UNIVERSITE SAAD DAHLAB DE BLIDA

Faculté des Sciences

Département d'Informatique

MEMOIRE DE MAGISTERE

Spécialité : Ingénierie des systèmes et de la connaissance

ETUDE ET CONCEPTION D'UN FRAMEWORK A BASE DE PATRONS, POUR LA CONSTRUCTION D'UN AGENT INTELLIGENT ET MOBILE SOUS J2ME.

Par

Redha MAZARI

Devant le jury composé de :

A. Guessoum	Professeur, U de Blida	Président
S. Oukid	Maitre de conférences, U. de Blida	Examinatrice
N. Benblidia	Maitre de conférences, U. de Blida	Examinatrice
D. Djenouri	Maitre de recherche, CERIST	Examinateur
M. Mahieddine	Maitre de conférences, U. de Blida	Promoteur

Blida, Mai 2011

RESUME

Cette mémoire de magistère présente *to (True Object)*, un cadre d'application (framework) pour le développement d'agents intelligents et mobiles sur téléphones mobiles.

Les dispositifs mobiles et les agents montrent un regain de popularité ces dernières années, et la tentation de les associer ensembles pourrait offrir des possibilités intéressantes. Les téléphones mobiles continuent à avoir des ressources limitées, et l'utilisation des agents mobiles, bien qu'elle soit très séduisante, elle identifie certaines limitations dont on doit tenir compte lorsqu'on voudrait développer un framework d'agents à base de patrons pour [illegible] se trouvant sur le marché.

Cette thèse présente une illustration de comment utiliser des patrons pour concevoir un framework d'agents avec des possibilités de réutilisabilité.

Les patrons de conception peuvent simplifier la complexité de la conception au niveau de la micro-architecture, et pourrait donc constituer une base d'expérience réutilisable pour construire du logiciel réutilisable.

Pendant le processus de conception, nous utilisons plusieurs patrons de conception, tels que l'*Observer*, le *Template Method*, le *Command*, le *Proxy*, le *Decorator*, le *Singleton*, ainsi que le *Layered Architecture*.

Cette thèse décrit une investigation empirique dans la conception de frameworks à base de patrons. Son but est d'identifier les problèmes majeurs de réutilisabilité et de l'impact de la conception à base de patrons sur ces problèmes. Une approche qualitative est employée et quatre problèmes majeur de réutilisabilité sont identifiés

comme étant une barrière à la réutilisabilité: comprendre la fonctionnalité des composants, comprendre l'interaction entre les composants, le passage du domaine du problème à l'implémentation du framework et la compréhension des hypothèses architecturales dans la conception du framework.

Le *to* (True Object) est un framework multi agents, à base de patrons, pour téléphones mobiles qui va permettre de concevoir une collection d'agents appelés *to*, qui fournissent des capacités d'intelligence et de mobilité.
Ce qui est unique en *to* est la combinaison des technologies de conception par patrons et frameworks pour la résolution des problèmes et l'utilisation du multi-agents où chaque *to* exécute son propre comportement et travaille avec les autres *to*s pour résoudre des problèmes spécifiques. Chaque *to* exécute son propre comportement en utilisant un moteur de raisonnement (actuellement un moteur d'inférence

Ce travail va servir d'exemple qui va encourager les autres chercheurs développeurs d'appliquer plus d'évaluation de la conception du framework sur le téléphone mobile, dans le futur.

Mots clés : Agents, Framework, Pattern, J2ME, MIDlet, Conception orienté objet, appareil sans fil.

الملخص

يعرض في هذه الأطروحة To، إطارا للإنشاء الوكلاء النقالة الذكية على الهواتف الجوالة.
ولقد زادت شعبية الأجهزة النقالة والوكلاء في خلال السنوات الماضية، وجمعهم معا يمكن أن تقدم مزايا مثيرة للاهتمام. اليوم أجهزة الجوال لديها موارد محدودة بالمقارنة مع أجهزة الكمبيوتر المكتبية، وإنشاء التطبيقات المتقدمة عليها يمكن أن يكون تحديا.

كائن صحيح (TO) هو إطار يعتمد على الأنماط لإنشاء نظام متعدد الوكلاء للهواتف النقالة التي تدعم مجموعة من الوكلاء ، والتي توفرعلى قدرات التفكير.

وهذا العمل بمثابة المثال الذي سوف يشجع الباحثين المطورين الآخرين لتطبيق المزيد من التقيم لتصميم الإطارات على الهاتف المحمول في المستقبل.

الكلمات الرئيسية: الوكلاء، الأجهزة النقالة، إطار البرامج، أنماط التصميم، J2ME، MIDlet، البرمجة الموجه للكائن.

ABSTRACT

This thesis presents To, a framework for developing mobile intelligent agents on mobile phones. Mobile devices and agents have increased in popularity throughout the last years, and bringing them together can offer interesting features. Today's mobile devices have limited resources compared to stationary PCs, and making advanced applications for them can be challenging. This thesis identifies some limitations that have to be considered when developing an agent pattern-based framework for mobile devices Java 2 Micro Edition enabled currently in the market.

Observer, the Template Method, the layered Architecture pattern, the Proxy, the Decorator, and the Singleton.

This thesis describes an empirical investigation into pattern-based framework design. Its aim is to identify the major problems of reuse and the impact of the pattern-based design techniques on these problems. A qualitative approach is employed and four major reuse problems are identified as barriers to reuse: understanding the functionality of components; understanding the interactions between components; the mapping from the problem domain to the framework implementation and understanding the architectural assumptions in the framework design.

The True Object (TO) is a pattern-based, multi-agent framework for mobile phones that supports a collection of agents called TO, which provide reasoning capabilities.

What is unique about TO is in the combination of patterns and framework design technologies for problem solving and the use of a multi-agent system where each TO executes its own behavior and works with other TO to solve specific problems. Each TO executes its behavior using a Reasoning Engine (currently an inference engine) that is capable of combining all of its internal structures, mental state and goals for the purpose of problem solving especially involving Qualitative Reasoning.

The thesis shows that the combination of a pattern language and micro architecture can provide useful support for framework reuse but both require modification to become more effective. The thesis also concludes that the evaluation of framework design is an essential activity for the advancement of framework comprehension. It serves as an example to encourage other researchers to perform more evaluation of framework design on mobile phone, in the future.

REMERCIEMENTS

Les remerciements témoignent qu'une thèse, quoiqu'on en pense, est avant tout une entreprise collective. Si ce travail a pu être mené à son terme, je le dois en effet à de nombreuses personnes dont je ne pourrais faire la liste exhaustive ici.
Je veux néanmoins marquer ma reconnaissance à ceux qui m'ont encadré, aidé ou encouragé.

Je tiens à remercier notre créateur **ALLAH** pour m'avoir permis d'achever ce travail avec tout son sérieux et son exigence scientifique. Je lui exprime ma gratitude pour la confiance qu'il m'a toujours manifestée.

Cette thèse n'existerait pas sans lui, source intarissable d'idées, d'enthousiasme et d'énergie, qui m'a apporté bien plus qu'un encadrement technique".

Je souhaite à tous les thésards de bénéficier de l'encadrement d'un directeur tel que Mahieddine M. . Son expérience, sa disponibilité, son ouverture, son soutien sans faille m'ont permis de mener cette "aventure" à son terme en confiance.

Ce travail n'aurait pu s'aboutir sans l'implication de Madame Oukid S. et Mademoiselle Benblidia N. qui m'ont défendue contre vents et marrées. Qu'elles trouvent ici tout mon témoignage de reconnaissance, de gratitude et de profond respect. Il doit aussi aux nombreux étudiants qui m'ont aidé par quelques mots gentils, une question au sujet de l'avancement de mon travail, ou une quelconque bonne intention à mon égard.

Je tiens à remercier aussi Mr GESSOUM A. pour avoir accepté de juger ce travail.

Je ne peux pas oublier de mentionner la grande compétence et la justesse d'appréciation de Monsieur Le Recteur, Monsieur Le Vice-recteur chargé de la post-graduation, et Monsieur Le vice doyen chargé de la post-graduation, concernant la gestion des problèmes des étudiants, et des conflits.

Que le conseil scientifique de la faculté des sciences, de par son directeur et tous ses membres acceptent ma profonde gratitude.

J'ai apprécié l'ouverture d'esprit et la curiosité qu'ont manifestés les enseignants du département d'informatique, étudiants de Magister et chercheurs du laboratoire LRDSI.

J'ai apprécié l'aide de Mademoiselle N. Djillali concernant sa précieuse aide pour l'aboutissement de ce mémoire

Merci aussi à mes parents, grands parents, particulièrement à ma mère et à mon père qui ont accepté de me soutenir sentimentalement et matériellement durant toute la durée de mes études, et particulièrement pendant le déroulement de cette thèse.

TABLE DES MATIERES

RESUME 2
ABSTRACT 5
REMERCIEMENTS 7
TABLE DES MATIERES 9
LISTE DES FIGURES 12
LISTE DES TABLEAUX 10
INTRODUCTION 11
1. DÉFINITION DU PROBLÈME 13
2. BUTS VISÉS PAR LE PROJET 13
3. CONTEXTE DU PROJET 14
4. CONTRIBUTIONS 14
5. MÉTHODOLOGIE SUIVIE 16
6. STRUCTURE DU MÉMOIRE 17
1. ETAT DE L'ART 21
1.1. INTRODUCTION 21
1.2. PROGRAMMATION ORIENTÉE OBJETS 21
1.3. L'ARCHITECTURE DU LOGICIEL 21
1.3.1. ARCHITECTURE DU LOGICIEL RÉUTILISABLE 23
1.4. PATRONS 23
1.4.1. PATRONS DE CONCEPTION 23
1.4.2. PATRONS ARCHITECTURAUX 24
1.5. FRAMEWORKS 25
1.5.1. SQUELETTE D'UN FRAMEWORK 25
1.5.2. TYPES DE FRAMEWORKS 25
1.5.3. DESCRIPTION DES FRAMEWORKS 26
1.6. RELATION ENTRE LES PATRONS ET LES FRAMEWORKS 27
1.7. LES SYSTÈMES MULTI-AGENTS 27
1.7.1. LE PARADIGME AGENT 28
1.7.2. QU'EST-CE QU'UN AGENT? 29
1.7.3. QU'EST-CE QU'UN SYSTÈME MULTI-AGENTS ? 31
1.7.4. AGENTS ET OBJETS 32
1.7.5. AGENTS INTELLIGENTS 33

1.7.6. SYSTÈME EXPERT 36
1.7.7. AGENTS MOBILES 39
1.7.8. AGENTS MOBILES POUR DISPOSITIFS MOBILES 40
1.8. PATRONS POUR LA CONCEPTION DES AGENTS 41
1.9. CONCLUSION 43
CHAPITRE 2 : ETAT DE L'ART 44
2.1. INTRODUCTION 44
2.2. DOMAINE DES AGENTS 44
2.3. LES SYSTÈMES D'AGENT MOBILE EXISTANTS SUR ORDINATEURS 44
2.3.1. AJANTA 45
2.3.2. JADE 45
2.3.3. MTA 46
2.3.4. BEE-GENT : (HTTP://WWW2.TOSHIBA.CO.JP/BEEGENT/INDEX.HTM) 46
2.3.5. AGLETS: (HTTP://WWW.TRL.IBM.CO.JP/AGLETS) 46
2.3.6. JATLITE 47
2.3.7. VOYAGER SYSTEM 47
2.4.1. JADE-LEAP 48
2.4.2. SAFE-M 49
2.4.3. MDA 50
2.4.5. µOS 50
2.4.6. AGENTLIGHT 51
2.4.7. MIA 51
2.4.8. µFIPA-OS 52
2.4.9. KSACI 52
2.5. CONCLUSION 52
CHAPITRE 3 : PROGRAMMATION DES DISPOSITIFS MOBILES SOUS J2ME 53
3.1 INTRODUCTION 53
3.2 LA DIVERSITÉ DE PÉRIPHÉRIQUES 53
3.3 L'ARCHITECTURE DU JAVA MICRO EDITION(J2ME) 54
3.3.1 MIDP (MOBILE INFORMATION DEVICE PROFILE) 55
3.4 LES MIDLETS 55
3.5 APPLICATION DÉVELOPPÉE AVEC LE J2ME TOOLKIT 56
3.6 CONCLUSION 57
CHAPITRE 4 : CONCEPTION DE L'ARCHITECTURE DE TO 59
4.1. INTRODUCTION 59
4.2. L'ARCHITECTURE PROPOSÉE 60
4.3. CONCEPTION DE L'ARCHITECTURE 62
4.3.1. LE COMPORTEMENT DE L'AGENT (BEHAVIOR) 63

4.3.2. L'ENVIRONNEMENT DE L'AGENT (INTERACTION) 65
4.3.3. SYSTÈME DE COMMUNICATION DE L'AGENT 67
4.3.4. SYSTÈME DE RAISONNEMENT DE L'AGENT 73
4.4. IMPLÉMENTATION DU FRAMEWORK TO 76
4.4.1. L'ENVIRONNEMENT 76
4.4.2. LA COMMUNICATION 78
4.4.4. LE SYSTÈME DE RAISONNEMENT 83
4.4.5. LE COMPORTEMENT 85
4.5. CONCLUSION 86
CHAPITRE 5: APPLICATION 87
5.1. INTRODUCTION 87
5.2. INSTANCIATION DU FRAMEWORK 87
5.3. TEST SUR L'OBJET MAN 88
5.3.1. PRINCIPE 88
5.3.2. L'APPLICATION 90

LISTE DES FIGURES

Figure 1-1: Types de Framework..... 26
Figure 3-1: Diversité des appareils embarqués……..…….54
Figure 3-2: Une vue générale des environnements Java……..54
Figure 3-3: Les API de MIDP et CLDC……..……..55
Figure 3-4: Emulateur J2ME Toolkit……..…….57
Figure 4-1: Le Système à Couches ……..……61
Figure 4-2: Architecture en couche de la plate-forme de l'agent To ……62
Figure 4 3: le Comportement de To et son Etat ……..…..64
Figure 4-4: L'interaction socio-environnement ……..65
Figure 4-5: les Classes Message de TO 70
Figure 4-7: Diagramme de classe pour la Communication 79
Figure 4-8: Diagramme de classe pour Message 80
Figure 4-9: Diagramme de classe pour Méthode de traitement des messages 81
Figure 4-10: Diagramme de classe pour MobileAgent 82
Figure 4-11: Diagramme de classe pour le serveur 83
Figure 4-12: Diagramme de classe pour le Raisonnement 84
Figure 4-13: Diagramme de classe pour le Comportement 85
Figure 5-1: Diagramme de classe de ToMan 89
Figure 5-2: Ecran d'accueil ToApplication 92
Figure 5-3 Le Menu principal 93
Figure 5-4: Affichage de la liste des faits 94
Figure 5-5: Affichage des règles de la base de règles ……..95
Figure 5-6: La vue de l'agent To ……..96
Figure 5-7: démarrage du système expert ……...97
Figure 5-8: L'agent toMan1 ……..98
Figure 5-9: La MIDlet affiche le KMessage ……...99
Figure 5-10: Migration de l'agent "Partie 1" ……105
Figure 5-11: Migration de l'agent "Partie 2" ……..105

LISTE DES TABLEAUX

Tableau 1-1: Schéma de classification des patrons de conception [62]. 24

INTRODUCTION

Les téléphones mobiles et les PDAs sont devenus de plus en plus avancés.
Les téléphones sont passés du stade de simples téléphones en des véritables assistants. L'utilisation de ces téléphones s'est grandement généralisée, et est devenue donc d'un intérêt public.

Le marché de dispositifs mobiles continue à se développer, avec un nombre toujours croissant de consommateurs de domaine, de professions et d'exigences divers.

d'offre d'applications intelligentes pour le domaine des dispositifs mobiles.
Ce manque d'applications intelligentes est dû partiellement aux contraintes imposées au logiciel pour les dispositifs mobiles, avec la croyance que les limitations de la mémoire et de la capacité de traitement rendraient les dispositifs mobiles incapable d'adopter la technologie puissante de l'IA. La difficulté de produire des applications intelligentes réutilisables pour le marché des dispositifs mobiles va aggraver le problème de la diversité du marché de dispositifs mobiles, exigeant que n'importe quelle application intelligente puisse être portable.

Les applications Internet deviennent de plus en plus omniprésentes dans notre vie quotidienne. Leur dernière cible étant le réseau téléphonique mobile et son énorme

base d'utilisateurs. Les gens utilisent de plus en plus des équipements ultra-portables pour accéder à l'information d'une manière immédiate et continue.

La technologie des agents présente le potentiel de jouer un rôle primordial dans cette révolution technologique en permettant d'automatiser les processus d'acquisition et de traitement de l'information assurant ainsi une prestation de services plus intelligents.

Dans l'optique des objectifs que nous nous sommes assigné, nous avons essayé de comprendre la conception des systèmes multi-agents en étudiant les différentes méthodes existantes [01], [02], [03], et, parallèlement, nous avons étudié le génie logiciel [04], plus particulièrement le génie logiciel objet et les techniques de réutilisation [05], [07], [08], [09].

L'approche To proposée dans ce mémoire va permettre peut être de s'attaquer beaucoup dans le futur proche à plus de développement d'applications intelligentes portatives sur dispositifs mobiles réutilisables.

1. Définition du Problème

Les agents logiciels et les dispositifs mobiles deviennent les deux de plus en plus populaires, ainsi une étude de conception sur un domaine combinant les deux serait bien intéressante et d'actualité :

- Il y a un besoin de systèmes d'agent sur de petits dispositifs qui adressent les contraintes de mémoire et d'exécution des dispositifs mobiles.

- Le taux de croissance de téléphones mobiles sur le marché augmente plus rapidement que les ordinateurs, en particulier dans les pays en voie de développement.
- Le potentiel pour déployer les frameworks d'agents à base de Java est plus grand pour le marché mobile parce que la plupart des téléphones viennent avec une machine virtuelle Java (JVM) J2ME.
- Les plateformes traditionnelles d'agent ont été développées pour les environnements de bureau.

Il y a quelques éléments incertains au sujet de la mobilité des agents sur téléphones portables aussi; c'est une technologie tout à fait nouvelle, et là il n'y a pas beaucoup de systèmes existants qui combinent J2ME, agents, et conception basée sur des frameworks.

La définition du problème pour cette thèse serait de voir comment J2ME peut être

dispositifs mobiles, et identifient les défis et les limitations les plus importants que nous avons rencontrés.

Cette recherche s'adresse au manque d'environnements de développement d'agents intelligents et mobiles sur téléphones portables.

L 'objectif de cette recherche présente à la fin, un framework ouvert [illegible] développement et au test d'application des techniques et résultats de recherche sur des agents intelligents et mobiles.

Le principal but de cette recherche est de concevoir et développer un modèle de framework basé sur les agents intelligents et mobiles sur des dispositifs de téléphone portables.

Les débuts de cette recherche consistent en l'analyse des conditions spécifiques des agents intelligents et mobiles, et de leurs caractéristiques, en vu de la conception d'un framework.

Un certain nombre de publications se rapportent à l'utilisation de patrons [12] dans la conception des agents [13], cependant notre travail diffère principalement par les points suivants :

- nous avons proposé notre propre choix de patrons,
- nous avons conçu notre architecture générale au moyen d'un patron,
- nous avons incorporé chaque caractéristique des agents comme des micro-architectures à patrons,
- nous avons favorisé la qualité de réutilisabilité de logiciel dans le choix de nos patrons.

3. Contexte du projet

Ce projet est entrepris au sein de l'équipe GLODOO, de Génie LOgiciel et de Développement Orienté Objets faisant partie du LDRSI (Laboratoire de Développement et de Recherche en Systèmes Informatisés), situé à l'université de Blida.

4. Contributions

Notre objectif est donc d'aider à la conception de systèmes multi-agents pour le domaine de l'ubiquitaire en induisant une émergence forte au sein du processus même de conception. Une meilleure compréhension de ce processus, permettrait à la fois au concepteur et au développeur de mieux le maîtriser.
Le but du génie logiciel est d'optimiser et de rationaliser le travail de conception de logiciels et le suivi de celui-ci. L'optimisation consiste à:

- minimiser la complexité du travail à accomplir tant lors de la création que lors de la maintenance,
- augmenter la qualité et la fiabilité logicielles.

Dans ce mémoire, nous nous sommes penchés plus particulièrement sur deux techniques de réutilisation : les patrons (motifs ou patterns), et les frameworks.

La contribution de ce projet s'articule principalement autour de :

- la conception,
- l'implémentation et
- l'application

d'un framework à base de patrons pour la réalisation d'agents intelligents et mobiles pour téléphones portables sous J2ME.

Cette thèse couvre des aspects avancés de la technologie des objets, avec une visée particulière de résolution des problèmes avec des patrons de conception aussi bien que des architectures à base frameworks pour la conception d'agents mobiles et

Les patrons de conception sont présentés comme des propulseurs de la productivité, c'est-à-dire d'outils d'aide aux développeurs pour résoudre les problèmes de conception en orienté objets.

A un niveau plus élevé, les modèles architecturaux structurent l'organisation globale du framework. Les patrons architecturaux résolvent les forces principales en structurant l'application de manière optimisée et permettent la réalisation d'un framework d'agents modifiable et réutilisable. En conclusion, on propose un certain nombre de patrons et on montrera le rôle joué par chacun de ces patrons dans la conception du framework visé.

Cette thèse vise à améliorer de manière significative des possibilités de développement des réalisateurs de frameworks. La conception par les patrons architecturaux est explorée, et leur utilisation dans la conception est pratiquée comme facteurs significatifs découlant d'un choix judicieux de patrons.

Après étude de ce travail, les concepteurs pourront être en mesure :

- d'identifier les concepts fondamentaux et avancés de la conception et des patrons architecturaux,

- d'Apprendre et d'appliquer directement les patrons de conception aux problèmes de développement,
- de structurer les systèmes en appliquant les modèles architecturaux.
- de concevoir les systèmes et les framework flexibles et maintenables

L'intention de ce projet est d'étudier comment J2ME (édition micro de Java 2) peut être utilisé pour développer un framework à base de patrons pour déployer les objets mobiles intelligents qui sont destinés pour des dispositifs mobiles qui sont tenus dans la main (tels que téléphones cellulaires, PDAs etc..) . La tâche principale de ce projet est d'étudier comment fonctionne le framework sur la plateforme de J2ME fonctionne, ainsi que l'investigation de ses avantages et de ses limitations, et recommande finalement une solution sur la façon pour développer le framework pour déployer des agents mobiles intelligents.

5. Méthodologie suivie

Avec les avancées potentielles dans les domaines des agents il n'y avait aucune surprise de trouver une pléthore de projets relatifs aux agents. Il est impossible de mentionner toutes les différentes solutions qui ont été mises en application dans la recherche et par les entreprises commerciales.

