UNIVERSITÉ PARIS-EST

Université Paris 12 Val de Marne

École doctorale Sciences et ingénierie : matériaux, modélisation et environnement

Faculté des Sciences et Technologie

Département d'informatique

Laboratoire d'algorithmique, complexité et logique

Surveillance logicielle à base d'une communauté d'agents mobiles

Thèse présentée et soutenue publiquement le 30 novembre 2009

en vue de l'obtention du grade de Docteur en Informatique

Mâamoun BERNICHI

Directeur de thèse Fabrice MOURLIN

Composition du jury :

Président	Anatole SLISSENKO
Rapporteurs	Hanna KLAUDEL
	Christophe CERIN
Examinateurs	Elizabeth PELZ
	Frédéric LOULERGUE

Le style de ce document est basé sur la feuille de style mise à disposition par l'université Lyon 2 (http://theses.univ-lyon2.fr/?q=fr/node/26)

À mon grand père,
pour me guider, m'a tant transmis

À mon père,
pour toutes ces valeurs qu'il m'a transmises.

À ma fille,
puissions-nous à notre tour te guider
et te transmettre ce qui nous a été transmis.

Remerciements

Mes remerciements vont en premier à Fabrice Mourlin mon directeur de thèse et au Professeur Elizabeth Pelz responsable de l'équipe "systèmes communicants" du laboratoire LACL. Leur confiance, leurs manières d'aborder la discipline, leurs démarches scientifiques, leur humanité et patience m'ont profondément touché tout au long de mes recherches.

Je tiens également à remercier les membres du jury pour l'intérêt porté à ce travail et pour leurs commentaires et critiques constructives.

Mes remerciements s'adressent aussi au Professeur Anatol Slissenko et au Professeur Gaétan Hains directeurs du laboratoire qui m'ont apporté des soutiens aussi bien financiers que moraux, et à tous les membres du laboratoire pour leur accueil.

Mes remerciements à Flore Tsila et Brigitte David pour leurs aides administratives précieuses.

Merci à tous les collègues du Groupe Initiatives Mutuelles qui m'ont supporté et appuyé durant les moments difficiles.

Avant de terminer, je remercie toute ma famille pour son réconfort, et tout particulièrement ma mère qui m'a soutenu tout le long de mon cursus universitaire.

Merci à toutes les personnes qui m'ont soutenu de loin ou de près.

J'adresse en dernier lieu un petit mot pour ma femme qui m'a accompagné et a supporté les bruits des clics, parfois, à des heures tardives. Je la remercie aussi pour son soutien et ses encouragements et pour le beau cadeau qu'elle m'a offert le 24 février 2009.

Table des matières

Introduction

Les agents mobiles sont des composants, des entités ou tout simplement du code applicatif ayant une propriété mobile leur permettant de se déplacer d'un dispositif à l'autre afin d'effectuer des traitements. Cette notion n'est pas une première, en effet les agents mobiles ont vu le jour dans les années 80 pour répondre entre autre à des problèmes d'adaptabilité. Ils interviennent plus généralement dans tout dispositif dont la conception n'a pas permis de prendre en compte les évolutions futures.

A ce jour les architectures logicielles sont de plus en plus complexes et sont limitées par un manque d'évolutivité. Cette contrainte porte sur plusieurs aspects, d'une part l'aptitude à redéployer une application existante sur une architecture physique plus vaste, d'autre part la possibilité de faire évoluer tout ou partie du logiciel déjà présent. Par exemple, considérons une application embarquée à bord d'un véhicule mobile autonome. Le parcours du véhicule dans un réseau autoroutier amène un flot d'informations qui permet au véhicule de circuler. Lorsque la limitation de vitesse est fixée à 90, le véhicule doit éventuellement ralentir. Lorsque l'information météorologique signale la présence de pluie, la distance réglementaire avec le véhicule précédant est rallongée. Des évolutions de cette application entraînent l'accroissement de la quantité des données dues en particulier au nombre de véhicules circulant. Le réseau autoroutier étant européen, d'autres signalisations sont à prendre en compte, le format des données change. Lorsque l'application est déjà sur un parc de véhicules, il n'est pas possible de demander le retrait de ceux-ci afin de les remplacer. Il est essentiel de prévoir l'adaptabilité au changement.

La surveillance de ces véhicules lève aussi le problème de la gestion des versions. Comment envisager d'interagir avec une future application sans avoir préalablement prévu ce changement. Lorsque le trafic automobile sera intense comment s'assurer que l'application de surveillance ne sera pas saturée par la quantité des données reçues et que la qualité du service ne sera par perturbée.

Ces exemples illustrent les problèmes auxquels nous nous sommes attelés dans ce travail. Notre réseau d'étude n'est pas autoroutier mais informatique. Les véhicules sont alors remplacés par des ordinateurs permettant d'accueillir des agents pour effectuer des traitements.

Dans un premier temps, nous décrivons l'état de l'art de la mobilité et certaines études récentes et anciennes faisant référence dans ce domaine afin de montrer les avancées auxquelles nous sommes parvenus dans notre travail.

Dans le deuxième chapitre, nous développons notre problématique et abordons les limites des approches d'agents mobiles existantes pour avancer vers une solution optimale. Cette dernière est présentée dans les trois chapitres suivants où nous exposons respectivement notre analyse formelle de la mobilité, notre nouvelle approche conceptuelle des agents mobiles sachant que leur illustration est facile à formuler mais leur réalisation fait apparaître des choix techniques non triviaux et, pour terminer, nous présentons l'application de notre approche à la surveillance logicielle.

En conclusion, nous établissons le bilan de notre travail de thèse et nous abordons les perspectives que nous souhaitons mettre en oeuvre pour nos travaux futurs.

I. Etat de l'art

Dans ce premier chapitre préliminaire nous présentons les travaux antérieurs et récents ayant un lien avec notre travail de recherche. Ainsi, nous définissons les différentes caractéristiques d'un agent mobile, décrivons les applications à base d'agents mobiles, introduisons la notion de communauté d'agents mobiles, et nous apportons des exemples de plates-formes d'agent mobile pour enfin aborder la question de la surveillance logicielle.

I.1. Les caractéristiques d'un Agent mobile

Le terme de mobilité est souvent associé à des agents mobiles. Elle est considérée, dans la majorité des cas, comme une propriété orthogonale des agents qui ne sont pas tous, forcément, mobiles. Un agent est une entité qui peut résider sur un hôte et communiquer avec son environnement en utilisant les concepts conventionnels, comme l'appel des méthodes distantes (RPC) ou par un message de notification. Ces agents sont communément nommés agents stationnaires ou agents immobiles. Ils agissent seulement sur leur environnement d'origine où ils sont initialement construits. Toutefois, s'ils ont besoin d'informations ou d'interagir avec des agents résidants sur d'autres machines, ils utilisent des mécanismes de communication directe, basée sur des protocoles spécifiques basés sur le modèle client/serveur.

Contrairement aux agents stationnaires, nous trouvons des agents dits agents mobiles. Ces derniers, même en dehors de leur environnement, sont capables de communiquer avec d'autres agents et de se déplacer d'un hôte à l'autre. Ils peuvent ainsi exécuter des tâches malgré l'arrêt de leur machine initiale. Ces agents sont souvent qualifiés d'autonomes. Une propriété qui leur permet de se déplacer avec une totale liberté aménageant ainsi leur parcours initial et décidant des tâches à effectuer. Cependant, le paradigme d'agent mobile a soulevé, à ses débuts, plusieurs critiques liées à la sécurité considérée comme un obstacle à sa généralisation. Fort heureusement, les technologies ont évolués entraînant un intérêt pour les chercheurs et industriels au développement d'applications basées sur des agents mobiles dans ces domaines d'accès sur les télécommunications, PDA ou les bus

logiciels… il ne faut pas faire de confusion entre la mobilité de code et la mobilité d'utilisation. Celle-ci est plus rigoureusement nommée nomade. Pourtant, à ce jour, aucune solution industrielle disponible sur le marché n'utilise des agents mobiles [1] comme base de son architecture. Leur utilisation reste plus expérimentale ou spécifique à certains projets initiés par des indépendants ou des équipes de recherche.

Notre choix d'utiliser les agents mobiles n'est pas motivé par la technologie ou l'effet de mode, mais par les avantages qu'ils apportent à la construction d'un système distribué. À ce titre, nous citons sept raisons [2] optimisant leur utilisation :

Diminution de la charge du réseau : les systèmes distribués s'appuient souvent sur les protocoles de transmission impliquant différentes interactions, ce qui augmente le trafic réseau. Les agents mobiles permettent aux utilisateurs d'empaqueter une conversation puis de l'envoyer à une destination pour que l'interaction se fasse localement. Cette raison est très perceptible quand la taille des messages est nettement supérieure à celle des agents.

