République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université Mostafa Ben Boulaid - Batna 2

Département d'informatique

MEMOIRE

Présenté pour l'obtention du diplôme de Magistère en Informatique

Option : Système Informatique Intelligent et Communicant **(SIIC)**

Thème

MA-CET : Exécution d'une tâche d'un agent mobile sur plusieurs sites

Présenté par : Dirigé par :

Monsieur Kais YAHIAOUI. Docteur Hamoudi KALLA.

Soutenu le : / /Devant le jury composé du :

Président :	Dr Souheila BOUAM	MCA à Université Batna 2
Rapporteur :	Dr Hamoudi KALLA	MCA à Université Batna 2
Examinateur :	Dr Hamouma MOUMEN	MCA à Université Batna 2
Examinateur :	Dr Mourad AMAD	MCA à Université Bouira

RESUME

L'objectif de ce projet de recherche est de proposé une approche pour résoudre un des problèmes de la mobilité des agents mobiles dans un système multi agents, la continuité de l'exécution de leurs tâches sur plusieurs hôtes d'un réseau hétérogène. La mobilité d'un agent est définie comme suit : son exécution est interrompue sur le nœud courant, appelé nœud source, ensuite les données représentant l'état de l'agent sont transférées du nœud source vers un nœud destination et enfin, arrivé au nœud destination, son exécution se poursuit là où elle avait été interrompue sur le nœud de départ, cette opération, appelée migration du contexte d'exécution.

MASA (Modélisation *multi-**A**gents de **S**ystèmes à base de composants **A**utonomes et hétérogènes)* est une méthodologie de conception et de développement des systèmes multi agents. **CPN** *(Colored Petri Net*). Nous avons adopté cette approche pour résoudre le problème de la continuité de l'exécution des taches de l'agent mobile dans un réseau hétérogène, la continuité de l'exécution a été réalisée par la sauvegarde et restitution du marquage du CPN.

Mots-Clés : Systèmes multi-agent, Agents mobiles, Réseaux hétérogène, Réseau de Petri coloré.

ABSTRACT

The objective of this research project is to propose an approach to solve a mobile agent mobility problem in a multi agent system, the continuity of the execution of their tasks on several hosts of a heterogeneous network. The mobility of an agent is defined as follows: its execution is interrupted on the current node, called the source node, then the data representing the state of the agent is transferred from the source node to a destination node and finally, arrived at the node destination, its execution continues where it was interrupted on the starting node, this operation, called migration of the execution context.

MASA (Multi-Agent System Modeling Based on Autonomous and Heterogeneous Components) is a methodology for designing and developing multi-agent systems. **CPN** (Colored Petri Net). We adopted this approach to solve the problem of the continuity of the execution of the tasks of the mobile agent in a heterogeneous networks, the continuity of the execution was carried out by the safeguarding and restitution of the marking of the CPN.

Keywords: Multi-Agent Systems, Mobile Agents, Heterogeneous Networks, Colored Petri Net.

Remerciements

Avant tout, J'aimerais remercier mon dieu **Allah** qui m'a donné la santé, la volonté et le courage pour me permettre de finaliser ce projet.

Je désire témoigner toute ma gratitude aux membres du jury pour m'avoir fait l'honneur de juger mon travail, sur tous Dr **Mourad AMAD**, qui est venu gracieusement de loin pour discuter de ce travail.

Mes remerciements vont en premier lieu au Dr **Hamoudi KALLA** pour sa confiance, ses efforts et ses encouragements, et pour le soutien qu'il m'a témoigné tout au long de cette étude placée sous sa direction.

J'adresse également mes vifs remerciements et toute ma gratitude au Dr **Souheila BOUAM** pour son dynamisme, son soutien, son humour, ces précieux conseils, sa collaboration fructueuse et sa grande disponibilité qui ont été des atouts précieux pour moi.

Un grand merci à Dr **Hamouma MOUMEN**, Je lui adresse toute ma reconnaissance pour son aide, soutien et collaboration.

Pour finir je tiens à remercier tous qui m'ont aidé de près ou de loin pour l'élaboration de ce travail.

TABLE DES MATIÈRES

CHAPITRE II : MIGRATION DES PROCESSUS

CHAPITRE III : RESEAUX DE PETRI COLORES

CHAPITRE VI : Solution Proposé

LISTE DES TABLEAUX

LISTE DES FIGURES

INTRODUCTION GENERALE

L’évolution rapide des réseaux sans fil et la diffusion de dispositifs portables utilisant le réseau comme infrastructure ont imposé de nouveaux aspects pour les applications réparties, telles que celles à base d’agents mobiles**.** La mobilité en générale concerne le déplacement physique de dispositifs matériels ainsi que la migration des applications logicielles. De ce fait, beaucoup de concepts ont émergé dans le domaine de la mobilité, parmi lesquels nous citons: le Code Mobile, le Calcul mobile et les Agents Mobiles.

La mobilité est une propriété orthogonale des agents, c'est-à-dire, elle ne concerne que certains agents. Un agent peut rester là et communiquer avec son environnement par des moyens classiques, tels que les invocations de méthodes à distance et la messagerie. Nous appelons les agents qui ne sont pas ou ne peuvent pas se déplacer "agents stationnaires." Un agent stationnaire ne s'exécute que sur le système sur lequel il commence son exécution. S’il a besoin d'informations qui ne sont pas dans ce système ou désire interagir avec un agent sur un autre système, il utilise en général un mécanisme de communication, tels que les invocations de méthodes à distance.

En revanche, un agent mobile n'est pas lié au système sur lequel il commence son exécution. Il peut se déplacer parmi les hôtes du réseau. Créé dans un environnement d'exécution, il peut se transporter (de façon autonome) dans un autre environnement d'exécution dans le réseau, où il reprend l'exécution. Cette capacité lui permet de passer à un système contenant un objet donné avec lequel il veut interagir et de prendre avantage d'être dans le même hôte qu’un tel objet.

Les agents mobiles ont plusieurs avantages : Ils réduisent la charge du réseau, ils s’exécutent de façon asynchrone et autonome, ils s'adaptent dynamiquement, ils sont robustes et tolérants aux pannes. Plusieurs applications bénéficient du paradigme des agents mobiles tels que le e-commerce, l'assistance personnelle, la recherche d'informations distribuées et la

surveillance & notification. Cependant, ils posent aussi de nombreuses difficultés telles que les problèmes de leur sécurité et celles de leurs hôtes, … et la continuité de l'exécution de leurs tâches sur plusieurs hôtes d'un réseau. De tels hôtes peuvent être d'un hardware hétérogène ou avoir des plateformes logicielles différentes.

Dans ce travail, nous nous intéressons, plus particulièrement, au dernier problème celui de la continuité de l'exécution des tâches de l'agent mobile. Ce problème est tellement embarrassant au point que certains travaux l'ont évité carrément alors que d'autres proposent des solutions particulières. Ces solutions concentrent leurs efforts sur la mobilité de code.

L'agent mobile doit être exécuter des tâches, une tâche est élémentaire ou est une combinaison de tâches plus élémentaires qu'on qualifie de sous taches. Une tache élémentaire est celle dont l'interprétation se fait par un processus indivisible. Une sous tache est une tache, pouvant être décomposable. Les taches élémentaires peuvent être effectuées séquentiellement et/ou en parallèle, i.e., concurremment.

Dans notre travail, nous avons opté pour la combinaison entre deux approches : **MASA-méthode** (**M**odélisation multi-**A**gents de **S**ystèmes à base de composants **A**utonomes et hétérogènes) **[Lahlouhi, 2002]** comme une méthodologie pour la conception d'un système multi-agent, et les **CPN** (**C**olored **P**etri **N**ets : Réseaux de Petri Colorés) autonome et non autonome **[Jensen, 1994]**, pour résoudre le problème de la continuité de l'exécution d'un agent mobile dans un système multi agents.

La continuité de l'exécution a été réalisée par la sauvegarde et restitution du marquage du CPN. Le programme de l'agent sera transféré et lancé sur les différents nœuds des réseaux alors qu'il continu son exécution. Cette solution est rendue simple, portable et robuste par l'utilisation des CPN qui sont un formalisme de haut niveau d'abstraction pour la modélisation

des comportements des agents. Ce qui nous place à un haut niveau d'abstraction indépendant de la machine exécutante.

Dans ce mémoire nous aborderons les détails de cette étude suivant la structure suivante :

Chapitre1 : Dans ce chapitre, nous étudions les agents mobiles et leurs spécificités. Premièrement, nous commençons par définir en mettant l'accent sur les caractéristiques propres aux agents logiciel, Ensuite, nous étudions la technologie d'agent mobile : généralité, définition, domaines d'application et avantages, les différentes plates-formes d'agents existantes .

Chapitre2 : Ce chapitre est une étude sur les mécanismes de la migration des processus dans des machines hétérogènes, qui est la base de la mobilité des agents mobiles.

Chapitre3 : Dans ce chapitre on a définir les réseaux de Petri et les notions de base des réseaux de Petri colorés.

Chapitre 4 : Dans ce chapitre nous propose une solution qui garantir la continuité d'exécution des taches d'un agent mobile, et comme cas d'étude on a choisi la recherche d'information coopérative dans réseau hétérogène.

Enfin de ce mémoire on a terminons par une conclusion générale.

CHAPITRE- I

Introduction

Le terme d'agent mobile est issu principalement de deux domaines distincts : l'intelligence artificielle et les systèmes distribués avec la migration de processus. La technologie à base d'agent mobile aide à améliorer la performance de plusieurs applications distribuées avec des moyens intelligents. Amélioration peut être ressentie dans la réduction du trafic réseau, la répartition dynamique de charge, la commodité par rapport aux programmeurs ou simplement dans l'habilité de continuer l'interaction avec un utilisateur durant une déconnexion du réseau. L'utilisation des agents mobiles apport plusieurs avantages. Néanmoins, ils posent quelques problèmes, dont les plus importants sont ceux relatifs à l'interopérabilité, la sécurité et la tolérance aux pannes des systèmes d'agents mobiles.

Ce chapitre introduit le concept d'agent mobile, en mettant l'accent tout d'abord sur la notion d'agent (définition, caractéristiques, classification et architecture). Nous donnons ensuite des définitions de l'agent mobile, un aperçu sur les concepts fondamentaux des agents mobiles. Et enfin nous terminons par les travaux antérieurs et récents concernant les agents mobiles.

1. Les agents logiciels

Il existe à l'heure actuelle, une pléthore de définition de l'agent. Nous avons retenu les définitions suivantes :

1.1 Définitions

Définition 1 : « *un agent est un système informatique situé dans un environnement, et qui agit d'une façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu ».* **[Jensen, 98].**

Définition 2 : « *On appelle agent une entité réelle ou abstraite qui est capable d'agir sur elle-même et sur son environnement, qui dispose d'une représentation partielle de cet*

environnement, qui, dans un univers multi-agents, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de sa connaissance et des interactions avec les autres agents. » **[Ferber, 95].**

Parmi ces définitions, nous avons choisi celle de Ferber, un agent est :

- Une entité autonome qui peut offrir des services.
- Une entité dont le comportement est la conséquence de ses objectifs, de sa perception, de ses compétences et des communications qu'elle peut avoir les autres agents.
- Une entité qui possède des ressources.
- Une entité qui est apte à agir sur l'environnement du système auquel il appartient.
- Une entité qui peut communiquer avec les autres agents.
- Une entité qui est capable de se reproduire.

1.2 La structure interne des agents

Dans un agent on distingue essentiellement : le savoir-faire, les croyances, la connaissance de contrôle, l'expertise de l'agent et la connaissance de la communication.

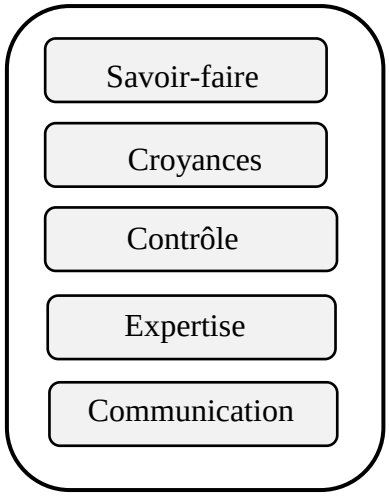


Figure I.1 : Structure interne d'un agent **[Guessoum, 01]**

- Savoir-faire : est une interface permettant la déclaration des connaissances et des compétences de l'agent. Il permet la sélection des agents à solliciter pour une tache donnée.
- Croyances : dans un univers multi-agent, chaque agent possède des connaissances sur lui-même et sur les autres. Ces connaissances ne sont pas nécessairement objectives.
- Contrôle : la connaissance de contrôle dans un agent est représentée par les buts, les intentions, les plans et les taches qu'il possède.
- Expertise : c'est la connaissance sur la résolution du problème.
- Communication : l'agent doit posséder un protocole de communication lui permettant d'interagir avec les autres agents pour une bonne coopération et une bonne coordination.

1.3 Les propriétés des agents

Parmi les nombreuses propriétés pouvant qualifier un agent, on opte les suivantes :

- **L'autonomie :** Permet à l'agent de produire des actions à partir de perception, et éventuellement d'états internes, ou d'une mémoire. Aucune intervention extérieure. Cela signifie que l'agent doit être capable d'accomplir sa mission son supervision ou intervention de tiers (humain ou agent) **[Caire, 07].**
- **La réactivité :** C'est la capacité qu'un agent puisse percevoir les changements environnementaux et modifier son comportement en élaborant des réponses dans les temps requis. **[Caire, 07].** Un agent réactif est un agent qui est capable de prévoir son environnement (qui pourra être utilisateur à travers une interface graphique…) et cela en maintenant un lien constant avec celui-ci afin de répondre aux changements qui y survienne.

- **Sociabilité :** Permet à l'agent de communiquer avec les autres agents, ou avec l'utilisateur par le biais un langage symbolique **[Hong, 07] [Panzoli, 08]** dont le but de résoudre son problème et d'aider les autres dans leurs activités.

- **Situation :** Un agent est un système informatique situé dans un environnement, cela signifié que l'agent peut recevoir des entrées sensorielles provenant de son environnement et qu'il peut effectuer des actions qui sont susceptibles de changer cet environnement.

- **L'adaptabilité :** C'est la capacité de l'agent à s'adapter aux changements de l'environnement. Pour cela l'agent adaptatif possède des mécanismes d'apprentissage et/ou d'évolution, qui s'appliquent durant la vie de l'agent dans l'environnement.

1.4 Taxonomie des agents

1.4.1 Les agents réactifs

Ils possèdent quelques compétences leur permettant d'accomplir une ou plusieurs actions, ces compétences sont statiques et n'évoluent pas a travers le temps, et ceux, a cause du fait qu'ils ne peuvent pas tenir compte des actions passées parce qu'ils n'ont pas d'histoire c'est-à-dire qu'ils ne possèdent pas de mémoire. Leur comportement est de type « stimuli réponse ».

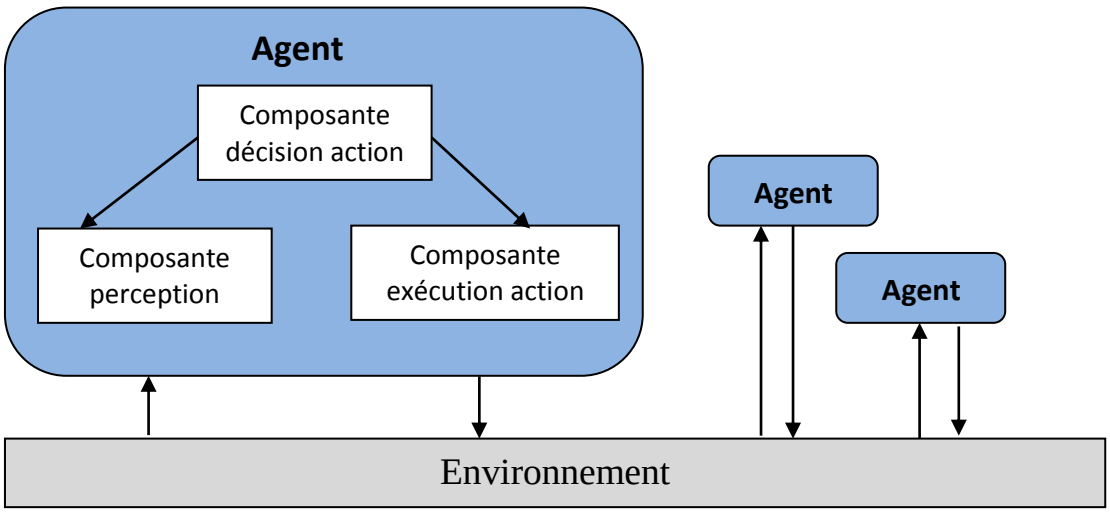


Figure I.2 : structure général d'un Agent réactif **[Florea, 02].**

1.4.2 Les agents délibératifs

Les agents délibératifs, également appelés cognitifs **[Panzoli, 08],** sont issus des premiers travaux de IA. Et d'après **[Guessoum, 01]** sont les agents qui disposent d'une capacité de raisonnement, d'une aptitude à traiter des informations diverses liées au domaine d'application, et des informations liées à la gestion des interactions avec les autres agents et l'environnement.

Les deux caractéristiques principales de l'agent cognitif sont :

1. Une forte intentionnalité. aspire continuellement à la réalisation d'un certain nombre de buts explicitement définis par le propriétaire **[Panzoli, 08].**

2. L'importance centrale du processus de décision. Synthétiquement, l'agent est capable d'évaluer toutes les situations avant de choisir la meilleure action, en adéquation avec les buts à accomplir **[Panzoli, 08].**

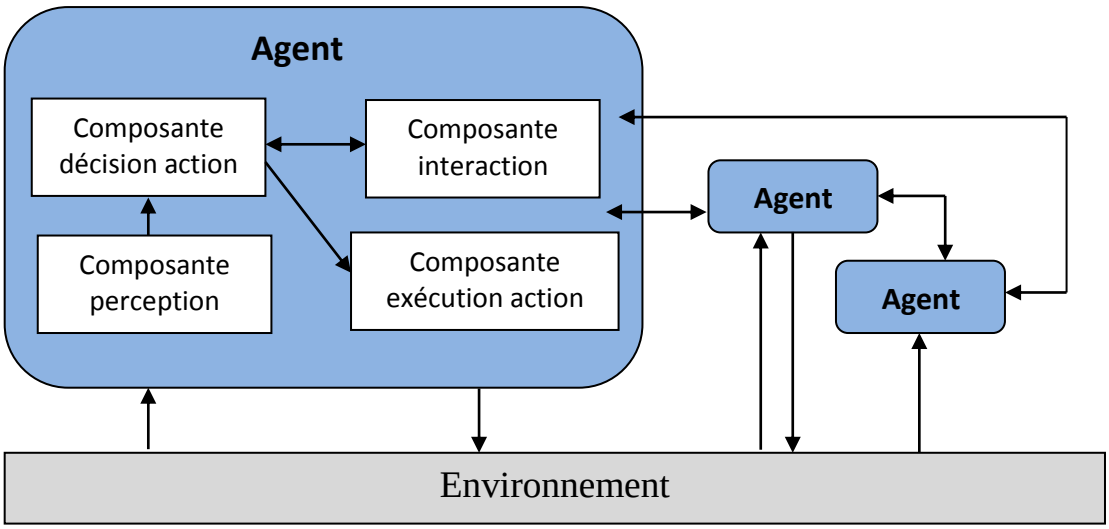


Figure I.3: structure général d'un Agent cognitif **[Florea, 02].**

1.4.3 Les agents hybrides

C'est une architecture composée d'un ensemble de modules organisés dans une hiérarchie, chaque module étant soit une composante cognitive, soit une réactive. De cette manière, on combine le comportement proactif de l'agent, dirigé par les buts, avec un comportement réactif aux changements de l'environnement. En plus, on espère obtenir simultanément les avantages des architectures cognitives et réactives, tout en éliminant leurs limitations **[Florea**, **02].**

2. Les Agents mobiles

La mobilité est considérée dans la majorité des cas comme une propriété orthogonale des agents, qui ne sont pas tous forcement mobiles. Il existe deux sorte d'agents, ceux qui s'exécutent uniquement sur le système sur lequel ils sont créés, on appelle se genre les agents stationnaires, et ceux qui ont la capacité de se migrer d'un site à un autre, sont les agents mobiles. Les agents stationnaires ne peuvent pas se déplacer sur le réseau, s'ils ont besoins d'informations ou d'interagir avec un agent sur un autre système, ils utilisent généralement les mécanismes de communication classiques tel que l'appel de procédure à distance (RPC). En revanche, l'agent mobile n'est pas fixé au système sur lequel il commence l'exécution **[Jensen 98],** il est libre de voyager à travers un réseau hétérogène, d'un hôte à un autre au cours de son exécution accéder à des données ou à des ressources nécessaires pour l'accomplissement de sa mission. Il se déplace avec son code, ses données propres et aussi avec son état d'exécution.