Après le temps dépensé on étudiant les nombreux projets et les articles de recherches, les choses ont commencé à s'éclaircir pour nous, et nous avons compris

aussi qu'il y'avait plusieurs méthodologies qui étaient généralement suivies, pour entreprendre le développement dans le domaine des agents.
Par conséquent dans cette section, au lieu d'aborder une liste de solutions spécifiques nous allons plutôt exposer ce que nous avions synthétisé comme caractéristiques principales des différentes méthodologies proposées.

Après avoir passé en revue les systèmes d'agent nous avons remarqué qu'il y a au fait qu'un nombre réduit, d'outils, capable de fournir à des réalisateurs pour développer des agents.

Par conséquent un framework de développement est fortement souhaitable. Il ferait avancer considérablement les capacités des réalisateurs pour résoudre les problèmes qu'ils doivent résoudre sans être entravés en développant également un

environnement de réception avec d'autres agents, et il va avoir besoin d'observer les autres agents (interprétant leurs actions) aussi bien que d'effectuer ses propres actions.

Développer une compréhension si détaillée du scénario en interprétant l'environnement est une tâche considérable pour un agent dans une plateforme à ressources limitées, surtout pour traduire conceptuellement les processus tels que la perception, le raisonnement, et l'action, qui sont fondamentaux pour la conception d'un agent.

La première partie de cette thèse présente un tel agent, conçu autour de la notion de la réactivité adaptative où chaque processus d'agent est interruptible par un

raisonneur. Ce raisonneur met en application actuellement un algorithme basé sur des règles de production qui interprète des perceptions et permet à l'agent de s'impliquer, s'il est suffisamment réactif, dans son environnement courant et lui donne ainsi une dynamique mentale.
Quand une action est demandée, le processus de raisonnement produit une décision d'action, ce qui va assurer la réactivité de l'agent.

La conception de l'architecture de l'agent pour produire le comportement intelligent est non triviale. Cette thèse présentera une architecture basée sur des patrons, où chaque constituant est introduit en termes de couches qui sont conçus séparément ; et montre que les cinq principaux processus pour le développement des agents (comportement, communication, interaction, raisonnement, et mobilité) peuvent être mis en commun dans un framework collectif.

Cette architecture est élaborée pour concevoir un framework approprié pour fournir un guide structurel pour faciliter l'exécution du comportement par un agent complexe fonctionnant dans un environnement dynamique.

Cette architecture est censée être plus facilement comprise, menant à une re-conception, une modification et à des améliorations plus efficaces; ma[illegible] niveau ces avantages ne sont pas facilités par la limitation du langag[illegible] servir dans l'implémentation.

Le contenu de ce mémoire varie beaucoup, que ça soit concernant le champ de l'application ou de l'approche suivie. Certaines parties du rapport pourraient être plus intéressantes à certains lecteurs, que d'autres. Afin d'augmenter la lisibilité du rapport, nous présenterons donc un aperçu de chaque chapitre.

Grosso modo, le plan de ce mémoire comporte deux principales parties : La première partie concerne l'étude que nous avons menée pour aboutir à la réalisation de ce travail, elle permettra de décrire les concepts d'agent, de techniques de conception et de réutilisation, du langage *java* et de son bénéfice pour l'implémentation des agents, de la notion de patron et framework. Dans la seconde nous exposerons la

démarche que nous avons adoptée pour réaliser notre propre optique de conception, ainsi que la spécialisation de notre framework dans une application.

Chapitre I

Dans le premier chapitre, nous aborderons l'aspect génie logiciel de l'état de l'art. Nous y définirons la réutilisabilité et exposerons quelques techniques de réutilisation utilisées en génie logiciel et dans le domaine des systèmes d'agents en particulier. Nous aborderons notamment les concepts d'objet, de patron, de framework, et d'agent.
Ce chapitre constituera un état de l'art des méthodes, des techniques et des outils actuellement mis à la disposition du concepteur des agents.

Dans ce chapitre nous allons exposer les caractéristiques des dispositifs sans fil visés par notre recherche. Aussi nous allons exposer Java 2 Micro Edition comme un langage de programmation destiné à ce type de dispositifs.

Chapitre IV

Dans le quatrième chapitre, nous présentons notre plateforme et décrirons sa structure, et c'est dans ce chapitre que nous expliquons comment la plateforme incorpore les attributs de qualité de l'architecture de logiciel et agit l'un sur l'autre avec l'environnement.
Ce chapitre donnera un exemple de mise en application de notre credo : utiliser les patrons orientés logiciels pour développer et documenter un framework pour agents.

Ainsi, nous avons présenté dans le chapitre trois la réutilisabilé comme qualité du logiciel, nous allons dans ce chapitre montrer comment l'introduire pratiquement dans une vraie architecture, celle du framework *To*.

C'est dans ce chapitre que nous développerons la méthode que nous avons choisie : les frameworks et les patrons de conception. Nous essaierons de débroussailler les notions intervenant lors du passage d'un patron à son utilisation. Ce chapitre expose la liste des motifs que nous avons introduit dans notre framework. Dans ce chapitre nous allons détailler le processus suivi pour d'abord découvrir les principales caractéristiques d'un agent, ensuite comment nous avons procédé dans la phase de conception en utilisant un patron architectural pour la conception globale du framework, et comment nous avons choisi les patrons les plus adapté pour les incorporer en couches à notre squelette architecturale, pour la description des caractéristiques de base d'un agent, en ayant en tête un constant souci de réutilisabilité du logiciel.

Ce chapitre nous a permis de fournir des descriptions et de démontrer l'utilisation des patrons comme outils aidant à la conception.

Chapitre V

L'application choisie a pour but principal de prouver la réalisabilité d'une implémentation concrète de notre framework. Dans ce chapitre nous allons montrer comment nous avons spécialisé notre framework par une application visant surtout à démonter la faisabilité de notre travail, dans un domaine d'application réduit.

Conclusion

Un bilan et quelques unes des perspectives envisageables sont présentés comme conclusion.

Annexe 01

Cette première annexe listera tous les outils et logiciels utilisés dans les étapes de conception et de réalisation du framework ainsi que ceux utilisés pour rédiger le mémoire.

CHAPITRE 1 :

CONCEPTS FONDAMENTAUX

1.1. Introduction

Le but final de cette recherche est de fournir une architecture réutilisable pour organiser notre framework.

Ce chapitre explique les concepts théoriques centraux au sujet de la conception des frameworks à base de patrons [14], et comment ils peuvent être em… dispositifs mobiles. Ce chapitre explique également quelques concep… sujet des agents et de système multi-agents.

To est un framework pour agents intelligents, qui est conçu autour de plusieurs technologies. Les technologies courantes qui sont à la base de To sont divisées en domaines de recherche suivants : les Systèmes d'Multi-Agent (MAS), les frameworks, les patrons, le raisonnement, et la programmation de la mobilité pour les dispositifs mobiles sous j2me.
Tous ces domaines seront expliqués dans les sections suivantes.

1.2. Programmation Orientée Objets

Certains des avantages de l'utilisation de la programmation orientée objets sont la compréhensibilité de la conception, l'encapsulation de l'information, l'extensibilité et la réutilisation du programme pour un futur développement, et la modularité de la conception. Ce qui rendra le programme facile à examiner, à tester et à retrouver d'éventuelles erreurs.

Les principales techniques utilisées pour réutiliser le comportement sont

- Héritage
- composition

Dans l'héritage

déléguant les requêtes aux objets entrant dans la composition.

1.3. L'Architecture du Logiciel

Le terme *architecture* comporte plusieurs significations, parfois contradictoires.
Dans notre recherche nous considérons que l'architecture traite de la structure des composants d'un système, de leurs relations et de directives régissant leur conception et évolution en un temps fini [15] [16] [17].

1.3.1. Architecture du Logiciel Réutilisable

La réutilisabilité du logiciel [18] fournit plusieurs avantages aux développeurs elle réduit :

- le coût,
- le temps,
- l'effort, et
- elle permet également d'améliorer la qualité du logiciel qui est créé [19].

Les principales étapes dans le développement pour la réutilisation sont [20]:

- de re-concevoir les composants des systèmes existants pour les rendre plus généraux et réutilisables,
- d'améliorer la conception des composants pour s'assurer qu'elles sont de qualité acceptable, et
- [illegible] relatif au but et aux dépendances afin de faciliter leur futur utilisation.

Les facteurs de qualité d'un composant qui affectent la réutilisabilité sont [21] :

- la fiabilité,
- la testabilité, and
- la maintainabilité.

1.4. Patrons

1.4.1. Patrons de Conception

Les patrons (motifs ou modèles) de conception [62] sont principalement prévus pour offrir des solutions génériques aux problèmes qui se répètent dans la conception du logiciel.

La *Bande des Quatre* répertorie 23 patrons de conception, classés en 3 catégories [22] :

- créationnels : pour construire des composants,

- structurels : pour connecter des composants, et
- organisationnels (comportementaux) ; pour la communication entre les composants.

Tableau 1-1: Schéma de classification des patrons de conception [22].

BUT			
PORTEE	CREATIONEL	STRUCTUREL	COMPORTEMENTAL
CLASSE	Factory Method	Adapter (class)	Interpreter
			Template Method
OBJET	Abstract Factory	Adapter (object)	Command.
	Builder	Bridge	Iterator
	Prototype	Composite	Mediator
	Singleton	Decorator	Memento
		Flyweight	State
		Proxy	Strategy
			Visitor
			Chain Of Resp.

Les plus importants avantages des patrons sont:

- **l'abstraction** : le patron fournit une solution claire et bien fondé élevé d'abstraction;
- **la communication** : les patrons fournissent des noms communs pour des problèmes et des solutions typiques, qui facilitent la communication parmi les développeurs de logiciel ;
- **la documentation** : un modèle décrit clairement un problème bien défini et une solution générale, dans un format fixe mais extensible ;

1.4.2. Patrons Architecturaux

Un modèle architectural adresse le principe pour la structuration globale de l'architecture du logiciel.

Shaw a identifié sept modèles qui guident la conception de système à niveau élevé et ont discuté la manière qu'ils guident la composition des systèmes des types particuliers de composants [12].

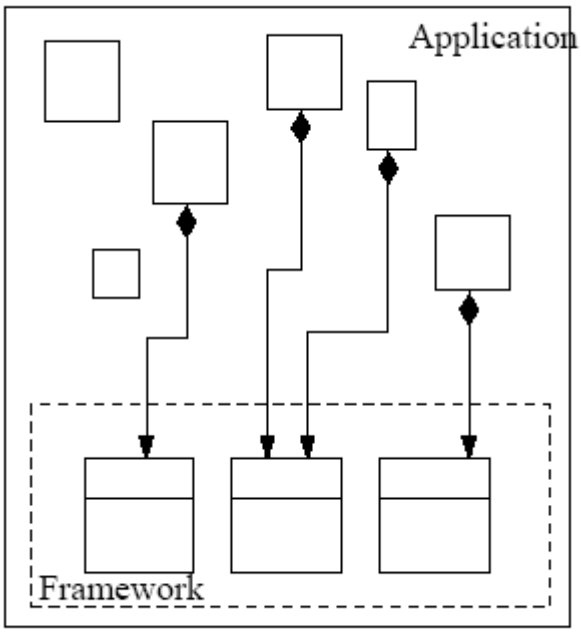
Buschman et al [23] présentent le patron *Model-View-Controller* comme un patron architecturel.

1.5. Frameworks

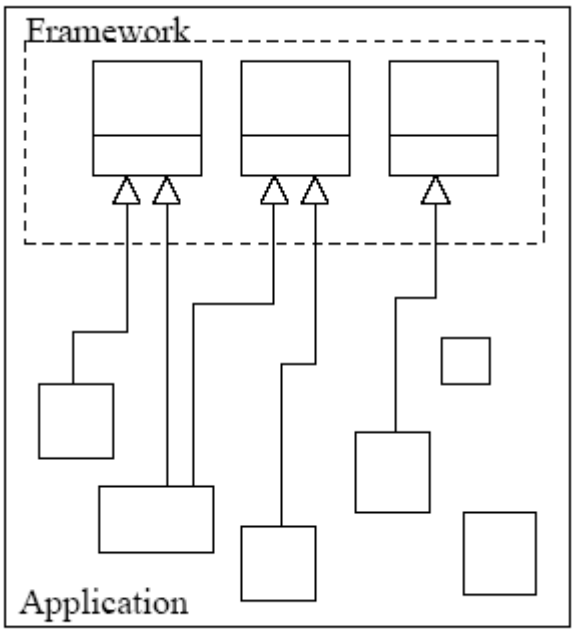
Les frameworks (ou les cadres de logiciel) sont des applications logicielles de grande échelle, qui sont conçues pour être réutilisés.
En dépit de leur utilité, les frameworks sont des structures compliquées à apprendre et l'effort et le temps passé à comprendre la façon de les utiliser peut dépasser leur

1.5.2. Types de Frameworks

Les frameworks sont divisées en frameworks de boîte noire, et en frameworks de boîte blanche [14]. La couleur se rapporte à la quantité de commande que le développeur de l'application a appliqué sur la personnalisation du framework.



Framework de boîte noire



Framework de boîte blanche

fournis par le framework pour le configurer pour les différentes applications.

1.5.3. Description des Frameworks

La compréhension des frameworks orientés objets est difficile.
La conception du framework fixe certains rôles et responsabilités parmi les classes, aussi bien que des protocoles de leur collaboration. La variabilité dans la famille des applications est factorisée dans des *hotspots*, et le framework fournit les mécanismes pour adapter chaque *hotspot*. L'adaptation est faite en sous-classant une classe existante du framework et en redéfinissant les méthodes abstraites.

Un framework existe pour soutenir le développement d'une famille d'applications.

Un framework est habituellement conçu par des experts en la matière dans un domaine particulier et il est utilisé par des non experts. Il permet à l'utilisateur de réutiliser des conceptions abstraites, et des composants préfabriqués afin de développer un système dans le domaine d'application du framework.

1.6. Relation entre les Patrons et les Frameworks

Les patrons de conception constituent une base en expérience pour construire du logiciel réutilisable, et ils agissent comme des blocs de construction à partir desquels des conceptions complexes peuvent être élaborées.
Chaque patron de conception est une micro-architecture pour un élément de conception récurrent.

1.7. Les systèmes multi-agents

La matière de base de cette thèse s'articule autour des agents, des environnements, et de l'interaction des agents dans leur environnement [25].
Ce travail vise donc l'exploration de l'interaction d'agents capables de raisonner avec d'autres agents dans un environnement. Un agent capable de raisonner est n'importe quelle entité qui agit dans son environnement et dont l'action est appropriée (c.-à-d., contribue à réaliser les buts de l'agent) relativement à ses connaissances et ses intentions du moment [26].

Il existe une multitude de définitions d'un agent, selon

- la source d'entrée à l'agent (ou à la manière avec laquelle les agents agissent l'un avec l'autre dans le monde dans lequel ils vivent),
- les fonctions de l'agent, et

- le rôle de l'agent en tant que partie d'une société multi-agents (aucune frontière ou caractéristique claire ne distingue un agent complexe d'une société des agents).

Pour faire face à une telle complexité, les agents sont conçus sur la base d'une structure ou d'une architecture particulière.

L'agent, appelé *To*, que nous développons dans ce travail est un agent rationnel, dont les actions du comportement sont basées sur les buts de l'agent, son état mental, et ses intentions.

1.7.1. Le paradigme agent

d'organisations empruntées aux domaines de la vie et de la société [25].

Les SMA sont particulièrement adaptés pour proposer des solutions à des problèmes complexes pour lesquels il n'existe pas de contrôle centralisé [27]. Ils sont des systèmes idéaux pour représenter des problèmes possédant de multiples méthodes de résolution, de multiples perspectives et/ou de multiples résolveurs.

Ces systèmes possèdent les avantages traditionnels de la résolution distribuée et concurrente de problèmes, et ils héritent aussi des bénéfices de l'Intelligence Artificielle comme le raisonnement symbolique (au niveau des connaissances).

Les types courants d'*interaction* incluent :

- la *coopération* qui implique une activité commune d'un ensemble d'agents afin d'atteindre un but commun,
- la *coordination* qui suppose l'organisation de l'activité de résolution de chaque agent afin d'éviter les interactions inutiles et d'exploiter celles qui sont bénéfiques,
- la *négociation* dont l'objet est d'arriver à un compromis acceptable entre tous les agents engagés en compétition.

Jusqu'à maintenant nous n'avons pas de définition formelle de ce que c'est qu'un agent ou un système multi-agent, qui soit acceptée par tout le monde [27].

1.7.2. Qu'est-ce qu'un agent?

Dans la littérature, les définitions du concept agent présentent certaines similitudes, mais dépendent aussi du type d'application pour laquelle est conçu l'agent.

Dans le cadre de notre travail, nous porterons notre attention sur les définitions qui suivent :

Ferber a proposé assimile le terme agent à une entité physique ou virtu

- qui est capable d'agir dans un environnement,
- qui peut communiquer directement avec d'autres agents,
- qui est mue par un ensemble de tendances, sous la forme d'objectifs individuels, d'une fonction de satisfaction ou de survie, qu'elle cherche à optimiser,
- qui possède des ressources propres,
- qui est capable de percevoir partiellement son environnement,
- qui ne dispose que d'une représentation partielle de l'environnement,
- qui possède des compétences et offre des services,
- qui peut éventuellement se reproduire,

- dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa perception, de ses représentations et des communications qu'elle reçoit.

Wooldridge et Jennings [29] ont défini l'agent comme étant un système Informatique,

- situé dans un environnement,
- qui agit d'une façon Autonome et
- flexible pour atteindre certains objectifs pour lesquels il a été conçu.

Situé

Un agent est dit situé, s'il est capable d'agir sur son environnement à partir des capteurs sensoriels qu'il reçoit de ce même environnement.

Autonome

Un agent est dit autonome s'il est capable d'agir sans l'intervention d'un tiers (humain ou agent) et contrôle ses propres actions ainsi que son état interne, en peut

- **réactif** : les agents intelligents qui peuvent percevoir leur environnement, et répondent dans un certain délai opportun aux changements qui se produisent afin de satisfaire leurs b
- **proactif** : les agents intelligents qui peuvent exhiber le comportement dirigé par ses buts en prenant l'initiative afin de satisfaire leurs objectifs.

Flexibilité

Un agent est dit flexible s'il est capable d'agir de manière réactive, proactive et sociale.

Sociabilité

La sociabilité est la capacité permettant à un agent d'interagir avec les autres agents quand la situation l'exige (pour compléter ses tâches ou coopérer avec eux).

Un agent est donc une entité physique ou virtuelle en situation dans un environnement avec lequel il interagit de façon autonome et flexible.

Cette notion de situation ou d'agent situé implique que les agents autonomes doivent agir dans un monde réel :

- changeant,
- partiellement observable, et
- imprévisible.

De même ils doivent réagir en temps réel parce que l'environnement change constamment, et parce qu'ils doivent éventuellement tenir compte des actions d'autres agents.

Dans cette thèse, nous nous intéresserons aux notions de perception partielle de [illegible] précédents auteurs représentant bien la définition que nous adoptons pour notre agent.

Comme pour les agents, plusieurs définitions ont été proposées, et nous allons commencer par présenter la définition introduite par Ferber[28].

Un système multi-agents (SMA) est composé des éléments suivants :

- un Environnement E c'est à dire un espace disposant généralement d'une métrique,
- un ensemble d'objets O, ces objets sont situés, c'est à dire que pour tout objet, il est possible, à un moment donné, d'associer une position dans E. ils sont passifs, ils peuvent être perçus, détruits, créés et modifiés par les agents,
- un ensemble d'Agents A qui sont les entités actives du système,
- un ensemble de Relations R qui unissent les objets entre eux,
- un ensemble d'Opérations Op permettant aux agents de percevoir, de détruire, de créer, de transformer et de manipuler les objets de O,

- un ensemble d'opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification (les lois de l'univers).

Une autre définition est celle donnée par Wooldridge et Jennings, qui présente un SMA comme étant un ensemble d'agents en interaction afin de réaliser leurs buts ou d'accomplir leurs tâches [29]. Les interactions peuvent être directes par l'intermédiaire des communications, comme elles peuvent être indirectes via l'action et la perception de l'environnement.

Les interactions peuvent être mises en œuvre dans un but de :

- coopération entre les agents, lorsqu'ils ont des buts communs,
- coordination, c'est à dire d'organisation pour éviter les conflits et tirer le
- compétition, lorsque les agents ont des buts antagonistes.

Dans la thèse nous adoptons la définition de Woldridge et Jennings [29], un SMA étant un ensemble d'agents ayant des buts ou des tâches, et qui interagissent pour les accomplir en mode de coopération et/ou de compétition.

1.7.4. Agents et Objets

Un objet est défini par un ensemble de services offerts (ses méthodes) qu'il ne peut refuser d'exécuter si un autre objet le lui demande. Les objets exécutent des tâches ; ils n'ont ni but, ni recherche de satisfaction.

En revanche les agents, bien plus élaborés, disposent d'objectifs qui leur donnent une autonomie de décision vis à- vis des messages qu'ils reçoivent.
Les objets utilisent un mécanisme d'envoi de message qui se résume à un simple appel de méthode. Pour les agents, les interactions sont plus complexes et font intervenir des communications de haut niveau, où l'important est que l'agent décide par lui-même comment interagir et réagir aux messages qu'il reçoit.

Un objet est contraint de répondre aux requêtes qui lui sont soumises, contrairement à l'agent qui dispose de son propre libre arbitre.
Concrètement, les objets font ce qu'on leur demande, tandis qu'avec les agents, on peut s'attendre à un refus et des fois il faut négocier.
Les agents doivent contrôler eux-mêmes leur comportement et les ressources qu'ils possèdent. C'est ce qui caractérise leur autonomie.

Il est ici indispensable de combiner une communication plus évoluée que celle des objets avec la recherche de satisfaction des objectifs de chaque agent.

Néanmoins, le lien existant entre la notion d'objet et celle d'agent reste fort.
Un aspect qui rajoute de la confusion entre la notion d'objet et celle d'agent, est que

1.7.5.1. Les caractéristiques principales des Agents intelligents

Pour réaliser leurs buts, les agents doivent comporter un type particulier de raisonnement connu sous le nom de *raisonnement pratique*.

Les attributs ou les caractéristiques des agents intelligents sont :

- *Délégation* : accomplir les tâches au nom d'un utilisateur
- *Communication* : avec l'utilisateur ou avec d'autres agents
- *Autonomie* : fonctionne sans interposition directe

- *Surveillance* : surveiller son environnement afin de pouvoir accomplir des tâches de façon autonome
- *Intelligence* : pouvoir interpréter les événements surveillés pour prendre des décisions appropriées de mise en action (adaptation automatique)

1.7.5.2. Intelligence des Agents

Dans cette section, nous présenterons un certain nombre de concepts du domaine des systèmes de raisonnement et des agents intelligents [31].
Une approche intéressante pour distinguer les agents intelligents peut être basée sur le rapport entre

- les actions externes (comportement) et
- l'adaptation interne.

