République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur & de la Recherche Scientifique
Université Mentouri de Constantine,
Faculté des Sciences de l'Ingénieur
Département d'Informatique,
Laboratoire LIRE

N ° d'ordre :
Série :

Mémoire présenté en vue de l'obtention du diplôme de

Magister en Informatique

Option : Génie logiciel et intelligence artificielle

PROTECTION DE L'AGENT MOBILE VIA UNE POLITIQUE D'ADAPTABILITE DYNAMIQUE

Présenté par: M^{elle} **Haoua CHERIBI** Dirigé par: **Pr.Zizette BOUFAÏDA**

Soutenue-le : **…./….** **/2007,** devant le jury composé de:

Président :
Mr M. BENMOHAMED Professeur, Université Mentouri de Constantine

Rapporteur :
Mme Z. BOUFAÏDA Professeur, Université Mentouri de Constantine

Examinateurs :
Mr A. CHAOUI MC , Université Mentouri de Constantine

Mr S. CHIKHI MC , Université Mentouri de Constantine

Invitée:
Mme S. HACINI Chargée de cours, Université de Constantine

REMERCIEMENTS

« Nous sommes des nains juchés sur les épaules des géants » Isaac Newton

Je ne serais pas arrivée jusque là si « ALLAH » le tout puissant ne m'avait donné la force et le courage pour continuer et ne m'avait aider et éclairer le chemin. Je tiens à lui exprimer toute ma reconnaissances et ma gratitude.

Je tien à remercier le Professeur M. BOUFAIDA, le chef de l'équipe SIBC, de m'avoir accueilli dans son équipe et d'être toujours présent avec ses précieux conseils, son encouragement et son aide.

Je remercie vivement Mme Z. BOUFAIDA, mon encadreur, de m'avoir pris sous son aile bienveillante en m'encadrant dans ce travail, et de m'avoir fait profité de sa grande expérience et expertise. Je tiens à lui exprimer ma grande reconnaissance pour son orientation, son aide et sa gentillesse. Qu'elle soit assurée de mon très grand respect et appréciation.

Je remercie également Mme S. HACINI, de m'avoir guidé et suivi de prés durant toute cette période. Je tien à lui exprimer mes sincères remerciements pour ses précieux conseils et remarques, pour sa disponibilité, pour sa patience ainsi que sa contribution à ce travail.

Mes vifs remerciements iront naturellement vers l'ensemble des membres du jury; Mr M. BENMOHAMED, Mr A. CHAOUI et Mr S. CHIKHI, qui me font l'honneur d'avoir accepté de juger mon travail.

Je remercie tout particulièrement pour le soutien permanent qu'ils m'ont manifesté, mes très chères parents qui ont cru à mon avenir, qui n'ont jamais cessé de m'encourager jusqu'au bout, et qui m'ont permit de mener à bien mes études.

Merci infiniment à Mr Dj. BENMERZOUG pour son aide et à tous ceux qui m'ont rendu service de prés ou de loin...

RESUME

***Résumé* :** Ces dernières années sont témoins de la naissance d'une technologie nouvelle et prometteuse pour le développement des applications distribuées et ouvertes, celle des agents mobiles. Les principaux objectifs de l'utilisation de cette technologie concernent l'exécution asynchrone et autonome ainsi que la réduction de la charge dans le réseau. Cependant, cette avancée a immédiatement mis en évidence un sérieux problème de sécurité qui a freiné son expansion. Ce travail traite l'aspect de la sécurité de l'agent mobile contre les attaques des hôtes malveillants. Il propose une approche sécuritaire générique, basée sur l'adaptabilité dynamique et adopte la réflexivité comme modèle de conception et d'implantation. L'approche proposée consiste à doter l'agent mobile d'une capacité de flexibilité lui permettant de présenter un comportement variable et imprévisible afin de le protéger contre les tentatives d'analyse de comportement. Dans le but de renforcer le niveau de sécurité, des mécanismes de protection classiques tels que la cryptographie sont aussi utilisés.

***Mots clés* :** agent mobile, sécurité des agents mobiles, adaptabilité, réflexivité

Abstract: These last years are witnesses of the birth of a new technology which is very promising for the development of distributed and open applications, the mobile agents one. Its main objectives are the asynchronous and autonomous execution and reduction of communication with respect to latency and connection time. However, one of the main obstacles to widespread adoption of the mobile agent paradigm seems to be security. This work treats the security of the mobile agent against malicious host attacks. It describes generic mobile agent protection approach. The latter is based on the dynamic adaptability and adopts the reflexivity as a model of conception and implantation. In order to protect it against behaviour analysis attempts, the suggested approach supplies the mobile agent with a flexibility faculty allowing it to present an unexpected behaviour. Furthermore, some classical protective mechanisms are used to reinforce the level of security.

Key words : Mobile Agent, Mobile Agent security, Adaptability, Reflexivity.

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLES

INTRODUCTION GENERALE

« Le commencement est beaucoup plus que la moitié de l'objectif » Aristote

En raison de la demande incessante pour l'amélioration et l'augmentation de la capacité et du rendement des différents systèmes informatiques, une grande diffusion des réseaux informatisés et des systèmes distribués a était lancée. De plus, les exigences et les préférences des utilisateurs qui ne cessent d'accroître, ainsi que la nécessité d'une évolutivité progressive des applications font qu'une vision centralisée, rigide et passive atteint ses limites. En fait, Il y a quelques années la plupart des applications étaient conçues pour fonctionner dans un environnement d'exécution relativement bien connu et maîtrisé. Cependant, cela n'est plus possible dans le contexte actuel, où les conditions d'exécution pour une application donnée sont diversifiées et souvent variables. En conséquence, concevoir de telles applications est de plus en plus complexe.

En revanche, les progrès technologiques récents ont permis l'émergence d'une panoplie de nouveaux moyens facilitant la tâche de la conception et du développement de ces applications. En effet, l'essor de l'informatique distribuée et nomade s'est accompagnée de l'apparition du paradigme des agents mobiles. Ce dernier offre une plus grande souplesse et efficacité de traitement.

Un agent mobile est généralement défini comme une entité logicielle autonome ayant une activité propre, agissant pour le compte d'une personne ou d'une organisation et pouvant migrer d'un site à un autre au sein d'un réseau informatique. L'une des motivations les importantes de ce paradigme est la minimisation des communications distantes permettant ainsi d'économiser la consommation de la bande passante. La technologie des agents mobiles présente d'intéressantes perspectives pour divers domaines d'applications de l'ère actuelle. On trouve, parmi ces domaines, le commerce électronique, la télécommunication, la recherche d'informations dans le web, la gestion des bases de données et des réseaux, etc.

Cependant, les environnements particuliers dans lesquels ces applications sont déployées font que leur construction représente un vrai défi pour les programmeurs. Leur conception impose la prise en compte non seulement des aspects purement fonctionnels mais aussi des problèmes déterminant leur qualité de service comme le temps de réponse, la tolérance aux pannes et en particulier la sécurité [1,9,16,30,32,…].

La sécurité est d'une importance majeure; une faille dans la sécurité d'un système peut causer de graves dégâts. Ainsi, l'idée qu'une transaction puisse être non sécurisée freine son expansion et son utilisation, d'où la nécessité de compter ce service parmi les plus urgents et de mettre en pratique les précautions nécessaires. Toutefois, la mise en œuvre d'une sécurité exagérée représente parfois un obstacle à la productivité. Il est donc important, de conserver un équilibre afin d'assurer la sécurité sans pour autant prohiber l'utilisation légitime du système.

La problématique de la sécurité est par nature multidimensionnelle; les éléments à protéger sont de différentes natures, les catégories d'attaques sont nombreuses, les besoins de sécurité sont divers et les mesures sous-jacentes sont de multiples niveaux et dépendent fortement du contexte en question. En détaillant chaque point, on se rend compte très rapidement qu'il n'existe aucune solution miracle pouvant couvrir tous les aspects requis.

La sécurité dans les systèmes d'agents mobiles couvre principalement deux volets différents : la sécurité des agents mobiles et celle des plateformes qui les hébergent et les exécutent. Le problème de la protection de la plateforme a reçu de considérables attentions de la part des recherches scientifiques, tandisque la protection de ces agents est un domaine qui reste ouvert. En fait, il n'existe actuellement pas de solution pratique satisfaisante, conséquence probable de sa difficulté [1,32,…]. Un agent mobile au cours de son cycle de vie peut être la cible d'une variété d'attaques. En particulier, l'hôte qui l'exécute possède un plein pouvoir sur lui. Du moment qu'il l'héberge, il peut facilement accéder et modifier ses données ainsi que son code.

Les mesures de protection classiques telles que la cryptographie et le mot de passe, ne suffisent malheureusement pas à protéger efficacement l'agent mobile ; des hôtes malveillants peuvent espionner l'agent mobile en analysant son comportement afin de déduire des informations pouvant servir à nuire au bon fonctionnement de l'agent, sans avoir à accéder explicitement à ses données ou à son code. Face à cette réalité, il va falloir trouver une méthode permettant de protéger l'agent mobile contre ces menaces. Cette problématique est peu prise en compte par les travaux de recherches existants [1,9,28,30,32,…], même si quelques uns ont essayé de traiter la sécurité de manière générale.

Tous ces paramètres ont attiré notre attention, et nous ont poussé à réfléchir sérieusement à ce sujet. La sécurisation de l'agent mobile contre les attaques des hôtes malveillants nous intéresse particulièrement. La solution de sécurisation que nous proposons consiste à doter l'agent mobile d'une capacité de flexibilité lui permettant de se comporter d'une manière imprévisible vis à vis des plateformes, afin de compliquer les éventuelles opérations d'analyse de comportement. Le changement de comportement de l'agent mobile doit être établi en tenant compte des contraintes de l'environnement exigées ainsi que de leurs éventuelles variations. Par conséquent, notre agent mobile doit s'adapter aux différents contextes de l'environnement d'exécution. La solution réside donc dans l'établissement d'une politique sécuritaire fondée sur le mécanisme d'adaptabilité dynamique.

L'adaptabilité, assez peu abordée précédemment, occupe le centre d'intérêt d'un ensemble des travaux de recherches actuels. Cette discipline a pour objectif, la conception d'applications pouvant poursuivre leur fonctionnement malgré les changements possibles des contraintes d'exécution. L'adaptabilité trouve de bonnes fondations dans les méthodes de réflexivité [14,34,37,...]. Cette dernière présente un modèle d'organisation permettant à un système de manipuler comme données quelque chose faisant partie de soi durant son exécution. Cela lui permet de raisonner et d'appliquer ses propres capacités d'action sur lui même. Dans le cadre de notre travail, nous adoptons cette démarche afin d'implanter la capacité d'adaptation sur laquelle se base notre approche de protection de l'agent mobile. Nous détaillerons la solution de sécurisation proposée tout au long de ce mémoire.

Ce mémoire est organisé en quatre chapitres. Afin de permettre aux lecteurs de se familiariser avec le domaine considéré, les deux premiers chapitres introduisent les concepts utilisés. Alors que, les deux derniers chapitres sont consacrés à la description de l'approche proposée qui représente l'essentiel de notre contribution.

Le premier chapitre, présente une étude détaillée sur le problème de la sécurité dans les systèmes d'agents mobiles. Nous décrivons en premier lieu, le paradigme d'agents mobiles, en mentionnant les définitions les plus usuelles, leurs intérêts, leurs domaines d'application ainsi que quelques problèmes pouvant être rencontrés lors de leurs utilisations. Ensuite, nous détaillons l'aspect sécuritaire de cette technologie. Nous énumérons les différentes attaques possibles ainsi que les différentes contre-mesures envisagées.

Le second chapitre est consacré à l'explication de la notion d'adaptabilité, le concept clé sur lequel se fonde notre approche ainsi que la réflexivité qui est le modèle de conception adopté. Nous expliquons ces deux techniques, leurs principales catégories et les différents concepts utilisés. Nous achevons ce chapitre en mentionnant les travaux qui ont utilisé les deux techniques (l'adaptabilité et la réflexivité).

Au niveau du chapitre 3, nous exposons l'approche de protection de l'agent mobile que nous proposons. Nous cernons la problématique à laquelle nous nous intéressons ainsi que l'objectif que nous recherchons. Nous développons ensuite notre stratégie de protection. Nous essayons de l'illustrer à travers une structure au niveau de laquelle nous désignons les fonctionnalités de base que nous expliquerons profondément.

Le chapitre 4 traite une étude de cas. Un domaine bien précis a été choisi sur lequel sont projetés les principaux aspects de notre approche. De plus, une implémentation faisant apparaître les aspects techniques de l'approche est présentée.

Enfin, ce mémoire s'achève avec une conclusion générale au niveau de laquelle, une évaluation du travail élaboré est réalisée et des perspectives sont suggérées.

Chapitre I

LA SÉCURITÉ DANS LES SYSTÈMES D'AGENTS MOBILES

«La magie d'hier est la science d'aujourd'hui et demain elle sera la technologie»

Ce chapitre permet aux lecteurs de se familiariser avec le domaine de la sécurité dans les systèmes d'agents mobiles. Pour ce faire, nous commençons par présenter une description des systèmes d'agents mobiles, permettant d'éclaircir les éventuelles ambiguïtés dans ce paradigme. Ensuite, nous étudierons, l'aspect sécuritaire dans ce domaine, pour pouvoir nous situer par la suite et préciser le cadre de notre travail. Enfin, nous survolons quelques travaux ayants traité ce problème, et à partir desquels nous essayerons de nous inspirer en guise de pouvoir élaborer et dresser notre contribution dans ce domaine.

INTRODUCTION

Le monde de l'informatique a subi plusieurs bouleversements dans son histoire. L'une des plus récentes est la banalisation des réseaux et l'émergence du World Wide Web. L'informatique devient ainsi de plus en plus distribuée. On est ainsi naturellement conduit à chercher à donner plus d'autonomie et d'initiative aux différentes applications.

La communauté d'administration des systèmes et des réseaux informatiques a reconnu le fort potentiel qu'offrent le paradigme d'agents et en particulier, des agents mobiles, pour répondre à ces besoins. Cependant, cette technologie manque encore de maturité et pose de sérieux problèmes. Et notamment celui de la sécurité la sécurité représentant un point crucial pour son utilisation. Malheureusement, aucun travail ne peut définir de manière exhaustive les mesures sécuritaires applicables car la sécurité change d'un système à un autre, d'où la nécessité de procéder à une analyse soigneuse avant de choisir un mode sécuritaire pour une application précise.

Dans un système d'agents mobiles, les plateformes, par le fait qu'elles exécutent le code des agents mobiles, sont menacées par leurs attaques. D'autre part, lors de leurs déplacements à travers le réseau, les agents mobiles peuvent visiter des plateformes malveillantes et sont vulnérables aux nombreuses attaques qui peuvent nuire à leur bon fonctionnement. De nombreux travaux de recherche ont vu le jour pour traiter le problème de la sécurité dans les systèmes d'agents mobiles.

Dan ce chapitre nous exposons le problème de la sécurité dans les systèmes d'agents mobiles. Dans le but d'une meilleure compréhension**,** nous étudions en premier lieu, le paradigme d'agents mobiles, afin de pouvoir les traiter par la suite. Nous commençons par définir le mot agent dans ses significations usuelles, puis nous mettons en place une taxinomie des différents types d'agents existants. Nous détaillons le paradigme d'agents mobiles, en mentionnant son intérêt, son domaine d'application, sa structure, son infrastructure, ainsi que quelques problèmes rencontrés lors de sa manipulation. Par la suite, nous étudierons l'aspect sécuritaire dans le cadre des systèmes d'agents mobiles. Nous verrons les différentes attaques possibles. Nous décrivons ainsi, les différentes mesures de sécurisation envisageables après avoir dénoter les besoins de la sécurité requis. Ensuite, nous exposons les approches les plus connues ayant traité le problème dont nous parlons. Enfin, nous achevons le chapitre avec une conclusion.

I) LE PARADIGME D'AGENTS MOBILES

1 La notion d'agent

Il n'existe pas encore un consensus sur la définition d'un agent. Cela est du principalement à la relative jeunesse du domaine. Nous reprenons la définition de *Ferber* [20].

1.1 Définition:

« *Un agent est une entité logicielle ou matérielle, à qui est attribuée une certaine mission qu'elle est capable d'accomplir de manière autonome, disposant d'une connaissance partielle de ce qui l'entoure (son environnement), et agissant par délégation pour le compte d'une personne ou d'une organisation*».

Cette définition assez générale insiste sur deux caractéristiques importantes:

(i) l'agent est une entité qui agit par *délégation*, l'agent doit respecter la stratégie du client vis à vis des choix qu'il est amené à faire, cela afin que le client soit responsable des tâches effectuées par l'agent.

(ii) l'agent est une entité *autonome* qui dispose de son propre environnement, c'est à dire que l'agent n'est pas lié constamment au client qui l'utilise.

1.2 Taxonomie

Nous trouvons dans la littérature une multitude de définitions d'agents qui se ressemblent toutes, mais se diffèrent selon le type d'application pour laquelle est conçu l'agent [11,18,39,45].

Agent matériel: c'est un agent doté d'un équipement physique, tel un robot. La réalité physique d'un robot apporte des problèmes spécifiques (imprécision de la perception et de l'action, aspects temps réel,…) qui rendent particulièrement difficile, mais aussi particulièrement riche, la conception de tels agents robotiques. On peut aussi envisager des coopérations entre robots, par exemple dans le cadre de la compétition, les agents footballeurs, présentant un exemple de problèmes de l'intelligence artificielle.

Agent logiciel: Un agent logiciel est purement logique, incluant du code, des données, et un état. Les premiers et les plus simples sont les démons Unix (processus informatiques autonomes capables de se réveiller à certaines heures ou en fonction de certaines conditions). Les virus informatiques en sont des versions déjà plus sophistiquées (notamment douées de la capacité de reproduction) et malfaisantes.

Agent réactif. Cet agent est marqué par un comportement basé sur le principe de stimulus-Réponse. La structure des agents purement réactifs tend à la simplicité, mais ces derniers peuvent être capables d'actions de groupe complexes et coordonnées. Les agents de ce type sont habituellement de petite taille et de grand nombre dans l'environnement. Ils ne sont pas nécessairement intelligents, néanmoins des comportements collectifs intelligents peuvent émerger. La communication entre ces agents se fait principalement par des traces ou des signaux.

Agent cognitif : Un agent est cognitif ou encore intentionnel, s'il ne se contente pas d'agir en fonction des conditions de son environnement mais en fonction de ses buts propres et s'il possède une intentionnalité, c'est-à-dire une volonté consciente d'effectuer un acte. Un tel agent doit posséder une certaine représentation de son environnement. Un agent intentionnel et un agent purement réactif ne sont pas antagonistes. Par exemple, un agent intentionnel peut tout à fait réagir à des stimulus de son environnement tout en conservant une haute représentation de celui-ci et en agissant en fonction de ses intentions. Un exemple d'agent intentionnel est décrit par le modèle *BDI** (voir la section 7.1).

Agent mobile: C'est un agent matériel capable de se déplacer dans l'environnement ou bien logiciel pouvant migrer d'un site à un autre en cours d'exécution en fonction de ses besoins et suit parfois un itinéraire; il peut donc suspendre son exécution sur une machine et migrer vers une autre où il reprend son exécution là où il s'était arrêté avant de migrer. Un agent mobile dispose donc de dispositifs assurant sa mobilité.

Agent adaptable: un agent est dit *adaptable* si certains de ses mécanismes internes, opérationnels (envoi de messages, déplacement...) ou fonctionnels (comportement), sont modifiables en cours d'exécution. Conformément à la propriété d'autonomie, l'agent contrôle lui-même ses propres évolutions.

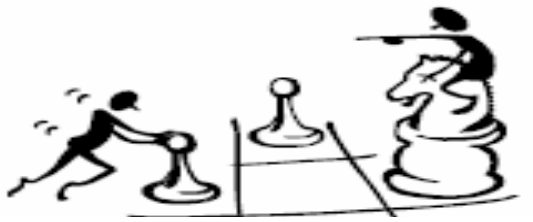
Agent social: ce type d'agents opère des interactions évoluées avec d'autres agents au sein de son environnement. Ces interactions peuvent faciliter la tâche d'un agent (coopération) ou au contraire le gêner dans l'accomplissement de ses buts (encombrement, compétition).

Notons que ces différents types possèdent plus ou moins de recouvrements entre eux. Nous constatons également qu'ils cadrent plus ou moins la définition que nous avons donnée plus haut, car cela dépend des caractéristiques mises en avant selon le domaine en question. En fait, il n'existe pas de compétition entre les différents paradigmes, mais plus tôt, des opportunités de mise en œuvre complémentaire. Le choix dépend de l'architecture et du but poursuivi.

*BDI : Belief Desir Intention

1.3 Système multi-agent

« Un système multi-agent est un ensemble organisé d'agents ». Nous ne faisons que suivre ici la définition usuelle du terme système : (un ensemble organisé d'éléments). Cela signifie que dans un système multi-agent, il existe une ou plusieurs organisations qui structurent les règles de cohabitation et de travail collectif entre agents sociaux (définition des différents rôles, partages de ressources, dépendances entre tâches, protocoles de coordination, de résolution de conflits, etc.). Ce type de système peut se révéler utile pour certains phénomènes complexes dont on ne connaît pas de modèle global [11].



2. Présentation de la technologie d'agents mobiles

Le client donne une mission à un agent. Pour la réaliser, l'agent se déplace dans le réseau de machines accédant localement aux services offerts par ces machines. On peut distinguer trois phases :

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

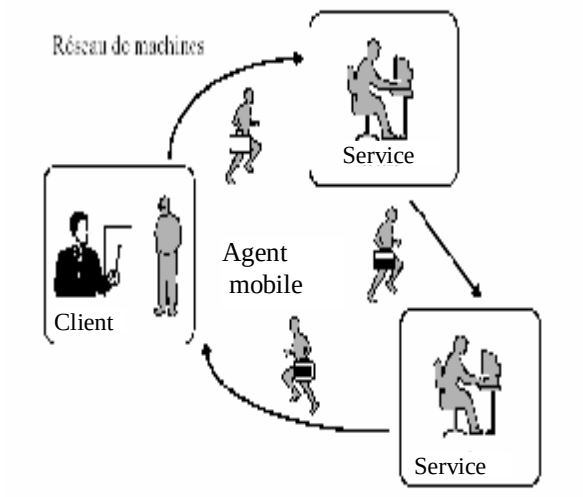


Fig 1.1 - Le paradigme des agents mobiles [45]

2.1 Caractéristiques des agents mobiles

Un agent mobile doit répondre aux propriétés suivantes [45]:

- *Mobilité :* la capacité d'un agent à se déplacer dans un réseau informatique.

- *Autonomie :* les agents opèrent sans intervention directe d'être humain ou autre, et ont un certain contrôle sur leurs actions et leur état interne.

- *Réactivité :* les agents perçoivent leurs environnements et répondent aux changements qui apparaissent.

- *Comportement intentionnel (la pro-activité):* les agents sont capables d'avoir un comportement dirigé vers un but et de prendre des initiatives.

- *Comportement social :* les agents interagissent avec d'autres agents (éventuellement humains) via une sorte de *"langage de communication agent*".

- *Rationalité :* la conjecture selon laquelle un agent agira de façon à atteindre ses objectifs, au moins dans la limite de ses convictions.
- *Véracité :* la conjecture selon laquelle un agent ne communique pas de mauvaises informations sans le savoir.

Notons que ces caractéristiques sont plus ou moins accentuées selon le domaine considéré, le but recherché et la stratégie adoptée. Toutefois, la mobilité est intrinsèque dans ce paradigme. Nous distinguons deux types de mobilité: la mobilité faible et la mobilité forte [45].

- *Mobilité faible* (non transparente) : L'agent doit préparer sa migration en sauvegardant son état dans des variables et une fois sa migration effectuée, il doit restaurer son état et reprendre son exécution, en général à un nouveau point du programme.
- *Mobilité forte* (transparente) : L'agent après sa migration repart dans le même état et exactement au même point dans son code que celui avant migration.

2.2 Motivations d'utilisations du paradigme d'agents mobiles

Plusieurs raisons peuvent expliquer l'engouement soudain pour cette technologie [59].

- De nombreux problèmes sont par nature distribués.
- les technologies réseau ont reçu de considérables attentions de la part des chercheurs et des techniciens, rendant ainsi la généralisation des agents mobiles envisageable.
- La vitesse des transmissions, la sécurité des communications et la standardisation des protocoles rendent plus facile leur mise en oeuvre.
- L'accroissement du pouvoir d'expression des calculs formels autorise la modélisation d'architectures distribuées indispensable à la notion d'agents mobiles.
- D'autre part, des langages de haut niveau (Java, C++, …) permettent l'écriture rapide de système d'agents mobiles.
- En plus, la migration des agents mobiles à des sites distant permet la confrontation de plusieurs experts, permettant ainsi de bénéficier et d'appendre de leurs savoir faire.

3. Comparaison entre les paradigmes "Client/Serveur" et "Agent mobile"

Dans le paradigme**:** "*client/serveur"*, un client demande un service auprès du serveur, d'une façon interactive et synchrone, ceci signifie que le client est bloqué tant que le serveur n'a pas répondu. Tandisqu'au niveau du modèle "*agent mobile*", l'agent se déplace d'une façon autonome entre les machines d'un réseau pour réaliser certaines tâches localement. Ces agents s'exécutent d'une façon asynchrone ce qui permet de dire que ce paradigme est mieux adapté que le client/serveur à des traitements longs et nécessitant des interrogations fréquentes du serveur [45].

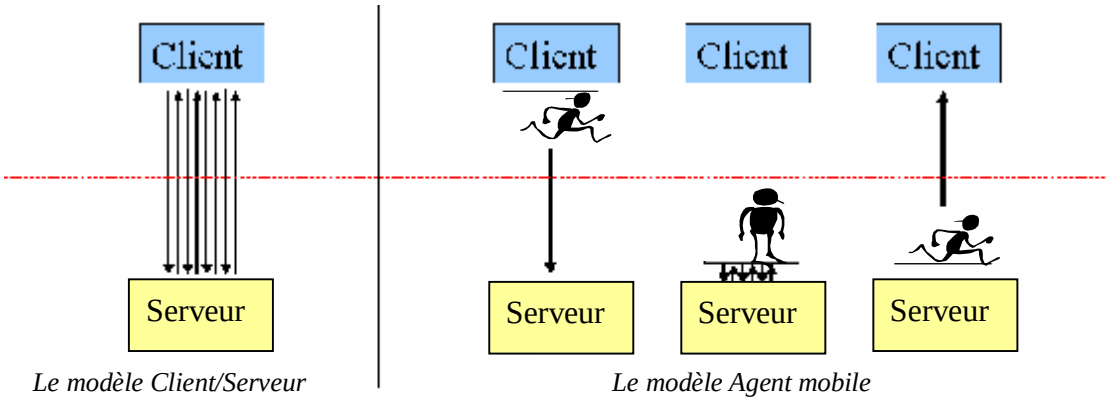


Fig 1.2- Client/Serveur Versus Agent mobile

3.1 Intérêts des agents mobiles

Parmi les intérêts de l'utilisation des agents mobiles [8, 47] :

- *Réduction de la charge dans le réseau:* Dans le cas des applications faisant des interrogations très fréquentes de la base de données, il serrait certainement moins coûteux de faire migrer le code de traitement que les données à traiter, qui peuvent être beaucoup plus volumineuses. Ceci induit une minimisation des communications distantes.
- *Exécution asynchrone et autonome:* Le client peut faire autre chose pendant que l'agent mobile accompli sa mission -voire même être *éteint*-.
- *Efficacité & souplesse du traitement :* La mobilité permet plus d'efficacité et de souplesse, en privilégiant les interactions locales évitant ainsi le transfert de données intermédiaires et permettant plus de confidentialité de traitement.
- *Adaptation dynamique à l'environnement :* Les agents mobiles adaptatifs savent détecter les changements dans leur environnement, y réagir de manière autonome et s'adapter en conséquence en se déplaçant vers des endroits plus rationnels pour leurs taches.
- *Robustesse et tolérance aux fautes:* Lorsqu'un système/machine hôte est en difficulté (rupture de communication,…), les agents mobiles visiteurs prévenus ont la possibilité de se dispatcher ailleurs dans le réseau.
- *Performance:* La nature du paradigme d'agent mobile supportant la décomposition du travail, la modularité, la réutilisabilité et le parallélisme, procure un traitement performant et compétitif. Le système est plus *robuste* car réparti et parallélisé. De plus, une partie de l'application s'exécute dans le réseau et peut bénéficier des ressources importantes.

4. Domaines d'applications

Nous avons montré que les agents mobiles offrent des perspectives intéressantes. Nous allons donner une liste non exhaustive d'applications de ces agents tout en montrant leurs utilités [27].

Le commerce électronique : C'est une application qui se déroule au sein d'un réseau de places de marché électroniques où des agents se rencontrent afin de proposer, rechercher et négocier différents types de services, pour le compte d'utilisateurs individuels ou de sociétés (réservations, achat de biens divers...). Ce type d'application, met en évidence deux types d'agents : agents vendeurs et agents acheteurs. Un agent vendeur est un agent chargé de vendre un produit. Il voyage de serveur en serveur à la recherche d'agents acheteurs de ce même produit. Une fois une rencontre effectuée, un algorithme de marchandage peut se mettre en oeuvre. Si un accord d'achat est trouvé, il est réalisé directement ou/et un courrier électronique est envoyé aux propriétaires de ces agents.

La recherche d'informations sur le Web : La quantité d'informations disponible sur le Web croît sans cesse que les outils de recherches disponibles deviennent inadaptés. Ainsi la recherche de l'information pertinente sur le Web devient une tache pénible et ennuyeuse. Une solution consiste en la délégation de cette mission à un ou plusieurs agents qui visitent les sites Web, et qui coopèrent et recherchent les sites d'intérêts, puis rentrent avec les meilleurs résultats.

L'équilibrage de charge dynamique: En parallélisme, dans les machines de type *MIMD**, l'enjeu principal dans l'écriture d'un programme efficace est l'équilibrage de charge. Cet équilibrage est difficile à résoudre sur des problèmes très irréguliers. Les agents mobiles permettent d'équilibrer la charge d'une machine : si un noeud est en avance sur son travail tandisqu'un autre est en retard, un ou plusieurs agents peuvent migrer du dernier au premier réduisant ainsi l'écart.

La mises à jour: Le problème de la mise à jour des multiples copies, dans certains cas, peut être résolu par un agent dont le rôle est d'apporter la modification de site en site, selon un chemin prédéterminé.

5. Composition d'un modèle d'agents mobiles

Un modèle à agents mobiles comprend deux concepts fondamentaux; l'agent et son environnement appelé système d'agent ou encore plateforme d'agents [45].

**MIMD : Multiple Information Multiple Data*

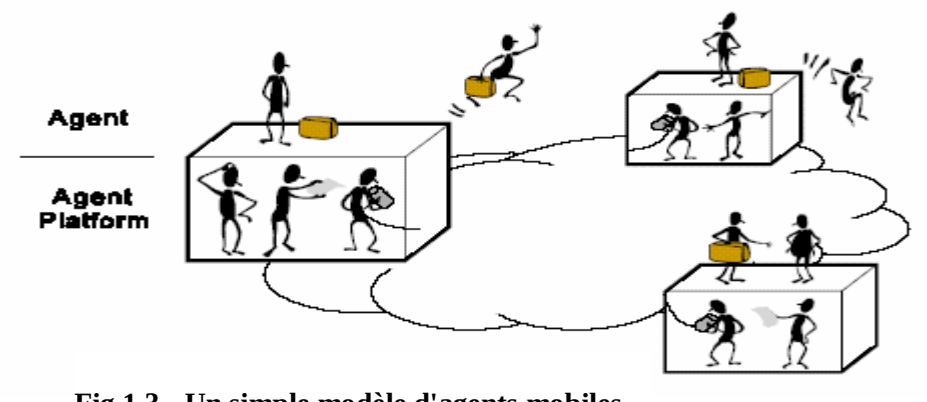


Fig 1.3 - Un simple modèle d'agents mobiles

5.1 L'agent mobile

Une entité qui possède cinq attributs. Il les transporte quand il se déplace à travers le réseau; un état, une implémentation, une interface, un identifiant et une autorité [45].

