Académie de Montpellier
Université Montpellier II
Sciences et Techniques du Languedoc

Diplôme d'Etudes Approfondies

Informatique

Simulations Distribuées Et Multi-Agents

Lieurain Emmanuel

Date de soutenance : le 30 juin 1998

Tuteurs de stage : Jacques Ferber (LIRMM), François Bousquet (CIRAD)

Remerciements

Je tenais tout d'abord à remercier Gilles Caraux qui, à l'Agro Montpellier m'a fait découvrir les possibilités de ce DEA.

Je remercie aussi mes tuteurs, Jacques Ferber et François Bousquet, pour ma formation sur les systèmes multi-agents par leurs cours et par les discussions qui ont permis d'élaborer ce stage.

Un grand merci à toutes les personnes de l'équipe GREEN du CIRAD qui m'ont intégré dans leur équipe et qui m'ont apporté leur soutien.

Enfin, merci à François De Coligny et à Olivier Gutknecht pour leur aide en programmation et pour les réflexions « prises de têtes » que l'on a pu avoir ensemble ! !

Résumé

Les systèmes Multi-Agents ont, depuis quelques années, réalisé une percée dans le domaine de la simulation informatique. Associés à une programmation orientée objet, ils apportent une grande flexibilité au modèle conceptuel simulé et de ce fait, donnent des résultats convaincants.

Les limites d'une simulation se font ressentir lorsque celle-ci devient de plus en plus étoffée, lorsque le nombre d'acteurs augmente, ou bien lorsque les actions de ces acteurs nécessitent une grande quantité de ressources.

Un moyen possible pour palier à cet inconvénient est de distribuer la simulation afin de répartir de manière judicieuse les diverses charges de chaque élément qui la constitue.

Le but est donc de trouver la meilleure stratégie pour distribuer le problème : distribution des agents en fonction de leur charge, scission de l'environnement en morceaux séparés. Mais il faut aussi s'intéresser aux nouveaux problèmes qui peuvent subvenir : synchronisation de la simulation, reprise après défaillance.

Ce stage permet d'estimer les possibilités de simulation distribuée avec comme support la plate-forme Madkit, basée sur le modèle organisationnel Aalaadin. On peut constater que la distribution d'agents est possible mais impose une nouvelle pensée de modélisation et une amélioration de la plate-forme elle-même pour en faciliter la tâche.

REMERCIEMENTS

RESUME

I. INTRODUCTION

Réaliser des simulations est un moyen de tester des modèles provenant de domaines variés et souvent pluridisciplinaires. Les simulations donnent des résultats pouvant ainsi infirmer ou confirmer l'efficacité de ce modèle. Par la suite, on peut alors revenir sur ce modèle pour l'affiner afin qu'il respecte au mieux les contraintes du monde réel. Les simulations nous facilitent la compréhension de notre monde en tentant de prévoir des faits (comme la météorologie par exemple) ou bien en copiant une partie de notre monde et en essayant de le plagier pour mieux le comprendre ou pour tenter d'expliquer certains phénomènes.

Le concept informatique des Systèmes Multi-Agents est très adapté pour réaliser des simulations car il met en œuvre des entités différentes, indépendantes, ayant chacune des buts bien précis.[FER95] Il est alors facile de définir les acteurs de la simulation comme des agents informatiques et de réaliser un modèle conceptuel. L'apport des langages Orientés-Objets a largement favorisé la création de ce type de système.

Certaines simulations peuvent mettre en œuvre beaucoup d'acteurs « réfléchissant », peu ou alors très peu d'acteurs mais très sophistiqués. Dans ces deux types de simulation, un facteur limitant à sa réalisation se situe au niveau des ressources disponibles sur la machine. Les agents nécessitent bien sûr une partie de ces ressources pour exister, mais il faut aussi tenir compte des ressources utilisées pour les interactions entre agents. Si ces agents sont très communicant, il se peut que la majeure partie des ressources utilisées par la simulation proviennent de ces interactions. L'idée principale est donc d'utiliser un second concept : les systèmes distribués, susceptibles d'apporter les ressources nécessaires pour réaliser de telles simulations. La distribution est la mise en commun de ressources provenant de diverses machines reliées entre elles. Ce type de distribution se réalise par l'intermédiaire d'un réseau (de type Ethernet par exemple) et non au sein d'une machine parallèle. Le réseau est facile d'accès, présent dans tous les laboratoires de recherche et de plus en plus au sein des entreprises. Cela semble donc le moyen le plus facile à utiliser et qui nécessite le moins de mise en œuvre au niveau du matériel.

Le but de ce stage est donc d'allier les systèmes Multi-Agents avec les systèmes distribués, d'en faire apparaître des propriétés intéressantes et aussi de voir les problèmes nouveaux que cela créé. En effet, l'utilisation du réseau incombe des modèles de conceptions nouvelles avec une nouvelle réflexion sur les conséquences des transferts de données via le protocole TCP/IP, des envois de messages et des problèmes d'interblocage.

La plate-forme multi-agents testée est la plate-forme MadKit implementée en Java et qui repose sur le modèle conceptuel Aalaadin de Jacques Ferber [FER98]. Nous verrons les

deux types d'implémentations possibles pour réaliser une simulation sur cette plate-forme et pour cela nous utiliserons un modèle d'application choisi par le CIRAD qui est le modèle économique d'Epstein et d'Axtell (SugarScape). Ce modèle est assez simple à implémenter mais propose une complexité et des résultats de simulation intéressants.

II. Etat de l'art

1. Du point de vue de la simulation multi-agents.

De nombreuses méthodes sont utilisées pour réaliser des simulations et de nombreuses plates-formes multi-agents existent en fonction des diverses écoles formées pendant l'évolution de la branche informatique de l'intelligence artificielle (Acte de langages, IAD, Réseaux de Pétri, Langages d'acteurs, Agents Réactifs). La simulation est un des domaines d'applications privilégiés pour les systèmes multi-agents et est très porteuse grâce à l'efficacité des modèles et à la facilité de représentation des acteurs de la simulation. [FER95]

La plate-forme MANTA [FER95, MAG96], est exclusivement orientée vers la coopération entre agents sans système de contrôle centralisé. Elle s'appuie sur des exemples précis tels que la mise en place d'une société de fourmis et ses résultats portent sur l'émergence de ce type de société.

La plate-forme CORMAS est une plate-forme générique pour réaliser des simulations. Elle dispose d'un ensemble d'outils pour décrire des règles de comportements pour les agents situés ou communicants, et un autre panel d'outils pour réaliser un environnement de simulation.[CIR98].