Le paradigme des agents mobiles propose d'utilisé la migration du code afin de supprimer la contrainte de connexion constante. Les deux parties doivent se connecter seulement durant la phase de migration. La figure suivante illustre ce paradigme, un client donne une mission à un agent, qui pour la réaliser se déplace dans le réseau de machines accédant localement aux services offerts par ces machines. On distingue trois phases :

- L'activation de l'agent mobile avec la description de sa mission.
- L'exécution de la mission par l'agent qui se déplace pour accéder aux services.
- La récupération éventuelle des résultats de l'agent mobile.

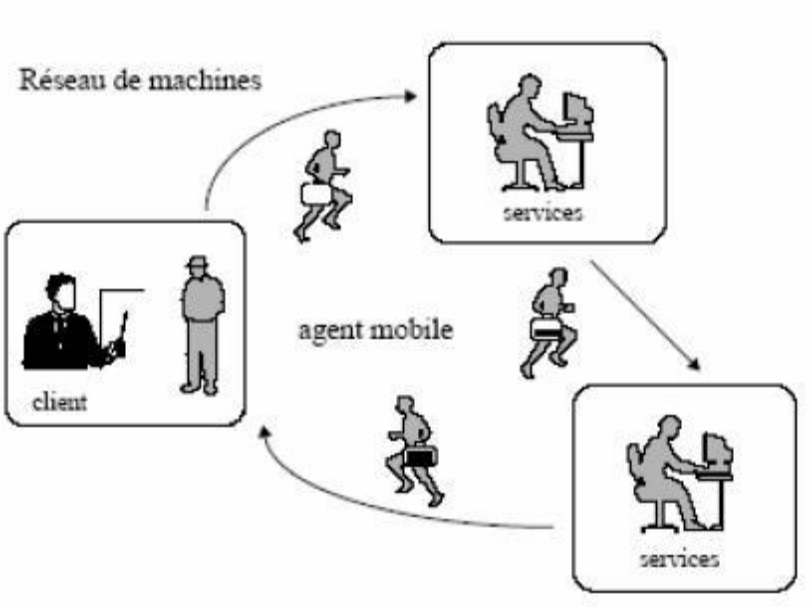


Figure I.4 : le paradigme des agents mobiles. **[Perret, 97]**

2.1 Quelques Définitions

Les agents mobiles sont définis comme étant :

Définition 1 : J. Ferber a définit l'agent mobile comme *« Un agent logiciel qui peut se déplacer d'un site à un autre en cours d'exécution pour se rapprocher de données ou de ressources. Il se déplace avec son code et ses données propres, mais aussi avec son état d'exécution. L'agent décide lui même de manière autonome de ses mouvements. En pratique, la mobilité ne se substitue pas aux capacités de communication des agents mais les complète (la communication distante, moins coûteuse dans certains cas, reste possible). Afin de satisfaire aux contraintes des réseaux de grande taille ou sans fil (latence, non permanence des liens de communication), les agents communiquent par messages asynchrones. »* **[Ferber 95].**

Définition 2 : *Un programme qui migre à sa propre initiative d'une machine à une autre dans un réseau hétérogène. Dans chaque machine l'agent interagi avec les services des agents stationnaires et d'autres ressources afin d'accomplir une tache particulière, définie par son propriétaire.* » **[Bernichi, 09].**

Les agents mobiles sont des programmes dotés des capacités d'adaptabilité, d'autonomes qui peuvent se déplacer à travers un réseau hétérogène d'un hôte vers un autre sous leur propre contrôle. Dans le but est de fournir un service précis à son propriétaire

2.2 Attributs d'un agent mobile [Braun, 05]

Un agent mobile est une entité qui possède cinq composants : son état, son implémentation, son interface, son identifiant et son autorité. L'agent transporte ses attributs lors son déplacement à travers le réseau.

a. L'état : L'état d'un agent peut être considéré comme une photo instantanée de son exécution, dans la plupart des langages de programmation. On peut partitionner cet état en état d'exécution (qui inclut ses programmes et ses interfaces) et un état d'objet (qui contient les valeurs des variables d'instance dans un objet).

b. Le code: C'est la logique de l'agent. Il contient les différentes fonctionnalités de l'agent. Les agents de même type ont le même code. Ce code doit être identifiable, lisible et surtout facilement séparable de sa plateforme locale pour pouvoir être éventuellement transférer à une plateforme distante. Pour cela il est généralement sous forme de code interprété.

c. L'interface : Un agent fournit une interface qui permet aux autres agents et systèmes d'interagir avec lui. Cette interface peut être un ensemble de méthodes qui permet aux agents et applications d'accéder aux méthodes de l'agent par un système de messagerie.

d. L'identifiant : Chaque agent possède un identifiant unique durant son cycle de vie, qui lui permet d'être identifié et localisé. Puisque l'identifiant est unique, il peut être utilisé comme clé dans les opérations qui exigent un moyen pour référencer une instance particulière d'agents.

e. L'autorité : Une autorité est une entité dont l'identité peut être identifiée par n'importe quel système auquel elle essaye d'accéder. Pour les agents il existe principalement deux types d'autorité :

- Le fabriquant (manufacturer) qui est le fournisseur du code d'implémentation de l'agent.
- Le propriétaire (owner) qui a la responsabilité du comportement de l'agent.

2.3 Caractéristiques des agents mobiles

Les principales caractéristiques d'un agent mobile sont :

1. La mobilité : c'est le mécanisme utilisé pour transporter l'agent entre les différents sites. Le déplacement d'un agent est contrôlé par la fonction de migration faible (transport du code et de données) ou forte (transport du code, de données et de l'état de l'exécution) **[Menacer, 04].**

2. La communication : les agents mobiles peuvent communiquer sur le même site (communication locale) ou sur des sites différents (communication à distance). Les moyens de communication sont multiples tel que les envois des messages, le tableau blanc, les points de rendez-vous.

3. La gestion des ressources : l'agent mobile contrôle ses propres ressources afin d'optimiser leurs performance et satisfaire aux exigences des hôtes.

4. La tolérance aux pannes : comme un agent mobile s'exécute sur un site distant, il pourra tomber en panne (déconnexion, défaillance,…) il faut donc définir des mécanismes de récupération pour relancer l'agent à nouveau, une certaine redondance

(copie par exemple) est nécessaires.

5. La sécurité : l'agent mobile doit s'assurer que ni son code ni ses données ne seront modifiés par le système hôte.

6. Sérialisation/Désérialisation : l'agent mobile est composé du code, données et état d'exécution. Avant d'être migré à une autre machine, ceux-ci doivent être codés sous la forme d'une chaine de caractère ; on dit que l'agent est sérialisé (processus de sérialisation). La désérialiation est le processus inverse **[Menacer, 04].**

2.4 Les types d'agents mobiles

Plusieurs types d'agents mobiles peuvent exister. On détaille ci-dessous les principaux types d'agents mobiles qu'on a rencontrés dans la littérature :

- **Agent statique :** Un agent statique est un agent mobile qui n'exécute en général qu'une seule migration. Cette migration s'effectue depuis la station de départ vers un nœud bien défini au lancement. A l'arrivée sur le nœud, l'agent mobile ne migre plus mais exécute une tâche prédéfinie.
- **Agent visiteur :** Un agent visiteur est un agent mobile qui visite successivement les différents nouds de la plate-forme afin d'y appliquer la même fonction d'administration du réseau, par exemple, en récupérant les versions des systèmes d'exploitation installées sur les éléments du réseau.
- **Collecteur de données** : Un agent collectant les données est un agent mobile qui nécessite un point de rendez-vous avec un ou plusieurs autres agents mobiles pour les

décharger des données que ces derniers ont collectées. Après la collecte, cet agent transporte les données jusqu'à un système défini à l'avance.

- **Agent de suivi d'incident :** Un agent de suivi d'incident est un agent qui est programmé pour réagir aux données qu'il analyse.

2.5 Services requis pour l'exécution d'agents mobiles

Dans cette section, nous allons décrire les fonctions nécessaires pour construire des applications reparties à base d'agents mobiles, nous allons présenter les services suivants :

- ✓ création d'un agent,
- ✓ transfert d'un agent,
- ✓ désignation d'un agent, c'est-à-dire offrir un nom globalement unique a un agent,
- ✓ localisation d'un agent,
- ✓ communication entre agents,
- ✓ exécution d'un agent,
- ✓ protection d'un agent.

Ces différents services sont détailles dans les paragraphes suivants.

2.5.1 Création d'agents mobiles

Dans la mesure où ces agents peuvent se déplacer, on peut se poser la question, lors de leur création, de l'endroit où l'on souhaite qu'ils démarrent leurs activités. Une fois crée, l'agent peut être actif, c'est-à-dire que l'exécution de son code est déclenchée. La création/lancement d'un agent peut prendre plusieurs formes :

- Création locale,
- Création et exécution à la demande (code on demand),
- Création et exécution à distance (remote exécution).

La création de l'agent n'est pas directement liée à la mobilité, mais a des mécanismes sur lesquels s'appuie la mobilité. A sa création, un nom globalement unique doit être attribue à l'agent ce qui va permettre de localiser l'agent et de communiquer avec lui.

2.5.2 Migration d'un agent

La migration permet le transfert d'un agent en cours d'exécution d'un site à un autre à travers le réseau. Nous allons prendre le cas d'un agent qui désire migrer entre deux machines avec la possibilité de recevoir des messages pendant son déplacement. Cette migration comporte les étapes suivantes **[Ismail, 99]** :

1. la sérialisation du contexte de l'agent et la production d'un message incluant le contexte sérialise de l'agent et son code;

2. le processus de l'agent étant suspendu sur sa machine d'origine, toutes les communications avec d'autres agents sont donc suspendues et l'agent est détruit;

3. l'envoi du contexte et du code de l'agent vers la machine de destination;

4. l'état de l'agent migrant est restaure sur la machine de destination, un nouveau processus léger est créé pour la poursuite de son exécution;

5. les communications en cours sont redirigées vers la machine de destination, l'agent n'étant pas encore active, ces messages seront traites après sa réactivation;

6. réactivation de l'agent sur la machine de destination, dès que la partie de son contexte nécessaire à son exécution est reçue ; la machine de destination s'occupe des liens dynamiques entre l'agent et son code. A partir de ce moment, la migration prend fin. Le processus sur la machine d'origine peut être définitivement efface.

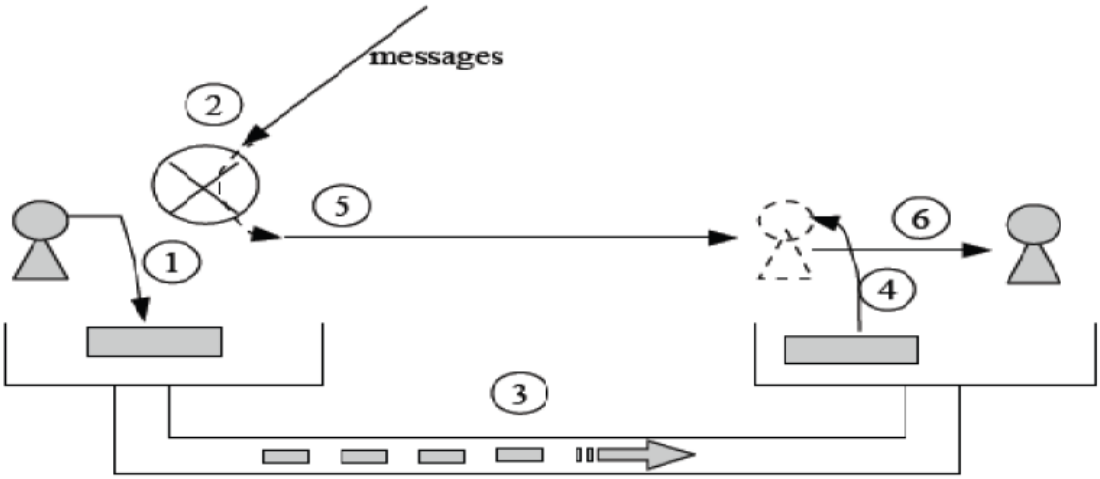


Figure I.5 : Migration d'un agent mobile

Les différentes stratégies de migration d'agents dépendent de la manière dont sont traitées ces diverses étapes et des données transférées avec l'agent lors de sa migration.

Deux types de migrations ont été proposés dans les plates-formes existantes :

1. La migration forte permet à un agent de se déplacer quel que soit l'état d'exécution dans lequel il se trouve. Dans ce type de migration l'agent se déplace avec son code, son contexte d'exécution et ses données. Dans ce cas, l'agent reprend son exécution après la migration exactement là où elle était avant son déplacement. La migration forte nécessite un mécanisme de capture instantanée de l'état d'exécution de l'agent. Elle peut être proactive ou réactive. Dans la migration proactive, la destination de l'agent est déterminée par l'agent lui-même. Dans la migration réactive, la migration de l'agent est dictée par une partie ayant une relation avec l'agent mobile.

2. La migration faible ne fait que transférer avec l'agent son code et ses données. Sur le site de destination, l'agent redémarre son exécution depuis le début en appelant la méthode qui représente le point d'entrée de l'exécution de l'agent, et le contexte d'exécution de l'agent est réinitialisé. Pour que l'agent se branche sur une instruction particulière de son code après sa migration, le programmeur doit inclure dans l'état de l'agent des moments privilégies (explicites)

dans le code de l'agent (point d'arrêt) pour pouvoir le relancer. La migration forte, bien que beaucoup plus exigeante à implanter **[Dillenseger, 02]** que la migration faible **[Grasshopper 98],** n'en est pas moins indispensable pour toutes les applications pour lesquelles les notions de fiabilité et de tolérance aux pannes sont primordiales.

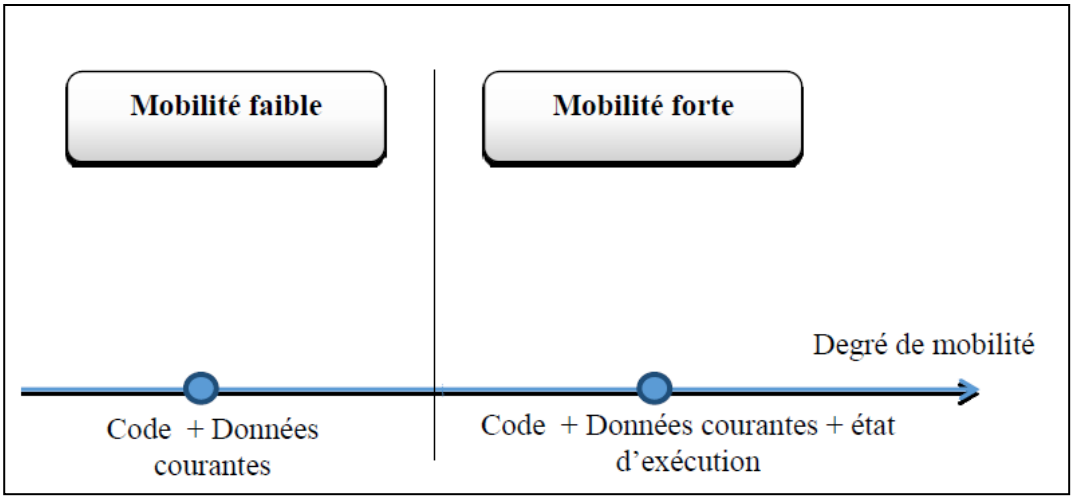


Figure I.6 Degré de mobilité

2.5.3 Service de nommage

Dans un système à agents mobiles, les agents ont besoin de communiquer entre eux, ce qui nécessite de les nommer. Le nom donne à l'agent doit être globalement unique. Dans la plupart des systèmes à agents mobiles ce nom est construit à partir du nom de la machine (ou adresse IP) où l'agent s'exécute, d'un numéro de port et d'un identificateur localement unique.

2.5.4 Service de localisation

Afin de communiquer avec un agent mobile, il est nécessaire de le retrouver. La mobilité de l'agent introduit des contraintes sur les communications qui n'étaient pas présentes avec les systèmes classiques ou les objets ne pouvaient pas changer de lieu d'exécution. En particulier, il faut que des entités mobiles puissent communiquer indépendamment de leur localisation et de leur mobilité **[Alouf, 02]**. Le point le plus important concerne la fiabilité, il faut pouvoir assurer les communications à tout moment avec les objets mobiles. Pour ces raisons, un système à

agents mobiles doit offrir un service de localisation des agents à travers un serveur de noms. Ce serveur de noms contient la localisation courante de l'agent ou bien suffisamment d'informations pour le localiser.

2.5.5 Communications entre les agents

La communication entre deux entités peut se faire de deux façons, la plus intuitive consiste à avoir un mécanisme permettant une communication directe entre les objets, ce qui correspond à la communication synchrone ou asynchrone entre les deux entités.

Une deuxième manière de la faire est d'avoir des mécanismes de communication indirecte. Dans ce cas un objet voulant communiquer envoie son message à un objet tiers qui se charge à son tour de l'envoyer au destinataire.

Dans un système à agents mobiles, il faut différencier deux types de communication, selon que la communication intervient entre deux agents ou entre un agent et un groupe d'agents. Les moyens de communication entre les agents sont multiples, nous pouvons citer la communication par message, session, tableau blanc, rendez-vous, par évènement distribué ou local.

2.5.6 Exécution d'un agent

Un système à agents mobiles doit offrir la possibilité d'exécuter un agent sur les machines qui vont accueillir ce dernier, ainsi l'agent utilise les ressources disponibles sur la machine afin d'accomplir la tache demandée par le client. Sur la machine d'accueil, les agents mobiles peuvent être amenés à effectuer des entrées/sorties, accéder à l'interface utilisateur, au système de gestion de fichiers, aux applications externes et à l'interface réseau. Les machines qui forment notre système d'exécution n'auront pas toutes le même type de système d'exploitation. Par conséquent, maintenir l'indépendance vis-à-vis du système d'exploitation d'un environnement d'agents mobiles tout en autorisant en même temps l'accès aux ressources est un problème important à surmonter dans les environnements d'exécution d'agents mobiles. C'est pour cette raison que la

plupart des environnements d'agents mobiles utilisent le langage Java comme langage d'implémentation du code mobile **[Gomoluch, 01].**

Java est un langage oriente objet développe par Sun et présente officiellement en 1995. Les applications écrites en Java [Java] sont compilées en bytecode et exécutées sur une machine virtuelle, la Java Virtuel Machine (JVM). Le succès du langage Java se traduit, entre autre, par le fait qu'on peut raisonnablement considérer que la machine virtuelle Java constitue aujourd'hui un environnement d'exécution "universel", car disponible sur toutes les machines d'un système reparti. Cette propriété en fait une plateforme de développement de choix pour les applications mobiles ou distribuées car le programmeur n'a pas à gérer différents langages ou différentes versions d'un programme suivant les systèmes sur lequel il va s'exécuter. En plus de cette propriété liée a la portabilité, Java propose toute une série de services qui permettent de construire des applications distribuées mobiles. On trouve notamment :

- ✓ la sérialisation/de sérialisation d'un objet afin de le transmettre à travers le réseau.
- ✓ la communication entre deux objets situés sur des machines distantes (RMI, socket, etc...)
- ✓ le chargement dynamique de code à partir d'un site distant.

L'utilisation de Java permet de résoudre le problème lie à l'exécution de l'agent sur les différentes machines du système repartie. Mais avec la diversité des plates-formes disponibles, nous pouvons nous interroger sur la migration entre les différentes plates-formes.

Chaque plate-forme possède sa propre façon pour définir le comportement des agents, ce qui pose un problème de compatibilité lors de la migration d'un agent entre les différentes plates-formes. Pour surmonter ce problème, des travaux comme dans **[Groot, 04]** proposent un langage générique, pour la description du comportement de l'agent, base sur le langage XML. Une projection de cette description permet à une plate-forme d'agents mobiles de reconstruire l'agent avec son propre langage.