- *L'état* : L'état d'un agent peut être considéré comme une photo instantanée de son exécution. Il permet à l'agent de reprendre son exécution quand il arrive à destination.
- *L'implémentation*: l'agent mobile a besoin d'un code pour pouvoir s'exécuter. Quand il se déplace à travers le réseau, l'agent peut soit emporter son code soit aller à destination, voir quel code est disponible sur la machine distante et récupérer le code manquant à partir du réseau (c'est la technique du "*code-on-demand*").
- *L'interface:* Un agent fournit une interface qui permet aux autres agents et aux autres systèmes d'interagir avec lui.
- *L'identifiant :* Chaque agent possède un identifiant unique durant son cycle de vie, qui lui permet d'être identifié et localisé.
- *L'autorité :* Une autorité est une entité dont l'identité peut être authentifiée par n'importe quel système auquel elle essaye d'accéder. Il existe principalement deux types d'autorité, le fabricant (manufacturer) qui est le fournisseur du code d'implémentation de l'agent, et le propriétaire (owner) qui a la responsabilité du comportement de l'agent.

5.2 Le Serveur d'agent/ système d'agent

a. Définition d'un serveur d'agents: Un "serveur d'agents" ou encore une "agence" est le contexte d'exécution d'agents, là où un agent exerce son activité et le destinataire d'une éventuelle migration ne peut-être qu'un serveur d'agents [59].

b. Définition d'un système d'agents: Un système d'agents est une plateforme qui peut créer, interpréter, exécuter, transférer et terminer un agent. Il peut être un réseau de serveurs d'agents. Autrement dit un système d'agents est la vision utilisateur de l'ensemble des serveurs et des agents sur un réseau [59].

Notons que les termes de serveur d'agents, système d'agents et réseau de système d'agents sont souvent confondus dans la littérature.

De la même manière qu'un agent, un système d'agent est associé à une autorité qui identifie la personne ou l'organisation pour laquelle il agit. Un système d'agent est identifié par son nom et son adresse. Une machine hôte peut contenir plusieurs systèmes d'agent. Cinq concepts jouent un rôle important dans le système d'agent [45].

- *La place* : Une place est un contexte au sein d'un système d'agent, dans lequel un agent s'exécute. Ce contexte peut fournir un ensemble de services uniformes sur lesquels l'agent peut compter indépendamment de sa localisation spécifique. La place de départ et la place de destination peuvent être situées au sein d'un même système d'agent ou sur des systèmes d'agents différents. Un système d'agent peut contenir une ou plusieurs places.
- *Le type du système d'agent :* Le type d'un système d'agent permet de définir le profil d'un agent. Par exemple, si le type d'un système d'agents est "AGLET", alors le système d'agent est implémenté par IBM et supporte Java comme langage de programmation.
- *Les ressources :* Le système d'agent et la place fournissent un accès contrôlé aux ressources locales et aux services (base de données, processeurs, mémoire, disques).
- *La localisation :* On définit la localisation d'un agent comme étant la combinaison de la place dans laquelle il s'exécute et l'adresse réseau du système d'agent où réside la place. Concrètement, elle est définie par l'adresse IP et le port d'écoute du système d'agent avec le nom de la place comme attribut.
- *La région :* Une région est un ensemble de systèmes d'agent qui appartiennent à la même autorité mais qui ne sont pas nécessairement de même type.

6. Infrastructure pour un système d'agents mobiles

Pour être opérationnel, l'agent a besoin d'interagir avec l'hôte sur lequel il s'exécute, ainsi que les autres agents. Il doit avoir accès aux informations qu'offre l'hôte, ou négocier avec les autres agents. Les agents doivent aussi être capables de migrer à travers le réseau d'ordinateurs hétérogènes. Ceci n'est possible que s'il existe un cadre de travail commun pour tout le système d'agents: Une infrastructure d'agent standard doit offrir les supports de base pour assurer: la création, la destruction, l'exécution, la communication, la migration, le contrôle de l'utilisation des ressources… ainsi que la prise en compte de la sécurité [45].

6.1 Création d'un agent mobile

Un agent est créé dans une place. La création peut être initiée soit par un autre agent résidant dans la même place ou par un agent ou un système non_agent en dehors de la place. Le créateur doit s'authentifier dans la place et établir les autorisations et les références que va posséder l'agent. La classe de définition nécessaire pour instancier l'agent peut résider dans l'hôte local ou sur une machine distante ou fournie par le créateu.

6.2 Destruction d'un agent mobile

Dans la plupart du temps, l'agent est tué lorsqu'il quitte sa place. Ce processus peut être initié par l'agent lui-même, ou par un autre agent ou système non-agent résidant ou non dans la même place. L'agent peut aussi être renvoyé de sa place pour l'une des raisons suivantes:

- La fin de son cycle de vie,
- L'agent n'est pas utilisé ou référencé,
- Violation des règles de sécurité,
- Le système est arrêté.

6.3 Transfert des agents mobiles (Migration) :

Le processus de transfert peut être initié par l'agent lui-même, par un autre agent résidant dans la même place ou par un agent ou système non agent résidant dans une autre place. Le processus de transfert va être détaillé dans ce qui suit :

L'envoi de l'agent : Quand l'agent mobile se prépare pour son voyage, il doit être capable d'identifier sa destination. Si la place est non spécifiée, l'agent va s'exécuter dans la place par défaut sélectionnée par le système d'agent de destination. Le processus se fait comme suit: *<Suspendre l'agent, le Sérialiser*, l'Encoder, puis le Transférer>*.

Réception de l'agent : par analogie, le processus de réception se fait comme suit: *<Recueillir l'agent, Le décoder, Le désérialiser**, puis reprendre l'exécution>*.

Fig 1.4- Processus de migration de l'agent mobile [45]

* *Sérialiser : Sauvegarder l'état.*
** *Désérialiser : Restauter*

6.4 Relations avec le site hôte

Deux possibilités sont envisageables, pour que l'agent mobile communique avec l'hôte client.

- Soit on crée une application interface qui se charge des lectures/écritures sur la console.
- Soit les agents eux-mêmes s'en chargent ; mais alors ils doivent être liés au gestionnaire de fenêtres, au gestionnaire d'entrée/sortie, ainsi que celui du réseau.

6.5 Communication entre les agents mobiles

Les agents peuvent communiquer avec d'autres agents résidents dans la même place ou avec des agents résidents dans d'autres places. Un agent peut évoquer une méthode d'un autre agent comme il peut lui envoyer des messages s'il est autorisé à le faire dans un certain langage prédéfini. Un langage de communication agents (LCA) fournit aux agents le moyen d'échanger des informations et des connaissances. Un message dans un LCA décrit un état désiré dans un langage déclaratif basé sur les actes de discours. les langages d'interaction comme FIPA et KQML sont le plus souvent définis indépendamment des modèles ou des types des systèmes d'agents utilisés, ce qui permet aux concepteurs de choisir leurs propres architectures d'agents.

7. Existant des systèmes d'agents

7.1 Exemple d'architectures existantes

L'architecture BDI *:*

C'est architecture basée sur le raisonnement pratique. Les intentions jouent un rôle central dans ce modèle [22].

Cette architecture est composée de:

- Structures de Données représentant les Croyances, désirs et intentions de l'agent,
- Les fonctions qui représentent les délibération des croyances, des désirs et des intentions, et
- Le raisonnement moyens-fins : décider comment faire.

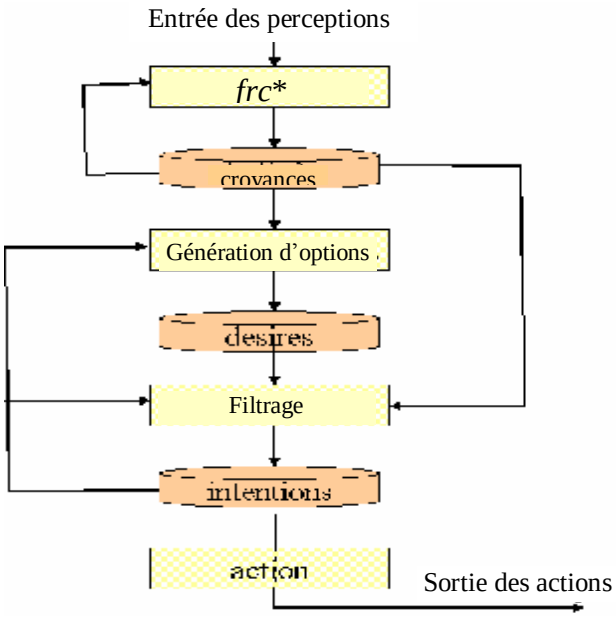


Fig 1.5- Architecture d'agent BDI [22]

**frc : fonction de révision des croyances*

7.2 Exemples de plateformes existantes

Cette partie tente de donner une idée sur les plateformes existantes permettant le développement et la manipulation des agents mobile. Cette tâche est assez difficile pour plusieurs raisons:

- Cette technologie est en plein essor bien que la notion d'agent soit relativement ancienne
- l'évolution très rapide du domaine ne permet pas d'avoir toujours une information actuelle.
- Il y a plusieurs modèles pour décrire le comportement des agents. Chacun fait l'objet de recherches et de documents qui ne permettent pas d'avoir une vue globale et homogène.
- La maîtrise technologique des agents est un enjeu économique et stratégique très important, ce qui explique la difficulté de trouver une normalisation acceptable.

Il existe à l'heure actuelle beaucoup d'implémentations différentes incompatibles entre elles. Certaines écrites en Java pur, bénéficiant de fait du portage multi-platformes. Citons :

Odyssey [65] (de General Magic*)*, Une des premières plateformes d'agents. Il est indépendant du protocole de transport utilisé et peut donc s'adapter à RMI (de Sun*)*, CORBA (de OMG*)* ou DCOM (de Microsoft*)*.

Aglets [63] d'*IBM*, utilisent la notion de référence distante (AgletProxy), le passage de messages (multicast, broadcast) et le protocole ATP développé par *IBM*. Il offre en outre une certaine notion de sécurité en limitant les ressources allouées aux aglets.

Dima [64] du laboratoire *LIP6*, est une plateforme pour les systèmes multi agents permettant le développement de différents types d'agents (réactif, cognitif et hybride). Cette plateforme ne supporte pas la mobilité. Cette dernière doit être programmée.

Jade [66] du CSELT de l'université de Parma, est un logiciel qui simplifie la mise en place des systèmes multi-agents par grâce à une interface personnalisée qui répond aux spécifications de la Fondation pour les Agents Intelligent Physique -FIPA- et grâce à un ensemble d'outils qui supportent la résolution des bugs. Jade propose en plus des services de sécurité.

Enfin, les systèmes suivants non écrit en Java peuvent être digne d'intérêt [59] :

Telescript (General Magic), **Agent Tcl** (Dartmouth College), **M0** (Univ. of Geneva), **Obliq** (DEC SRC), **Tacoma** (Univ. of Norway & Cornell Univ).

Malheureusement, un agent écrit dans un de ses systèmes ne peut migrer vers un système différent. C'est pourquoi plusieurs tentatives de standardisation ont vu le jour.

8. Quelques problèmes rencontrés dans le paradigme d'agent mobile

L'hétérogénéité des systèmes, le maintien des canaux ouverts et des communications entre agents ainsi que leur sécurité sont les principaux problèmes rencontrés dans un ces systèmes [45].

(i) La migration de l'agent dans un système de machines hétérogènes pose un problème car la représentation du code et des données du processus, est dépendante de la machine sur laquelle il s'exécute. Dans le cas de machines avec des architectures différentes, l'état d'un agent risque de ne pas être compris. Les solutions proposées sont :

- la traduction de l'état de l'agent de la représentation source à la représentation finale, ce qui implique autant de traducteurs que de couples d'architectures différentes
- la représentation de l'état dans un format standard intermédiaire, indépendante de la machine et compris par toutes les machines.

(ii) Que se passe-t-il quand un agent qui accède à un fichier local est déplacé sur une autre machine? Le descripteur de fichier n'a plus de valeur sur la nouvelle machine.

Une solution serait de créer un lien de poursuite entre le nouveau site et le site source. Ce lien permettra au processus d'accéder à distance au canal ouvert. Mais cette solution ne devient plus efficace lors d'une panne sur la machine source.

Une autre solution serait de fournir une identification standard des canaux ouverts qui auraient le même identificateur sur toutes les machines.

(iii) Le maintien des communications entre agents est un autre problème rencontré lors de la migration de l'agent. En effet, que se passe-t-il lorsqu'un message arrive à un agent en cours de migration?

Une solution est de considérer que l'état de agent en cours de migration est un état particulier dans lequel il ne peut recevoir de messages. Dans ce cas il devra prévenir les autres agents.

(iv) Un autre problème qui ne cesse de freiner l'expansion des agents mobiles, est celui de la sécurité. Ce sérieux problème présente un centre d'intérêt de multiples travaux de recherches à l'heure actuelle. Cependant, la problématique reste encore ouverte nécessitant ainsi plus d'attention de la part de la communauté de chercheurs et des connaisseurs du domaine.

Nous allons étudier et détailler ce problème dans ce qui va suivre.

II. L'ASPECT SÉCURITAIRE DES SYSTÈMES D'AGENTS MOBILES

Nous ne pouvons évidemment déterminer avec précision dans quelle mesure nous serons confronter à quels risques, puisque la probabilité d'occurrence des risque change selon la situation du système. Mais, il est primordial d'avoir une vision globale de la nature des risques afin de se protéger efficacement et de pouvoir mettre en place une défense efficace et réaliste.

1. Risque & Attaque

Risque: Le risque en terme de sécurité est la probabilité qu'une menace particulière puisse exploiter une vulnérabilité du système. Il est généralement caractérisé par l'équation suivante:

Risque = (Menaces * Vulnérabilités) / Contres *mesures* [46]

Menace: Une menace est un danger qui existe dans l'environnement du système.

Vulnérabilité: Une vulnérabilité est une faiblesse du système qui le rend sensible à une menace. Autrement dit; c'est le niveau d'exposition face aux menaces. Exemple: les bugs dans les logiciels et les mauvaises configurations, les erreurs humaines, services permis et non utilisés, saturation de la liaison d'accès à l'Internet…

Contre mesures: L'ensemble des actions mises en œuvre en prévention des menaces. Ils ne sont pas uniquement des solutions techniques mais également des mesures de formation et de sensibilisation à l'intention des utilisateurs.

Attaque : elle représente l'ensemble des actions de l'environnement pouvant menacer un certain système et entraîner des pertes. On distingue deus types d'attaques:

Passive : elle ne modifie pas l'information et porte essentiellement sur la confidentialité. Exemple: Espionnage, Copie illicite…

Active : elle modifie le contenu de l'information ou le comportement du système. Exemple: Altération, dénis de service…

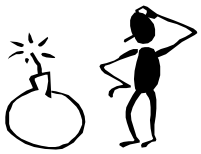
2. Les différentes attaques dans un système d'agents mobiles

On trouve dans la littérature une multitude de classifications des attaques dans les systèmes d'agents mobiles. Mais on distingue principalement deux grandes classes d'attaques

2.1 Les attaques des plateformes malveillantes:

F. Hohl définit une plateforme malveillante comme étant une plateforme capable d'exécuter un agent provenant d'une autre plateforme, et qui essaye de nuire au bon fonctionnement de l'agent mobile d'une quelconque manière. On a identifié les attaques suivantes d'une plateforme contre un agent mobile [1,9,28]:

- Mascarade.
- Espionnage.
- Déni de service.
- Altération.
- Exécution incorrecte du code.



Mascarade *(Masquerade) :* Une plateforme malveillante peut masquer son identité en trompant un agent mobile et s'identifier comme sa vraie destination. La mascarade est une des attaques qui introduisent d'autres types d'attaques; tels que l'espionnage, l'extraction des informations secrètes, l'exécution incorrecte,… Une plateforme qui se masque comme une troisième réception de confiance, peut en plus de leurrer des agents naïfs, nuire aux plateformes dont l'identité a été exploité.

Espionnage *(Eavesdropping) :* L'espionnage ou bien l'analyse est une attaque classique appelée aussi L'écoute aux ports. Elle implique l'interception de communications secrètes. Cependant, la menace de ce type d'attaque est exacerbée dans les systèmes d'agents mobiles du fait que la plateforme qui reçoit et exécute l'agent mobile contrôle en plus des communications de l'agent mobile, chaque instruction exécutée par ce dernier, et toute donnée apportée à la plateforme, ou produite lors de l'exécution [28]. Lorsque l'accès directe aux renseignements secrets de l'agent mobile devient impossible ou bien à la rigueur très coûteux à cause de certains mécanismes de sécurisation par exemple, elle essayera d'inférer la signification des services et la stratégie adoptée par l'agent mobile en analysant son comportement. Par exemple, un agent qui communique avec un agent du voyage, bien que le contenu du message ne puisse être exposé, cette communication peut indiquer que l'agent organise un voyage et sera loin de sa maison dans le futur proche.

Déni de service *(Denial of Service) :* Quand un agent arrive à une plateforme cliente, il attend qu'elle exécute les demandes de l'agent fidèlement, en lui allouant les ressources demandées, et se conformer aux contraintes exigées dans le contrat mis en accord. Cependant, une plateforme malveillante peut ignorer les demandes de l'agent

mobile, introduire des délais inacceptables pour les tâches critiques, ou simplement ne pas exécuter l'agent mobile, ou même terminer l'agent sans notification dans le but de gâcher le travail de l'agent mobile.

Alteration *(Alteration) :* Quand un agent arrive sur une plateforme cliente, il expose son code, son état, et ses données. De ce fait, cette plateforme aura l'habilité de les modifier. Elle peut modifier les données, aussi bien les résultats obtenus par l'agent mobile à l'intérieur des plateformes précédentes, que les données propres à l'agent mobile. La plateforme peut également effectuer une attaque en modifiant l'état de l'agent. Elle peut conduire le comportement de celui-ci en modifiant son état d'exécution. Cette modification peut être d'une manière permanente en implémentant des virus, des vers ou des chevaux de Troie afin d'attaquer ainsi les autres plateformes se trouvant dans l'itinéraire de l'agent mobile; ou bien temporaire en modifiant seulement la manière selon laquelle la plateforme exécute l'agent mobile.

Exécution incorrecte du code : Sans avoir à changer ni le code ni l'état d'exécution de l'agent mobile, une plateforme malveillante peut modifier la façon dont elle exécute le code de l'agent. Elle peut par exemple retourner une fausse valeur quand elle procède à la comparaison de son prix avec un prix nominal fixé par la plateforme d'origine. Ou bien ré-exécuter l'agent après avoir copié, une partie de lui ou un message de celui-ci. Par exemple, une plateforme malveillante peut ré-exécuter un message d'achat d'un produit ou de payement d'une facture, ou tout simplement ré-exécuter l'agent mobile avec des données différentes.

2.2 Les attaques des agents malveillants

Les attaques des agents mobiles peuvent prendre différentes formes. On distingue principalement quatre types d'attaques [1, 9,28]:

- Mascarade
- Accès non autorisé.
- Déni de service.
- Répudiation

Mascarade *(Masquerade) :* La mascarade se manifeste quand un agent malveillant masque son identité et se sert de l'identité d'un autre agent dans le but d'obtenir des privilèges dont il n'a normalement pas le droit, ou bien pour causer des dommages à la plateforme cliente et de faire endosser la responsabilité à un autre

agent. Il peut par conséquent détruire la confiance établie dans une communauté d'un système d'agents légitime.

Accès non autorisé *(Unauthorized access) :* Les mécanismes du contrôle de l'accès sont utilisés pour prévenir des utilisateurs non autorisés ou des processus d'accéder aux services et aux ressources dont ils n'ont pas été accordés l'autorisation comme il a été spécifié par la politique de sécurité adoptée. Chaque agent visitant une plateforme doit être soumis à sa politique de sécurisation. Appliquer les mécanismes du contrôle de l'accès adéquats exige que l'agent certifie l'identité avant d'accéder à la ressource sollicitée. Un agent qui arrive à détenir des privilèges dont il n'a pas le droit, peut nuire aux autres agents aussi bien qu'à la plateforme elle-même. Une plateforme qui accueille des agents représentant plusieurs utilisateurs et organisations doit assurer un contrôle d'accès garantissant les droits de chaque partie.

Déni de service *(Denial of Service) :* Un *déni de service* a lieu lorsqu'un agent consomme trop de ressources (UCT, bande passante de la connexion au réseau). Ces attaques peuvent être intentionnelles ou non (erreur de programmation). Le problème avec un agent mobile est que le code est en principe écrit en dehors de la plateforme qui l'exécute et il peut contenir du code malveillant.

Répudiation *(Repudiation) :* La *Répudiation* ou encore le *Reniement* se produit quand un agent, participant à une transaction, arrive à causer une rupture de la communication qui a eu lieu entre les éléments du système. La répudiation que se soit causer d'une manière accidentelle ou planifiée, peut mener à de sérieux problèmes qui ne peuvent pas être résolus facilement à moins que les contre-mesures adéquates soient mises en place. Si une plateforme de l'agent ne peut pas prévenir les agents malveillants de répudier une transaction, elle doit être en mesure pour résoudre les désaccords occasionnés.

3. Les conditions de la sécurité

Comme dans tout système d'information, pour assurer la sécurité d'un système d'agent mobile, il faut répondre aux besoins que l'organisation « *ISO* »* a fait ressortir dans ses études sur les sécurités des réseaux. Il s'agit de l'authentification, la confidentialité, l'intégrité, la disponibilité, et la responsabilité. Cette section fournit une vue d'ensemble brève de ces exigences de la sécurité [1,16,46].

** ISO : International Standard Organisation*

3.1 Confidentialité

On parle de *confidentialité* lorsque l'information n'est pas accessible à ceux qui n'en ont pas l'autorisation. Toutes les données secrètes placées sur une plateforme ou transportées par un agent doivent demeurer confidentielles. Les structures de l'agent ainsi que celles de la plateforme doivent être capables d'assurer la confidentialité des données ainsi que les communications pouvant avoir lieu. Un agent mobile peut vouloir garder en secret ses déplacements et ses communications. Il peut utiliser un pseudonyme pour dissimuler son identité dans certain cas (anonymat). Toutefois l'identité doit être fournie lors de l'application des règles de sécurisation exigées par le contrat mis en accord.

3.2 Authentification

L'authentification signifie le processus vérifiant que l'identité de l'utilisateur d'une ressource est comme déclarée. Elle concerne toutes les entités de système : agent ou plateforme. L'authentification est souvent décrite comme le service le plus urgent à intégrer dans les réseaux de télécommunication de nos jours. Elle est aussi indispensable pour d'autres services de sécurité tel que le contrôle d'accès .Il y a trois méthodes permettant la vérification de l'identité. Elles reposent sur la connaissance :

- D'un élément connu de l'individu : qui est à la fois secret et difficile à deviner comme un mot de passe.
- D'un élément possédé par l'individu : peut être une carte magnétique.
- Un élément au sujet de l'individu : ce sont en général les systèmes bio-métriques.

3.3 Intégrité

La propriété d'intégrité assure que l'information n'est modifiée que par les personnes ayant le droit, et de façon volontaire. Ainsi, Les fonctions d'authentification n'ont pas beaucoup d'intérêt si elles ne sont pas couplées d'une fonction d'intégrité; en effet, il est inutile de vérifier l'identité de l'émetteur d'un message reçu si on ne peut pas déterminer si le message a été modifié ou non. Inversement, il importe peu de savoir qu'un message est intact si on ne connaît pas son origine Les mécanismes assurant l'intégrité des données sont étroitement liés à ceux de la confidentialité. La plateforme doit protéger ses agents des modifications non autorisées, et assurer d'un autre côté, que seuls les agents autorisés emportent des modifications aux informations qu'elle expose. Si l'agent ne peut pas prévenir une plateforme malveillante d'altérer ses attributs, il doit à la limite être en mesure de détecter cette altération.

3.4 Contrôle d'accès et responsabilité

Le contrôle d'accès détermine qui doit avoir accès à quelle ressource, autrement dit garantir qu'une ressource n'est accessible que par les entités autorisées. Toutefois, contrôler l'accès ne se résume pas à limiter l'accès. Un bon contrôle d'accès nécessite que les entités autorisées puissent accéder à la ressource demandée très facilement tout en assurant les plus grandes difficultés d'accès aux utilisateurs non autorisés. Ainsi, Chaque entité plateforme ou agent dans le système doit être tenue responsable pour ses actions. Pour cela chacune de ces entités doit être identifiée de façon unique, certifiée, et authentifiée. Cette propriété exige la maintenance d'un fichier "*audit*" d'événements critiques. Ces événements sont définis par la politique de sécurité adoptée. Ces fichiers d'audit doivent être eux mêmes protégées avec soin contre les accès non autorisés et les altérations.

3.5 Disponibilité

Cette propriété pourvoit que les ressources du système soient accessibles par ceux qui en ont besoin à chaque fois qu'ils en ressentent la nécessité. Elle vérifie que l'accès aux différentes ressources du système est possible, une fois que l'identité de l'entité légitime (agent ou plateforme) a été établie et authentifiée .Autrement dit, La disponibilité empêche les données et les services d'être supprimés ou de devenir inaccessibles. L'impossibilité d'accéder à des ressources requises est appelée un refus de service. La plateforme de l'agent doit être capable d'assurer la disponibilité des données et des services aux agents locaux et distants. Celle-ci doit permettre la concurrence d'exécution et les accès simultanés. Elle doit fournir aussi une gestion d'inter-blocage, et les accès exclusifs aux données partagées devant être disponibles sous forme utilisable. Il faut qu'elle soit capable de détecter et de retrouver les échecs du matériel.

4. Les mesures de sécurité

L'identification des objectifs de la sécurité doit conduire à la définition de mesures concrètes pour les réaliser. Ces mesures se classifient en plusieurs types [21,32,54].

4.1 Les mesures de détection

Ces mesures essayent de cibler les menaces potentielles qui se concrétisent par les agressions, et de détecter toute modification non autorisée.

- La détection peut être pendant ou après la réalisation de la modification.
- Elle peut être exécutée d'une manière automatique ou bien à la demande.
- On peut encore envisager la détection de toutes les altérations possibles ou bien seulement quelques unes jugées critiques.

4.2 Les mesures de prévention

Celles-ci ont pour finalité d'empêcher l'aboutissement d'une agression. Elles essayent de rendre impossible –ou à la limite difficile- l'accès ou les modification illégales des informations. Ces mesures peuvent être subdivisées selon le cas où elles:

- préviennent les attaque sur l'agent ou bien l'hôte en entier ou seulement une partie de lui.
- préviennent les attaques en permanence ou seulement temporairement.

4.3 Les mesures de palliation et de correction

Ces dernières pallient ou réparent les dégâts engendrés suite aux éventuelles violations, et de récupérer les biens ou bien à la rigeur, limitent l'ampleur des dommages occasionnés.

5. Les approches de protection existantes

On trouve dans la littérature principalement, deux classes d'approches qui s'inscrivent dans le cadre de la sécurité des systèmes d'agents mobiles. La première classe regroupe les approches ayant pour objectif la protection des plateformes qui hébergent et exécutent les agents mobiles. La deuxième classe s'occupe de la sécurisation des agents mobiles.

5.1 La protection de la plateforme

Le but de la protection de la plateforme est d'empêcher l'agent d'interférer avec celle-ci. La plupart des techniques associées à cette protection sont basées sur le modèle de "*forteresse"* selon lequel on cherche à protéger l'hôte en ayant un système fermé qui n'est accessible que par des interfaces bien définies. Dans ce contexte, on utilise le concept de *"moniteur de référence"* qui est un programme contrôlant l'accès de l'agent mobile aux ressources de la plateforme. Ce moniteur de référence applique un certain nombre de règles pour l'accès de l'agent à ces ressources. Ces règles sont regroupées dans la "*politique de sécurité*" de la plateforme. Les techniques de protection de la plateforme les plus connues sont :

- **Authentification des agents:** Le problème de l'authentification des agents mobiles est similaire à celui qui se pose en milieu distribué. Ici, on a deux buts : vérifier l'intégrité du code et authentifier ses auteurs et ses utilisateurs.

Greenberg et Byington [23], proposent de signer l'agent mobile numériquement avec un algorithme cryptographique à clé publique. Cet algorithme peut soit générer un certificat qui assure l'intégrité du code et qui permet d'authentifier le propriétaire et le producteur de l'agent mobile pour l'hôte, soit crypter l'agent de sorte que seul le réceptionnaire puisse le décoder.

ý **Carré de sable** *(Sandboxing) :* La technique du carré de sable, consiste à isoler un programme dans son propre espace de fautes de façon logicielle. L'environnement est restreint en termes de privilèges et de ressources. Le code est exécuté dans une sorte de « carré de sable ». Un agent ne pourra pas modifier la plateforme, ni les autres agents qui circulent sur la plateforme. Par exemple, une applet java ne peut pas accéder au système de fichier local [60].

ý **Contrôle d'accès et d'autorisation:** Cette technique se rapproche du "carré de sable". Le moniteur de référence donne des autorisations à un agent mobile pour un certain nombre de ressources en fonction du résultat de son authentification. Pour savoir quelles autorisations donner à quel agent, le moniteur de référence applique les règles de la politique de sécurité. Ces règles touchent toutes les ressources dont un agent peut avoir besoin : accès au réseau, aux disques, etc. Dans le langage Java, le moniteur de référence est le « security manager », il prend en charge tous les accès aux ressources [59]. Cette technique est différente du « carré de sable » car elle permet, d'une part, d'avoir une granularité plus fine sur la protection et l'autorisation et, d'autre part, une adaptation des droits d'accès en fonction des agents.

ý **Estimation de l'état** *(State Appraisal):* Cette technique est utilisée pour protéger une plateforme à l'encontre d'un agent dont l'état a été modifié par une autre plateforme défaillante ou ennemie. À ce titre, elle pourrait aussi apparaître dans les techniques de protection de l'agent, car elle permet de détecter une altération de son état.

Farmer, Guttman et Swarup [19], ont proposé l'idée de la fonction d'estimation de l'état qui prend comme entrée l'état de l'agent et qui donne en sortie un ensemble de ressources dont l'agent a besoin pour exécuter sa tâche. Le producteur de l'agent et l'utilisateur créent leurs propres fonctions. L'auteur écrit une fonction "*max*" qui donne le maximum de ressources que l'agent peut demander. L'utilisateur de l'agent écrit une fonction *"req*" qui donne les ressources que demande l'agent. S'il n'y a pas de modifications de l'état de l'agent, on aura *req(E)* inclus dans *max(E)*. Si jamais ce n'est pas le cas, c'est qu'il y a eu modification. Ces fonctions permettent aussi à la plateforme de savoir de quelles ressources l'agent a besoin.

ý **Historique des hôtes: (***Path Histories*) **:** L'idée est d'évaluer la confiance qu'a un agent dans un hôte à partir de son identité et des plateformes sur lesquelles il s'est exécuté auparavant. Pour cela, il est nécessaire de mettre à jour un journal avec l'identité de toutes les plateformes qui ont été visitées par l'agent. Ce journal est protégé cryptographiquement pour éviter toutes tentatives de modifications : à chaque fois qu'un agent mobile migre, la plateforme signe le journal. Lorsqu'un agent arrive sur une nouvelle plateforme, cette dernière décide si elle l'exécute ou non et quelles ressources elle lui accorde en fonction de l'historique des hôtes traversés par l'agent. Cette technique est efficace dans le sens où, avec la signature numérique, une entrée dans l'historique est non répudiable. Le problème est que lorsque le nombre d'hôtes visités augmente, l'historique peut prendre un poids prohibitif [15].

5.2 Protection des agents mobiles

Alors que les contre-mesures visant à protéger les plateformes sont largement inspirées des systèmes conventionnels en employant des méthodes préventives, les techniques de protection des agents existantes, font plutôt de la détection. Cela est dû au fait que l'agent est complètement dépendant de la plateforme sur laquelle il s'exécute et ne peut, par lui-même, empêcher une attaque aisément. Par contre, il est plus facile de la détecter ou de la rendre moins dangereuse. Le principal problème vient des données et des informations d'état.

Parmi les techniques de protection de l'agent mobiles, on trouve :

ý **Enregistrement d'itinéraires avec des agents coopérants** (*Mutual Itinerary Recording*) : Roth [49] propose de donner une tâche à deux agents qui vont parcourir un ensemble de plateformes et qui ont des itinéraires disjoints, un des deux agents mobiles enregistre l'itinéraire de l'autre. Avant de migrer, un agent envoie à l'autre agent, à travers un canal authentifié, l'identité de la dernière plateforme, celle de la plateforme actuelle, ainsi que celle de la plateforme où il va. L'autre agent maintient un journal de l'itinéraire et vérifie qu'il n'y a pas de contradictions.

Roth a fait plusieurs hypothèses. La première est qu'aucun hôte sur l'itinéraire d'un agent ne coopère avec un hôte de l'itinéraire de l'autre agent. Il suppose aussi l'existence d'un canal de communications. Et que les identités sont données sans altération. Toutefois, la méthode recèle un certain nombre d'inconvénients. En effet, mettre en place le canal de communications à chaque migration est une opération coûteuse. Si un agent est tué, le protocole n'est pas capable de savoir lequel des deux

hôtes est responsable. Si une plateforme déclare avoir reçu un agent et que cela n'est pas vrai, le protocole ne peut décider qui est coupable. Enfin, si un hôte reçoit deux agents qui viennent du même hôte, il est possible d'inter changer les agents qui enregistrent l'itinéraire des deux autres.