MANTA est très spécifique et sa distribution n'est pas très importante car elle gère très bien et sans problème les divers types de simulation. Au contraire, CORMAS étant très modulaire au niveau de la programmation et au niveau de son fonctionnement, il serait intéressant de tenter d'utiliser cette plate-forme en mode distribué. Cependant, cette modularité est due à sa programmation en Smalltalk et ce langage ne gère pas les protocoles réseaux (en fait, la dernière version de VisualWorks le permet mais elle est très récente et en est au stade de découverte). Des travaux futurs au CIRAD sont envisagés pour implémenter CORMAS avec cette nouvelle version.

La plate-forme Madkit est elle aussi très modulaire, mais n'est pas optimisée spécialement pour la simulation [FER98]. Par contre, le fait qu'elle soit implementée en Java lui offre des méthodes « clé-en-main » pour utiliser les protocoles réseaux. De plus, les outils de

communication sont déjà implémentés et gèrent les envois de messages distants (en utilisant le protocole TCP/IP). C'est donc sur cette dernière plate-forme que nous effectuerons nos tests.

2. Du point de vue de la simulation distribuée.

De nombreuses recherches ont été effectuées sur la simulation distribuée. On peut distinguer deux cas :

- L'utilisation de machines parallèles pour réaliser des simulations distribuées est une possibilité. Cependant, l'utilisation d'une machine parallèle n'étant pas encore très répandue, ce n'est pas un moyen accessible pour tous et donc, n'a pas été retenu. Ceci dit, ces machines ont l'avantage d'avoir un type d'implémentation classique, à quelques instructions près, et évitent tous les problèmes de réseaux et de messageries distantes. [ICPP]
- Dans les autres cas, où on utilise un réseau classique, les simulations distribuées ont été implémentées à partir d'Automates Cellulaires [GRO96] ou bien de réseaux de Pétri [CIR98,SCH96], mais les méthodes employées ne sont pas adaptées au besoin que nous avions, c'est à dire adaptées au modèle Aalaadin en particulier.

Même si l'on ne trouve pas beaucoup de travaux appliqués aux systèmes multi-agents, on peut toutefois noter l'existence de méthodes utiles pour distribuer une simulation :

- Les méthodes utilisant les actes de langages [DEA97, AGH91] ou bien la programmation par continuation [HEW77, AGH86] peuvent s'avérer utiles dans le cas où les acteurs seraient modélisés comme des agents communicants :en effet, ces méthodes traitent les messages de manière à optimiser les interactions entre agents et sont susceptibles de lever des interblocages. Malheureusement, ces méthodes ne sont pas accessibles rapidement et ne peuvent pas s'intégrer dans le cadre de ce stage. Cependant il est à souligner que ces méthodes ne doivent pas être placées à l'écart et que de plus profondes recherches devraient être mises en œuvre .
- La méthode la plus approfondie a été le « Scheduling » [JER96, BUS96]. Elle met en place un système synchrone d'agents qui évoluent dans une interface asynchrone. Les agents sont orchestrés par un

scheduler qui est là uniquement pour indiquer les débuts (et/ou les fins) de chaque pas de temps. Ce type de méthode s'intègre très facilement dans un système multi-agents dit « Réactif » où l'évolution et le comportement de chaque agent doivent être ordonnées. Il est le chef d'orchestre de la simulation et peut agir de deux manières sur les acteurs. Soit le scheduler donne le départ d'un pas de temps aux acteurs les uns après les autres et dans ce cas ils effectuent leurs tâches successivement, soit le départ est donné à tous les acteurs en même temps et chaque acteur s'exprime quand il le peut s'ils sont en concurrence au niveau des processus.

Cette dernière méthode est donc la plus pertinente car, en général, les simulations que l'on réalise sont toutes constituées d'agents réactifs et très peu d'agents cognitifs sont implémentés. En fait, même si la complexité de l'agent augmente fortement, on le considère toujours comme un agent réactif.

Les agents Madkit sont des Threads Java, ils sont donc indépendants les uns des autres et évoluent de manière asynchrone. Le scheduler est donc nécessaire pour ordonner une simulation dans ce type de plate-forme.

Nous allons donc voir les possibilités de distribution d'un modèle simulé d'agents réactifs, implémenté en utilisant un scheduler. Ce modèle sera décrit au niveau de son fonctionnement, puis nous verrons deux types d'implémentations avec des fonctionnements différents (et donc des buts différents).

III. Matériels et méthodes

1. Présentation du modèle d'études : The SugarScape

1.1 Pourquoi celle-ci?

Afin de distribuer une simulation, il fallait au départ en trouver une ! Il s'est avéré que les modèles économiques étaient au cœur des investigations du CIRAD [ANT98]. D'un commun accord entre François Bousquet et Jacques Ferber, nous avons donc poursuivi sur cette voie. L'intérêt est alors devenu double : la simulation en elle-même pour la distribuer plus tard et les résultats de la simulation.

1.2 Présentation

Epstein et Axtell ont travaillé sur la simulation de sociétés artificielles dans leur livre « Growing Artificial Societies » [EPS96]. Une des propriétés de leur modèle qui le rend assimilable à un économique est l'échange de ressources entre les agents avec formation d'un prix au sens de Pareto, c'est à dire que le prix ne dépend pas d'une personne tel qu'un commissaire priseur, il n'est pas calculé par rapport aux offres et aux demandes au sein d'un marché. Le prix est déterminer entre deux acteurs, en fonction de leurs besoins propre.

Le modèle se compose d'agents possédant divers attributs et évoluant dans un environnement qui contient des ressources renouvelables. Le comportement des agents se définit par diverses règles que nous détaillerons par la suite. On peut noter que le modèle contient beaucoup de règles très variées, mais nous nous contenterons les règles de déplacement, d'échanges et de renouvellement Agents/Ressources.

1.2.1 L'Environnement

Celui-ci est une grille de 50 cases sur 50, dont certaines possèdent des ressources. Il existe deux types de ressources : le sucre et l'épice. Ces deux ressources peuvent se régénérer lorsqu'elles sont consommées par les agents. Chaque ressource est disposée sur deux zones, à quantité égale et répartie selon une fonction de gradient (paraboloïde) centrée sur chaque zone. Ainsi, deux pics pour chaque ressource existent dans l'environnement.

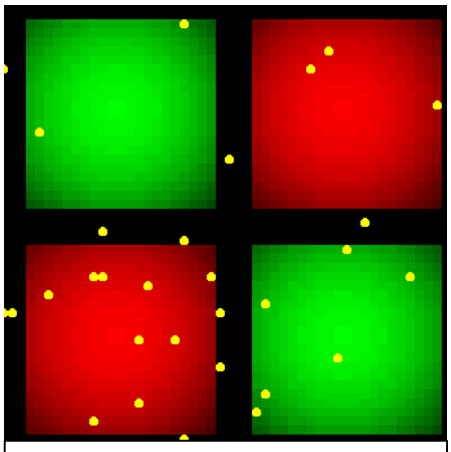


Figure 1: L'environnement et quelques agents

La règle de régénération des ressources se définit au début de la simulation. En général on autorise une augmentation de un pour toutes les cases à ressources n'ayant pas atteint leur maximum.