L'exécution d'un agent mobile sur une machine hôte pose le problème du piratage des sites visites. Ce problème, non spécifique aux agents mobiles, est symétrique pour ces derniers. En effet, un agent mobile est expose au problème du piratage par les logiciels qui fonctionnent sur les sites qu'il visite. Dans la section suivante, nous allons détailler les problèmes liés à la sécurité dans un environnement d'agents mobiles.

2.5.7 Sécurité

Garantir la sécurité dans les environnements d'agents mobiles est important du fait de la nature même du modèle d'exécution des agents et du fait des interactions des agents mobiles avec plusieurs systèmes et ressources. Un système à agents mobiles doit offrir des mécanismes permettant une exécution sécurisée des agents au sein du système. Trois types de problèmes de sécurité ont été identifiés pour les environnements d'exécution d'agents mobiles:

- **La sécurité du site d'accueil contre un agent :** Au cours de son exécution, un agent mobile peut avoir accès aux ressources du site sur lequel il est situé. Un agent malveillant peut donc profiter des accès aux ressources locales pour propager des virus, des worms ou des chevaux de Troie. Il peut masquer sa véritable identité et lancer une tâche lourde ce qui va entrainer l'indisponibilité du service **[Bellavista, 01]**. La machine qui va accueillir l'agent mobile doit être amenée à détecter le mauvais déroulement de l'exécution de l'agent. Dans **[Galtier, 01],** les auteurs présentent une méthode qui permet à l'agent de définir ses besoins d'une façon indépendante de la machine sur lequel il s'exécute. Le site d'accueil peut détecter les abus commis par l'agent permettant ainsi de le contrôler. Une approche classique pour protéger les sites des agents malveillants est de limiter les accès des agents aux ressources locales du site **[Hantz, 06].** Dans cette approche, le comportement de l'agent est écrit sous la forme d'un langage interprété. L'agent est exécuté dans un environnement qui a son tour s'exécute au-dessus du site. Le contrôle de l'agent est ainsi fait dans l'environnement d'exécution **[Carvalho, 04].** Une seconde approche consiste à ne laisser s'exécuter que les

agents authentifies, une signature est attribuée à l'agent permettant au site d'accueil de l'authentifier avant son exécution.

- **La sécurité d'un agent contre le site d'accueil :** La protection des agents mobiles contre des sites hostiles est spécifique au domaine. Les agents sont à protéger à deux niveaux : la protection du code de l'agent (changement du comportement) et la modification de l'état de l'agent. Plusieurs approches ont été proposées: l'approche organisationnelle permet, seulement aux sites dignes de confiance, de gérer des systèmes d'agents mobiles; l'approche de détection de manipulation offre des mécanismes fondés sur le traçage permettant de détecter les manipulations de données effectuées sur un agent **[Diaz, 01]**; enfin l'approche de protection par boite noire. La cryptographie mobile est une étape vers l'approche de protection par boite noire. La spécification de l'agent est convertie en code exécutable et en un ensemble de données encryptées **[Claessens, 03].** Le cryptage de données empêche un éventuel attaquant de lire ou de modifier les données. Dans **[Benachenhou, 05],** les auteurs présentent une nouvelle approche qui consiste à exécuter un agent clone sur un site sur afin de surveiller l'exécution du vrai agent. Cette solution engendre une augmentation dans le trafic réseau (échange entre l'agent et son clone).

- **La sécurité d'un agent contre un autre agent :** L'agent doit être protégé contre une éventuelle attaque par un autre agent **[Hohlfeld, 02]** situé sur le site hôte ou sur un autre site. Un agent doit avoir sa propre politique de sécurité qui peut être soit assurée par l'agent lui-même, soit par le site d'accueil.

2.6 L’apport des agents mobiles

Les agents mobiles ont rapidement suscité un intérêt tout particulier dans les domaines de recherche portant sur les applications réparties. Très rapidement, cette nouvelle méthode de programmation a été évaluée afin de voir ce qu’elle pouvait apporter comme caractéristiques propres. La raison de cet intérêt, est en fort simple, les agents mobiles permettent de dépasser les

limitations des méthodes de programmation classiques (modèle client/serveur). Dans cette section, nous allons exposer ces différents apports.

2.6.1 La performance

Les premières améliorations apportées par les agents mobiles portent sur le gain de performances dues à une meilleure utilisation des ressources physiques mises à disposition.

L'amélioration va intervenir à différents niveaux permettant d'optimiser la tâche globale des agents **[Cubat, 05].**

a) **Diminution de l'utilisation du réseau :** Le déplacement des agents mobiles permet de réduire significativement, voir supprimer, les communications distantes entre les clients et les serveurs. En privilégiant les interactions locales, l'utilisation du réseau va se limiter principalement au transfert des agents. Cette situation présente trois principaux avantages.

- la diminution de la consommation de bande passante.
- la diminution des temps de latence.
- des brèves périodes de communication.

b) **Des calculs indépendants :** Avec les agents mobiles, un client peut déléguer les interactions avec le service sans maintenir une connexion de bout en bout. Avec cette possibilité d'un calcul indépendant, le client peut demander un service, se déplacer (ou simplement terminer une session) puis venir récupérer les résultats plus tard.

c) **Optimisation du traitement :** L'optimisation des phases de traitement se produit à deux niveaux. Premièrement, en localisant les ressources et le savoir faire sur un même site, on supprime les phases de dialogue entre le client et le serveur qui sont perturbées par des temps de latence dus aux communications réseaux Ensuite, le déplacement du savoir faire va permettre de déléguer les calculs sur des machines serveurs, un super - calculateur par exemple, qui sont

généralement plus puissantes qu'une machine cliente. Cela est particulièrement vrai dans l'informatique nomade où la miniaturisation s'accompagne d'une perte de puissance significative.

d) **Tolérance aux fautes physiques :** En se déplaçant avec leur code et données propres, les agents mobiles peuvent s'adapter facilement aux erreurs systèmes. Ces erreurs peuvent être d'ordre purement physique, disparition d'un nœud par exemple, ou d'ordre plus fonctionnel, arrêt d'un service par exemple. Si on prend le cas d'un site perdant une partie de ses fonctionnalités, un service tombant en panne, l'agent pourra alors choisir de se déplacer vers un autre site contenant la fonctionnalité désirée. Ceci permet une bien meilleure tolérance aux fautes que le modèle statique classique.

Cependant, notons que dans le cas d'un agent unique, s'il vient à disparaître pour une erreur interne à son savoir faire ou pour une erreur système, il est généralement très difficile de détecter sa disparition. énumère plusieurs mécanismes basés sur la réplication, transparents aux agents, permettant de résoudre en partie ce problème.

2.6.2 La conception

Avec la méthode classique, il va être très contraignant de décrire des algorithmes d'exploration (de réseau) ou bien encore de caractériser les déplacements des utilisateurs nomades. Avec les agents mobiles, les concepteurs disposent d'une méthode qui permet de décrire naturellement ce genre de comportement. Ainsi, on peut facilement mettre en place un déploiement et/ou une maintenance d'application sur un réseau ou encore suivre les utilisateurs dans leurs déplacements.

2.6.3 Le développement

Le domaine des agents mobiles étant encore relativement jeune, ils se heurtent à de fortes contraintes lors des phases de développement. Qu'il existe à l'heure actuelle un trop grand

nombre d'intergiciels pour agents mobiles qui possèdent chacun leurs propres défauts et qualités. Avec cette offre pléthorique, il est difficile de parler de standardisation et aucune ne semble encore s'imposer. Le manque de standardisation se retrouve aussi dans les interactions entre agents, ainsi il n'existe pas de langage partagé par toutes les plates-formes.

2.6.4 La sécurité

Pour ce qui est des agents mobiles, d'un point de vue de la sécurité, nous sommes face à un problème qui n'est pas encore intégralement résolu. C'est d'ailleurs le principal argument avancé pour expliquer la faible utilisation de ce paradigme. En effet, les agents mobiles représentent un nouveau champ d'investigation pour le domaine de recherche en sécurité, d'une part dans la protection des sites vis-à-vis des agents malveillants et d'autre part dans la protection des agents vis-à-vis des sites malveillants.

Selon leurs caractéristiques les agents mobiles ne construisent pas une solution idéale pour tous les types des applications répartis, mais simplement une possibilité nouvelle pour la construction des applications.

2.7 les normes des agents mobiles

Afin d'assurer un niveau d'interopérabilité entre les différentes plateformes d'exécution d'agents mobiles. Il existe au temps présent deux normes principales : il s'agit de la norme MASIF (Mobile Agent System Interoperability Facility) **[Milojicic, 98]** et de la norme FIPA (Foundation for Intelligent Physical Agents) **[FIPA, 02].**

2.7.1 MASIF

MASIF **[Milojicic, 98]** est un standard, pour les systèmes à base d'agents mobiles, qui a été spécifiée par l'Object Management Group (OMG), qui se préoccupe généralement de l'hétérogénéité entre les systèmes **[Bernichi, 09].** L'objectif principal de ce travail était de

proposer un ensemble de définitions et d'interfaces afin d'avoir d'un niveau d'interopérabilité entre différents agents mobiles. Il ne s'agit pas de standardisé les opérations locales des agents. Plutôt MASIF standardise la manière de gérer le code des agents, leur identification, la migration et l'adressage local.

MASIF est une collection de définitions et d'interfaces, permettant une interopérabilité pour les SMA mobiles, ce système présente deux interfaces **[Loulou, 10]:** MAFAgentSystem et MAFFinder. Ces deux interfaces ont été définies au niveau de système d'accueil des agents et n'ont pas au niveau de l'agent.

L'interface MAFAgentSystem définit les opérations pour la gestion du cycle de vie de l'agent, assure aussi le transfert et la réception des classes d'agents mobiles. La deuxième interface MAFFinder se préoccupe de l'enregistrement et la localisation des agents.

2.7.2 FIPA

FIPA est une organisation de standardisation fondée en 1996 à Genève (suisse) **[Loulou, 10].** La communauté d'origine de FIPA étant celle des systèmes multi-agents, plus proche de l'intelligence artificielle, elle va se situer à un niveau plus élevé, c'est-à-dire le niveau applicatif en décrivant les éléments nécessaires à la réalisation d'une application et principalement en détaillant la communication entre agents. **[Cubat, 2005] [Bernichi, 09].** Les spécifications FIPA sont réparties en cinq catégories **[Loulou, 10] :**

- Les applications : des exemples de domaines d'application sur lesquels peuvent être déployé des agents FIPA.
- Les architectures abstraites : s'intéressent à la définition des entités abstraites nécessaires pour le développement d'un environnement d'agent.
- La gestion des agents : traitent le contrôle et la gestion des agents dans/entre les plateformes d'agents.

- Le transport des messages d'agents : traitent le transport et la représentation des messages à travers différents protocoles réseaux.
- La communication des agents : ces spécifications traitent les messages ACL (Agent Communication Lanquage), les protocoles d'échange de message, etc.

FIPA a publié plusieurs documents qui présentent les spécifications pour ces différentes catégories. Ces spécifications ont passé par plusieurs versions FIPA 97, FIPA 98 et FIPA 00.

Ces deux normes tendent plus vers la complémentarité que vers la divergence **[Bernichi, 09].** Leurs différences résident dans le fait que MASIF s'intéresse à l'interopérabilité des agents mobiles, elle lui permettre de migrer entre les systèmes d'agents mobiles du même profil par l'intermédiaire des interfaces normalisées de CORBA, contrairement à FIPA qui permet l'interopérabilité des agents intelligents par l'intermédiaire de la communication normalisée des agents et des langues.

2.8 Travaux concernant les agents mobiles

Pour déployer une application à base d'agent mobile, if faut disposer d'une plateforme appropriée.

On distingue trois approches pour concevoir et implanter une plateforme d'agent mobile. La première consiste à recourir à un langage de programmation qui comprend des instructions pour les agents mobiles. La deuxième approche consiste à implanter le système d'agents mobiles comme des extensions du système d'exploitation. Enfin, la dernière approche construit la plateforme comme une application spécialisée qui tourne au –dessus d'un système d'exploitation **[Pierre, 03].**

Dans cette section, on va présenter quelques travaux concernant les agents mobiles. Ces plateformes se diffèrent dans leurs buts, motivations et implémentation mais elles fournissent toutes des fonctionnalités communes qui supportent la migration, la communication entre agents,

plusieurs langages de programmation et plusieurs formes de sécurité. Nous allons commencées par les travaux antérieurs, qui constituent les prémices de la notion d'agent mobile, allant jusqu'aux travaux qui sont apparut récemment.

2.8.1 Telescript

La notion d'agent mobile dans les réseaux est apparue en 94 avec Telescript, qui est développé par la société américaine General Magic, il est appliqué dans le commerce électronique. Il utilise différents concepts : les places et les agents. Les places sont constituées par les zones de services et les stations clientes du réseau.

Les agents sont des entités capables de :

- Migrer entre les places par l'instruction *go*.
- Rencontrer des autres agents sur la même place par l'instruction *meet*.
- Communiquer par établir des connexions permettant l'envoie de message à des agents qui se trouvent dans d'autres places.

Telescript est un langage interprété proche de LISP intégrant la notion d'objets. Il fournit deux niveaux de langage : High Telescript et Low Telescript. La migration d'un *agent* est réalisée au niveau de la machine Low Telescript.

2.8.2 Agile Applets (Aglets)

Aglets est une plateforme d'agents mobiles développée par IBM Japon, un Aglets est un objet Java mobile qui visite des environnements d'exécution (serveur). L'architecture des Aglets est similaire à celle des applets Java. Le système Aglets a son propre protocole pour le transfert des Aglets entres les hôtes : Aglet Transfer Protocol **[Pierre, 03].**

2.8.3 Mole

Mole est une plateforme d'agents mobiles construite à l'Université de Stuttgart en Allemagne. **[Pierre, 03]** Elle implémente la migration partielle, ou seuls les données et l'état de l'agent sont transférés. La migration partielle a été développée après que les bâtisseurs du système se soient rendu compte que la migration totale (déplacement de processus d'exécution de l'agent) était trop couteuse quand il s'agissait d'agents « multi –threads ». Un agent est traité comme une grappe d'objet Java, un ensemble fermé sans aucune référence avec l'extérieur, excepté avec le système hôte.

2.8.4 Voyager

Voyager est un produit commercial proposé par ObjectSpace. Il s'agit d'un système « branché » incorporant plusieurs normes industrielles populaires à l'heure actuelle, notamment CORBA, DCOM et Distributed JavaBeans. Différent des autres systèmes, il ne s'agit pas d'un système dédié uniquement aux agents mobiles : toute sorte de développement distribué est supportée. Dans leur documentation, cependant, le concept d'agent mobile est explicitement présenté. Puisqu'il s'agit d'un système bien soigné, d'une nature générale, et compatible avec les normes existantes, sa popularité est croissante **[Objectspace @].**

2.8.5 Java Agent DEvelopment Framework (Jade)

JADE (*Java Agent Development Framework - Bellifemine, Poggi, Rimassa, 1999*) est une plate-forme qui permet de construire des systèmes multi agents (*SMA*) entièrement implémenté en JAVA ; développé par le groupe CSELT1 Telecom Italia avec la coopération de l'université de Parme (*Italie*) **[Bernichi, 09].**

JADE est un middleware qui facilite le développement des systèmes multi agents (*SMA*). JADE contient :

- **Un runtime Environment** : l'environnement ou les agents peuvent vivre. Ce runtime environment doit être activé pour pouvoir lancer les agents.
- **Une librairie de classes :** que les développeurs utilisent pour écrire leurs agents.
- **Une suite d'outils graphiques :** qui facilitent la gestion et la supervision de la plateforme des agents **[Caire, 07].**

JADE est basée sur le protocole IIOP conforme aux normes FIPA (*Foundation four Intelligent Physical Agents*). **[Bernichi, 09]** La communication des agents est basée sur l'envoi de messages FIPA-ACL (langage de communication des agents), qui est le langage standard des messages et impose le codage et la sémantique des messages.

Jade est compatible avec la plupart des configurations matérielles (PC, MAC, OS) et logicielles (Windows, Unix). Le seul système nécessaire pour son fonctionnement est la machine virtuelle JAVA.

Chaque instance du JADE est appelée conteneur " container ", et peut contenir plusieurs agents. Un ensemble de conteneurs constituent une plateforme. Chaque plateforme doit contenir un conteneur spécial appelé main-container et tous les autres conteneurs s'enregistrent auprès de celui-là dés leur lancement

2.9 Domaine d'application des agents mobiles

La technologie des agents mobiles est une approche très puissante pour les applications réseau avec des moyens intelligents. Une partie des applications ne nécessiteront pas cette technologie, mais certaines se révèleront plus efficaces en l'utilisant. Voici quelques exemples :

- **La collecte de données à partir de sources multiples** : les agents mobiles sont capables de collecter des informations réparties sur plusieurs machines reliées à un réseau.

- **Recherche et filtrage** (recherche personnalisée): un agent mobile pourrait visiter plusieurs sites, cherche à travers les informations disponibles dans chaque site et construire un index des liens pour les portions d'informations qui répondent aux critères de sélection **[Maamar, 98].**
- **La surveillance [Bernichi, 09] :** si tel évènement se produit sur le réseau, alors l'agent prévient le client. Exemple : un agent peut aller à une place boursière, attendre qu'une action particulière atteigne certain prix puis en acheter à la place de l'utilisateur **[. Espinasse, 2010].**
- **Le calcul parallèle** : l'agent peut se dupliquer sur différents serveurs avec les différentes parties d'un calcul à effectuer.
- **Négociation** : l'agent part avec une offre du client et revient avec le meilleur prix obtenu.
- **Le commerce électronique :** l'agent pourrait localiser l'élément le plus approprié pour l'utilisateur.

3. Conclusion

Nous avons présenter dans ce chapitre les agents logiciels (Définitions, structure interne, propriétés et Taxonomie). Ensuite les agents mobiles, qui une catégorie particulière des agents logiciels dont la caractéristique prédominante est leur capacité de se migrer entre les nœuds d'un réseau ou de plusieurs réseaux.

Dans ce chapitre, nous nous sommes intéressés au domaine des agents mobiles. Nous avons commencé par définir le concept d'agent logiciel logiciels (Définitions, structure interne, propriétés et Taxonomie). Ensuite, nous avons définir les agent mobiles et les efforts de standardisation des plateformes d'agents mobiles, à savoir les normes MASIF et FIPA , les travaux concernant les agents mobiles et le domaine d'application.

Le prochain chapitre on va présenter un état de l'art sur la mobilité des agents mobiles, on concentrons sur la migration des processus sur des sites hétérogènes. Rappelons qu'un agent mobile est un processus où la migration est initiée par son code.

CHAPITRE- II

1. Introduction

Excepté les avantages potentiels suffisamment examinés, actuellement, les agents mobiles impliquent aussi des couts et des risques. Les plus évidents sont ceux abordés à la reprises d'exécution à distance : la migration de l'agent et l'exécution doivent être portatives à travers des plate-forme hétérogène. Ce problème est tellement embarrassant au point que certains travaux l'ont évité carrément et le laissent à la charge du développeur, ce qui peut amener à des exécutions incohérentes du fait que l'agent recommence l'exécution dès le départ, mais avec un état intermédiaire, pouvant ne pas satisfaire les états avec lequel l'agent doit être lancé.

Il existe deux degrés de mobilité des agents : faible et forte.

La mobilité faible intervient au cours de l'exécution d'un agent, elle consiste à transférer l'exécution d'une machine source vers une machine destination, en passant par :

- ✓ l'interruption de l'exécution sur le site source;
- ✓ puis le transfert du code et de l'état courant des données utilisées du site source vers le site destination;
- ✓ enfin la reprise de l'exécution sur le site destination;

Arrivée sur le site destination, l'agent mobile reprend son exécution depuis le début, tout en possédant les valeurs mises à jour de ses données.

En plus des informations prises en compte par la mobilité faible (code et état courant des données), la mobilité forte prend en compte l'état courant de l'exécution de l'agent. Ainsi, un agent fortement mobile, qui se déplace au cour de son exécution d'un site source à un site destination, commence son exécution sur le site destination au point ou il a été interrompu sur le site de départ. La mobilité forte se traduit alors par :

- ✓ L'interruption de l'exécution sur le site source;
- ✓ puis, le transfert du code, de l'état courant des données et de l'état courant de l'exécution de l'agent de site source vers le site destination;
- ✓ enfin la reprise de l'exécution sur le site destination;

Arrivée sur le site destination, l'agent mobile ne reprend pas son exécution depuis le début, mais il là poursuit au point même où il à été interrompu sur le site source.