ý **Traces cryptographiques (***Execution Tracing*) : Vigna [57,58], propose une technique qui permet de détecter toute exécution anormale d'un agent sur une plateforme à partir d'une sorte de « résumé de l'exécution » appelé trace. Après exécution, la plateforme crée un fichier en appliquant une fonction de hachage aux traces de l'exécution. Ce fichier est ensuite signé par la plateforme grâce à sa clé privée, puis envoyé au prochain hôte avec le code et l'état de l'agent. Une fois que l'agent a terminé son exécution et est rentré sur la plateforme de son propriétaire, ce dernier peut décider de vérifier l'exécution de l'agent. Si c'est le cas, il ré-exécute l'agent à partir de son état initial, puis compare le résultat de la fonction de hachage de ses traces avec celui de l'exécution à distance.

Cependant, des critiques peuvent être émises. La première est que les algorithmes à clé publique sont particulièrement consommateurs de temps UCT, la taille des traces peut devenir vraiment importante. De plus, il était supposé que le code était statique. Or, la plupart des langages de type Script proposent le code dynamique. Une autre limite est le fait qu'il faille attendre la fin de l'exécution de l'agent pour pouvoir détecter une attaque.

Tan et Moreau [56] proposent une amélioration de en faisant la vérification lors de l'arrivée d'un agent sur la nouvelle plateforme de son itinéraire.

ý **Calcul de fonctions cryptographiques** (*Computing with Encrypted Functions***) :** Sander et Tschudin [51] ont proposé une technique pour que l'agent mobile devienne une boite noire pour la plateforme sur laquelle il s'exécute. Il s'agit de faire exécuter à la plateforme un programme qui contient une fonction cryptée sans qu'elle soit capable de la décoder.

Exemple : *«Sara a un algorithme qui calcule la fonction f. Samy a une entrée x et veut calculer f(x) pour Sara, mais elle ne veut pas que Samy apprenne quoique ce soit sur f. De plus, Sara et Samy ne doivent pas s'échanger de message pendant le calcul»* Pour résoudre ce problème, il faut supposer que Sara puisse crypter f. Voilà la méthode.

- *Sara crypte f ce qui donne E(f), crée un programme P(E(f)) qui implémente E(f), puis envoie P(E(f)) à Samy.*
- *Samy applique P(E(f)) à x, envoie P(E(f))(x) à Sara, qui décrypte P(E(f))(x) et obtient f(x).*

Cette méthode est utilisable pour l'instant dans un domaine très restreint. Sander et Tschudin ont trouvé une méthode pour les fonctions f de type polynomial. Le problème est maintenant de trouver des modèles génériques qui permettent de calculer E(f).

ý **Boite noire limitée dans le temps** *(Time limited Blackbox) :*L'idée principale de l'approche de Hohl [29] est de générer un agent exécutable à partir des spécifications d'un agent initial qui ne peut être menacé par les attaques. On suppose que la protection n'est pas permanente mais a une durée limitée connue d'avance, au bout de laquelle l'agent devient obsolète. Si cette définition peut être appliquée à un agent, seules les entrées et les sorties sont observables d'où l'appelation " boîte noire". L'approche est basée sur les fonctions cryptographiques décrites par Sander et Tschudin [51]. Le mécanisme de conversion qui génère l'agent avec la boîte noire utilise des paramètres de configuration qui permettent de créer différentes sorties à partir d'une même spécification.

Etant donné que la plateforme qui l'exécute l'agent est capable de lire chacune de ses variables et ses lignes du code. Mais, si on veut comprendre le sens des instructions et les relations entre les variables, on doit réfléchir sur la sémantique du code. Cette sémantique n'est pas exprimée par le code, mais elle peut être devinée par un « modèle mental » d'un programmeur. L'idée principale de l'approche est d'interdire à un attaquant de bâtir un tel modèle mental. Ceci est réalisé en créant une nouvelle forme de l'agent dynamiquement, et en utilisant des algorithmes de conversion qui produisent des formes d'agent difficiles à analyser. On les appelle des *algorithmes de confusion*. Dont le rôle est de générer un nouvel agent à partir d'un agent initial qui a un code et des données différents mais aboutit au même résultat.

ý **Génération des clés à partir de l'environnement (***Environmental Key Generation***) :** Riordan et Schneier [48], proposent une méthode pour qu'un agent génère une clé quand une condition de l'environnement est vraie. Cette clé peut être utilisée pour décrypter un message destiné à l'agent dont l'expéditeur ne veut qu'il soit déchiffrable que dans certaines conditions. La plateforme ne doit pas savoir de quoi dépend cette condition car elle maîtrise tout l'environnement. Si l'environnement n'a pas les conditions pour générer la clé, il est impossible de deviner la fonction de l'agent. Il est aussi possible de générer une clé à partir du temps : certaines méthodes

permettent de générer la clé seulement avant une certaine date, alors que d'autres ne le permettent qu'après une date. En utilisant les deux, on peut arriver à générer une clé seulement pendant un intervalle de temps fixé.

ý **Politique de sécurité spatiale** (*Spatial Security Policies)* Scott, Beresford & Mycroft [53], ont défini la notion d'un environnement Sensible « Sentient Computing environment » dont un système qui y existe peut percevoir son état et utiliser ces renseignements pour personnaliser son comportement. Ils ont présenté une technique qui exprime des politiques spatiales (c.-à-d. basé-emplacement) de la sécurité pour les agents mobiles. Ces politiques peuvent être utilisées pour rendre les assertions (positive et négative) -au sujet de l'emplacement d'agents- dynamiques. Sachant que les assertions peuvent faire référence à l'emplacement d'objets physiques et virtuels dans le monde. Ces assertions constituent un trait utile pour Les applications sensibles aux emplacements. Cette technique fournit une façon utile de contraindre la mobilité de agents, afin d'utiliser la technologie de l'agent mobile sans risque et pour simplifier le processus du développement des futures applications Sensibles.

ý **Examen de sécurité auto-exécuté** (*Self-executiNg security Examination) «SENSE»,* Page, Zaslavsky & Inrawan [44] présentent une architecture logicielle générale pour la protection de l'agent mobile. Il s'agit d'un schéma de sécurité dans lequel un agent mobile est programmé de telle sorte qu'il porte sa propre implémentation de sécurité. Cela lui donne plus d'indépendance. L'idée consiste à utiliser un algorithme de scanne par l'agent mobile. Ce dernier scanne son code et vérifie ainsi son intégrité dans des intervalles de temps aléatoires. Le résultat retourné par le scanne est comparé avec une image de code portée par l'agent. L'avantage de cette approche réside dans le fait que l'agent mobile puisse détecter les éventuelles tentatives d'attaques et s'assure de son intégrité sans avoir à consulter son hôte propriétaire.

CONCLUSION

Après tout ce qui a été vu, nous pouvons résumer la définition d'un *agent mobile* comme étant une entité informatique qui réalise de manière autonome une ou plusieurs tâches déléguées par un individu ou une organisation, sur des machines distantes grâce à sa propriété de migration.

Il existe une panoplie de raisons qui justifient le choix d'utilisation les agents mobiles

- Réduire la charge des réseaux,
- S'exécuter de manière asynchrone et autonome,
- S'adapter dynamiquement aux environnements d'exécution hétérogènes,
- Robustes, performance et tolérants…

Cependant, le domaine des agents mobiles est encore récent et manque de maturité; une multitude de problèmes et de limitations restreignent leurs utilisations, tel que le problème d'hétérogénéité, de l'interopérabilité, le maintien des communications distantes… et notamment le problème de sécurité. Ce dernier peut être défini autour de deux axes:

La protection de l'hôte à l'encontre des agents qui y migrent: ce problème a fait l'objet d'une attention considérable des chercheurs. Les études ont montré que ces recherches sont assez bien avancées. En particulier, l'authentification des agents ainsi que l'historique des hôtes constituent des idées intéressantes.

La protection de l'agent vis-à-vis de l'hôte qui l'exécute: Cela est plus complexe, du fait qu l'hôte possède un plein pouvoir sur l'agent.

Le problème de la sécurisation de l'agent mobile contre les attaques des hôtes malveillants nous intéresse particulièrement. Les approches qui ont été proposées jusqu'à présent bien qu'elles révèlent des idées intéressantes, souffrent de sérieuses limitations; Ainsi, la boite noire limitée dans le temps n'a donné que des algorithmes de confusion très simples, le calcul de fonctions cryptographiques est aussi possible mais uniquement pour les fonctions polynomiales et rationnelles. Il serait intéressant de trouver des algorithmes applicables à un contexte plus large. Aussi, il est possible d'améliorer le protocole de Roth "enregistrement d'itinéraires avec des agents coopérants" pour ne plus avoir les faiblesses mentionnées. De plus, les travaux présentés ne prennent pas en considération le cas où les conditions de travail changent!

En revanche, d'autres travaux sont entrain de se faire, traitant et développant les techniques de l'adaptation des agent mobiles dans des environnements changeants et variants. Une solution intéressante au problème de la sécurité des agents mobiles, nous paraît se situer dans ce dernier domaine. Au niveau du chapitre suivant, nous étudierons et détaillerons les techniques d'adaptabilité pour voir par la suite comment les utiliser pour le compte de la protection de l'agent mobile.

Chapitre II

OUTILS DE CONCEPTION: L'ADAPTABILITÉ & LA RÉFLÉXIVITÉ

« Tout devrait être réalisé aussi simple que possible, mais rien n'est simple »
Aalbert Einstein

Pour des raisons de clarté, et dans le but d'une meilleure compréhension de ce mémoire, nous expliquerons dans le présent chapitre, les principaux concepts que nous utiliserons dans l'élaboration de notre contribution -s'inscrivant dans le cadre de la sécurité des agents mobiles-. Nous commençerons par l'explication de l'adaptabilité; l'idée clé sur laquelle se base l'approche de protection que nous proposons. Nous verrons également les différentes approches permettant son implantation. Parmi ces dernières nous insisterons sur celle que nous adopterons pour la réalisation de notre travail; celle de la réflexivité. Nous survolerons, à la fin de ce chapitre quelques travaux ayant traité les deux concepts mentionnés (L'adaptabilité & la réflexivité), en guise d'avoir une idée plus pratique sur leurs usages.

INTRODUCTION

S'il y a quelques années une application se contentait de réaliser une tâche dans un environnement d'exécution bien défini, la complexité croissante des systèmes informatiques nous oblige à repenser à ce schéma ; Les applications évoluent aujourd'hui dans des environnements variants et changeants. Ayant reconnu comme très complexe, voire impossible, la construction d'un système fournissant toute la panoplie de services dont on pourrait avoir besoin, les recherches autour des applications réparties ont pris une nouvelle direction. *L'adaptabilité* est le mot clé de cette nouvelle tendance. Cette dernière se révèle être une réponse prometteuse aux besoins de gestion de la qualité de service « QoS * », tel que la performance, la sûreté, la tolérance aux pannes, la sécurité, etc.

Dans le cadre des systèmes d'agent mobile, quand ce dernier se déplace vers un nouvel hôte, il peut rencontrer de nouvelles conditions d'exécution. Étant donné qu'un agent mobile comme tout système informatique, peut être vu comme une implémentation d'une solution à un problème donné dans un contexte particulier, un tel système dépend donc à la fois des caractéristiques du problème à résoudre et de celles du contexte de l'implémentation. Si l'un de ces deux éléments vient à changer, la solution initiale peut ne plus être adaptée. L'agent doit être en mesure de détecter ces nouvelles conditions, afin d'y répondre. Autrement dit, il doit s'ajuster au nouvel contexte d'exécution. En effet, l'adaptation est un mécanisme permettant à un système de fournir ses services sous des conditions particulières ou nouvelles et lui permet d'assurer les modifications nécessaires d'une manière transparente [43].

La *réflexivité* construit une approche attrayante pour le développement des applications adaptables, du fait qu'elle présente un support de développement d'applications fournissant des mécanismes permettant d'accéder à certains détails d'implémentation du système. Le grand intérêt de ce type d'approche est de permettre d'exprimer des traitements en termes extrêmement génériques, qui n'utilisent que les notions constitutives du système.

Dans ce chapitre, nous expliquons ce que cache le terme "Adaptabilité". Nous définissons d'abord cette notion en spécifiant le sens le plus commun du terme. Nous considérerons les différentes classes d'adaptation, les entités concernées par ce processus, les fonctionnalités requises et les principales approches permettant son implantation. De plus nous essayons de fournir les arguments nous menant à l'utilisation de la réflexivité. Ainsi, nous détaillerons cette dernière: nous verrons en quoi elle consiste, sa structure, ainsi que les problèmes pouvant être rencontrés l'ors de sa conception.

---

* *QoS (Quality of Service): toutes les propriétés souhaitables imposées par l'usager autres que le comportement fonctionnel d'une applicati*on.

I) L'ADAPTABILITÉ

1. Définition

L'adaptation désigne l'action de s'ajuster et réagir face aux variations des contraintes de l'environnement et des besoins des utilisateurs. On entend par 'adaptabilité' la capacité ou le degré d'adaptation. [41]

Notons que ces deux termes sont "Adaptation" & "Adaptabilité" sont souvent confondus dans la littérature.

2. Différentes classes d'adaptation

Dans la littérature, il n'y a pas de classification standard pour d'adaptation Les chercheurs utilisent différentes classifications. Quelques uns la catégorisent en adaptation des données, adaptation de l'état, adaptation du code, et adaptation des fonctionnalités [33,35,38].

2.1 L'adaptation des données

L'utilisateur peut accéder à des données de différentes sortes et modèles. L'adaptation des données est une méthode qui transforme les données selon les contextes des clients repondant ainsi à leurs besoins. Par exemple, sélectionner les champs des données pertinents, réarranger les renseignements en un format plus approprié, ou changer les propriétés d'images vers une résolution adéquate. Les techniques de l'adaptation des données portent donc plus sur la disposition et l'organisation des données. De nombreux projets de recherches traitent ce type d'adaptation.

2.2 L'adaptation de l'état

L'adaptation de l'état est un moyen de changer l'état interne d'une application pour se conformer à un certain contexte. Cette adaptation diffère de celle des données dans le sens où, d'un côté, l'adaptation de l'état dresse l'état interne des variables -y compris l'état des données aussi bien que l'état de l'exécution- d'une application, alors que l'adaptation des données se concentre sur la disposition de données. D'un autre côté, l'adaptation de l'état supporte l'adaptation pendant la migration dans un contexte des codes mobiles, pendant que l'adaptation des données supporte l'adaptation après migration seulement. Néanmoins, peu de recherches se concentrent sur ce type d'adaptation.

2.3 L'adaptation du Code

Ce type d'adaptation concerne beaucoup plus l'univers des codes mobiles dans un environnement évoluant et changeant. Les codes doivent être déplacés sur plusieurs machine en vu d'exécution. L'adaptation du code permet, donc, de changer le code d'une application de telle sorte à s'ajuster à l'environnement du client. Par exemple, ajouter de nouveaux paramètres, modifier les arguments et les données, enlever les parties du code inutiles, ou encore reconstruire le code pour le nouveau contexte en se basant sur le code existant.

2.4 L'adaptation des fonctionnalités

L'adaptation des fonctionnalités implique le changement de la manière dont la tâche est réalisée pour répondre aux changements [5]. Il sélectionne les différentes implémentations du code selon le contexte. L'adaptation des fonctionnalités diffère de celle du code dans le sens où la première - L'adaptation des fonctionnalités- télécharge de nouveaux codes quand le contexte change, pendant que l'adaptation du code transforme le code existant en un nouveau code. Par exemple, si un composant n'a pas suffisamment de ressources pour s'exécuter, l'application fonctionnelle peut choisir une autre implémentation nécessitant moins de ressource.

3. Dynamisme de l'adaptation

L'adaptation peut être statique ou dynamique [36] :

3.1 Adaptation statique

Ce cas correspond à une adaptation effectuée avant l'exécution en fonction des connaissances détenues de l'environnement de déploiement. Il peut s'agir d'une adaptation de la gestion pour assurer un niveau de qualité de service (choisir l'implémentation qui convient le mieux). Dans la même catégorie nous plaçons les approches où l'adaptation est réalisée pendant la phase d'initialisation de l'application.

3.2 Adaptation dynamique:

Au niveau de ce type d'adaptation nous distinguons deux catégories :

Adaptation spécifiée statiquement mais effectuée dynamiquement: Cette solution consiste à prendre en compte pendant la construction de l'application les différentes variations de l'environnement et de définir les actions d'adaptation en conséquence (la stratégie). La stratégie est ainsi spécifiée de manière statique (avant le

lancement) mais les adaptations sont effectuées dynamiquement. C'est l'approche utilisée dans le cas des applications réparties "classiques" où les constructeurs décident d'utiliser un protocole donné pour la gestion d'un problème relié aux fluctuations de l'environnement. Le choix est statique alors que le fonctionnement de ce protocole prend en compte des informations dynamiques [41]. Ce type d'adaptation peut se faire par connexion avec des bibliothèques dynamiquement chargeables et en tenant compte des ressources disponibles, ou par modification du code exécutable : (ajout, optimisations particulières tenant compte des ressources disponibles ou de l'utilisation particulière qui en est faite) [40]. Dans cette catégorie, se trouvent les projets utilisant le principe de *réflexivité* permettant un choix dynamique entre les implémentations existantes [34].

Adaptation spécifiée et effectuée dynamiquement: Cette dernière approche va encore plus loin et permet non seulement d'effectuer de l'adaptation pendant l'exécution mais aussi la définition et la mise en place dynamique de la stratégie d'adaptation. Pour l'instant, nous ne sommes pas au courant d'une sérieuse implémentation de cette idée, elle n'est qu'envisagée [41].

4. Entités concernées par une adaptation:

Que peut être l'objet de l'adaptation? Et qui participe à ce processus??

4.1 Sujet de l'adaptation:

Autrement dit, Parmi les différents éléments constituant le système, lesquels vont être la cible des modifications [17].

Les caractéristiques inhérentes du réseau comme l'hétérogénéité, l'asynchronisme, la distribution et la limitation des ressources imposent lors de la construction d'une application la prise en compte non seulement des aspects purement fonctionnels mais aussi des problèmes (appelés par opposition non fonctionnels) comme la sécurité, les encombrements réseau, la tolérance aux pannes, le temps de réponse, etc. L'ensemble de ces propriétés et leur gestion définit la qualité de service du système**.** De ce fait, on distingue dans le code constituant un programme le code fonctionnel implémentant la logique de l'application et sa sémantique, du code non fonctionnel qui implémente les divers services utilisés pour supporter le bon fonctionnement de l'application.

En pratique, cette séparation sera plus ou moins explicite dans le code source d'une application ; il s'agit ici d'une distinction purement conceptuelle. On peut noter cependant que, comme le montrent les approches de séparation de problèmes (*Separation of Concerns)* telles que *la programmation par aspects* « AOP *», la *Programmation par sujets* « SOP**», la ***Réflexivité***...[36]), il est effectivement possible de rendre cette séparation explicite, et obtenir ainsi de nombreux avantages comme la réutilisation.

Cette séparation au niveau conceptuel est particulièrement intéressante. Elle permet en effet de distinguer deux sujets différents sur lesquels focalise le processus d'adaptation.

La première alternative: choisit de porter les efforts d'adaptation sur le code fonctionnel lui-même. Le programmeur d'application fournit plusieurs implémentations différentes de certains éléments du code fonctionnel, et spécifie dans quelles conditions ils devront être utilisés. Cette approche permet au programmeur de conserver un contrôle total sur le code exécuté, mais au prix d'un travail supplémentaire.

La seconde alternative: à l'inverse, choisit de se concentrer sur le code non fonctionnel. Le rôle du code non fonctionnel est de modifier, dans une certaine mesure, la façon dont le code fonctionnel sera interprété, mais sans altérer sa sémantique. On suppose ici que le code non fonctionnel est organisé sous forme de *"modules"*, en fonction du type de modifications qu'ils effectuent. La seconde approche va donc consister à modifier les services non fonctionnels et donc l'interprétation du code fonctionnel.

Pour conclure sur ces deux alternatives possibles de l'adaptation, on peut noter que la séparation entre code fonctionnel et non fonctionnel peut aussi être vue comme la séparation de ce qui est spécifique à une application particulière et de ce qui en est indépendant. De ce point de vue, la première alternative ne peut pas être aussi bien généralisée que la seconde. Le travail qu'elle demande est à fournir de nouvelles implémentations pour chaque application, et ne peut pas être factorisé. À l'inverse, la seconde permet de développer des services non-fonctionnels indépendamment d'une application particulière. En ayant des services non-fonctionnels réutilisables, le travail à fournir pour adapter une nouvelle application est beaucoup plus réduit.

***** *AOP: Aspect oriented programation*
****** *SOP : Subject oriented programation*

4.2 Acteur de l'adaptation:

Une fois décidée ce qu'on veut adapter, reste à savoir qui effectue cette adaptation (l'adaptateur). Là encore, il y a plusieurs possibilités [17].

La première, la plus simple, est de laisser ce problème entièrement à la charge du programmeur et de l'utilisateur de l'application. Le programmeur devra alors établir explicitement les réactions de son système aux variations de l'environnement, ainsi l'utilisateur devra intervenir à chaque fois lors de l'exécution. L'application se contente alors de notifier par un moyen ou un autre que les conditions ont changé. La grande majorité des travaux choisit cette approche.

La seconde possibilité est de tenter de rendre ce mécanisme le plus automatisé possible. Le but étant de minimiser le travail à effectuer par le programmeur d'application, il semble donc naturel que l'application prenne en charge le plus gros du travail. Cependant. Nous ne pouvons pas espérer atteindre une automatisation complète, mais nous devons tenter de limiter le plus possible le travail demandé au programmeur d'application.

Nous pouvons résumer ces deux approches en parlant de systèmes *adaptables* dans le premier cas, et de systèmes *adaptatifs* dans le second. La différence est que dans le premier type, il s'agit d'un système supportant l'adaptabilité mais il est essentiellement passif ; l'acteur de l'adaptation est extérieur (le programmeur). Tandisque, dans le second cas, le système est actif, et est lui même moteur de l'adaptation, c-à-d, il set capable d'intègrer à la fois le sujet de l'adaptation et l'adaptateur.

Au niveau de ce dernier cas: nous distinguons au niveau du système, plusieurs parties participants au processus d'adaptation: *Le maître*: devant être en mesure de décider en fonction de quoi va être effectuée l'adaptation, décider du moment de l'adaptation ainsi que des mécanismes d'adaptation à utiliser. Ou *le servant*: effectuant l'action décidée par le maître.

Remarque : Les systèmes adaptatifs sont appelés aussi *"systèmes auto-adaptables"* [40].
"Dynamic adaptation helps applications to become self-adaptive, i.e. applications become able to autonomously undertake appropriate reconfiguration in response to the evolution of their execution context" [4].

5. Moments de la réalisation de l'adaptation:

L'adaptation est implantée durant plusieurs phases du cycle de vie de l'application en question, toutefois, selon le type et le dynamisme de l'adaptation mis en évidence, certaines phases sont plus ou moins accentuées [17,40].

5.1 Conception et codage

Dans cette phase, nous parlons de l'adaptation de cadres d'applications pré-existants au cas particuliers de l'application à développer. Même si l'adaptation ne peut avoir lieu au moment de la conception d'une application et de son implémentation, il est essentiel quant même de préparer le terrain à ce moment là. Pour pouvoir permettre des adaptations ultérieures, une application doit avoir certaines caractéristiques. Par exemple, elle doit être *modulaire*, pour permettre des modifications localisées. Un système *monolithique* ne peut être modifié que s'il est remplacé intégralement, mais dans une application construite sous forme de modules, il est possible de modifier ou de remplacer certaines parties du système sans interférer avec le reste. Pour permettre *l'adaptabilité dynamique*, ces différents modules doivent exister explicitement à l'exécution, c'est exactement ce que nous proposent les approches à objets et à composants.

5.2 Déploiement

Au moment du déploiement, c'est-à-dire de l'installation du système sur une machine (ou un ensemble de machines), on peut effectuer une autre forme d'adaptation : la configuration. Une fois que l'on connaît les ressources physiques et logicielles de la machine, qui ne changent qu'à un rythme relativement lent, le degré d'incertitude qui régnait au moment de la conception et du codage peut être considérablement réduit. On peut alors configurer l'application de façon plus spécifique, quitte à perdre en généralité ce que l'on peut gagner en performances (optimisations par rapport à l'architecture cible par exemple).

5.3 Exécution

C'est à l'exécution que la partie la plus intéressante de l'adaptation a lieu. Il s'agit de reconfigurer dynamiquement les différents éléments du système et/ou leurs associations en fonction des évolutions rapides de la disponibilité des ressources et des conditions d'exécution. La réalisation de l'adaptation dans cette phase diffère selon le types, le dynamisme, le sujet et le réalisateur de l'adaptation, ainsi que la stratégie adoptée pour son implantation.

6. Infrastructure pour l'implantation de l'adaptabilité

6.1 La programmation modulaire

Comme nous l'avons déjà mentionné, la structure modulaire facilite et rend la réalisation de l'adaptation une tâche plus évidente. En particulier la programmation par composants présente un bon support pour sa réalisation. En effet, la programmation par composants est une technique de construction d'applications par intégration d'entités logicielles. Les composants sont des objets pouvant être activés/désactivés dynamiquement, et qui sont conçus pour être réutilisés de manière homogène dans différentes applications et contextes d'exécution, sans modification de leur implémentation.

Cette approche permet d'envisager un processus de développement d'applications à deux niveaux : un premier correspondant à l'architecture globale du système et le second correspondant à l'écriture des composants. La phase de programmation macroscopique définit l'assemblage de composants logiciels formant l'application globale, ainsi que son adaptation dans le cas d'une application possédant cette faculté. Elle est à l'origine de l'architecture du système. Pendant que, la phase de programmation microscopique permet la réalisation d'un composant logiciel donné à l'aide de langages de programmation classiques, mettant en évidence les fonctionnalités du système. Nous pouvons constater au niveau de la phase macroscopique, la nécessité d'un modèle de spécification de l'assemblage de composants. Un grand nombre de chercheurs ont proposé des modèles formels pour représenter et analyser les concepts architecturaux. Souvent appelés ADL, ces modèles fournissent à la fois une base conceptuelle et une syntaxe concrète pour la caractérisation d'architectures logicielles. Par exemple, le séquencement entres les différents composant peut être établi via un graphe (graphe des composants). Dans un tel graphe, les noeuds sont les composants et les arcs représentent les liaisons entre ces nœuds, Ces derniers peuvent être interprétés par des chemins de passage des messages entre les composants ou d'invocation des méthodes [2].

6.2 Fonctionnalités requises pour l'adaptabilité dynamique

Cette section présente brièvement les différentes fonctionnalités nécessaires pour un système adaptatif, à savoir: l'observation, La décision, et l'action [17].

Observation: Le système doit bien entendu être capable d'observer l'environnement d'exécution. Cette fonctionnalité est obligatoire si l'on veut que les modifications apportées à l'application soient liées à cet environnement. Plus

précisément, le système doit être capable de découvrir quelles sont les ressources disponibles, à la fois physiques et logicielles, ainsi que de détecter les variations dans la disponibilité de ces ressources. Ce sont ces variations qui seront la cause première déclenchant les adaptations. Il ne faut pas non plus oublier que le système doit avoir une bonne connaissance de lui-même pour qu'il puisse effectuer des transformations ayant un sens tout en conservant sa cohérence.

Décision: Ayant une bonne connaissance à la fois de l'environnement extérieur et de lui-même, notre système doit maintenant être capable de décider quelles modifications de cet environnement ont un impact sur les applications en cours d'exécution et quelles modifications mettre en oeuvre pour y répondre. Ces informations étant en règle générale, dépendantes des applications, elles ne peuvent pas être toujours prévues à l'avance.

Certains systèmes se contentent à ce niveau de laisser la main au programmeur d'application. Cette fonctionnalité est donc réduite au minimum dans ces systèmes ; au mieux, ils sont capables de détecter des changements dans l'environnement d'exécution et d'en notifier l'application. Si nous voulons prendre en charge cette fonctionnalité d'une façon plus automatique, le programmeur doit fournir au système, certaines *règles* définissant une *politique d'adaptation*. Ces dernières peuvent avoir différentes formes suivant les solutions adoptées, mais leur rôle est d'indiquer quelles sont les variations de l'environnement qui ont un impact sur les applications en cours d'exécution, et quelles mesures il doit prendre pour adapter ces applications aux nouvelles conditions. Évidemment, le but recherché est de concevoir des politiques d'adaptation simples et d'un niveau d'abstraction élevé.

Action: Une fois que le système a détecté une modification significative de l'environnement d'exécution et qu'il a décidé que celle-ci devait entraîner une réaction de sa part (en se basant sur une certaine politique d'adaptation), reste à déterminer quels types de modifications du système sont possibles. Là encore, suivant le sujet d'adaptation choisi, les types d'actions possibles varient.

Dans le cas où le sujet d'adaptation est le code fonctionnel, les modifications possibles vont consister à remplacer certains éléments de ce code fonctionnel par une version alternative, plus adaptée aux nouvelles conditions détectées. Les

éléments de code fonctionnel que l'on peut ainsi modifier dépendent du système ; cela peut être la classe d'un objet, remplacée par une classe alternative.

Si nous choisissons le code non fonctionnel comme sujet de l'adaptation, les différentes actions possibles à ce niveau correspondront à la reconfiguration des associations entre composants fonctionnels et les services non fonctionnels. Par exemple, nous pouvons modifier la sémantique de l'invocation de messages pour certains composants en la rendant asynchrone. Il doit être possible de spécifier de façon précise quels services doivent (ou ne doivent pas) être associés à tel ou tel composant. Nous devons aussi pouvoir configurer ces services, en spécifiant par exemple des paramètres. Ces actions ne doivent pas être globales au système, mais locales aux composants pour lesquels elles ont un sens.

7. Approches de réalisation de l'adaptabilité

La généralité de la notion, implique que selon les besoins et le type d'application considéré, il est possible d'avoir de nombreuses approches différentes à la réalisation de l'adaptabilité. Nous considérons que la stratégie se situe à un niveau plus haut que les mécanismes d'adaptation particuliers. Deux approches principales qui traitent le problème de la stratégie dans les travaux autour de l'adaptabilité sont identifiées. Ces dernières sont nommées approche de la stratégie implicite et approche de la stratégie explicite [41].

7.1 Approche implicite*:*

La stratégie implicite est l'approche la plus fréquemment utilisée. Il s'agit des cas où l'application fait usage d'un certain protocole encapsulant une solution à l'adaptation vis à vis d'un problème donné sans avoir le droit de contrôle, ni de remplacement. Il s'agit donc de bénéficier de l'adaptabilité offerte par ce protocole particulier pour gérer son propre problème. Les exemples à citer sont nombreux: utilisation d'un service de communication donné (facilités proposés par CORBA, JAVA-RMI ou autre), d'un type de sécurité donné ou un type de gestion de la persistance, etc.

L'adaptabilité proposée dans ces cas a un domaine de validité limité. Elle ne concerne que les variations de l'environnement considéré. Si l'application doit faire face à des changements non prévus dans cet environnement d'exécution ou être déployée dans un environnement "hostile' où ces protocoles ne sont pas efficaces et ne gèrent pas les problèmes de manière adaptée, elle peut perdre beaucoup au niveau de la performance et même se retrouver en état de blocage.

7.2 Approche explicite:

Les applications réparties qui ont le droit de contrôler la définition ou/et le choix des activités d'adaptation exhibent une stratégie explicite. Cette stratégie peut être fournie soit de manière procédurale, soit de manière déclarative, ou encore d'une manière réflexive.

La manière procédurale consiste à programmer dans l'application la gestion des aspects liés à l'adaptation. Dans ce cas, le programmeur de l'application manipule directement dans le code les mécanismes fournis par le système. La granularité de ces mécanismes peut varier et il est possible d'aller de la gestion propre des ressources système pouvant être assez difficile. Ça peut demander beaucoup d'expérience à l'utilisation de services fournis par une couche intermédiaire (dans le cas d'application répartie c'est le middleware) qui encapsulent des services de plus bas niveau. La nature des services peut être très diverse et en conséquence les moyens fournis au programmeur pour leur manipulation peuvent différer considérablement**,** ce qui complique notoirement la tâche du programmeur.

La manière déclarative: Comme son nom l'indique, La manière déclarative consiste à fournir une spécification de la qualité de service et de sa gestion sans nécessairement considérer les mécanismes effectifs à mettre en place lors de la réalisation. L'approche vise donc la possibilité de pouvoir exprimer et documenter explicitement les aspects liés à la qualité de service pendant la phase de conception et d'éviter la pratique d'implémentation directe dans le code.