Maîtrise de la latence du réseau : les systèmes en temps réel, tels que des robots dans des processus de fabrication, doivent répondre instantanément aux changements de leur environnement. Le contrôle d'un tel système par un réseau d'une taille substantielle implique une latence significative, ce qui est inacceptable pour des systèmes critiques. Les agents mobiles peuvent se déplacer vers une machine afin d'agir localement et pouvoir exécuter certaines directives.

Encapsulation des protocoles : quand les données sont échangées dans un système distribué, chaque hôte dispose d'une implémentation pour envoyer les données et/ou interpréter les données reçues. Seulement, les protocoles évoluent pour s'adapter à de nouveaux besoins, pour être plus efficaces ou pallier à des problèmes de sécurité. Or, il est difficile voire quasi impossible de procéder à cette amélioration des protocoles. En conséquence, ces derniers peuvent poser problème. Les agents mobiles peuvent se déplacer sur un hôte distant pour ouvrir ou établir un canal basé sur des protocoles propriétaires. Dans ce cadre, l'agent mobile possède les moyens d'accéder au message qu'il transporte. Le format du message évolue, le moyen de consommer ce message aussi. Ce n'est pas le receveur qui change, c'est l'agent qui le transporte.

Exécution asynchrone et autonome : les tâches d'un dispositif mobile se fondent souvent sur des connexions réseau coûteuses ou fragiles et exigent une connexion ouverte sans interruption. Ce qui n'est pas économiquement ou techniquement faisable. Pour résoudre ce problème, ces tâches sont incluses dans les agents mobiles, qui sont alors expédiés dans

le réseau. Après avoir été envoyés, les agents deviennent indépendants du processus qui les a créés et peuvent fonctionner d'une façon asynchrone voire autonome. Le dispositif mobile peut se reconnecter après un temps nécessaire au traitement pour récupérer l'agent.

Adaptation dynamique : les agents mobiles pressentent leur environnement d'exécution et réagissent de façon autonome aux changements. Les agents mobiles multiples, ou plutôt un groupe d'agents mobiles a la capacité unique de travailler en communauté, de se repartir parmi les hôtes dans le réseau afin de maintenir la configuration optimale et résoudre un problème particulier. Cette collaboration s'établit de la manière suivante s'il n'y a pas d'agent sur un hôte1 alors l'agent de hôte2 exportera un agent dessus et réciproquement. Nous pouvons aussi dire que si un hôte s'arrête, les agents mobiles se notifient pour redéfinir leurs chemins et ainsi éliminer l'hôte en question de leurs parcours.

Hétérogénéité naturelle : les traitements en réseau sont fondamentalement hétérogènes, du point de vue matériel et logiciel. Les agents mobiles fournissent des conditions optimales pour l'intégration des systèmes sans faille. Ceci est dû à l'indépendance de la couche du transport et celle du traitement, ces deux dernières ne dépendent que de l'environnement d'exécution.

Robustesse et insensibilité aux défaillances : la capacité des agents mobiles à réagir dynamiquement aux situations et aux événements défavorables, facilite la construction de systèmes distribués robustes et insensibles aux défaillances. Si un hôte est en train de s'arrêter, tous les agents s'exécutant sur ce dernier sont avertis pour se déplacer et continuer leur opération vers un autre hôte. Les frontières de cette communauté ne sont pas inamovibles et le périmètre d'une communauté évolue au cours du temps en fonction, par exemple, des défaillances ou des nouvelles ressources à prendre en compte.

Ces diverses raisons sont à l'origine de travaux passés et présents dont les principaux sont mentionnés dans la section suivante.

I.2. Application à base d'agents mobiles

Depuis l'apparition de la notion de mobilité, de nombreuses plates-formes ont été développées afin de faciliter la programmation d'applications structurées en termes d'agents mobiles. Plusieurs papiers de synthèse [3,4] existent sur les principales plates-formes utilisées de nos jours. Le plus souvent, lorsque l'on parle de plates-formes, nous

regroupons deux niveaux élémentaires [5]. Le premier est le langage de programmation qui offre les primitives nécessaires à l'exécution, à la communication et à la mobilité. Il constitue le niveau de base. Le deuxième étant les services de haut niveau liés aux besoins ou l'administration, comme les annuaires, permettant aux agents la réalisation de leurs tâches.

I.2.1. Les langages sous-jacents

Un langage de programmation adapté aux agents mobiles doit proposer un ensemble de propriétés particulières décrites par [6]. Il doit prendre en compte principalement :

P1 - La manipulation du code : le langage doit être capable d'identifier, d'envoyer, de recevoir et d'exécuter le code exécutable représentant un agent mobile. Le langage proposera une abstraction de haut niveau pour réaliser une action de migration.

P2 - L'hétérogénéité : la construction d'un système distribué est caractérisée par l'hétérogénéité des éléments le composant. Le langage devra être indépendant de l'architecture matérielle et du système d'exploitation. L'agent doit pouvoir aller potentiellement sur tous les éléments du système. On voit ici que les langages interprétés sont certes les plus adaptés. Ces langages utilisent une machine virtuelle, dont le byte code est interprété.

P3 - La sécurité : en offrant un environnement à l'exécution de code étranger, on accroît la possibilité de recevoir des éléments indésirables au sein des systèmes. Le langage devra offrir un ensemble de mécanismes de protection, que ce soit par le cloisonnement ou une identification.

P4 - L'exécution autonome : pour permettre une réelle mobilité, le langage ne doit pas exprimer des liens directs vers les ressources des sites visités. Cette règle est similaire au principe de la loi Déméter [7]. Ceci évite de gérer les problèmes de disparition des sites et facilite la représentation d'un utilisateur déconnecté.

P5 - La performance : dans un système multi agents, chaque application utilise un ensemble d'agents s'exécutant en parallèle. Chaque agent nécessite un espace mémoire et des mécanismes de mise en route, de déplacement, etc., que le langage doit minimiser afin d'éviter les surcharges permettant ainsi un déroulement des applications dans un temps acceptable.

P6 – La structuration : le besoin d'un traitement ne correspond pas nécessairement au besoin d'un agent mobile. Ce découpage logiciel doit rester masqué à l'utilisateur. Une

fonctionnalité de recherche de ressources peut être réalisée par un agent mobile seul lorsque le domaine de recherche est de petite taille. En revanche, cette même fonctionnalité peut correspondre au déploiement d'une communauté d'agents de recherche dont le but commun est le résultat demandé. Aussi le langage doit offrir la notion de composition d'agents mobiles à la communauté afin de cacher cette notion d'échelle.

Plusieurs langages supportent la mobilité du code [4]. Le premier langage à proposer ces propriétés fut Téléscript [8] de Général Magic, il est conçu pour le développement de grands systèmes distribués. La sécurité a été l'un des facteurs qui ont orienté sa conception, ainsi que la capacité de la migration des unités de traitement pendant leur exécution. Néanmoins, le système, conçu pour une application industrielle, exigeait des ressources importantes et fût plutôt coûteux. Par conséquent, le système n'existe plus et sa valeur est surtout historique. Téléscript a été suivi de nombreux autres tels Obliq [9] qui est un langage interprété, basé sur les objets et non typé. Les objets Obliq sont locaux aux sites mais il est possible de déplacer des traitements d'un site vers un autre, autrement dit il prend en charge la mobilité forte mais ne permet pas de gérer les aspects sécurité de façon satisfaisante. Toutefois, le langage Java, un langage objet, reste aujourd'hui le plus répandu. Une des premières plates-formes à utiliser cette technologie fut Mole [10].

L'environnement Java a connu un rapide succès pour le développement d'applications distribuées, en partie parce que la machine virtuelle Java (JVM) est portée sur la plupart des systèmes courants. Elle permet une indépendance envers le réseau physique et les systèmes d'exploitation des sites hôtes. Mis en oeuvre en Java, les agents peuvent migrer entre des sites hétérogènes (P2). Java permet la mobilité du code et des objets grâce au mécanisme de sérialisation [11]. La sérialisation étant la transcription de l'état de la structure complexe d'un objet dans un format particulier, ainsi elle permet de capturer et de restaurer l'état des objets avant et après leurs déplacements entre sites (P1). La stratégie de sécurité est gérée à un niveau local pour l'emploi d'objets propres à la JVM d'accueil. Cette approche interdit ainsi à tout agent mobile d'être un potentiel agent intrusif voire envahissant. (P3)

Initialement, l'environnement Java ne permettait pas de capturer ou de restaurer l'état d'un processus en cours d'exécution. Il n'est donc pas possible de faire migrer simplement un agent entre deux machines virtuelles. De ce fait, elle ne supporte pas la mobilité forte dans la mesure où il n'est pas possible de redémarrer un agent au point où il s'est interrompu sur un autre site. Des travaux, sur l'extension de la machine virtuelle, ont été proposés [12,13]

afin d'étendre la machine virtuelle. Ces travaux n'ont pas abouti parce que c'est toujours aux développeurs de gérer les différents états rencontrés après une migration de ses agents. Plusieurs algorithmes de gestion de mémoire sont proposés par les différentes JVMs, certains de ces algorithmes font d'ailleurs partie de la spécification du langage tel que ConcurentGC. Des configurations particulières de cette machine virtuelle permettent de minimiser la recopie d'objet et diminuer aussi le coût de gestion des espaces mémoire (P5). Enfin, la composition d'agents mobiles en communauté d'intérêt commun (CIC) est, dans ce cadre, une structuration de données et de traitements à mettre en œuvre par la conception de l'application (P6). Cela correspond à la définition et la mise en œuvre de design pattern, présenté au chapitre IV.