1.2.2 Les Agents

Les agents représentent des entités capables de se déplacer, de voir autour d'eux, de récupérer des ressources provenant de la case où il se situe et d'en échanger une partie à ses congénères.

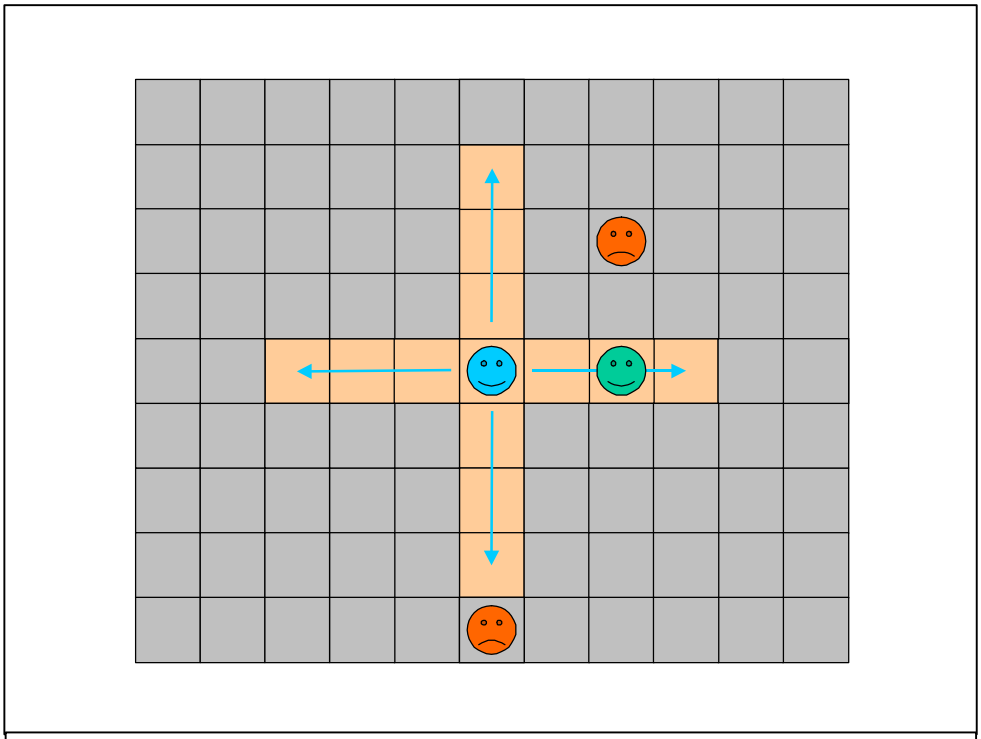


Figure 2: Zone de visibilité d'un Agent

Voici la liste de leurs attributs :

Age	Age maximum de l'agent : compris en 900 et 1000 pas de temps.
StockR1	Quantité de sucre en réserve. Initialisé à 200.
StockR2	Quantité d'épice en réserve. Initialisé à 200.
Metab1	Consommation de sucre par tour : comprise entre 1 et 6.
Metab2	Consommation d'épice par tour : comprise entre 1 et 6.
Visibilité	Nombre de cases visibles par l'agent dans un espace 4-Voisinage.

On définit plusieurs règles de comportement qui utilisent ces attributs.

- **Règle M** : règle de mouvement de l'agent. L'agent peut se déplacer en un pas de temps d'une case. Il recherche parmi les cases perçues grâce à sa visibilité la case qui contient le plus de ressources dont il est dépendant, i.e. l'agent se dirige vers le type de ressource dont il a le plus besoin.
- **Règle R** : cette règle permet le renouvellement des attributs quand l'agent encore vivant atteint son âge maximal. Tous les attributs aléatoires sont réinitialisés.
- **Règle S** : un agent peut se reproduire si ses stocks en ressources sont suffisants. Si les stocks ont plus de 350 unités alors un nouvel agent est créé et les stocks du parent diminuent de 200.
- **Règle T** : un agent est capable de détecter les agents qui évoluent dans sa zone de visibilité. Il peut alors réaliser des échanges de ressources avec eux. On définit *le taux marginal de substitution* , MRS qui représente la propension à préférer une ressource plutôt qu'une autre. On définit le MRS ainsi :

$$MRS = \frac{StockR2 \cdot Metab1}{StockR1 \cdot Metab2}$$

L'établissement d'un prix entre deux agents se fait en calculant la moyenne géométrique des deux MRS. Si le prix p est supérieur à un, l'échange est alors de p ressources1 contre une ressource2, sinon on échange une ressource1 contre $1/p$ ressources2, jusqu'à l'équilibre des deux MRS. La direction de l'échange est déterminée en comparant les MRS initiales.

1.3 Utilisation de ce modèle

Afin de valider une partie de ce modèle, nous nous sommes penchés principalement sur le processus d'échanges économiques et résultats des prix qui en découlaient. Nous avons donc utilisé l'ensemble des règles {M,T,R}. Nous voulons donc mettre en évidence l'impact des échanges durant la simulation en visualisant la moyenne des prix établis pendant un pas de temps. On doit s'attendre à une homogénéisation des quantités de ressources stockées par les agents grâce aux échanges effectués entre eux. En effet, le prix établi lors d'un échange doit tendre vers un. C'est que nous allons vérifier en l'implémentant.

2. Introduction au méta-modèle organisationnel Aalaadin et à la plate-forme MadKit

2.1La notion d'agents, de groupe et de rôle dans Aalaadin

Chaque agent est une entité autonome et communicante qui joue un ou plusieurs rôles dans un ou plusieurs groupes. Chaque groupe permet un rapprochement de certains agents ayant des affinités de communication ou de fonctionnalité. Les communications entre agents sont autorisées uniquement au sein d'un même groupe. Les rôles des agents représentent leurs capacités à effectuer un service pour un groupe donné. Un agent peut avoir plusieurs rôles et appartenir à plusieurs groupes en même temps, ce qui offre des possibilités de communication entre groupes ou alors un agent mixte peut servir d'intermédiaire entre deux groupes distincts.[GUT98]

2.2 La plate-forme MadKit

Madkit est la plate-forme qui met en œuvre les concepts développés dans Aalaadin. Celle-ci a été implémentée en Java 1.1. Elle se caractérise par la mise en place d'un micro-noyau qui sert d'environnement d'exécution pour les agents et gère les groupes, les rôles, les cycles de vie de agents et le transfert local des messages entre deux agents.