La manière réflexive est une solution intermédiaire qui gagne de plus en plus de popularité. La visibilité qu'elle offre du système peut être placée entre la transparence de l'approche déclarative et la visibilité totale dans l'approche procédurale où tout le code est écrit manuellement. La réflexivité répond au principe d'implémentation ouverte où certains points décisifs du système sont accessibles et adaptables. Elle permet de maintenir automatiquement la liaison entre un niveau abstrait (méta-modèle) et le code. Avec la réflexivité il est possible non seulement de rajouter des services dans le code mais aussi de modifier l'implémentation des services existants.

Remarque *: C'est cette dernière approche –La réflexivité- que nous allons adopter pour implanter l'adaptabilité dans le contexte qui nous occupe. La suite de ce chapitre est consacrée pour étudier et détailler cette approche là.*

II. LA RÉFLEXIVITÉ

1. Présentation:

1.1 La réflexivité au quotidien

Un système est réflexif s'il est capable d'appliquer à lui même ses propres capacités d'action (capacités de description, de calcul, de pensée…). Par exemple, l'être humain, en sa qualité d'animal qui pense, est un système réflexif, puisqu'il peut penser à lui-même. Un autre exemple très expressif est celui de la langue, toute langue humaine obéit à un ensemble de règles (grammaire) qui la gouvernent. Or, il est tout à fait possible, de discuter de la grammaire d'une langue *dans cette langue elle même*. Une langue, *outil* de discours, devient alors l'*objet* de son propre discours. C'est de *ce double rôle* que surgit la réflexivité : un manuel de grammaire française pour écolier, écrit en français, est un texte en français qui parle de la langue française, c'est à dire un texte *réflexif*. Cet *auto-référencement* constitue la première caractéristique de la réflexivité. La deuxième, a trait aux liens qu'entretient une langue avec l'ensemble des règles qui la gouvernent, lorsque l'une ou l'autre évolue. L'évolution d'une langue se répercute sur la grammaire : de nouvelles règles de grammaire sont introduites pour prendre en compte les changements constatés. Réciproquement, il arrive que l'évolution des règles précédent alors celle de la langue. Il existe donc *une double relation de causalité* entre une langue et les modèles qu'en construisent les grammairiens. Cette double relation de causalité constitue la deuxième caractéristique de la réflexivité [41].

La grammaire constitue donc une modélisation d'une langue à l'intérieur d'elle-même, que nous appellerons dans un contexte réflexif un *méta-modèle*.

1.2. La réflexivité en informatique

La machine universelle d'Alan Turing (une machine de Turing capable d'émuler toute autre machine de Turing, et en particulier donc, de s'émuler elle même) constitue un exemple de la structure réflexive. De manière fort intéressante, l'une des toutes premières architectures proposées pour les ordinateurs, celle de John Von Neumann, était justement par essence réflexive. Elle consiste à stocker les instructions d'un programme et les données qu'il manipule sur le même support physique. Les instructions d'un programme sont donc à la fois les outils du traitement, et potentiellement aussi des données pouvant être traitées. Cette caractéristique n'était cependant probablement pas l'un des objectifs principaux recherchés par John Von Neumann. C'est au fil du développement des langages évolués, notamment avec les

recherches sur le langage de programmation LISP, que la réflexivité est peu à peu apparue comme une solution élégante à des problèmes nécessitants l'adaptabilité et la réutilisation [41].

1.3 La réflexivité dans le cadre formel

La réflexion, est une manière d'organisation interne d'un système pour faciliter son développement, son adaptation, et sa réutilisation, en offrant un cadre conceptuel permettant un découplage des fonctionnalités d'un programme [41].

Définition1: « *La **réflexivité** est la capacité pour un programme de manipuler comme données quelque chose représentant l'état de ce programme durant son exécution* » [40].
Autrement dit; c'est la capacité pour un programme de raisonner et d'agir sur soi.

Remarque *:* les termes « réflexion » & « réflexivité » sont aussi souvent confondus dans la littérature.

Autres définitions: Nous citons là quelques définitions mettant en évidence dans leur ensemble les principaux aspects de la réflexivité [40].

*« A **computational system** is something that **reasons** about and **acts** upon some part of the world, called the **domain** of the system. »*

*«… A computational system can be **causally connected** to its domain. This means that the system and its domain are linked in such a way that if one of the two changes, this leads to an effect upon the other. »*

*« A **meta-system** is a computational system that has as its domain another computational system, called its **object-system**. . . . A meta-system has a representation of its object-system in its data. Its program specifies **meta-computation** about the object-system and is therefore called a **meta-program**. »*

*« **Reflection** is the process of reasoning about and/or acting upon oneself. A **reflective system** is a causally connected meta-system that has as object-system itself . . . When a system is reasoning or acting upon itself, we speak of **reflective computation**.”» . **Paty. Maes***

« *An entity's integral ability to represent, operate on and otherwise deal with its self in the same way that it represents, operates on and deals with its primary subject matter*» ***Brian Smith.***

2. Structure d'un système réflexif.

Cette démarche consiste à considérer un même système à deux niveaux d'abstractions : Un niveau de base et un niveau méta (ou méta-niveau) [42,55]**.**

- ***Le niveau de base***: désigne l'application standard, il manipule les entités du domaine d'application et décrit la fonctionnalité de l'application.

- ***Le méta-niveau*****:** C'est le niveau qui réfléchit sur le niveau de base, il manipule des abstractions de ses entités. Ce niveau réifie* ces abstractions dans des entités appelées méta- objets.

Autrement dit: Au niveau de base se situe le code fonctionnel de l'application (ce que l'application fait). Et au méta-niveau se situe le code non fonctionnel de l'application (comment elle le fait) (voir Fig.2.1)

Les objets du niveau de base sont communément appelés objets de base, et ceux du méta-niveau: méta-objets. On regroupe sous le terme de méta-système l'ensemble des objets du méta- niveau.

Les *méta-objets* manipulent comme données, les méthodes ou les classes du niveau de base, afin de pouvoir raisonner sur les objets de ce niveau. Par exemple, le langage Java fournit des classes (*Method, Field, Class, etc.),* qui permettent de raisonner sur la structure des objets Java pendant l'exécution [17].

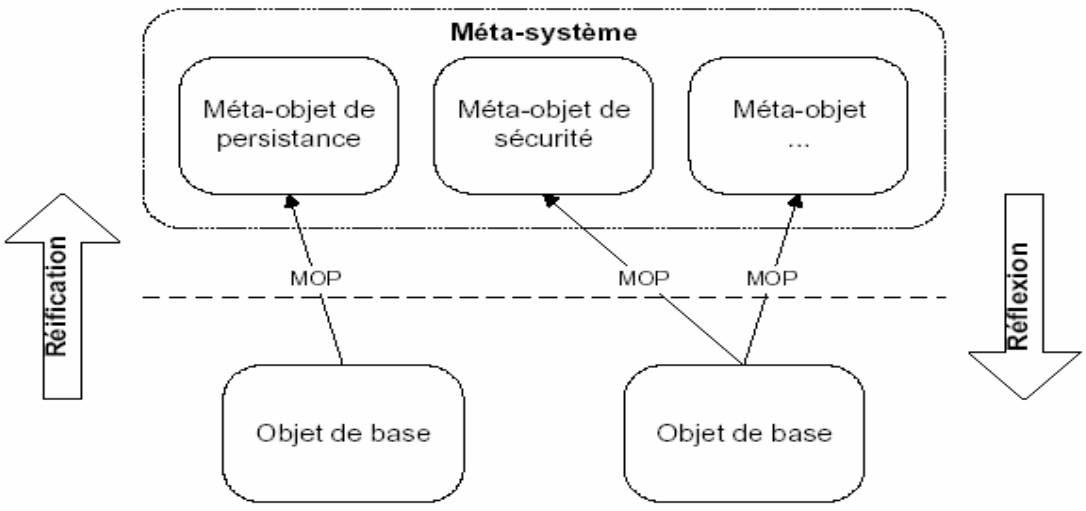


Fig 2.1- Structure globale d'un système réflexif [42]

***Réification** : fournir un encodage du programme et de son état d'exécution comme données.

Réifier : Transformer en chose ; donner un caractère de chose. V. ***Chosifier****.*
Réification : n.f. Action de réifier son résultat……..Le Petit Robert.
Aspect opérationnel : rendre manipulable comme une chose (matérialiser) [40].

Un rôle particulier peut être affecté à chaque méta-objet. Ces méta-objets sont associés ensuite à des objets du niveau de base pour qui ils vont gérer les aspects qu'ils implémentent. Notons bien ici que la programmation modulaire tel que l'Orienté Objets se prête bien à la réflexivité.

Le processus consistant à rendre accessible l'implémentation de certains aspects du système à travers un méta objet s'appelle réification. Son processus dual, c'est à dire le processus consistant à modifier le comportement d'un objet de base s'appelle réflexion.

Les interactions entre un niveau de base et son méta-niveau s'appuient sur deux processus :

- ***L'introspection:*** permet au système de *réifier* des abstractions de son niveau de base dans des méta-objets, qui font aussi partie de ce système. L'introspection permet donc à un système de raisonner sur lui-même pendant son exécution [17].
- ***L'intercession:*** permet aux méta-objets de modifier pendant l'exécution du système les propriétés des objets du niveau de base dont ils sont une abstraction. Donc des interactions avec des méta-objets, qui font partie du système, permettent de modifier les propriétés du système. L'intercession permet donc à un système de modifier lui-même ses propriétés pendant son exécution [17].

Le lien entre des méta-objets et les objets de base dont ils sont une abstraction, est appelé lien "méta". Les interactions entre des objets dans un même niveau d'abstraction se font sur des liens "non-méta". Il est possible qu'un méta-niveau ait lui-même un méta-niveau (un "méta-méta- niveau") : Un tour de méta-niveaux peut ainsi être formée [17]. Seulement, tout comme une récursivité bien fondée, un programme réflexif bien fondé doit faire appel à un nombre borné de méta-niveaux lors de chacune de ses exécutions. Il faut prévoir donc un niveau terminal pour cette structure [40].

3. Méta programmation versus réflexion

En programmation, le programme est le système de base. Tout programme manipulant d'autres programmes est donc un méta-système(Interprète, compilateur, évaluateur partiel, imprimeur, …).

Contrairement à la réflexivité, en méta-programmation : le méta-niveau est pré-existant, et l'adaptabilité se fait par modification du méta-programme avant l'exécution du programme. Autrement dit, seul le méta-programme a accès à la représentation du programme; le programme n'a pas conscience de l'existence du méta-programme. En réflexion, le programme a accès à sa propre représentation et peut donc agir sur lui-même [40].

« *The main difference between a meta-level architecture and a reflective architecture is that a meta-level architecture only provides* ***static access*** *to the representation of the computational system, while a reflective architecture also provides a* ***dynamic access*** *to this representation.* » ……..***Patty Maes*** [39].

Notons que la distinction faite par Maes pointe bien l'idée que la forme d'adaptabilité fournie par la méta-rogrammation est statique et non dynamique car elle se fait par modification du métaprogramme, ce qui n'est pas nécessairement lié aux moments d'adaptabilité du programme de base.

4. Critères de classification des systèmes réflexifs

Les architectures réflexives existantes sont souvent différentes les unes des autres et difficilement identifiables. Dans cette partie, nous essayons d'établir quelques critères de classification des systèmes réflexifs.

4.1 Réflexivité à la compilation / au chargement / à l'exécution:

Le moment de la mise en place de la structure du méta-niveau et son imbrication avec le niveau de base définissent ce qu'on appelle le temps de la réflexivité. Selon les systèmes, elle peut avoir lieu à la compilation, au chargement ou à l'exécution. Le ratio *flexibilité/efficacité* est à chaque fois différent [42].

A la compilation: Dans ce type de système, les liens entre le niveau de base et le méta-niveau sont effectués lors de la compilation. Ainsi la compilation du niveau de base est effectué (ou réglementée) par son programme de méta-niveau (voir Fig 2.2).

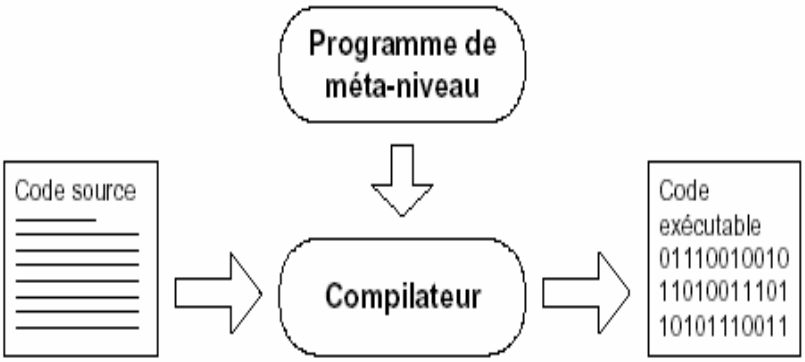


Fig 2.2- Réflexivité à la compilation [42]

Ce type de système préserve l'efficacité de l'exécution de l'application. La principale surcharge de travail se produit à la compilation, car la structure réflexive n'est présente qu'à cette étape. Par contre, la flexibilité est relativement réduite, aucune modification n'étant possible à l'exécution. Les entités dynamiques (par exemple les objets, qui n'ont une existence qu'à l'exécution) ne peuvent que difficilement être pris en compte de façon individuelle par un tel système.

Au chargement: La réflexivité s'effectue cette fois au moment du chargement et de l'édition de liens, juste avant l'exécution du programme (voir Fig.2.3).
Cette solution offre un compromis entre l'efficacité d'exécution et le niveau de flexibilité offert. Le surcoût induit l'est uniquement au moment du démarrage de l'application ou du chargement d'une nouvelle classe.

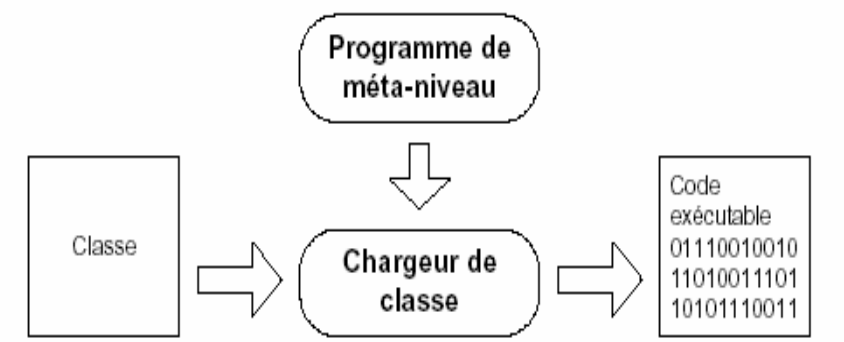


Fig.2.3- Réflexivité au chargement [42]

A l'exécution: Cette fois, l'architecture réflexive est présente à l'exécution (voir Fig 2.4). L'exécution d'un programme du niveau de base est adaptée en accord avec le programme de méta-niveau associé. Une architecture de ce type présente un niveau de flexibilité important. Le comportement d'un programme est paramétrable directement à l'exécution. On peut ainsi adapter le comportement d'un objet dynamiquement en lui associant un méta-objet adéquat, en lui retirant son méta-objet ou en le remplaçant, ce qui ne serait pas possible à la compilation ou au chargement.

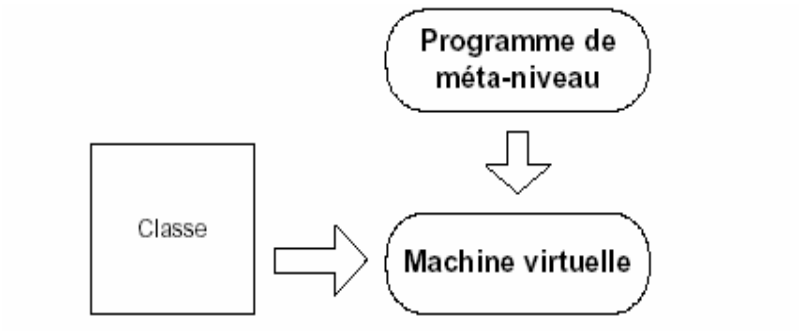


Fig 2.4- Réflexivité à l'exécution [42]

4.2 Réflexivité explicite / implicite:

Une architecture réflexive est dite de type explicite si le protocole à méta-objets chargé de la communication entre le niveau de base et le méta-niveau est visible à partir du niveau de base. Si celui-ci est invisible au niveau de base, l'architecture réflexive est dite de type implicite [42].

Réflexivité explicite: Dans une architecture réflexive de type explicite, les objets de base contrôlent directement leur propre comportement en ordonnant des changements dans leur méta-niveau. Un objet de base peut par exemple demander l'instanciation d'un nouveau répliqua au méta-niveau, sa migration vers un lieu donné ou encore sa mutation en objet persistant.

Réflexivité implicite: Au contraire, dans une architecture réflexive de type implicite, le protocole à méta-objets mettant en évidence les interactions entre les deux niveaux de la structure réfleive, est conçu pour être totalement transparent pour les objets du niveau de base. La prise de contrôle du comportement d'un objet de base par un méta-objet s'effectue le plus souvent par des mécanismes d'interception de messages. Du fait de la transparence du protocole à méta-objets, la sémantique d'un objet de base peut être modifiée au méta-niveau sans changer une seule ligne de code du niveau de base.

4.3 Réflexivité structurelle / comportementale

Réflexivité structurelle: Ce type de réflexivité touche aux aspects les plus statiques [40]. Il réifie les aspects structuraux du programme, comme les graphes d'héritage et de composition, ou encore les types de données (voir Fig. 2.5). Un système de ce type permet donc de modifier la structure interne d'un objet. Cela oblige à maintenir une description complète du niveau de base d'un programme pouvant donc se traduire par un surcoût important.

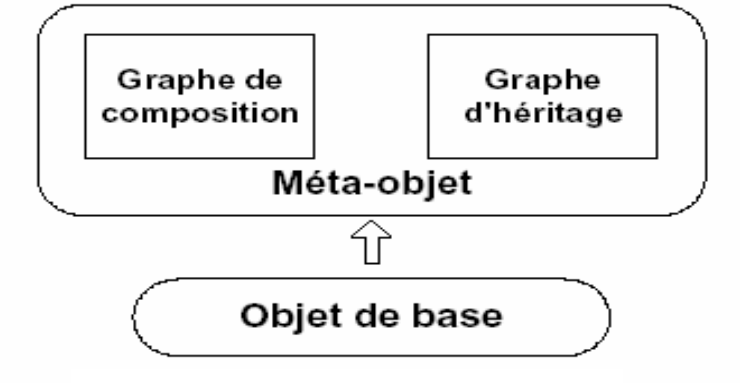


Fig 2.5 Réflexivité structurelle [42]

De plus, les processus de réification peuvent être parfois assez compliqués à manipuler. C'est pourquoi cette technique est relativement peu employée dans les systèmes d'exploitation, mais est assez largement répandue dans les langages extensibles (langages dont la sémantique et la syntaxe peuvent être étendues) [42].

Réflexivité comportementale concerne tout ce qui touche aux aspects les plus dynamiques, liés à l'état d'exécution et à la façon d'exécuter les programmes [40]. Elle réifie les aspects de calcul et de comportement d'une application (voir Fig . 2.6). Un système de ce type va par exemple proposer deux implémentations alternatives d'un seul et même module [42].

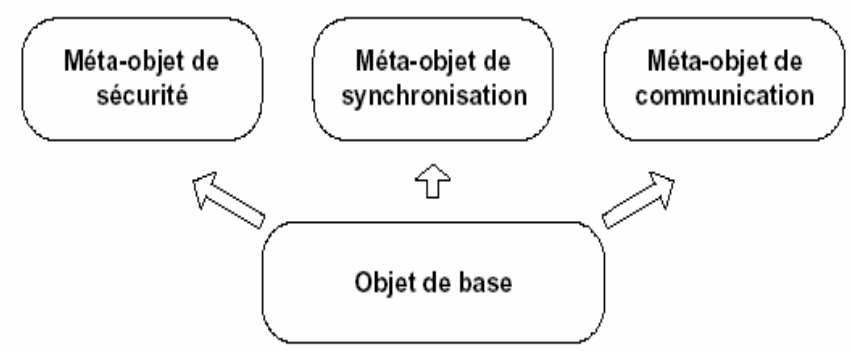


Fig 2.6 Réflexivité comportementale [42]

Ce type de réflexivité est utilisé principalement dans les systèmes d'exploitation et les systèmes de communication, pour par exemple changer les politiques de gestion de mémoire, de synchronisation, ajouter des instructions de monitoring, changer de protocole de communication, changer de comportement etc.

Systèmes mixtes: Dans un certain nombre de systèmes récents, le type de réflexivité offert est mixte (à la fois structurel et comportemental). Blair, propose une architecture réflexive pour les middleware de demain offrant les deux types de réflexivité. Dans cette architecture, chaque objet est associé à un *méta-espace* (voir Fig.2.7). Chaque méta-espace est divisé en trois parties, composition, encapsulation et environnement. Les parties composition et encapsulation fournissent des interfaces d'accès à des représentations des constituants d'un objet, de leur graphe d'héritage et de leurs méthodes et attributs. Elles autorisent ainsi une réflexivité structurelle. La partie environnement permet de contrôler tous les aspects de bas niveau de l'application, comme la gestion de l'envoi et la réception de messages, la synchronisation de processus, etc... La réflexivité est cette fois de type comportemental [42].

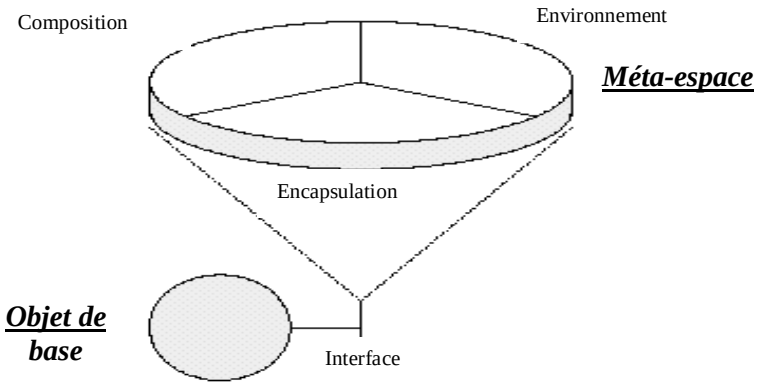


Fig.2.7- Structure d'un méta-espace [42]

5. Problèmes posés par les architectures réflexives

Cette partie traite des principaux problèmes auxquels se heurtent les développeurs lors de la conception et de l'implémentation des architectures réflexives [42,7].

Connexion causale: D'après Pattie Maes[39], un système est dit causalement connecté à son domaine si les structures internes et le domaine qu'elles représentent sont liés de telle sorte qu'un changement sur l'un se traduit par un effet sur l'autre. Autrement dit, tout changement effectué par le méta-niveau sur sa propre représentation du niveau de base doit se répercuter sur la réalité du niveau de base et inversement. Le problème consiste à trouver le moyen de préserver en permanence, une vision cohérente d'un tel système.

Séparation des aspects: Selon Kiczales, la partie non fonctionnelle d'une application peut se décomposer en différents aspects qu'il est utile de séparer au niveau du code, tels que, la synchronisation de processus, la sécurité, la gestion de pannes, et les services de persistance et de mobilité. Un aspect se distingue d'un autre s'il est parfois souhaitable de réifier l'un mais pas l'autre. Si tel est le cas, ces deux aspects devront être développés dans deux entités distinctes. La modularité ainsi obtenue permet d'augmenter considérablement l'adaptabilité et favorise la réutilisation.

Composition des aspects: Même si deux aspects semblent tout à fait indépendants, il arrive fréquemment qu'ils soient liés d'une manière ou d'une autre. Les aspects de synchronisation de processus sont par exemple souvent nécessaires aux aspects concernant la sécurité. Il est donc nécessaire de trouver une méthode de

composition des aspects. Ce problème est loin d'être trivial, et peu de solutions existent à l'heure actuelle. Mulet, par exemple, a proposé une solution dans laquelle il traite le problème de composition de méta-objet comme un cas particulier de la composition d'objets. Une relation orientée d'agrégation ou de spécialisation associe les différents méta-objets. Cette solution impose toutefois une conception particulière des méta-objets pour faire apparaître les liens de composition.

Méta-circularité: Les systèmes réflexifs dans lesquels le langage défini par le méta-niveau est utilisé pour la propre implémentation du méta-niveau sont appelés systèmes réflexifs méta-circulaires. De tels systèmes sont facilement et rapidement extensibles, car les nouvelles fonctionnalités du système sont implémentées en utilisant les facilités offertes par le langage en cours d'extension. Toutefois, la propriété de méta-circularité est parfois dommageable à la clarté du système. Il peut y avoir une confusion entre le langage avant l'extension et le langage obtenu après l'ajout de fonctionnalités. Les concepts de classe des langages étendus et non étendus peuvent être différents.

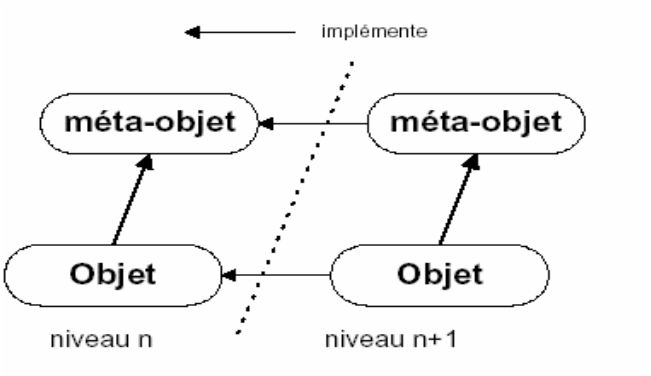


Fig 2.8- Relation de méta-circularité [42]

Tour réflexive infinie: Conceptuellement, rien n'interdit à un méta-objet d'être l'objet de base d'un autre méta-objet. Une architecture réflexive peut posséder donc plusieurs méta-niveaux (empilés), éventuellement infinie. Une telle structure est nommée *tour réflexive infinie (infinite reective tower*). (voir figure 2.9).

Dans la pratique, il est impossible d'implémenter une architecture basée sur une tour réflexive infinie. Différentes techniques à cet effet. La première est la plus simpliste et la plus rigide, elle consiste à limiter physiquement le nombre de méta-niveaux, en ne réservant en mémoire qu'un espace correspondant à un nombre fini de niveaux.
Une deuxième, plus souple, consiste à créer une nouvelle couche de méta-niveau uniquement sur demande explicite.

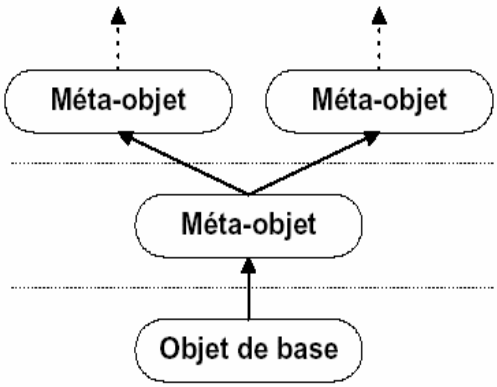


Fig.2.9- Tour réflexive infinie [42].

III) QUELQUES TRAVAUX TRAITANT L'ADAPTABILITE & LA REFLEXIVITE

ý N. Amara-Hachmi et A. El Fallah-Seghrouchni [3] considère que le paradigme orienté composants est mieux adapté pour la conception des systèmes ouverts, complexes et évolutionnaires .Il permet d'augmenter la productivité et diminuer le coût de développement des logiciels. Elles ont proposé à cet effet un modèle d'architecture basé composants pour un agent mobile de manière à accroître la modularité, la réutilisabilité, l'extensibilité, et l'auto-adaptabilité. Lorsque l'agent mobile se déplace d'un hôte à un autre, le contexte change. Le résultat de l'adaptation dynamique est la sélection des composants adéquats et la mise en relation des composants sélectionnés pour le nouveau contexte.

ý N. Amano & T. Watanabe [2] considèrent que l'adaptabilité dynamique constitue un mécanisme de base d'évolution du logiciel. À cet effet, ils ont proposé un modèle adaptable nommée *"DAS"* et son langage de description nommée *"LEAD++"*. Ils ont insisté sur les besoins de sûreté de l'adaptabilité dynamique, via un modèle DAS amélioré basés-composants nommé *"Safe DAS"*. La sûreté vérifie que le comportement adaptable du système ne viole pas sa consistance. Le modèle *"Safe DAS"* supporte des exceptions & les mécanismes des assertions.

ý Dans le cadre de leurs travaux sur l'aide au développement d'applications réparties ouvertes, S. Leriche & J-P Arcangel [37] ont proposé un modèle d'agent mobile auto-adaptable ainsi que son implémentation au sein d'un middleware Java. Ils ont étudié une évolution de ce modèle afin de lui donner davantage de flexibilité. Ils ont discuté le besoin de flexibilité et la possibilité de définir différents modèles d'agents par le biais d'un méta-niveau configurable. Puis, ils ont présenté les éléments d'une architecture à base de micro-composants permettant l'adaptation dynamique.

ý T. Ledoux & M.N.Bouraqadi-Saâdani [34] présentent leur contribution bâtie avec la mobilité du code. Leur étude porte sur l'adaptation des systèmes d'agents mobiles au cours d'exécution, dans le but de faciliter la construction des systèmes ouverts et garantir une meilleure QdS. Dans leur travail, ils ont utilisé la *réification* des différents aspects du système permettant ainsi la "*séparation des problèmes*". L'idée principale est que *l'introspection* du réseau peut être utilisée pour choisir

dynamiquement la meilleure politique de l'exécution. L'approche proposée permet cette faculté d'adaptation de l'infrastructure des systèmes d'agents mobiles en se basant sur le modèle de la réflexivité.

ý Pour supporter le développement des applications devant s'adapter aux changements de l'environnement, D. Chefrour & F. André [14], proposent un modèle de composants auto-adaptatif qui se repose sur un mécanisme de notification par événement des variations de l'environnement. Ils adoptent le principe de fournir l'accès à l'adaptation des différents niveaux depuis l'intergiciel. Et que les services offerts par cet intergiciel ne doivent pas être implantés sous forme de boite noire qui rend l'adaptation transparente. Cela permet aux applications d'intervenir et de faire des choix d'adaptation. Le modèle *"ACEEL"* est conçu selon un modèle réflexif. Il comporte deux niveaux séparés : Le niveau de base incluant l'objet « *context»,* réalise l'interface. Et le niveau méta, contenant l'objet « *adapter »* qui prend en charge l'auto-adaptation en utilisant une politique d'adaptation sous forme de script s'appuyant sur le système de détection-notification.

ý W. Zhi & G. Zhogwen [62] Présentent un mécanisme d'adaptation dynamique de sécurité en utilisant le modèle de conception orienté objets. Utilisant ce mécanisme, différentes implémentations de sécurités peuvent être remplacées à tour de rôle dynamiquement. Le but étant *l'adaptation des procédures de la sécurité*. L'approche proposée met en évidence un cadre de travail dans le quel les agents mobiles peuvent choisir dynamiquement une implémentation de sécurité parmi plusieurs. La solution de l'adaptation dynamique de sécurité mise en évidence fait que l'agent mobile change l'implémentation de sécurité selon le changement de l'état de l'environnement dans lequel il s'exécute. L'adaptation dynamique de la sécurité, selon Zhi et Zhogwen, augmente l'efficacité de l'agent mobile.

Discussion : L'adaptabilité présente désormais un sujet d'actualité. En effet, un nombre intéressant de travaux ont vu le jour pour traiter cet aspect, et contribuant ainsi à son développement. Nous avons cité entre autre quelques uns présentant des idées intéressantes. Notons ainsi que beaucoup d'entre eux ont confirmé le rendement et l'efficacité que présente la réflexivité pour la conception et l'implantation de l'adaptabilité [14,34,37,…].

CONCLUSION

Avec l'énorme expansion de l'Internet et les facilités de collaboration permises, la technologie de distribution gagne de plus en plus de popularité. La réalisation de la qualité de service comprenant la sécurité des systèmes informatiques -*qui est en fait notre objectif*- trouve une solution dans la technologie d'adaptabilité, qui permet aux applications de fournir leurs services malgré les changements pouvant être apportés aux environnements d'exécution.

La nouveauté de l'idée de l'adaptabilité et la courte période de recherche font que la problématique est loin d'être résolue. De plus, la co-existence des plusieurs approches permettant son implantation montre bien que pour l'instant il n'y a pas de solution universelle qui réconcilie l'efficacité et la facilité, Cependant la réflexivité semble construire une méthode encourageante pour la conception de tels systèmes.

