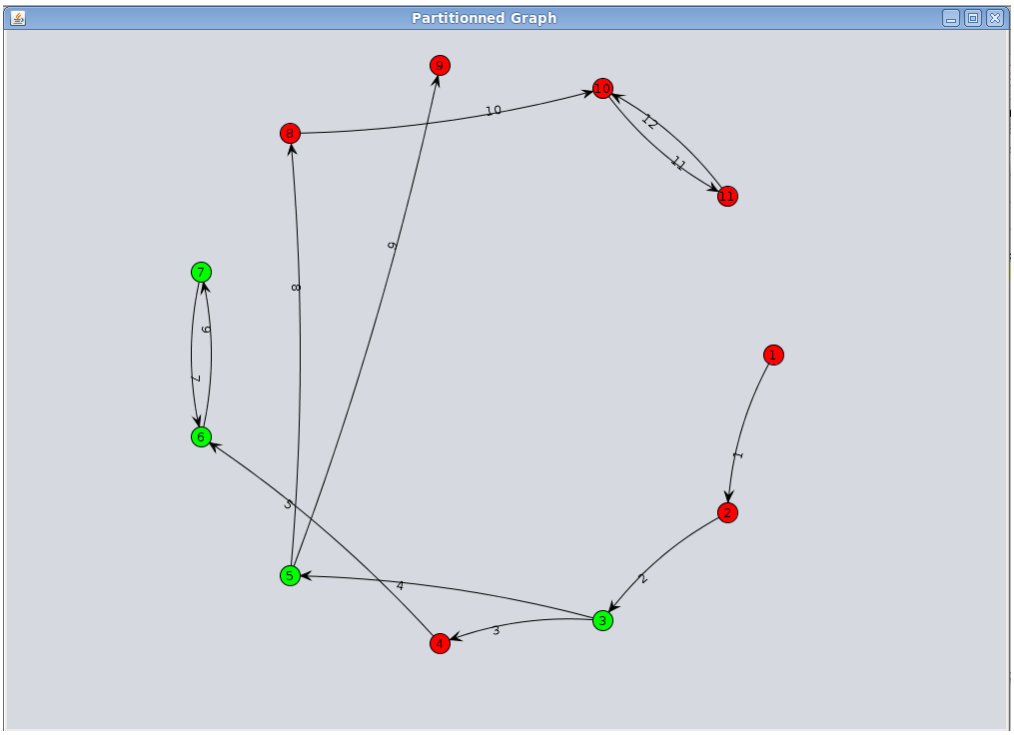

Figure 5.18 : Le graphe après le partitionnement

Conclusion et perspectives

Notre travail a mis l'emphase sur le problème de partitionnement M/L, c'est un problème qui est toujours ouvert. Nous l'avons approché par un paradigme IA : *les SMA*. Les SMA apportent une plus grande puissance de calcul, une meilleure exploration de l'espace solutions par conséquent une solution de meilleure qualité.

Outre l'originalité de l'approche de modélisation, elle a été implémentée dans un environnement plus approprié multi agents (*MadKit, Multi Agent Development Kit*). Le système résultant *SPMA* a été validé sur un exemple simple, et les résultats obtenus sont encourageants.

Dans l'avenir nous espérons étendre ce travail à :

- Une combinaison des nouvelles heuristiques (Recuit simulé,…) avec les SMA.
- Réimplémenter des solutions itératives (colonie de fourmis, Algorithme génétique, …) avec des SMA vue que la nature de ces heuristiques est de nature parallèle.
- Utiliser des heuristiques complémentaires telles que la colonie de fourmis avec heuristique de choix de voisinage (Recuit simulé).
- …

Références Bibliographiques

[Agimont, 1996] Agimont, G. (1996). *Modélisation et Simulation des Organisations Mutli-agents*, Thèse de doctorat, Université de Valenciennes et du Hainaut-Cambrésis.

[Admane, 2004] Admane L., Benatchba K., Koudil M. and Drias H., ”Evolutionary methods for solving data-mining problems”, IEEE International Conference on Systems, Man & Cybernetics, Netherlands, October 2004.