L'ensemble graphique, créé à partir de Beans Java, la G-Box est le lieu d'expression des agents. Les outils de gestion supplémentaires sont des agents eux-mêmes. Les outils les plus intéressants pour nous sont les agents qui autorisent la communication entre deux plates-formes distantes : le GroupSynchronizer et le SocketCommunicator. Le premier sert à fusionner les groupes entre les deux plates-formes, cette fusion est totalement transparente pour les agents lancés. Le second réalise les transports de données au travers d'un réseau TCP/IP. Il gère aussi les envois de messages entre deux agents distants, il prend donc le relais du micro-noyau. Ces deux agents doivent être lancés avant toute simulation et sur les deux plates-formes.

Chaque Agent est caractérisé par une carte d'identité unique, son AgentAddress, qui l'identifie sur n'importe quelle plate-forme, même si elles sont toutes synchronisées.

3. Une première implémentation du modèle applicatif

3.1 La ReactiveLibs de MadKit

Les simulations d'agents réactifs schedulés sont réalisées sous MadKit en utilisant une bibliothèque d'outils appelée ReactiveLibs. Celle-ci contient sous la forme d'un seul agent MadKit un scheduler, l'environnement dans lequel évolue les agents et les agents réactifs. L'utilisation de cette bibliothèque n'utilise aucune fonctionnalité du modèle Aalaadin étant donné que tout est encapsulé dans un seul agent.

Cet agent est donc un Thread Java autonome. Chaque agent réactif possède deux méthodes appelées par le scheduler :

- « runMe() » qui correspond au comportement de l'agent. C'est dans cette méthode que l'on pourra définir les règles {M,T,R} des Agents.
- « drawMe() » est la méthode dessinant l'agent dans l'environnement

L'environnement est une grille de cellules représentée par un tableau à deux dimensions. Le comportement de l'environnement (comme la régénération des ressources sur les cellules) sont décrits eux aussi par une méthode « runMe() » et chaque cellule est dessinée par une méthode « drawMe() ».

Le scheduler est directement intégré à un Canvas associé au Thread et les pas de temps sont définis par le rafraîchissement du Canvas (repaint() ;). Entre deux rafraîchissements du Canvas les méthodes runMe() et drawMe() de chaque agent réactif sont invoquées, ainsi que ces mêmes méthodes de l'environnement.

Les agents réactifs sont contenus dans un tableau par l'agent MadKit unique (que l'on appellera le manager). Toutes les méthodes utilisées sont donc invoquées par des appels directs.

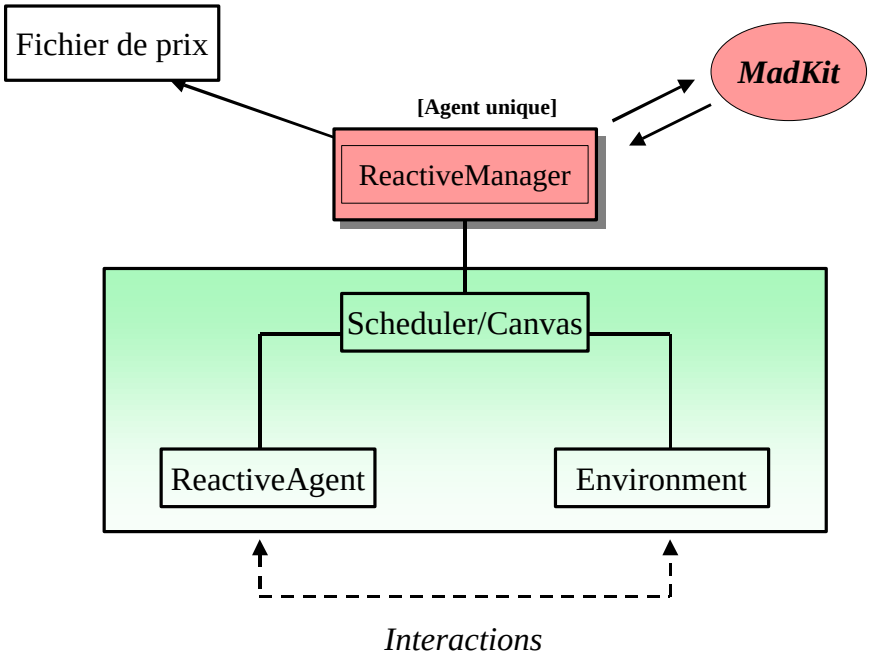


Figure 3 : Première version, un seul Agent Madkit

3.2 La mise en place du modèle SugarScape

Les règles de comportements des agents économiques et des cellules de l'environnement sont donc implémentées dans la méthode « runMe() ».

On récupère la moyenne et l'écart-type des prix établis lors des échanges pendant un tour qui sont stockés dans un fichier.

3.3 Premiers résultats

En étudiant les courbes de prix réalisées à partir des fichiers, on s'aperçoit que la moyenne des prix tend bien vers un. Ce résultat traduit une homogénéisation des stocks en ressources des agents économiques grâce aux échanges économiques.

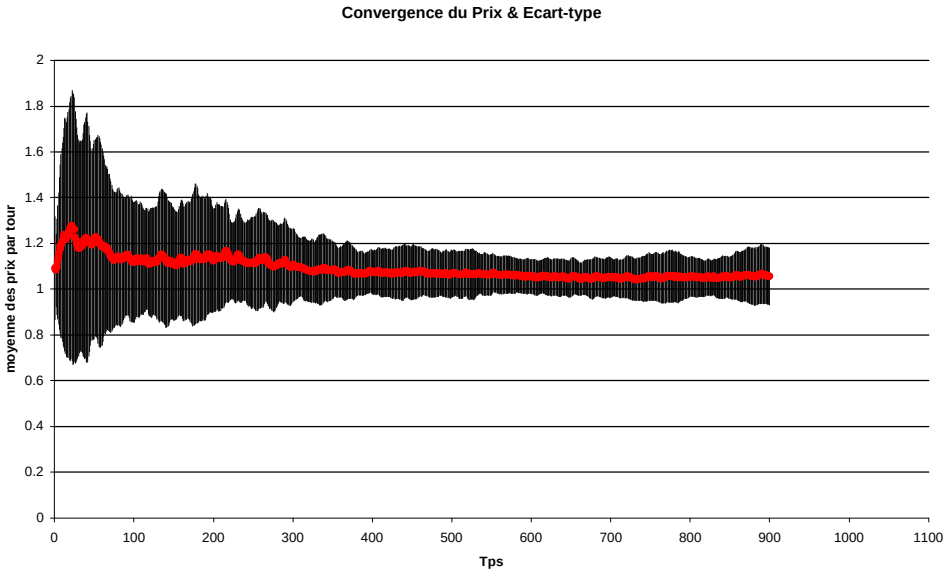


Figure 4 : Convergence du prix

Nous pouvons donc dire que le modèle applicatif est validé car on retrouve les résultats d'Epstein et d'Axtell.