En se basant sur l'étude faite si dessus, nous dégageons que la réflexivité fourni un bon support pour la réalisation de l'adaptabilité, du fait qu'elle fourni une bonne représentation de soi même tout en rendant possible la manipulation et par conséquent la modification de cette représentation. Un système réflexif est un système qui possède une auto-représentation décrivant la connaissance qu'il a de lui-même. Un tel système peut alors répondre à des questions le concernant (capacité *d'introspection*), mais aussi s'auto-modifier (capacité *d'intercession*).

Dans le cadre de notre travail, nous proposons de protéger l'agent mobile contre quelques attaques des plateformes malveillantes, par une politique de protection basée sur *l'adaptabilité dynamique*. Nous envisageons de doter l'agent mobile d'une *adaptabilité des fonctionnalités, conçu statiquement et exécutée dynamiquement*. Nous essayons, en effet, d'estimer les différentes variations possibles de l'environnement de travail, que peut rencontrer notre agent mobile durant son cycle de vie, et de définir un ensemble de règles lui permettant de décider et d'entreprendre les actions adéquates. Notre agent mobile constitue donc un système *adaptatif*. Nous allons le concevoir suivant une approche *explicite* en adoptant la démarche *réflexive*. Nous adopterons pour ce faire, *la réflexivité comportementale*, se prêtant bien à notre cas de figure. Bien entendu, la relation entre les objets des deux niveaux mis en évidences par la structure réflexive, aura lieu lors de *l'exécution*, ce qui fournit le niveau de l'adaptabilité souhaité. Au niveau du chapitre suivant, nous allons détailler l'approche de protection que nous venons d'évoquer.

Chapitre III

APPROCHE PROPOSÉE POUR LA PROTECTION DE L'AGENT MOBILE

« Je ne me préoccupe pas des vulnérabilités que je connais car j'ai déjà pris des précautions à leur encontre, ce sont ceux que j'ignore qui me causent du souci» Marcus Ranum

Ce chapitre met en évidence notre contribution dans le cadre de la sécurité des systèmes d'agents mobiles. Il explique de manière détaillée l'approche que nous proposons pour la protection de l'agent mobile. Il cerne en premier lieu l'étendu du travail. Ensuite, il montre via une structure proposée pour l'agent mobile la stratégie qu'adoptera ce dernier durant son cycle de vie. Bien entendu, cette stratégie lui permet de se protéger contre les éventuelles attaques des hôtes malveillants de type analyse de comportement.

INTRODUCTION

La sécurité, comme nous l'avons déjà mentionné plus haut, présente un point crucial dans les systèmes d'agent mobile, et risque de freiner l'expansion et l'utilisation de ce paradigme. Compte tenu de l'étude que nous avons faite sur ce problème, nous constatons qu'il n'y a malheureusement aucune méthode qui permet de répondre de manière exhaustive et efficace à tous les besoins de la sécurité. Par ailleurs, essayer de s'attaquer à tous les problèmes de sécurité se révèle être une expérience irréalisable. D'où la nécessité de procéder par bien cerner le cadre du problème, et de tracer en conséquence une trajectoire de résolution étudiée soigneusement.

Dans le cadre qui nous préoccupe, nous nous intéressons à la protection de l'agent mobile contre les attaques des hôtes malveillants. En fait, certaines recherches ont vu le jour, ayant le même objectif. Ces derniers bien qu'ils révèlent des idées intéressantes, peuvent être critiqués dans leur majorité, par le fait qu'ils ne s'adaptent plus lorsque les conditions d'exécution changent. En revanche, peu de travaux ont essayé dernièrement, de remédier à cette limite. Parmi lesquels nous mentionnons celui de Mr 'Wang Zhi' et al [62]. Cependant, l'adaptabilité proposée par ce dernier est assez limitée et ne répond malheureusement pas à l'ampleur de la diversité et des variations des contraintes de travail envisageables.

Dans notre travail, et dans le but d'offrir un niveau de sécurité plus satisfaisant, nous optons à pousser encore cet aspect d'adaptabilité, nous essayerons d'exploiter les opportunités offertes par cette technique, pour les utiliser dans le compte de la sécurité. Nous nous inspirons pour ce faire, des travaux qui ont été fait à cet effet. Et en particulier ceux de Mme 'Hacini' et al [24,25] qui nous servirons de base pour l'élaboration de notre contribution. La solution de sécurisation que nous proposons, utilise l'adaptabilité pour modifier le comportement de l'agent mobile dans le but de le rendre imprévisible de la part les hôtes qui l'accueillent et l'exécutent. L'idée est de doter l'agent mobile d'une capacité de flexibilité lui permettant de diversifier ses traitements et compliquer ainsi leurs analyses, en vu de le protéger contre les tentatives d'espionnage. Dans le but de renforcer le niveau de sécurité présenté, des mécanismes de protection classiques tels que la cryptographie et la signature numérique sont aussi utilisés.

Dans le reste de ce chapitre, nous détaillerons l'approche de protection proposée. Nous commençons par cerner le cadre de travail. Nous développerons ensuite la stratégie de protection de l'agent mobile contres les attaques des hôtes malveillants auxquelles nous nous intéressons. Nous illustrons la structure proposée pour notre agent mobile à travers un schéma, dans lequel nous citerons les fonctionnalités nécessaires. Ensuite, nous expliquerons ces dernières avec plus de détails.

1. Délimitation du cadre de travail

Nous avons vu, que la sécurité est un problème assez large et délicat. Et que, répondre à tous les besoins de sécurité tout en l'assurant à 100% est une mission pratiquement impossible, plusieurs raisons:[29]

- Les bugs dans les programmes courants et les systèmes d'exploitation sont nombreux.
- De nouvelles technologies (et donc vulnérabilités) émergent en permanence.
- Plus les mécanismes de sécurité sont stricts, moins le système est efficace.
- On peut s'attaquer aux systèmes de sécurité eux-même.
- Même un système fiable peut être attaqué par des personnes abusant de leurs droits.
- La cryptographie a ses faiblesses: les mots clés peuvent être cassés…

Cependant, il est toujours possible d'assurer un niveau acceptable de sécurité par la mise en œuvre et le suivi d'une stratégie sécuritaire bien fondée et bien étudiée. Le contexte qui nous intéresse concerne la sécurisation de l'agent mobile contre les attaques des hôtes malveillants. Toutefois, et comme nous l'avons déjà vu, les attaques des hôtes malveillants contre un agent mobile sont aussi nombreuses et de différents types. Elles incluent les attaques passives, visant de manière générale à introduire d'autres types d'attaques, telle que la mascarade et l'espionnage. Elle comprend aussi les attaques actives qui modifient les données, le code, l'état ou les communications de l'agent mobile telles que l'altération et la re-exécution. D'un autre côté, les besoins de sécurité sont multiples. À savoir, la confidentialité, l'intégrité, l'authentification, la disponibilité et le contrôle d'accès. De plus, les mesures sécuritaires sont de différents niveaux; les mesures visant la détection, prévention ainsi que la palliation et récupération.

Dans le cadre de notre travail, nous nous limitons à la protection de l'agent mobile en prévenant les éventuelles attaques des hôtes qu'il doit visiter durant son cycle de vie. Ceci permet de minimiser l'ampleur des dégâts occasionnés. Plus précisément, nous comptons protéger l'agent mobile principalement contre les attaques de type *"analyse de comportement*" appelés aussi "*espionnage de comportement*" ("*Eavesdropping*" en anglais). Nous rappelons, que ce type d'attaques peut avoir lieu lorsque l'accès direct aux informations secrètes de

l'agent mobile devient impossible ou bien à la rigueur trop difficile ou trop coûteux -à cause des éventuelles précautions et des mécanismes de sécurisations que l'agent a mis en œuvre-. Dans ce cas, l'hôte malveillant essayera d'observer l'agent, d'étudier son comportement et de l'analyser à chaque fois qu'il se déplace au sein du réseau. Cela lui permet de dégager et de déduire sa stratégie ou rassembler des renseignements confidentiels afin de les utiliser par la suite pour des besoins personnels (voir chapitre I).

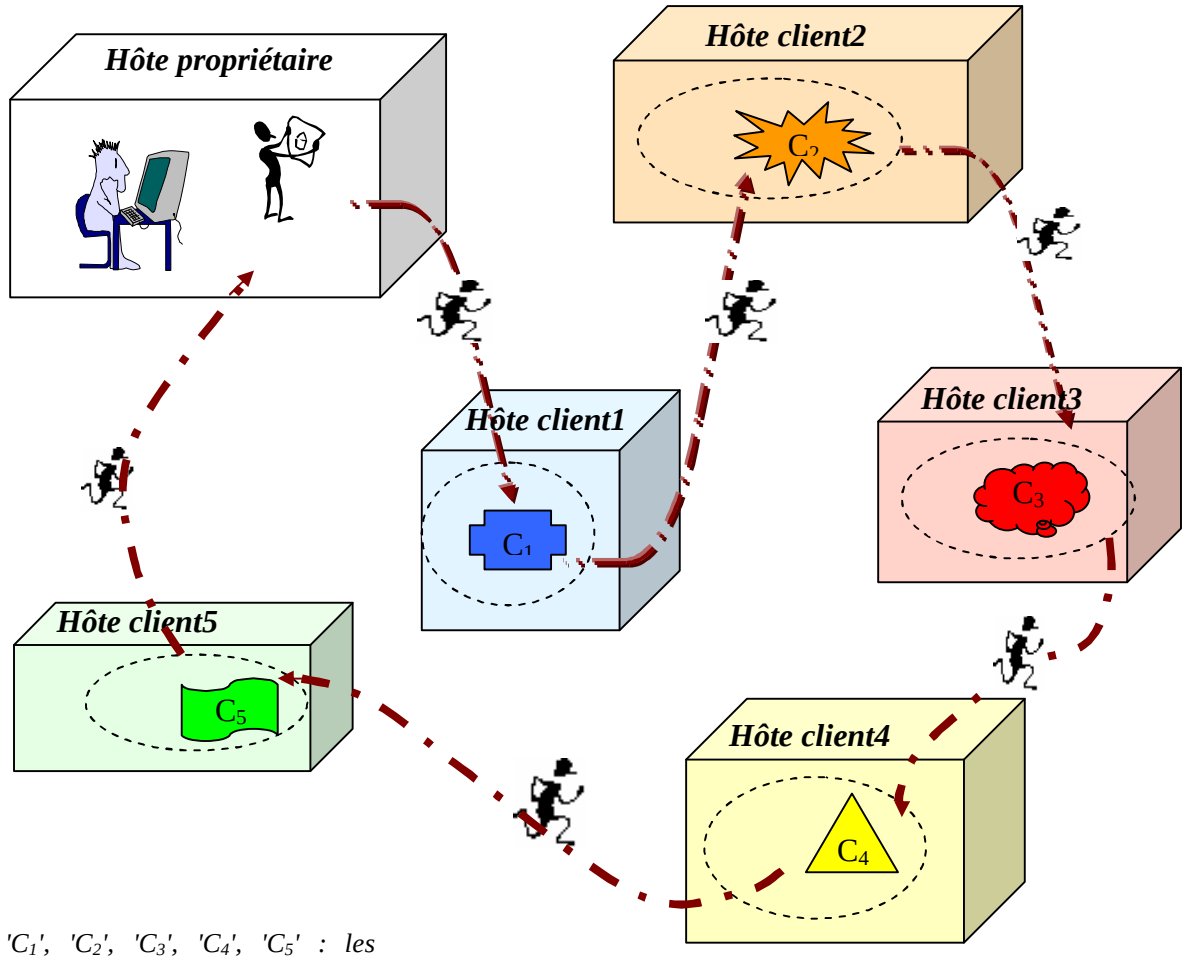
En effet, assurer la confidentialité de l'agent revient à protéger ses informations contre une divulgation non autorisée. La prévention contre ce type d'attaques a pour finalité d'empêcher l'aboutissement de l'agression.

2. Description de l'approche de protection proposée

Notre stratégie de sécurisation consiste à faire en sorte que l'agent mobile se comporte, au cours de son cycle de vie, d'une manière imprévisible par les hôtes exécuteurs. Il s'agit de lui affecter une aptitude de flexibilité lui permettant de modifier son traitement et de diversifier ses apparences, de telle sorte à donner un comportement varié et différent à chaque fois (voir Fig 3.1). Ceci empêche l'hôte malveillant de comprendre sa stratégie ou de déduire des informations le concernant. Ce qui rend toute tentative d'analyse de comportement difficile voire inefficace.

Le changement du comportement de l'agent mobile doit être réalisé en tenant compte des variations possibles de l'environnement en question. Celles-ci peuvent avoir trait au niveau de sécurité détecté, au service demandé, aux exigences imposées ou aux opportunités offertes. Leur analyse permet d'engendrer le choix du traitement approprié.

Ceci conduit l'agent mobile à s'ajuster aux nouvelles conditions d'exécution et s'y adapter afin de pouvoir poursuivre son travail. Il s'agit en fait, d'une *adaptation conçue statiquement et exécutée dynamiquement*.



'C_1', 'C_2', 'C_3', 'C_4', 'C_5' : les différents comportements de l'agent mobile.

Fig 3.1- changement du comportement de l'agent mobile selon l'environnement

2.1 Stratégie de protection

L'approche proposée consiste donc à sécuriser l'agent mobile via une politique d'adaptation dynamique. Par conséquent, nous devons faire appel aux approches de l'adaptabilité dynamique. Nous essayons de profiter et de bénéficier des opportunités et des facilités offertes par cette technique que nous avons déjà étudiée au niveau du chapitre II.

Dans le but d'augmenter le niveau de sécurité proposé et de présenter un modèle de protection satisfaisant, nous nous servons aussi des mécanismes de protection classiques tel que le mot de passe, la cryptographie et la signature numérique. Ces dernières permettent ainsi de préserver l'intégrité des données de notre agent mobile et de contrôler l'accès à ses ressources.

3. Principes structuraux adoptés

3.1 Structure réflexive

Nous avons vu qu'une structure réflexive représente un bon support pour l'adaptabilité, du fait qu'un système réflexif est capable d'inspecter et de modifier sa propre structure et sa sémantique de la même manière qu'il opère pour son domaine applicatif. [55].

En effet, afin d'effectuer les bons choix et s'adapter au contexte d'exécution, notre agent mobile doit connaître non seulement l'état de son l'environnement d'exécution, mais aussi son propre état et sa structure. Nous avons appelé « *niveau de base »,* le niveau habituel de programmation d'une entité spécifiant ainsi le comportement fonctionnel de l'application, appelé aussi dans certains langages *"comportement de base"*. Le *méta-niveau* concerne sa description, et permet de réaliser des transformations sur le niveau de base. De même, ce niveau spécifie le *"méta-comportement*" (voir Fig 3.2).

Cette séparation entre niveaux permet de rendre transparent au niveau de base un changement structurel ou opératoire dans le méta-niveau.

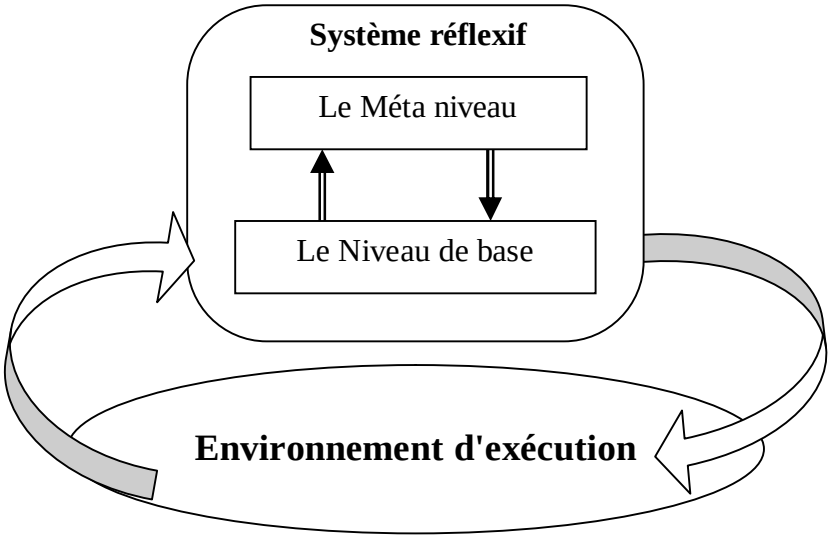


Fig 3.2- Modèle d'exécution réflexif

3.2 Structure modulaire

Nous avons vu également, que la modularité se prête mieux à l'adaptabilité et facilite ainsi son implantation. Une structure atomique du code ne peut être adaptée qu'en le remplaçant par un autre, de façon intégrale. Alors qu'un code partitionné en plusieurs modules (micro-composants) peut être facilement adapté. En particulier un paradigme orienté composants, en

plus des facilités et des perspectives intéressantes qu'il présente -en augmentant la productivité et en diminuant le coût de développement- est mieux adapté à ce type d'applications. L'adaptation se réduit au choix des micro-composants et du séquencement de ces derniers définissant ainsi un traitement précis. En conséquence, un remplacement d'un micro-composant par un autre ou une modification au niveau de la séquence des micro-composants permet de spécifier un comportement différent.

Dans le cadre de notre travail, le paradigme orienté composants nous semble un bon candidat pour supporter la stratégie de protection de notre agent mobile. De plus, la conception par composant spécifiée permet de spécialiser la liaison devant être établie entre le méta-niveau de l'agent mobile agent et son niveau de base. Nous avons proposé [26] de placer les composants fonctionnels au niveau de base. Tandisque les composants non-fonctionnels sont regroupés dans le méta-niveau (voir Fig, 3.3). Cela facilite l'adaptation dynamique de l'agent mobile.

4. Structure proposée pour l'agent mobile

4.1 Structure réflexive adaptative

Pour obtenir la flexibilité souhaitée, nous avons adopté une structure réflexive qui nous permet de supporter la stratégie décrite ci-dessus et d'offrir une meilleure efficacité du traitement adaptatif. Il s'agit d'une réflexivité comportementale mettant en évidence deux niveaux conceptuels:

(i) *Le niveau de base* contenant le code fonctionnel et formulant l'implémentation logique de l'application et sa sémantique. Il comporte trois composants:

- *Une interface*, permettant à l'agent mobile de communiquer avec son environnement extérieur.
- Une *mémoire*, au niveau de laquelle l'agent mobile sauvegarde ces informations qui lui sont nécessaires pour l'accomplissement de ses traitements.
- Et une *bibliothèque*, contenant les différents micro-composants du code de l'agent mobile.

(ii) Le *méta niveau,* comprenant le code non fonctionnel exhibant les divers services qui supportent le fonctionnement de l'application, et décrivant la manière d'assurer l'adaptation de l'application (voir Fig.3.3). Pratiquement, ce niveau renferme la portion du code qui assure l'adaptation de l'agent. Nous l'avons appelé: *Adaptateur*.

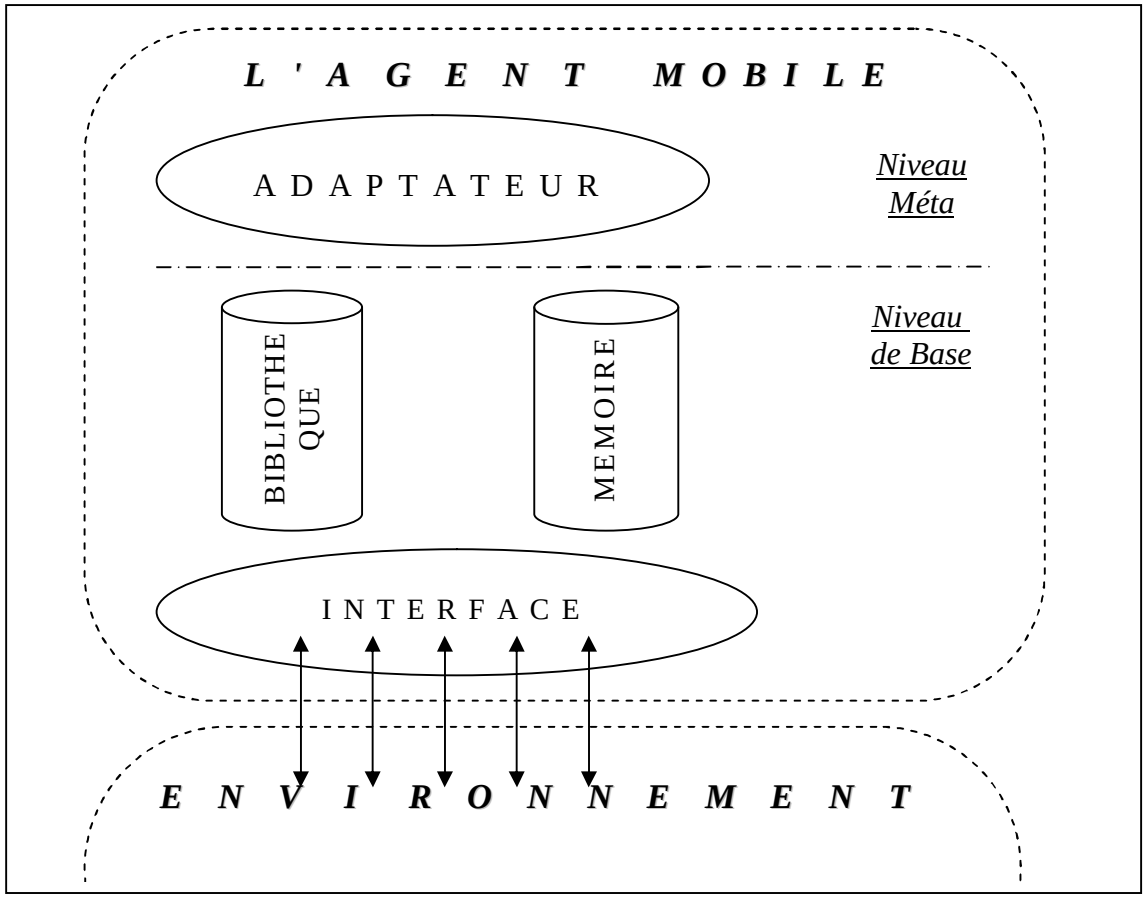


Fig.3.3- Structure réflexive d'un agent mobile adaptable [26].

Notons que l'environnement de travail au niveau de l'architecture montrée ci-dessus correspond à l'hôte d'accueil de l'agent mobile qui va l'héberger et l'exécuter.

4.2 Structure détaillée

Pour une identification précise des rôles et donc pour mettre en lumière les liens mis en évidence inter-niveaux ainsi qu'intra-niveaux, nous proposons une architecture détaillée au niveau de laquelle nous distinguons:

(i) Des composants de granularité fine, implémentant les fonctionnalités de notre agent mobile, nous les appelons micro-composants. Ces derniers n'offrent qu'une unité de service opératoire. La combinaison et le séquencement entre ces micro-composants devant être préalablement étudiés et spécifiés. Cette étude permet de donner une tâche précise présentant ainsi le service espéré de l'agent mobile.

(ii) Des composants de granularité plus large, communément appelés "*Composants*" ou "*Modules*", implémentant dans leur ensemble l'aspect non fonctionnel de notre agent mobile et spécifiant dans certaines mesures, la façon dont le code fonctionnel sera effectué. En fait, c'est la synchronisation et la coordination entres ces composants qui permet de supporter la stratégie sécuritaire de notre agent mobile. Cela nécessite d'indiquer comment et dans quelles conditions les *"micro-composants"* préalablement définis, devront être utilisés en vu d'assurer ainsi le bon fonctionnement de l'agent mobile.

La structure que nous venons de présentée est étalée à travers l'architecture illustrée par la figure 3.4. Au niveau de cette dernière, nous reprenons le système de la figure 3.3, qui se compose de notre agent mobile adaptable et l'hôte exécuteur représentant son environnement de travail.

Nous avons repris bien évidemment les composants de base de la structure que nous avons proposée ; à savoir, l'interface, la mémoire, la bibliothèque et l'adaptateur. Ce dernier se chargera, de réaliser l'adaptation de l'agent mobile selon une certaine politique dite "politique d'adaptation", implémentant ainsi sa stratégie de sécurisation. Cette politique se base sur quelques règles regroupées au niveau de la "Base de règles d'adaptation". Compte tenu des conditions de l'environnement d'exécution, notre agent mobile, va se servir de cette base des règles d'adaptation, pour décider du traitement qu'il doit exécuter. Ceci va lui permettre de présenter à chaque fois un comportement différent.

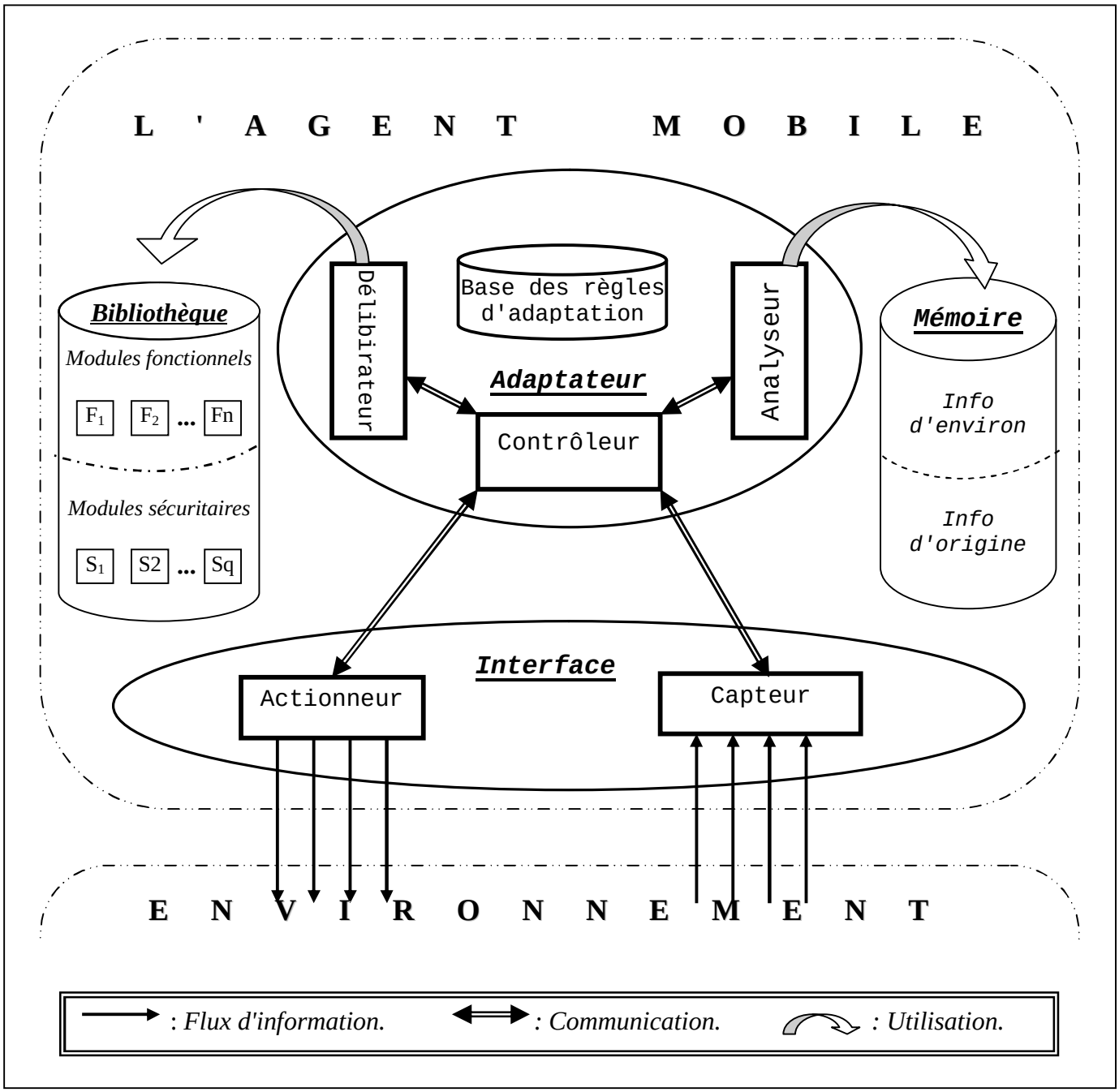


Fig 3.4- Structure Sécuritaire détaillée de l'agent mobile adaptable [26]

4.3 Description des composants

4.3.1 L'interface: C'est le composant à travers lequel l'agent communique et interagit avec l'environnement extérieur. Il contient les dispositifs nécessaires lui permettant d'authentifier les clients, de détecter les accès des hôtes extérieurs à ses ressources, de recevoir les requêtes et de présenter ses services sous une forme adéquate.

L'interface réalise toutes ces tâches à travers deux sous composants; le "Capteur" et "l'Actionneur" :

(i) *Le Capteur*: il permet à l'agent mobile de percevoir l'environnement d'exécution afin d'être informé des variations de l'environnement et d'acquérir des informations nécessaires. Nous pouvons envisager trois types d'acquisition, à savoir: l'observation, l'inspection et l'interaction [25] (voir Fig. 3.5).

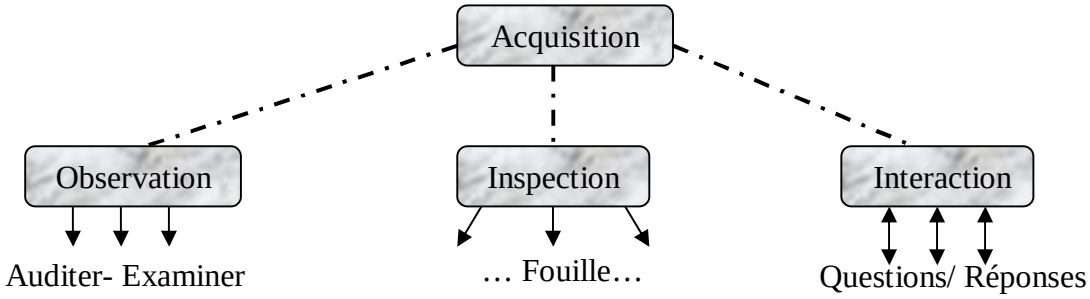


Fig.3.5- Types d'acquisitions

Le type d'acquisition dépend du type de l'agent mobile (logiciel/matériel) ainsi que des informations demandées (identité, mot de passe, contrat, certificat, infrastructure…). En conséquence, le capteur peut prendre diverses formes (caméra, ondes, formulaire…)

(ii) *L'actionneur*: permet l'exécution de la séquence d'actions délibérée représentant dans leur ensembles le service attendu de l'agent mobile.

Les actions à exécuter peuvent varier d'une simple notification ou alerte, à un changement du traitement, jusqu'à arrêter le traitement ou encore quitter l'hôte définitivement.

Là aussi, suivant le type de l'agent mobile et de la nature du traitement devant être exécuté, l'actionneur peut prendre plusieurs formes.

4.3.2 La mémoire: Contient les informations nécessaires pour l'accomplissement du travail de l'agent mobile. Ces informations concernent l'agent lui-même ou bien les hôtes clients avec qui il doit travailler. Nous pouvons distinguer deux catégories d'informations au niveau de cette mémoire.

(i) La première regroupe ce que nous avons nommé "l*es informations d'origine*". Ce sont celles transportées à partir de son hôte d'origine représentant les données qui lui sont nécessaire pour exécuter son code, (exemple les paramètres nécessaires pour le calcul du degré de confiance) ou bien celles lui permettant de vérifier la validité de la deuxième catégorie d'informations (exemple: les identités et les mots de passes des différents hôtes à visiter). Pour ne pas surcharger l'agent mobile. Ce dernier sauvegarde les informations collectées dans les mêmes emplacements tout en écrasants les anciennes valeurs.

Pour des raisons de confidentialité, ces informations sont chiffrées à l'aide de la clé publique de l'hôte propriétaire, en utilisant un algorithme de cryptage asymétrique. Le choix du cryptage asymétrique ici, étant justifié par le fait d'éviter que l'agent mobile ne transporte une clé secrète au sein d'un réseau qui n'est pas forcément sécurisé.

(ii) La seconde catégorie comprend les informations que l'agent mobile doit collecter à partir de son environnement de travail via *le capteur*. Nous les avons, par conséquent, appelé "*Informations environnementales*". Cette dernière catégorie d'informations va permettre à l'agent mobile d'authentifier les hôtes visités, en les comparant avec les informations d'origine, et de lui faire connaître l'état de l'environnement de travail selon lequel il décidera du traitement à effectuer. Pour des raisons de performances –pour ne pas alourdir notre agent mobile-, les informations environnementales sont sauvegardées au niveau du même emplacement -en écrasant les anciennes valeurs- à chaque fois.

Puisque les informations d'origine sont chiffrées, l'agent mobile doit procéder au cryptage des informations environnementales avec la clé publique de son propriétaire avant de les comparer.