L'adaptation effectuée à l'intérieur de l'architecture du système évolue avec le temps

- comment traiter des agents comme systèmes au niveau de la connaissance en leur assignant de la connaissance et de
- comment décider pour chaque agent quelle tâche va-t-il

qu'il faudrait apprendre sur les rapports entre les tâches (un agent accomplissant certaines tâches pourrait pouvoir accomplir d'autres tâches).

Dans de tels modèles, des agents adaptatifs peuvent être caractérisés par la tâche qu'ils accomplissent.

1.7.5.3. Agents et Raisonnement Basé sur les Règles

Le système (ou moteur) de raisonnement de *To* est un framework général acceptant n'importe quel système de raisonnement.
Pour l'instant, on n'a spécialisé ce framework de raisonnement qu'avec un moteur d'inférence chaînage avant à base de règles.

L'agent *To* raisonne donc dans son domaine, concernant les décisions à prendre au sujet des actions qu'il pourrait entreprendre, en utilisant un système de déduction à base de règles de production.

Un système de déduction logique (ou de raisonnement) *basé sur les règles* est un système expert qui emploie un ensemble de règles en tant que base de modèle de son savoir-faire.

Les systèmes de raisonnement basés sur les règles se combinent bien avec les agents pour deux raisons :

- les règles rendent une définition compacte du comportement possible,
- sous sa forme la plus simple, un comportement est un ensemble d'actions et de conditions sous lesquelles ces actions devraient se produire.

Dans cette section, nous allons surtout explorer certaines des techniques employées pour représenter la connaissance dans un domaine dans les programmes

Nous allons principalement nous concentrer sur la façon dont les diverses représentations de la connaissance peuvent être employées pour implémenter le raisonnement pour les agents.

1.7.5.4. Représentation de la Connaissance

La connaissance est définie comme "fait ou état de savoir quelq

la connaissance gagnée par l'expérience ou l'association."

Il y a deux types primaires de représentations de la connaissance, *procédural* et *déclaratif*.

La *logique de prédicats* ou *logique propositionnelle* peut être employé pour représenter des objets, des fonctions, et des relations. *La résolution* et *l'unification* sont des techniques employées pour prouver des théorèmes et pour déduire de nouveaux faits en utilisant la logique.

Une base de connaissance est une collection de la connaissance liée à un domaine spécifique du problème.

1.7.6. Système Expert

Un *système expert* ou un *système à base de connaissanc*es est le terme utilisé pour décrire un système de traitement à base de connaissance et déduction logique.

Un système expert consiste principalement, en:

- une base de connaissance, et
- un moteur d'inférence, qui comporte la logique de raisonnement utilisé pour traiter les règles (inférer) et les données (faits).

1.7.6.1. Base de Connaissance

Elle organise une grande partie de la connaissance de résolution des problèmes.
La base de connaissance est le dépôt central d'information contenant les faits que

Parmi les raisons de ceci, que nous pouvons mentionner :

- La connaissance représentée comme les règles si alors est facilement compréhensible,
- la plupart des personnes peuvent les lire facilement,
- chaque règle peut être vue comme un morceau autonome de la connaissance ou comme unité d'information de connaissance de base,
- de nouvelles connaissances peuvent être facilement ajoutées, et la connaissance existante peut être changée simplement en créant ou en modifiant différentes règles,
- les règles sont facilement manipulables par les systèmes de raisonnement.

La connaissance dans les systèmes de raisonnement à base de règles est représentée sous la forme de règles de productions.

1.7.6.3. Règles de Production

Les règles de production sont des relations simples de la forme : *Si une certaine condition est vraie, faire alors quelque chose.*
Une règle définit l'ensemble de faits qui doivent être vrais (partie *si*) avant qu'un ensemble d'actions (la partie *alors*) puisse être exécuté.

Le format d'une règle est :

- partie si : condition, lieux, antécédent,
  Elle est habituellement une conjonction de fait - (faits particuliers dans la base de données pour effectuer une action)
  partie alors : action, conclusion, conséquence.
  Si la partie gauche de la règle, qui peut inclure :
  - des actions qui affectent le monde extérieur,
  - [illegible]

1.7.6.4. Faits

Un fait est une construction qui définit une information qui est connu pour être vrai, comme par exemple le soleil est chaud, le ciel est bleu.

1.7.6.5. La Base de Connaissance ou Base de Règles

Elle contient une grande partie de la connaissance de résolution des problèmes.
La connaissance permet d'interpréter l'information dans les bases de données.

1.7.6.6. Le moteur d'inférence

L'inférence est aux ordinateurs ce qu'est le raisonnement pour les humains.
La méthode de raisonnement employée dans un moteur d'inférence est *le modus ponens* :
Si A est vrai, et A ➔B est alors B est dérivé pour être vrai.

Quand une règle est satisfaite la conséquence peut arriver à une conclusion à placer d'autres faits dans la base de faits. Ce fait additionnel, peut aussi induire par enchaînement d'autres faits à rejoindre la base de faits.
Le processus de raisonnement par *chaînage avant* est l'interprétation de règles.

1.7.6.7. Avantages de la méthodologie des règles

Les systèmes basés sur les règles de raisonnement sont puissants parce que les actions elles-mêmes peuvent déduire de nouveaux faits.
Parmi leurs avantages, on pourrait citer :

- La modularité des règles simplifie la tâche de mettre à jour la base de connaissance.
- Différentes règles peuvent être ajoutées, supprimées, ou modifiées sans
- Documentation: Un système expert peut fournir une documentation permanente du procédé de décision.
- Largeur: La connaissance des experts humains multiples peut être combinée pour donner à un système plus de largeur qu'une personne simple est susceptible de réaliser.

1.7.6.8. Inconvénients de la méthodologie des règles

- Le programmeur devrait pouvoir fournir directement la connaissance au système.
- Pour mettre à jour la base de règle directement: le travail doit être entrepris par le programmeur.
- La difficulté d'acquisition des règles: la formulation de la connaissance par des règles.
- Les enchaînements infinis: les cycles entre les règles.
- Les contradictions par de nouvelles connaissances: les nouvelles connaissances prévues pour fixer un problème peuvent présenter des contradictions non désirées.

- La modification des règles existantes: en plus de l'enchaînement et des contradictions infinis, les règles additionnelles peuvent résulter en des modifications
- L'inefficacité: les algorithmes de recherche basée sur un modèle de règle ne sont souvent pas efficaces, et nécessitent des optimisations (*Algorithme de Rete*).
- La puissance d'expression de la connaissance par des règles ne s'adapte pas toujours à d'autres modèles de la connaissance.

1.7.7. Agents Mobiles

La réalisation des agents mobiles est un paradigme qui convient surtout pour les systèmes mobiles et distribués [32].

Un agent mobile est un agent logiciel capable de se déplacer d'une machine vers une autre à travers un réseau hétérogène sous son propre contrôle pour être exécuté à distance. Les agents mobiles transportent, à la fois, les données et le contrôle d'exécution. Ils peuvent revenir à son site natal, une fois sa tâche accomplie [33].

Un système d'agent mobile doit supporter les caractéristiques suivantes :

La communication :

Dans un système à base d'agent mobile, l'existence d'un modèle de communication est nécessaire pour que les agents puissent communiquer entre eux, avec des serveurs ou bien avec des humains. En communiquant, les agents peuvent échanger des informations et coordonner leurs activités.

On distingue deux types de communication :

- Communication locale : les agents communiquent entre eux sur le même site.
- Communication à distance : les agents sont situés sur différents sites et communiquent entre eux.

La mobilité *:*

La mobilité d'un agent est le mécanisme utilisé pour le transporter entre les différentes machines à travers le réseau (qui est pour être le World Wide Web). Il décide lui-même de manière autonome, de ses déplacements [33]. La fonction qui contrôle le déplacement d'un agent mobile d'une machine vers une autre s'appelle migration.

Il existe deux types de migration :

- Migration forte : après la migration, l'agent mobile continu son exécution la où il à été interrompu (transmission de l'état d'exécution et les données d'un agent).
- Migration faible : après la migration, l'agent mobile reprend son exécution dès le début (transmission que les données d'un agent).

1.7.8. Agents mobiles pour dispositifs mobiles

Les appareils mobiles sont aujourd'hui très populaires et largement utilisé. Les nouvelles générations de PDA ont plus de mémoire et de puissance de [illegible] Les téléphones mobiles récents ont des capacités de se connecter [illegible]

D'autre part, les applications pour les appareils mobiles semblent encore sous-développées. Bien que de nombreux fabricants de périphériques [illegible] majeurs essayent de pousser le développement d'applications [illegible] progrès réel n'a pu être observé jusqu'à présent. Une des raisons est que les profils de J2ME comme CLDC / MIDP ne permettent pas vraiment le développement d'applications puissantes et sophistiqués.

La raison pour laquelle les vrais agents mobiles ne fonctionnent pas sur Java Connected Limited Device Configuration (CLDC) est surtout le manque de supporter la redéfinition du chargeur de classes de Java et L'API de réflexion. Le profil CLDC est capable d'exécuter que le code pré-vérifié.

MIDP par exemple, est limitée à MIDlets. La plate-forme MIDlet a déjà son propre cycle de vie, avec des contraintes de sécurité et ne supporte pas les environnements ou les services dynamiques.

Toutes les limitations des appareils mobiles devraient motiver le développeur de penser à la manière de contourner ces problèmes et d'écrire des applications plus efficaces.

Le développement des agents mobiles pour les appareils mobiles est un champ de recherche tout à fait nouveau. Cependant, l'intérêt grandit tous les jours, et le nombre de projets qui intéresse au développement d'agents mobiles pour les appareils mobiles augmente.

1.8. Patrons pour la conception des Agents

La plupart des développements de système d'agents jusqu'ici ont été réalisé de manière *ad-hoc*.

seulement utiles dans le domaine des agents, mais également pour tous les développeurs de logiciels qui emploieront les patrons pour concevoir leurs systèmes à l'avenir.

Le problème qui reste à résoudre est de comment décider pour le choix du patron à utiliser. Il est très rare de trouver une argumentation claire dans la littérature. Dans ce mémoire nous allons, par contre, argumenter chaque patron utilisé dans notre framework.

Les patrons de conception sont « des solutions simples et élégantes aux problèmes spécifiques dans la conception de logiciels orientés objets ». Elles capturent la

structure statique et dynamique de ces solutions sous une forme cohérente et facile à appliquer.
Elles contiennent la connaissance et l'expérience qui est à la base de beaucoup de conceptions et d'efforts de recodage des développeurs qui luttent pour réaliser une plus grande réutilisation et flexibilité dans leur logiciel.

Les patrons de conception sont les patrons qui se reproduisent dans le code de programmation ou dans l'architecture des composants de logiciels.

Des patrons pour l'architecture de logiciels sont concernés par la structure globale d'un système logiciel ou d'un sous-système qui est appropriée au domaine du problème et clarifient les intentions du concepteur au sujet de l'organisation du système ou du sous-système [13].

Weiss [35] décrit un groupe *de patrons architecturaux* pour les systèmes de e-commerce basés sur des agents.

Kendall et Malkoun [36] ont utilisé des patrons généralement connus, tels que l'Adapter, Mediator, Broker, Strategy**,** et l'Observer et des patrons spécialisés pour agents comme les patrons Active Object, Interpreter, Reasoner et le Negotiator.
Ils suggèrent un modèle d'architecture à sept couches pour les agents, et un ensemble de patrons pour chacune de ces couches. Les sept couches sont: la mobilité, la traduction, la collaboration, les actions, le raisonnement, la croyance et le sensoriel.

Lind [37] a introduit un certain nombre des patrons d'agents dans la technologie du logiciel orientée agents.Il utilise une collection de vues comme catégories de patrons d'agents dans le développement les des applications à base d’agents.

1.9. Conclusion

Nous avons vu dans ce chapitre les concepts théoriques centraux au sujet de la conception des frameworks à base de patrons, et comment ils peuvent être employés sur les dispositifs mobiles. Nous avons vu également quelques concepts de base au sujet des agents et de système multi-agents.

Les systèmes à base d’agents mobiles joueront un rôle très important dans

CHAPITRE 2 :

ETAT DE L'ART

2.1. Introduction

Etant donné l'âge du paradigme agent, les méthodologies de conception d'agents ([13] [38] [39] [40]) n'ont pas encore bien muries. Dans un passé proche, on ne pouvait pas oser donner une définition des concepts agent — ne serait-ce que celui d'agent lui-même — sans lever une certaine polémique.

Les systèmes multi-agents sont encore essentiellement empiriques. Ce sont ces connaissances, cette expérience, que les différents outils et [illegible] plus ou moins de succès.

2.2. Domaine des Agents

Beaucoup de travail a été effectué dans le domaine des systèmes d'agents. On pourrait citer les exemples suivants : Aglets d'IBM [41], Concordia de Mitsubishi [20] ou la plateforme (framework) JADE/LEAP [42]. La plupart d'entre eux ne concernent pas les dispositifs mobiles limités en ressources en tant que plateformes d'accueil pour les agents.

L'apparition de Java a mené au développement de plusieurs systèmes d'agents mobiles, tels qu'Aglets [41] et Voyager[43].

2.3. Les systèmes d'agent mobile existants sur ordinateurs

Les plate-formes de développement des systèmes à base d'agent mobile doivent fournir toutes les fonctionnalités dont les agents mobiles ont besoin, en particulier le modèle de navigation [28]. Pour le cycle de vie, il faut définir les services de créations, de destruction, de démarrage, de suspension, d'arrêt…etc. Le modèle de calcul indique les moyens de calcul de l'agent. Le modèle de sécurité décrit la façon dont les agents ont le droit d'accéder aux ressources de réseau et aux terminaux. Le modèle de communication définit les modes de communication entre agent et entre un agent et une autre entité, comme le réseau. Le modèle de navigation se charge de tous les transports de l'agent entre deux entités de réseau.

Actuellement, plusieurs plate-formes ou systèmes d'agents sur ordinateurs qui supportent la mobilité existent. Parmi eux, il y'a des systèmes qui sont commercialisés et il y'a des systèmes qui restent comme projet de recherche. Nous

2.3.1. AJANTA

Ajanta [xx] est un framework à base de Java pour programmer des applications basées des agents mobiles autonomes.

Ajanta emploie les possibilités de Java tels que la sérialisation, la réflexion, l'appel de méthodes à distance, et un modèle de sécurité basé sur la sérialisation et le chargement dynamique de classes pour mettre en application la mobilité d'agent.

2.3.2. JADE

La plate-forme JADE (Java Agent DEvelopment framework- Lightweight Extensible Agent Platform) [46], est un projet open source sur J2SE très populaire lancé par le Labo de Italia Télécom et l'Université de Parme.

Le framework de développement d'agents en Java (JADE) est un framework logiciel entièrement implémenté dans le langage Java.

2.3.3. MTA

Le *MTA* [47] fournit le service basé sur la localisation et exécute des fonctions basées sur l'information géographique au sujet des points d'intérêt comme des hôtels et des restaurants proches et montre cette information sur une carte interactive. Le MTA fournit des services additionnels qui intéressent et aident les touristes comme un convertisseur de devise, des prévisions météorologiques et des temps à jour en ligne de prière. MTA veut montrer comment la distribution de la logique client/server d'application pourrait s'appliquer sur les dispositifs mobiles dans le développement d'applications mobiles.

L'architecture de MTA suit l'architecture à base du patron *MVC* (Modèle-Vue-Contrôleur).

2.3.4. Bee-gent [48]

Bee-gent est un système développé par Computer & Network Systems Laboratory Corporate Research & Development Center, TOSHIBA Corporation en [illegible] et un système d'agents pur à 100%. Au [illegible] quelque caractéristiques d'agents, Bee-gent a complètement "Agentifié" la communication qui prend une place entre les logiciels. Les applications deviennent des agents [illegible] messages sont portés par les agents. Be-gent permet aux développeurs de construire des systèmes distribués ouverts et flexibles, et est compatible avec Java.

2.3.5. Aglets[41]

Le « Aglet Workbench » d'IBM est sans doute un des systèmes les plus connus. Danny Lange, un des pionniers de la communauté, était à la base de son développement au Japon [90]. Le système est facile à charger du Web et à installer sur tout système supportant Java. Les Aglets sont des objets de Java qui peuvent migrer d'un hôte sur l'Internet à un autre. Un Aglet qui s'exécute sur un hôte peut simplement arrêter son exécution et migrer vers un hôte distant, et reprendre l'exécution dans le nouvel hôte. Quand l'Aglet se déplace, il prend son code source

aussi bien que ses données avec lui. Les Aglets sont peut-être l'une des technologies des agents mobiles la plus utilisée au moment présent.

2.3.6. JATLite

Java Agent Template Lite (*JATLite*) [49] est un ensemble de packages Java qui est développé à L'Université Stanford. JATLite fournit une architecture en couches pour la construction des systèmes multi-agents.

JATLite permet aux utilisateurs de créer rapidement des nouveaux "agents" qui communiquent sur l'Internet.

La couche de base utilise le TCP/IP comme mécanisme de la communication et permet aux développeurs de définir le protocole d'échange du message. La couche KQML permet aux agents d'utiliser KQML comme langage d'échange des messages.

un atout et une faiblesse. Un atout car cela permet de ne pas « verrouiller » le système et de faire cohabiter plusieurs types d'agents ; cet aspect est primordial pour les évolutions futures. Mais cette absence de modèle d'agent constitue une faiblesse car il est de fait nécessaire de développer ses propres agents.

2.3.7. Voyager System

Le Framework Voyager [43], d'ObjectSpace, Inc., est un logiciel d'agents qui est basé sur le (Object Request Broker (ORB)) et développé totalement en Java. Un ORB fournit la capacité de créer des objets sur un système éloigné et invoquer des méthodes sur ces objets.

Les agents Voyager ont une mobilité et une autonomie qui sont fournies dans la classe de base, qui permet à l'agent de migrer lui-même d'un emplacement à un autre.

Le voyageur utilise la sérialisation pour sérialiser l'état de l'agent quand l'agent déplace d'un emplacement à un autre.

Les agents voyageurs ne sont pas intelligents. Ils ne contiennent pas un moteur d'inférence, ni de réseaux neuronaux ou toute autre technologie de l'intelligence artificielle.

2.4. Framework pour Agents mobile sur des dispositifs mobiles

Bien que les dispositifs mobiles deviennent de plus en plus puissants d'année

plateforme d'agents intelligents ou mobiles.

La plupart des plateformes d'agent sont implémentées en Java et exigent beaucoup de mémoire et une unité centrale de traitement puissante afin de s'exécuter efficacement. Ceci signifie qu'il peut être très difficile d'exécuter un système d'agents mobiles sur par exemple à un téléphone mobile supportant Java.

L'intérêt de mettre en application les systèmes d'agents mobiles dispositifs mobiles a augmenté pendant ces récentes années.

Dans la littérature, un nombre réduit de frameworks à base de patrons pour agents ont apparu.

Nous pourrions citer certains frameworks qui ont attiré notre attention, pour des raisons de disponibilité, d'implémentation sous java, ou de leur conception à base de patrons :

2.4.1. JADE-LEAP

LEAP[50] (plateforme extensible légère d'agents) est une plateforme de FIPA[51] qui peut être utilisée pour déployer la diffusion de systèmes multi-agents à travers un réseau hétérogène de dispositifs à câble ou mobiles, s'étendant des

téléphones mobiles aux serveurs d'entreprise [50]. LEAP a développé un nouveau noyau pour JADE qui permet à des agents de JADE de s'exécuter sur de petits dispositifs sans aucune modification, à condition que ces dispositifs offrent des ressources et une capacité de traitement suffisantes.

Le projet LEAP fournit un environnement JADE avec les principales fonctionnalités adaptées pour J2me.

L'environnement JADE-LEAP possède une bonne flexibilité et un système de messagerie intéressant. Il fournit un propre module pour exploiter la connexion sans fil tel que TCP/IP par GSM et GPRS. Pour le moment, la sécurité, la persistance et la stabilité sont moins étudiées.

La version initiale du LEAP possède les caractéristiques fonctionnelles suivantes :

- Elle fonctionne sur des PCs de bureau, sur des PDAs et sur des

fonctionnalités adaptées pour J2me.

L'environnement JADE-LEAP possède une bonne flexibilité et un système de messagerie intéressant. Il fournit un propre module pour exploiter la connexion sans fil tel que TCP/IP par GSM et GPRS. Pour le moment, la sécurité, la persistance et la stabilité sont moins étudiées.

2.4.2. 3APL-M

3APL-M est un framework pour agents incorporant le moteur de Prolog [52].

C'est une plateforme pour construire des applications utilisant le langage Artificial Autonomous Agents Programming Language (3APL) comme une logique permettant les cycles de délibération ainsi que pour la représentation interne de la connaissance.

2.4.3. MDA

L'Environnement Agents pour dispositifs mobiles ou Mobile Device Agent-Environment (MDA) basé sur Java2 Micro Edition (J2ME) Il vise les dispositifs à ressources limitées comme les PDAs ou les téléphones mobiles.

Une des aplications se trouve dans le secteur de la mobilité d'agents. En raison des concepts de sécurité du CLDC, n'importe quel agent prévu pour être utilisé au temps d'exécution doit être présent au temps d'installation dans la plateforme elle-même.

2.4.4. Système D'Agents Mobile Okeanos

Friedrich et al. [33] ont présenté une approche d'un logiciel personnalisé basé sur les agents mobiles pour les dispositifs mobiles. Ils l'étendent par un système d'agents mobiles à déployer sur les PDA.

Une implémentation du système pour agents mobiles Okeanos pour [illegible] mobiles existe mais elle ne concerne que le routage et la mobilité des agents.

[illegible] localement dans Okeanos. Il n'y a [illegible]

Quand un agent mobile veut migrer vers une certaine destination, il est sérialisé et transféré par l'intermédiaire de Java RMI.

2.4.5. AMobe

AMobe [53] est un Framework qui assure une communication client-serveur dans un environnement sans fil. Ce Framework se concentre sur l'implémentation d'applications agent dans lesquelles les agents communiquent par une liaison sans fil tel que GPRS (General Packet Radio Service). Le Framework tourne sur les appareils mobiles qui exigent un système d'exploitation adapté, un environnement de programmation et une plate-forme dédiés pour les agents.

2.4.6. AgentLight

AgentLight est un projet de PhD à l'Université d'Utrecht de la Hollande ; il est adapté pour l'usage limité du mémoire et du CPU [54].

AgentLight utilise *J2se* et *J2me* (MIDP) et implémente un système d'inférence basé sur la logique du premier ordre. Cependant, AgentLight n'a pas les fonctionnalités suffisantes pour être prêt à l'utilisation.

2.4.7. MIA

C'est une approche qui permet aux dispositifs supportant Java moins puissants d'accéder à un système mobile [56].

Le but du projet de Mobile Information Agent (MIA) est de développer un système d'information intelligent, qui met l'information d'importance locale à partir du WWW dans les mains des utilisateurs de mobiles [56]. Le projet MIA utilise une architecture Client/Serveur où l'utilisateur porte le client. Le système est fait pour l'intégration transparente et efficace sans fil à l'Internet (par exemple par à l'aide d'un téléphone mobile capable).