[Agerbeck, 2008] C. Agerbeck, Mikael, O. Hansen, K. Lyngby , « *A Multi-Agent Approach to Solving. NP-Complete Problems*”, IMM-Masters Thesis-2008.

[Benatchba, 2004] Benatchba K., Admane L., Koudil M. and Drias H. ”Application of ant colonies to data-mining expressed as Max-Sat problems”, International Conference on Mathematical Methods for Learning, MML’2004, Italy, June 2004.

[Boissier, 1996] Boissier, O. and Demazeau, Y. (1996). Asic: An architecture for social and individual control and its application to computer vision. In John W. Perram and Jean-Pierre Müller, editor, *Distributed Software Agents and Applications, 6th European Workshop on Modelling Autonomous Agents - MAAMAW ’94*, volume 1069, pages 1–18, Denmark. Springer.

[Bourdon, 2001] Bourdon, F. (2001). *http://www.iutc3.unicaen.fr/ bourdf/cours/DEA-I3/index.html.*

[Baeijs and Demazeau, 1996] Baeijs, C. and Demazeau, Y. (1996). Les organisations dans les systèmes multi-agents. In *5éme Journée Nationale du PRC-IA sur les Systèmes Multi-Agents*, pages 35–46.

[Benatchba, 2004] Benatchba K., Admane L., Koudil M. and Drias H. ”Application of ant colonies to data-mining expressed as Max-Sat problems”, International Conference on Mathematical Methods for Learning, MML’2004, Italy, June 2004.

[Costanzo, 2006] COSTANZO Andrea, LUONG Thé Van, MARILL Guillaume,” Optimisation par colonies de fourmis”, 19 mai 2006.

[Costa, 1997] Costa D. and Hertz V., ”Ants can Colour Graphs”, Journal of the Operational Research Society”, 48:295-305, 1997.

[Cormen, 2001] T. H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cli®ord Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.

[De Kock, 2000] E.A. de Kock, G. Essink, W.J.M. Smits, P. van der Wolf, J.-Y. Brunel, W.M. Kruijtzer, P. Lieverse, K.A. Vissers, YAPI: Application Modeling for Signal Processing Systems, *Design Automation Conference (DAC'00),* Los Angeles, june 2000.

[Demazeau and Costa, 1996] Demazeau, Y. and Costa, A. R. (1996). Populations and organisations in open multi-agent systems. In *1st Symposium on Parallel and Distributed AI*, Hyderabad, India.

[Demazeau, 1995] Demazeau, Y. (1995). From interactions to collective behavior in agent-based systems. In *European Conference on Cognitive Science*, Saint-Malo France.

[Drogoul, 1993] Drogoul, A. (1993). *De la Simulation Multi-agents à la Résolution Collective de Problèmes*, Thèse de doctorat, Université Paris VI.

[Dorigo, 1992] Dorigo M., "Optimization, Learning and Natural Algorithms", PhD thesis, Politecnico di Milano, Italy, 1992.

[Ernst, 93] R. Ernst, J. Henkel, T. Benner, «Hardware-Software Co-synthesis for microcontrôleur », IEEE Design and Test of Computer, Vol 10, N°4, Decembre 1993.

[Emin Aydin, 2004] M. Emin Aydin and Terence C. Fogarty. Teams of autonomous agents for job-shop scheduling problems: An experimental study. *Journal of Intelligent Manufacturing*, 15(4):455{462, 2004.

[Freund, 1997] L. Freund, D. Dupont, M. Israel, F. Rousseau : «Interface Optimization During Hardware-Software Partitioning», 5th International Workshop on Hardware/ Software Co-Design, Codes/CASHE '97, Braunschweig, Germany - 24-26 March 1997.

[Franklin and Graesser, 1996] Franklin, S. and Graesser, A. C. (1996). Is it an agent, or just a program?: A taxonomy for autonomous agents. In *conference of the Third International Workshop on Agent Theories, Architectures and Languages (ATAL'96)*, pages 21–35.

[Ferber, 1995] Ferber, J. (1995). *Les Systèmes Multi-Agents : vers une intelligence collective*. InterEditions.