4.3.3 La bibliothèque: contient les microcomposants du code de l'agent mobile dont les différentes compositions envisagées génèrent les divers comportements de cet agent. Le but étant d'offrir autant de flexibilité que de confidentialité au traitement.

Nous proposons de spécifier des alternatives pour quelques micro-composants afin de varier les traitements. Nous envisageons aussi un ensemble de dégradations de services, qui seront effectuées en fonction des conditions d'exécution proposées. De plus, nous pouvons envisager des modules fictifs afin de compliquer les opérations d'analyse et d'augmenter en conséquence le niveau de sécurité.

Les diverses combinaisons des microcomposants sont données par un ensemble d'expressions abstraites utilisées pour implémenter les différents comportements de l'agent mobile. Soit E= $\{E_1, E_2...E_n\}$ l'ensemble de ces expressions et soit A= $\{A_1, A_2...A_p\}$ l'ensemble des microcomposants disponibles. Chaque expression Ei (i<=n) est une séquence d'appels aux modules Aj, et pouvant être vus comme une séquence de bits (chaque bit indique un module spécifique) [24].

Au niveau de 'A' l'ensemble des micro-composants, nous pouvons distinguer deux sous ensembles également.

(i) Le premier regroupe les modules fonctionnels du code, implémentant les tâches élémentaires ou encore les unités de services mis en évidence. F= $\{F_1, F_2,\ldots F_q\}$ est l'ensemble de ces micro-composants.

(ii) Le deuxième sous ensemble englobe des modules de code implémentant des méthodes de sécurisation classique telles que les algorithmes de cryptage, de signature numérique,… S=$\{S_{q+1},\ldots,S_p\}$ est ce sous ensemble là. Le but de l'utilisation de ces derniers micro-composants, étant de renforcer le niveau de sécurité proposé par la stratégie adoptée par l'agent mobile.

Là encore, nous pouvons envisager plusieurs alternatives pour le même module sécuritaire (plusieurs algorithmes de cryptage par exemple) que pourra utiliser l'agent différemment à chaque fois, pour les même raisons de diversification de traitement.

La bibliothèque comprend donc des ressources critiques, nécessitant ainsi d'être protégées avec soin. Pour cette raison, nous proposons que son contenu soit signé par le propriétaire de l'agent. Cela permet de s'assurer de leur l'intégrité.

4.3.4 **L'adaptateur:** C'est le composant responsable de réaliser l'adaptation du comportement de l'agent mobile en fonction des informations détenues de l'environnement de travail. Ce composant implémente une politique d'adaptation, selon laquelle, un certain raisonnement peut avoir lieu. Cette politique est déterminée par un ensemble de règles regroupées dans une base dites "*Base des règles d'adaptation*". L'adaptateur, après avoir observé et analysé les conditions de travail dénotées par le capteur, décide du nouveau comportement à adopter. Cela est réalisé grâce à la synchronisation et la collaboration de trois sous composants, qui sont: le "*Contrôleur*", "l'Analyseur" et le "*Délibérateur*".

(i) *L'Analyseur*: Ce sous-composant exploite les informations contenues dans la mémoire et se sert de la *Base des règles d'adaptation* pour déterminer le traitement approprié pour une situation donnée. Il procède par l'observation et l'analyse de quelques informations environnementales -telles que l'identité de l'hôte en question et le mot de passe-, en les comparant avec les informations d'origine afin de s'assurer de leurs validités et d'estimer le degré de confiance de l'hôte visité. En plus, il étudie d'autres informations toujours collectées à partir de l'environnement de travail, lui permettant de connaître l'état ou encore le *contexte* d'exécution, ce qui l'aide à effectuer un premier filtrage des comportements à entreprendre.

(ii) *Le Délibérateur*: Un sous-composant qui détermine avec précision les actions à exécuter. Il génère une clé dite « clé environnementale » -du fait qu'elle dépende de l'environnement d'exécution-. Cette clé lui permet de préciser les actions devant êtres exécutées par l'actionneur. Ces actions constituent dans leur ensemble le nouveau comportement de l'agent mobile. La clé environnementale sera utilisée par la suite pour déchiffrer l'expression abstraite renfermant la séquence des micro-composants appropriée. Ensuite, Le délibérateur appelle les microcomposants précisés à partir de la bibliothèque afin de les envoyer à l'actionneur en vu d'exécution.

La clé environnementale est générée selon l'algorithme présenté au niveau de la section 8.

(iii) *Le Contrôleur*: Il joue le rôle de coordinateur entre les composants de l'agent mobile. Il permet aussi le déclenchement des opérations devant d'êtres exécutées par les composants adéquats. Il est responsable également du contrôle du bon fonctionnement de tout le système de l'agent mobile ; C'est le rôle également de ce sous-composant de détecter les éventuelles anomalies de traitements telle qu'une tentative à une mauvaise exécution de l'agent mobile de la part de la plateforme en question. Il peut vérifier cela en maintenant le temps nécessaire pour la réalisation de chacune des tâches confiées à l'agent mobile, et en observant le temps consommé lors de l'exécution. Si le contrôleur note un dépassement du temps réglementaire, il peut supposer une utilisation malveillante, et par conséquent, il doit arrêter le travail sur l'hôte en question pour migrer vers l'hôte suivant dans son itinéraire.

En résumé, ce composant déclenche le commencement du travail et assure la coordination et le bon fonctionnement des différents composants de l'agent mobile.

(iv) Base des règles d'adaptation : Cette base représente un élément très important pour supporter la stratégie de protection que nous proposons pour notre agent mobile. En effet, c'est en fonction de ces règles là que l'adaptation est effectuée. Cette base comporte des règles simples de type:

« ***Si*** *Condition* ***Alors*** *Action* »

Les parties gauches de chacune des règles d'adaptation présentant les conditions de leurs déclenchements. Elles correspondent aux paramètres de l'état de l'environnement. Les parties droites dénotent la suite des actions à exécuter pour le cas considéré.

Les règles que nous avons proposées sont génériques, indépendantes de toute application et de tout domaine d'application, offrant ainsi l'avantage de la flexibilité et de la réutilisation.

La synchronisation et le séquencement entre les règles que nous avons défini, sont exprimés à travers les scénarios d'exécution expliqués au niveau de la section 5.

La table 3.1 illustre quelques règles d'adaptation.

N° de règle	Condition	Action
1	Le degré de confiance ∈ [a1-b1]	Assurer un service complet
2	Le degré de confiance ∈ [a2-b2]	Assurer un service dégradé
3	Le degré de confiance ≤ Seuil	Arrêter, Notifier et Quitter
4	Absence Ressource non critique	Assurer un service dégradé
5	Ressource critique non disponible	Arrêter, Notifier et Quitter
6	Chemin déjà visité	Choisir une autre alternative
7	Échec génération Clé environnementale	Arrêter, Notifier et Quitter
8	Temps consommé ≥ limite	Arrêter, Notifier et Quitter
9	Blocage ou Échec Exécution	Voir exceptions
10	Échec Exceptions	Arrêter, Notifier et Quitter

Tab 3.1- Exemples de règles d'adaptation [26]

Afin de fournir plus de flexibilité, nous pouvons envisager un ensemble d'exceptions, supportant quelques anomalies de traitement.

Nous tenons à mentionner ici que la simplicité des règles d'adaptation proposées et par conséquent du raisonnement mis en évidence, offre un avantage en matière de performance. Cela permet d'alléger l'agent mobile. En fait la mobilité exige qu'il ne doit pas être chargé en vue de diminuer son coût de transfert, et par conséquent, minimiser la consommation de la bande passante.

5. Scénarios d'exécution

L'agent mobile transporte l'itinéraire à partir duquel il choisit à chaque fois le prochain client à visiter. Des informations relatives aux clients devants être visités, sont déplacées avec l'agent sous forme cryptée pour des raisons de confidentialité.

En arrivant sur l'hôte d'exécution, l'agent mobile doit authentifier l'hôte en question, et vérifier quelques informations nécessaires pour l'accomplissement de sa tâche. Pour ce faire, l'agent doit obtenir certaines informations à partir de l'environnement d'exécution (exemple: identité de l'hôte visité, mot de passe, …).

L'acquisition des données se fait grâce au capteur qui doit les ranger au niveau de la mémoire.

L'analyseur crypte alors les informations collectées à l'aide de la clé publique de l'hôte d'origine, pour les comparer avec celles qu'il détient, afin de pouvoir estimer le degré de confiance établi pour cet hôte.

Le degré de confiance permettra à l'agent mobile d'effectuer un premier filtrage de traitement à exécuter. Il lui permet de décider de s'exécuter sur cet hôte si le résultat appartient à un intervalle jugé très favorable (règle1), ou d'assurer un service dégradé dans le cas d'un degré de confiance acceptable (règle2), ou encore, s'arrêter et quitter l'hôte pour aller au suivant tout en notifiant la cause dans le cas restant (règle3).

Dans le cas où le degré de confiance (noté DG.Conf) est favorable ou à la rigueur acceptable, l'agent mobile, procèdera à la vérification des conditions de travail (Exemple: la disponibilité des ressources requises, la qualité,…). Le résultat de cette deuxième vérification représente un deuxième filtrage, lui permettant de préciser le service à assurer (règles: 4 et 5).

Un ensemble d'alternatives sont envisagées pour le cas où l'agent visite des hôtes là où les conditions de travail sont semblables. Si c'est le cas, l'agent mobile choisira la moins récemment utilisée (rège 6).

Une fois terminé, l'agent mobile passe à l'étape suivante, qui est la génération de la clé environnementale, lui permettant de déchiffrer l'expression abstraite dégagée. C'est le délibérateur qui se charge du calcule de cette clé, en se basant sur des informations toujours collectées à partir de l'environnement de travail.

Notons ici, que le déchiffrement des expressions abstraites par une clé générée à partir de l'environnement lors de l'exécution, constitue un autre contrôle et une autre vérification de la véracité de l'hôte receveur.

Si jamais, l'agent mobile n'arrive pas à générer cette clé, éventuellement, à cause de quelques informations erronées ou manquantes, le délibérateur en se basant sur la règle 7 demande au contrôleur d'arrêter et de quitter l'hôte actuel après avoir notifié la raison pour la quelle il a pris de cette décision.

Dans le cas où la génération de la clé environnementale se termine avec succès, le délibérateur, procèdera au décryptage de l'expression abstraite, lui permettant d'obtenir la séquence des micro-composants, les identifier et les appeler à partir de la bibliothèque.

Les résultats sont transmis au contrôleur, qui les renvoie à son tour à l'actionneur avec un ordre d'exécution.

L'actionneur se charge de leur exécution. Une fois terminé, il transmet au contrôleur un compte rendu pour qu'il décide de l'étape suivante.

Si le contrôleur constate que l'exécution d'une certaine tâche, consomme un temps dépassant celui estimé, il peut interrompre l'exécution pour quitter l'hôte et aller au suivant (règle 8).

Si encore, le contrôleur remarque la non progression dans une des phases, il déclenche une vérification de l'ensemble des exceptions devrant être effectué par l'analyseur pour une éventuelle récupération (règle 9).

Dans le cas où l'ensemble des exceptions ne répond pas, il arrêtera immédiatement pour aller suivant le cas, à un nouvel l'hôte ou bien revenir à son propriétaire (règle10).

Bien entendu, c'est le rôle du contrôleur de déclencher le commencement et la fin des différentes phases, et d'assurer la coordination entre les différents composants.

Les scénarios décrits ci-dessus, sont résumés par l'organigramme d'exécution suivant:

5.1 Exécution de l'agent mobile sur un hôte donné

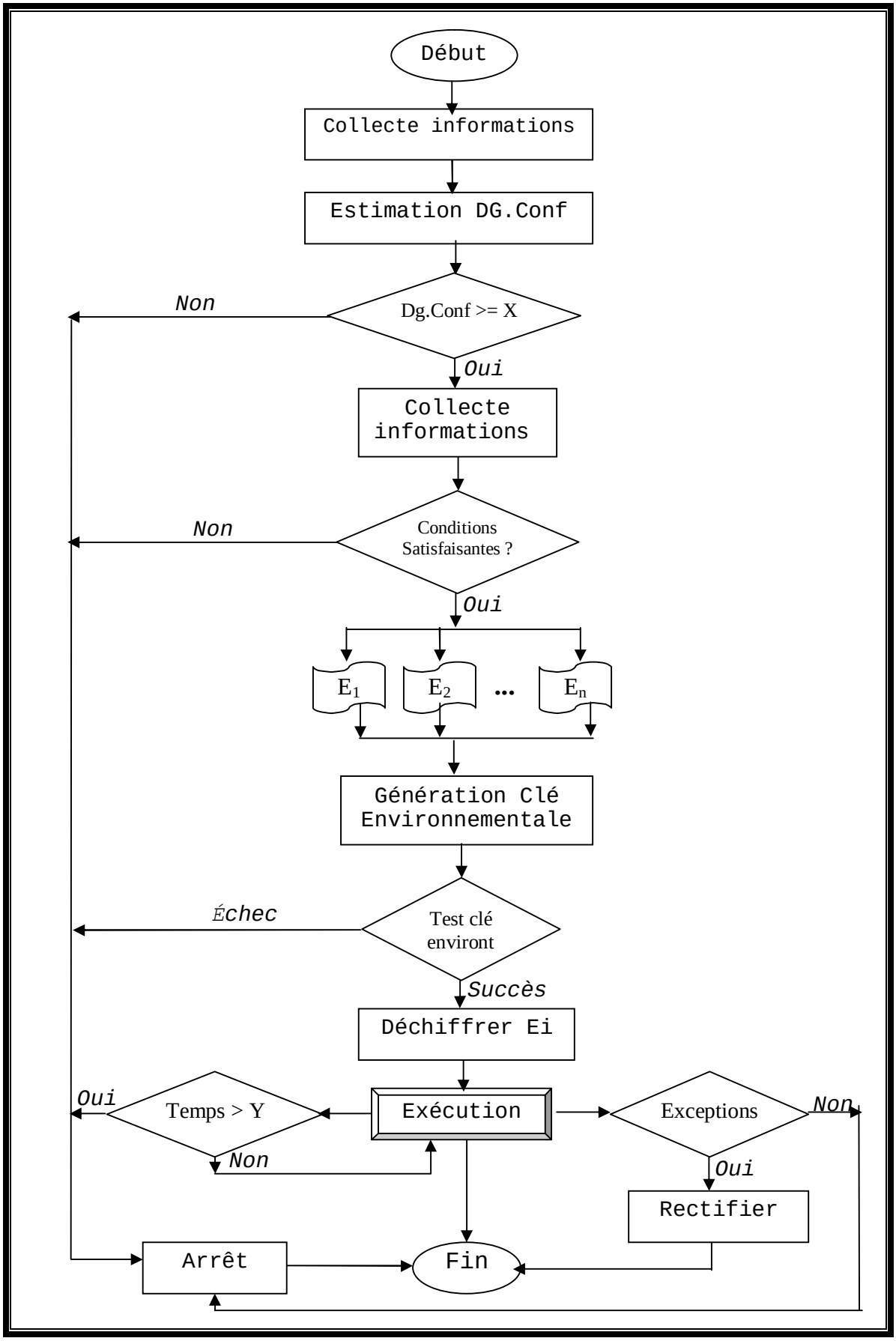


Fig 3.6- Organigramme de l'exécution

6. Estimation du degré de confiance

6.1 Notion de confiance

Toute étude de sécurité pose des questions de confiance et de crédibilité. *Castelfranchi* admet que la crédibilité ne peut pas et ne devrait pas être résumée à une probabilité subjective simplement estimée par l'acteur pour sa décision [13]. Une étude basée sur une observation prudente est nécessaire pour pouvoir établir une confiance dans un partenariat.

Plusieurs définitions ont été proposées pour la notion de *confiance*. Par exemple, Josang et al Ont proposé la suivante, s'adaptant bien à un environnement dynamique:

"trust is the extent to which one party is willing to depend on somebody, or something, in a given situation with a feeling of relative security, even though negative consequences are possible" [31].

6.2 Finalités et exigences de l'estimation de la confiance

Dans un système d'agents mobiles, pour que l'agent mobile puisse décider du traitement à effectuer, à chaque fois qu'il se déplace au sein du réseau, il doit tout d'abord estimer le degré de confiance vis-à-vis de l'hôte visité. L'estimation du degré de confiance se fait, bien entendu, en fonction des informations acquises à partir de l'environnement d'exécution correspondant à l'hôte en question.

Les informations selon lesquelles, le degré de confiance doit être bâti, devront être préalablement spécifiés. Elles doivent refléter l'honnêteté et la crédibilité des hôtes destinataires. La confiance pouvant être affectée aux plateformes de la part de l'agent mobile, dépend donc de plusieurs paramètres. Afin d'arriver à estimer une telle information, nous devons tout d'abord répondre aux questions suivantes [25].

- Comment est-ce que l'agent peut percevoir son environnement d'exécution, afin qu'il puisse émettre une bonne opinion.
- Comment est-ce que les diverses perceptions peuvent-elles estimer un degré de confiance de manière réaliste?
- Comment est-ce que cette information peut-elle déterminer exactement la catégorie de client?
- Comment peut on utiliser cette information, pour décider de l'action à exécuter?

Afin de pouvoir estimer une grandeur quantitative pour la confiance, nous devons tout d'abord donner des valeurs quantitatives pour ses paramètres. L'estimation du degré de confiance dépend des facteurs suivants [25]:

- Paramètres permettant d'identifier et d'authentifier l'hôte client.
- Paramètres rendant une transaction digne de confiance.
- Paramètres des ressources permettant d'assurer le bon fonctionnement des transactions.
- Paramètres portants sur la réputation des hôtes fournis par le propriétaire de l'agent ou une tierce partie de confiance. Cette information peut être bâtie en tenant compte d'un certain historique.

6.3 Processus de l'estimation du degré de confiance de l'hôte client

Bien entendu, L'agent mobile détient des tables contenant des renseignements relatifs aux différents hôtes qu'il va visiter (voir Tab 3.2), une comparaison entres les informations contenues dans ces tables avec celles collectées l'ors de l'exécution lui permet de s'assurer de leurs identités et les authentifier.

Afin de pouvoir quantifier la confiance, nous attribuons aussi, à chacun des paramètres (j) envisagés trois attributs; un poids (W), une importance (I) et un facteur Sj qui est égal à 1 dans le cas de succès (conformité de l'information), et 0 dans le cas d'un échec (non-conformité de l'information).

L'estimation du degré de confiance (T) s'effectue en fonction des paramètres et des facteurs décrits ci-dessus à l'aide de la formule suivante [25] :

$$T = \sum_{j=1}^{k} w_j \, I_j \, s_j$$

En fonction de l'importance et le poids donnés à un certain paramètre comparé à un autre, l'estimation du degré de la confiance diffère. Un paramètre est d'une importance très élevé si ce dernier servira de base pour prouver la crédibilité de l'hôte considéré. Par conséquent une erreur au niveau d'un tel paramètre affaiblira considérablement le degré de confiance dégagé (Exemple: identité, mot de passe,…), pendant qu'un paramètre moins important, peut être

recomposé par d'autres (Exemple: acronyme, e-mail,…). En revanche, deux paramètres d'une même importance peuvent avoir des poids différents (Exemple: le mot de passe a un poids plus grand que celui de l'identifiant d'une certaine entité).

Notons qu'Il est possible de retrouver la cause d'une certaine décision négative, en observant les facteurs dont le paramètre *Sj* vaut 0.

Nous considérons que les paramètres sont classés en trois groupes:

- Paramètres d'une importance très importante (3).
- Paramètres d'une importance moyenne (2).
- Paramètres d'une faible importance (1).

À chaque paramètre nous attribuons un poids appartenant à l'intervalle [1 – 20].

Facteur	Importance	Poids
Identifiant	3	15
Acronyme	2	10
Mot de passe	3	20
E_mail	1	5

Tab 3.2- Exemple des valeurs des paramètres de quelques facteurs.

Afin de pouvoir décider de la réaction adéquate pour une situation donnée, nous cernons quelques limites pour les différentes valeurs du degré de confiance estimé (voir Tab 3.5). Si ce dernier fait partie d'un intervalle jugé très satisfaisant (exemple, [71-100]), nous pouvons dire que l'hôte est digne de confiance, et par conséquent, notre agent mobile devra assurer un service complet. Sinon, si le degré de confiance obtenu appartient à un autre intervalle dit *moyen* (exemple [41-60], l'agent mobile assure un certain service *dégradé*. En revanche, si le degré de confiance estimé est considéré *inacceptable* (soit inférieur à un certain seuil; Exemple 40), l'agent mobile devra s'arrêter et quitter l'hôte en question.

Intervalle de l'estimation	Réaction
[0 – 40]	Arrêter
[41 - 61]	Service dégradé
[71 - 100]	Service complet

Tab 3.3- Exemple d'intervalles d'estimation avec les réactions correspondantes.

7. Génération de la clé environnementale.

L'agent mobile transporte des informations critique nécessitant ainsi d'être protégées soigneusement, Dans notre cas, les expressions abstraites déterminants la séquence des micro-composants du code à exécuter, constitue des informations cruciales et secrètes devrant être tenu confidentielles. De ce fait, cette partie de l'agent mobile, ne doit pas être sauvegardée et transmise sous sa forme intelligible. La meilleure façon pour garder leur confidentialité, est de les chiffrer à l'aide d'un algorithme de cryptage efficace.

La question qui se pose, concerne le choix de cet algorithme de cryptage et comment maintenir la clé de décryptage. En fait, il se trouve qu'il n'est pas prudent de faire circuler une telle information aussi critique et secrète (la clé), au sein du réseau. Il est question donc de trouver un moyen pour assurer la confidentialité de cette clé.

Dans le cadre de notre stratégie, nous allons éviter le transfert de cette clé éliminant ainsi des risques de sa divulgation. Nous proposons plus tôt sa génération par l'agent mobile lors de l'exécution, en arrivant sur le site client. De plus, et pour encore plus de prudence, cette clé ne sera générée que si les conditions de bon fonctionnement sont vérifiées, afin d'empêcher que cette clé soit générée et non utilisé, diminuant ainsi la probabilité de ses utilisations malveillantes.

Étant donné l'ensembles E= {E1, E2, …En} des expressions abstraites, faisant appel aux différents micro-composants A = {A1, A2, …,Ap}, une clé environnementale Kj devrait être généré pour chaque expression abstraite Ej.

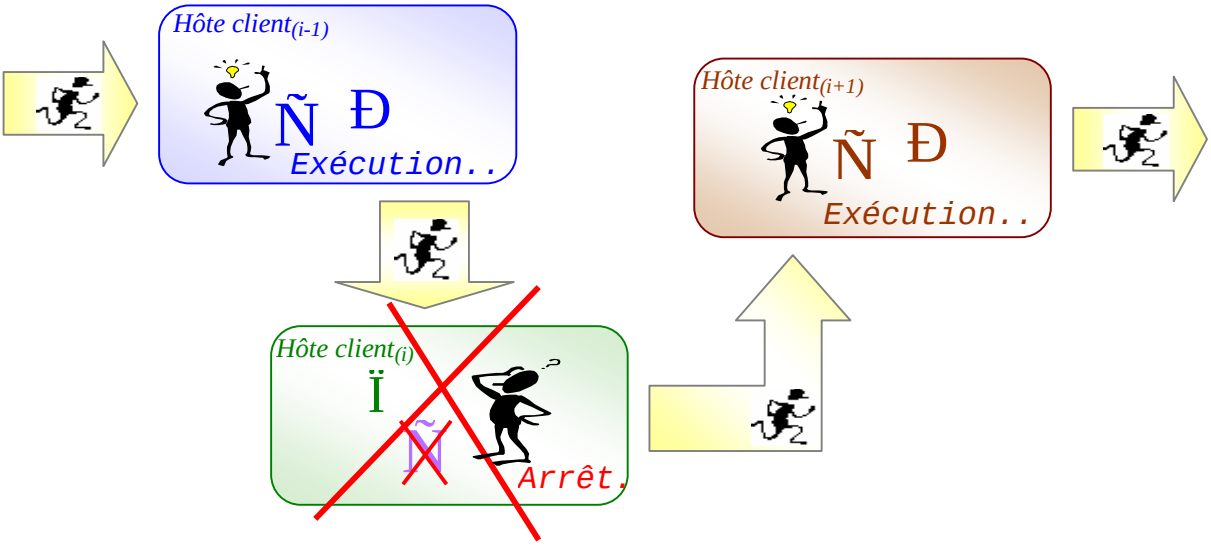


Fig 3.7- Nécessité de la clé environnementale pour l'exécution

7.1 Algorithme de la génération de la clé environnementale

La génération de cette clé dépend comme nous l'avons déjà mentionné des informations environnementales ainsi que, de l'identité de l'agent mobile qui est unique. Nous utilisons aussi une fonction de hachage.

La génération de la clé environnementale se fait selon l'algorithme suivant inspiré de [25] :

Algorithme: Génération de la clé environnementale

1. Collecter les données, soit $D=\{d_1, d2, \ldots, d_k\}$ l'ensemble des ces données.
2. Concaténer les données collectées, soit $C=(d_1\ d_2 \ldots\ d_k)$.
3. Appliquer la fonction de hachage *SHS* (Secure Hach Standard) sur le résultat de la concaténation.
4. Appliquer $C \oplus id$ (avec id est l'identifiant unique de l'agent mobile).

Fig 3.8- Génération de la clé environnementale [26]

Notons que :

- Une clé environnementale bien spécifique pour chaque expression abstraite, permet de renforcer la protection de cette information aussi critique. Ainsi, même si une plateforme malveillante arrive à détenir une clé venant d'être générée, elle ne lui servira à rien du moment où dans un autre environnement, une nouvelle clé devra être engendrée.
- Le choix des paramètres: $\{d_1, d2, \ldots, d_k\}$, selon lesquels la clé environnementale devra être générée, doit effectuer de telle sorte que cette clé reflète bien le contexte et la situation de l'environnement. Le degré de confiance estimé peut être pris en compte lors de cette opération.

CONCLUSION

Nous avons proposé une approche permettant de protéger un agent mobile contre les éventuelles attaques des hôtes malveillants de type espionnage de comportement. Nous rappelons que ces attaques se produisent lorsque l'accès aux informations secrètes de l'agent mobile devient impossible ou bien à la rigueur très difficile, en essayant d'observer et d'analyser son comportement afin de déduire sa stratégie.

Nous avons fait appel aux techniques de l'adaptation dynamique pour faire face à ce type d'attaques. Ces techniques permettent à l'agent mobile de s'adapter et diversifier ses traitements au cours de son cycle de vie en tenant compte des variations de l'environnement. Il se caractérise ainsi par un comportement changeant et imprévisible et donc transparent vis-à-vis de l'hôte récepteur. Ceci rend toute tentative d'analyse difficile et même inutile.

En outre, l'utilisation des techniques de sécurisation classique telles que la cryptographie et signature numérique, permet de préserver en plus de la confidentialité, l'intégrité de l'agent et renforcer le niveau de sécurité proposé. Ainsi, notre approche répond indirectement à d'autres types d'attaques.

L'architecture que nous avons proposée met en évidence une stratégie générique, indépendante de toute application et tout domaine d'applications, offrant ainsi l'avantage de son extensibilité et sa réutilisabilité. Nous avons adopté la réflexivité comme structure de modélisation. Cette dernière que l'on pourrait qualifier de bon support du fait qu'elle fournit une bonne représentation et une manipulation aisée de soi, en fournissant des mécanismes permettant d'accéder à certains détails d'implémentation du système et d'exprimer les traitements en termes extrêmement génériques. Ce qui rend l'implantation de l'adaptation une tâche faisable et évidente.

Chapitre IV

ETUDE DE CAS & IMPLEMENTATION: APPLICATION AU DOMAINE DU E-COMMERCE

«J'écoute et j'oublie, je vois et je me souviens,
je fais et je comprends »
Proverbe chinois

Le présent chapitre, présente une étude de cas réalisée dans le domaine du commerce électronique. Ce dernier qui a bouleversé le monde avec les facilités et les conjonctures qu'il offre, souffre encore du problème de la sécurité. Le but de ce chapitre, est de clarifier encore les idées mises en évidence par l'approche que nous avons étalée au niveau du chapitre précèdent. Nous essayons de les concrétiser en les appliquant sur le domaine que nous venons de mentionner et qui souffre notamment du problème que nous avons opté de traité. Aussi nous décrivons l'implémentation de notre application réalisée à l'aide de la plateforme JADE, version 3.4.

INTRODUCTION

Après une représentation théorique de notre approche basée-adaptabilité dynamique, pour la sécurisation d'un agent mobile, -contre les attaques de type analyse de comportement- au sein d'un environnement changeant et variant. Dans le but d'une meilleure compréhension, nous appliquons notre approche sur un domaine bien précis.

Nous avons choisi pour ce faire, le domaine du commerce électronique, l'une des utilisations les plus révolutionnaires d'Internet, où des entreprises présentent sous forme de fenêtres ouvertes, des sites effectuant des transactions de vente et d'achat sur le Web, permettant de proposer des services via des vitrines numériques qui ne ferment jamais, et de commander n'importe quel article en un simple clic sans sortir de chez soi.

Il s'agit d'une simple application partielle du e-commerce ayant recours à des partenaires externes au cours de son expérimentation. Le but est de dérouler les principaux aspects de notre approche sur un exemple concret afin de montrer la faisabilité des idées mises en évidence. En plus, et en vue de pouvoir constater certains aspects techniques de l'approche, nous avons choisi de l'implémenter en utilisant la plateforme JADE, un outil assez répandu pour le développement et la manipulation des systèmes d'agents mobiles. Nous utilisons également le langage JAVA pour la programmation des composants de notre agent mobile.

Pour ce faire, nous procédons comme suit. Nous commençons par limiter le cadre de notre application dans le domaine du e-commerce, en justifiant l'utilisation des agents mobiles. Nous décrivons ensuite notre application de manière générale via un organigramme mettant en évidence les différentes activités rentrant en jeu. Puis nous essayons de la détailler selon le modèle présenté dans le chapitreIII. Par la suite, nous décrivons brièvement la plateforme que nous avons adoptée pour l'implémentation de notre application, pour montrer par la suite comment nous l'exploitons dans le cadre de notre travail. Nous fournissons au fur et à mesure des exemples d'interfaces et de codes de manipulation de cette dernière.

1. Présentation du domaine d'application

« Allumez votre ordinateur, connectez vous à Internet et le monde entier est chez vous … »

Le commerce électronique –ou économie numérique, marché électronique ou encore commerce par l'Internet– désigne un système économique dans lequel les entreprises et les consommateurs se servent du réseau Internet pour générer un marché totalement ouvert.

Tout simplement c'est l'achat et vente (transactions commerciales) des biens et des services effectués en ligne. Une application de commerce électronique nécessite la prise en compte des commandes en ligne des clients, la gestion des bases de données des clients et des entreprises, et notamment fournir un moyen de paiement en ligne sécurisé. [52]

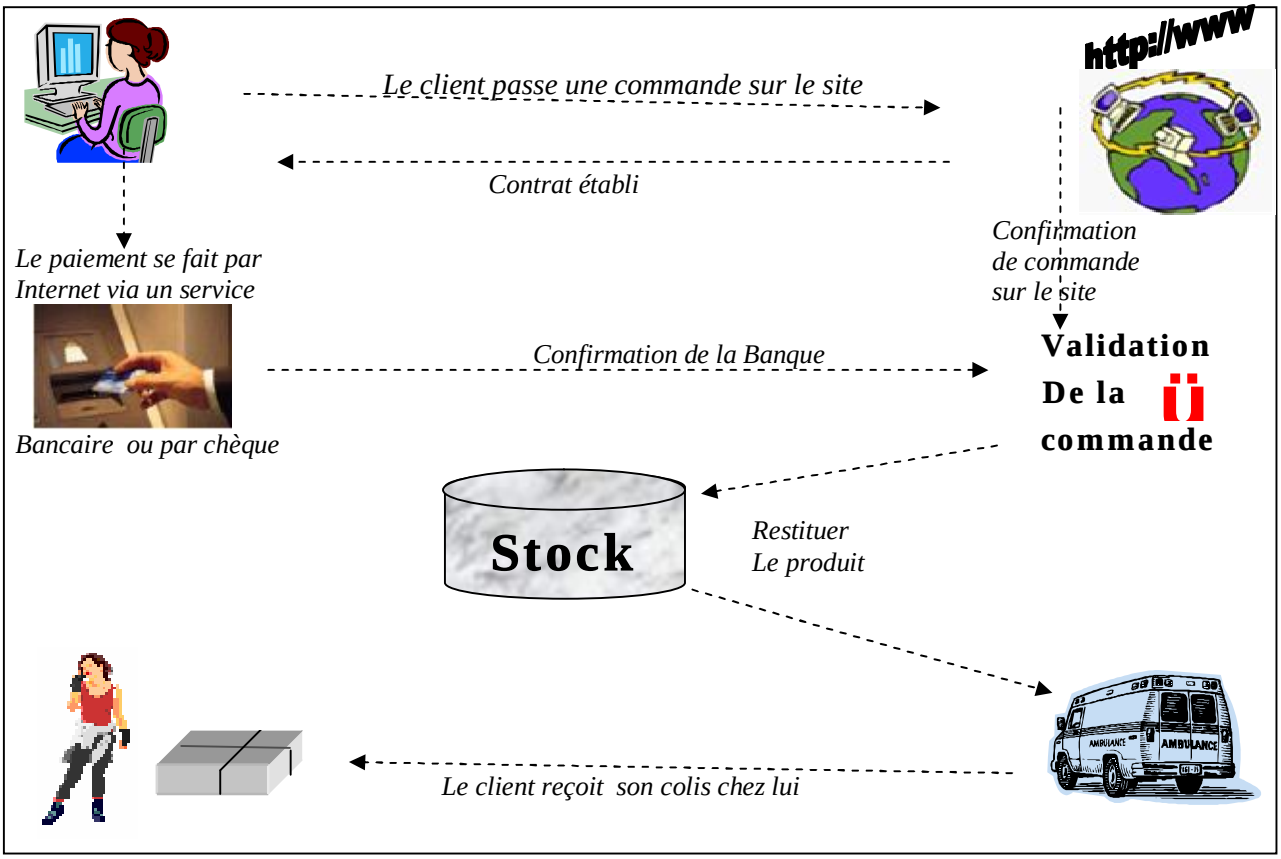


Fig 4.1- Le processus de vente et d'achat en ligne

Le commerce électronique classique ne fait que permettre à l'utilisateur d'accéder à la boutique en ligne des produits offerts par l'entreprise, à partir desquels il choisit les articles qui l'intéressent, les valider après avoir réglé son payement et attendre jusqu'à ce que sa commande soit arrivée chez lui. Toutefois, tout le savoir faire doit être fait par le client lui même (Effectuer son choix; comparer les articles, leurs prix, la qualité, leurs caractéristiques, ….).