3.4 Distribution et limites de la ReactiveLibs

On s'aperçoit aux vues de la construction de cet agent « tout-en-un » qu'il est difficile d'en extraire une éventuelle distribution. Il faut donc repenser à un nouveau système de scheduling qui exploitera les concepts d'Aalaadin et les fonctionnalités de MadKit

On peut assimiler cette implémentation au type Agent-Objet, que l'on retrouve aussi dans CORMAS, qui offre une certaine compacité dans la simulation, mais offre des possibilités plus restreintes d'extensions. Dans la partie suivante, nous allons voir une implémentation du type Agent-Processus qui offre plus de marges de liberté de fonctionnement, des possibilités de migration et ouvre les portes du multi-tâches.

4.Une seconde implémentation plus adaptée à la distribution

4.1 L'idée principale

L'idée est donc de réaliser la simulation en utilisant le modèle Aalaadin. Pour cela, il faut intégrer chaque agent dans un groupe de simulation et leur affecter un rôle. Par conséquent, il n'est pas question d'utiliser l'agent précédent qui encapsulait les différentes parties

de la simulation. Chaque agent doit être autonome et les interactions entre eux se feront par des envoies de messages.

4.2 Définitions des différents acteurs

Les différentes parties de la simulation précédente sont transformées en agent et sont ainsi tout indépendantes les unes des autres.

Le scheduler est un agent assez simpliste qui consiste à envoyer des messages indiquant le début du pas de temps et qui attend le message retour indiquant la fin de celui-ci. Il appartient au groupe *Simulation* et au groupe *SimSystem*. Ce dernier groupe permet de distinguer les agents organisateurs qui mettent en place la simulation des agents acteurs de la simulation.

4.2.2 L'affichage

L'affichage est aussi un agent système qui récupère par messages les informations afin de dessiner le résultat. Cet agent pourrait se généraliser pour obtenir un agent plus général, récupérateur d'informations et qui redirigerait ces informations vers d'autres agents plus spécialisés comme un afficheur, une sortie fichier ou une redirection vers une base de données.

4.2.3 Les acteurs économiques

Chaque acteur économique est un agent propre, il évolue dans un environnement qui lui aussi est un agent particulier. Toutes les interactions nécessaires avec l'environnement ou d'autres agents se font alors par envoi et réception de messages.

4.2.4 L'environnement

L'agent environnement s'occupe de gérer les cellules. Il met à jour la quantité de ressources disponibles pour les agents économiques, donne aux agents les informations correspondant à leurs requêtes pour subvenir à leurs besoins et pour obtenir la liste des agents qu'il voit afin d'échanger avec eux.

4.3 Schéma organisationnel du modèle

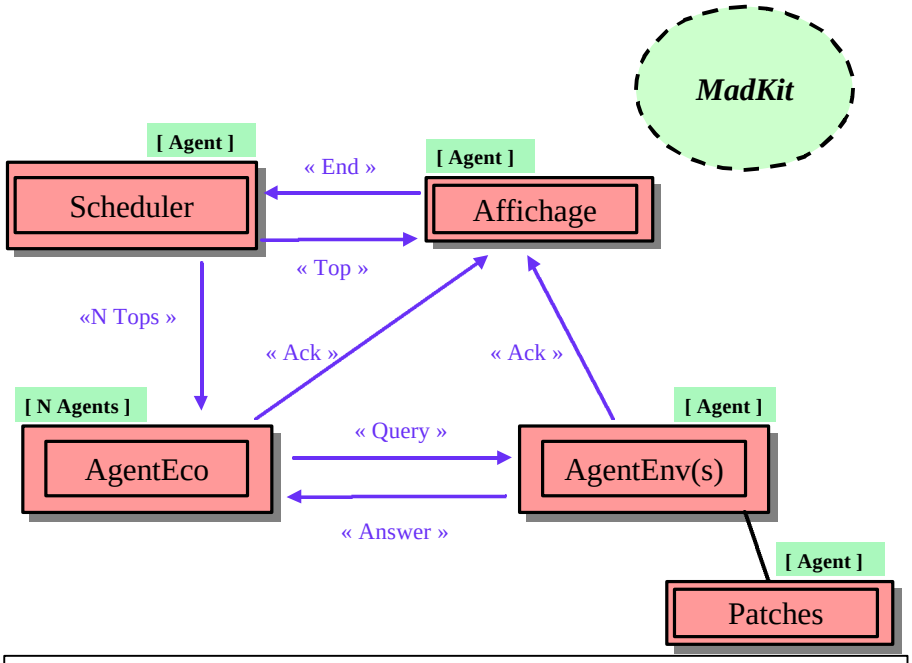


Figure 5 : Deuxième version, plusieurs Agents Madkit

4.4. Le fonctionnement de la simulation en mode distribué

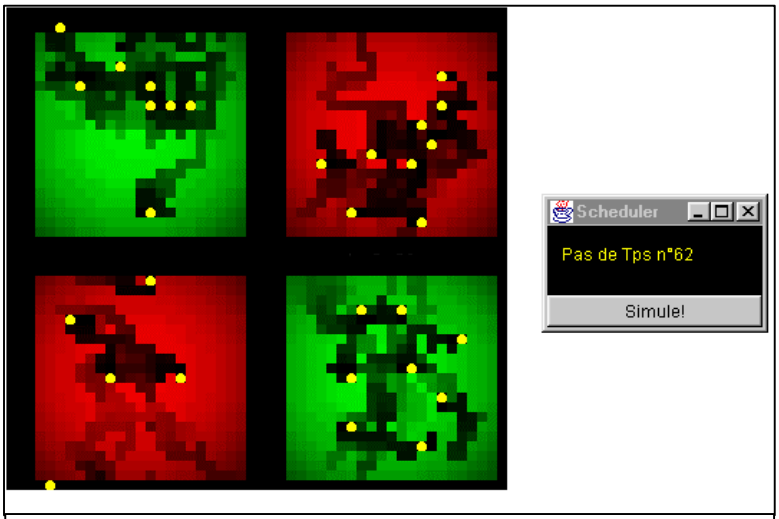


Figure 5 : La simulation en cours

La plate-forme MadKit est uniquement un support pour la simulation et il est possible d'en fusionner plusieurs au moyen des deux agents systèmes cités précédemment (cf partie MadKit… ? ? ?). Les groupes identiques fusionnent et cette notion est totalement transparente au niveau d'un agent, la simulation peut donc aussi bien fonctionner sur une ou plusieurs machines. Il ne reste alors que des problèmes de positionnement initiaux des agents réalisés par des agents « Launchers » qui installent les agents sur chaque plate-forme. A partir du moment où tous les agents se sont installés, il suffit de lancer le scheduler pour qu'il envoie son premier « Top ».