Le système doit être capable de reconnaître où se trouve l'utilisateur. Ceci peut être réalisé soit par un système de positionnement, soit par le fait que le mobile doit être dépisté dans le réseau mobile. Le système MIA supporte actuellement trois types d'utilisateurs [57]:

- un utilisateur mobile équipé d'un PDA, d'un GPS et d'un téléphone mobile;
- un utilisateur mobile équipé d'un téléphone mobile permettant le WAP et en plus un PDA; et
- un utilisateur stationnaire avec un PC et un navigateur réseau standard.

Dans [45], la conception d'un système d'agents mobiles qui fournit à un utilisateur mobile un service personnalisé de recherche documentaire est présentée. Le système incorpore un système d'apprentissage basé sur les règles.

2.4.8. µFIPA-OS

µFIPA-OS provient de l'Université de Helsinki, Finlande. C'est une version minimale de FIPA-OS [58], elle est une plate-forme open-source d'agents qui implémente les caractéristiques de FIPA. Elle tourne sur Java Standard et manipule des gros agents. Cependant, µFIPA-OS était juste un projet dont l'investissement et le développement est s'est arrêté en 2001.

2.4.9. KSACI

KSACI[59], est un outil qui propose une infrastructure pour la communication inter-agents qui s'exécutent sur les appareils portables. KSACI permet aux agents embarqués dans des appareils portables d'échanger de l'information et de la connaissance avec d'autres agents embarqués ou avec des agents sur des hôtes standards en utilisant le langage KQML.

2.5. Conclusion

Nous avons présenté ci-dessus un aperçu sur les principales plates-formes d'agents. Plusieurs autres systèmes existent et plusieurs autres sont en développement. Cela veut dire qu'au fur et à mesure de nouveaux systèmes vont apparaître.

Chacun de ces systèmes d'agents mobiles est basé sur au moins une caractéristique de l'agent (autonomie, communication, et mobilité) mais aucun de ces systèmes n'a de module dédié que pour le raisonnement.

Dans le chapitre suivant nous allons voire une plateforme java qui soit compatible avec une large gamme des petits appareils connectés aux ressources systèmes très limitées, cette plateforme s'appelle J2ME[61].

CHAPITRE 3 :

PROGRAMMATION DES DISPOSITIFS MOBILES SOUS J2ME

3.1 Introduction

L'évolution technologique fait qu'aujourd'hui des téléphones plus puissants, dimensionnent et l'évolution de la téléphonie continue.

En parallèle, l'Internet se développe à grands pas et de plus en plus de personnes veulent accéder et obtenir des informations sur leurs appareils mobiles

Coté programmation, Sun nous a proposé une version allégée de J2SE, adaptée aux appareils de faible puissance appelée J2ME[61].

Dans ce chapitre, nous allons exposer la technologie *J2ME*.

3.2 La diversité de périphériques

Chaque appareil possède des spécificités qui impliquent aux applications de s'adapter à différentes caractéristiques d'affichage ou de pointage. Microsoft et Sun n'ont pas du tout la même approche concernant ce problème.

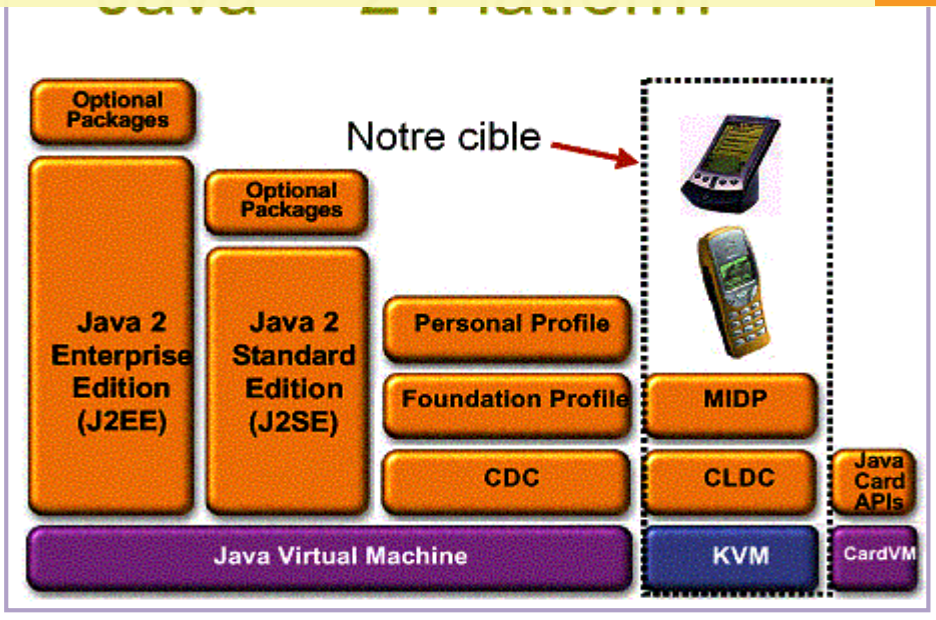


Figure 3-2: Une vue générale des environnements Java.

Dans cette optique, la plate-forme propose deux configurations :

- CDC (Connected Device Configuration) : Spécifie un environnement pour des terminaux connectés de forte capacité tels que les « *set top boxes* », les téléphones à écran, la télévision numérique, …
- CLDC (Connected Limited Device Configuration) : Cible les périphériques à ressources limitées ou faibles tels que les téléphones mobiles, les assistants personnels, ou les périphériques légers sans fil (wireless).

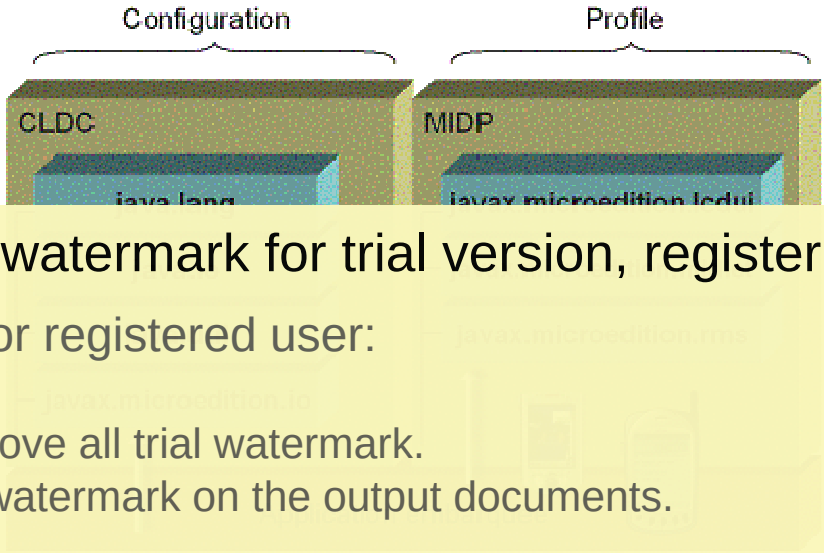


Figure 3-3: Les API de MIDP et CLDC.

3.3.1 MIDP (Mobile Information Device Profile)

MIDP est à la base de l'implémentation des classes liées à un profil donné. On y trouve les méthodes permettant de gérer

- l'affichage,
- la saisie utilisateur, et
- la persistance (base de données).

3.4 Les MIDlets

Les MIDlets sont l'élément principal d'une application Java embarquée.

Les Midlets représentent l'équivalent des Applets et des Servlets pour J2ME avec comme conteneur le téléphone mobile ou l'assistant personnel. Ainsi, en cas de mise à jour d'une application embarquée, un simple téléchargement de code Midlet est nécessaire à partir d'un quelconque serveur.

De cette manière, un programme développé pour un profile donné est en mesure de s'exécuter sur tous les périphériques correspondant à cette famille. C'est aussi une manière de découpler le socle technique des applications puisque le seul lien existant entre les logiciels embarqués et le terminal est l'API J2ME.

Alors, les Midlets sont des programmes utilisant le Framework J2ME et dérivant de la classe ***javax.microedition.midlet.MIDlet.***

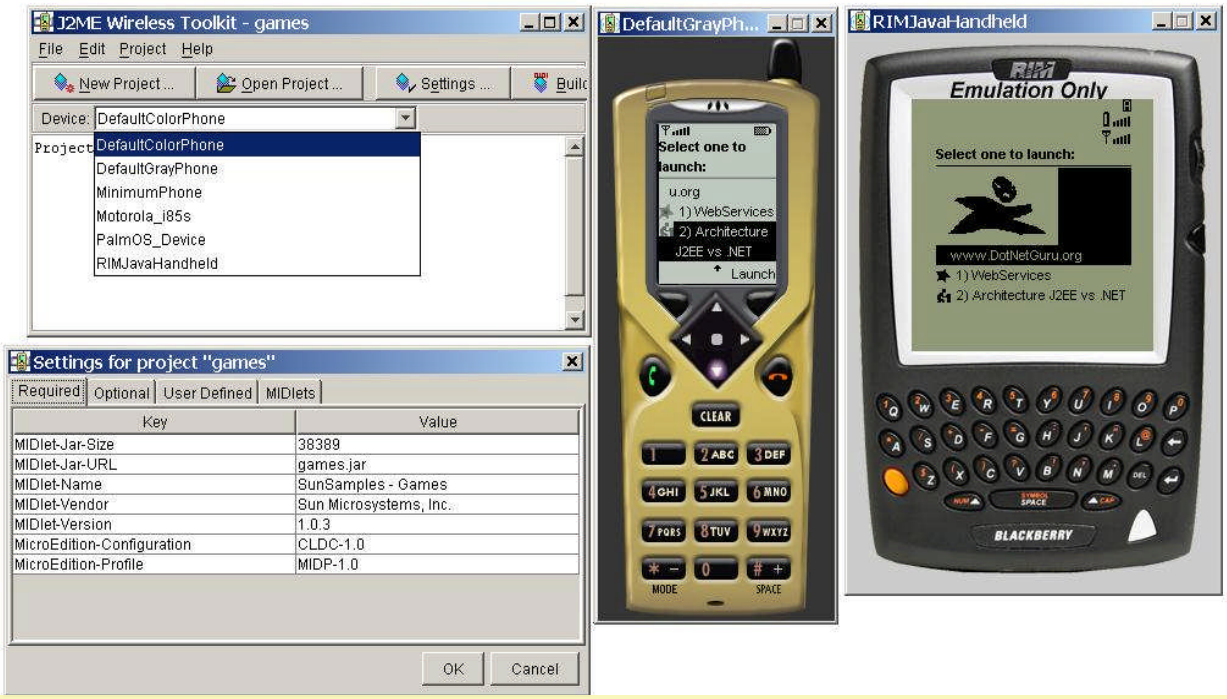
Cette classe est responsable du cycle de vie de la MIDlet en invoquant les méthodes *startApp()*, *destroyApp()* ou *pauseApp()* en fonction du contexte d'exécution. Le

méthodes et objets fournis par le Framework J2ME.

3.5 Application développée avec le J2ME Toolkit

E Toolkit. Tout d'abord, un exécuteur d'applications MIDP, c'est à dire un programme destiné à lancer les exécutables (fichiers jad) créés par le développeur tout en propos d'émulateur allant du **Palm Pilot** classique aux téléphones mobiles **Mo**

Plus de six émulateurs sont pré-installés afin de vous démontrer que la même application développée pour un profile donné pourra s'exécuter sur plusieurs périphériques sans modification préalable.



Java c'est le langage le plus adéquat pour réaliser un système multi-agents grâce à ces points forts qui sont :

- Java est un langage orienté objets dynamique, multi-threads, et portable.
- La réflexibilité,
- La sérialisation des objets, et
- Le chargement dynamique de classes.

Mais bien dommage que ces trois derniers points ne sont pas supportés sous J2ME ce qui rend notre travail plus difficile surtout pour le coté mobilité qui est basé sur la sérialisation et le chargement dynamique de classes.

Malgré que le plus grand nombre des *MIDlet*s actuelles s'inscrive dans le cadre des jeux ou des petits utilitaires tels que les agendas ou convertisseurs d'unités, ce n'est de loin pas le principal intérêt de cette technologie. Avec le standard MIDP 2.0, les opportunités de créer des services mobiles dans des domaines très variés sont grandes.

Ainsi, on peut imaginer des applications de m-commerce (m pour mobile) d'achats en ligne et de consultation de la bourse, d'enchères ou d'informations en temps réel. C'est justement ce genre de domaines qui présente un grand intérêt.

Les agents mobiles sont l'une des solutions qui nous permet de créer ce genre d'application évolué, notre travail démontre comment Java Midro Edition peut aider à construire des systèmes multi-agents puissants et fiables.

CHAPITRE 4 :

CONCEPTION DE L'ARCHITECTURE DE TO

4.1. Introduction

construire des modules architecturaux permettant de réaliser l'agent, en gardant toujours en vue les principes directeurs qui sont la modélisation par pattern et le développement réutilisable. Il s'agit donc de confectionner une architecture visant ces deux buts et de l'instancier dans le contexte de l'application.

La conception de notre agent est basée sur cinq caractéristiques fondamentales. Il peut être considéré comme un système de raisonnement, en visant à déterminer les actions possibles, privilèges et coordination avec l'environnement et les autres agents.

Ces cinq caractéristiques sont:

- les fonctionnalités liées aux capacités d'actions ou le comportement de l'agent,
- les fonctionnalités liées aux capacités de perception, interaction et coopération de l'agent dans l'environnement,
- les fonctionnalités de communication de l'agent avec les autres agents,
- les fonctionnalités de raisonnement interne de l'agent,
- les fonctionnalités de migration ou de mobilité de l'agent d'un environnement à un autre.

l'agent, comme une représentation comportementale de l'agent. Cette architecture est organisée selon le patron *Couches Hiérarchiques* (Layered) dont chaque couche est modélisée au moyen d'un patron.

L'architecture que nous proposons est organisée autour de cinq couches bien séparées selon leur fonctionnalité, la première couche représente la partie active de notre plate-forme (Partie Comportement), une autre couche représente la partie intelligente (Partie Raisonnement), la troisième actes comme un lien entre les agents et l'environnement (Partie Interaction) ; La quatrième couche concerne la communication entre les agentes de la plate-forme (Partie Communication) ; La cinquième couche traite la gestion de la mobilité ou la migration des agents (Partie Mobilité).

En général un système organisé selon le patron Couches Hiérarchiques sera composé des couches qui sont fortement couplées en utilisant des pointeurs de liaison dans chaque couche (Figure 4-1).

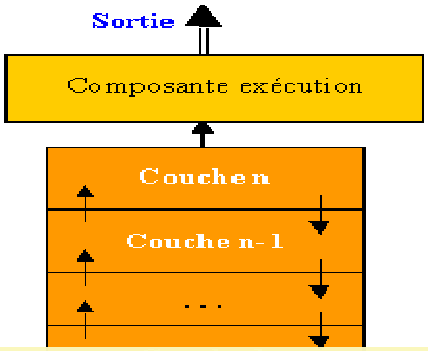


Figure 4-1: Le Système à Couches

Dans notre conception nous avons proposé une nouvelle archi
patron Couches Hiérarchiques, dans laquelle les couches seront faiblement liées, et cela en remplaçant les pointeurs par une technique de patron Observer, ainsi dans notre architecture, chaque couche et un Observer de la couche adjacente qui rend le couplage très faible entre les couches de l'architecture principale du système *To* (Figure 4-2).

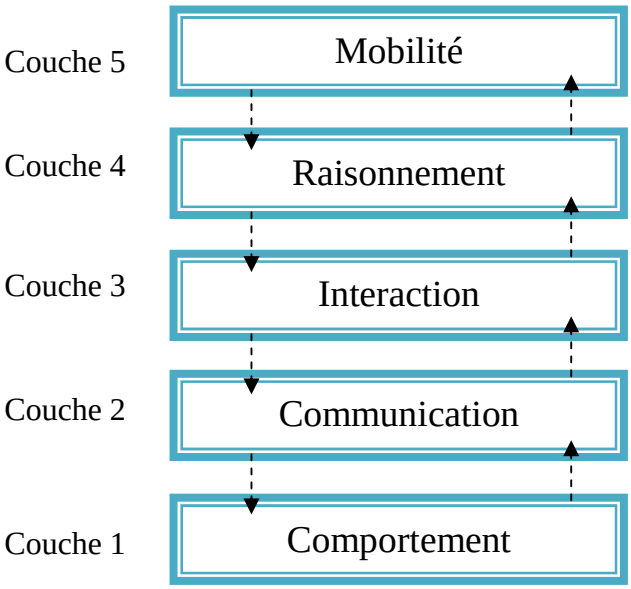


Figure 4-2: Architecture en couche de la plate-forme de l'agent To

patrons Module, Type de données abstrait, Couches Hiérarchiques, et Observer par exemple ont une application très étendue par rapport aux patrons Proxy ou Itérateur qui eux restent applicables seulement dans des situations précises.

Dans ce que suit, nous allons décrire la conception de chaque composant ou module de notre framework.

4.3. Conception de L'Architecture

Notre architecture est décomposée en cinq couches, dont chacune représente une caractéristique de l'agent *To*, correspondant à un module architectural de notre plate-forme.

C'est un défi pour l'utilisateur de patrons de décider quels sont les patrons qu'il devrait utiliser dans la conception.

L'implication est que l'utilisateur du patron doit contrôler le contenu de chaque patron avant de prendre la décision de l'utiliser. Les auteurs de patrons tendent à discuter les avantages et les responsabilités d'appliquer leur patron, qui sont identifiées comme *les conséquences* dans un certain format de patrons. En lisant les conséquences, l'utilisateur du patron peut déterminer subjectivement à travers l'étude de ces patrons, pour choisir celui dont il perçoit le plus de mérite. Mais, le nombre de candidats possibles du patron proposé à l'attention de l'utilisateur de modèle, pourrait rendre cette approche peut pratique.

Le défi de raffiner le nombre de patrons candidats à un nombre réduit pour le

Les agents doivent réellement exécuter des fonctions spécifiques, ou méthodes, accéder à l'environnement local et contacter d'autres agents en utilisant des mécanismes de la coordination.

Le modèle de traitement utilise par conséquent les méthodes fournies par le système agent pour déterminer comment une tâche spécifique peut être implémentée. Habituellement, quand on implémente un agent, une question clé est comment permettre à l'agent de prendre des décisions basées sur la vue courante sur son environnement et de son état mental et structurel.

Le problème reste alors comment choisir le comportement à exécuter. Pour un but de généralité, modularité et réutilisation c'est mieux de garder le comportement

externe des objets à la base des agents eux-mêmes. Cela réduit le couplage, et donc encapsule d'une manière plus efficace le comportement.

Pour fournir une grande flexibilité à l'agent, *To* améliore un Objet au sens de l'orienté objets.

Donc, un *To* peut exécuter, dans l'environnement, toute méthode fournie par l'objet amélioré, il n'est donc pas restreint seulement à quelques actions comme le sont les agents standards.

L'innovation dans *To* est de s'octroyer le comportement de l'objet tout en incorporant à l'agent un système de raisonnement pour le rendre un agent intelligent.

quand l'agent *To* décore un objet, il hérite et décore toutes ses méthodes, et ainsi l'appel de chaque méthode devient négociable selon l'état de l'agent.

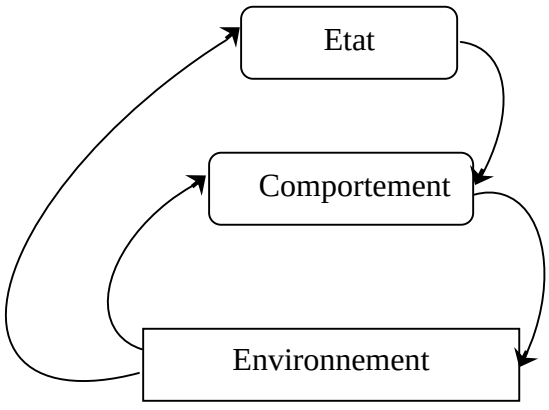


Figure 4 3: le Comportement de To et son Etat

4.3.2. L'environnement de l'agent (Interaction)

Nous allons nous intéresser ici, à modéliser le contexte d'interaction des agents, c'est-à-dire il s'agit ici de caractériser architecturalement l'environnement dans lequel évolue l'agent.

L'environnement est tout ce qui met en rapport l'agent avec le monde dans lequel il évolue, c'est-à-dire ce tout ce qu'il peut percevoir, modifier ou interagir avec (représentation du monde).

L'interaction permet aux agents de:

- coordonner leurs actions et comportement, une propriété d'un SMA avec des agents agissant dans un environnement commun**,**
- essayer de modifier les états des autres agents.

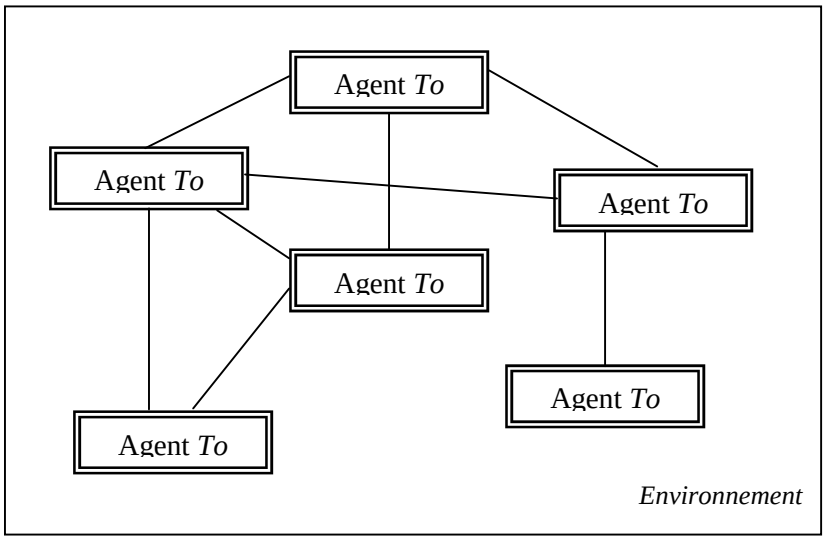


Figure 4-4: L'interaction socio-environnement

Les concepteurs de systèmes qui vont utiliser *To* doivent fournir les capacités de chaque *To* individuel sur comment il va vivre avec les autres *To*s de l'environnement. L'approche de *To* permet au concepteur des systèmes multi-agent d'implémenter le système en utilisant des *To* multiples, avec chaque *To* ayant son propre état mental et intelligence pour une tâche particulière. Tous ces *To*s doivent réagir ensembles pour accomplir leur comportement propre basé sur leurs buts communs prédéfinis.

L'environnement offre une infrastructure où les interactions se déroulent.

Cette infrastructure comprend :

- les protocoles de communication permettent aux agents d'échanger et comprendre les messages, et
- les protocoles d'interaction permettent aux agents les échanges

Pour modéliser la structure de l'agent *To* il faudrait prévoir un modèle de l'environnement.

Les caractéristiques de l'environnement vont influer sur la façon dont on conçoit un agent car il faut tenir compte de l'évolution de l'environnement, de la capacité de l'agent de saisir cette évolution et de sa capacité à décider en conséquence.