2. Concept de migration de processus dans des systèmes hétérogènes

Rappelons qu'un agent mobile est un processus où la migration est initiée par son code. La migration de processus dans des systèmes hétérogènes peut être définie comme la capacité à déplacer un processus en cours d'exécution entre les différents processeurs qui sont reliées uniquement par un réseau (c'est-à-dire n'utilisant pas une mémoire localement partagée). Le système d'exploitation de la machine d'origine doit emballer tout l'état du processus de sorte que la machine de destination peut continuer son exécution. Le processus ne devrait normalement pas être concerné par les changements de son environnement autre qu'une variation de performance. En effet, lors de déplacement des pages de mémoire ou la capture et la restauration de l'état du processus, les liaisons de communication du processus doivent être maintenues.

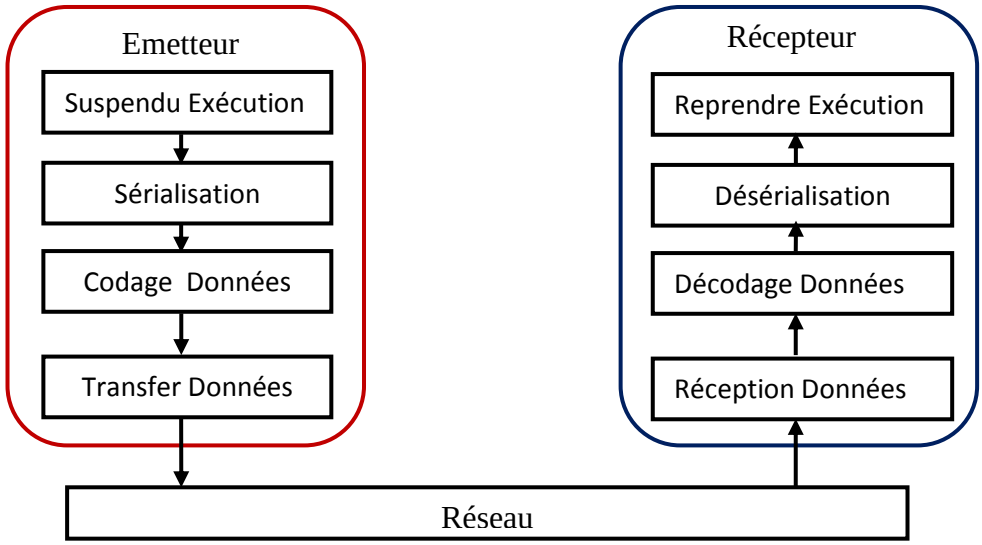


Figure II.1: Concept de migration de processus

Dans les systèmes homogènes, la migration de processus est basée sur le fait que les machines source et destinataire ont la même architecture. C'est-à-dire leurs unités centrales de traitement comprennent le même ensemble d'instruction et leurs systèmes d'exploitation ont des ensembles d'appels de système et de conventions de mémoire similaires. Ceci permet aux informations d'état d'être copiées mot à mot entre les hôtes pour que l'image de la mémoire soit identique.

La migration de processus dans les systèmes hétérogènes élimine cette supposition des machines sources et destinataires de même architecture. En plus des questions de migration sur les machines homogènes, il faut des mécanismes de traduction de l'état complet du processus de sorte qu'il peut être compris par la machine de destination. Cela nécessite la connaissance du type et de l'emplacement de toutes les valeurs de données (dans des variables globales, les trames de pile et le tas).

3.Travaux relatifs à la mobilité des agents mobiles

Différentes recherches ont été menées dans le domaine de la mobilité des agents et différents systèmes sont expérimentaux ont été réalisés. Des synthèses décrivant les principales techniques développée dans le domaine de la mobilité sont proposées par Milojicic **[Milojicic, 98].**

Dans ce travail, nous nous intéressons que de la mobilité forte, qui assure une continuité d'exécution des taches de l'agent au point ou il a été interrompu sur le site de départ.

3.1 Niveau de mise en œuvre

Les concepteurs de mécanismes de mobilités fortes suivent une des deux approches suivantes :

- Ils construisent un nouveau système, Cette approche a l'avantage de fournir une liberté de mise en œuvre.
- Ou il construisent leurs mécanismes sur la base d'un système existant et largement diffusé, soit au-dessus du système, soit par extension du système.

Le terme système désigner un micro-noyau, un système d'exploitation, un middleware ou un langage/environnement de programmation.

3.1.1 Nouveau système

Les nouveaux systèmes construits, qui proposent des mécanismes de mobilité, se déclinent sous la forme de nouveaux langage de programmation ou de nouveaux middleware et systèmes d'exploitation.

Facile est un langage conçu pour la programmation d'agent mobile**[Knabe, 95]** et Tycoon est un langage qui fournit les fonctionnalités nécessaires à la mobilité des applications mobiles **[Matthiske, 95].**

3.1.2 Système existant

Cette approche se base sur un système existant et largement diffusé, Elle est mise en œuvre :

- Soit au-dessus du système considéré, sans le modifier;
- Soit en étendant le système avec les fonctionnalités nécessaires à la mobilité;

Parmi les mécanismes de mobilité réalisés au-dessus d'un système existant, nous pouvant citer le package Condor qui propose des outils de sauvegarde/reprise de processus Unix, réaliser au-dessus du système d'exploitation Unix **[Litzkow, 92].**

Nous pouvons également citer les travaux effectués au-dessus de langage de programmation, tel que C/C++ ou Java. Ces travaux se basent sur un préprocesseur du programme de l'application, préprocesseur qui augmente le code de l'application de

traitement qui rendent l'application mobile. En se basant sur un tel préprocesseur, le langage de programmation reste inchangé. Le service *Arachne*de migration de threads dans les langage C/C++ a été mis en œuvre avec un préprocesseur des programme C/C++ **[Dimitrov, 98].** Cette technique est également utilisée pour la mis en œuvre de la plate-forme à agents mobiles proposée par le projet Wasp **[Fünfrocken, 98]**ou la plate-forme à agents mobiles JavaGo**[Sekiguchi, 99].**

3.2 Migration des agents par traduction

Lorsque la représentation de l'état d'exécution d'un agent ne dépend pas de la plate-forme sous-jacent, la mise en œuvre de la mobilité dans un système hétérogène est équivalente à sa mise en œuvre dans un système homogène. Mais si la représentation de l'état d'exécution de l'agent fortement dépendant de la plate-forme sous-jacent et varie d'une plate-forme à une autre, l'état d'exécution ne peut être directement transféré et utilisé par des plate-forme de nature variées. Pour faire face à ce problème il faut procéder à des traductions entre les différentes format de la plate-forme source vers le format de la plate-forme destination.

Dans ce cas il faut autant de traducteurs qu'il existe de couples de plate-forme différentes. De plus, lorsqu'une nouvelle plate-forme doit être prise en compte par le mécanisme de mobilité, il faut construire 2N nouveaux traducteurs, N étant le nombre de plate-formes déjà prise en compte par le mécanisme de mobilité. Cette approche a été suivie pour la mise en œuvre d'un mécanisme de migration de processus proposé par Shub**[Shub, 90].**

Une alternative à la traduction directe des formats de deux plate-forme est de se baser sur un format intermédiaire pour représenter l'état d'exécution d'un agent mobile. Ici, la représentation l'état d'exécution est traduit du format de la plate-forme source vers le format intermédiaire puis du format intermédiaire vers le format de la plate-forme destination. Cette solution nécessite alors deux traductions à chaque déplacement. Mais lors de la prise en

compte d'un nouveau type de plate-forme, elle présente l'avantage de n'exiger la construction que de deux traducteurs : un traducteur du format de la nouvelle plate-forme vers le format intermédiaire et un traducteur inverse. Les systèmes distribués Emerald **[Steensgaard, 95]** et Stardus **[Cabillic, 93]** et les systèmes de migration Tui **[Smith, 96]** fournissent un mécanisme de migration de processus qui suit une telle approche.

3.3 La mobilité des processus dans la machine virtuelle Java

La machine virtuelle Java offre des services de sérialisation et dé-sérialisation qui permettent respectivement de capturer et de restaurer l'état d'un **objet** Java. Ces services peuvent être utilisés pour déplacer des objets entre différentes machines. De plus, la machine virtuelle Java offre un service de chargement dynamique des classes qui permet de déplacer du code Java vers d'autres nœuds.

En revanche, Java ne fournit pas de service permettant de capturer et de restaurer l'état d'un flot de contrôle (c'est-à-dire un processus) car sa pile n'est pas accessible. Un tel service devrait permettre d'une part d'extraire l'état courant de l'exécution du flot et,

d'autre part, de restaurer cet état pour que l'exécution reprenne au point où elle en était au moment de la capture. Autrement dit, ce service doit permettre la mobilité forte de l'application. Pour fournir un tel service, les auteurs proposent **[Bouchenak, 04] :**

La conception d'une machine virtuelle Java étendue qui prend en charge la migration de processus Java avec les propriétés suivantes:

a. La syntaxe du langage Java n'est pas modifiée;

b. Le compilateur Java n'est pas modifié;

c. Les APIs Java existantes ne sont pas affectées;

d. La nouvelle API Java est proposée pour un mécanisme de sérialisation de processus générique;

Les travaux dans **[Bouchenak, 04]** se concentrent sur la conception et la mise en œuvre d'un mécanisme de sérialisation de processus Java au-dessus duquel la mobilité et la persistance d'état d'exécution sont construites. Il est pertinent de préciser comme pour la sérialisation d'objets Java, la sérialisation de processus permet à un état d'un processus d'être transmis à un ordinateur distant afin de le restituer. Cependant, contrairement à la sérialisation d'objet, la sérialisation de processus ne traite pas la distribution, le partage d'objet entre les processus, la synchronisation et la gestion des objets d'entrées/sorties (les sockets ou les fichiers).

La principale difficulté lors de la capture de l'état d'exécution d'un processus est son inaccessibilité aux programmes Java. Afin de remédier à ce problème, deux approches principales peuvent être suivies **[Bouchenak, 04]**, à savoir:

- L'approche du niveau noyau (machine virtuelle Java);
- L'approche du niveau application.

L'approche la plus intuitive pour accéder à l'état d'un processus Java consiste à ajouter de nouvelles fonctions à l'environnement Java pour exporter l'état des *processus* de la JVM. Les systèmes tels que Sumatra **[Acharya, 96]**, Merpati **[Suezawa, 00]**, ITS **[Bouchenak, 04]** et CIA **[Illmann, 01]** utilisent l'approche de l'extension de la machine virtuelle Java. Cette approche permet l'accès complet à tout l'état d'un processus Java mais son principal inconvénient réside dans sa dépendance à une extension particulière de la JVM. En effet, le mécanisme de sérialisation de processus ne peut donc pas être utilisé sur des machines virtuelles existantes.

Pour remédier au problème de la non-portabilité du mécanisme de sérialisation de processus dans des différents environnements de la machines Java, certains projets proposent une approche au niveau de l'application sans avoir recours à une extension de la JVM. Dans cette approche, le code de l'application est transformé par un préprocesseur

avant l'exécution afin d'attacher un objet de sauvegarde du programme exécuté par le processus Java et d'ajouter de nouveaux états dans ce programme. Les codes ajoutés gèrent la capture de l'état du processus et les opérations de restauration. Ils stockent les informations d'état dans un objet de sauvegarde qui peut ainsi être sérialisé. Plusieurs systèmes de migration de processus Java suivent cette approche: Wasp **[Funfrocken, 98]** et JavaGo**[Sekiguchi, 99]** fournissent un préprocesseur de code source Java alors que Brakes**[Truyen, 00]** et JavaGoX**[Sakamoto, 00]** s'appuient sur un préprocesseur de bytecode.

Le principal avantage de la mise en œuvre du niveau application est la portabilité des mécanismes à tous les environnements Java. Cependant, ces mécanismes ne sont pas en mesure d'accéder à l'état d'exécution complète d'un processus Java. Dans le cas du système Wasp, par exemple, l'approche proposée ne permet pas de capturer les valeurs des résultats intermédiaires.

Les principales difficultés lors de l'extension de la JVM afin de réaliser la sérialisation d'un processus sont les suivantes:

a) L'accès à l'état d'exécution des processus. Cela est nécessaire pour construire un mécanisme de sérialisation de processus Java.

b) La mise en œuvre d'un état des processus Java portable. Cela est nécessaire pour suivre l'esprit de la portabilité Java (portabilité du code, portabilité des données et dans ce cas la portabilité du fil d'exécution).

c) La mise en œuvre d'un mécanisme de sérialisation de processus compatible avec le compilateur Java à la volée (le compilateur JIT). Cela est nécessaire afin de fournir une approche efficace pour les environnements Java.

d) La proposition d'un service générique de sérialisation. Pour les fils d'exécution, cette approche a été suivie afin de respecter la généricité et la réutilisabilité nécessaires.

Les approches proposées pour surmonter les difficultés lors de l'extension de la JVM afin de réaliser la sérialisation d'un processus sont les suivantes **[Bouchenak, 04]:**

a) L'extension de la JVM afin d'externaliser l'état du fil et la mise à disposition d'une nouvelle API qui permet au programmeur d'accéder à cet état.

b) La mise en œuvre d'un mécanisme d'inférence de types qui transforme une structure de données non-portable d'un état de fil à une structure de données portable.

c) L'utilisation d'une technique de dés-optimisation dynamique de code compilé à la volée (la compilation JIT). L'utilisation dés-optimisation dynamique est la seule technique qui permet de revenir aux méthodes compilées à la volée afin de fournir un mécanisme de sérialisation de processus Java qui est compatible avec la compilation à la volée. En effet, cette technique permet de revenir, au cours de l'exécution d'un code compilé, à la version interprétée de ce code. Elle est utilisée par exemple lorsqu'il faut «défaire» la compilation afin de faire une expansion des méthodes pour prendre en compte un nouveau code Java chargé dynamiquement.

d) Une conception générique qui suit l'approche orientée objet et la hiérarchie des classes afin de proposer un mécanisme de fil générique, adaptable et une sérialisation réutilisable.

4. Conclusions

Dans ce chapitre, nous nous sommes intéressés particulièrement à la mobilité des agents. Nous avons commencé par définir les différents éléments qui permettent la migration de processus. Les mises en œuvre de migration au niveau application ne permettent pas d'accéder à l'état de noyau d'où leur incapacité à faire migrer tous les processus.

La migration de processus au niveau noyau implique des extensions du système d'exploitation sous-jacent. L'avantage de la migration au niveau noyau est qu'elle offre une mobilité de processus complète (donc forte) plus efficacement mais elle mène à une complexité supplémentaire. Un autre avantage de mobilité de processus au niveau noyau est la transparence à l'application.

Nous avons également exposé la migration de processus sur la machine virtuelle Java. La migration complète de processus est mise en œuvre via une extension de la machine virtuelle Java. Nous avons décrit comment la sérialisation d'objets (fournie déjà par machine virtuelle Java) peut être transformée en une sérialisation de processus portable. Pour garantir les performances et la portabilité dans la machine virtuelle Java, une technique de dés-optimisation, utilisée dans le domaine de débogage et de chargement de classes dynamiques, est employée.

CHAPITRE- III

1. Introduction

Les systèmes dynamiques ne peuvent pas être décrits en ne prenant en compte que leurs états initiaux et finaux. En effet, on doit tenir compte de leur comportement permanent qui est une séquence d'états, pour qu'on puisse parler d'une description bien fondée

Carl Adam Petri a introduit le formalisme des réseaux de Petri **[Tran, 05],** dans sa thèse "Communication avec des Automates" à l'université Darmstadt en Allemagne, en 1962. Un peu plus tard et au début des années 70s, ce travail a été développé par Anatol W. Holt, F. Commoner, M. Hack et leurs collègues dans le groupe de recherche de Massachussetts Institute Of Technology (MIT). En 1975 MIT organise la première conférence sur les réseaux de Petri et les méthodes relationnels. Par la suite, J. Peterson a publié en 1981, le premier ouvrage sur les réseaux de Petri .

Le formalisme des réseaux de Petri (RdP) a été proposé comme un outil mathématique permettant la modélisation des systèmes dynamiques à événements discrets. Les réseaux de Petri offrent un outil formel avec une bonne représentation graphique permettant de modéliser et d'analyser les systèmes discrets, notamment les systèmes concurrents et parallèles.

Grace à leur facette graphique, les réseaux de Petri, aident à comprendre facilement le système modélisé. En outre, les activités dynamiques et concurrentes des systèmes sont aisément simulées, par les RDP, vue leur puissance d'expression mathématique. L'intérêt primordial de ces réseaux réside dans leur possibilité d'analyser les systèmes modélisés.

2. Concepts de base & définition

2.1 Définitions informelles

Intuitivement, un réseau de Petri est un graphe orienté biparti (ayant deux types de nœuds) : des places représentées par des cercles et des transitions représentées par des rectangles. Les arcs du graphe ne peuvent relier que des places vers des transitions, ou des

transitions vers des places (pas d'arcs reliant les places ni d'arcs entre transitions). Un exemple de réseau de Petri est illustré par la figure III.1.

Un réseau de Petri décrit un système dynamique à événements discrets. Les états possibles du système (qui sont discrets), sont décrits par les places. Les transitions quant à elles, permettent la description des événements ou des actions qui causent le passage du système d'un état vers un autre.

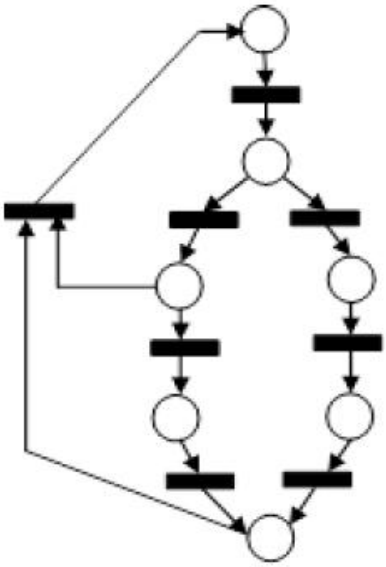


Figure III.1 : Un RdP comportant 7 places, 8 transitions et 17 arcs orientés

- **Condition** L'état du système peut être vu comme un ensemble de conditions qui sont de ce fait, des prédicats ou des descriptions logiques d'un état du système. Une condition est soit vraie soit fausse.
- **Evénement** Les actions se déroulant dans le système définissent Les événements. L'état du système conditionne le déclenchement d'un événement.
- **Déclenchement, pré-condition, post-condition** Les pré-conditions de l'événement sont les conditions nécessaires au déclenchement de celui-ci. Quand un événement se produit, certaines de ses pré-conditions peuvent cesser d'être vraies alors que d'autres conditions (post-conditions de l'événement), deviennent vraies.
- **Le Marquage d'un réseau de Petri** Un réseau de Petri permet de décrire la dynamique du système représenté, étant donné que c'est un graphe muni d'une

sémantique opérationnelle, c'est-à-dire qu'un comportement est associé au graphe. Pour cela, on associe un troisième élément: les jetons, aux places. À un instant donné, une répartition des jetons dans les places est appelée marquage du réseau de Petri. Grace au marquage d'un réseau de Petri, l'état du système modélisé par ce réseau est défini. Le marquage consiste à placer initialement un nombre m_i entier (positif ou nul) de jetons dans chaque place Pi du réseau. Le marquage du réseau sera défini par un vecteur M = {mi}.

Pour un marquage donné, une transition peut être sensibilisée ou non. Si chacune des places en entrée d'une transition contient au moins un jeton, elle est sensibilisée. Pour un marquage donné, L'ensemble des transitions sensibilisées défini l'ensemble des changements d'états possibles du système depuis l'état correspondant à ce marquage. C'est un moyen de définir l'ensemble des événements auxquels ce système est réceptif dans cet état.

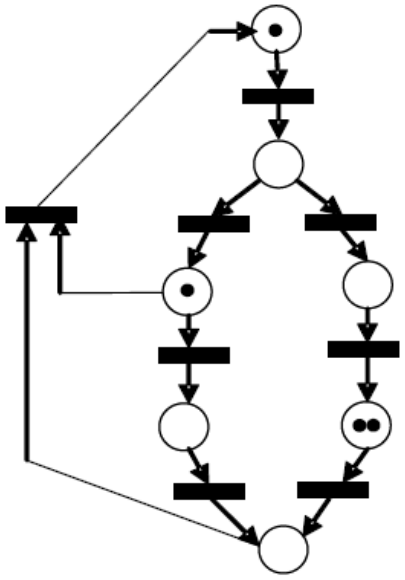


Figure III.2 : Un RdP marqué avec un vecteur de marquage M : M = (1,0,1,0,0,2,0).