Le scheduler envoie un « Top » aux agents économiques et à l'affichage. Dès que les agents ont reçu ce « Top », il font une requête « Query » vers l'environnement qui leur répond par un message « Answer » contenant la liste des agents avec qu'il puisse échanger des ressources et la case sur laquelle ils satisferont leur besoin en ressources. Dès que l'environnement a répondu à tous les agents dont il avait fait la liste auparavant, il envoie son « Ack » à l'affichage et le nouveau tableau de cellules à dessiner. Les agents entament alors leurs discussions d'échanges avec leurs voisins. Une fois ce travail terminé, ils renvoient à leur tour leur « Ack » à l'affichage. Quand ce dernier a reçu les « Ack » des agents économiques et de l'environnement, il redessine et envoie le message « End » au scheduler. Le pas de temps est alors terminé.

IV. Résultats d'efficacité & Discussion

1 Les différences entre les deux implémentations

Même si les résultats sont identiques, la seconde implémentation nécessite beaucoup plus de ressources, ce qui n'étonnera personne. Ceci se justifie par les structures supplémentaires autour de chaque agent indispensable pour fonctionner sur la plate-forme, et aussi par l'utilisation de messages entre agents alors que dans la première version, les appels de méthodes sont directs (Agent-objets).

2 Les nouveaux problèmes rencontrés

2.1 Les possibilités d'interblocages

Dès qu'il y a des échanges de messages, on peut s'attendre à des interblocages. En effet, cela peut se produire lorsque les échanges de messages s'effectuent entre deux agents identiques. Ces phénomènes sont apparus ici lors des échanges économiques quand deux agents attendent mutuellement la réponse de l'autre ou bien lorsqu'un cycle d'échange apparaît. Chaque agent fait une requête à l'autre et ils s'attendent mutuellement la réponse.

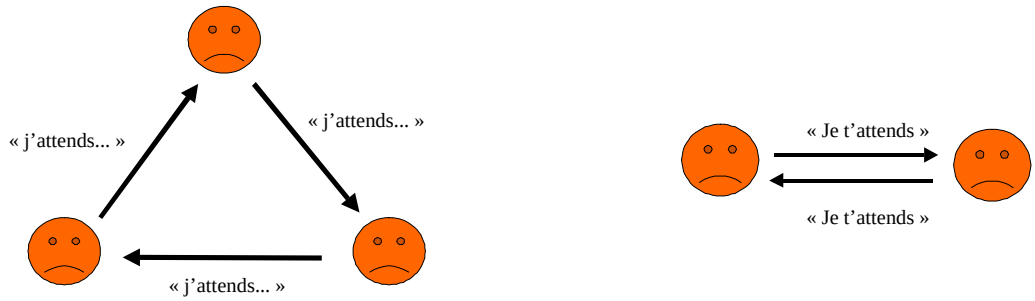


Figure 6: Les Deux types d'interblocage Rencontré

On peut résoudre ces problèmes par des méthodes classiques provenant des systèmes transactionnels, mais il était préférable d'essayer de s'en passer, pour des raisons de simplicité.

Dans ce cas précis les interblocages ont été résolus en créant une dissymétrie dans la réponse des agents. Leur décision d'attendre la réponse ou bien d'abandonner et de réessayer plus tard est anticipée. Celle-ci dépend de la carte d'identité de chaque agent (L'AgentAddress), fournie par le micro-noyau et toutes différentes les unes des autres. Une simple comparaison des AgentAddress suffit à privilégier l'un ou l'autre des agents. Les agents étant ainsi tous triés et les AgentAddress étant toutes uniques, les blocages disparaissent (Cf. Algorithme d'échange en Annexe).

2.2 La synchronisation des messages

Un second problème rencontré est la synchronisation des messages entre agents. Les messages étant stockés dans une boîte aux lettres, il a fallu forcer le tri de cette boîte afin d'en tirer le message attendu. Ceci a été réalisé en surchargeant les méthodes d'attentes actives et passives de messages (disponible sous MadKit). Ces méthodes sont primordiales pour les launchers d'une part, car chaque agent doit être lancé en étant sûr que sa phase d'initialisation sera faisable : en effet lorsque le scheduler est lancé, il doit récupérer les adresses des autres agents présents pour la simulation ; si cette recherche s'effectue alors que tous les agents ne sont pas lancés, il risque d'y avoir des pertes de messages et donc des erreurs. D'autre part, pendant la simulation elle-même, il est utile d'attendre un message précis prévenant tout risque de mauvaise interprétation de messages ou d'arrivée désordonnée de messages. [COL98]

La méthode d'attente active est « waitNextMessage() », cette méthode bloque le thread de l'agent tant que sa boite aux lettres est vide. Dès qu'un message s'y trouve, il est renvoyé et peut être traité par la suite. Si ce message ne nous intéresse pas pour le moment il est alors stocké dans une seconde boîte aux lettres annexe. La méthode d'attente active de message et les méthodes dérivées doivent alors gérer cette seconde boite aux lettres. Les méthodes faisant appel à la boîte aux lettres ont été réécrites pour prendre en compte les messages laissés de coté. L'avantage est donc que l'on peut retirer de la boite aux lettres un message contenant un libellé précis, tel que « Top », « Ack », etc.…Cela permet donc une meilleure gestion des entrées de messages : la synchronisation est alors possible.

Cette arrivée désordonnée de messages est d'autant plus vraie que la simulation est distribuée sur différentes machines. Les capacités variées des machines et leur charge de processus, la charge de données circulantes dans le réseau font que les messages n'arrivent pas dans l'ordre chronologique. C'est un problème important qu'il ne faut pas négliger.

2.3 Le nombre de messages échangés

Un calcul rapide permet de déterminer le nombre de messages échangés lors d'un pas de temps pour N agents économiques. Sans compter les messages ayant lieu à cause des échanges économiques (allant de 0 à N(N-1)), 4N+2 messages sont échangés durant chaque tour. On voit donc bien que les messages doivent être correctement gérés au niveau de la mémoire et que leur contenu doit être allégé au maximum afin de limiter des échanges de données trop coûteuses.