Alors que dans la littérature le modèle le plus utilisé pour modéliser l'interaction entre les agents de l'environnement est celui du BlackBoard (tableau noir), de notre part, nous avons de nouveau encore une fois préféré utiliser l'Observer, pour les mêmes raisons de réutilisabilité citées précédemment.

4.3.3. Système de communication de l'agent

La communication permet aux agents de vivre leur vie communautaire.

To vit en commun avec d'autre *To*s, et va avoir besoin de communiquer pour accomplir une tâche partagée.

Une architecture du système de la communication de l'agent devrait fournir un environnement qui assure la communication entre les agents.

Dans la littérature, les agents disposent principalement de deux méthodes de coopération, une directe et l'autre indirecte. La méthode indirecte utilise le service de tableau noir où un agent peut trouver/déposer des informations et la méthode directe est mise en œuvre grâce à l'utilisation d'un service d'annuaire local qui permet de mettre en place un schéma de type Client/Serveur entre deux agents.

Alors que le modèle le plus utilisé pour modéliser la communication entre les agents, dans un même environnement, est celui du tableau noir [14], de notre part, pour modéliser le service de localisation dans To nous avons de nouveau opté pour une méthode basée sur le patron Observer qui est un modèle spécial du Subscript-Publish (Abonnement/Publication).

Le système de communication de l'agent *To* est appelé World.

Ce World est donc conçu à l'aide de patron Observer.

Ce patron nous permet d'avoir un système fortement découplé, donc bien plus intéressant que l'approche qui consiste à utiliser une boîte à lettres, à la manière de la poste restante où sont stockés les messages à destination d'un agent qui en prend connaissance périodiquement.

La différence réside dans le fait que dans notre approche la notification du destinataire est faite par la boîte aux lettres elle-même, au moment de la réception d'un message.

L'avantage d'utiliser l'*Observer* est son couplage faible qui nous donne des possibilités d'une véritable réutilisation. Les agents ne sont pas obligés de connaître

leur correspondant, et peuvent, d'une manière plus facile, assurer une communication anonyme.

4.3.3.1. Le Message

Nous avons choisi l'échange des messages comme une méthode de communication pour les raisons suivantes.

Les avantages de la communication basée sur l'échange des messages sont:

- La communication entre les agents (partage de la connaissance) doit être encapsulée comme ne faisant pas partie du langage d'implémentation,
- un seul type de message doit être compris par tous les types d'agents (i.e. une façon standard de communiquer),
- le message doit être de taille minimal pour réduire la charge du réseau,
- le message peut être de formats différents.

Si un objet envoie un message à un autre objet, le receveur n'a aucun contrôle sur l'exécution de la méthode correspondante; la méthode va toujours s'exécuter. Ce contrôle est fourni par le langage de base, et non pas par l'objet. Avec les agents, nous voyons la communication comme une demande (une requête) par un agent à un autre agent. Dans le cas où l'agent récepteur est en train d'exécuter une certaine action à la réception de la requête, cette dernière va rester à sa disposition sous son contrôle.

Nous avons jugé utile d'abstraire un message par un message GMessage général, dont trois autres classes de messages vont découler :

- les messages *KMessage* demandant au destinataire l'exécution d'une de ses méthodes,
- les messages *DMessage* pour la destruction du *To* destinataire,
- les messages *AMessage* pour la migration d'un *To* vers un environnement hôte.

Un message *GMessage* est conçu comme un message qui pourrait migrer.

Les messages *KMessage* échangés entre les agents pour exécuter une méthode comportent:

- les identités de l'émetteur (source du message), et
- celle du receveur (destination) du message,
- la méthode pour exécuter par l'agent receveur, et
- les arguments de la méthode.

L'agent *To* réagit à un *KMessage* avec son propre comportement déterminé après que le message soit reçu.

Figure 4-5: les Classes Message de TO

4.3.3.2. Traitement des Messages

Chaque agent *To* a un conteneur pour stocker les messages qu'il reçoit.

Les messages sont traités de manière asynchrone et sont utilisés pour activer des capacités ou exécuter un comportement spécifique de l'agent.

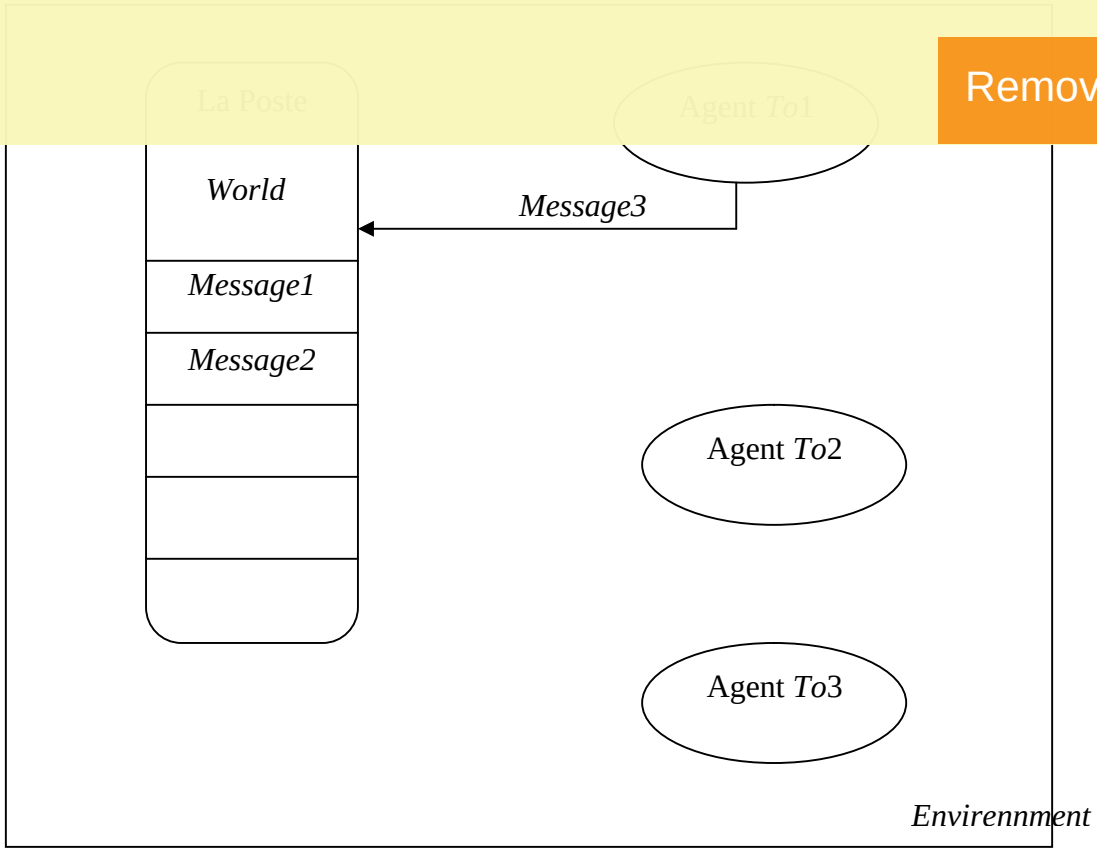
Cette méthode de traitement a été conçue au moyen du patron *Command* de telle sorte qu'on pourrait avoir plusieurs sortes de méthodes de traitement de messages, comme par exemple selon l'ordre d'arrivée des messages, comme:

- l'ordre premier arrivé, dernier traité, *FILO*,
- l'ordre premier arrivé, premier traité, *FIFO*, ou
- un ordre aléatoire, le traitement se fait selon une certaine stratégie donnée.

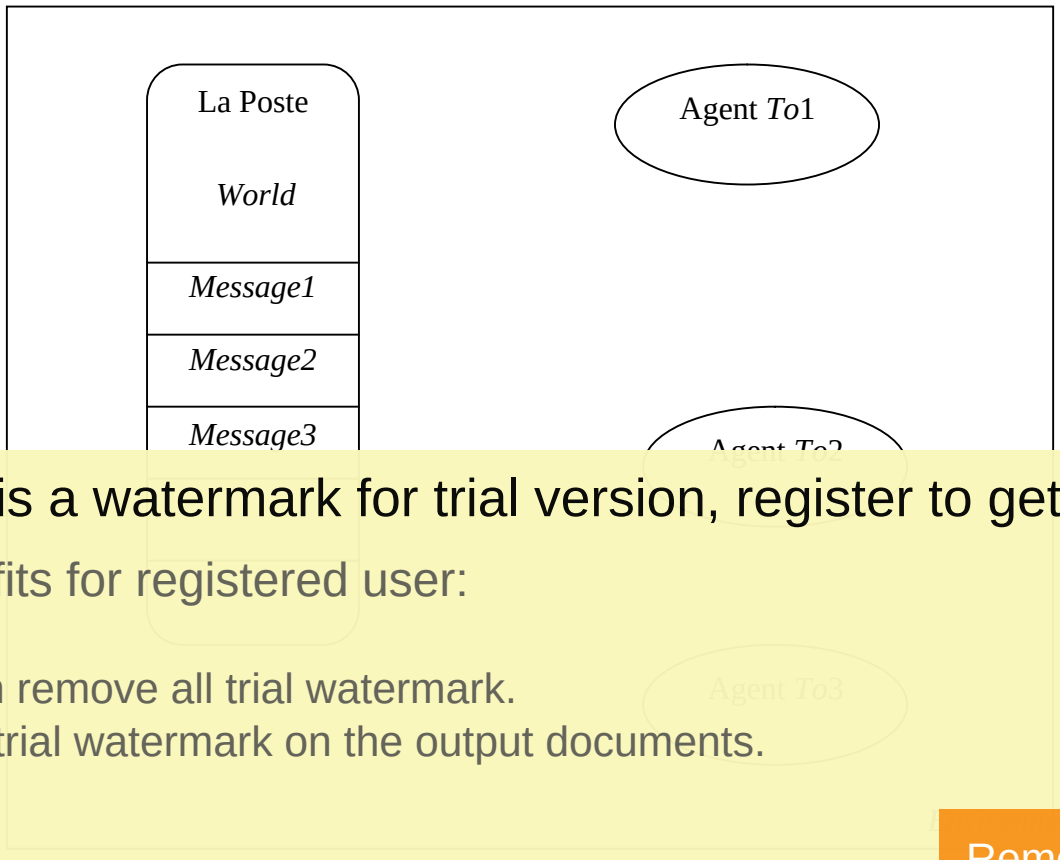
Pour résumer :

Nous allons donner un exemple d'échange de message entre deux agents.

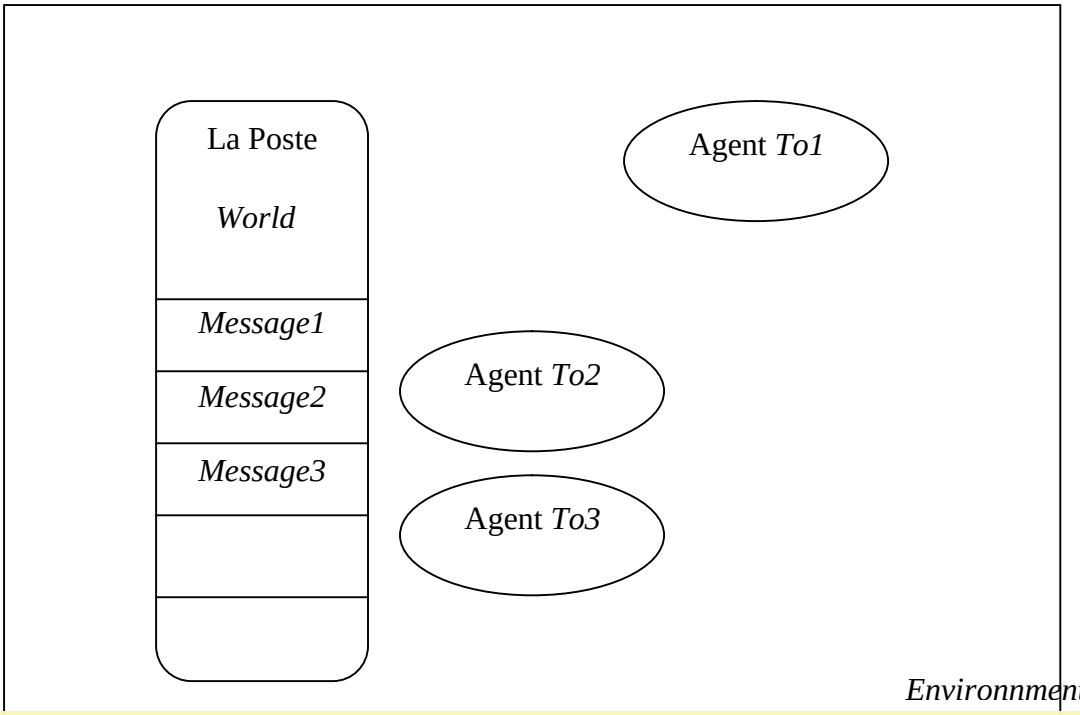
Dans notre exemple l'agent *To1* veut envoyer un message *Message3* à l'agent *To2*.

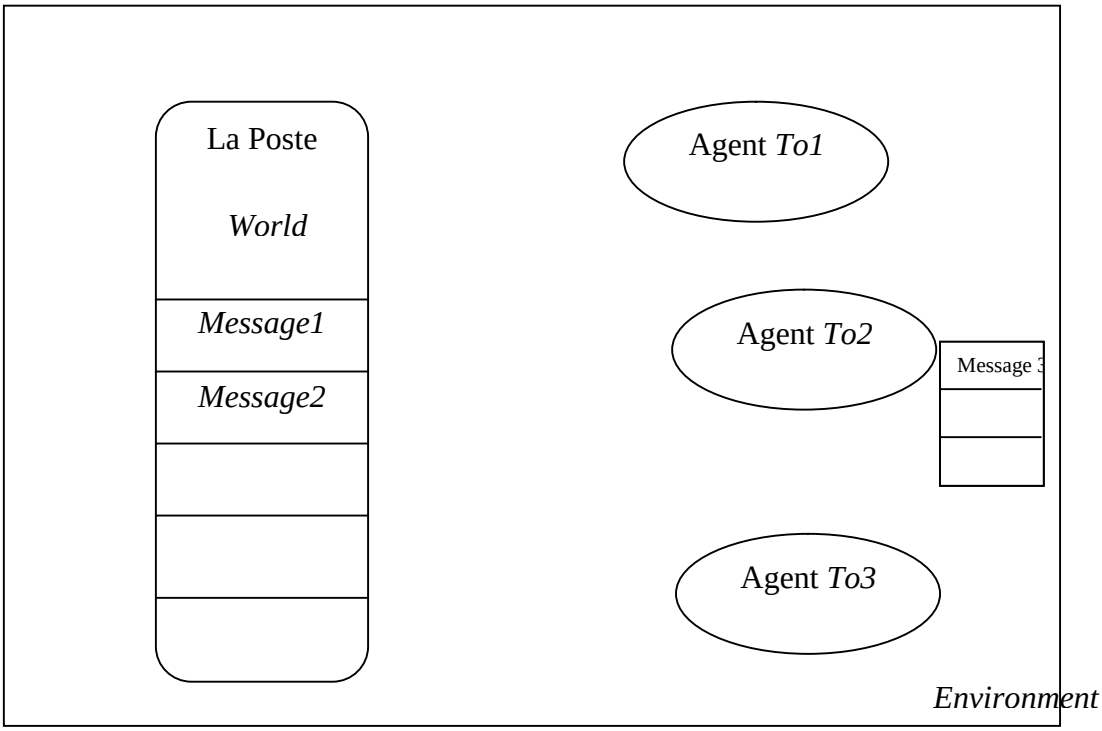


Dans la première étape l'agent To1 envoi le message vers le World (La poste) pour que cette dernière s'occupe de délivrer ce message.



Quand un message à destination d'un certain *To* arrive à la poste, elle le stocke chez elle et notifie tous les autres agents *To* qu'il y a un message qui vient d'arriver, ces derniers vont automatiquement comprendre qu'ils sont invité à voir chacun s'il lui est destiné. Pour concevoir cette solution nous avons utilisé le patron *Observer*, c'est-à-dire nous avons conçu les *To*s comme des *Observer*s de *World.*





En peut s'apercevoir que l'envoi des messages est conçu de manière asynchrone.

L'avantage du traitement de message asynchrone provient du fait que lorsque l'agent destinataire est occupé, l'agent émetteur n'attend pas que le receveur soit libre pour lui envoyer le message.

Mais cette solution a un inconvénient lorsque le *World* comporte un nombre important d'agents, alors il faudrait que tous ces agents aillent vérifier la liste des messages quand un certain message arrive au *World*, ce qui va alourdir le module de communication.

4.3.4. Système de Raisonnement de L'Agent

Les individus ont des sens, une croyance, un raisonnement, et des actions. Une fois qu'on les met face à un problème ou on leur demande d'accomplir une tâche, ils raisonnent au sujet de leurs modèles (habituellement pas d'une façon hiérarchique) pour déterminer une action à entreprendre.

Nous allons implémenter le chaînage en avants pour traité les règles Si-Alors, on va ajouter aussi la négation pour notre logique.

Nous définissons brièvement Les objets qui concernent le module de raisonnement :

La mémoire de travail Base De Faits (FactsBase) dans laquelle les Faits (Facts) sont stockés, et la Base De Règles (RulesBase) dans laquelle les Règles

(Rules) valides sont incorporées, sont les deux éléments importants utilisés par le moteur d'inférence.

Le *Fait* est un objet Java, et il est simplement inséré à la *Base De Faits* quand la méthode assert() a été invoquée.

Une fois que le moteur d'inférence a été activé, toutes les règles valides qui sont stockées dans la base des règles sont exécutées (fired). En d'autres termes, quand les conditions d'une règle deviennent vraies, la partie *action* de la règle sera exécutée.

Le comportement d'un d'agent peut être décrit par un ensemble de règles, qui encapsule de plus quelques connaissances du monde extérieur.

L'état de l'agent peut influer sur le système de raisonnement de cet agent et vis versa.

Pour l'instanciation de notre framework nous avons choisi d'instancier la partie système de raisonnement par un système expert à base des règles mais en laissant la possibilité aux développeurs d'implémenter leur propre système de raisonnement.

Pour concevoir cette possibilité de généricité du système de raisonnement nous avons utilisé le pattern Command.

Dans l'implémentation du patron Command dans l'agent To, on [illegible] référence sur une interface, et c'est cette dernière que doit [illegible] développeurs pour attacher leur propre système de raisonnement, alors que l'agent To lui est capable d'utiliser n'importe quel système de raisonnement proposé.

Notre système de raisonnement actuel est un système expert à base des règles, une règle à des hypothèses et des conclusions, ces règles est de type :

Si hypothèse$_1$ **et** hypothèse$_2$ **et** … **et** hypothèse$_n$

Alors conclusion$_1$ **et** conclusion$_2$ **et** … **et** conclusion$_m$

Pour résoudre le problème des règles avec des opérateurs de disjonction **OU** , nous avons remplacé ces dernières par des règles **ET** comme suit :

- A ou B → C = A →C , B→C.
- A→B ou C = A→B, A→C.
- A ou B →C ou D = A→C, B→C, A→D, B→D.

Ces règles sont appliquées sur une base de faits, cette base est un ensemble dynamique de faits de telle sorte que nous pourrions lui ajouter, ou supprimer des faits.

Le fait c'est un objet java que nous pouvons manipuler facilement, il comporte un nom et une valeur de vérité, quand un *fait* est ajoutée a la base des faits le système expert lance automatiquement le moteur d'inférence, et cela est conçu grâce à l'utilisation du patron *Observer/Observable*.

Le fait est un *Observable* de la base de faits, ainsi à chaque fois qu'un fait est ajouté au système expert, il sera inséré automatiquement à la base des faits.

De même, la base de faits et la base de règles sont des *Observable*s pour le système expert, alors pour chaque changement au niveau de la base des règles ou de la base des faits le système expert lance automatiquement le moteur d'inférence.

4.4. Implémentation du Framework TO

4.4.1. L'environnement

La conception de notre Framework repose sur une architecture par couches dont chaque couche représente un module de l'agent.

Ce qu'il y'a d'intéressant dans notre conception, c'est que l'architecture générale par couches et les couches elles-mêmes sont modélisés à l'aide de patrons.

Tout d'abord il fallait assurer l'unicité de l'*Environnent*, nous allons utiliser le patron *Singleton*,

Le But du patron *Singleton* est de créer une seule instance d'un objet. Pour ce [illegible] de créer l'environnement s'il n'existe pas, ou de renvoyer une référence sur cet *Environnent* là dans le cas où il existe.


```
public static synchronized Environment getInstance(){
        if (instance == null)
                instance = new Environment();
        return instance;
    }
private Environment() {

}
```

Cet *Environnement* doit arbitrer tous les agents qu'ils l'appartiennent pour cela il doit être un observable pour ces agents, alors que ces derniers sont des observateurs de leur *Environnement*.

Malheureusement observer/observable n'est pas implémenté en J2ME, ce qui nous exige de les implémenter. Voilà le code pour l'interface *Observer :*

```
public interface Observer extends Linkable{

        public void update(Observable observable , Object arg );
}
```

Et pour la classe l'*Observable :*

```
public class Observable {

    public synchronized void addObserver(Observer observer){
        if(!hasChanged()){
            Linkable head=list.getHead();
            if(head==null) return;
            for(Linkable p=head; p!=null; p=p.getNext())
                ((Observer)p).update(this,null);
        }
    }
}
```


Le diagramme suivant représente l'utilisation du patron *Observer/Observable* pour la conception de l'environnement.

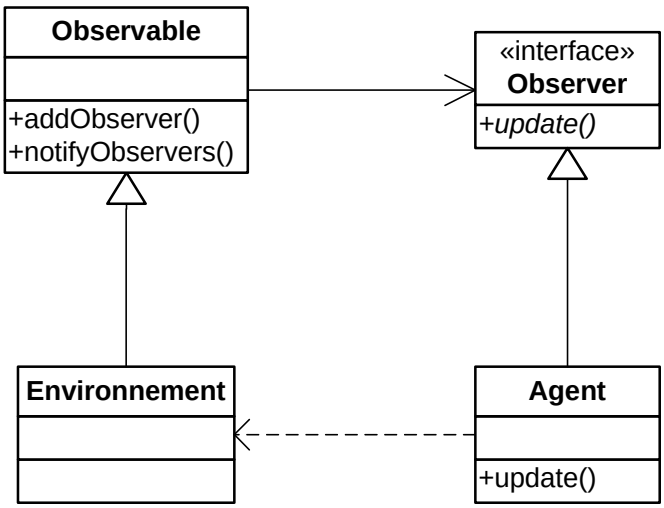


Figure 4-6: Diagramme de classe pour la relation Environnement-Agent

A la création d'un nouvel Agent, l'*Environnement* ajoute ce nouvel agent à sa

A chaque fois que l'Environnement subit un changement, il avertit tous ses agents, de telle façon que ces agents restent toujours informés de l'état actuel de l'Environnement.