Le langage Java autorise l'utilisation de références sur des machines physiques ou des ports particuliers de communication. Aussi la quatrième propriété (P4) est à mettre en œuvre par l'utilisateur du langage Java qui n'effectue pas ce contrôle. Malgré ces inconvénients, nous retiendrons que Java est le principal langage utilisé pour le développement des plates-formes supportant les agents mobiles. Nous retrouvons ce postulat dans d'autres études antérieures [14].

L'idéal serait de disposer d'un langage où cette notion de mobilité serait intrinsèque, à la manière des classes dans les langages de classes actuels. Il est difficile de ne pas faire de classes dans un langage comme Java. Mieux encore, serait de masquer la notion de mobilité au développeur pour lequel cette notion nécessite une abstraction trop forte. Un langage tel que Ocaml [15] permet ainsi une plus grande rapidité de développement en masquant à ses utilisateurs des structures jugées trop techniques. De tels langages sont du domaine futuriste au moment où ces lignes sont écrites, mais à n'en pas douter nous disposerons de ce pouvoir d'expression dans un avenir proche.

Actuellement l'environnement Java dispose d'une API Jini [16] permettant la fédération de plusieurs services. Cette technologie est une extension réseau de l'infrastructure, modèle de programmation pour les services de la technologie Java. Une vue de l'ensemble de la technologie Jini est présentée ultérieurement.

I.2.2. Les normes

L'émergence des nombreuses plates-formes expérimentales a rendu essentielle la proposition d'une harmonisation grâce à la standardisation des différents concepts

communs pouvant être identifiés. Cette normalisation devrait permettre à terme de rendre compatibles les différents systèmes.

On peut trouver à l'heure actuelle deux normes principales : il s'agit de la norme FIPA (Foundation for Intelligent Physical Agents) [17] et de la norme MASIF (Mobile Agent System Interoperability Facility) [18].

Lorsque deux normes existent au sein d'une même technologie, elles ont une tendance naturelle à s'opposer alors que, dans notre cas, elles penchent plus vers la complémentarité due à la différence de leurs domaines d'origine.

MASIF : la norme MASIF a été spécifiée par l'Object Management Group (OMG) qui se préoccupe généralement de l'hétérogénéité entre les systèmes, comme dans le cas de CORBA. Dans cette optique, le but, dans la norme MASIF, est de décrire les notions élémentaires permettant l'échange des agents entre différentes plates-formes. Pour ce faire, elle standardise la manière de gérer le code des agents, leur identification, la migration et l'adressage local.

FIPA : en revanche, la communauté à l'origine de FIPA étant celle des systèmes multi agents, plus proche de l'intelligence artificielle, elle va se situer à un niveau plus élevé, c'est-à-dire le niveau applicatif en décrivant les éléments nécessaires à la réalisation d'une application et principalement en détaillant la communication entre les agents. Le but est de décrire un ACL (Agents Communication Language), les ontologies et les protocoles de négociation permettant ainsi de définir parfaitement les interactions entre les agents.

Les deux standards MASIF et FIPA sont considérés comme des religions de la communauté de logiciels d'agents [19]. Leurs différences résident dans le fait que MASIF vise à permettre aux agents mobiles de migrer entre les systèmes d'agents du même profil par l'intermédiaire des interfaces normalisées de CORBA, contrairement à FIPA qui permet l'interopérabilité d'agents intelligents par l'intermédiaire de la communication normalisée des agents et des langues.

Ces deux normes tendent plus vers la complémentarité que vers la divergence. C'est déjà le cas de FIPA qui a inscrit dans son planning l'intégration des règles de MASIF sur la gestion de la migration.

I.2.3. Autre approche

Au premier abord, les applications à base d'agents mobiles nous laissent penser à des plates-formes de services web. Puisque pour les deux, nous parlons de composants logiciels écrits dans divers langages de programmation et surtout des standards employés.

Les services Web (en anglais Web services) [20] représentent un mécanisme de communication entre applications distantes à travers le réseau Internet indépendant de tout langage de programmation et de toute plate-forme d'exécution. Ils utilisent le protocole HTTP, comme moyen de transport pour que les communications s'effectuent sur un support universel et standard, également un échange basé sur le langage XML pour décrire les appels et les données échangées.

Grâce aux services Web, les applications peuvent être vues comme un ensemble de services métiers, structurés et correctement décrits, dialoguant selon un standard international plutôt qu'un ensemble d'objets et de méthodes entremêlés. Cela facilite la maintenance de l'application. L'interopérabilité fournie par les échanges en XML, autorise à modifier un composant (service Web) pour le remplacer par un autre, éventuellement développé par un tiers. Qui plus est, les services Web permettent de réduire la complexité d'une application car le développeur peut se focaliser sur un service, indépendamment du reste de l'application.

Il nous semble nécessaire de dire que, sur certains aspects et propriétés, les agents mobiles et les services Web ont des similitudes. Cependant, leurs aspects divergent sur le rôle des traitements par rapport aux données. Le traitement d'un Web service est immobile alors que les données en entrée lui arrivent en XML. En revanche un agent mobile fait entrer des traitements alors que les données sont immobiles. Celles-ci peuvent d'ailleurs être aussi au format XML. Un service Web implémente le métier et reçoit les données afin de les traiter alors qu'un agent mobile se déplace avec le métier et éventuellement les données.

Des études antérieures de comparaison des deux approches ont été réalisées pour aboutir à la même conclusion : les deux approches ont des similitudes et demeurent complémentaires pour palier à la problématique de sécurité [21].

I.3. Concept de communauté d'agents mobiles

Une communauté, d'une façon générale, désigne une association de plusieurs entités. Pourtant, une communauté d'agents mobiles ne peut pas être considérée comme un simple groupement d'un ensemble d'agents mobiles. Une communauté est décrite en fonction des caractéristiques qui lui sont propres, ce qui fait la particularité de chaque communauté. Prenons pour exemple la communauté des fourmis. En effet, chaque fourmilière peut être considérée comme une communauté de fourmis et chaque fourmilière possède ses propres particularités qui la différencient des autres communautés. En nous référant à ces communautés de fourmis, on peut dire que notre communauté d'agents mobiles est composée d'un ensemble d'agents mobiles ou chaque groupe de ces agents est dévoué à une mission bien précise. Ces groupes sont autonomes et complémentaires, il n'y a pas un groupe qui se distingue des autres, ils sont tous essentiels au bon fonctionnement de la communauté. Il est à noter que ces groupes d'agents communiquent entre eux à l'instar des fourmis qui utilisent la sécrétion des phéromones pour transmettre des informations aux autres fourmis de la même communauté.

Habituellement, les communautés d'agents sont connues pour arborer des interactions existantes entre agents autonomes**.** On cherche à déterminer l'évolution de ce système afin de prévoir l'organisation qui en résulte. Par exemple, en sociologie, on peut paramétrer les différents agents composant une communauté. En ajoutant des contraintes, on peut essayer de comprendre quelle sera la composante la plus efficace pour parvenir à un résultat attendu (construction d'un pont). Ce qui importe c'est le comportement d'ensemble et non pas le comportement individuel. La notion de communauté apparaît de façon naturelle dès que plusieurs agents mobiles interagissent dans un but commun. L'exemple type est la collecte d'informations où plusieurs agents se partagent le ramassage des données afin que la collecte dans son ensemble soit plus efficace. Un second exemple est l'emploi d'agents de surveillance logicielle pour être plus rapidement informé, plusieurs agents sont déployés sur des sites stratégiques. Un gestionnaire d'agents joue le rôle de concentrateur ou de responsable du groupe. Ces observations montrent le besoin de structurer un groupe en niveau afin que des traitements structurés puissent être effectués. Le dernier exemple illustre en plus la nécessité de communiquer facilement et rapidement au sein d'une

communauté sans avoir à négocier un protocole d'échanges sophistiqué pour assurer la confidentialité.