2.2 Définitions formelles

Formellement, un réseau de Petri (R) est un triplé R= (P, T, W) où P est l'ensemble des places (les places représentent les conditions) et T l'ensemble des transitions (les transitions

représentent les événements ou les actions) tel que $P \cap T = \varphi$ et W est la fonction définissant le poids porté par les arcs où $W : ((P \times T) \cup (T \times P)) \rightarrow N = \{0, 1, 2, \ldots\}$.

Le réseau R est fini si l'ensemble des places et des transitions est fini, c-à-d $|P \cup T| \in N$.

Un réseau R = (P, T, W) est ordinaire si pour tout $(x, y) \in ((P \times T) \cup (T \times P))$: $W(x, y) \leq 1$.

Pour chaque $x \in P \cup T$:

*x représente l'ensemble des entrées de x : $*x = \{y \in P \cup T / W(y, x) \neq 0\}$

x* représente l'ensemble des sorties de x : $x^* = \{y \in P \cup T / W(x, y) \neq 0\}$

Remarque Si $*x = \varphi$, x est dite source, si $x^* = \varphi$, x est dite puits.

Pour exprimer l'état d'un réseau de Petri, les places peuvent contenir des jetons qui ne sont que de simples marques (points noirs). La structure et l'état d'un réseau de Petri, déterminent son comportement.

- **Notion d'état dans un réseau de Petri** Dans la théorie des réseaux de Petri, l'état d'un réseau est souvent appelé marquage du réseau. Il est défini par l'association des jetons aux places. Le marquage d'un réseau de Petri R= (P, T, W) est défini par la fonction de marquage $M : P \rightarrow N$.

Un réseau de Petri marqué est donné par $\Sigma = (P, T, W, M0)$ où M0 est le marquage initial. La règle de franchissement définit le comportement d'un réseau de Petri marqué.

- **Notion de changement d'état et de règle de franchissement** Le mouvement de jetons des places d'entrée vers les places de sortie d'une transition, traduit le changement d'état du système. Ce mouvement est causé par le franchissement d'une transition. Le franchissement représente une occurrence d'un événement ou d'une action. La présence de jetons dans les places d'entrée de la transition, conditionne son franchissement.

Quand toutes les places en entrée d'une transition sont suffisamment marquées, la transition peut être tirée, suite à ce tir ou franchissement, les jetons sont retirés des places en entrée (ancien état) et ajoutés aux places en sortie (Nouvel état) de la transition franchie.

La relation de transition qui définit le changement d'état dans un réseau marqué lors de l'exécution d'une action, est appelée **règle de franchissement**. La formalisation de la possibilité d'exécution d'une action définit la règle de franchissement: on dit qu'une transition $t \in T$ peut être franchie à partir d'un marquage M (qui représente l'état du system à un instant donné) si et seulement si chaque place d'entrée $p \in {}^*t$ de la transition t contient au moins un nombre de jetons qui est supérieur ou égale au poids de l'arc reliant cette place d'entrée p à la transition t, tel que: $M(p) \geq W(p, t) \; \forall \; p \in P$.

Une règle de franchissement est définie par $M'(p) = M(p) - W(p, t) + W(t, p)$ pour tout $p \in P$, ceci traduit le fait que, lorsque la transition t est franchie à partir d'un marquage M, il faut retirer W(p, t) jetons à partir de chaque place en entrée à la transition t et ajouter W(t, p) jetons dans chaque place en sortie de la transition t ce qui permet de produire un nouveau marquage M'. Le franchissement d'une transition t, dénoté par $M\ [t > M'$, est appelé occurrence de t.

- **Exécution d'un réseau de Petri : Notion de marquage** L'exécution d'un réseau de Petri est définie par un ensemble de séquences d'occurrence. Une séquence d'occurrence est une séquence de transitions franchissables dénotée par $\sigma = M0\ t1\ M1\ t2 \ldots$ tel que $Mi\ -1\ [ti > Mi$. Une séquence t1 t2… est une séquence de transitions (commencée par le marquage M) si et seulement si il existe une séquence d'occurrence M0 t1 M1... avec M=M0. Si la séquence finie t1 t2 …tn conduit à un nouveau marquage M', à partir du marquage M, on écrit $M\ [t1\ t2 \ldots tn > M'$ ou simplement $M[t1t2 \ldots tn>$ si on ne veut pas spécifier le marquage résultat.

L'ensemble de marquages accessibles d'un réseau marqué (P, T, W, M0) est défini par [M0 > = {M ∃ t1 t2…tn: M0 [t1t2…tn > M}.

Exemple:

Dans le cas des réseaux dits à arcs simples ou de poids égal à 1 (cf. la figure 3.3), Le franchissement d'une transition consiste à retirer un jeton dans chacune des places d'entrée de la transition et à en ajouter un dans chacune de sorties de places de celle-ci.

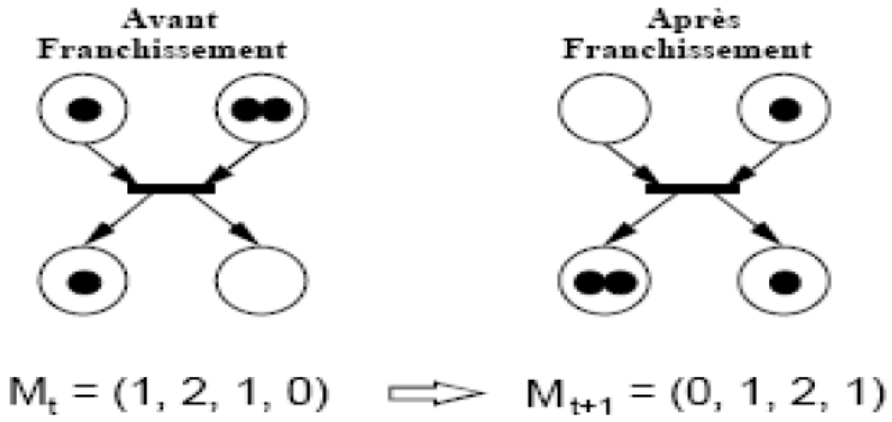


Figure III.3 : Evolution d'états d'un réseau de Petri

L'évolution des états d'un réseau de Petri marqués simple obéit aux règles suivantes :

- Quand chacune des places en entrées d'une transition, possède au moins le nombre de jetons correspondant au poids de l'arc la reliant à cette transition, on dit que la transition est **franchissable** (**sensibilisée** ou **tirable)**.
- lorsque plusieurs transitons sont validées, le réseau ne peut évoluer que par franchissement d'une seule transition sélectionnée à la fois.
- Le franchissement d'une transition est indivisible et de durée nulle.

On voit donc, que dans l'évolution des réseaux de Petri, ceux-ci peuvent passer par différents états dont l'apparition est conditionnée par le choix des transitions tirées. D'où l'indéterminisme introduit par ces règles. Ceci cadre bien avec les situations réelles où il n'y a pas de priorité dans la succession des événements **[Claud, 01].**

2.3 Représentation matricielle

Les tâches d'analyse et de vérification effectuées sur un modèle RdP peuvent être simplifiées, grâce à la représentation matricielle d'un RdP. En effet, manipule une représentation graphique d'un modèle RdP est une tâche délicate comparée à une représentation matricielle. Il est également possible, de représenter la fonction *W* (fonction de poids) par des matrices.

Définition

Soit Un réseau de Petri *R*= (*P, T, W*) avec *P*= {p1, p2, …, pm} et *T*= {t1, t2, …, tn}.

On appelle matrice des pré-conditions ***pré***, la matrice m × n à coefficients dans N telle que *pré* (i,j)= *W*(pi, tj) indique le nombre de marques que doit contenir la place pi pour que la transition tj devienne franchissable. De la même manière on définit la matrice des post-conditions ***post***, la matrice m × n telle que *post* (i,j)= *W*(pi, tj) contient le nombre de marques déposées dans pi lors du franchissement de la transition tj. La matrice C= *post* – *pré* est appelée matrice d'incidence du réseau (m représente le nombre de places d'un réseau de Petri et n le nombre de transitions).

Le marquage d'un réseau de Petri est représenté par un vecteur de dimension M à coefficients dans N. La règle de franchissement d'un réseau de Petri est définie par :

M' (p) =*M* (p) + C (p, t).

Exemple

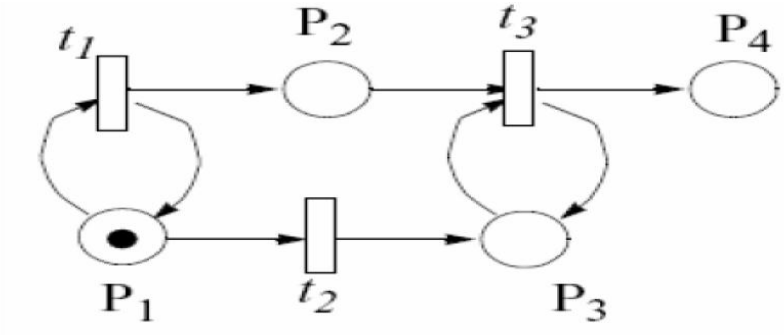


Figure III.4 : Un RdP marqué avec un vecteur de marquage M : M = (1,0,0,0).

Pour le réseau ci-dessus (cf. la figure 3.4) , *P*= {p1, p2, p3, p4} *T*= {t1, t2, t3}, la représentation matricielle est donnée ci-dessous (cf. la figure 3.5).

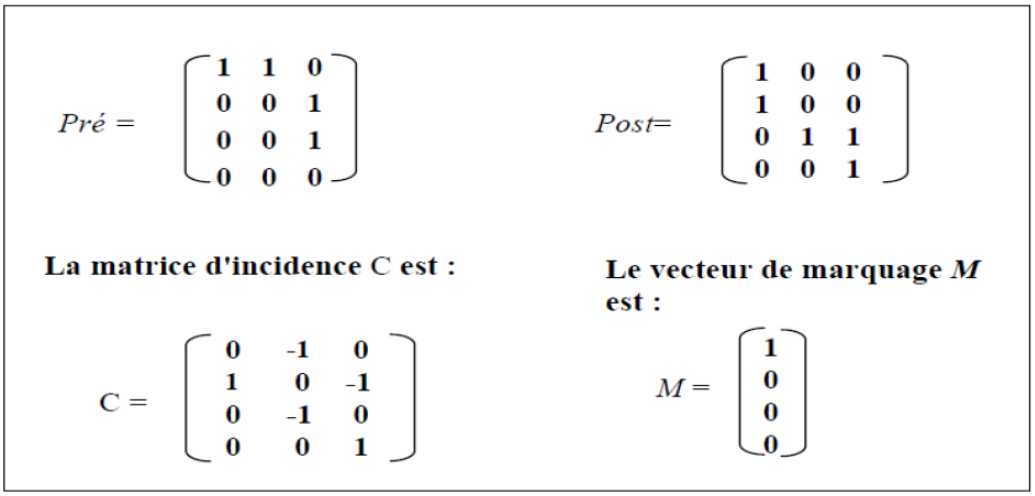


Figure III.5 : Matrice d'incidence et vecteur de marquage du RdP de la figure 8

- **Sémantique du parallélisme et problème de conflits**

Quand plusieurs transitions peuvent être franchissables à un moment donné dans un réseau de Petri, on parle de parallélisme de tir des transitions. Cet aspect pose un problème de choix pour l'état futur du réseau qui dépend de la transition sélectionnée parmi toutes celles offertes. En général, ce conflit est résolu par le choix d'une sémantique dite du parallélisme qui définit une stratégie de tir, exemple : tirer une seule transition à la fois.

3. Les réseaux de Petri colorés (RdPCs)

3.1 Définition informelle

Les RdP colorés sont une voie possible pour obtenir des modèles de taille largement inférieure aux modèles équivalents que l'on obtient avec les RdP ordinaires. Les réseaux de Petri colorés sont des réseaux de Petri dans lesquels les jetons portent des couleurs. Une couleur est une information attachée à un jeton. Cette information permet de distinguer des jetons entre eux et peut être de type quelconque **[Jensen 98].** Ainsi, les arcs ne sont pas seulement étiquetés par le nombre de jetons mais par les couleurs de ces jetons.

Le franchissement d'une transition est alors conditionné par la présence dans les places en entrée du nombre de jetons nécessaires qui, en plus, satisfont les couleurs qui étiquettent les arcs. Après le franchissement d'une transition, les jetons qui étiquettent les arcs d'entrée sont retirés des places en entrée tandis que ceux qui étiquettent les arcs de sortie sont ajoutés aux places en sortie de cette transition.

3.2 Définition formelle

Avant de donner la définition de RdPC, il est nécessaire d'introduire quelques notations :

- Un multi-ensemble m, dans un ensemble non vide S, est une fonction $m \in [S \rightarrow N]$ qu'on représente comme une somme formelle : $\sum s \in S \ m(s).s$
- Les entiers non négatifs $\{m(s) / s \in S\}$ sont les coefficients du multi-ensemble, $s \in m$ si, et seulement si $m(s) \neq 0$.
- Le type d'une variable v est noté par Type (v).
- Le type d'une expression expr est noté par Type (expr).
- L'ensemble de variables, dans une expression, expr est noté par Var (expr).
- Une expression sans variables est dite expression fermée.

On utilise **B** pour noter le type Booléen (Contient les éléments {Faux, Vrai} et peut subir les opérations standards). De plus, la notation Type(v) est étendue en $Type(A)=\{Type(v)/v \in A\}$ où A est un ensemble de variables.

On note aussi par :

- N, l'ensemble de tous les entiers non négatifs,
- S_{MS}, l'ensemble de tous les multi-ensembles dans S,
- ∅, le multi-ensemble vide,

Définition 1 [Jensen 98]

Un réseau de Petri coloré est un tuple CPN = (Σ, P, T, A, N, C, G,E, I) ou :

1. Σ est un ensemble fini non vide de types, appelés aussi ensembles de couleurs,

2. P est un ensemble fini de Places,

3. T est un ensemble fini de Transitions,

4. A est un ensemble fini des Arcs tel que : $P \cap T = P \cap A = T \cap A = \emptyset$.

5. N est une fonction de nœud. Elle est définie de A dans $P \times T \cup T \times P$.

6. C est une fonction de couleur. Elle est définie de P dans Σ.

7. G est une fonction de garde. Elle est définie de T dans l'ensemble des expressions tel que :

$\forall t \in T : [Type (G (t)) = Booléen \wedge \Sigma type (Var (G(t))) \subseteq \Sigma]$.

8. E est une fonction de l'expression d'un arc. Elle est définie de A dans l'ensemble des expressions tel que :

$\forall a \in A : [Type (E (a)) = C(p)_{Ms} \wedge \Sigma type (Var (E (a))) \subseteq \Sigma]$ Où p est la place de N(a).

9. I est une fonction d'initialisation. Elle est définie de P dans l'ensemble des expressions fermées tel que : $\forall p \in P : [Type (I(p))=C(p)_{Ms}]$

Remarque

1. L'ensemble de types détermine les valeurs des données, les opérations et les fonctions qui peuvent être utilisés dans les expressions du RdPC.

On suppose que chaque type a, au moins, un élément.

2+3+4 Les places, les transitions et les arcs sont décrits par trois ensembles P, T, et A qui doivent être finies et disjointe deux à deux.

5. Une fonction du nœud, affecte à chaque arc, une paire de nœud où le premier est le nœud source et le second est le nœud destination. Les deux doivent être différents (c.-à-d., l'un doit être une place pendant que l'autre doit être une transition).

6. La fonction de couleurs C, affecte à chaque place, un type C(p). Intuitivement, cela veut dire que chaque jeton dans p doit avoir une valeur de donnés qui appartient à C(p).

7. La fonction de garde G, affecte à chaque transition "t", une expression booléenne; où toutes leurs variables ont des types qui appartiennent à Σ.

8. La fonction de l'expression de l'arc E, affecte à chaque arc "a", une expression de type $C(p)_{MS}$. Cela veut dire que chaque expression d'un arc doit être évaluer vers des multi-ensembles sur le type de la place adjacente p.

9. La fonction d'initialisation I, affecte à chaque place "p", une expression fermée qui doit être de type $C(p)_{MS}$.

3.3 Notations

Pour tout $t \in T$ et pour toutes les paires des nœuds $(x_1, x_2) \in (P \times T) \cup (T \times P)$, on a les notations suivantes :

- $A(t) = \{a \in A \mid N(a) \in P \times \{t\} \cup \{t\} \times P \}$.
- $Var(t) = \{v \mid v \in Var(G(t)) \vee \exists\, a \in A(t): v \in Var(E(a))\}$.
- $A(x_1,x_2) = \{ a \in A \mid N(a) = (x_1,x_2)\}$
- $E(x_1,x_2) = \sum_{a \in A(x_1,x_2)} E(a)$.

3.4 Affectation

Une affectation d'une transition t est une fonction b défini sur Var(t), tel que:

1. $\forall v \in Var(t): b(v) \in Type(v)$.

2. G(t)<b>.

On note par B(t) l'ensemble de toutes les affectations de t.

G(t)<b> dénote l'évaluation de l'expression du garde G(t) dans l'affectation b.

On écrit, souvent, les affectation sous la forme $<v_1=c_1,v_2=c_2,...,v_n=c_n>$ où :

$Var(t) =\{ v_1 , v_2 ,\ldots\ldots, v_n\}$.

3.5 Jeton, affectation et marquage

Un jeton est une paire (p, c) où $p \in P$ et $c \in C(p)$, alors qu'une affectation est une paire (t, b) où $t \in T$ et $b \in B(t)$.

L'ensemble des jetons est noté par TE (Token Element), alors que l'ensemble de toutes les affectations est noté par BE (Binding Element).

Un marquage est un multi-ensemble sur TE, alors qu'un pas est un multi-ensemble non vide et fini sur BE. Le marquage initial M_0 est le marquage obtenu en évaluant les expressions d'initialisation :

$\forall(p, c) \in TE: M_0(p, c) = (I(p))(c)$.

3.6 Principe de fonctionnement d'un RdPC

Un pas Y est valide dans un marquage M si, et seulement, si :

$\forall\ p \in P: \sum_{(t,b) \in Y} E(p,t)<b>\ M(p)$.

On dit alors que (t, b) est sensibilisé ou t est sensibilisée (franchissable). Les éléments de Y sont sensibilisés concurremment (quand $|Y| \geq 1$).

Quand un pas Y est sensibilisé dans un marquage M_1, il peut être franchi, en changeant le marquage M_1 à un autre marquage M_2, défini par :

$\forall p \in P: M_2(p) = (M_1(p) - \sum_{(t,b) \in Y} E(p,t)<b>) + \sum_{(t,b) \in Y} E(t,p)<b>$.

M_2 est directement accessible à partir de M_1. Cela est écrit : M_1 [Y>M_2.

3.7 Séquence de franchissement

Une séquence finie de franchissement est une séquence de marquages et de pas :

M_1 [Y_1< M_2 [Y_2< M_3 ... M_n [Y_n< M_{n+1} tel que n ∈N, et M_i [Y_i> M_{i+1} pour tout i∈{1,2,...,n}

M_1 est le marquage initial, M_{n+1} est le marquage final et n est la langueur.

D'une manière analogue, une séquence infinie de franchissement est une séquence de marquages et des pas :

M_1 [Y_1> M_2 [Y_2> M_3 ... tel que M_i[Y_i > M_{i+1} pour tout i ≥1.

Un marquage M'' est accessible à partir d'un marquage M' si, et seulement si, il existe une séquence finie de franchissement commençant de M' et se termine par M''.

L'ensemble des marquages qui sont accessible à partir M' est noté par [M'>.

Un marquage est dit accessible si, et seulement si, il appartient à [M_0>.

3.8 Exemple de RdPC

Pour mieux comprendre les concepts de base des RdPC, nous donnons ici une partie de l'exemple qui modélise un protocole simple de transmission proposé par **[Jensen, 96].**

Les états du RdPC sont représentés par les places et chaque place à un type associé qui détermine le type de données que la place peut contenir. Comme montré dans la Figure 3.6, les places **Envoyer** (Send) et **A** ont le type : Entier x Donnés (INT x DATA). Ce type est le produit cartésien des nombres entiers et donnés. Les éléments du type représentent les paquets qui seront transmis sur le Réseau. Chaque paquet est une paire où le premier élément est le nombre de paquets (de type INT), pendant que le second élément est les données contenues dans le paquet, c.-à-d., un texte de string (de type DATA). La place **Prochain Envoie** (Next Send) a le type INT.