4.4.2. La Communication

Un autre module qui assure la communication Inter-Agent c'est le World (on l'appelle aussi la Poste), ce World est un singleton et un *Observable* pour les agents qui appartiennent à ce World, alors il est développé de même façon que l'environnement.

Pour que ce World implémente la communication, il a une liste pour stocker les messages reçues, quand le World reçoit un nouveau message il avertie tous ces Observers (les agents) pour qu'ils peuvent retirer les messages qui les concernent.

Chaque agent doit avoir une liste waitingMessage pour stocker les messages reçus et les traiter ultérieurement.

Pour rendre l'agent capable de communiquer il implémente l'interface Messageable. L'interface Messageable a sendMessage( ) comme une méthode.

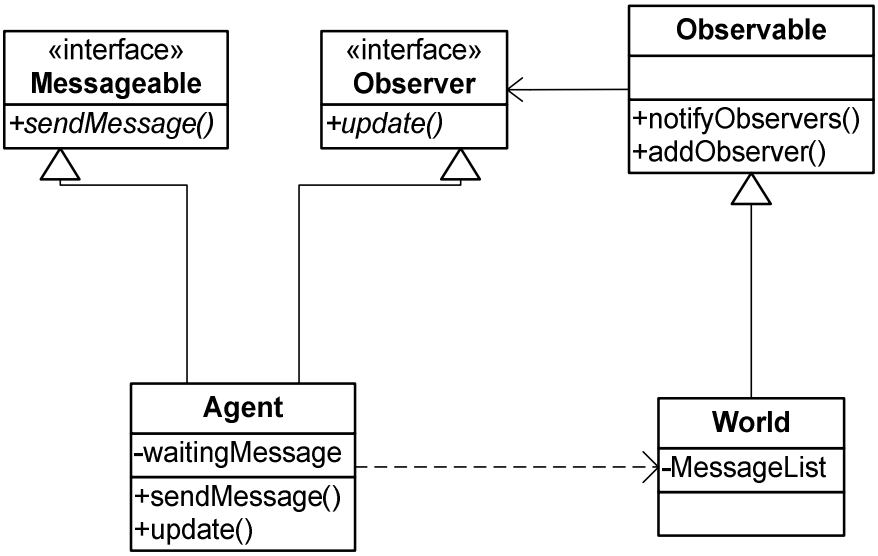


Figure 4-7: Diagramme de classe pour la Communication

public byte[] persist() throws IOException;

public void resurrect(byte[] data) throws IOException;

}

La méthode persist() est pour sérialiser un objet en tableau de byte, et la fonction resurrect() est pour le travail inverse, qui veut dire qu'elle reconstruit un objet depuis un tableau de byte.

L'interface Migratable a migrate() comme méthode qui est définie par les classes qui implémentent cette interface.

Nous avons trois types de message :

- La classe KMessage, elle comporte une KMethode pour demander au destinataire l'exécution de cette KMethode,
- la classe DMessage pour la destruction du destinataire,
- la classe AMessage pour créer un nouvel agent dans un environnement distant.

Le diagramme suivant représente les différentes classes pour les messages :

Figure 4-8: Diagramme de classe pour Message

Comme l'agent a une liste d'attente des messages reçus il doit avoir une méthode de traitement de ces messages.

Cette méthode de traitement a été conçue au moyen du pattern Commande de tel sort qu'on peut avoir plusieurs sortes de méthodes de traitement de messages, et cela selon l'ordre de traitement.

Pour permettre aux développeurs d'implémenter leurs propres méthodes de traitement, nous avons utilisé le pattern Command.

Nous avons implémenté trois méthodes prêtes à utiliser FIFO, FILO et Random.

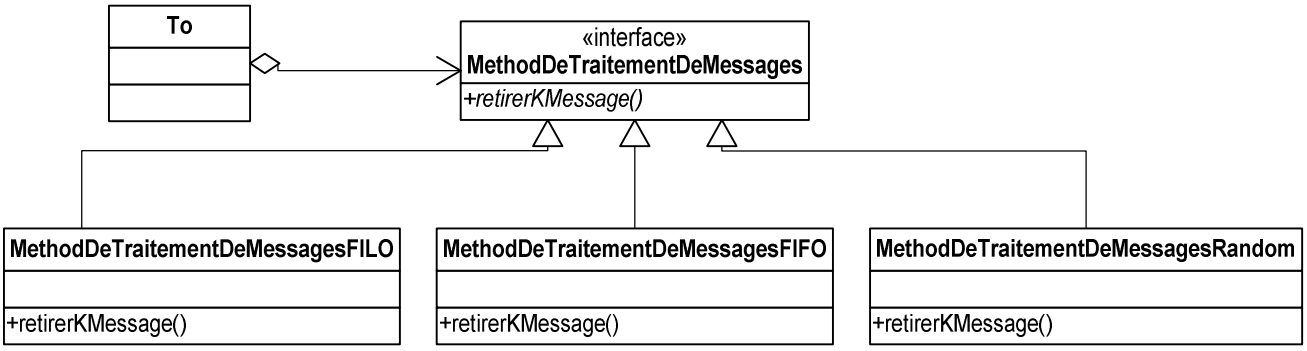


Figure 4-9: Diagramme e classe pour Méthode de traitement des messages

D'autre part, il y a seulement peu de publications décrivant l'exécution réelle d'un système d'agents mobiles pour les dispositifs mobiles.

4.4.3.1. Conception de la Mobilité

La classe MobileAgent est une classe abstraite qui hérite de la classe Agent et implémente l'interface Migratable. Le procédure de migration de l'agent doit développer dans la fonction migrate().

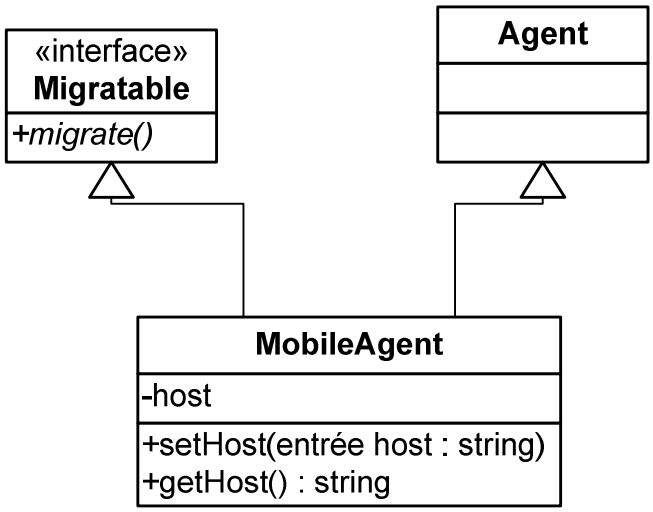


Figure 4-10: Diagramme de classe pour MobileAgent

La partie serveur c'est la partie principale pour la mobilité, elle est modélisée

Notre serveur d'agent MultiToServer a un proxy MultiToServerProxy. Ces deux classes implémentent l'interface MultiToServerInterface. Le MultiToServerProxy s'occupe de contrôler les connexions.

Le MultiToServer utilise des Sockets TCP comme un protocole de communication mais comme nous avons déjà dit l'implémentation d… doit être en Thread, pour cela le MultiToServer implémente au… Runnable.

Le MultiToServer a aussi un ServerConnection pour chaque requête de connexion, ce ServerConnection gère les connexions de tel façon il reçoit un message, ce message peut être un KMessage, un DMessage ou un AMessage, s'il est KMessage il sera envoyé directement au monde, s'il est un AMessage ou DMessage ils seront traités dans la fonction servicing( ).

Le diagramme suivant représente les différentes classes utilisées dans l'implémentation de la mobilité coté serveur de To.

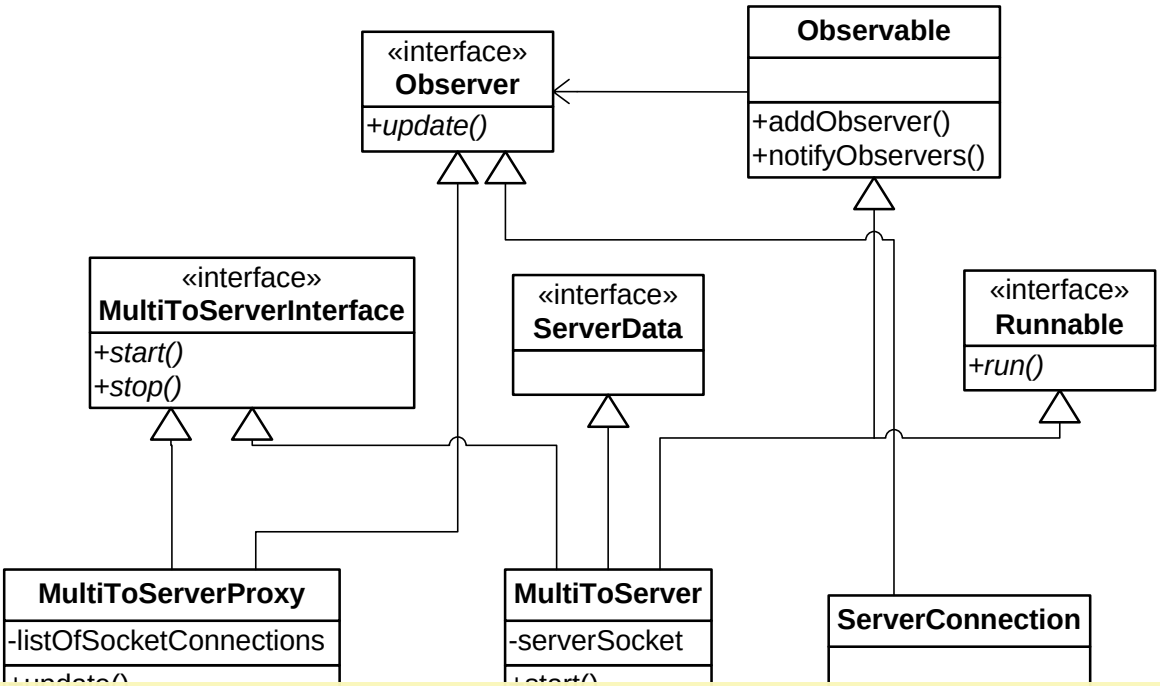


Figure 4-11: Diagramme de classe pour le serveur

4.4.4. Le Système de Raisonnement

Nous pensons que le système de raisonnement est le point fort dans la conception du framework To.

To a un système de raisonnement qui est basé sur un système d'inférence mais il donne toujours aux développeurs qui utilisent l'agent To la possibilité pour implémenter un système de raisonnement propre, ce système doit implémenter l'interface Thinkable pour être intégrer facilement au Framework To.

Le pattern Strategy nous permet de modéliser cette partie.

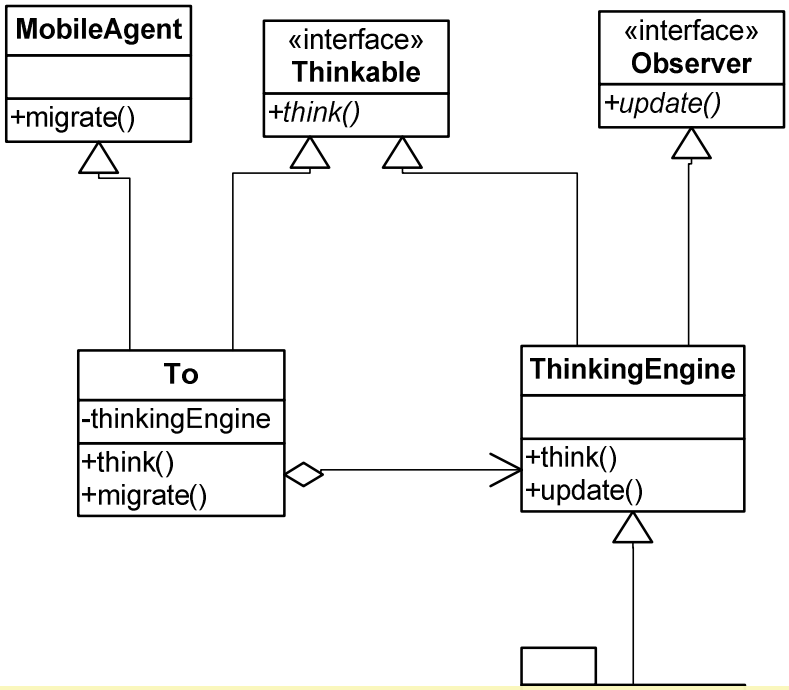


Figure 4-12: Diagramme de classe pour le Raisonnement

Alors notre agent To a un Thinkable avec une fonction Think(), l'appel de cette méthode lance le système de raisonnement.

Nous avons développé un système d'inférence à base de règles, c'est un système expert (ExpertSystem) qui utilise le chaînage en avant comme une méthode d'inférence.

La classe ExpertSystem doit hériter de la classe ThinkingEngine et implémente la méthode think(), elle a une basse des règles et une base des faits, la base des règles (RulesBase) est une liste des règles (Rules) et la base des faits (FactsBase) est une liste des faits (Fact). L'appelle de think() lance la fonction forwardLinking( ) qui implémente le chaînage en avant.

Le diagramme de classe de ce système expert soit :

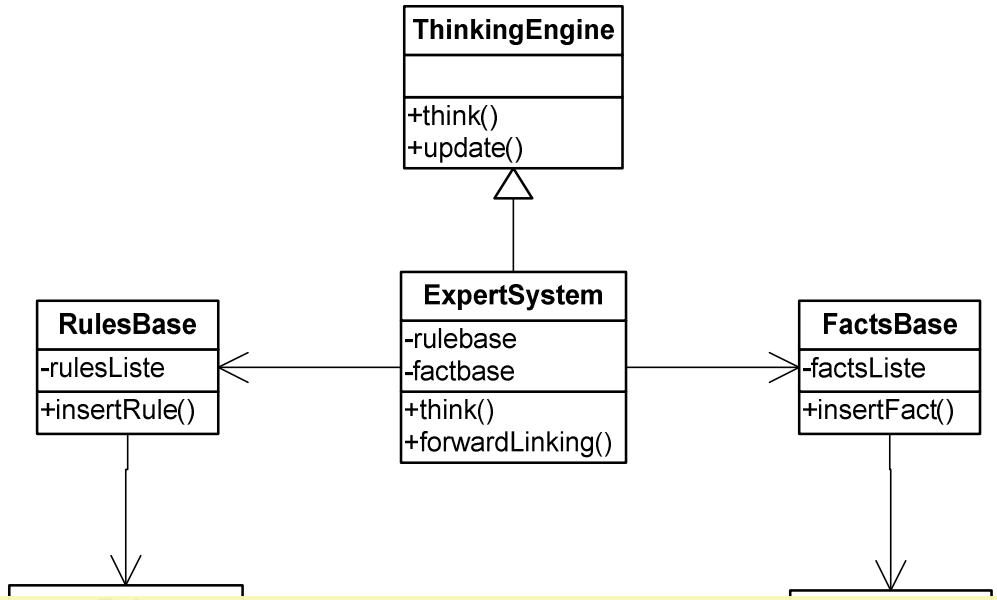


Figure 4.13: Diagramme de classe pour le Système Expert

4.4.5. Le Comportement

L'agent To améliore un objet en lui ajoutant un état et un système de raisonnement, mais il garde toujours le comportement de cet objet, pour le faire nous avons utilisé le pattern Decorator de telle manière que l'agent garde toujours les méthodes de l'objet mais il ajoute un raisonnement sur l'exécution de ces méthodes.

Alors, To a une liste des états ToState() qui fait corresponde un état à chaque méthode, alors que l'exécution de ces méthodes est ce fait selon l'état de l'agent.

Dans le chapitre suivant (Application), on peut mieux comprendre ce module.

4.5. Conclusion

Dans ce chapitre nous avons présenté le processus suivi pour d'abord découvrir les principales caractéristiques d'un agent, ensuite comment nous avons procédé dans la phase de conception en utilisant un patron architectural pour la conception globale du framework, et comment nous avons choisi les patrons les plus adaptés pour les incorporer en couches à notre squelette architectural, pour la description des caractéristiques de base d'un agent, en ayant en tête un constant souci de réutilisabilité du logiciel.

Pour montrer la faisabilité de notre framework nous allons présenter un test de notre framework dans le chapitre suivant.

CHAPITRE 5:

APPLICATION

5.1. Introduction

L'application choisie a pour but principal de prouver la réalisabilité d'une application concrète de notre framework. Dans ce chapitre nous allons montrer [illegible] démontrer la faisabilité de notre travail, dans un domaine d'application réduit.

Notre Framework consiste en un logiciel pour la création de s[illegible] agents. Nous l'avons conçu de telle manière qu'il soit le plus générique [illegible] possible afin qu'il puisse servir dans des domaines variés, grâce aux faits suivants :

- On utilise la programmation orienté objet,
- il est conçu à base des couches de composants (qui modélisent les caractéristiques de l'agent) au moyen du patron layered à base d'Observer,
- est un Framework,
- chaque module est conçu en utilisant des patrons,
- en principe n'importe quel objet pourrait devenir un agent tout en gardant son comportement d'objet propre.

Au début, pour créer le monde (la poste de messages) qui est un singleton nous utilisons la méthode statique World.getInstance(), et de c'est de la même manière pour l'environnement, en appelant cette fois ci Environment.getInstance().

Une fois le monde et l'environnement sont créés, l'utilisateur du Framework n'à qu'a créer ces agents To en héritant de la class To.

5.3. Test sur l'objet Man

Dans notre test, nous avons créé un agent ToMan, c'est à dire qu'on a transformé un Man en un agent ToMan.

5.3.1. Principe

Le principe est de décorer un objet Man et de le rendre un agent TO.

Notre classe *Man* est définie comme suit:

```
public class Man implements Maneable {
    public void eat(Integer number, String thingToEat) {
    ...
    }
...
}
```

Donc on voit bien qu'elle a comme comportement celui de l'interface Maneable.

Nous allons le décorer par un agent ToMan, l'agent ToMan implémente l'interface Maneable et hérite de la class To. Par ce fait, ToMan devient un agent, il hérite de toutes les caractéristiques d'un agent; et il implémente en plus l'interface Maneable, pour que cet agent puisse avoir un comportement de base de l'objet Man.

Tout cela est conçu au moyen du patron Decorator, qu'on utilise pour décorer un Man en vue de lui rattacher le côté raisonnement « exécution de comportement raisonné ou intelligent ».

```
public class ToMan extends To implements Maneable {
private Maneable man; // //Cela fait de lui un patron décorateur (DECORATOR)
public void eat(Integer number, Banana food){
if(ICanEat().booleanValue())
                    man.eat(number, unity);
          else
                    System.out.println("I can't !");
}
    boolean ICanEat(){
    return eatStat. booleanValue() ;
    }

}
```

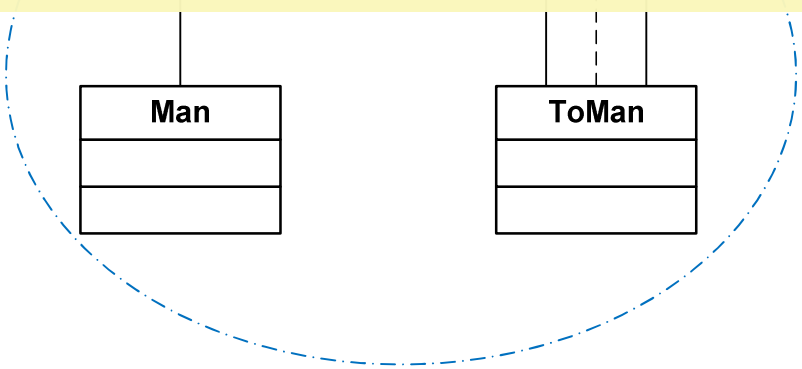



Figure 5-1: Diagramme de classe de ToMan

On remarque d'après ce diagramme que l'agent ToMan a le même comportement que l'objet Man, c'est-à-dire celui défini par l'interface Maneable.

5.3.2. L'application

La MIDlet de notre framework démarre par un écran SplashScreen qui va être l'Observer de cette MIDlet.

```
public class ToMIDlet extends MIDlet {
        public void startApp() throws MIDletStateChangeException {
            SplashScreen splashScreen =new SplashScreen();
            Observable observable=new Observable(this);
             observable.addObserver(splashScreen);
            observable.setChanged();
            observable.notifyObservers(this);
        }
}
```


Donc, lorsque la MIDlet démarre, elle lance un SplashScreen. Ce dernier est un écran d'accueil qui va servir aussi pour lancer le Menu de notre application.

Le Menu comporte, entre autres, un moyen pour lancer l'application.

Dans Menu on lance l'application en appelant la fonction startTo

```
public class Menu extends List implements CommandListener, Observer {
private void startToApplication(){
        observable =new Observable(this);
        ToApplication toApplication = new ToApplication();
        observable.addObserver(toApplication);
        observable.setChanged();
         observable.notifyObservers(strStartTo);
        toApplication.launch();
    }
}
```

La classe de base du framework est la classe AbstractToApplication.

C’est la classe AbstractToApplication qui va comporter la Template Method application ().

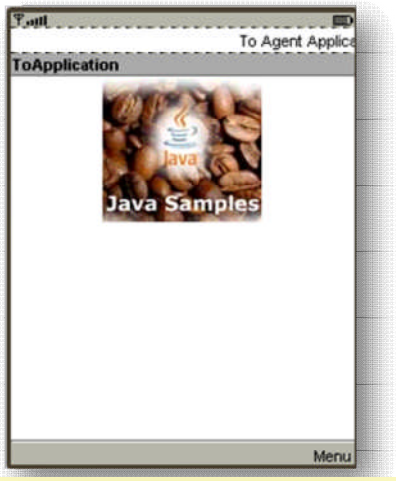
Donc lorsqu’on veut créer une application on doit étendre cette classe, c'est-à-dire implémenter la méthode application(), et c’est dans celle-ci qu’on doit mettre les éléments de notre application.

La classe ToApplication est la classe principale de l’application, elle hérite de la classe abstraite AbstractToApplication.

```
public class ToApplication extends AbstractToApplication{
public ToApplication(){
        super();
}
```

5.3.3. Le Test

Après le lancement de notre application la MIDlet démarre le SplashScreen qui affiche l'écran d'accueil :



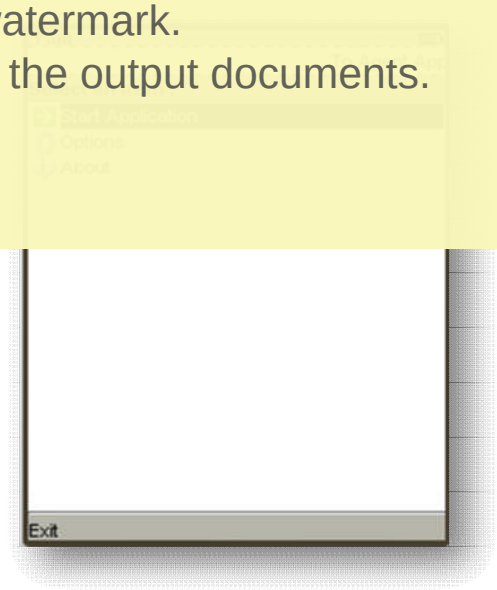


Figure 5-3 Le Menu principal

Dès que l'utilisateur lance Start Application la méthode application() va s'exécuter.