Nous avons donné une première définition opérationnelle d'une communauté d'agents. Initialement localisée sur un site d'exécution pour des raisons d'implémentation, une communauté était un simple ensemble d'agents en cours d'exécution. Nous nous orientons vers une approche plus générale où une communauté d'agents se répartie sur plusieurs sites connectés. Sa frontière physique évolue en fonction des déplacements d'agents. L'illustration de cette première implémentation s'apparente à une toile d'exécution dont la dimension et les caractéristiques physiques évoluent au cours du temps. Initialement, la toile est repliée sur quelques sites où des agents de la communauté sont présents. Au cours de leurs travaux, ces agents sont amenés à bouger pour satisfaire aux besoins de leur mission. Les liens de communication entre agents d'une même communauté sont préservés ce qui assure que la toile ne soit jamais coupée en deux ou plus. En revanche, cette toile est évolutive.

L'idée maîtresse est de masquer l'architecture physique par la notion de communauté afin de faciliter l'écriture des échanges de données entre membres de la communauté. Qu'ils soient locaux à un site ou distants, les échanges applicatifs restent, la seule solution technique qui diffère. Dans une optique semblable, l'échange d'informations entre membres de communautés différentes doit suivre une démarche similaire : masquer le découpage physique mais en ajoutant les contrôles sécuritaires pour assurer les aptitudes des agents et leur intégrité à communiquer au sein d'une communauté ainsi que sur les données transmises.

Il apparaît alors incontournable d'utiliser cette notion de communauté comme une propriété structurante d'un projet à base d'agents mobiles [22]. En revanche, elle en diffère car plusieurs communautés peuvent cohabiter à un même niveau et utiliser une communauté sous-jacente.

I.4. Exemples de plateformes d'agents mobiles

Nos recherches se sont focalisées sur les différentes plates-formes mobiles existantes non pas pour les comparer mais pour mettre en avant certaines de leurs caractéristiques afin de savoir quelle implémentation choisir pour répondre à nos besoins ou celle qu'il faut adapter

pour l'appliquer à notre approche de surveillance. Notre volonté n'est pas de réinventer la roue mais d'utiliser des études précédentes et de les valoriser.

Certaines plates-formes ne sont plus d'actualité, soit elles ont été abandonnées, soit elles sont pauvres en terme d'informations ou compliquées à utiliser. D'autres sont anciennes mais restent prometteuses et pertinentes malgré le critère d'âge, d'ailleurs elles sont toujours proposées dans différentes études récentes. [5].

Malheureusement, nous ne pouvons pas toutes les décrire. Nous allons en retenir six en fonction de leurs approches de communication, un aspect important pour une communauté d'agents mobiles.

Nous avons choisi les plates-formes suivantes :

1 **Agile Applets (Aglets)** : simpliste, utilisant le langage KQML et respectant la norme MASIF émulée sans CORBA depuis 2003. D'autres plateformes respectent la même spécification comme Concordia [23], Telescript [24], Grasshopper [25] ou MOA [26]. Seulement elles ne sont plus maintenues.

2 **Java Agent DEvelopment Framework (JADE)** : une plate-forme parmi tant d'autres, respectant le standard FIPA et toujours d'actualité. Le tableau ci-après (cf. Tableau 1) donne un état sur l'activité des plates-formes respectant le standard FIPA. En aucun cas il remet en cause le standard, il nous permet de motiver notre choix.

3 **Linda In Mobile Environnement (LIME)** : elle s'appuie sur le modèle Linda, un modèle de coordination et de communication parmi plusieurs processus parallèles opérant sur des objets stockés dans une mémoire partagée, virtuelle et associative. Ce modèle est implémenté comme un langage de coordination.

4 **PLANGENT :** est la plate-forme la plus proche de l'intelligence artificielle, en fait elle se concentre sur la question de l'intelligence comme un préalable aux fonctions des agents afin que ces derniers puissent comprendre l'utilisateur pour effectuer leurs tâches.

5 **Troms And COrnell Moving Agents (TACOMA)** : C'est un projet universitaire qui se concentre sur l'utilisation des agents mobiles pour palier aux problèmes traditionnellement abordés par d'autres paradigmes tels que le modèle client/serveur. Par ailleurs, ce projet offre une abstraction portant sur la communication déléguée qui propose un mécanisme fondé sur le concept du rendez-vous et réalisé par l'intermédiaire d'un "Briefcase" échangé entre les deux agents.

Tableau 1 Activité des plateformes d'agents mobiles conforme à la FIPA

Plateforme	***Activité/Mise à jour***
April Agent Platform	Aucune, depuis la dernière version Octobre 2002
Comtec Agent Platform	Indisponible
FIPA-OS	Aucune
Grasshopper*	Aucune, depuis novembre 2003
JACK Intelligent Agents	Maintenue (commerciale)
JADE	Maintenue
Java Agent Services (JAS)	Aucune, depuis Mai 2002
LEAP	Plus accessible
ZEUS	Aucune, depuis la dernière version Octobre 2001

*Grasshoper est conforme, également, à la spécification MASIF

I.4.1. Aglets

Les Aglets [27] sont des composants développés par une équipe de chercheurs du laboratoire de recherche d'IBM à Tokyo au début 1995, dans le but de fournir une plate-forme uniforme pour les agents mobiles dans un environnement hétérogène tel que celui de l'Internet. Ce sont des objets Java mobiles qui réagissent comme des agents mobiles. Ils peuvent se déplacer d'une machine à une autre [28]. Ainsi, un Aglet qui s'exécute sur un hôte peut stopper son exécution, se déporter vers un hôte distant et continuer cette exécution dans son nouvel environnement. L'API Aglet est une norme basée sur Java, proposée pour développer des agents mobiles. Les principaux éléments de cette plateforme sont :

- ***Aglet*** : objet mobile de Java qui visite des hôtes où les agents sont autorisés dans un réseau informatique. Un Aglet est autonome puisqu'il peut reprendre son exécution dès son arrivée à destination. Il est aussi réactif, car il peut réagir dynamiquement à des événements ou à une situation de son environnement.

- ***Proxy*** : un Proxy est un représentant d'un Aglet. Il sert de bouclier à l'Aglet contre l'accès direct à ses méthodes publiques. Le Proxy masque l'emplacement de l'Aglet. C'est-à-dire qu'il peut cacher le vrai emplacement de l'Aglet.
- ***Contexte*** : le contexte est l'environnement d'exécution de l'Aglet. Il fournit des moyens pour mettre à jour et contrôler des Aglets dans un environnement uniforme d'exécution.
- ***Hôte :*** Un hôte est une machine capable d'héberger plusieurs contextes. L'hôte est généralement un noeud dans un réseau.

La communication des Aglets est basée sur l'échange d'objets de la classe Message. Ce modèle de communication, fondé sur le langage de haut niveau KQLM, est indépendant de la localisation de l'Aglet. Il peut être asynchrone ou synchrone.

Un Aglet désirant envoyer un message doit obligatoirement passer par le Proxy du destinataire. En effet, le Proxy reste l'intermédiaire obligatoire pour tout échange.

Chaque Aglet possède un gestionnaire de messagerie qui lui permet de traiter les messages un par un et dans l'ordre de leur arrivée respective. Toutefois, cet ordre peut être changé par l'Aglet en modifiant les priorités des messages dans la file d'attente.

La messagerie asynchrone est implémentée grâce à la notion de futur objet. L'envoi d'un message asynchrone retourne un lien vers la réponse, même si cette dernière n'existe pas encore. Ce lien permet de tester l’arrivée ou non de la réponse. Cette technique permet à l'Aglet d'envoyer un message sans être obligé d'interrompre son exécution en attente de la réponse.

Les Aglets disposent d’un système de sécurité en couches. La première couche est offerte par Java et ses mécanismes de sécurité. La seconde est constituée du gestionnaire de sécurité qui permet aux utilisateurs d’implémenter leurs propres mécanismes de protection. La troisième et dernière est composée des APIs de sécurité Java permettant au programmeur d’inclure des fonctionnalités de sécurité sans leur agent.

Les Aglets offrent une très grande simplicité d'implémentation. Ceci constitue un avantage certain par rapport à d'autres plates-formes. La séparation entre l'environnement d'exécution et l'agent mobile correspond à notre besoin de mobilité. Toutefois, La localisation d'un Aglet reste une tâche qui pourrait poser des problèmes surtout dans un environnement à forte fréquence de pannes où nous désirons suivre le déplacement des agents. D'autre part, le mécanisme de sécurité Java représente certaines limites dans un

système dynamique car nous ne pouvons pas redéfinir les droits dynamiquement pour pouvoir accéder à des sources locales et exécuter des tâches. À notre avis, des mécanismes pour le recensement des Aglets comme un annuaire ou un mécanisme de pages jaunes devraient accroître l'efficacité de cette technologie.

I.4.2. JADE

JADE est un projet libre distribué par Telecom Italia Lab (TILab) sous licence LGPL, développé par le groupe CSELT[1] Telecom Italia avec la coopération de l'université de Parme (Italie). Son but est de simplifier le développement des systèmes multi agents (SMA) tout en fournissant un ensemble complet de services et d'agents conformes aux spécifications FIPA : service de nommage, service de pages jaunes, service d'analyse grammaticale et une bibliothèque d'interaction des protocoles, prête à l'emploi [29].