3 Comparaison Mono-site / Multi-site

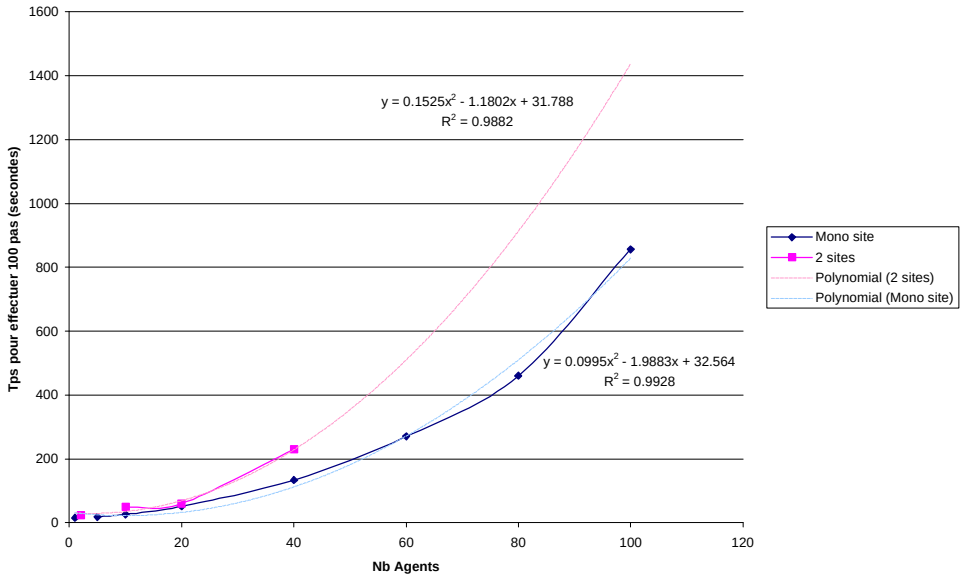
Nous avons comparé la rapidité d'exécution de la simulation, entre une version fonctionnant sur une seule machine et une version fonctionnant sur deux machines où la charge des agents serait à peu près homogène de part et d'autre. Les résultats mono-site présentés sont en fait la moyenne des résultats obtenus sur les deux machines qui serviront pour le test multi-site. Il faut aussi signaler que la charge du réseau est nulle et que les informations circulantes seront uniquement celle de la simulation (les deux machines sont ainsi isolées).

Voici les fiches signalétiques des deux machines utilisées

Pentium 233MMX, 48Mo RAM	Carte Ethernet PCI, Linux, Madkit v1.0.4
AMD K6 233 , 64Mo RAM	Carte Ethernet PCI, Linux, Madkit v1.0.4

3.1 Voici les résultats

Le test réalisé a été de calculer le temps que la simulation mettait pour réaliser 100 pas de simulation, en fonction du nombre d'agents présents. La version mono-site est en fait la moyenne des expériences réalisées sur les deux machines utilisées pour la version multi-sites, pour que l'on puisse comparer les deux résultats. Cependant les écarts étaient très minimes, les puissances des deux machines étant quasi égales. Voici les courbes obtenues :



3.2 Interprétation

On peut constater la bonne correspondance entre les deux nuages de points et leur courbe de régression polynomiale de degré 2 à partir de leur coefficient de corrélation. Le degré 2 provient du fait que les messages entraînent une complexité en $O(n^2)$. On peut donc voir que la version utilisant deux sites est plus lente que la version mono-site. Cela s'explique par le fait que les messages transitant par le réseau nécessitent un temps de l'ordre de la milliseconde alors qu'un message interne est de l'ordre de la nanoseconde. On remarque bien que c'est le coefficient du second degré de l'équation de la courbe multi-site qui augmente par rapport à la version mono-site. Au contraire le coefficient constant est le même dans les deux cas ce qui correspond à l'infrastructure de la simulation (la plate-forme, et les agents tels que l'affichage, le scheduler et l'environnement).

Néanmoins, dans les conditions actuelles de performances des deux machines, il est difficile de dépasser la centaine d'agents sur un seul site. Donc le fait d'en utiliser deux nous permet d'en atteindre le double.

Un autre test aurait été important mais n'a pas encore été réalisé pour des raisons de machines : utiliser trois machines ou quatre pour refaire la même expérience aurait été intéressant, car on aurait pu voir si la part du réseau impliqué dans la lenteur de la simulation restait constante. Dans ce cas le fait d'utiliser plus de deux machines devrait donner de meilleurs résultats que la version à une machine. Il serait intéressant de voir alors si toutes ces courbes s'insèrent entre la version mono- et bi-sites, ou bien si la part du réseau est tellement compensée par le nombre de machines que la simulation devient alors plus performante.

4. Les améliorations possibles

4.1 Distribuer l'environnement

Il paraît évident que l'environnement peut devenir le goulet d'étranglement de la simulation étant donné que tous les agents, acteurs de la simulation, lui envoient un message.

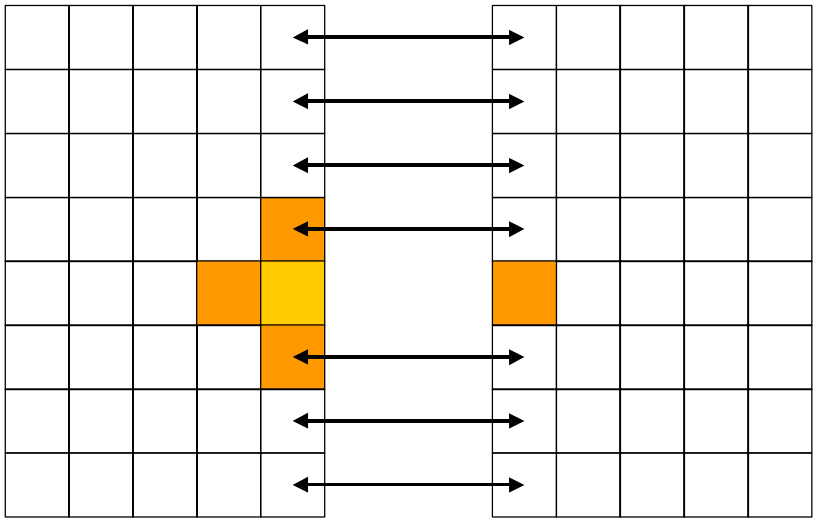


Figure 7 : 4-Voisinage des cases frontière entre deux zones

On peut alors imaginer de distribuer l'environnement, c'est à dire de le scinder en plusieurs agents qui s'occuperaient chacun d'une zone précise. Ces agents appartiendraient alors à un groupe Environnement et ils seraient capables de se demander mutuellement des requêtes afin de répondre aux demandes des agents. Les cellules de l'environnement connaissant leurs cases 4-voisines, il serait facile d'indiquer aux cellules frontières entre deux zones que leurs cases voisines se situent « chez » un autre agent.

4.2 Minimiser les messages circulant sur le réseau

Le passage de données par le réseau est un frein très important (de l'ordre des millisecondes au lieu des nanosecondes à l'intérieur d'une machine). Il est donc très important de limiter le plus possible les échanges de données entre plates-formes.