Pendant l'exécution d'un RdPC, chaque place contiendra un nombre variable de jetons. Chacun de ces jetons porte une valeur de données qui appartient au type associé à la place.

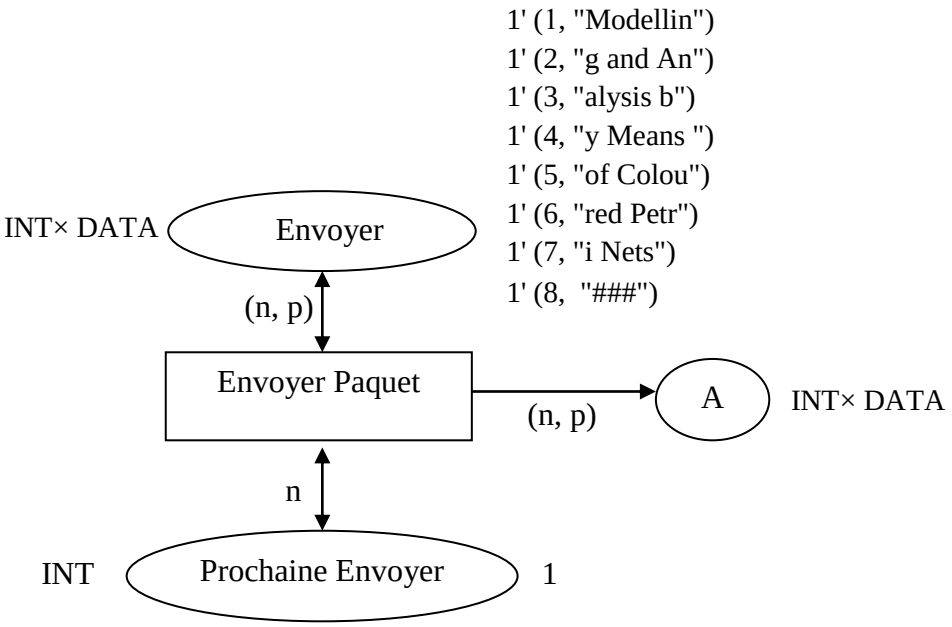


Figure III.6 : Exemple de RdPC (RdPC décrivant un protocole simple)

Dans l'exemple, la place Envoyer débute avec les huit valeurs de jetons chacun représente un paquet qui est transmis dans tout le réseau:

Dans la Figure 3.6, il y a un 1' devant chaque valeur de jeton. Cela nous dit qu'il y a exactement un jeton qui porte la valeur. En générale, plusieurs jetons peuvent avoir la même valeur de jeton, et alors on a un multi-ensemble de valeurs de jetons, tel que:

1'(2,"g and An") + 2'(3,"alysis b") + 1'(5,"of Colou")

Dans chacune on a un jeton avec la valeur (2, "g et An"), deux jetons avec la valeur (3, "alysis b"), et un jeton avec la valeur (5, "of Colou"). Un multi-ensemble est semblable à un ensemble, excepter qu'il peut y avoir plusieurs apparences du même élément. Si on ajoute l'élément (3, "alysis b") à l'ensemble:

{ (2,"g and An"), (3,"alysis b"), (5,"of Colou") }

Rien ne se passe, parce que l'élément appartient déjà à l'ensemble. Cependant, si on ajoute l'élément (3, "alysis b") au multi-ensemble:

1'(2,"g and An") + 1'(3,"alysis b") + 1'(5,"of Colon")

On obtient un multi-ensemble avec quatre éléments au lieu de trois:

1' (2,"g and An") + 2'(3,"alysis b") + 1'(5,"of Colou")

Les nombres entiers devant l'opérateur (') sont appelés coefficients. Dans cet exemple (2, "g et An") et (5, "of Colou") ont un (1) comme coefficient, alors que (3, "alysis b") a deux (2) comme coefficient. Toutes les autres valeurs du type ont zéro comme coefficient (ils sont alors omis).

Un état d'un RdPC est appelé un marquage. Il consiste en plusieurs jetons placés sur les places individuelles. Chaque jeton porte une valeur qui appartient au type de la place sur lesquelles les jetons résident. Les jetons qui sont présentés dans une place particulière appelée marquage de cette place.

Les actions d'un RdPC sont représentées au moyen de transitions. Dans l'exemple , il y a une seule transition. Un arc entrant indique que la transition peut enlever des jetons de la place correspondante alors qu'un arc sortant indique que la transition peut ajouter des jetons. Le nombre exact de jetons et leurs valeurs de données est déterminé par les expressions de l'arc (lesquels sont placés à côté des arcs).

La Transition Envoyer Paquet a trois alentours arcs avec deux expressions d'arc différentes: (n, p) et n. Deux arcs sont des arcs doubles. Chacun d'eux est une sténographie pour deux arcs de sens contraire avec une expression d'arc identique. D'ici il y a cinq arcs vraiment différents (deux arcs entrant et trois sortant). Les expressions de l'arc contiennent deux variables libres: n de type INT et p de type DATA.

On suppose l'affectation de variable n (de transition Envoyer Paquet) à la valeur 1 et la variable p à la valeur "Modellin". Cela donne l'affectation:

<n = 1, p = "Modellin">

Cela signifié qu'un événement de transition Envoyer Paquet (avec l'affectation précisée) enlèvera un jeton avec la valeur (1, "Modellin") de la place Envoyer et un jeton avec la valeur 1 de la place Prochaine Envoie. Les deux jetons sont disponibles, c.-à-d, présentés aux deux places, et d'ici la transition Envoyer Paquet est franchissable avec l'affectation donnée. Quand la transition est franchie, les deux jetons spécifiés seront enlevés des places d'entrée Envoyer Prochaine et Envoie simultanément, trois jetons seront ajoutés aux places de sortie : Envoyer et A obtiendront un jeton avec la valeur (1, "Modellin"), alors que Prochaine Envoie obtiendra un jeton avec la valeur 1.

3.3 Conclusion

Nous avons présenté, dans ce chapitre, les concepts de base des réseaux de Petri et les réseaux de petri colorés, ainsi leur utilité. Un réseau de Petri (RdP) permet, d'une manière générale, de modéliser le flot de contrôle, incluant la synchronisation des activités, par le marquage. Un cycle d'exécution d'un RdP fait passer son état un marquage à un autre. La poursuite de l'exécution se fait en se référant uniquement au dernier marquage. Nous avons exploité ce fait pour réaliser la continuité d'exécution de l'agent d'une machine à une autre.

CHAPITRE- IV

1. Introduction

Excepté les avantages potentiels suffisamment examinés, actuellement, les agents mobiles impliquent aussi des couts et des risques. Les plus évidents sont ceux abordés à la reprises d'exécution à distance : la migration de l'agent et l'exécution doivent être portatives à travers des plate-forme hétérogènes. Ce problème est tellement embarrassant au point que certains travaux l'ont évité carrément et le laissent à la charge du développeur, ce qui peut amener à des exécutions incohérentes du fait que l'agent recommence l'exécution dès le départ, mais avec un état intermédiaire, pouvant ne pas satisfaire les états d'exécutions avec lequel l'agent doit être lancé.

En plus, des différences technologiques qui peuvent exister entre les différentes machines d'un réseau, surtout sur Internet, les solutions sont confrontées au problème de différences qu'il y a entre les différents langages de programmation. Ceci a amené certains chercheurs à proposer des solutions particulières pour des langages particulières **[Bouchenak, 01]** pour arriver à des solutions qui sont mono-langages tel que Telescript **[Jonathan, 97]** ou multi-langages tel que D'Agent **[Robert, 01],** Ara **[Peine, 02].** C'est pourquoi à proposer une solution qui assure la continuité de l'exécution des tâches des agents mobiles sur plusieurs sites hétérogènes.

2. Solution proposé

Dans notre travail, nous avons opté pour la combinaison entre deux approches **MASA-méthode** (*Modélisation multi-Agents de Système à base de composants Autonomes et hétérogènes*) **[Lahlouhi, 2006]** et **CPN** (*Colorde Petri Net*) **[Jensen, 1998].**

2.1 MASA-Méthode

Nous utilisons MASA-Méthode qui est une méthodologie multi-agent complète et mixte et qui prend en considération la coopération au niveau organisationnel. MASA-Méthode,

parmi ses spécificités, la spécification d'un complément méthodologique pour un développement détaillé d'un système multi-agent, une structuration de l'organisation en sous organisation pour permettre un développement hiérarchique descendant de l'organisation, une utilisation de l'approche basée coopération pour la réutilisation des agents (logiciels, physiques et/ou humains) **[Lahlouhi, 06]**.

2.1.1 Organisation

La structure d'une organisation est composée d'éléments, d'une relation de communication entre eux, et de représentants. Dans la réalité, nous fondons une organisation pour pouvoir atteindre un objectif donné. Du point de vue méthodologique, nous mettons, dans notre esprit, un objectif et nous fondons, ensuite, l'organisation qui permettra d'atteindre cet objectif.

Évidemment, l'objectif sera noyé dans l'organisation, une fois qu'elle est établie. Cependant, il est présent, méthodologiquement. En conséquence, l'organisation est définie dans MASA comme un couple (Structure de l'organisation, Objectif global).

Nous pouvons remplacer l'expression « objectif », qualifiée de déclarative, par l'expression procédural « atteindre l'objectif». La tâche « atteindre l'objectif» est la tâche de l'organisation. Nous définissons, en conséquence, l'organisation comme un couple (Structure de l'organisation, Tâche globale).

Une organisation peut être élémentaire ou composée. Dans la première, tous les composants sont des rôles tandis que dans la deuxième ses éléments sont des sous-organisations. Une sous organisation est une organisation (élémentaire ou composée) composant (faisant partie) d'une organisation composée. Elle possède alors une structure et une tâche globale.

2.1.2 Agents et systèmes multi-agents

Un système multi-agent est un ensemble d'agents assumant des rôles d'une organisation donnée. Un agent est un système jouissant des propriétés d'autonomie, de réactivité, de pro-activité et d'habilité sociale. Les agents sont coopératifs dans le sens que leurs objectifs sont des sous-objectifs d'un objectif global d'une organisation. Les objectifs des agents sont alors dérivés de ceux de l'organisation selon une attribution de rôles.

2.1.3 Processus méthodologique de MASA-Méthode

Le processus méthodologique de MASA-Méthode consiste-en :

1. Fondation d'une organisation permettant de satisfaire les besoins du système. Ceci comprend, principalement, la description de l'objectif global de l'organisation et la structure de communication.

2. Dérivation d'un modèle multi-agent permettant de satisfaire l'organisation. Ceci comprend, principalement, l'attribution des rôles aux agents et la dérivation, selon cette attribution, de leurs tâches individuelles à partir de la tâche globale de l'organisation,

3. Implémentation du modèle multi-agent sous forme d'un système orienté objet distribué.

Pour mieux illustrer l'utilisation notre solution, nous avons opté pour le choix de la recherche d'information comme cas d'étude.

3. Techniques de recherche d'information usuelles

Devant la révolution du WEB, plusieurs travaux ont été menés afin de faciliter l'accès aux informations disponibles dans ce gigantesque espace d'information. Les moteurs de recherche représentent une aide inestimable pour la recherche et l'accès aux documents du WEB. Nous pouvons noter que les moteurs de recherche actuels ne tiennent pas compte des spécificités du WEB et sont basés sur des modèles de RI qui ont été développés pour des

documents textuels classiques depuis les années 70 **[Salton, 83].** Ils fonctionnent actuellement sur le principe de page HTML.

3.1 La Recherche d'Information classique

De manière générale, la recherche dans un SRI consiste à comparer la représentation interne de la requête aux représentations internes des documents. La requête est formulée, par l'utilisateur, dans un langage de requêtes qui peut être le langage naturel, un langage à base de mots clés ou le langage booléen. Elle sera transformée en une représentation interne équivalente, lors d'un processus d'interprétation. Un processus similaire, dit indexation, permet de construire la représentation interne des documents de la base documentaire **[Boubekeur, 08].**

La Recherche d'Information (RI) concerne les mécanismes qui facilitent l'accès à une base d'informations. C'est une démarche faite par un utilisateur pour obtenir, à l'aide du système de recherche d'informations (SRI), les informations (ou les références vers les informations) qui peuvent répondre à son besoin.

À l'origine de l'utilisation d'un système de recherche d'information (SRI) se base sur la fonction de correspondance qui n'utilise pas les informations de la base de documents, mais les indexations, qui sont des représentations des informations, dont le but est d'améliorer les performances de la fonction de correspondance (temps et qualité des résultats). Nous distinguons donc quatre composants principaux (Figure VI.1)

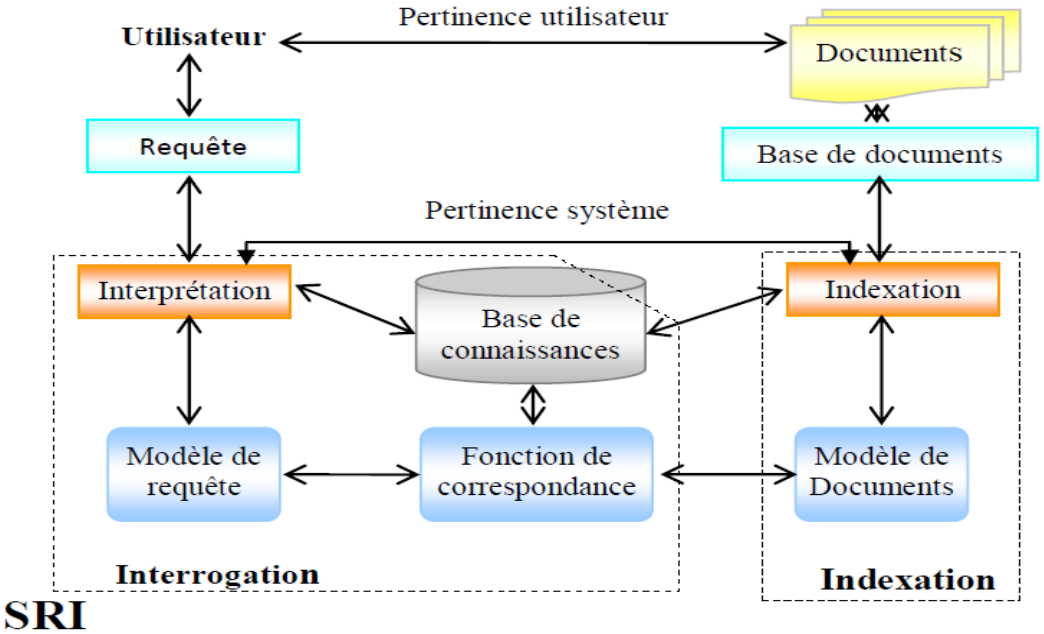


Figure VI.1 : Schéma général d'un modèle de système de recherche

Dans un système de recherche d'information (SRI) la formulation d'une requête impose une représentation du document et un modèle d'extraction d'informations. Dès 1958, Lunh, un des pionniers de la recherche en SRI, a établi dans **[Luhn, 58]** les bases de l'hypothèse fondamentale des travaux sur l'extraction et la sélection d'informations : «*Le contenu textuel d'un document discrimine le type et la valeur des informations qu'il véhicule*». Le quasi totalité des SRI actuels se base sur ce principe. L'analyse de la présence de mots dans un corpus de texte permet de déterminer les documents susceptibles de répondre aux souhaits d'un utilisateur du SRI.

3.2 Systèmes de recherche d'information distribuée (SRID)

Un SRID simple est compose de plusieurs serveurs et un courtier (figure VI.2). Le courtier est le cœur d'un SRID. Le courtier contient cinq composants logiciels : un modèle de gestion, un modèle frontal (une interface utilisateur), un modèle de sélection de serveurs, un modèle de communication et un modèle de fusion de résultats.

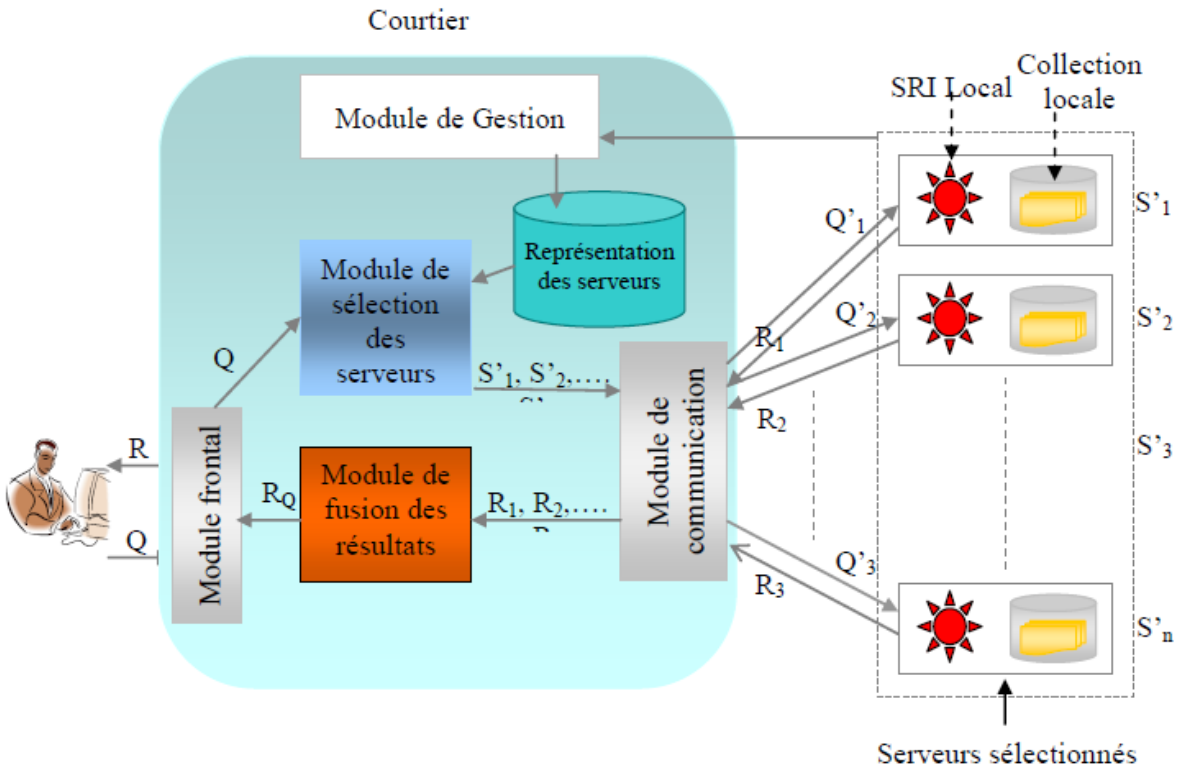


Figure VI.2: Les modules fonctionnels d'un SRID

D'une manière globale, le module de gestion construit les représentants de serveur du système dont il connaît les spécificités. L'utilisateur soumet sa requête par le biais du module frontal (interface utilisateur), le module de sélection de serveurs choisit les serveurs qui sont susceptibles de répondre à cette requête par des documents pertinents, le module de communication adapte alors la requête au langage d'interrogation de chaque serveur sélectionné, leur achemine la requête et réceptionne leur réponse. Enfin, le module de fusion des résultats groupe les réponses reçues par le module de communication et rend via l'interface utilisateur une liste unique de résultats à l'utilisateur.

3.3 RI à base des Systèmes Multi-Agents

D'après Koch **[Koch, 04],** le *paradigme agent* offre des méthodologies et des mécanismes pour la création d'applications distribuées, intelligentes, intégrées et coopératives. Un agent peut fournir des informations adaptées au système de RI auquel il accède. Dans le Web, le contexte de la recherche de l'utilisateur est dynamique puisque les utilisateurs peuvent se déplacer d'un site à un autre, d'un endroit à un autre. Les sources d'informations sont multiples et hétérogènes, le volume d'information est en évolution considérable. Ces changements de l'environnement du Web provoquent des changements dans les tâches et les besoins d'information de l'utilisateur. En conséquence, le paradigme agent peut fournir un apport important à la RI dans le Web**.**

3.4 Recherche d'information par agent mobile

Dès les premières publications, la recherche d'information a été présentée comme une application potentielle importante des agents mobiles voire comme la "killer application" **[Arcangeli, 01].** Divers travaux significatifs ont été menés dans le domaine de la recherche d'information et s'appuient sur la technologie d'agents mobiles. Parmi ceux-ci on distingue:

- Dans **[Brewington, 99],** les auteurs présentent une application de recherche de documents textuels à base d'agents mobiles. Il s'agit cependant d'une application de recherche simple dans un réseau local (avec centralisation des informations sur les serveurs via un mécanisme de pages jaunes, sans contrainte de sécurité, ni découverte dynamique de nouveaux serveurs). L'agent de recherche s'appuie sur des agents d'observation de l'état du réseau et sur un agent stationnaire qui sert d'interface avec le serveur d'information local.