JADE est une implémentation notoire parce que c'est une marque déposée, résultant principalement d'une activité de recherches. Elle a passé, avec succès, les tests d'interopérabilité de la FIPA (Séoul, janvier 1999 et Londres, avril 2001). D'ailleurs, JADE a été intensivement utilisée par de nombreuses universités et dans le cadre de plusieurs projets de recherche comme TeSCHeT, PRIMO, IM@GINE IT, MicroGrids ou E-Commerce Agent Platform (E-CAP) toujours d'actualité.

La plate-forme d'agents de Jade inclut tous les composants obligatoires qui contrôlent un SMA. Ces composants sont : Agent communication Channel (ACC), Agent Management System (AMS) et Director Facilitor (DF).

La plate-forme d'agents peut être répartie sur plusieurs serveurs. Une seule application Java, et donc une seule machine virtuelle de Java (JVM), est exécutée sur chaque serveur. Chaque JVM est un conteneur d'agents qui fournit un environnement complet pour l'exécution d'agents et permet à plusieurs agents de s'exécuter en parallèle sur le même serveur.

Toute la communication entre agents est exécutée par des messages FIPA ACL (Agent Communication Language) [30]. Cette architecture offre la transmission de messages flexibles et efficaces. JADE crée et contrôle une file d'attente de messages entrants pour chaque agent. Le modèle global de communication FIPA a été mis en application. Ses

[1] Company for study, research, development and experimentation in telecomunication and information technologies. http://www.cselt.it

composants ont été clairement distingués et ils ont été entièrement intégrés : protocoles d'interaction, ACL, langues, schémas de codage, protocoles de transport...

Le mécanisme de transport fonctionne comme un caméléon. Il s'adapte à chaque situation en choisissant de manière transparente le meilleur protocole disponible. Java RMI, HTTP, et IIOP sont actuellement employés, mais la plupart des protocoles d'interaction définis par FIPA sont déjà disponibles et peuvent être instanciés.

Concrètement, un "thread" est lancé pour chaque agent, mais ces derniers doivent souvent exécuter des tâches parallèles. Avec la solution du "multithreading" offerte directement par Java, Jade supporte également la gestion des comportements coopératifs. Le "run-time" inclut également quelques fonctions complexes prêtes à l'emploi pour les tâches les plus communes dans la programmation d'agents, comme des protocoles d'interaction de FIPA. Jade offre, entre autre, un composant *JessBehaviour* qui s'intègre dans le système Java Expert System Shell (JESS) [31], un moteur de règle et un environnement de "scripting" permettant de concevoir un logiciel en Java capable de raisonner en s'appuyant sur des connaissances fournies sous forme de règles déclaratives. Cette approche offre la possibilité de créer des agents intelligents en déléguant le raisonnement à d'autres outils.

La plate-forme d'agents fournit une interface graphique utilisateur (GUI) pour la gestion à distance des agents (Remote Management Agent), surveillant le statut des agents, permettant par exemple d'arrêter et de remettre en marche des agents. Le GUI permet également de créer et de lancer un agent sur un serveur à distance, à condition qu'un conteneur d'agents fonctionne déjà. Le GUI permet de commander d'autres plates-formes d'agents à distance.

Le DF est un composant faisant office d'annuaire. C'est un service de pages jaunes permettant de mettre en relation les agents avec leurs compétences. Un agent peut enregistrer ses compétences dans le DF ou l'interroger pour connaître les compétences proposées par les autres agents. Le GUI permet d'associer ces DF afin de créer plusieurs niveaux de domaines. Le GUI peut accéder de la même manière à n'importe quel DF même s'il est situé sur une plate-forme distante d'agents non Jade.

Il existe dans Jade un certain nombre d'outils graphiques qui soutiennent la phase de correction, habituellement très complexe dans les systèmes répartis.

- **L'agent Dummy** est un outil simple et très utile pour visualiser des échanges de messages entre agents. *L'agent Dummy* facilite la validation d'un agent avant l'intégration dans le SMA et facilite le "debugage" au cas où un agent échoue.

L'interface graphique fournit un support pour éditer, composer et envoyer des messages ACL aux agents. Elle permet également de recevoir et de consulter les messages des agents, et, par la suite, de sauver ou de charger des messages sur le disque dur.

- **L'agent Sniffer** permet de suivre des messages échangés dans une plate-forme d'agents Jade. Quand l'utilisateur décide de surveiller un agent, ou un groupe d'agents, chaque message dirigé vers ou venant de cet agent, ou du groupe, est dépisté et montré dans la fenêtre de Sniffer. L'utilisateur peut alors regarder, sauvegarder et charger chaque message pour une analyse postérieure.
- **L'agent Introspector** permet de surveiller et de commander le cycle de vie d'un agent courant ainsi que ses messages échangés (provenant de la file d'attente des messages envoyés et reçus).

Le développement de Jade continue toujours. D'autres améliorations, perfectionnements, et réalisations ont été déjà projetés. D'ailleurs une nouvelle version est disponible depuis juillet 2009. Cette plate-forme correspond à nos besoins de mobilité, nous pouvons l'utiliser pour l'intégrer dans notre approche, malgré le fait que son mode de communication, basé sur des messages FIPA ACL, présente une limite. Les messages FIPA ACL peuvent transmettre des objets sérialisés, c'est-à-dire que si nous voulons transmettre une tâche à un agent mobile, il doit disposer de la classe de la tâche en question pour pouvoir la reconstituer une fois arrivée à destination. Cela réduit l'hétérogénéité de notre approche puisque les tâches sont prédéfinies, donc leurs évolutions et leurs améliorations deviennent une mission difficile.

Les messages de FIPA ACL n'étaient pas la seule raison de l'impossibilité d'utiliser la plate-forme. Il faut noter que, nos études sont basées sur la version 2 de JADE, une version qui représentait certaines limites de sécurité, publiées sur le site officiel de la plate-forme. Ces limites sont:

- Les permissions liées à la mobilité manquent toujours. Par conséquent, afin d'être sécurisée, une plate-forme JADE (basée sur la version 2) ne devrait en aucun cas supporter la mobilité des agents.
- Les informations échangées sont transférées via des canaux (SSL), mais elles ne sont pas signées.

JADE propose aussi la possibilité d'intégrer des services décrits et référencés dans un annuaire de pages jaunes. Cette description est réalisée grâce aux ontologies que l'on

trouve dans les messages et s'accompagne d'une description du protocole que doivent suivre les clients du service demandé. C'est une notion proche des "WebServices". Pour aider les concepteurs d'agents et de services, un outil de génération automatique d'ontologies est fourni.

Grâce à une conception des agents basée sur la notion de comportement, à une expressivité de la communication basée sur les ontologies et à ses outils d'aide au développement, JADE offre un environnement complet pour la réalisation d'applications réparties construites sur un ensemble d'agents mobiles. Mais son architecture centralisée et la limite apportée à la migration des agents représentent une limite à l'adaptation de JADE aux environnements dynamiques. Comme pour LIME (cf. § I.4.3), on ne peut se satisfaire d'un gestionnaire centralisé fragilisant l'intégralité du système s'il venait à disparaître. De plus, pour garantir leur autonomie vis-à-vis du système, les agents doivent pouvoir se déplacer suivant leurs besoins et même dans des plates-formes différentes.

I.4.3. LIME

LIME est un middleware orienté vers la complexité des environnements mobiles "ad hoc". Il est basé sur Java et propose une couche de coordination pour les agents en réutilisant le modèle Linda [32,33]. Ce dernier est un modèle de programmation parallèle, il est composé de deux notions spécifiques : l'espace de "tuples" et le "tuple" ; et de quatre méthodes primitives d'accès : *out(), eval(), in()* et *read()*. Avec les concepts de Linda, nous pouvons développer des applications parallèles sans considération du lieu, des données ni du moment de synchronisation des processus.

LIME n'est pas considérée comme une plate-forme, car elle ne prend pas en charge les différents éléments décrits par les normes. Elle se focalise plus sur la coordination pouvant être exploitée pour construire des applications réparties, tel un système basé sur une communauté d'agents mobiles où les agents sont considérés comme des services. Comme son nom l'indique, LIME est basée sur le modèle de synchronisation Linda qui propose une communication à travers une mémoire partagée, appelée espace de "tuples". C'est une collection de structures de données élémentaires appelées "tuples". La synchronisation a lieu lors de la mise en jeu d'un "tuple" en accès ou en modification.

Ce type de communication est découplé (communication indirecte à distance) en temps et espace. Par exemple : l'expéditeur et le destinataire n'ont pas besoin d'être connectés au même instant ni de connaître leurs emplacements respectifs. Cette caractéristique s'adapte

particulièrement à un environnement mobile où les deux composantes d'une communication migrent de façon dynamique. Cette démarche est spécifique à une communauté d'agents mobiles où les agents ne sont pas forcément connectés en même temps, ne connaissent pas leurs emplacements respectifs et ils doivent communiquer entre eux.