5.3.3.1. Test du système expert

On est obligé d'attacher un moteur de raisonnement à chaque *To.*
Dans notre cas on a instancié le moteur de raisonnement par un système expert (chaînage avant).

Pour le faire, on va suivre les étapes suivantes :

- Création de la base de faits :

Dans notre exemple, on va créer une base de faits avec les faits suivants *: "a la gorge rouge", "a des frissons", "a mal au ventre", "a le nez bouché",* et *"a de la grippe".*

Le code correspondant est :


```
LinkedList lf=new LinkedList();                    // Création une liste des faits
lf.insertAtHead(new Fact("a la gorge rouge"));//Ajout d'un nouveau fait à la liste
lf.insertAtHead(new Fact("a des frissons"));
lf.insertAtHead(new Fact("a mal au ventre"));
lf.insertAtHead(new Fact("a le nez bouché"));
lf.insertAtHead(new Fact("a de la grippe"));
FactsBase fb=new FactsBase();                      // Création de la base des faits
fb.setFactsLinkedList(lf);                         // affectation de la liste des faits à la base des faits
```

- Création de règles :
  Si on veut créer une règle A : a mal au ventre → a l'appendicite ;
  Le code correspondant sera :

```
LinkedList lh=new LinkedList();                    // Création une liste des hypothèses
lh.insertAtHead(new Fact("a mal au ventre"));
LinkedList lc=new LinkedList();                    // Création une liste des conclusions
lc.insertAtHead(new Fact("a l'appendicite"));
Rule a=new Rule("A");                              // Création d'une Règle vide
a.setListofHypothesis(lh);                         //l'ajout de la liste des hypothèses a la règle
a.setListofConclusions(lc);                        //l'ajout de la liste des conclusions a la règle
```

De la même façon on définit les autres règles :

B : *a de la grippe, a mal aux oreilles, a mal à la gorge → a les oreillons*

C : *a la gorge rouge, a un rhume → a de la grippe*

D : *a de la grippe, a le nez bouché → a un rhume*

Un exemple de règle avec la négation est la règle E : *a mal au ventre,*

eat too bananas → est guéri, ¬a mal au ventre.

On la définit comme suit :

```
lh=new LinkedList();
lh.insertAtTail(new Fact("a mal au ventre"));
lh.insertAtTail(new Fact("eat too bananas"));
```


- Création de la base des règles :

Elle est analogue à la création de la base de faits :

```
RulesBase rb=new RulesBase(); // création de la base des règles
rb.insertRuleAtTail(a); // Insertion de la règle A à la base des règles
rb.insertRuleAtTail(b);
rb.insertRuleAtTail(c);
rb.insertRuleAtTail(d);
rb.insertRuleAtTail(e);
```

- Création du système expert :

  Il suffit de rattacher la base de faits et la base de règles au système expert de la manière suivante :

  ExpertSystem es = new ExpertSystem(); //Création du système expert

  es.setFactsBase(fb); // ajout du base des faits au système expert

  es.setRulesBase(rb); // ajout du base des règles au système expert

On peut voir la base de faits de notre système expert sur l'écran du simulateur :

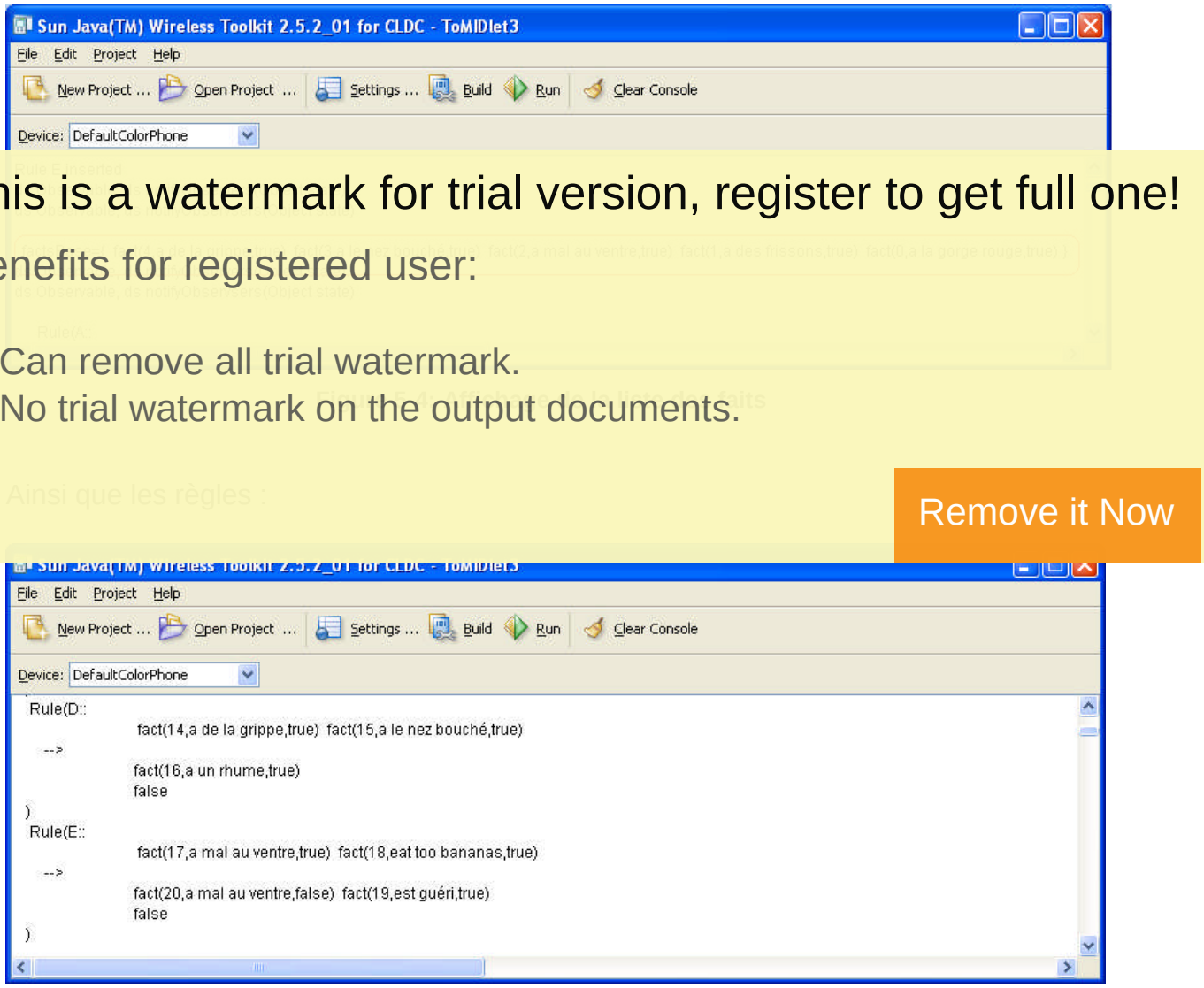


Figure 5-5: Affichage des règles de la base de règles

- Affecter le système expert à un agent *To* :

Après qu'on eu créé le système expert on va décorer un objet *Man* en un objet *ToMan,* on doit lui rajouter son moteur de raisonnement c'est-à-dire un système expert dans notre cas :

```
// Création d'un objet Man
Man man=new Man("Mazari", "Redha", new Integer(22));
// Décoration de l'objet Man en un ToMan
ToMan  toMan= new ToMan("redha",man, es);
```

Pour le reste, c'est le Framework qui s'occupe d'attacher automatiquement les agents To au World et à l'Environment .

Quand le World reçoit un nouvel agent il lance le View de l'agent To, qui n'est rien

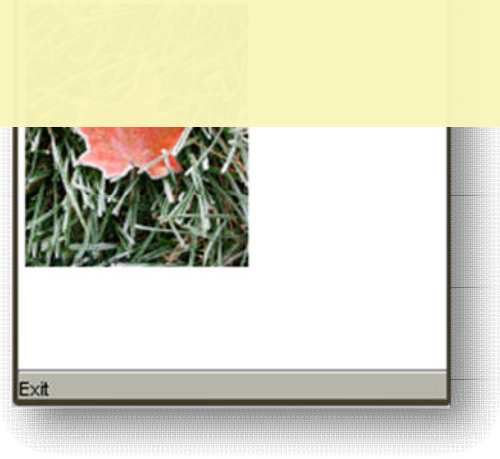


Figure 5-6: La vue de l'agent To

A la création de l'agent ToMan, le démarrage du moteur de raisonnement se fait automatiquement.

Dans notre cas c'est le moteur d'inférence qui va inférencier automatiquement.

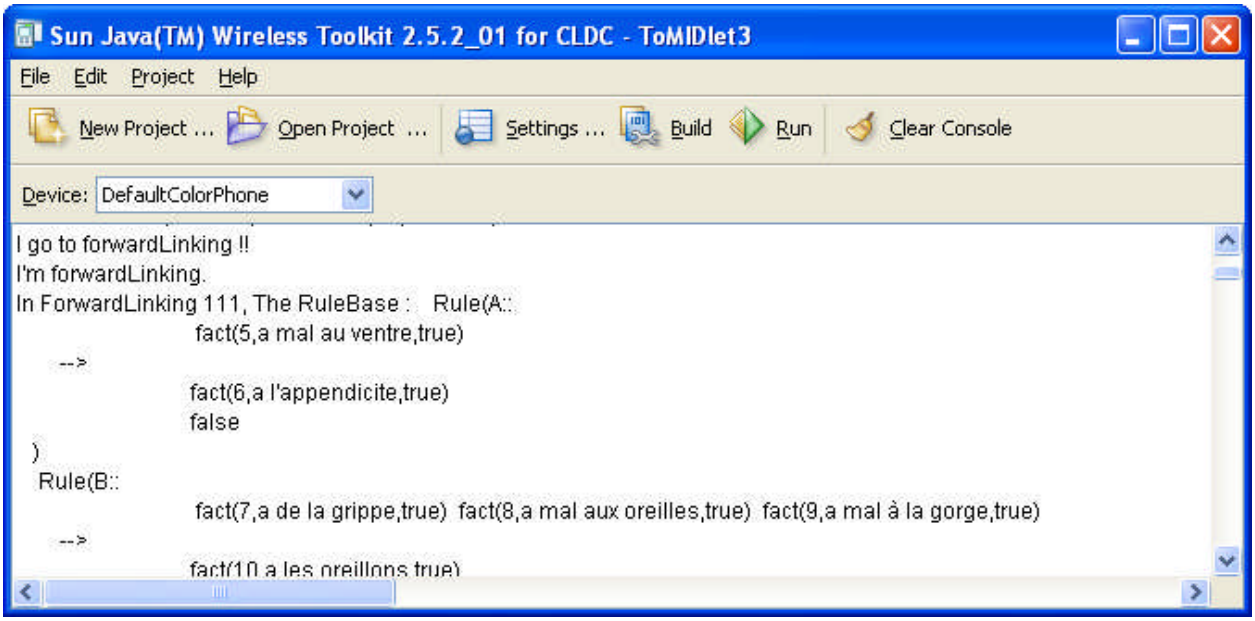


Figure 5.7: démarrage du système expert

Dans notre exemple le résultat de l'inférence de ce système expert est la production de la base de faits suivante :

factsBase={ fact(4,a de la grippe,true) fact(3,a le nez bouché,true) fact(2,a mal au ventre,true) fact(1,a les oreillons,true) fact(0,a la gorge rouge,true) fact(24,a l'appendicite,true) fact(26,a un rhume,true) }.

On constate qu'il y'a eu introduction des nouveaux faits suivants :

a l'appendicite et a un rhume.

Donc, c'est le moteur d'inférence qui a rajouté ces deux nouveaux faits à la base de faits.

5.3.3.2. Test de la communication

On va tester l'envoi de message entre deux agents. Premièrement en doit créer un deuxième agent :

```
Man man1=new Man("Sidoumou", "Mohamed Redha", new Integer(21));
ExpertSystem es2=new ExpertSystem();
ToMan  toMan1= new ToMan("mohamed",man1,es2);
```

Figure 5-8: L'agent *toMan1*

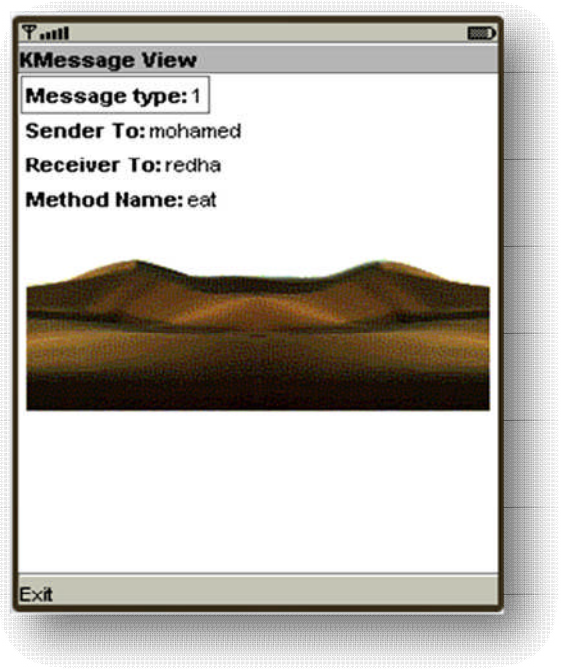
On voit apparaître la vue affichée après la création du deuxième agent.

L'agent toMan1 envoie un message à l'agent toMan pour exécuter la méthode eat(), le code correspondant est :

```
Banana banana = new Banana(new Integer(3));
Object[] objet s= {new Integer(5),banana};
    // On va definir la méthode EatFoodKMethod pour l'agent toMan
EatFoodKMethod eatFoodKMethod= new EatFoodKMethod(toMan,objets);
    //On va créer un Message a toMan1 comme émetteur et
    eatFoodKMethod comme objet de message
KMessage kMessage=new KMessage(toMan1,eatFoodKMethod);
    // L'envoi du message
getWorld().postMessage(kMessage);
```

La classe EatFoodKMethod est définie comme suit:

```
public class EatFoodKMethod extends KMethod{
        public EatFoodKMethod(){
    }
    public EatFoodKMethod(To rec, Object[] args){
        setMethodName("eat");
        setFullMethodName("eatIntegerBanana");
        Class[] klasses={new Integer(1).getClass(),new Banana().getClass()};
        setParamTypes(klasses);
        setAgentRec(rec);
        setArgs(args);
        setClassName("Man");
```




```
        public void execute(KMethod method){
…
                if(c.equals("eatIntegerBanana")){
                        man.eat((Integer)method.getArgs()[0],(Banana)method.getArgs()[1]);
                        if((man.getIsIll()).booleanValue() == true){
                                getToState().putToState("eat too bananas",true);
                        }
                }
        }
}
```

L'exécution de ce message va lancer l'exécution de la méthode eat() de l'objet man. Aussi il vérifie s'il est malade alors il ajoute eat too bananas dans son état.

Cet état se transforme en un fait et s'ajoute à la base des faits de l'agent.

Voici la nouvelle base de faits :

factsBase={ fact(4,a de la grippe,true) fact(3,a le nez bouché,true) fact(2,a mal au ventre,true) fact(1,a des frissons,true) fact(0,a la gorge rouge,true) fact(24,a l'appendicite,true) fact(26,a un rhume,true) fact(37,eat too bananas,true) }

Dès qu'un nouveau fait rentre dans la base de faits, le système expert relance l'inférence et la règle E s'applique, et dans ce cas on obtient la nouvelle base de faits :

factsBase={ fact(4,a de la grippe,true) fact(3,a le nez bouché,true) fact(2,a mal au

5.3.3.3. Test de la mobilité

On va tester la migration de l'agent To entre deux Environments.

L'Environment lance avec lui un MultiToServer qui écoute sur le port par défaut « 9191 ».

En réalité le MultiToServer utilise le même port pour l'écoute et l'envoi, mais pour le test on doit changer le port d'envoi du premier Environment, c'est juste pour le simulateur parce qu'il lance les MIDlets sur le même host qui nous oblige de définir le port d'écoute de MultiToServer du deuxième Environment avec la valeur du port d'envoi de MultiToServer du premier Environment.

Dans ce cas là les deux Environments vont pouvoir communiquer sur le même host.

La méthode application() de la classe ToApplication du première MIDlet sera définie comme suit :

```
public void application(){
        Man man=new Man("Mazari", "Redha", new Integer(22));
        ToMan  toMan= new ToMan("redha",man, es);
        toMan.migrate("127.0.0.1");
}
```

Pour lancer le deuxième Environment on lance une deuxième MIDlet en définissant la méthode application() de la classe ToApplication comme suit.


```
        catch (InterruptedException ie) {}
    }
}
```

Le principe de la migration est de sérialiser l'agent puis l'envoyer vers le host destinataire pour que ce dernier désérialise l'agent et le relance.

Comme J2me ne supporte pas la sérialisation et le Classloader, l'idée de notre solution et de créer un tableau de bytes comportant toutes les informations sur l'agent et de les envoyer vers le host destinataire pour que ce dernier puisse créer un nouvel agent à partir des ces information là.

La méthode migrate(host) de To va envoyer un AMessage pour créer un agent To avec les paramètres renvoyés par la méthode abstraite persist() de l'agent To, cette dernière qui doit être implémentée au niveau des vrais To (classe concrète).

Pour ToMan on la définie comme suit :

```
public synchronized byte[] persist() throws IOException{
        ByteArrayOutputStream bout=null;
        try{
            bout = new ByteArrayOutputStream();
            DataOutputStream      dout = new DataOutputStream( bout );
            dout.writeUTF(getAgentName());
            dout.writeUTF(getClass().getName());
            byte[] data=null;
            while(data==null){
                data=toMan.persist();
            }
            dout.writeInt( data.length );
            if( data.length > 0 ){
                bout.write( data );
            }
            dout.flush();
        }catch(IOException ioe){
            ioe.printStackTrace();
        }
    return bout.toByteArray();
}
```

Comme ToMan décore un Man, on doit aussi envoyer les informations sur l'objet Man, alors elle va appeler la méthode persist() de l'objet Man :

```
public byte[] persistFct() throws IOException{
        byte[] bytesResult=null;
        try{
            ByteArrayOutputStream bout = new ByteArrayOutputStream();
            DataOutputStream      dout = new DataOutputStream( bout );
            dout.writeUTF(getClass().getName());
            dout.writeUTF(name);
            dout.writeUTF(firstName);
            dout.writeInt(age.intValue());
            dout.flush();
            bytesResult=bout.toByteArray();
```


Donc l'Environment reconstruit l'agent en utilisant la méthode resurrect(byte[] data) de l'agent To.

Dans notre cas la fonction resurrect(byte[] data) de ToMan sera définie comme suit :

```
public void resurrect( byte[] data ) throws IOException{
        ByteArrayInputStream bin = new ByteArrayInputStream( data );
        DataInputStream      din = new DataInputStream( bin );
        setAgentName(din.readUTF());
        int    len = din.readInt();
        byte[] tmp = new byte[ len ];
        din.readFully( tmp );
        man=(Maneable)new Man();
```

```
        man.resurrect(tmp);
        bin.close();
        din.close();
    }
```

On doit aussi définir la méthode resurrect(byte[] data) pour l'objet Man :

```
    public void resurrect( byte[] data ) throws IOException{
        ByteArrayInputStream bin = new ByteArrayInputStream( data );
        DataInputStream      din = new DataInputStream( bin );
        name=din.readUTF();
        firstName=din.readUTF();
        age=new Integer(din.readInt());
        din.close();
```

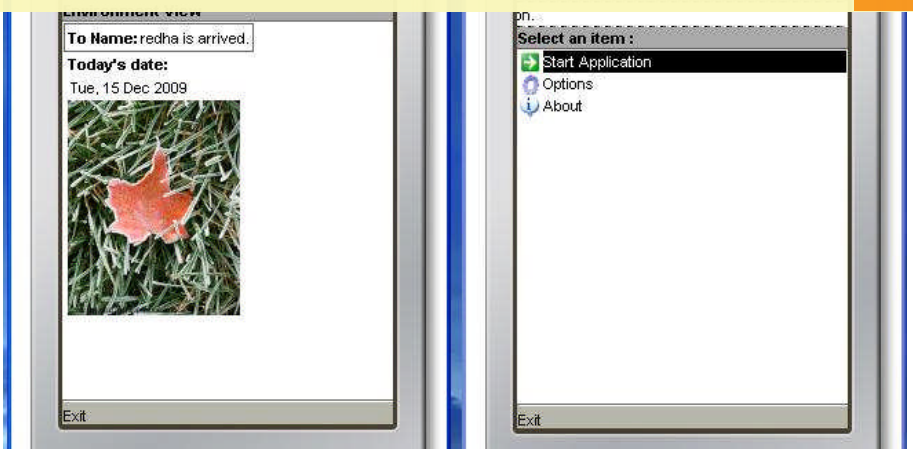



Figure 5-10: Migration de l'agent "Partie 1"

Après que l'agent migre vers le deuxième Environment la deuxième MIDlet affiche le View correspondant à l'agent ToMan.

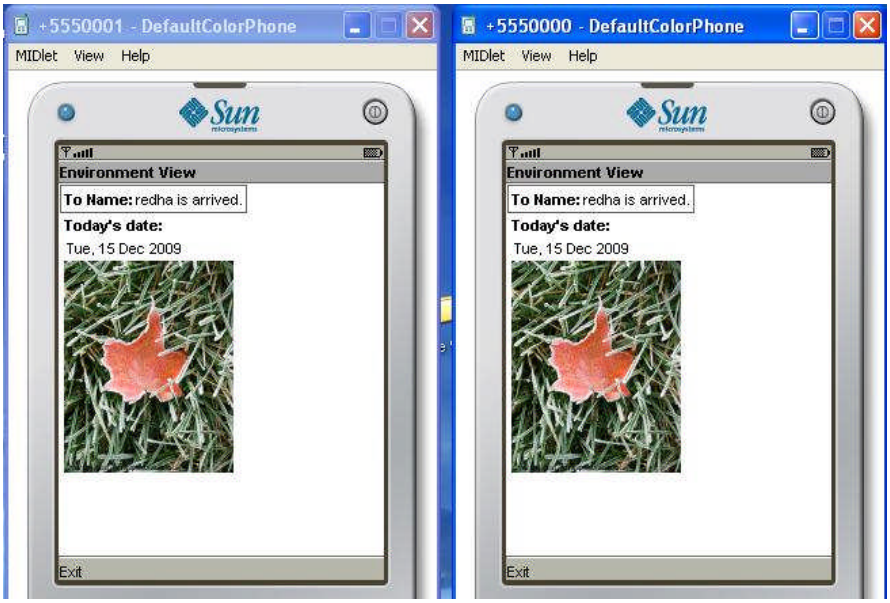


Figure 5-11: Migration de l'agent "Partie 2"

Il reste toujours à améliorer certains modules comme la communication et la mobilité qui font un objet de perspectives que nous allons présenter dans la section suivante.