- Le système DBMS-Aglet **[Papastavrou, 00**] implante une solution à base d'agents mobiles en Java pour l'interrogation de bases de données hétérogènes via le Web. Un agent mobile transporte la requête sur le site serveur où il acquiert dynamiquement le pilote JDBC qui convient, il pose ensuite sa requête et retourne sur le site client avec les résultats.

- M3 "MultiMedia Database Mobile agents" **[Kosch, 01]** est un système de recherche de données multimédia par le contenu qui repose sur les agents mobiles, Java et CORBA. L'agent mobile peut mémoriser les informations recueillies sur un site, les utiliser sur les sites visités ensuite, les faire évoluer pendant le parcours. Les problèmes de sécurité sont pris en compte via des mécanismes de sessions indépendantes, les mécanismes de sécurité de CORBA, et des restrictions de droits.

- Enfin d'autres travaux comme AGATHE **[Freitas, 07],** ARCADIA **[Camps, 97],** JAVANE **[Arcangeli, 04],** NETSA **[Côté, 98],** ISAME **[Pelletier, 03]** proposent une autre alternative pour la recherche d'information : des modèles d'agents mobiles et multi-agents.

4. Conception de la solution proposé

On décrit le système de la recherche coopérative d'information dans un réseau par des agents mobiles, comme un système multi-agent. De ce fait, nous appliquons l'approche MASA-méthode pour le développement de ce SMA, et les CPN sont employés pour la description de la tâche globale seulement et non pas pour la description des agents.

4.1 Proposition d'une structure de l'organisation

L'organisation est un aspect important pour l'efficacité du système. Nous notons que l'appui sur une structure organisationnelle dans un système multi-agent peut (et doit) varier selon les besoins de chaque système. Autrement dit, il n'y a pas d'organisation qui est valable

pour toute situation. L'organisation que nous avons choisi est formée d'une organisation élémentaire.

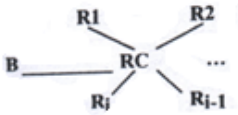
Identificateur	ORI	
Description	Organisation pour RCI	
Tache	Figure VI.3	
Rôles	**Identificateur**	**Modèle**
	R	MR
	B	MB
	CCh	MCCh
Communication entre rôles	**Analytique**	**Graphique**
	R1	
	Ri	
	B	
Légende	**Symbole**	**Description**
	CCh	Chercheur Chef
	B	Bénéficiaire de la recherche
	Rj	Chercheurs
	MCCh	Modèle du Chercheur Chef
	MB	Modèle du Bénéficiaire
	MR	Modèle du Chercheurs
N.B	1. Cette Organisation sera utilisée pour développer un système multi-agents. 2. Cette organisation est élémentaire (elle ne comporte pas de sous organisations)	

Table VI.1 : Organisation de la recherche d'informations

4.2 Formalisation de la tâche globale

Nous décrivions la tâche globale de l'équipe de recherche d'informations dans la Figure 3, nous définissons dans ces équipes trois rôles : un rôle de Bénéficiaire de la recherche **B**, un rôle de Chef-chercheur **CCh** et un rôle de Chercheur **R**. Les abréviations de RdPC associée à cette tâche sont données dans Table VI.2 :

	Symbole	Description
Places	Pj, j [1, 18]	Les places du CPN
Marquage initial	P6, P3	ok
	P7, P11, Rq	Ensemble vide
	P15	0 (zéro)
Transitions	FR	Formuler requête
	SD	Subdivision de ES en ESp associés à des requêtes
	AS	Acquisition de sites pour la recherche
	Ad	Ajout d'un résultat de recherche
	Stc	Structuration résultats
	Int, SE	Servent à minimiser les communications
	S	Rechercher des informations
	Ex	Prend un site de ES
	Tr	Traitement du résultat final de la recherche
Fonctions des arcs	RF	Résultat final
	RRj	Résultat de la recherche du chercheur j
	Sr	Site d'origine
	ESp	Ensemble de sites particuliers
	ES	Ensemble de sites
	D	Site destinataire
	RFS	Résultat final-structuré
	ESj2	Reste de l'ensemble ESj 1, après retrait de D
	U, - et Card	Opérations ensemblistes : Union, différence et Card
	ES	Ensemble de sites
	Rq	Requête
	R	Résultat de recherche
	RR	Résultat de la recherche
	ESj 1	Ensemble de sites associés à Rq
Gardes	V	Ensemble vide
	NV	Ensemble non Vide

Table VI.2 : Abréviations utilisées dans le réseau de Pétri

4.2.1 Description informelle

1. B : formule (FR) une requête Rq,
2. CCh : Acquiert les sites (AS) à visiter ES,
3. CCh : Subdivise (SD) l'ensemble de sites ES en sous ensemble (ESp) et il associe à chacun d'entre eux Rq,
4. Rj (un chercheur) : Prend un ensemble de sites ESj avec Rq,
5. Rj : Prend un site D et Rq,
6. Rj : Va à (Go) D,
7. Rj : Cherche (S) selon Rq,
8. CCh : Ajoute RRj (résultat de la recherche de Rj sur le site D) aux autres résultats,
9. CCh : Une fois tous les sites sont visité (le nombre de sites acquis = nombre de résultats), il structure (Stc) le résultat final,
10. B : Traite (Tr) le résultat de la recherche.

4.2.2 Description formelle

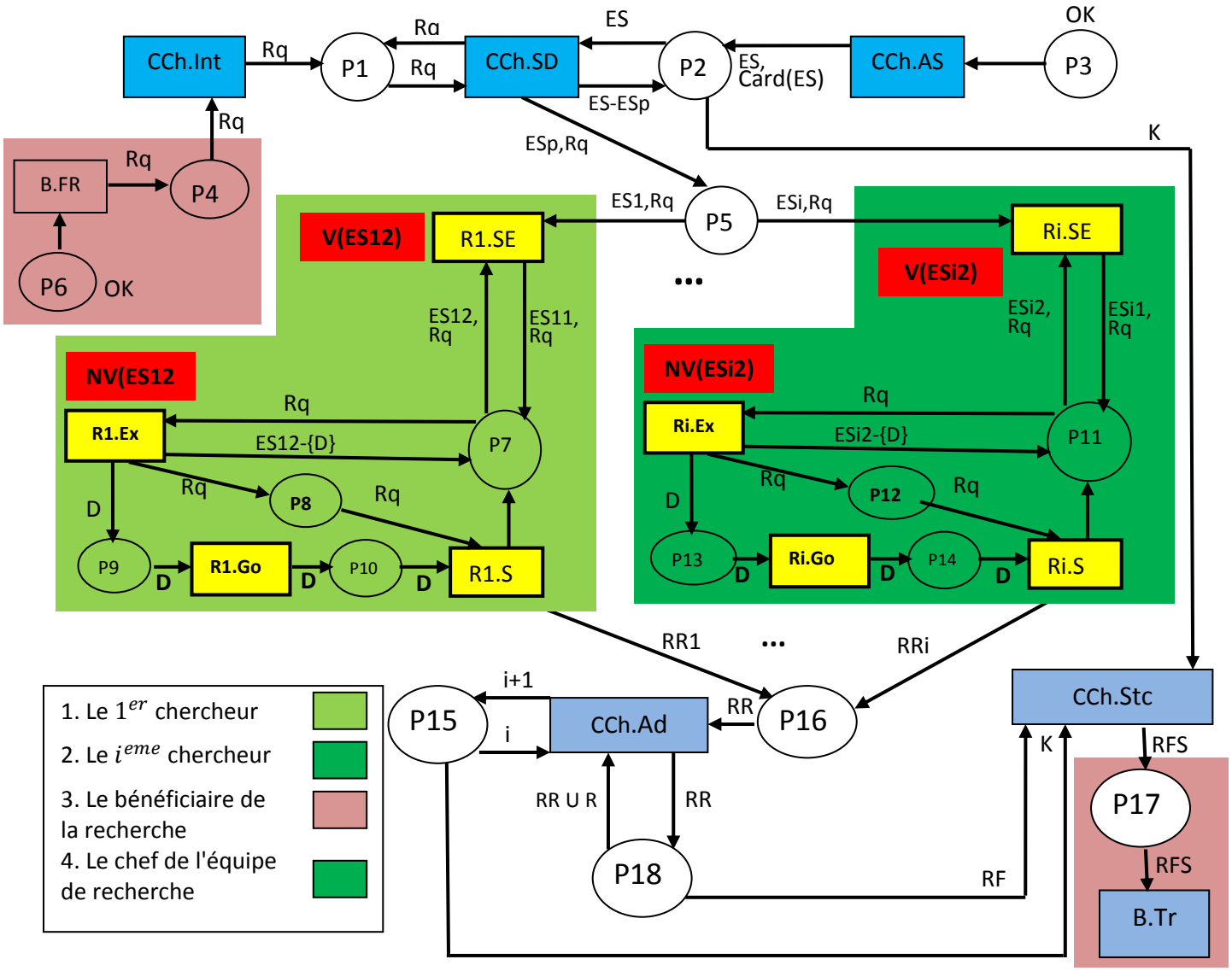


Figure VI.3: La tâche globale de l'organisation de la recherche d'information

4.3 Modélisation

4.3.1 Proposition d'un modèle de système multi-agent

La dérivation d'un modèle multi-agent permettant de mettre en œuvre une organisation donnée se fera selon le processus suivant :

1. Identifier les agents qui mettront en œuvre l'organisation.
2. Attribuer les rôles.
   - Déterminer une association entre les rôles et les agents qui vont les assumer
3. Dériver les aspects collectifs des agents.

- Dériver les tâches individuelles des agents selon l'attribution des rôles
- Vérifier que chacune des tâches individuelles est consistante, en particulier, elle ne doit pas comporter de tâches parallèles

Identificateur	ERI	
Organisation	ORI	
Tache Globale	Figure VI.3	
Description	modèles du système multi agents	
Attribution des rôles	**ID Agent**	**Rôle**
	Humain	B
	AR1	R1
	ARi	Ri
	ACCh	CCh
Légende	**Symbole**	**Description**
	ERI	Equipe de recherche d'informations
	ORI	Organisation de recherche d'informations
	ACCh	Agent Chef-chercheur
	AR1 à ARi	Agent de recherche d'informations
	AB	Agent d'interface
	B	Bénéficiaire
	CCh	Chef-chercheur
	R	Chercheur

Table VI.3 : Modèle de système multi-agent pour la recherche d'information

4.3.2 Distribution des rôles

Puisque notre organisation est élémentaire, elle est constituée de trois rôles distribués sur des agents.

L'attribution des rôles est faite comme suit :

1. Le bénéficiaire (**B**) est un agent humain.
2. Chef chercheur (**CCh**) est un agent logiciel.
3. Pour chaque chercheur (**Ri**), nous associons un agent logiciel.

Les senseurs et les effecteurs des agents sont déterminés à partir des relations des communications entre leurs rôles (Voir Table VI.7, VI.8 et VI.9).

4.3.3 Distribution de la tâche globale sur les agents

Maintenant, nous donnons les taches individuelles de chaque agent.

Identificateur	Humain	
Organisation	ORI	
Rôle	B	
Tache individuelle	**informelle**	1. Formule (FR) une requête Rq 2. Envoyer (AE) ' Requête ' au Chef-chercheur CCh 3. Recevoir (AR) ' Résultat Finale de la recherche ' du Chef-chercheur CCh 4. Traite (Tr) le résultat de la recherche
	formelle	Figure VI.4
Lien de communication	Chef chercheur	
Légende	AE : Acte d'Emission Ar : Acte de Réception	

Table VI.4 : Description de la tache de l'agent Humain

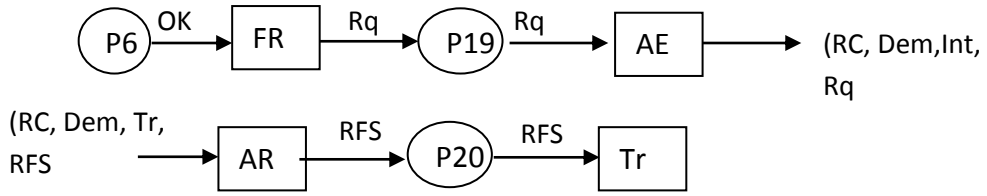


Figure VI.4 : La tâche individuelle de l'agent humain (ceci sera supporter par un agent d'interface).

Identificateur	Agent logiciel	
Organisation	ORI	
Rôle	CCh	
Tache individuelle	**informelle**	1. Acquiert les sites (AS) a visiter ES, 2. Subdivise (SD).l'ensemble de sites ES en sous ensemble (ESp) et il associe à chacun d'entre eux Rq, 3. Recevoir (AR) ‘Requête' du Bénéficier B 4. Envoyer (AE) ‘Ensembles de sites + Requête' au chercheur Rj 5. Recevoir (AR) Resultat de la recherche' du Chercheur Rj 6. Ajoute RRj (résultat de la recherche de Rj sur le site D) aux autres résultats, 7. Une fois tous les sites sont visite (le nombre de sites acquis = nombre de résultats), il structure (Stc) le résultat final, 8. Envoyer (AE) Résultat final de la recherche' au bénéficier B
	formelle	Figure VI.5
Lien de communication	Bénéficiaire et Chercheur	

Table VI.5 : Description de la tache individuelle de l'agent Chef-chercheur

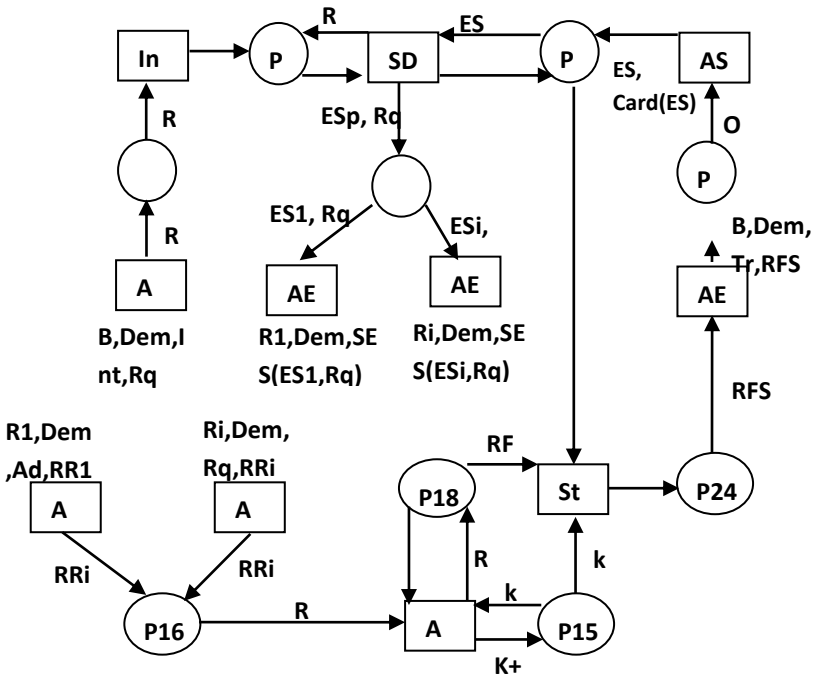


Figure VI.5 : La tache individuelle de l'agent Chef-chercheur de l'équipe de recherche

Identificateur	Agent logiciel	
Organisation	ORI	
Rôle	R	
Tache individuelle	**informelle**	1. Recevoir (AR) ‘Ensembles de Sites Particuliers + Requête' du Chef-chercheur CCh 2. Prend un ensemble de sites ESj avec Rq (SE), 3. Prend un site D et Rq (Ext), 4. Va à (Go) D, 5. Cherche (S) selon Rq, 6. Envoyer (AE) `Résultat de la recherche' au Chef-chercheur CCh
	formelle	Figure VI.6
Lien de communication	Chef-chercheur	

Table VI.6 : Description de la tache individuelle de l'agent Chercheur

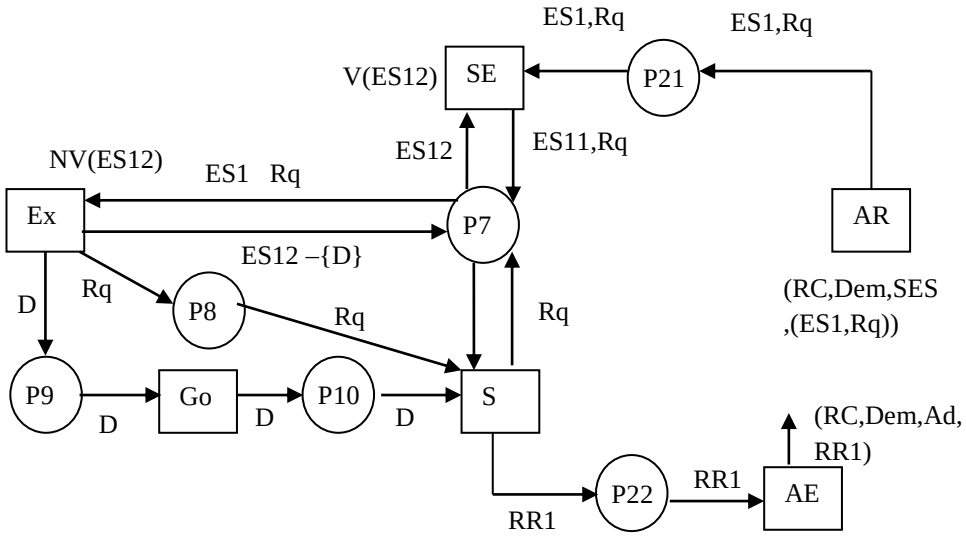


Figure VI.6 : La tache individuelle de l'agent Chercheur

4.3.4 Détermination des savoir-faire des agents et leurs états mentaux

Les savoir-faire de chaque agent sont l'union de tous les savoir-faire des rôles qu'il assume. Son état mental est fonction de sa tâche individuelle, de ses savoir-faire, des liens de communication, des variables qu'il traite.

Agent	Bénéficiaire	
Rôles	B	
Contrôles	Humain	
Ressources	Editeur	
Senseur	S- Bénéficiaire	
Effecteur	E-Bénéficiaire	
Etat	Requête est prêt ou non. Résultat final de la recherche est disponible ou non.	
Tâche individuelle	RdPC Bénéficier	
Savoir-faire		
Procédure	**Entrées**	**Sorties**
Formuler_Requête	0k	Requête
Traite_RésultatDeRecherche	Résultat final Structuré	

Table VI.7 : Modèle de l'agent Bénéficiaire.

Agent	Chef-chercheur	
Rôles	CCh	
Contrôles	Intelligent	
Senseur	S-Chef-chercheur	
Effecteur	E-Chef-chercheur	
Etat	Les Sites disponibles on non. Requête disponibles ou non. Résultat de la recherche disponible ou non..	
Tâche individuelle	RdPC Chef-chercheur	
Savoir-faire		
Procédure	Entrées	Sorties
Sites Acquits	Ok	Ensemble de sites acquits + Card sites
Subdivise_Sites	Ensemble de sites + Requête	Ensemble de sites particuliers + Requête
Int	Requête	Requête
Ajoute_RésultatDeRecherche	Résultat de la Recherche	Résultat de Recherche
Structure Résultats	Résultat de la recherche + Card	Résultat final Structuré

Table VI.8 : Modèle de l'agent Chef-chercheur**.**

Agent	chercheur	
Rôles	R	
Senseur	S-chercheur	
Effecteur	E-chercheur	
Etat	Ensemble de sites particuliers + Requête disponibles on non. Requête disponible on non.Site disponible ou non. Ensemble de sites + Requête disponibles on non.	
Tâche individuelle	RdPC-chercheur	
Savoir-faire		
Procédure	**Entrées**	**Sorties**
SE	Ensemble de sites particuliers + Requête	Ensemble de sites + Requête
Extraire Site	Ensemble de sites + Requête	Ensemble de sites + Requête + Site
Go	Site	Site
Chercher	Site + Requête	Requête + Résultat de la recherche

Table VI.9 : Modèle de l'agent Chercheur

5. Conclusion

Dans ce chapitre, nous avons présenté un modèle multi-agent pour la recherche d'informations coopératives dans un réseau par des agents mobiles en appliquant MASA-Méthode et les réseaux de Petri. Nous commençants par la description de la méthode MASA-méthode. Ensuite une petite recherche sur les techniques de RI (*Recherche d'Information*) .