Cette méthode de communication est utilisée dans LIME en associant un espace de "tuples" aux agents ainsi que les mécanismes permettant la mise en commun de ces "tuples". Chaque site possède une interface d'espace de "tuple" (ITS) à laquelle pourront souscrire les agents en proposant les tuples à partager. Une ITS est l'union des tuples partagés par les agents locaux.

Les agents pourront alors se synchroniser sur les tuples appartenant à l'ITS. Une interface d'espace de "tuple" peut être assimilée à une communauté d'agents mobiles et si un agent désire communiquer avec un ou plusieurs autres agents, il propage un "tuple" dans l'espace.

Pour étendre ce mécanisme aux communications distantes, les ITS peuvent être regroupées au sein d'une Fédération d'espaces de tuples (FTS) créant ainsi une union d'ITS. Chaque FTS est gérée par un site leader, désignée par un mécanisme d'élection classique, dont le rôle est de prendre en compte les ITS entrants et sortants. Cette gestion centralisée permet de laisser la possibilité aux agents de dialoguer à distance et surtout permet de prévenir les agents de toute modification pour s'adapter afin de maintenir la configuration optimale de leur environnement.

La partie qui nous intéresse dans LIME réside dans sa possibilité à définir une synchronisation avec un mécanisme très simple et surtout de permettre aux agents de contrôler les ressources mises en jeu en choisissant les éléments qu'ils souhaitent partager. Les sites sont réduits à un support d'exécution.

Avec les environnements très dynamiques que nous envisageons, la gestion centralisée par un leader des FTS n'est pas adaptée. En effet, comment gérer la disparition du leader du FTS et, plus grave, comment garantir une synchronisation distante entre deux agents lorsque les liens de communication sont inconstants. D'ailleurs une communauté, n'a pas forcement besoin d'un leader pour fonctionner mais plutôt d'une stratégie de collaboration.

D'un autre côté, LIME permet à un utilisateur de réagir selon l'état courant du système, une contrainte, de plus, pour un système inaccessible aux défaillances qui doit réagir, également, à un événement et non seulement à son état.

I.4.4. PLANGENT

PLANGENT est une plate-forme d'agents mobiles conçue pour proposer aux agents des mécanismes pour modifier les objectifs intermédiaires nécessaires à la réalisation de leur tâche finale. PLANGENT est basée sur le langage Java s'intéressant à l'adaptation des agents durant leurs déplacements au sein des environnements dynamiques. Il est considéré comme le middleware le plus proche de l'intelligence artificielle [34].

Cette plate-forme est très ancienne et il existe très peu de documentation. Nous avons choisi de l'étudier pour appréhender sa démarche d'intelligence artificielle qui peut présenter des atouts non négligeables à notre propre approche, surtout en ce qui concerne la stratégie des déplacements des agents mobiles et peut-être la stratégie d'analyse des données collectées. Cela peut aussi être utile lors de protocole de négociation entre agents mobiles ou agent mobile et agent host [35].

Un site supportant l'architecture PLANGENT inclut un middleware agent comportant : un médium de communication, un gestionnaire d'information du site, un générateur d'agents, un gestionnaire de migration et une partie du système comprenant deux bases de données d'information : une pour l'environnement local et l'autre pour l'environnement distant. Les deux parties sont reliées entre elles et sont inter-accessibles.

Cette séparation permet de bien distinguer les éléments évoluant de ceux qui ne changent pas durant l'exécution d'une application. En effet, le middleware agent va rendre les services de base sans se soucier de l'évolution de l'environnement, en répondant aux différentes requêtes énoncées par les agents. Ces derniers, lors de la visite d'un site, peuvent consulter les deux bases de données et ainsi connaître les éventuelles modifications de l'environnement. Grâce à un planificateur automatiquement incorporé, ils peuvent alors modifier leurs parcours en fonction des nouvelles informations récoltées. Ce planificateur intégré aux agents détermine un parcours à suivre selon les buts à atteindre pour accomplir la tâche de l'agent. Ces buts correspondent principalement aux ressources nécessaires à la réalisation d'une tâche. À l'aide de sa représentation de l'environnement, un parcours est calculé à l'initialisation de l'agent. Il est ensuite modifié au cours des différentes visites opérées par l'agent durant son exécution à cause des nouvelles informations récupérées.

Ce mécanisme d'adaptation modifie uniquement le parcours initial de l'agent mais pas l'agent lui-même. Par exemple, en fin de journée, un père doit récupérer ses enfants,

acheter le pain et rentrer à la maison. Avant de quitter son travail, il établit son parcours pour éviter des grands détours. Mais si, pendant le parcours, il apprend par la radio que le trafic de son chemin initial est bouché ou si sa femme lui téléphone pour l'informer qu'elle a quitté plus tôt que prévu et récupère les enfants alors il modifie immédiatement son itinéraire afin de prendre en compte les nouvelles informations.

L'action des bases de données sur l'environnement et sur le planificateur, fait que les agents connaissent les changements de l'environnement et s'adaptent afin d'atteindre leurs buts. Cependant avec la mobilité, les liens de communication étant instables et les sites étant accessibles par intermittence, l'actualisation des bases de données, dont la méthode n'est pas précisée, semble difficile à mettre en place. Dans ce cadre, le planificateur, qui repose sur les bases de données des sites, met en route l'adaptation du parcours des agents avec des informations qui peuvent être erronées. Il serait donc intéressant de découpler le planificateur d'un composant système des sites afin de considérer uniquement les informations collectées par les agents durant leurs déplacements.

Avant de définir l'intelligence comme fonction préalable aux fonctions des agents mobiles, il faut se poser certaines questions avant de penser que les agents doivent comprendre l'utilisateur pour chercher des informations et des services d'une façon intelligente. Quelle sorte d'intelligence est prévue ? L'intelligence artificielle ou plutôt l'apprentissage ? Modifier le plan d'action d'un agent mobile selon des conditions réelles quand elles diffèrent de ceux prévus au commencement est une idée subtile. Seulement, autant que cette fonction soit effectuée par des règles paramétrables au lieu d'une implémentation d'intelligence lourde et moins fiable.

I.4.5. TACOMA

TACOMA se propose de savoir comment les agents peuvent être employés pour résoudre des problèmes traditionnellement soulevés par d'autres paradigmes, comme le modèle client/serveur. Le projet de TACOMA a été développé dans le but d'offrir différentes abstractions de haut niveau pour les agents [36]. L'accent est mis sur un découplage entre le langage et le niveau agent, semblable à celui introduit dans les normes à travers la notion de place.

La première abstraction concerne la manière de capturer l'état d'un agent. Celui-ci est matérialisé par un porte-documents (*BriefC*) associé à chaque agent, et qui est transporté lors de ses migrations. Ce porte-documents est constitué d'un ensemble de classeurs contenant des suites de données engendrées par les actions passées de l'agent et nécessaires aux futures actions. Les opérations classiques (création, suppression etc.) sont disponibles, ainsi que des actions de sérialisation pour le transport pouvant être adaptées selon le langage sous-jacent.

La deuxième abstraction concerne la méthode de communication qui s'obtient uniquement par la méthode *meet* sur un porte-documents. Lorsqu'un agent A souhaite dialoguer avec un agent *B* sur son porte-documents *BriefC-A*, il effectue l'opération *meet B* avec *BriefC-A*.

La troisième abstraction porte sur la communication indirecte déléguée. En effet, pour enrichir le mécanisme de communication, chaque site contient au moins un meuble de classeurs permettant aux agents de laisser les classeurs qu'ils ne souhaitent plus transporter et qui pourront être utilisés par les futurs visiteurs d'un site. Cette méthode permet de définir une communication indirecte et asynchrone entre des agents qui visiteront le même site à des instants différents.

Nous retenons de ce projet les deux abstractions qui permettent d'exprimer l'intégralité du système en termes d'agents et la communication indirecte déléguée. Dans un environnement où l'accessibilité accordée aux sites est limitée à cause de leurs déplacements fréquents, il est important de pouvoir mettre les agents au coeur du système et de leur permettre de communiquer grâce à un intermédiaire.

Néanmoins, nous voyons que la communication indirecte reste basée sur les sites qui possèdent, répétons-le, une accessibilité intermittente et par conséquent limitée. Pourquoi ne pas incorporer, alors, cette méthode de communication directement dans des agents ? Ils serviront, aussi, d'intermédiaires en permettant d'avoir une totale séparation entre le système et les sites supports.

Le problème crucial venant de ce type de communication différée, apparaissant lors de la construction des applications, est le contrôle de la validité des informations récupérées grâce aux intermédiaires. C'est un problème qui a déjà été abordé lors de la mise en place de mécanismes de rumeurs qui appliquent les méthodes de communication des fourmis dans le cadre de l'informatique répartie [37].