Utilisation des agents mobiles et problème de sécurité

Avec la grande variété de produits et services proposés en vente, il n'est pas toujours facile d'effectuer le meilleur choix. Par conséquent, un client souhaitant faire une telle transaction peu épuiser plusieurs heures pour trouver ou ne pas trouver ce qu'il cherche.

On a besoin d'un côté d'un système qui prend des initiatives en satisfaisant les besoins de l'entreprise lui permettant d'aller vers les clients et leur présenter les articles et les services les plus adaptées selon leurs profils. D'un autre côté on a besoin d'un système qui satisfait les besoins des clients en cherchant les produits qui les intéressent, et qui est capable de négocier avec les différents marchands d'une manière autonome.

Les agents mobiles rendent les interfaces web plus conviviales et interactives et permettent de simplifier les processus d'achat et de vente, de renforcer les liens entre les internautes et les marchands, d'améliorer la gestion de la relation client et de la personnaliser [50]. Les fournisseurs des services ainsi que les clients peuvent déléguer aux agents mobiles la responsabilité de les représenter, chercher les meilleures offres, négocier, effectuer des choix, et revenir avec les résultats. Toutefois, ces agents peuvent visiter des bons et des mauvais sites, et confrontent par conséquent une multitude d'attaques et de risque -comme nous l'avons montré au niveau du premier chapitre-. Cela pose une sérieuse demande de sécurité. Nous avons proposé au niveau du chapitre précèdent une approche protectrice de l'agent mobile. Et nous allons l'appliquer dans le présent chapitre sur un exemple de e-commerce, qui souffre notamment du problème évoqué.

2. Délimitation du cadre de l'application

Cette étude de cas vise à concrétiser les différents concepts rencontrés dans le chapitre précédent. Pour cela, un exemple d'application de commerce électronique simple a été choisi. L'application en question ne prend en considération que la partie d'exploration des sites destinataires, l'étude des offres et des conditions de travail rencontrés et l'établissement du contrat approprié dans les cas favorables. Ces tâches, bien entendu, devront être accomplies par un agent logiciel mobile.

Nous admettons que notre agent mobile présente un point de vente des produits informatique (matériels & logiciels). D'un côte, Il doit consulter certaines entreprises en vu d'effectuer des commandes d'achat des biens (en gros) et d'autres en se contentant de voir ce qu'ils proposent comme produits et privilèges. De l'autre côté, il doit visiter quelques clients, afin de leur proposer la vente de quelques produits (en détails).

Notre agent mobile commence par identifier et authentifier ses partenaires afin de pouvoir déterminer les services qu'il doit assurer, et spécifier les informations devrant être collectées. Par la suite il expose ses services aux différents partenaires, et essaye d'accomplir ses traitements en tenant compte des conditions de travail rencontrées et de sa stratégie de sécurisation adoptée.

3. Description générale de l'application

Etant donnée que la stratégie de protection que nous proposons pour notre agent mobile, se base sur la notion d'adaptabilité dynamique, s'agissant du fait que notre agent mobile doit se comporter différemment à chaque fois qu'il se déplace à travers le réseau, en assurant des traitement variés, nous envisageons les cas de figures suivantes:

(i) L'agent mobile assure trois tâches principales:

- *Achat*: si les conditions d'achat exigées par le propriétaire sont satisfaites, et que l'offre en question est jugée intéressante de la part de l'agent mobile, suivant les informations qu'il détient (exemple: prix est strictement inférieur à un certain seuil, marque faisant partie de quelques unes mentionnés de bonne qualité, éventuellement; quelques privilèges sont offerts tel que garantie après vente,…), L'agent mobile opte pour un achat immédiat.
- *Vente:* compte tenu des possibilités offertes par le propriétaire de l'agent mobile et les exigences qu'il impose, si le client en question décide d'acheter une certaine quantité des produit proposés par l'agent mobile, ce dernier optera pour l'établissement du contrat de la vente considéré.
- *Recherche d'une meilleure offre:* L'agent mobile se contente d'observer et de comparer les opportunités offertes de la part des quelques entreprises (Prix, quantité, qualité, délais, garantie, service après vente, …), et rapporte un compte rendu à son propriétaire, afin d'effectuer une éventuelle planification.

(ii) Un ensemble de dégradations de service est envisagé, par exemple:

- *Cas d'achat:* l'agent mobile opte pour acheter toute la quantité requise si l'affaire est jugée très satisfaisante, selon des critères d'évaluation prédefinis, comme il peut se contenter d'acheter une quantité restreinte dans le cas d'une affaire peu intéressante, avec d'éventuelles degrés de dégradation de service selon le cas, ou encore ne pas acheter du tout.
- *Cas de vente*: l'agent mobile peut vendre un produit permanent ou limité par le temps selon le cas en question.

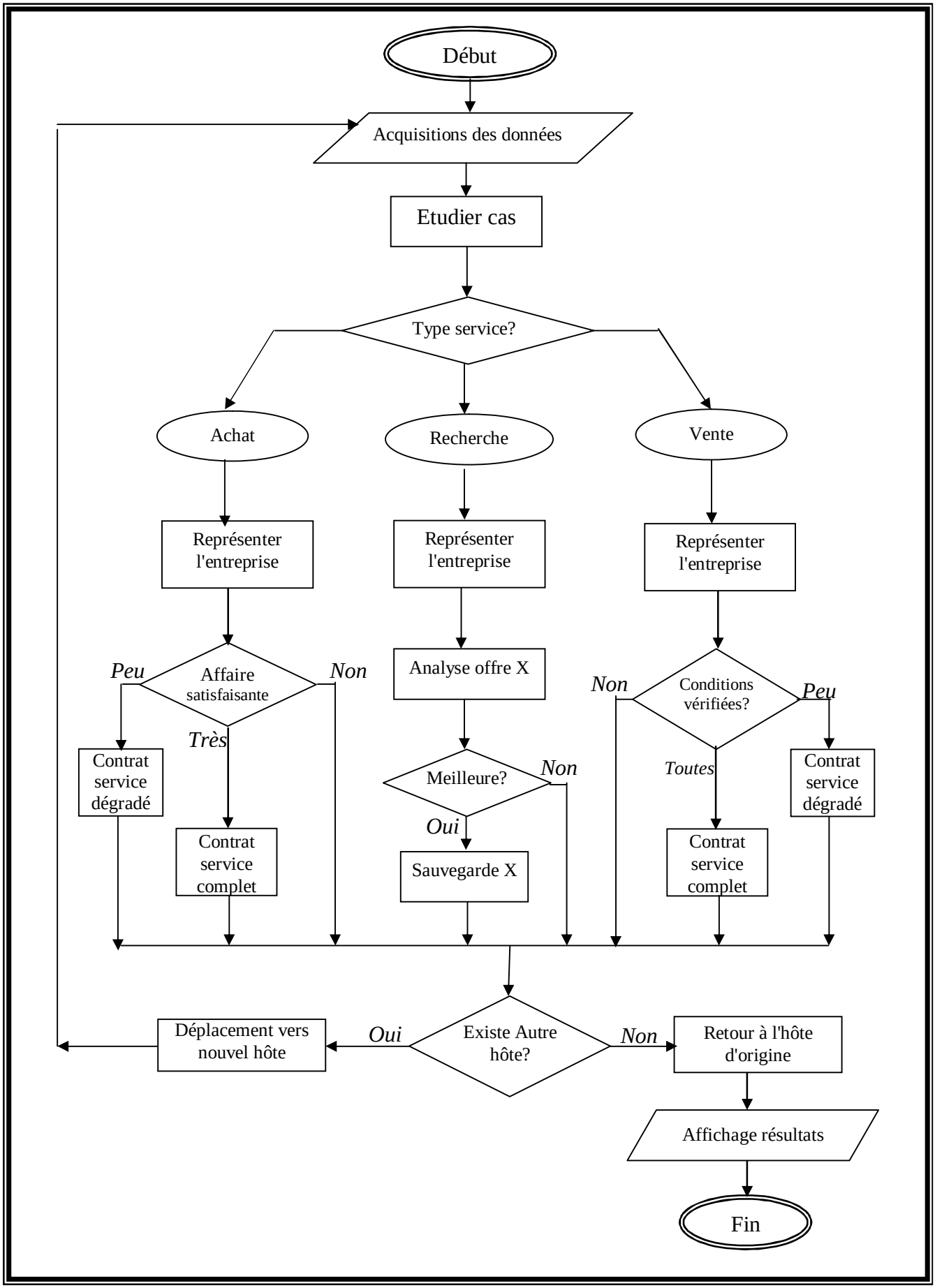


Fig 4.2- Organigramme des tâches de l'agent mobile

(iii) Nous pouvons envisager également un ensemble d'alternatives des traitements à effectuer par l'agent mobile si les conditions de travail rencontrées se ressemblent. Ces dernières peuvent se manifester sur les apparences de l'agent mobile (ses interfaces), aussi bien que sur le contenu; L'agent mobile peut par exemple proposer à chaque fois des offres différentes, portant sur des collections de produits différents ou bien jouer sur les exigences imposées selon le partenaire à qui il a affaire.

4. Description détaillée de l'application

4.1 L'interface

Nous rappelons que ce composant permet à l'agent mobile de communiquer avec les hôtes destinataires représentant son environnement de travail. L'interaction avec l'environnement extérieur de l'agent mobile se traduit, comme nous l'avons déjà dit au niveau du chapitre précédant, en l'acquisition des informations environnementales reflétant l'état de cet environnement, et l'agissement sur cet environnement. Ces deux opérations sont réalises grâce aux deux sous composants de l'interface, qui sont le capteur et l'actionneur.

4.1.1 Le capteur

C'est le sous composant responsable de l'acquisition des donnés. Nous avons envisagé dans le chapitre précédant trois types d'acquisition des informations, qui sont: l'observation, l'inspection et l'interaction. Les deux premières formes d'acquisition nécessitent des fonctionnalités compliquées pour l'agent mobile, à savoir des outils lui permettant d'accéder aux ressources des plateformes visitées, des mécanismes lui permettant d'observer son environnement de travail, etc. Pour des raisons de simplicité, nous nous contentons de la troisième forme d'acquisition.

L'interaction va être résumée en des questions/réponses établis entre l'agent mobile et l'hôte de réception. Ces questions/réponses seront établies par l'interface de l'agent mobile. Elles précisent les conditions de travail et les exigences de chacun d'eux, ce qui leur permettra d'aboutir à un accord. Le capteur donc, se contente d'acquérir les informations demandées et les sauvegarder dans les tables appropriées

au niveau de la mémoire, afin de les utiliser par la suite dans l'estimation du degré de confiance, la génération de la clé environnementale, etc.

Algorithme Capteur

```
Début
            /*Déclarations */
   Par_cnf: table des paramètres du confiance;
   Par_clé: table des paramètres de la clé;
   Nb: nombre des paramètres de confiance;
   Nbre: nombre des paramètres de clé environnementale;
   i: indice;

    /* acquisition des informations environnementales */

   Pour i de 1 jusqu'à Nb faire
        Lire Par_cnf[i];
   Fin pour

   Pour i de 1 jusqu'à Nbre faire
        Lire Par_clé[i];
   Fin pour

Fin
```

Fig 4.3- L'algorithme du composant « Capteur »

4.1.2 L'actionneur

De même, l'actionneur prend différentes formes suivant le type de l'agent mobile et la mission qui lui est affectée. Comme l'interaction entre l'agent mobile et son environnement de travail dans notre cas se résume en un ensemble de questions/réponses de l'agent mobile, l'actionneur se contentera d'afficher les messages que l'agent mobile désire transmettre à son partenaire d'une façon compréhensible.

Algorithme actionneur

```
Début
     /* affichage des résultats */
Fin
```

Fig 4.4- L'algorithme du composant « Actionneur »

4.2 La mémoire

Comme nous l'avons mentionné également dans le chapitre précédant, ce sont les informations collectées par l'agent mobile (informations environnementales) qui vont lui permettre d'identifier et d'authentifier les hôtes visités, ainsi que de spécifier le type de service à accomplir et préciser les traitements à effectuer. Exemple: identifiant de l'hôte visité, acronyme, intitulé, mot de passe, service demandé, propriétés du service (quantité, qualité, délais, Durée de temps, …)

Bien entendu, l'agent mobile détient un ensemble de données fournies par son propriétaire (informations d'origine), lui permettent de vérifier la véracité des informations environnementales, et de pouvoir effectuer ses choix.

4.2.1 Exemples d'informations d'origine

Nous donnons ici quelques exemples de ces informations concernant les contractants.

	identifiant	**acronyme**	**Type de produits**	**Mot de passe**	**@ IP**
Entreprise 1	F00991	SUN	Logiciels	Ax53frg63o	193102151230
Entreprise 2	F00992	DELL	Ord complets	Matfold521	192097132019
Entreprise 3	F00993	SISCO	Outils réseaux	Sistel62141	193202510122
Entreprise 4	F00994	DISCOVRY	Matériels	Dismal52ok	181852963258
Entreprise 5	F00995	VERBATIM	Unité de disques	Infdisc152ss	191255526845
…	…	…	…	…	…

Tab 4.1- Table des entreprises

	identifiant	**acronyme**	**Mot de passe**	**@ IP**
Client 1	C00501	BBX	Alima258	172586214526
Client 2	C00502	AWT	Chiko258	172698559874
Client 3	C00503	ACCL	Accl25fr	162584662255
Client 4	C00504	XXM	Xyz52com	122556668551
Client 5	C00505	NINI	25nini1258	225223115451
…	…	…	…	…

Tab 4.2- Table des clients

L'agent mobile détient d'autres informations sur les produits que son propriétaire expose à ses clients:

	Identifiant	**Désignation**	**Marque**	**Prix unit Achat**	**Prix unit Vente**
Produit 1	P100122	PC complet	IBM	320.00€	410.00€
Produit 2	P100123	PC complet	DELL	350.00€	450.50€
Produit 3	P200110	PC mobile	HP	1120.50€	1200.30€
Produit 4	P200110	PC mobile	DELL	1020.00€	1100.00€
Produit 5	P300330	CPU	Intel	80.00€	100.00€
Produit 6	P300331	CPU	Motorola	65.50€	80.50€
Produit 7	P400501	Disque amovible	Verbatim	35.00€	40.50€
Produit 8	P400502	Disque amovible	Sonny	25.50€	30.50€
…	…	…	…	…	…

Tab 4.3- Tables des produits matériels

	Identifiant	**Intitulé**	**Marque**	**Prix unit Achat**	**Prix unit Vente**
Produit 80	P811661	Système Exploitation	Unix	80.50€	90.50€
Produit 81	P811662	Anti-virus	Norton	50.00€	60.00€
Produit 82	P811663	Anti-virus	Avast	40.00€	45.50€
Produit 83	P811664	Anti-virus	AVG	35.50€	50.00€
Produit 84	P911665	Traducteur	Systran	36.20€	41.50€
…	**…**	**…**	**…**	**…**	**…**

Tab 4.4- Tables des produits logiciels

Des informations détaillées sur les produits peuvent êtres données:

	RAM	**Cache**	**Disque dure**	**Fréquence H**
P100122	512 Mo	512 Mo	80 Go	3.6 GH
P100123	01 Go	01 Go	80 Go	3.2 GH
P200110	512 Mo	01 Go	80 G0	4.2 GH
P200110	1024 Mo	02 Go	80 Go	3.8 GH
…	…	…	…	…

Tab 4.5- Table des propriétés des PC

Des informations concernant les intervalles des degrés de confiance

Intervalle de confiance	Réaction
[0 – 30]	Arrêter
[31 - 61]	Service dégradé
[61 - 100]	Service complet

Tab 4.6- Tables des intervalles de confiances

Facteur	Importance	Poids
Identifiant	3	15
Acronyme	2	10
Mot de passe	3	20

Tab 4.7- Table des propriétés des facteurs du degré de confiance

Notons ici, que les informations prises en compte lors du calcul du degré de confiance, sont celles ayant trait à l'authentification. Exemples: identité, acronyme, mot de passe, etc. Tandisque les données qui participent à la génération de la clé environnementale sont celles renfermant les conditions de travail. Exemples: l'identité de l'hôte en question, permet à l'agent mobile de décider du type de service à réaliser (achat, vente ou recherche). En plus, quelque détails peuvent permettre de préciser les actions à entreprendre par l'agent mobile, tels que les privilèges proposées dans le cas d'un achat (exemples: garantie, service après vente, système des cadeaux,…), ou encore les préférences des clients dans le cas d'une vente (qualité, quantité, prix, etc).

4.3 Bibliothèques des micro-composants

Elle contient les modules du code de l'agent mobile, appelés aussi les micro-composants. Nous avons distingué deux classes de ces micro-composants:

(i) La première classe, regroupe les portions du code de l'application proprement dite, que nous avons appelé les modules fonctionnels. Dans notre cas, cette classe des micro-composants implémente les différents services d'achat, de vente et de recherche de la meilleure offre, ainsi que leurs dégradations et alternatives envisagées.

(ii) La deuxième classe de micro-composants permet de renforcer le niveau de sécurité proposé. Nous les avons appelés *'modules sécuritaires'*. Il s'agit des méthodes de sécurisation classique, telles que, la cryptographie, le hachage et la signature numérique. A ce stade nous utiliserons les solutions offertes par l'environnement d'implémentation adopté.

4.4 L'adaptateur

Ce composant permet d'adapter le comportement de l'agent mobile par la collaboration des trois composants: L'analyseur, Le déliberateur et le contrôleur. Ces derniers se servent de la base des règles d'adaptation pour accomplir leurs tâches.

4.4.1 L'analyseur

Ce composant se charge de vérifier la validité des informations données par l'hôte visité, en les comparant avec celles qu'il détient, telles que l'identité et le mot de passe...

Ces dernières lui permettent d'identifier et d'authentifier l'hôte et de déterminer la nature de service à entreprendre (achat, vente ou recherche). Elles lui serviront également à estimer le degré de confiance et par conséquent, effectuer un premier filtrage des comportements à entreprendre en se basant sur la base des règles à savoir; s'arrêter, assurer un service complet ou bien assurer un service dégradé (Voir : Fig 4.5).

4.4.2 Déliberateur

Ce composant est responsable de la génération de la clé environnementale lui permettant d'effectuer une délibération précise des traitements à accomplir. Cela est réalisé en fonction des conditions de l'environnement de travail exprimées par la clé dite – environnementale- (voir : Fig 4.6).

Par exemple, si un produit proposé pour vente est garanti pendant une certaine période (deux/quatre/six mois par exemple), ou bien si l'achat d'une certaine quantité permet d'avoir certains bonus, l'agent ajuste la quantité du produit à acheter (100%, 50%, 25%,…). Dans le cas d'une vente, l'agent mobile peut décider du délai d'expiration du produit vendu (exemple 2 mois, 4 mois, 5 mois….).

Ce module réalise aussi le déchiffrement des expressions abstraites dénotant l'ensemble des traitements à effectuer, en appelant la fonction appropriée à partir de la bibliothèque des micro-omposants (modules sécuritaire).

4.4.3 Contrôleur

C'est le composant responsable de la coordination et la synchronisation de tous les composants de l'agent mobile. Il est responsable également de la vérification de l'ensemble des exceptions en cas de problèmes, ainsi que du contrôle du temps d'exécution des tâches. Si par exemple, l'hôte de réception ne répond pas au bout d'une certaine période (60 secondes par exemple) aux questions de l'agent mobile, il arrête immédiatement le traitement pour se déplacer vers l'hôte suivant (voir : Fig 4.7).

Nous proposons là un algorithme pour chacun de ces composants.

Algorithme Analyseur

```
Début
                    /* Déclarations */
   Par_cnf: table des paramètres environnementaux;
   Par_org: table des paramètres d'origine;
   T_S: table de conformité des paramètres;
   T_W: table des poids des paramètres;
   T_I: table des importances des paramètres;
   Trust: degré de confiance;
   Nb: nombre des paramètres;
   j: indice;

         /* Vérification des informations */
   Pour j de 1 jusqu'à nb faire
      Si (Par_cnf[j] = par_org[j]) alors
          T_S[j] := 1;
      Sinon
          T_S[j] := 0;
      Fin si
   Fin pour

          /* Estimation du degré de confiance */
   Trust :=0 ;
   Pour j de 1 Nb faire
    Trust := Trust + T_w[j]*T_I[j]*T_S[j];
   Fin pour

          /* Vérification de la base des règles */
   Si (60 <= Trust <=100) alors
     Retourner (E1)  /* Service complet*/
   Sinon
      Si (30 < Trust <=60) alors
          Retourner (E2)  /* service dégradé*/
      Sinon
          Retourner (E3) /*Arrêter + Noter + Quiter*/
      Fin si
   Fin si
Fin
```

Fig 4.5- L'algorithme du composant « Analyseur »

Algorithme Délibirateur

```
Début
        /* Déclarations */
    Id: identificateur unique de l'agent mobile;
    Key: La clé environnementale;
    Par_clé: table des paramètres de la clé;
    Nbre: nombre des paramètres de la clé;
    Ej: expression abstraite donnée par l'analyseur;
    Rslt: Résultat de décryptage;
    i: indice;

        /* génération de la clé environnementale */
    Key := (" "),
    Pour i de 1 jusqu'à Nbre faire
        Key := Concaténation (Key, Par_clé[i]);
    Fin pour
    Appliquer Hachage (Key);
    Key := Concaténation (Key, Id);

        /* Application de la fonction de décryptage*/
    Rslt:= décryptage (Ej,Key);

        /* Teste de la clé environnementale*/
    Si (décryptage réussi) alors
        Retourner (Rslt);
    Sinon
        Message d'erreur; /*demande d'arrêter et de quitter*/
    Fin si

Fin
```

Fig 4.6- L'algorithme du composant « Déliberateur »

Algorithme Contrôleur

```
Début
   /* déclarations */

   /* acquisition des information environnementales */
   Activer Capteur ();

   /* Analyse des données */
   Activer analyseur ();

   /* Délibération des résultats */
   Activer Délibérateur();

   /* exécution des traitement délibérées */
   Activer Actionneur ();

   /* Exceptions */
   Si  (Temps consommé >= Temps estimé) alors
        Vérifier ensemble exceptions ;
        Si (Ok Exceptions) alors
             Rectifier;
        Sinon
             Arrêter; Noter;
        Fin si

   Fin si

  /* Transfère */
   Si (Existe autre hôte) Alors
        Migrer vers nouvel hôte;
   Sinon
        Revenir à l'hôte d'origine;
   Fin si

Fin
```

Fig 4.7- L'algorithme du composant « Contrôleur »

5. Environnement de développement

5.1. Choix de la plateforme d'agent mobile

Il existe à l'heure actuelle une panoplie de plateforme d'agents, facilitant le développement et la manipulation de ces systèmes. Le choix d'une plateforme d'agent dépend fortement de l'application en question et de l'objectif recherché.

Dans le cadre de notre travail, la plate forme devra répondre aux contraintes suivantes:

- Puisque notre agent est mobile, la plateforme choisi doit fournir les supports et les facilités nécessaires pour cette fonctionnalité.
- Notre agent mobile est adaptatif, d'où la nécessité de choisir une plateforme lui permettant de réaliser ses traitement sans difficultés.
- Le langage de programmation sous-jacent et la nature des messages échangés sont aussi un point important. Un langage assez répandu sera plus utile.
- De plus, une plateforme riche en terme de documentation est plus facile à exploiter.

En se basant sur l'étude effectuée sur les plateformes au niveau du chapitre I, notre choix est porté sur la plateforme JADE se rapprochant mieux de nos critères.

5.2 Description générale de la plateforme JADE.

La plateforme JADE (Java Agent DEvelopment Framework) [6] est un environnement de développement d'agents implanté totalement dans le langage JAVA développée par la société CSELT (Italie). La plate forme JADE est fondée sur la spécification de FIPA (Foundation for Intelligent Physical Agent). JADE possède trois modules principaux (nécessaire aux normes FIPA). Ces derniers sont activés à chaque démarrage de la plateforme.

- DF «*Director Facilitor*»: fournit le service de « *pages jaunes*» à la plate-forme.
- ACC «*Agent Communication Channel* » gère la communication entre les agents.
- AMS « *Agent Management System* » supervise l'enregistrement des agents, leur authentification, leur accès et l'utilisation du système.

L'interface graphique de JADE (GUI) assure un traitement plus commode de la plateforme. Elle permet à l'utilisateur d'exécuter plusieurs ordres tel que créez un nouvel agent dans la

même plate-forme, cloner l'agent, le déplacer, le suspendre, le tuer …etc. L'interface elle-même a été implémentée comme un agent, appelé **RMA** « `Remote Monitoring Agent` ». Un RMA peut être lancé à partir de la ligne de commande :

« `Java jade.Boot myconsole :jade.tools.rma.rma` »

Ou bien avec l'option *–gui : « `java jade.Boot -gui` »*

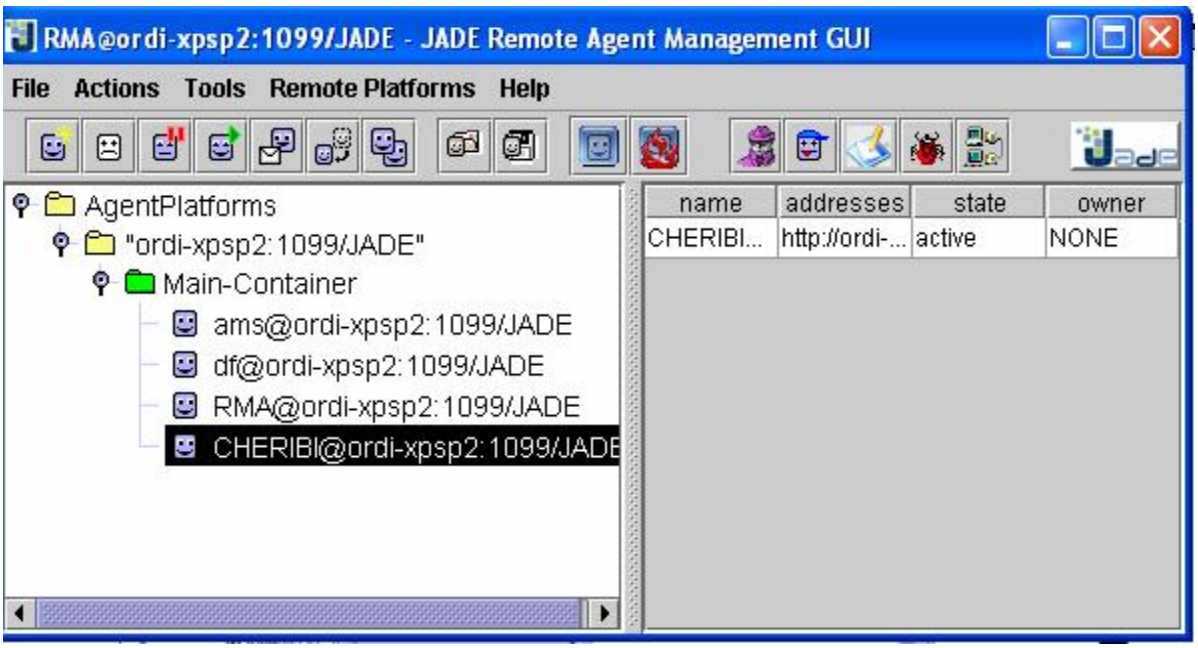


Fig 4.8- Interfaces graphiques de la plateforme JADE

La plate forme peut être répartie sur plusieurs hôtes. Une seule application JAVA et par conséquent une seule machine virtuelle Java est exécutée sur chaque hôte. Chaque machine virtuelle Java est un conteneur d'agents fournissant un environnement complet d'exécution de ces agents et permettant à plusieurs agents concurrents de s'exécuter sur le même hôte.

6. Mapping des concepts de notre application dans JADE.

Nous allons dans ce qui suit, montrer comment nous implémentons notre application dans le cadre de la plateforme JADE, en se servant des concepts et des outils de contrôles proposés par cette dernière.

6.1 Création de l'agent Jade

Pour la création de l'agent mobile, nous définissons une classe qui hérite de la classe «Jade.core.agent», puis nous implémentons la méthode « setup()», au niveau de laquelle, nous spécifions la fonctionnalité de l'agent. Cette dernière doit être développée dans un environnement JAVA, et avoir une extension « .class » pour pouvoir être exécutée par JADE. Dans notre cas, nous développons les corps des classes dans l'environnement de programmation « BleuJ ».

Notre agent étend donc la classe «Agent» qui représente une super-classe commune pour tous les agents pouvant être définis. Cela permet à l'agent d'hériter les aspects fondamentaux cachés (qui traitent les facilités fournies par la plate-forme, telles que la mobilité, le clonage, la communication,… etc.), ainsi qu'un ensemble de méthodes qui peuvent être appelées pour implémenter les tâches spécifiques à l'agent, par exemple l'envoi des messages, la suspension et la reprise de certaines tâches, …etc.

```
import jade.core.Agent;
import jade.core.AID;
public class AgentCommerce extends Agent  {
  // declaration des objets de l'agent

            ……

  protected AgentCommerce (Agent a) {super(a);}
  protected void setup {

  // Affichage d'un commentaire
  System.out.println("Hello, AgentCommerce"+getAID().getName()+ "est prêt.");

  // Définition de la fonctionnalité de l'agent mobile.

            ……
    }
  }
```

Fig 4.9- Spécification partielle d'un agent exprimée avec JADE

La méthode «getAID()» de la classe «Agent», permet d'extraire l'identifiant de l'agent. De même, la méthode « getName()» permet d'avoir le nom attribué à l'agent. Notre agent mobile est identifié par l'"agent identifier" présentant une instance de la classe «jade.core.AID ».

Le service RMA de la plateforme JADE, permet au propriétaire de l'agent mobile, de le créer et de le manipuler d'une manière commode (voir Fig 4.5).