CONCLUSION

Cette recherche ne peut pas espérer faire des suggestions fiables et définitives au sujet des techniques à employer pour concevoir un framework pour agents intelligents et mobiles sur téléphones portables. Elle a pour but d'adresser ces questions, en n'utilisant que des patrons de conception standards dans toute la phase de conception, ayant pour résultat une phase claire d'exécution, et un framework fortement réutilisable.

1. Contributions

Les contributions de ce projet incluent principalement la conception et la réalisation d'un framework à base de patrons pour des agents intelligents et mobile pour des dispositifs mobiles.

Cette thèse couvre des aspects avancés de la technologie d'objet, avec une orientation particulière sur la résolution des problèmes avec des patrons de conception, aussi bien que sur une architecture à base de framework. Des modèles de conception sont présentés comme des incitateurs de productivité, c.-à-d., comme une aide à résoudre les problèmes que rencontrent les développeurs en orienté

objets. A un niveau plus élevé, nous avons montré comment nous avons pu utiliser les patrons architecturaux pour structurer l'organisation globale du framework. Les patrons architecturaux résolvent les forces principales en structure d'application et nous ont permis la création d'un framework d'agents modifiable et réutilisable.
Dans ce travail nous avonsproposé un certain nombre de patrons standards et nous avons examiné le rôle de ces patrons dans la conception de notre framework.

Cette thèse vise à améliorer de manière significative les qualifications de développement des concepteurs de frameworks. La conception et les patrons architecturaux sont explorés et leur utilisation et développement sont pratiqués comme facteur significatif pour créer des frameworks bien conçus question de qualité du logiciel. Après acquisition de la connaissance du modèle développé dans ce travail, les développeurs-utilisateurs seraient en mesure :

- d'apprendre à appliquer directement les patrons de conception aux problèmes de développement,
- de structurer les systèmes en appliquant les patrons architecturaux,
- de concevoir un framework sous *J2me* flexible et maintenable.

Les contributions techniques tombent dans trois principaux domaines :

- Premièrement, en construisant un SMA caractérisé par un concept de communication, d'interaction, en rajoutant un coté raisonnement aux objets (au sens de l'orienté objets) pour créer des agents nous pouvons fournir de vraies possibilités de simulation d'entités du monde réel,
- Une pseudo mobilité a été atteinte malgré l'absence de la sérialisation, la réflexivité et du chargement dynamique de la classe (ClassLoader),
- troisièmement, un framework sous J2me réutilisable est conçu et réalisé en utilisant la technologie des patrons.

Cette thèse présente un nouveau framework à base de patrons pour agents sur téléphone portable, confectionné à l'aide d'une architecture répondant à des attributs de qualité comme la disponibilité, la généricité et la réutilisabilité.

Nous avons montré par quelques décisions architecturales comment introduire les reconfigurations pour l'amélioration des attributs de qualité. Dans cette thèse nous avons aussi spécialisé, et donc appliqué le framework développé.
Notre approche peut être reprise à l'avenir avec d'autres décisions architecturales dans le contexte des systèmes complexes, puisqu'elle soutient fortement la généricité et la réutilisabilité.

Nous espérons que notre contribution principale devait fournir une conception à base de patrons, d'agents intelligents et mobiles, sous *J2me* un framework à base de patrons dont la spécialisation va être facile à comprendre et l'architecture plus réutilisable. Pour cela, nous avons adopté un procédé de conception qui se fait par des évolutions et des itérations des activités de conception grâce au modèle à base de patrons standards. Chaque itération incorpore la conception résultante avec

- la conception d'un pseudo réflexivité sous j2me,
- la confection et l'application de la sérialisation dans la mobilité,
- la généralisation de la réutilisabilité avec par l'utilisation du modèle d'observateur dans l'architecture entière,
- la conception d'un sous-framework de raisonnement spécialisé par un système chaînage avant basé sur les règles de production,
- la conception d'une pseudo-mobilité par le clonage sous j2me qui ne comporte pas de chargeur de classes (class loader).

2. Conclusion

La mise en place d'une plate-forme technique pour agents commune à tous les dispositifs mobiles est une opération très délicate. Chaque appareil possède des spécificités qui impliquent aux applications de s'adapter à différentes caractéristiques d'affichage ou de pointage. Toute la difficulté de concevoir des technologies pour l'embarquer réside dans cette complexité inhérente à la diversité de l'offre. Java 2 Micro Edition est une architecture technique dont le but est de fournir un socle de développement aux applications embarquées. L'intérêt étant de proposer toute la puissance d'un langage tel que Java associé aux services proposés par une version bridée de J2SE, néanmoins il se trouve que dans le cas des agents sur les dispositifs mobiles ils perdent beaucoup de leurs avantages.

Les applications écrites à partir du profil MIDP, en l'occurrence les MIDlets, doivent

temps réel. C'est justement ce genre de domaines qui va donc présenter un grand intérêt.

Les agents intelligents et mobiles sont l'une des solutions qui va permettre de créer ce genre d'application évoluée.

Dans ce mémoire nous avons exposé notre conception d'un framework dans cette optique, en utilisant les outils et techniques modernes de développement de logiciels comme les patrons et les frameworks pour concevoir de telles applications.

Dans ce mémoire nous avons présenté un framework à base de patrons conçus dans le but de développer des applications orientées agents intelligents et mobiles sur les dispositifs mobiles.

En concevant le nouveau framework, nous avons prêté une attention particulière aux problèmes de la réutilisabilité du logiciel.

Notre but le plus important était de résoudre la dépendance des modules de conception dans la conception basée sur des patrons ; cela fût réalisé au moyen du patron Layered Pattern modifié.

Cette thèse présente principalement une illustration de la façon de choisir puis d'employer les patrons standards dans la conception d'un framework pour agents sur des téléphones mobiles portables, en vue de sa réutilisation.
Concevoir avec des patrons peut simplifier la complexité de conception en séparant des soucis de conception au niveau des micro-architectures du framework, et constitue une base pour la réutilisabilité des frameworks.

Le travail dans la technologie de la programmation pour des systèmes d'agent se concentre principalement sur la recherche de nouveaux patrons orientés agents.
Notre travail diffère des autres travaux de recherches effectués sur les frameworks pour agents du fait nous n'avons utilisé que des patrons connus dans la conception architecturale du framework comme dans l'expression des caractéristiques de base des agents, comme la communication, la collaboration, l'aspect co
action, la mobilité et les systèmes de raisonnement, et surtout du fa
travail précurseur dans le domaines des frameworks à base de patrons pour agents sur des dispositifs mobiles.

L'expérience qu'on vient d'acquérir à travers cette étude nous a convaincu que les frameworks à base de patrons de conception ont le potentiel de contribuer pour atteindre ce but, c'est-à-dire une conception claire et réutilisable.

3. Bilan

Il est maintenant temps, après avoir exposé l'utilisation des patrons dans le développement d'un framework pour agents, de tirer un bilan de nos travaux. Ce bilan se décompose en deux parties : d'une part, sur le travail réalisé et, d'autre part, sur l'expérience et les connaissances acquises lors de cette réalisation.
Nous commencerons par la seconde partie, l'expérience acquise et les conclusions que l'on peut en tirer, pour revenir sur la première, nos réalisations, accompagnées des perspectives qu'elles laissent envisager.

La plus grande part de nos réalisations a consisté à retrouver les patrons adaptés à la conception orientés agent. Or, retrouver ces patrons, c'est-à-dire décrire des solutions partielles et éprouvées à des ensembles de contraintes, n'est pas une

L'intérêt des patrons réside dans le fait qu'ils forment un lexique, mais l'effet de mode induit une surabondance de patrons, dont une partie se compose de spécialisations de patrons déjà connus à cause de bien connaître les patrons, ce qui n'est pas souvent le cas, l'on en arrive facilement à noyer leur utilité dans un vocabulaire pas trop important.

La méthode que nous avons utilisée pour la découverte des patrons est la recherche de patrons utilisés dans un ensemble de logiciels (i. e. des structures d'associations répétées dans les différents codes et diagrammes).
Au début de leur confection, ces structures guident l'auteur à travers leur description. Ensuite, les retours après plusieurs étapes de re-ingénierie permettent d'améliorer cette description.

Donc, les patrons sont une capitalisation de l'expertise, réalisée par une formalisation de triplets contexte-problème-solution visant à décrire la solution proposée par une pratique éprouvée résolvant certaines contraintes récurrentes dans un contexte précis.

Les différents patrons pour la conception des agents que nous avons décrits dans ce mémoire apportent différents avantages à la conception de systèmes agents. D'une part, en tant que patrons, ils apportent au développement des agents tous les avantages que les patrons peuvent apporter au développement de logiciels.
L'utilisation des patrons de conception dans le domaine des agents nous ont permit à eux seuls de mieux comprendre et exprimer le paradigme agent : la collecte d'informations et le rapprochement des différents travaux qui servent d'exemples aux patrons apportent tant à la théorisation et à l'abstraction qu'à l'applicabilité et à l'application de modèles, d'architectures ou de méthodes qui semblaient bien connus, sans toutefois l'être réellement. De plus, l'adéquation des patrons comme aide à l'apprentissage du paradigme agent, comme le sont pour le paradigme objet, est une contribution importante à ce paradigme.

décrivant notre conception de l'agent To, se montre tout autant applicable à des niveaux d'abstraction plus élevés que la littérature sur les motifs tendait à le laisser supposer. C'est d'ailleurs cette utilisation des patrons à des niveaux d'abstraction élevés qui différencie principalement notre travail de celui des autres équipes utilisant des motifs orientés agents des autres équipes (les motifs que nous avons utilisé sont plus qualifiés orientés objet qu'orientés agent).

Parallèlement, il existe sûrement d'autres motifs applicables au développement des agents que nous n'avons pas décrits ici. Il serait tentant à l'avenir d'approfondir notre travail, en listant tous les autres motifs applicables, tout en comparant leurs avantages et inconvénients.

4. Perspectives d’Avenir

Nous avons évalué le framework en développant une application. Les premiers tests ont montré que le framework devrait mûrir un peu plus et reste pas facile à manipuler par des développeurs non confirmés pour être adopté sur une large échelle. D’autres fonctionnalités vont lui manquer encore. Par conséquent, une nouvelle re-conception (reingénierie) du framework serait nécessaire, pour lui adjoindre une meilleure sérialisation, le côté sécurité, le côté apprentissage devraient être réalisés dans des travaux futurs.

D’autre part du côté conception, il nous semble que notre approche est intéressante, néanmoins, nous allons surtout essayer de comparer l’efficacité du choix de nos patrons par rapport :

REFERENCES

1. Bigus J.P. et Bigus J., “Constructing Intelligent Agents Using Java: Professional Developer's Guide”, Wiley, London. 2001.
2. Tommy B. et Kim S., “A flexible framework for mobile collaboration”, Master of Science in Computer Science, Norwegian University of Science and Technology, 2006.
3. Dunin B. M., et Nawarecki E., “From Theory to Practice in Multi-Agent Systems”, Revised Papers, vol. 2296 de Lecture Notes in Computer Science. Springer-Verlag, Heidelberg, 2002.
4. [illegible] Using Software Components. Proc. CSCWD04, Xiamen, RPC. Mai 2004.
5. [illegible] do H. SAAS, une méthode d'analyse et de conception pour les systèmes multi-agents. Plate-forme AFIA. Grenoble. 25-26 juin 2001.
6. [illegible] rk for Object-Oriented Concurrent Programming Design and Experience”, pages 209–231. Hermès, Paris, 2000.
7. Carl-H. W. et Michael S. N., “A framework for mobile collaborative applications on mobile phones.”, Technical report, Norwegian Science and Technology, 2004.
8. [illegible] Graph Transformation”, in First International Workshop on Agent-Oriented Software Engineering, aose’2000, Limerick (Irlande), 10 juin 2000.
9. Girardi R., “Reuse in agent-based application development”, First International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, Florida, 2002.
10. Barthès J-P. A Chevallier P., Courdier R., Guessoum Z., Gutknecht O. Mathieu P, Ocello M. Comparaison des plates-formes SMA, dans Organisation et applications des SMA. René Mandiau, Ammanuelle Grislin-Le Strugeon, André Péninou, Eds. ISBN 2-7462-0439-8. Lavoisier. 207-242, 2002.
11. Dunin-K éplicz B. M. et Nawarecki E., “From Theory to Practice in Multi-Agent Systems”, Second International Workshop of Central and Eastern Europe on Multi-Agent Systems, ceemas2001, Cracow, Poland, September 2001.
12. Cooper J. W., “Java Design Patterns: A Tutorial”, Addison Wesley, Reading, Massachusetts (USA), janvier 2000.
13. Sauvage S., “Des patterns pour la conception de SMA” , in El Fallah Seghrouchni & Magnin 01, 2001.

14. Douglas Samuel Kirk, “Understanding Object-Oriented Frameworks”, Phd thesis, University of Strathclyde, Glasgow, Aout 2005.
15. Paul Clements, Rick Kazman et Mark Klein, “Evaluating Software Architectures: Method and Case Studies”, Addision Wesley, 2002.
16. Dias, M.S., and Richardson, D.J., “The role of Event Description in Architecting Dependable Systems”, In proceeding of WADS: Workshop on Architecting Dependable Systems. Orlando, USA, Mai 2002.
17. Medvidovic N., Mikic-Rakic M., Mehta N., and Malek S., “Software Architectural Support for Handheld Computing”, IEEE Computer Special Issue on Handheld Computing, 36 (9) 66-73, 2003.
18. Gacek, C., “Software Reuse: Methods, Techniques, and Tools”, 7th International Conference, ICSR7 Austin, TX, Proceedings, 2002.
19. R. S. Pressman, “Software Engineering, A Practitioner’s Approach”, 5th edition, McGraw-Hill, 2001.
20. Mili H., Mili A., Yacoub S., et Addy E., “Reuse-Based Software Engineering : Techniques, Organization, and controls”, John Wiley & Sons, N. Y., 2002.
21. Douglas K., “Identifying The Problems of Large Scale Reuse: A Personal Case Study”, Technical Report (EFoCS-41-2001), University of Strathclyde, UK. Kirk, Douglas, 2001.
22. Gamma E., Helm R., Johnson R. E. et Vlissides J., “Design Patterns: Reading, Massachusetts (USA), 2000.
23. Rohnert H. et Sommerlad P., “Pattern-Oriented Software Architecture - A System of Patterns”, Wiley, John Sons, Incorporated, 2001.
24. frameworks using pattern languages, Master of Science, University of Northern Iowa, December 2003.
25. Montgomery J. D., “Situated Observation and Participation in Multiple-Agent Systems”, Master of Science, University of British Columbia, July 2003.
26. Ma G. et Shi C., “Modeling Social Agents in BDO Logic”, in E International Conference on Multi-Agent Systems, icmas’2000, pages 412, Boston, Massachusetts, 2000.
27. Jennings N. R., “On agent-based software engineering”, Artificial Intelligence Journal, no 117, 277–296, 2000.
28. Ferber J., “Les systèmes multi-agents : vers une intelligence collective“, Interédition, Paris, 1999.
29. Wooldridge M. et Jennings N. R., “An introduction to multiagent systems”, John Wiley and Sons Ltd, 2002.
30. Zbigniew S. et Arciszewski T., “Intelligent Agents in Design”, 15th international Conference on Design Theory and Methodology Chicago Illinois, September 2003.
31. Barthès J-P. A. Capitalisation des connaissances dans l'entreprise in Organisation et applications des SMA R. Mandiau, E. G.-L. Strugeon, A. Péninou. 2002.
32. John F. Simon, Mobility Agents, Artist Book, 2005.
33. Michael Friedrich, Kirsten Terfloth, Gerd Nusser, and Wolfgang K¨uchlin, “Mobile Agents: A Construction Kit for Mobile Device Applications”,WSI for Computer Science, University of Tuebingen, 2004.

34. Paolo .B, Loris .P, Paolo .B, and Tsvi .K, “Agent Patterns for Ambient Intelligence”, ITC-irst via Sommarive 18 I-38050 Trento-Povo, Italy, 2004.
35. Michael Weiss,“Patterns for e-Commerce Agent Architectures: Using Agents as Delegates”, Carleton University, Ottawa, Canada, 2001.
36. Elizabeth A. Kendall, Margaret T. Malkoun, “The Layered Agent Patterns”,Computer Systems Engineering, Royal Melbourne Institute of Technology.
37. Lind .J, “Patterns in Agent-Oriented Software Engineering”, iteratec GmbH Inselkammerstr. 4D-8 2008 Unterhaching, Germany 2002.
38. Wood M. F. et DeLoach S. A., “An Overview of the Multiagent Systems Engineering Methodology”, in First International Workshop on Agent-Oriented Software Engineering, aose’2000, Limerick (Irlande), juin 2000.
39. Yim H., Cho K., Kim J. et Park S., “Architecture-Centric Object-Oriented Design Method for Multi-Agent Systems”, in Fourth International Conference on MultiAgent Systems, icmas’2000, Boston, Massachusetts (USA), juillet 2000.
40. Zambonelli F., Jennings N. R. et Wooldridge M. J., “Organisational Abstractions for the Analysis and Design of Multi-Agent Systems”, in First International Workshop on Agent-Oriented Software Engineering, aose’2000, Limerick (Irlande), juin 2000.

47. Rami Enbachi, Hani Al-Haddad, “MTA: Mobile Tourist Agent”,URL: http://www.ameinfo.com/.
48. Bee-gent URL: http://www2.toshiba.co.jp/beegent/index.htm.
49. Jeon, H., Petrie, C., Cutkosky, M.: “JATLite: a Java agent infrastructure with message routing”, In: IEEE Internet Computing 4 (2) (2000) 87-96
50. Bergenti, Federico, Poggi, and Agostino. “LEAP: A FIPA Platform for Handheld and Mobile Devices”, URL: http://leap.crmparis.com/public/docs/ATAL2001.pdf
51. FIPA. “The Foundation for Intelligent Physical Agents”, URL: http://www.fipa.org/.
52. Fernando Koch, “3APL-M: Platform for Lightweight Deliberative Agents”, URL: http://www.cs.uu.nl/3apl-m.
53. Bernaer, Stijn De Causmaecker, Patrick Maervoet et Joris Vanden Berghe, “Greet An agent framework for effective data transfer”, 2004.
54. Koch F. L., Meyer J-J C., “Knowledge Based Autonomous Agents for Pervasive Computing using AgentLight”, ACM/IFIP/USENIX International Middleware Conference, MIDDLEWARE 2003.

55. Sun Microsystems. “Java 2 Platform, Standard Edition (J2SE)”. URL: http://java.sun.com/j2se/.

56. MIA Research Group. “MIA - Mobile Information Agents for the WWW”, URL: http://www.unikoblenz.de/~bthomas/MIAHTML/, 2007.

57. Gerd Beuster, Bernd Thomas, and ChristianWolff. “MIA – A ubiquitous multi-agent web information system”. In Proceedings of International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA’2000), 2000.
58. Poslad S., Buckle P. et Hadingham R., “The FIPA-OS agent platform:Open Source for Open Standards”, published at PAAM, Machestor, UK, April 2000.
59. ALBUQUERQUE Ryan L., HÜBNER Jomi F., DE PAULA Gustavo E., SICHMAN Jaime S., RAMALHO Geber L., KSACI: A handheld device infrastructure for agents communication, International workshop No8, Seattle WA , ETATS-UNIS,01/08/2002
60. Sun Microsystems. “Java 2 Platform, Standard Edition (J2SE)” . URL: http://java.sun.com/j2se/.
61. Bruno Delb, “J2ME Application Java pour terminaux mobiles”, EYROLLES.
62. Lind J, “Patterns in Agent-Oriented Software Engineering”, iteratec GmbH Unterhaching, Germany 2002.
63. Vercouter L., Beaune P. & Sayettat C., “Towards open distributed information systems by the way of a multi-agent conception framework”, in Barcelone (Espagne), juin 2000.
64. Bergenti, Federico, Poggi, and Agostino, “LEAP: A FIPA Platform for Handheld and Mobile Devices”, URL: http://leap.crm-paris.com/public/docs/ATAL2001.pdf, 2001.
65. Heecheol, J., Petrie, C., Cutkosky, M., “JATLite : A Java Agent Infrastructure with Message Routing”, Internet Computing, M
66. Fernando Koch, “3APL-M Platform for Lightweight Deliberative Agents”, URL: http://www.cs.uu.nl/3apl-m 2004.
67. Fielding R., “Architectural styles and the design of network-based software architectures”, Ph. D. Dissertation, University of California at Irvine, 2000.
68. Medvidovic N. and Taylor R. N., “A Classification and Comparison Framework for Software Architecture Description Languages”, IEEE Transactions on Software Engineering 26 (1) 70-93, 2000.
69. Aldrich J., Chambers C., and Notkin D., “ArchJava: Connecting Software Architecture to Implementation”, Proc. 24th International Conference on Software Engineering, May 2002.
70. Buschmann F., Meunier R., Rohnert H. et Sommerlad P., “Pattern-Oriented Software Architecture - A System of Patterns”, Wiley, John Sons, Incorporated, 1996.
71. Gamma E., Helm R., Johnson R. E. et Vlissides J., ”Design Patterns: Elements of Reusable Object-Oriented Software”, Addison Wesley, Reading, Massachusetts (USA), 1994.
72. Pree W., Koskimies K., ”Framelets—small and loosely coupled frameworks”, ACM Computing Surveys (CSUR), March 2000.

73. Kirk D. S., “Understanding Object-Oriented Frameworks”, Phd thesis, University of Strathclyde, Glasgow, Aout 2005.
74. Len Bass, Paul Clements, and Rick Kazman, “Software in Practice”, Second Edition, Addison Wesley, 2004.
75.
76.

77. Koch F. L., Meyer J-J C., “Knowledge Based Autonomous Agents for Pervasive Computing using AgentLight”, ACM/IFIP/USENIX International Middleware Conference, MIDDLEWARE 2003.
78. http://www.microsoft.com/windows/embedded/default.mspx
79. Bruno Delb, “J2ME Application Java pour terminaux mobiles”, EYROLLES, 2002.
80. Eckel, B., “Thinking in Java (3rd edition)”, Prentice-Hall, 2001.
81. Ryan L. Albuquerque, Jomi F. Hübner, Gustavo E. de Paula, Jaime S. Sichman and Geber L. Ramalho, “KSACI: A Handheld Device Infrastructure for Agents Communication”, International workshop No8, Seattle WA , ETATS-UNIS (01/08/2002).
82. [AA] AdvetNet, Building SNMP Agents, www.advent net.com 2004.

LISTE DES SYMBOLES ET DES ABREVIATIONS

SMA : Systèmes Multi-agents

IA : Intelligence Artificiel

JVM : Java Virtual Machin

J2ME : Java 2 Micro Edition

MVC : Modèle-Vue-Contrôleur

J2SE : Java 2 Standard Edition