I.4.6. Autres modèles d'agents mobiles

Les précédentes descriptions des plates-formes d'agents mobiles ne sont pas exhaustives, nous avons essayé de retenir certains aspects jugés importants. Pour des contraintes de forme nous ne pouvons pas présenter la totalité des autres modèles. Néanmoins nous avons regroupé les principales caractéristiques de ces plates-formes dans le tableau récapitulatif ci-dessous (cf. Tableau 2) afin d'en avoir un aperçu rapide tout en regroupant, à titre comparatif, les différentes plates-formes que nous avons étudiées en ne donnant que quelques informations synthétiques [5,38]

Tableau 2 Tableau comparatif des plates-formes mobiles

Plate-forme	**Agents**		**Communication**		**Propriétés**		**Protocole**	**Langage**
	Type	**Mobilité**	**Mode**	**Type**	**État**	**Annuaire**		
Aglets	Proactif	Faible	Asynchrone, Synchrone	Asynchrone, Synchrone	Oui	Non	ATP, RMI, CORBA	Java
Ajanta	Proactif	Faible	Asynchrone, Synchrone	Asynchrone, Synchrone	Non	Oui	RMI	Java
D'Agents	Proactif	Forte	Asynchrone, Synchrone	Asynchrone, Synchrone	Oui	Non	Sockets	Multiple
JADE	Proactif, Réactif	Faible	Asynchrone	Asynchrone, Synchrone	Oui	Oui	RMI, http, IIOP	Java
JavAct	Proactif	Faible	Asynchrone, Synchrone	Asynchrone, Synchrone	Oui	Non	RMI, CORBA	Java
KAOS	Réactif	Faible	Asynchrone, Synchrone	Locale	Oui	Oui	Orb	Java
LIME	Supportées*		Synchrone	Asynchrone, Synchrone	Oui	Non	Sockets	Java
Odyssey	Proactif	Faible	Synchrone	Locale	Non	Oui	Rmi	Java
PLAGENT	Proactif	Faible	Synchrone	Locale	Oui	Non	Sockets	Java
TACOMA	Proactif	Faible	Les deux	Locale	Oui	Non	Sockets	Multiple

* Il supporte les différents types d'agents en fonction du système sous-jacent grâce à un médium de synchronisation

Plate-forme	Agents		Communication		Propriétés		Protocole	Langage
	Type	Mobilité	Mode	Type	État	Annuaire		
Voyager	Proactif	Faible	Asynchrone, Synchrone	Asynchrone, Synchrone	Non	Oui	Rmi	Java

I.4.7. Synthèse

Un point important, récurrent dans toutes les études faites auparavant, est que l'agent doit disposer d'un support d'exécution pour l'accueil pendant son déplacement. Cette conception n'interdit pas des mécanismes de distribution classiques avec un seul noyau fixe, mais les agents sont des composants considérés comme volatiles car ils peuvent à tout moment disparaître. Ce changement de l'environnement doit être supporté par les agents qui doivent avoir une connaissance de leur environnement afin qu'ils puissent s'adapter ou avoir un comportement spécifique.

Il faut également, gérer les différents états atteints par les agents au cours de leur exécution. Les changements d'états interviennent principalement lors de l'achèvement d'une tâche ou juste avant une migration et doivent être clairement définis lors de la conception. Ceci est parfaitement réalisable si nous proposons un modèle d'agents défini grâce à un cycle de vie référent et un ensemble de comportements/transitions.

Le déplacement des agents étant une méthode cruciale pour l'adaptabilité, un ensemble de mécanismes simples sont proposés pour sa mise en œuvre. Ces mécanismes permettent de préparer un déplacement en se séparant de données inutiles. Par exemple, migrer en donnant uniquement la destination et mettre en place son état post-migratoire. Cette migration, par expression explicite de la destination, permet à l'utilisateur de garder le contrôle des déplacements des agents. L'utilisateur peut également implémenter une migration au hasard en choisissant dans un ensemble de machines atteignables, qu'il peut obtenir depuis le service d'environnement ou d'annuaire. Dans ce cas, il s'agit d'un choix délibéré.

L'agent doit avoir le choix de déplacements nécessaires à la réalisation de leurs applications.

Les déplacements sont effectués via un service d'accueil auquel s'adressent les agents, lorsqu'ils souhaitent se déplacer vers la machine où le service s'exécute. Même dans le cas où l'agent gère de manière autonome ses déplacements, la migration ne donne pas des

droits de déplacements sans limite. Lorsque le service d'accueil reçoit une requête de migration d'un agent, il statue selon des règles établies s'il accepte ou non la demande puis envoie le verdict à l'agent. Cette méthode permet de garder un contrôle d'accès à certains sites critiques mais ne remet pas en cause la gestion des déplacements propres à l'agent. Ce dernier s'adaptera selon les verdicts du service d'accueil. Le verdict est le résultat d'un protocole de négociation [35] issu de la requête de la demande de l'agent mobile auprès du service d'accueil.

En effet, plusieurs types de migrations sont possibles portant sur la prise de décision de la migration et sur sa perception, qui seront ensuite utilisés par l'utilisateur lors de la conception des agents. Le type n'est pas figé durant le cycle de vie d'un agent. Il pourra s'adapter selon les besoins.

Les déplacements fréquents des agents ne permettent pas de mettre en place une communication distante qu'elle soit synchrone ou asynchrone. Car pour le premier mode, les communications peuvent être interrompues régulièrement. La seconde ne permet pas de garantir la livraison des messages dans des délais raisonnables. Par conséquent, la communication s'effectue uniquement entre des agents locaux afin de garantir un déroulement sans interruption. En permettant aux agents d'établir des liaisons distantes le temps de leurs déplacements.

Les agents doivent avoir la possibilité de dialoguer grâce à des agents intermédiaires pour avoir une communication plus souple. Ces agents intermédiaires peuvent être mobiles ou immobiles. Ce dialogue se fait toujours entre éléments locaux.

Une approche de communication, plus expressive, basée sur un ensemble d'ontologies permet de définir précisément la sémantique des échanges. Comme nous l'avons décrit précédemment dans la norme FIPA, l'utilisation des ontologies permet de garantir que deux agents se comprennent correctement lors des phases de dialogue.

Une plate-forme d'agents mobiles doit proposer au moins un annuaire de niveau "pages jaunes" permettant de regrouper les services atteignables localement. Ce service propose les fonctions habituelles d'enregistrement, de modification, de suppression et de recherche. Ces services sont représentés localement par un agent commis qui sera contacté par les autres agents effectuant la demande. Il est nécessaire de définir des protocoles d'interaction entre les agents présents dans les descriptions inscrites dans l'annuaire.

Pour terminer, un aspect primordial posant un inconvénient à l'utilisation des agents mobiles est le problème de la sécurité qui trouve tout son sens dans cette utilisation. Nous pouvons déjà identifier trois cibles possibles :

L'agent : l'agent qui visite un hôte pourrait être la cible d'une tentative d'extraction d'informations. Ce même scénario est valable par un agent malveillant. Enfin, les échanges d'informations entre les différents agents peuvent être interceptés.

L'hôte : un agent malveillant pourrait visiter un hôte dans le but d'accéder ou de corrompre ses fichiers. Cet agent ou entité malveillante peut envoyer ou créer un nombre important d'agents vers un hôte dans le but de le surcharger.

Le réseau : l'objectif de l'utilisation des agents mobiles est de réduire le trafic du réseau. Néanmoins, la multiplication sans fin d'agents peut encombrer et surcharger le réseau.

I.5. Surveillance des logiciels

Par définition, la surveillance est l'action de surveiller, d'observer dans une intention de contrôle ou veiller attentivement sur quelqu'un, sur quelque chose. Dans le domaine de la technologie informatique, la surveillance est souvent assimilée à la sécurité. C'est une réalité, mais la surveillance au sens large peut être considérée comme un processus de gestion des actifs, plus connu sous le nom Asset Management [39].

La sécurité informatique est l'acte de surveiller un dispositif matériel ou logiciel dans le but de le protéger de toutes actions malveillantes ou détecter un comportement anormal. Successivement nous pouvons parler d'une sécurité réactive et proactive. La première approche consiste à bloquer la charge d'attaque avant qu'elle se produise, la deuxième consiste à informer d'un événement douteux même s'il peut être jugé normal. Dans les deux cas, nous parlons des systèmes de détection d'intrusion (IDS).

L'IDS surveille l'activité d'un système d'information à la recherche d'une anomalie. Son intérêt est de pouvoir effectuer une surveillance en temps réel ou quasi réel, selon différents critères (réseau, système d'exploitation, système de fichiers etc.). Il analyse automatiquement et de façon continue les activités en cours et détecte celles qualifiées d'invalides ou d'anormales. Ces fonctionnalités engendrent l'analyse et le traitement d'une grande quantité de données, ainsi que la mise en place d'une architecture spécifique en ce qui concerne le déploiement de l'IDS : un agent doit tourner en permanence sur chaque

ressource critique du système d'information. Par ailleurs, sa capacité à appliquer les dernières mises à jour en matière de règles de sécurité (flexibilité / adaptabilité) constitue un critère de choix : la possibilité de pouvoir effectuer un tri sur les données analysées.