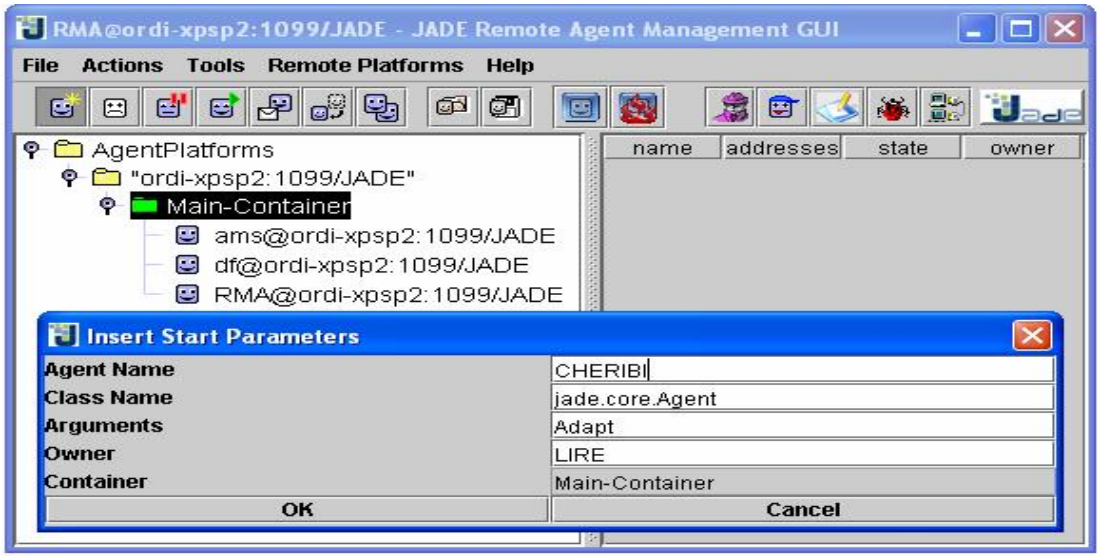


Fig. 4.10- Création d'un nouvel agent JADE.

6.2 Mobilité

Nous pouvons grâce à la structure du JADE (Jade3.4 version5847), de construire des agents qui propulsent la capacité de migrer d'une place à une autre dans le réseau. Le support pour la mobilité dans JADE consiste en un ensemble de classes de l'API et méthodes contenues dans le paquetage « jade.domain.mobility ». Nous utilisons pour programmer la mobilité de notre agent mobile, un ensemble de méthodes dédiées. Ce sont les méthodes doMove(), beforeMove()et afterMove (). Ces dernières sont contenues dans la classe «Agent» du paquetage «jade.core».

L'interface graphique de la plateforme JADE permet à l'hôte propriétaire de manager la mobilité de son agent; il suffit de le sélectionner, choisir l'emplacement de destination et active le bouton «Migrate Agent». Quand l'agent contrôleur reçoit l'événement par la méthode postGuiEvent(GuiEvent), il compose un message de DEMANDE avec un objet «MoveAction» (du paquetage jade.domain.JADEAgentManagement) qu'il envoie à l'agent intéressé.

6.3 Communication

Dans le cadre de notre travail, la seule communication que nous envisageons, est celle ayant lieu entre l'agent mobile et son propriétaire. Ce dernier peut à n'importe quel moment lui envoyer un message comportant un ordre d'arrêt, de migration, de retour, etc. De l'autre côté

l'agent mobile peut informer son propriétaire de son état d'avancement, ou d'éventuels problèmes rencontrés.

La communication est une des plus importantes opportunités offertes par JADE. Le paradigme de communication adopté est le passage des messages asynchrones «asynchronous message passing». Chaque agent détient une sorte de boite au lettres «the agent message queue» destinée à recevoir les messages provenant des autres agents. Tout message arrivant sur cette file est notifié pour qu'il puisse être traité par l'agent. Les messages échangés dans JADE ont le format spécifié par le langage ACL défini par FIPA. (http://www.fipa.org).

Pour implémenter l'envoie d'un message devant être envoyé par notre agent mobile à son propriétaire, nous l'instancions à partir de la classe ACLMessage.

Exemple.1

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.addReceiver(new AID("CHERIBI", AID.ISLOCALNAME));
msg.setLanguage("Français");
msg.setContent("Echec génération clé");
send(msg);
```

Fig 4.11- Exemple de spécification d'un message ACL

Grâce à l'interface graphique de JADE; envoyer un message par le propriétaire de l'agent mobile est aussi simple que le remplissage des champs d'un objet ACLMessage et l'appel d'envoi.

Le canal de communication ACC « Agent Communication Canal » fournit le chemin pour les interactions de base entre les agents dans et en dehors de la plateforme ; c'est la méthode de communication implicite qui offre un service fiable et précis pour le routage des messages. Le langage standard des messages FIPA ACL impose le codage, la sémantique et la pragmatique des messages. FIPA spécifie que les messages transportés entre les différentes plateforme doivent être codés sous forme textuelle.

Fig 4.12- fenêtre d'envoie de message ACL

Nous pouvons bénéficier également de quelques outils graphiques qu'offre JADE, permettant une meilleure gestion de la communication. Il s'agit de l'agent «Dummy» et l'agent

«Sniffer». «Dummy» est très puissant pour inspecter les échanges de messages entre agents, et donne une interface graphique pour l'édition, l'écriture et l'envoi des messages ACL vers les agents , et de recevoir et lire les messages des autres agents, et éventuellement sauvegarder ces messages. «Sniffer» quant à lui, suit les communications dans la plateforme et donne une interface graphique pour afficher les échanges de messages entre les différents groupes d'agents.

L'hôte propriétaire peut se servir de ces deux outils pour envoyer et superviser les messages de son agent mobile au cours de son cycle de vie. Cela lui permet ainsi de détecter les défaillances de réception des messages ou les éventuels traitements illégaux de ce système de messagerie.

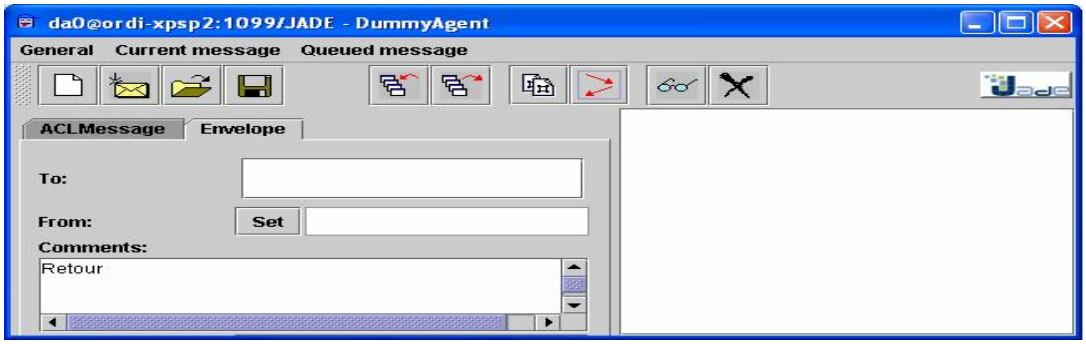


Fig 4.13- L'agent « Dummy»

Fig4.14- L'agent « Sniffer»

6.4 Sécurité

Nous utilisons «JADE-S version 2» qui remplace et étend «JADE-S version 1», fournissant le service de la sécurité dans JADE via le service «JADE-S add-on» [10].

En fait, «JADE-S» comprend quatre services :

- jade.core.security.SecurityService: ce service se charge du mécanisme d'authentification et fournit quelques fonctionnalités telles que le moteur de cryptage et la gestion des clés.

- `jade.core.security.permission.PermissionService:` assure le contrôle d'accès.
- `jade.core.security.signature.SignatureService:` signe les messages demandé par l'émetteur et vérifie la validité des messages reçus.
- `jade.core.security.encryption.EncryptionService:` se charge du cryptage des messages à envoyer et le décryptages des messages reçus.

Dans le cadre de notre travail, nous nous contentons d'utiliser le service de cryptage et celui de la signature assurant –respectivement- les propriétés de confidentialité et d'intégrité.

Sachant qu'un message ACL est composé de deux parties : l'«envelope», contenant les informations de transport, et le « payload » comprenant les données envoyées ; Dans «JADE-S», la signature et le cryptage sont toujours appliqués sur le «payload» afin de protéger les informations échangées.

6.5 Implémentation des comportements de l'agent mobile

Conformément à ce que nous avons conçu précédemment (au niveau du chapitre III), notre agent doit être capable d'accomplir plusieurs tâches en réponse à différents évènements extérieurs. Pour cela il est équipé de plusieurs comportements qui seront exécutés en fonction de sa stratégie d'adaptation. En terme d'implémentation, JADE nous facilite la tâche, par la proposition sous forme de classes d'un ensemble de schémas prédéfinis de comportements «Behavours», ainsi que des sous-comportements «subBehaviours» prêt à l'emploi.

Pour implémenter un comportement, nous devons définir un ou plusieurs objets de la classe «Behaviour», les instancier et les ajouter à la file des tâches de l'agent dans la méthode `setup()`. Pour chaque objet de type «Behaviour» nous implémentons une méthode «action()» comprenant le traitement à effectuer par celui-ci lorsqu'il est en cours d'exécution, ainsi que d'une méthode «done()» vérifiant si le traitement est terminé.

Pour que l'agent exécute les tâches implémentées dans les objets « Behaviours » que nous avons définis, nous appelons la méthode *addBehaviour(Behaviour)* de la classe *Agent* dans l'endroit voulu au sein de la méthode *setup()*. *L*a méthode *removeBehaviour(Behaviour)* nous permet d'assurer l'opération inverse. Pour

les comportements composés, nous utilisons les méthodes : *addSubBehaviour(Behaviour)* et *removeSubBehaviour(Behaviour).*

(i) Nous utilisons la classe «OneShotBehaviour » héritant de la classe «SimpleBehaviour» pour modéliser les comportements simples, s'exécutant une seule fois au niveau de chaque hôte de manière atomique, tel que l'acquisition des données, et l'affichage des résultats. (Dans ce type de comportement, la méthode action() est exécutée une seule fois et la méthode done() retourne toujours la valeur «true»).

Exemple2.

```
import jade.core.behaviour.*;
public class Acquisition extends OneShotBehaviour
{ public Acquisition (Agent a) {super(a);}
  public void action() {
   // Acquérir les donnés
      ……….
   }
}
```

Fig 4.15- Spécification d'un exemple d'un « OneShotBehaviour »

(ii) Nous utilisons la classe «CyclicBehaviour» héritant elle aussi de la classe «SimpleBehaviour» afin de spécifier les suites des actions opérations s'exécutant continuellement tant que l'agent est en vie, telles que la réception des messages venant de la part de l'hôte propriétaire. Ce type de comportements se caractérise par le fait que la méthode «done()» est toujours implémentée en retournant la valeur «false», pendant que la méthode « action()» est exécutée autant de fois qu'elle est invoquée.

Exemple3.

```
import jade.core.behaviour.*;
import jade.lang.acl.ACLMessage.*;
public class Reception-MSG extends CyclicBehaviour
{ public Reception-MSG (Agent a) {super(a);}
  public void action() {
    ACLMessage received = Reception_MSG.receive();
   // Traiter message reçu.
      ……..
   }
}
```

Fig 4.16- Spécification d'un exemple d'un « CyclicBehaviour »

(iii) Nous implémentons quelques traitements conditionnés par le temps via le comportement «`WakerBehaviour()`» dont les méthodes «`action()`» & «`done()`» sont toujours implémentées après une certaine période de temps. Par exemple, si le client ne répond pas au bout d'un certain temps ou bien consomme trop de temps, l'agent s'arrête immédiatement.

Exemple 4.

```
import jade.core.behaviour.*;
public class Attente extends Behaviour {
    public Reception-MSG (Agent a) {super(a);}
    public void setup() {
      addBehaviour(new WakerBehaviour(this, 10000) {
            public void handleElapsedTimeout() {
            // perform operation X
            ……..……
            }
      }}}
```

Fig 4.17- Spécification d'un exemple d'un « `WakerBehaviour` »

(iv) La classe «`CompositeBehaviour`» nous sert à modéliser les comportements spécifiés dans les expressions abstraites qui sont des comportements composés, dont les actions effectuées sont définies dans les *comportements enfants* « `ChildBehaviour` » (appelés aussi *sous-comportements*, «`SubBehaviours`»). Plus particulièrement, nous utilisons la Classe «`FSMBehaviour`» héritière de la classe «`CompositeBehaviour`» exécutant les *comportements enfants* représentant les micro-composants spécifiés suivant un automate à états finis que nous définissons.

Notons que dans un automate à état fini, chaque état « State » représente un *comportement enfant*, et les transitions entre les états représentent les règles de succession des sous-comportements. Lorsqu'un *comportement enfant* termine, sa valeur de fin indique le prochain état à atteindre. Le processus se termine lorsqu'un sous-comportement associé à un état final a été exécuté.

Exemple5 : Nous décrivons dans l'exemple suivant, un comportement composé de notre agent lui permettant d'assurer la fonction nommé « *commerce* » suivant un automate à état fini simple. Nous modélisons les comportements enfants comme des comportements simples en les instanciant à partir de la classe «`SimpleBehaviour`».

```
// Définition de l'agent_FSM
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
public class ComFSM extends Agent {
private Behaviour b;
public void setup() {
      b = new Commerce(this);
      addBehaviour(b);
}
}

//définition du FSM commerce

import jade.core.Agent;
import jade.core.behaviours.FSMBehaviour;
import jade.core.behaviours.OneShotBehaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;

public class Commerce extends FSMBehaviour {

// Déclaration des états FSM
private static final String E_DB = "Début";
private static final String E_CL = "Client";
private static final String E_FR = "Fournisseur";
private static final String E_AC = "AchComp";
private static final String E_AD = "AchDeg";
private static final String E_VC = "VntComp";
private static final String E_VD = "VntDeg";
private static final String E_AR = "Arrêt";
private static final String E_FN = "Fin";
private static final String E_ER = "Erreur";

// Déclaration des transitions FSM
private static final String Ach = "Acheter";
private static final String Vdr = "Vendre";
private static final String Cbn = "ConfBn";
private static final String Cmy = "ConfMy";
private static final String Cmv = "ConfMv";
private static final String AchC = "Achat Complet";
private static final String AchD = "Achat Degradé";
private static final String VtC = "Vente Complète";
private static final String VtD = "Vente Degradé";
private static final String Ar = "Arrêter";


//Définition du constructeur commerce
public Commerce(Agent a) {
      super(a);
```

```
// Instantiation des comportements enfants
registerFirstState(new PreComp(myAgent), E_DB);
registerState(new DeuComp(myAgent), E_CL);
registerState(new TrComp(myAgent), E_FR);
registerState(new QuComp(myAgent), E_AC);
registerState(new CnComp(myAgent), E_AD);
registerState(new SxComp(myAgent), E_VC);
registerState(new StComp(myAgent), E_VD);
registerState(new OtComp(myAgent), E_AR);
registerLastState(new NfComp(myAgent), E_FN);
registerLastState(new DxComp(myAgent), E_ER);

// Définition des transition du FSM
registerTransition(E_DB, E_CL, Ach);
registerTransition(E_DB, E_FR, Vdr);
registerTransition(E_CL, E_AC, Cbn);
registerTransition(E_CL, E_AD, Cmy);
registerTransition(E_CL, E_AR, Cmv);
registerTransition(E_FR, E_AR, Cmv);
registerTransition(E_FR, E_VC, Cbn);
registerTransition(E_FR, E_VD, Cmy);
registerTransition(E_AC, E_FN, AchC);
registerTransition(E_AD, E_FN, AchD);
registerTransition(E_AR, E_FN, Ar);
registerTransition(E_VC, E_FN, VtC);
registerTransition(E_VD, E_FN, VtD);

// Définition des transition par défaut
registerDefaultTransition(E_DB, E_ER);
registerDefaultTransition(E_CL, E_ER);
registerDefaultTransition(E_FR, E_ER);
registerDefaultTransition(E_AC, E_ER);
registerDefaultTransition(E_AD, E_ER);
registerDefaultTransition(E_AR, E_ER);
registerDefaultTransition(E_VC, E_ER);
registerDefaultTransition(E_VD, E_ER);

//Chargement du premier comportement enfant.
scheduleFirst();
} // Fin constructeur commerce


/**Définition des comportement enfants avec
*«Simplebehaviours»**/
      ……
```

Fig 4.18- Spécification d'un exemple d'un « `FSMBehaviour` »

CONCLUSION

Tout au long de ce chapitre, nous avons essayé de clarifier notre approche de protection de l'agent mobile. Grâce à l'étude de cas, nous avons pu voir les points restés vagues dans le chapitre précédent. Nous avons également pu constater les aspects techniques lors de l'implémentation de l'application considérée.

Nous avons vu comment exploiter les opportunités et les facilités offertes par la plateforme JADE dans notre cas de figure. Cette dernière nous a permis de développer les aspects clés de notre stratégie de manière essaie et transparente, par la proposition de la notion des comportements Jade.

CONCLUSION GENERALE

« Ce n'est pas la fin. Ce n'est même pas le commencement de la fin. Mais, c'est peut-être la fin du commencement »
Winston Churchill

Le contexte de notre travail concerne la sécurité dans les systèmes d'agent mobile. Bien que le domaine de la sécurité soit relativement ancien, le problème est encore loin d'être résolu. Nous pouvons même dire qu'il ne sera jamais possible de sécuriser totalement un système informatique. Il y aura toujours des hackers de génie pour découvrir de nouvelles failles et les exploiter. Seulement, il est toujours possible de rendre une attaque plus difficile, et si le pirate éprouve trop de difficulté à pénétrer un système plutôt qu'un autre, il changera certainement de cible pour en trouver une plus facile. Toutefois, pour pouvoir minimiser les risques et assurer un niveau de sécurité satisfaisant, il faut procéder à une analyse attentive avant de mettre en œuvre une politique de sécurité.

Le travail présenté dans ce mémoire s'inscrit dans le cadre de la sécurisation des agents mobiles contres les attaques des hôtes malveillants. Nous avons proposé une approche de protection basée sur l'adaptabilité dynamique. Le but est de permettre à l'agent mobile de varier son comportement à chaque fois qu'il se déplace dans le réseau afin de compliquer les éventuelles opérations d'espionnage. Bien entendu, le changement de comportement est réalisé en tenant compte des contraintes d'exécution reflétant les contextes de l'environnement de travail, d'où le terme 'adaptabilité'.

Afin de doter l'agent mobile de la faculté d'adaptabilité, nous avons proposé de le concevoir selon une structure réflexive. Ce modèle offre au système la capacité de réfléchir et d'agir sur soi. Il permet à l'agent mobile d'analyser aussi bien la situation de son environnement que son propre état, pour décider des actions à entreprendre et les réaliser. Sur le plan architectural, nous avons structuré notre agent mobile sous forme de composants. Nous avons mis en évidence deux niveaux de composants. Le premier englobe des composants de granularité fine, « micro-composants ». Ces derniers ont pour finalité, l'implémentation de la fonctionnalité de l'agent mobile. Le deuxième niveau comporte des

composants de granularité plus large dont la mission est la réalisation de l'adaptation de l'agent mobile.

Dans le but de montrer la faisabilité du modèle proposé et de mieux comprendre la stratégie mise en oeuvre, nous avons réalisé une étude de cas dans le domaine du e-commerce. De plus, nous avons implémenté les aspects les plus importants de l'application considérée. Nous nous sommes servi, pour cela, des facilités offertes par la plateforme JADE version 3.4. Cette implémentation nous a permis de percevoir les aspects techniques de notre approche.

Finalement, ce travail nous a permis de montrer comment bénéficier des techniques de l'adaptabilité dynamique pour les utiliser pour le compte de la sécurité des agents mobiles. Néanmoins, nous comptons apporter les améliorations suivantes à notre travail :

- La validation de notre approche à travers diverses applications avec une série d'expérimentation. Une démarche se voit validée si et seulement si elle est approuvée sur terrain.
- Nous avons considéré dans notre étude de cas, que l'agent mobile se base sur les questions/réponses pour communiquer avec son environnement extérieur. Nous souhaitons enrichir cet aspect pour qu'il puisse, à titre d'exemple, vérifier lui-même son environnement.
- L'approche de protection proposé peut être complétée à l'aide d'autres services ou bien combinée avec d'autres mécanismes confrontant d'autres types d'attaques telle que la ré-exécution de l'agent mobile, dans le but de donner un niveau de sécurité plus large.

REFERENCES

[1] M. Alfalayleh & L. Brankovic "*An Overview Of Security Issues And Techniques In Mobile Agents"*, University of Newcastle, Australia. Conference on Communications and Multimedia Security, (sec.isi.salford.ac.uk), 2004.

[2] N. Amano & T. Watanabe, *"Towards Constructing Component-based Software Systems with Safe Dynamic Adaptability"*, In Proceedings of the international workshop on Principles of …(portal.acm.org), Tatsunokuchi Ishikawa, Japan. 2002.

[3] N. Amara-Hachmi1 & A. El Fallah-Seghrouchni, "*Towards a generic architecture for self-adaptiv,"* Proceedings of 5th European Workshop on Adaptive Agents and MultiAgent Systems (AAMAS'05), Paris, 2005.

[4] L-F.Andrade, & J-L. Fiadeiro. "*Architecture Based Evolution of Software Systems"*. Third International School on Formal Methods for the design of Computer, communication and Software Systems, SFM 2003, Italy, Nov 2003.

[5] N-M. Belaramani. "*A component-based software system with functionality adaptation for mobile computing"*. Master's thesis, 2002.

[6] F. Bellifemine, G. Caire, T. Trucco, & G.Rimassa, *"Jade Programmer's Guide"*. JADE 3.0b1, 2003. <URL : http://sharon.cselt.it/projects/jade>.

[7] L. Bellissard, N. De Palma & M. Riveill, "*Dynamic reconfiguration of agent-based applications"*, ACM European SIGOPS Workshop, Sintra (Portugal), Sep 1998.

[8] M. Bergeron, *"Specification, modelisation et analyse du dialogue entre agents par l'intermediaire des engagements sociaux*", theses.ulaval.ca, volume TWLT-13 – 2005.

[9] E. Bierman, T. Pretoria & E. Cloete, "*Classification of Malicious Host Threats in Mobile Agent Computing"*, University of South Africa. In Proceedings of the annual research conference, portal.acm.org. 2002.

[10] Board "*Jade Security Guide*", JADE Security Add-On GUIDE, JADE 3.4, Copyright (C) 2004 TILAB S.p.A. Feb 2005.

[11] J-P. Briot & Y. Demazeau, "*Introduction aux agents*: *Principes et architecture des systèmes multi-agents"*, collection IC2, Hermès, 2001.

[12] J-P. Briot *"Objets et Agents pour Systèmes d'Information et Simulation*", DEA Laboratoire d'Informatique de Paris 6 CNRS, 2001.

[13] C. Castelfranchi, & R. Falcone, *"Trust is much more than subjective probability: mental components and sources of trust"* , 32nd Hawaii, International Conference on System Sciences - Mini-Track on Software, Agents, Maui, Hawaii, 2000.

[14] D. Chefrour & F. André. *"Auto-adaptation de composants ACEEL coopérants"*. 3ème Conférence Française sur les Systèmes d'exploitation(CFSE'3), France, Oct 2003.

[15] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris & G. Tsudik, *"Itinerant Agents for Mobile Computing,"* IEEE Personal Communications, vol. 2, no. 5, pp. 34-49 Oct 1995.

[16] M. Dageforde. *"Security Features Overview"*, from Sun Microsystems, Inc. The JavaTM Tutorialm, Dec 2003.

[17] P-C. David & T. Ledoux, "*Une infrastructure pour middleware adaptable"*. Rapport de DEA, École des Mines. Université de Nantes, Sep 2001.

[18] B. Drieu *"L'intelligence artificielle distribuée appliquée aux jeux d'équipe situés dans un milieu dynamique"* : l'exemple de la RoboCup. Oct 2001.

[19] W. M. Farmer, J. D. Guttman, & V. Swarup, "*Security for mobile agents: Authentication and state appraisal",* In Proceedings of the European Symposium on Research in Computer Security (ESORICS), pages 118--130, Sep 1996.

[20] J. Ferber, O. Gutknecht & F. MICHEL, "*Agent-Oriented Software Engineering", IV th InternationalWorkshop (AOSE),* 2935, pp 214-230. Springer Verlag, Australia, mars 2004.

[21] S Ghernaouti-Hélie, *"Stratégie & Ingenierie de la sécurité des reseaux"*, InterEditions, Dunod Paris 1998. <http://www.priceminister.com/navigation/list/category/ livres_informatique>.

[22] Z.Guessoum, & M. Occello. *"Environnements de développement. Dans Principes et architectures des systèmes multi-agents",* Hermès, Lavoisier, pp.177-202, 2001.

[23] M. S. Greenberg, J. C. Byington, T. Holding, D. G. Harper, " *Mobile Agents and Securiy*", IEEE Communications Magazine 374, 377, 379, 380, pp. 76-85, juillet 1998.

[24] S. Hacini, *"Using adaptability to protect mobile agents code,"* in IEEE International Conference on Information Technology pp. 49-53, ITCC, Las Vegas, 2005.

[25] S. Hacini, Z. Guessoum, & Z Boufaida, *"Using a trust-based key to protect mobile agent code",* Transactions On Engineering, Computing And Technology Enformatika V16 2006 ISSN 1305-5313, pp 326-331, Italy, 2006.

[26] S.Hacini, H.Cheribi, & Z.Boufaida *"Dynamic Adaptability using Reflexivity For Mobile Agent Protection"* the Proceedings XVII. International Conference on Computer and Information science and Engeneering, pp220-227 CISE, Cairo, Egypt, 2006.

[27] D. Hagimont *"Construction d'Applications Réparties, Systèmes à agents mobiles "* INRIA Rhône-Alpes, 1999. <http://sirac.inrialpes.fr/~hagimont>.

[28] M. Hefeeda & B. Bhargava, "*On Mobile Code Security",* Purdue University, West Lafayette, IN 47907, U.S.A, Okt, 2001.

[29] F. Hohl, *"Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts,"* Lecture Notes in Computer Science No. 1419, Springer-Verlag, pp. 92-113, 1998.

[30] W. Jansen & T. Karygiannis *"Mobile Agent Security",* NIST Special Publication 800-19 – National Institute of Standards and Technology, Computer Security Division Gaithersburg, MD 20899.

[31] A. Josang, & S. Lo Presti, *"Analyzing the relationship between risk and trust,"* in T. Dimitrakos (editor) the Proceedings of the Second International Conference on Trust Management, Oxford, 2004.

[32] Y. Kun, G. Xin & L. Dayou "*Security in Mobile Agent System: Problems and Approaches",* JiLin University, Changchun. ACM SIGOPS Operating Systems Review, Volume 34, Pages 21-28, (ACM Press New York, NY, USA). Jan 2000.

[33] S. Lam Pauline "*Context-Aware State Management for Supporting Mobility in a Pervasive Environment*". A thesis submitted for the degree of Master. The University of Hong Kong. Aug, 2004.

[34] T. Ledoux & Noury M.N.Bouraqadi-Saâdani, "*Adaptability in mobile agent systems using reflection,"* in ECOOP, Workshop on Reection and Metalevel Architectures, Cannes, France, 2000.

[35] T. Lemlouma & N. Layaida. "*Context-aware adaptation for mobile devices*". In Proceedings of the 2004 IEEE International Conference on Mobile Data Management, pages 106–111, Jan 2004.

[36] R. Lenglet. *"Conteneurs adaptables pour le déploiement de composants",* département DTL/ASR de France, 18-06-2001.

[37] S. Leriche, & J. Arcangeli., "*Vers un modèle d'agent flexible,"* In : Journées Multi-Agent et Composant, JMAC'06, Nîmes, 2006.

[38] W-Y Lum, C.Francis & M. Lau. "*On balancing between transcoding overhead and spatial consumption in content adaptation*". In Proceedings of the 8th Annual International Conference on Mobile Computing and Networking, pages 239–250, Sep 2002.

[39] P.Maes, *"Modeling adaptative autonomous agents"*. Tech. rep., MIT Media-Laboratory, (Postscript). Artificial Life Journal, edited by C. Langton, Vol. 1, No. 1 & 2, pp. 135-162, MIT Press,1998.

[40] J. Malenfant, F. Peschanski, L. Seinturier & J-P. Briot. *"ALADYN : Architectures logicielles pour l'auto-adaptabilité dynamique"*, Université Pierre et Marie Curie UFR d'informatique.

[41] Marangozova, "*Adaptabilité* ", Rapport de magister, Sep 1999. <URL:http://rangiroa@essi.fr/rapport/1900/99-magister3-marangozova.pdf>.

[42] C. Olivier, *"Approche réflexive des liaisons entre objets répartis"*, Rapport DEA Informatique, Effectué au laboratoire SIRAC, juin 2000.

[43] A. Occello *"Composants* : *Vers une adaptation dynamique cohérente*", DEA d'Informatique. Université de Nice Sophia Antipolis. Rapport de stage, 2002.

[44] J. Page, A. Zaslavsky, & M. Inrawan, "*Coontering Security Vulnirabilities In Agent Execution Using A Self Executing Security*", In Proceding of the 3rd International Joint conference on Autonomous Agents and Multi-Agents (AAMA2004), Vol 3, pp.1486-1487, New York (USA), Jul 2004.

[45] S. Perret, "*Agents mobiles pour l'accès nomade à l'information répartie dans les réseaux de grande envergure*", Thèse de Doctorat. Université Joseph Fourier - Grenoble I 1997.

[46] J-F Pillou, "*Tout sur la sécurité informatique*", - 1 vol. (XI-202 p.) – EAN 9782100496549. Dunod. Paris. 2005.

[47] F. Ramos, H. Unger, V. Larios & J.Ferber, *"Les systemes multi-agents, Vers une intelligence collective"*, Advanced Distributed Systems- 5th International School and Symposium, ISSADS, Guadalajara,... - pp 299, 2005.

[48] J. Riordan & B. Schneier, "*Environmental Key Generation Towards Clueless Agents,*" , Springer-Verlag, Lecture Notes in Computer Science No. 1419, 1998.

[49] V. Roth, *"Secure Recording of Itineraries Through Cooperating Agents"*, in Proceedings of the ECOOP, 4th Workshop on Mobile Object Systems: Secure Internet Mobile Computations, pp.147-154, INRIA, France, 1999.

[50] J-M. Sahut, "*Agents-négociateurs et usages dans le e-commerce*", Projets de recherche INT, Fondation LLR - Sept 2001.

[51] T. Sander & C. Tschudin, *"Protecting Mobile Agents Against Malicious Hosts"*, Springer- Verlag, G. Vigna (Ed.). Lecture Notes in Computer Science No. 1419, pp 44-60.1998.

[52] J-P Sansonnet, "*Applications agents pour le commerce électronique*" Master 2 Recherche Informatique – Université Paris Sud-XI , Feb, 2005 .

[53] D. Scott, A. Beresford & A. Mycroft, "*Spatial Security Policies for Mobile Agents in a Sentient Computing Environment*", Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 0302-9743 (Print), pp102-117, 19 Feb, 2004.

[54] S. Staamann & U. G. Wilhelm *"A technical approach to privacy based on mobile agents protected"*. Thèse N. O. 1961. Ecole Polytechnique Fédérale De Lausanne, 1999.

[55] F. Taœani. J-C Fabre , & M-O Killijian " *La Réflexivité dans les architectures multi-niveaux : application aux systèmes tolérant les fautes"*. Thèse de Doctorat de l'Université Paul Sabatier de Toulouse. Jan 2004.

[56] K. Tan & L. Moreau, "*Extending Execution Tracing for Mobile Code Security*", In Proceedings of Second International Workshop on Security of Mobile MultiAgent Systems (SEMAS'2002), pages 51-59, Bologna, Italy 2002.

[57] G. Vigna, *"Protecting Mobile Agents Through Tracing,"* Proceedings of the 3rd ECOOP Workshop on Mobile Object Systems, Jyvälskylä, Finland, Jun 1997. <URL:http://www.cs.ucsb.edu/~vigna/listpub.html>.

[58] G. Vigna, "*Cryptographic Traces for Mobile Agents,*" in: Giovanni Vigna (Ed.), Mobile Agent Security, LNCS 1419, Springer, pages 137-153, 1998.

[59] P. Vigneras, "*Agents Mobiles* : *Une implémentation en Java et sa modélisation*". Mémoire de DEA, Université Bordeaux I, Jan 2000.

[60] R. Wahbe, S. Lucco, T. E. Anderson & S. L. Graham, *"Efficient software-based fault isolation"*, In Proceedings of the 14th ACM Symposium on Operating Systems, Principles, pages 203-216, Dec. 1993.

[61] Y. Ye, X. Yi & S. Kumaran *"Coalition Signature Scheme in Multi-agent Systems"*, Victoria University, in Proceedings of 2nd International Workshop on Security of Mobile Multiagent Systems (SEMAS'02), Bologna, Italy, Jul 2002.

[62] W. Zhi, & G. Zhogwen, *"A Dynamic Security Adaptation Mechanism For Mobile Agents"*, Proceedings of the International Computer Congress, pp 334-339, China, 2004.

[63] AGLETS (IBM Corporation). <URL:http://www.trl.ibm.co.jp/aglets/>.

[64] DIMA, OASIS, Laboratoire d'Informatique de Paris 6 (LIP6)
<URL :http://wwww-poleia.lip6.fr/~guessoum/DIMA.html>.

[65] ODYSSEY (General Magic).
<URL:http://www.genmagic.com/technology/mobile_agent.html>.

[66] JADE: Java Agent Development Framework. <URL: http://www.jade.cselt.it/>.