A la fin nous avons présenté notre solution pour assurer la continuité de l'exécution des taches d'un agent mobile sur des sites hétérogènes, avec la sauvegarde et la restitution de marquage de la tache de l'agent mobile.

BIBLIOGRAPHIE

CONCLUSION

GENERALE

La mobilité est une propriété orthogonale des agents, c'est-à-dire, elle ne concerne que certains agents. Un agent peut rester là et communiquer avec son environnement par des moyens classiques, tels que les invocations de méthodes à distance et la messagerie. Nous appelons les agents qui ne sont pas ou ne peuvent pas se déplacer "agents stationnaires." Un agent stationnaire ne s'exécute que sur le système sur lequel il commence son exécution. S'il a besoin d'informations qui ne sont pas dans ce système ou désire interagir avec un agent sur un autre système, il utilise en général un mécanisme de communication, tels que les invocations de méthodes à distance.

En revanche, un agent mobile n'est pas lié au système sur lequel il commence son exécution. Il peut se déplacer parmi les hôtes du réseau. Créé dans un environnement d'exécution, il peut se transporter (de façon autonome) dans un autre environnement d'exécution dans le réseau, où il reprend l'exécution. Cette capacité lui permet de passer à un système contenant un objet donné avec lequel il veut interagir et de prendre avantage d'être dans le même hôte qu'un tel objet.

Les agents mobiles ont plusieurs avantages : Ils réduisent la charge du réseau, ils s'exécutent de façon asynchrone et autonome, ils s'adaptent dynamiquement, ils sont robustes et tolérants aux pannes. Plusieurs applications bénéficient du paradigme des agents mobiles tels que le e-commerce, l'assistance personnelle, la recherche d'informations distribuées et la surveillance & notification. Cependant, ils posent aussi de nombreuses difficultés telles que les problèmes de leur sécurité et celles de leurs hôtes, … et la continuité de l'exécution de leurs tâches sur plusieurs hôtes d'un réseau. De tels hôtes peuvent être d'un hardware hétérogène ou avoir des plateformes logicielles différentes.

Dans notre travail, nous avons adopté l'approche de **MASA** *(**M**odélisation multi-**A**gents de **S**ystèmes à base de composants **A**utonomes et hétérogènes*) **[Lahlouhi, 06]** comme une méthodologie pour la conception du système multi-agent, et les **CPN** (Colored Petri Nets) pour la modélisation des taches de l'agent mobile.

La continuité de l'exécution à été réalisée par la sauvegarde et restitution du marquage du CPN. Le programme de l'agent sera transféré et lancé sur les différents nœuds des réseaux alors qu'il continu son exécution. Cette solution est rendu simple, portable et robuste par l'utilisation des CPN qui sont un formalisme de haut niveau d'abstraction pour la modélisation des comportements des agents. Ce qui nous place à un haut niveau d'abstraction indépendant de la machine exécutante.

[Acharya, 96] Acharya A., Ranganathan M., Sumatra , S. J. (juillet 1996). A Language for Resource-aware Mobile Programs. Dans 2nd International Workshop on Mobile Object Systems (MOS'96); Linz, Austria.

[Alouf, 02] Sara Alouf and Fabrice Huet and Philippe Nain, Forwarders vs. Centralized Server: An Evaluation of Two Approaches for Locating Mobile Agents. Proceedings of the 2002 International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS-02) pp.278--279, 2002.

[Agha, 86] Agha, G. (1986). Actors : a model of concurrent computation in distributed sys-

tems. MIT Press, Cambridge, MA, USA.

[Aglets, 97] Aglets (1997). http://aglets.sourceforge.net/.

[Arcangeli, 04] Arcangeli, J.-P., Hennebert, V., Leriche, S., Migeon, F. et Pantel, M. (2004). Javact 0.5.0 : principes, installation, utilisation et d´eveloppement d'applications. Rapport technique, IRIT/2004-5-R, Toulouse.

[Bellavista, 01] Paolo Bellavista and Antonio Corradi and Cesare Stefanelli, How to Monitor and Control Resource Usage in Mobile Agent Systems , 2001.

[Bellifemine, 99] Bellifemine, F., Poggi, A. et Rimassa, G. (1999). JADE - a FIPA compliant agent framework. In the 4th International Conference on the Practical Applications of Intelligent Agents, PAAM-99, pages 97–108, London, UK. The Practical Application Company Ltd.

[Benachenhou, 05] Lotfi Benachenhou and Samuel Pierre, A New Protocol for Protecting a Mobile Agent Using a Reference Clone. MATA, pp.364-373, 2005.

[Bernichi, 09] M. Bernichi, Surveillance logicielle à base d'une communauté d'agents mobiles, doctorat en informatique, université Paris 12 Val de Marne, 2009.

[boubekeur, 08]Fatiha BOUBEKEUR-AMIROUCHE. Contribution à la définition de modèles de recherche d'information flexibles basés sur les CP-Nets, thèse doctorat, 2008.

[bouchenak, 01] Sara BOUCHENAK. "Mobilité et Persistance des Applications dans l'Environnement Java", thèse de doctorat de l'institut national polytechnique de Grenoble 19 octobre 2001.

[**Braun, 05]** P. Braun, and W. Rossak. Mobile Agents Basic Concepts, Mobility Models, and the Tracy Toolkit. San Diego. Elsevier Inc. (USA) and dpunkt.verlag (Germany), 2005. Article.

[Brewington, 99] Brian Brewington, Robert Gray, Katsuhiro Moizumi, David Kotz, George Cybenko and Daniela Rus "Mobile agents in distributed information retrieval".

[Cabillic, 93] G. Cabillic et I. Puaut. Stardust: An Environment for Parallel Programming on Networks of Heterogeneous Workstations. Journal of Parallel and Distributed Comuting, Systems, Languages and Applications, 1993.

[Caire, 07] G. Caire (TILAB, formerly CSELT). Jade Tutorial, Jade Programming for Beginner.2007. Livre.

[Camps, 97] V. Camps, M. P. Glelzes : Une technique multi-agent pour rechercher des informations réparties. Institut de Recherche en Informatique de Toulouse. Actes des cinquièmes Journées Francophones IAD et SMA, Editions Hermès, Avril (1997).

[Claessens, 03] Claessens and Preneel and Vandewalle, (How) Can Mobile Agents Do Secure Electronic Transactions on Untrusted Hosts? A Survey of the Security Issues and the Current Solutions. ACMTIT: ACM Transactions on Internet Technology, 2003.

[Carvalho, 04] Marco M. Carvalho and Thomas B. Cowin and Niranjan Suri and Maggie R. Breedy and Kenneth Ford, Using mobile agents as roaming security guards to test and improve security of hosts and networks. SAC pp.87-93, 2004.

[Claud, 01] C.Kaiser, ANNEXE 2 LES RÉSEAUX DE PETRI, Reproduit avec la permission de Francis Cottet, ENSMA décembre 2001.

[Coté, 98] M. Côté, N. Troudi : NetSA : Une architecture multiagent pour la recherche sur Internet. Université Laval. Département d'informatique. Pavillon Pouliot. Ste-Foy, Canada (1998).

[Cubat, 05] Cubat dit Cros, C. (2005). Agents Mobiles Coopérants pour les Envi-ronnements Dynamiques. Th`ese de doctorat, Institut National Polytechnique de Toulouse.

[Dagnat, 00] Dagnat, F., Pantel, M., Colin, M. et Sall´e, P. (2000). Typing concurrentobjects and actors. L'Objet, 6(1):83–106.

[Demazeau, 195] Demazeau, Y. (1995). From interactions to collective behaviour in agentbased systems. In the 1st European Conference on Cognitive Science, pages 117–132, Saint Malo, France.

[Demazeau, 97] Demazeau, Y. (1997). Steps towards multi-agent oriented programming. In the 1st International Workshop on Multi-Agent Systems, IWMAS'97, Boston, MA, USA.

[Diaz, 01] Jesus Arturo Perez Diaz and Dario Alvarez Gutierrez and Igor Sobrado, A fast data protection technique for mobile agents against malicious hosts. Electr. Notes Theor. Comput. Sci, 2001.

[Dillenseger , 02] Dillenseger, B., Tagant, A.-M. et Hazard, L. (2002). Programming and executing telecommunication service logic with moorea reactive mobile agents. In the 4th International Workshop Mobile Agents for Telecommunication Applications, MATA 2002, pages 48–57. Springer.

[Dimitrov, 98] B. Dimitrov and V. Rego. Arachne: A Portable Threads System Supporting Migrant Threads on Heterogeneous Network Farms. Proceedings of IEEE Parallel and Distributed Systems, Volume 9, Numéro 5, 1998.

[Espinasse, 10] B. Espinasse, Université d'Aix-Marseille, Brève introduction aux agents logiciels, 2010. Support de cours, disponible sur : www.lsis.org/espinasseb/Supports/RIWS-2010/IntroAgents-2010-4p.pdf

[Ferber, 95] Ferber, J. (1995). Les systèmes multi-agents : vers une intelligence collective.Inter Editions, Paris, France, 1'ere édition.

[FIPA, 02] Foundation for Intelligent Physical Agents (www.fipa.org). FIPA Abstract architecture Specification, December 2002.

[Florea, 02] A. M. Florea, Agents et systèmes multi-agents, University of Bucharest – 2002. Support de cours, disponible sur le site : http://turing.cs.pub.ro/auf2/

[**Freitas, 07]** F. Freitas, B. Espinasse, S. Fournier : AGATHE: une architecture générique à base d'agents et d'ontologies pour la collecte d'information sur domaines restreints du Web (2007).

[Fuggetta , 98] Fuggetta, A., Picco, G. P. et Vigna, G. (1998). Understanding codemobility. IEEE Transactions on Software Engineering, 24:342–361.

[Fünfrocken, 98] S. Fünfrocken. Transparent Migration of Java-based Mobile Agents(Capturing and Reestablishing the State of Java Programs). Proceedings of Second International Workshop Mobile Agents 98 (MAÐ) , Stuttgart, Allemagne, septembre 1998. http://www.informatik.tu-darmstadt.de/~fuenf

[Galtier, 01] Virginie Galtier and Kevin L. Mills and Yannick Carlinet and Stephen F.Bush and Amit Kulkarni, Predicting and Controlling Resource Usage in a Heterogeneous Active Network. Active Middleware Services, pp.35-44, 2001.

[Gomoluch, 01] J. Gomoluch and M. Schroeder, Information agents on the move: A survey on loadbalancing with mobile agents, 2001.

[Grasshopper, 95] Grasshopper (1995). http://cordis.europa.eu/infowin/acts/analysys/products/thematic/agents/ch4/ch4.htm.

[Groot, 04] David R. A. De Groot and Frances M. T. Brazier and Benno J. Overeinder, Cross-Platform Generative Agent Migration, 2004.

[Guessoum, 01] Z.Guessoum, & M. Occello. "Environnements de développement. Dans Principes et architectures des systèmes multi-agents", Hermès, Lavoisier, 2001.

[Hantz, 06] F. Hantz and H. Guyennet, A P2P Platform using sandboxing. HPCS'06, Workshop on security and high Performance computing systems, In conjunction with ECMS 2006, 20th European Conf. on Modelling and Simulation, Bonn, Germany, pp.736-739, 2006.

[Hohlfeld, 02] Matthew Hohlfeld and Aditya Ojha and Bennet Yee, Security in the Sanctuary System, http://historical.ncstrl.org/tr/ps/ucsd_cs/CS2002-0731.ps, 2002.

[Hong, 07] L. Hong. Architectural Design of Multi-Agent Systems: technology and techniques. Harshy New York : Information Science Reference, 2007.

[Illmann, 01] Illmann, T., Krueger, T., Kargl, F. et Weber, M. (décembre 2001). Transparent Migration of Mobile Agents Using the Java Debugger Architecture. Dans The Fifth IEEE International Conference on Mobile Agents (MA'2001), Atlanta, Géorgie, États-Unis.

[Ismail, 99] L. Ismail and D. Hagimont and J. Mossiere, Evaluation of the Mobile Agents Technology: Comparison with the Client/Server Paradigm, 1999.

[Jennings, 00] Jennings, N. R. (2000). On agent-based software engineering. Artificial Intelligence, 117(2):277–296.

[Jensen. 97] Kurt Jensen. A Brief Introduction to Coloured Petri Nets. Lecture Notes in Computer Science, No 1217, Springer-Verlag, 1997, pp 203-208.

[Jensen. 98] Kurt Jensen. An Introduction to the Practical Use of Coloured Petri Nets. Lecture Notes in Computer Science, No 1492, Springer-Verlag, pp 237-292, 1998.

[Jonathan, 97] Jonathan Dale. A Mobile Agent Architecture to Support Distributed Resource Tnformation Management. A thesis submitted to transfer feom Master of Philosophy to Doctor of philosophy in the Faculty of Engineering. University of Southampton Department of Electronics and Computer Science 22 January, 1997.

[Knabe, 95]F. Knabe. Language Support for Mobile Agent. Thèse de Doctorat, School of Computer Science, Carnegie Mellon University, 1995. http://www.cs.virginia.edu/~knabe

[kosch, 01] H. kosch, M Doller, L. Boszormenyi ,Content-based indexing and retrival supported by mobile agent technology 2001 .

[Lahlouhi, 02] A. LAHLOUHI, S. ZAIDI, A. REFRAFI, S. AZIZI, H. KHELIFA, F. MIMI, L. KAHLOUL and A. ATTAOUA " MASA Méthod : A Multi-agent Development méthodology " 6th world mulyi-conference on systemic, cybernetics and informatics (SCI 2002) Software engineering of multi agents systems session 14-18-2002.

[Lahlouhi,, 06]Lahlouhi, A., 2006: Modélisation multi-agent du processus multi-agent. PhD Thesis, University of Constantine

[Lange , 97] Lange, D. B., Oshima, M., Karjoth, G. et Kosaka, K. (1997). Aglets :Programming mobile agents in java. Worldwide Computing and Its Applications, 1274/1997:253–266.

[Litzkow, 92] M. J. Litzkow et M. Solomon. Supporting Checkpointing and Process Migration Outside the UNIX Kernel. USENIX Winter Conference, pages 283 – 290, San Francisco, CA, Etats-Unis, janvier 1992.

[Loulou, 10] M. Loulou, Approche Formelle pour la Spécification, la Vérification et l'Imposition des politiques de Sécurité Dynamiques dans les Systèmes à base d'Agents Mobiles, thèse de doctorat en informatique de l'université Sfax en cotutelle avec l'université de Bordeaux 1, 2010.

[Luhn, 58] H. P. Luhn. The automatic creation of literature abstracts. IBM Journalof Research and Development, pages 159–165, 1958.

[Maamar, 98] Z. Maamar, Aperçu général sur la technologie des agents mobiles, University of Waterloo, Waterloo, Ontario, Canada, 1998.

[Matthiske, 95] B. Matthiske, F. Matthes et J. Schmidt. On Migrating Threads. Proceedings of the Second International Workshop on Next Generation Information Technologies and Systems, Naharia, Israel, 27 – 29 juin1995. http://www.sts.tu-harburg.de/projects/Tycoon/

[Mencer, 04] D.E. Menacer, Un modèle d'architecture à base d'agents mobiles pour les applications réparties. 2004. Thèse doctorat.

[Milojicic, 98] Milojicic, D. S., Breugst, M., Busse, I., Campbell, J., Covaci, S.,Friedman, B., Kosaka, K., Lange, D. B., Ono, K., Oshima, M., Tham, C., Virdhagriswaran,S. et White, J. (1998). MASIF : The OMG mobile agent system interoperabilityfacility. In the 2nd International Workshop on Mobile Agents, MA'98, pages 50–67, Stuttgart,Germany.

[Outtagarts, 09] Outtagarts, A. (2009). Mobile agent-based applications : a survey. International Journal of Computer Science and Network Security, 9(11):331–339. [Russell et Norvig, 2009] Russell, S. et Norvig, P. (2009). Artificial Intelligence : A Modern Approach. Prentice-Hall, Englewood Cliffs, NJ, USA, 3éme édition.

[Panzoli, 08] D. Panzoli, Proposition de l'architecture « Cortexionist » pour l'intelligence comportementale de créatures artificielles, thèse doctorat de l'université de Toulouse. 2008.

[Papastavrou, 00]S. Papastavrou, G. Samaras, E. Pitoura: Mobile Agents for World Wide Web Distributed Database Access, IEEE Transactions on Knowledge and Data Engineering, (2000).

[Peine, 02] Holger Peine. "Run-Time Support for Mobile Code" Doctoral (PhD) thesis, University of Kaiserslautern, Germany, ISBN 3-925178-93-7, 21 october 2002.

[Pelletier, 03] S. J. Pelletier, S. Pierre, H. H. Hoang: Modeling a Multi-Agent System for Retrieving Information from Distributed Sources. CIT "Computing and Information Technology", (2003).

[Perret, 97] S. Perret, « Agents mobile pour l'accès nomade à l'information répartie dans les réseaux de grande envergure », Thèse pour obtenir le titre de docteur de l'université Joseph Fourrier – Grenoble I, Novembre 1997.

[Pierre, 03] S. Pierre, Réseaux et Systèmes Informatiques Mobiles : Fondements, Architecture. Livre.

[Robbert, 01] Robbert S Gray, George Cybenko, David Kotz, Ronald A. Peterson and Daniela Rus. D'Agents : Application and Performance of a mobile Agent System. Thayer School of Engineering / Department of Computer Science Dartmouth College HAnover, New Hampshire 03755, Nouvember 28, 2001.

[Sakamoto, 00] Sakamoto, T, Sekiguchi, T, Yonezawa, A. (septembre 2000). Bytecode Transformation for Portable Thread Migration in Java. 4th International Symposium on Mobile Agents 2000 (MA'2000), Zurich, Switzerland.

[Salton, 83]Gerard Salton, Edward A. Fox, et Harry Wu. Extended boolean information retrieval. Commun. ACM, 26(11), pages 1022–1036, 1983.

[Sekiguchi, 99] T. Sekiguchi, H. Masuhara et A. Yonezawa. A Simple Extension of Java Language for Controllable Transparent Migration and its Portable Implementation. Coordination Languages and Models, Lecture Notes in Computer Science, Volume 1594, avril 1999. http://web.yl.is.s.u-tokyo.ac.jp/amo/

[Shub, 90] C. M. Shub. Native Code Process-Originated Migration in a Heterogeneous Environment. Proceedings of the 1990 ACM Annual Conference on Cooperation, pages 266 – 270, Washington, Etats-Unis, 20 – 22 février 1990.

[Smith, 96] P. Smith et N. C. Hutchinson. Heterogeneous process Migration: The Tui System. Rapport Technique 96-04, British Columbia University, avril 1996.

[Tisseau, 01] Tisseau, J. (2001). Virtual reality — in virtuo autonomy —. Habilitation à diriger des recherches, Université de Rennes 1.

[Steensgaard, 95] B. Steensgaard et E. Jul. Object and Native Code Mobility Among Heterogeneous Computers. Proceedings of the 15 th ACM Symposium on Operating System Principles (SOSPÍ) , Copper Mountain Resort, Colorado, Etats-Unis, décembre 1995.

[Suezawa, 00] Suezawa, T. (juin 2000). Persistent Execution State of a Java Virtual Machine. ACM Java Grande 2000 Conference; San Francisco, CA, USA.

[Trillo, 07] Trillo, R., Ilarri, S. et Mena, E. (2007). Comparison and performance evaluation of mobile agent platforms. In the 3rd International Conference on Autonomic and Autonomous Systems, ICAS ’07, pages 41–46, Athens, Greece. IEEE.

[Tran, 05] V. Tran, V. Moraru, Réseau de Petri, Institut de la Francophonie pourl'Informatique Promotion 10 15 juillet 2005.

[Truyen, 00] Truyen, E, Robben, B, Vanhaute, B, Coninx, T, Joosen, W, Verbaeten, P. (septembre 2000) Portable Support for Transparent Thread Migration in Java. Dans 4th International Symposium on Mobile Agents 2000 (MA’2000); Zurich, Switzerland.

[Wooldridge, 02] Wooldridge, M. J. (2002). An introduction to MultiAgent systems. John Wiley & Sons, New-York, NY, USA, 1st édition.