[Fox, 1981] Fox, M. (1981). An organizational view of distributed systems. In *IEEE*

Transactions on Systems, Man, and Cybernetics, volume 11, pages 70–80.

[Gasser, 1992] Gasser, L. (1992). An overview of dai. In *Distributed Artificial Intelligence : Theory and Praxis*, Boston. Kluwer Academic Publishers.

[Gupta, 94] R. K. Gupta, C. N. Coelho Jr., and G. De Micheli, «Program implementation schemes for hardware-software systems», IEEE computer, pp. 48-55. Jan 1994.

[Hirayama, 2002] K. Hirayama and M. Yokoo. Local search for distributed sat with complex local problems. In *AAMAS '02: Proceedings of the ¯rst international joint conference on Autonomous agents and multiagent systems*, pages 1199{1206, New York, NY, USA, 2002. ACM Press.

[Hirayama, 2005] K. Hirayama and M. Yokoo. The distributed breakout algorithms. *Artif. Intell.*, 161(1-2):89{115, 2005.

[Holland, 1975] J. Holland, Adaptation in natural and artificial systems, Ann Arbor : *University of Michigan Press*, 1975.

[Hübner et al., 2002] Hübner, J. F., Sichman, J. S., and Boissier, O. (2002). A model for the structural, functional, and deontic specification of organizations in multiagent systems. In *16th Brazilian Symposium on Artificial Intelligence*, pages 118–128.

[Hammami, 2005] Moez Hammami and Khaled Gh¶edira. Cosats, x-cosats: Two multi-agent sys-tems cooperating simulated annealing, tabu search and x-over operator for thek-graph partitioning problem. In KES (4), pages 647{653, 2005.

[IEEE, 1997] Proceedings of the IEEE, Special issue on Hardware/Software co-design, Numéro de mars 1997.

[Israel, 97] M. Israel, D. Dupont, « De la spécification formelle au partitionnement matériel logiciel », Université d'Evry Bd des Coquibus, Traitement du Signal - Volume 14 - n°6 – Spécial, 1997.

[Koudil, 2003] Koudil M., Benatchba K., Dours D., ”Using genetic algorithms for solving partitioning problem in codesign”, Lecture Notes in Computer Science, Springer-Verlag, Vol. 2687, pp. 393-400, June 2003.

[Koudil, 2005] M. Koudil, K. Benatchba, S. Gharout, N. Hamani, "Solving Partitioning Problem in Codesign with Ant Colonies", IWINAC2005, LNCS Springer-Verlag, June 2005.

[Kirkpatrick, 1983] S Kirkpatrick, C D Gelatt, M P Vechi, "Optimization by simulated annealing", Science, V-220, 1983.

[Le Strugeon, 1995] Le Strugeon, E. (1995). *Une méthodologie d'auto-adaptation d'un système multiagents cognitifs*, Thèse de doctorat, Université de Valenciennes et du Hainaut-Cambrésis.

[Lind, 2001] Lind, J. (2001). *Interative Software Engineering for multiagent systems: The MASSIVE Method*, volume 2001 of *LNCS/LNAI*. Springer Verlag.

[Liu, 1994] J. Liu and Katia Sycara. Distributed problem solving through coordination in a society of agents. In *Proceedings of the 13th International Workshop on Distributed Arti¯cial Intelligence*, 1994.

[Leong, 2006] H. Leong and M. Liu. A multi-agent algorithm for vehicle routing problem with time window. In *SAC '06: Proceedings of the 2006 ACM sympo-sium on Applied computing*, pages 106{111, New York, NY, USA, 2006. ACM.

[Meignan, 2008] D. Meignan, J.-C. Créput, A. Koukam, « Un framework organisationnel pour la conception et l'implantation multi-agent de métaheuristiques », 2008.

[Malone, 1987] Malone, T. (1987). Modeling coordination in organizations and markets. In *Management science*, volume 33, pages 1317–1332.

[March and Simon, 1958] March, J. and Simon, H. (1958). Organizations. New-York.