Dans un premier temps, il paraît avantageux de disposer un scheduler par plate-forme. Ces schedulers seconds seraient commandés par un scheduler principal, qui leur enverrait le « Top » pour le redistribuer aux agents locaux. On passe alors de N messages envoyés (s'il y a N agents distants) à un seul message envoyé de scheduler à scheduler. Le retour par les « Ack(knowledge) » transiterait de la même manière pour de nouveau économiser N messages.

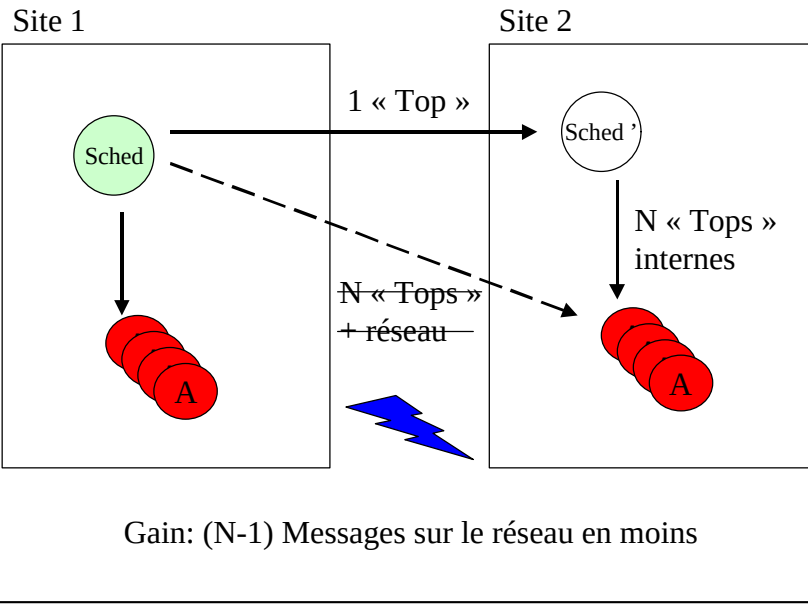


Figure 7 : Apport d'un scheduler par site

Il est possible d'envisager la migration d'agents. Cette migration pourrait s'envisager en fonction de la charge de la machine sur laquelle il se trouvait initialement, ou bien pour rapprocher des agents communicant souvent entre eux. En particulier, si l'environnement était lui-aussi distribué, il serait préférable d'avoir les agents évoluant dans une même zone sur la même plate-forme que l'agent environnement de cette zone, encore une fois pour minimiser les messages à travers le réseau.

La migration et la détection de la charge de la machine sont des outils à mettre en œuvre directement dans MadKit (actuellement en voie de développement).

4.3 Apporter une nouvelle couche préventive au protocole de communication

Ce protocole supplémentaire servirait à la détection de panne d'un site. Il serait préférable qu'une reprise après défaillance soit mise en place, pour éviter de perdre toutes les données d'une simulation, surtout si le site défaillant ne contient que des agents acteurs ou une

zone de l'environnement. Si les agents système ne sont pas touchés, il n'y a pas de raison de suspendre la simulation.

IV. Conclusions

Ce stage a montré que des simulations utilisant des agents réactifs sont facilement distribuables en s'appuyant sur l'utilisation de la plate-forme MadKit. Des outils supplémentaires et des études approfondies sur les améliorations sont cependant indispensables. En particulier, la mise en place d'une gestion « réactives » des agents par scheduling pourra bientôt être gérer par le micro-noyau directement et allégera la simulation (en cours de développement).

L'utilisation d'un réseau Ethernet est facile d'utilisation, et peut être envisagé sur d'autres plates-formes telles que CORMAS. Deux alternatives sont envisageables : utiliser la dernière version de VisualWork qui gère l'accès au réseau ou bien la réécrire en Java, langage au goût du jour, mais qui supprime la compilation « à la volée » de SmallTalk, très utile lors des formations aux multi-agents. En effet, l'utilisation de la plate-forme nécessite des connaissances limitées en informatique, car la programmation des règles des acteurs se fait directement dans celle-ci. Il faut donc peser le pour et le contre avant d'effectuer des changements.

En ce qui concerne l'expérience réalisé pendant le stage, il faut aussi noter l'importance du modèle applicatif utilisé pour réaliser l'expérience ; d'autres expériences sont souhaitées en utilisant un modèle totalement différent pour déterminer l'impact du modèle sur les résultats obtenus.

Enfin, le point le plus à approfondir est la distribution de l'environnement. Il est le point clé de toute simulation. Sa scission en plusieurs zones où chaque zone serait un agent augmenterait les performances de la simulation.

Références Bibliographiques

1. [AGH86], **Agha,** *Actor : a Model of Concurrent Computation in Distributed Systems,* MIT Press, 1986.
2. [ANT98], **Antona, Bousquet, Le Page, Weber, Karsenty, Guizol**, *Economic theory and renewable resource management,* Lecture Note in Artificial Intelligence, 1998.
3. [BUS96], **S. Bussman**, *A Multi-Agent Approach ti Dynamic Adaptative Scheduling of Material Flow,* Lecture Note in Computer Science N°1069, 1996.
4. [CIR98], Formation CIRAD Multi-Agent, support de cours, 1998.
5. [COL98], **DeColigny,** *Mémoire de DEA*, juin 1998.
6. [DEA97], **Dzerkaw,** *Analyse d'outils de simulation – Application à un problème de coordination,* Mémoire de DEA, juin 1997.
7. [EPS96], **Epstein, Axtell,** *Growing Artificial Societies,* MIT Press, 1996.
8. [FER95], **Ferber,** *Les Systèmes Multi-Agents, vers une intelligence collective,* InterEditions, 1995.
9. [FER95,GUT98], **Ferber, Gutknecht,** *Un méta-modèle organisationnel pour l'analyse, la conception et l'execution de systèmes multi-agents,* publication, ICMAS'98.
10. [GRO96], **Gronewold, SonnenSchein,** *Event-based modelling of ecological systems with asynchronous cellular automata,* 1996.
11. [HEW77], **Hewitt, Baker,** *Actors and Continuous Functionals,* Technical Report, Computer Science n°194, 1977.
12. [ICPP], ICPP, International Conference for Parallel Programming, 1996, 1998
13. [JER96], **Jernigan,** *Distributed Search Method for Scheduling Flow Through a Factory Floor*, Thesis from the University of Texas, 1996.
14. [LUX92], **Lux,** *A Multi-Agent Approach towards Group Scheduling,* Research Report, n° RR-92-41, 1992.
15. [MAG96], **Magnin,** *Modélisation et simulation de l'environnement dans les systèmes multi-agents,* Thèse IBP, 1996.
16. [SCH96], **Schökle,** *Distribution Simulation Software for Complex Continuous Systems,* 1996.