[Mintzberg, 1982] Mintzberg, H. (1982). *Structure et Dynamique des Organisations*. Organisation.

[Occello, 1998] Occello, M. and Demazeau, Y. (1998). Modelling decision making systems using agents for cooperation in a real time constraints. In *3rd IFAC Symposium on Intelligent Autonomous Vehicles*, volume 1, pages 51–56, Madrid, Spain.

[OGJF 1998] A model for social structures in multi-agent systems, *Olivier Gutknecht et Jacques Ferber*, LIRMM Research Report No 98040, 1998

[Rao and Georgeff, 1995] Rao, A. and Georgeff, M. (1995). Bdi agents : from theory to practice. In *conference of 1st International Conference on Multi-Agent Systems ICMAS*, pages 312–319. AAAI Press.

[Rodriguez, 1994] Rodriguez, M. (1994). *Modélisation d'un agent autonome: Approche constructiviste de l'architecture de contrôle et de la représentation de connaissances*, Thèse de doctorat, Université de Neufchâtel.

[Russell and Norvig, 1995] Russell, S. and Norvig, P. (1995). *Artificial Intelligence : a Modern Approach*. Prentice-Hall.

[Solnon, 2000] Solnon C., "Solving Permutation Constraint Satisfaction Problems with Artificial Ants", Proceedings of the 14th European Conference on Artificial Intelligence, pp 118-122. IOS Press, Amsterdam, The Netherlands, 2000.

[Talbi, 2002] E.-G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5):541{564, 2002.

[Vahid, 1998] F. Vahid, A three-step approach to the functional partitioning of large behavioral processes, Proceedings of the 11th international symposium on System synthesis, p.152-157, December 02-04, 1998, Hsinchu, Taiwan, China.

[Vercouter, 2000] Vercouter, L. (2000). *Conception et mise en oeuvre de systèmes multi-agents ouverts et distribués*, Thèse de doctorat, Université de Jean Monnet et Ecole des Mines de Saint-Etienne.

[Vidal, 2006] José M. Vidal. *Fundamentals of Multiagent Systems: Using NetLogo Models*. 2006. http://multiagent.com/2009/03/fundamentals-of-multiagent-systems.html

[Wooldridge et al., 2000] Wooldridge, M., Jennings, N., and Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. In *Autonomous Agents and Multi-Agent Systems*, volume 3, pages 285–312. Kluwer Academic Publishers.

[Wooldridge, 1999] Wooldridge, M. (1999). Intelligent agents. InWeiss, G., editor, *Multiagent systems : A modern approach to Distributed Artificial Intelligence*. MIT Press.

[Wooldridge and Jennings, 1995] Wooldridge, M. and Jennings, N. (1995). Intelligent agents : Theory and practice. In *Knowledge Enginneering Review*.

[Weiss, 1999] Weiss, G. (1999). Multiagent systems and distributed artificial intelligence. In Weiss, G., editor, *Multiagent systems : A modern approach to Distributed Artificial Intelligence*. MIT Press.

[Xiaolong, 2002] Xiaolong Jin and Jiming Liu. Multiagent sat (massat): Autonomous pattern search in constrained domains. In *IDEAL '02: Proceedings of the Third Inter-national Conference on Intelligent Data Engineering and Automated Learning*,pages 318{328, London, UK, 2002. Springer-

Verlag.

[Yokoo, 1995] M. Yokoo," Asynchronous Weak-commitment Search for Solving Distributed Constraint Satisfaction Problems", 1995

[Zhang, 2002] X. Zhang, "Constraint Programming, Simulated Annealing and Hill-Climbing Algorithm for Traveling Tournament Problems". Technical report, School of Computing, National University of Singapore, 2002.

[Zivan, 2007] R. Zivan and A. Meisels, "Asynchronous Forward-checking for DisCSPs", *Constraints, 12, pp. 131-150,(Jan 2007).*

[Zhou, 2005] T. Zhou, Yihong Tan, and L. Xing. A multi-agent approach for solving travelling salesman problem. *Wuhan University Journal of Natural Sciences*, 11(5), 2005.