L'IDS ne constitue pas en lui-même une solution de sécurité complète [40], il représente une partie du système de sécurité d'un système d'information. Il ne permet pas de bloquer des attaques comme le fait un coupe-feu, mais il permet de les détecter et d'en informer le bon acteur afin qu'il puisse réagir selon la politique de sécurité définie. Ceci représente la fonction principale d'un IDS mais représente certaines contraintes en terme de charge de travail, d'autant plus que la politique de sécurité n'est pas ou plus adaptée aux besoins du système d'information observé.

La principale évolution espérée des IDS est leur capacité d'autonomie grandissante [41]. Aujourd'hui, le mieux que l'on puisse attendre d'un IDS est qu'il soit capable de tenir à jour de façon autonome ses fichiers de signatures, exactement comme le fait un antivirus. D'ailleurs, la comparaison ne s'arrête pas là, car ce fonctionnement commun se basant sur des fichiers de signatures leur amène les mêmes vulnérabilités, le delta des mises à jour et le contournement des signatures. Certains IDS sont dotés d'une intelligence pour pouvoir détecter les attaques polymorphes

Actuellement, il existe plusieurs IDS regroupés en trois familles distinctes :

- Les NIDS (Network Based Intrusion Detection System), qui surveillent l'état de la sécurité au niveau du réseau. Par exemple Snort [42] un IDS de référence qui se base sur des signatures pour analyser le trafic réseau.
- Les HIDS (HostBased Intrusion Detection System), qui surveillent l'état de la sécurité au niveau des hôtes.
- Les IDS hybrides, qui utilisent les NIDS et HIDS pour avoir des alertes plus pertinentes. Par exemple PreludeIDS le plus connu des IDS hybride

Les HIDS sont particulièrement efficaces pour déterminer si un hôte est contaminé et les NIDS permettent de surveiller l'ensemble d'un réseau contrairement à un HIDS qui est restreint à un hôte.

Il existe beaucoup de classements différents, de catégories ou bien encore de différenciations de niches des IDS. Il en est fait plusieurs constats :

- Il n'y a pas de classement normalisé et/ou exhaustif des IDS :
- Il existe de nombreuses confusions de notions et de termes. Notamment entre le déploiement et l'approche d'audit utilisée par les IDS, ou bien dans la dénomination, ou bien entre IDS réseau et NIDS.

Ci-dessous, nous adoptons le schéma récapitulatif (cf. Tableau 3) d'une proposition de classement qui, par défaut, sera la nôtre dans ce document. Sous forme de matrice, elle nous servira à présenter dans les parties suivantes les différents déploiements des IDS, ainsi que les approches et les structures.

Tableau 3 Classement des IDS

IDS	***Déploiement***		***Approche***		***Structure***	
	Hôte (HIDS)	***Réseau (NIDS)***	***Comportementale***	***Scénario***	***Monolithique***	***Agents Mobiles***
Système	X		X	X	X	
Application	X		X		X	
Réseau	X	X	X	X	X	X
Autre(s)	X*	X	X	X	X	

* Ex. : Système de fichiers, Honey pot/Honey net, contrôleur d'intégrité, …

Asset Management est une gestion des équipements informatiques et non informatiques. Dans notre étude nous ne nous intéressons qu'aux équipements informatiques (postes de travail, serveurs, équipement réseau, périphériques, logiciel…) et plus précisément les logiciels pour deux raisons : les logiciels peuvent représenter approximativement 25 % du budget d'un système d'information. En 2006 le secteur public français a dépensé presque 6 milliards euros dans les logiciels et les services informatiques, une augmentation de 10 % par rapport à 2005[2]. La deuxième raison, c'est qu'en surveillant les logiciels nous ciblons toute la gestion des équipements informatiques puisque les logiciels surveillent tous les dispositifs informatiques.

[2] Source: International Markess, February 2007

La surveillance des logiciels nous permet d'optimiser la gestion du cycle de vie des ressources informatiques pour ne déployer que ce qui est nécessaire, ni plus, ni moins. Ainsi nous maîtrisons le coût des logiciels et les données connexes pendant tout leur cycle de vie. En s'appuyant sur le coût des ressources et leur utilisation, nous concordons les ressources avec les besoins métier. La surveillance des logiciels nous permet, également, de réduire le coût des ressources informatiques en redéployant les ressources sous-utilisées tout en évitant des frais rédhibitoires de licence logicielle afin d'améliorer la qualité des interventions de maintenance et écourter la durée de résolution des incidents.

Généralement, la surveillance des logiciels s'appuie, au même titre que les IDS, sur l'analyse et le traitement d'une grande quantité de données émanant des dispositifs qu'elle surveille. C'est une démarche primordiale car il est nécessaire d'avoir un minimum d'informations de l'activité du logiciel pour pouvoir le contrôler. Ces données peuvent être disponibles sous forme d'un journal d'événements (log) ou fournies à la demande. De plus, l'analyse des journaux d'activités peut s'avèrer bénéfique par ce que l'un des meilleurs moyens de détecter les intrusions est de surveiller les journaux d'événements (Log). Il est bien avéré que lors d'une attaque informatique, le pirate ne parvient pas à compromettre un système du premier coup et qu'il agit la plupart du temps par approximation en essayant différentes requêtes [43]. Ainsi, la surveillance des journaux permet de détecter une activité suspecte. Cependant, Josué Kuri, Gonzalo Navarro, Ludovic Mé et Laurent Heye avancent qu'il est difficile de déterminer s'il y a une attaque à partir d'un log, tandis qu'il est plus facile de dire qu'il n'y a pas d'attaque à partir d'un certain volume de fichier log [44]. Nous ajoutons que l'analyse des fichiers d'activité n'est pas forcément liée à la sécurité, elle peut aussi servir à relever des anomalies ou des disfonctionnements comme, par exemple, un avertissement de l'expiration de la définition de l'anti-virus ou une défaillance d'une application qui dépend d'un service indisponible.

Actuellement, il existe différentes solutions de surveillance (libre ou commerciale) : Applications Manager une application de surveillance des systèmes, LoFiMo une application qui analyse et centralise l'affichage des journaux d'activité, Trivoli Asset Management une solution de gestion des actifs [45], Pytheas une solution de gestion des infrastructures…

L'ensemble des ses solutions opèrent de la même façon, elles collectent des données à partir de différentes ressources pour les analyser. La différence est palpable au niveau du

processus d’analyse qui varie. A ce jour, nous pouvons énumérer trois approches de collecte des donnés :

- Monolithe : un seul processus chargé de collecter localement des informations à partir de différentes sources de données puis de les centraliser et de les analyser. L’inconvénient est qu’un système d’information performant est réparti sur plusieurs machines.
- Agent : sur chaque machine un agent capable est installé pour collecter des données puis de les transmettre à un autre composant central pour qu’il les analyse. Dans la majorité des cas, ce composant est un agent. L'incommodité de cette approche est que le paramétrage des agents est prédéfini. Et souvent, les tâches de collecte sont associées aux agents. Ce qui rend leur évolution difficile.
- Requête : un seul composant qui envoie des requêtes à des applications distantes pour récupérer des événements. Sauf que nous sollicitons des services à effectuer un travail supplémentaire, ce qui risque de réduire leur disponibilité. En plus, toutes les applications ne peuvent pas communiquer des informations, ceci ne constitue pas une défaillance, mais elles préfèrent stocker leurs événements dans des fichiers d’activités pour que l’information soit totale et surtout non-volatile. Notons que l'activité de sauvegarde des événements perturbe fortement l'activité de l'application. La différence entre les deux démarches réside dans le fait que la première nous permet de réagir activement contrairement à la deuxième qui ne peut que confirmer si un événement a eu lieu.

Dans le chapitre suivant, nous décrivons une nouvelle approche qui consiste à collecter des données à l’aide des agents mobiles sur une architecture type. Ceci nous permet d’énoncer nos besoins de mobilité.

II. Thèse soutenue

Le défaut majeur des approches de collecte citées dans le chapitre précédent est la surcharge du réseau, ce qui entraîne une dégradation du système d'information qu'elles surveillent. C'est pour cette raison que nous désirons intégrer la mobilité afin de diminuer la charge du réseau et réduire le trafic alloué à la surveillance, pour pouvoir mettre en place un système de surveillance dynamique, hétérogène et robuste et enfin pour bénéficier des avantages de la mobilité (§ I.1 Les caractéristiques d'un Agent mobile). D'ailleurs sur ce dernier point des résultats ont été présentés tels que [46] attestent que des améliorations significatives d'exécution peuvent être obtenues en utilisant les agents mobiles par rapport à une approche client/serveur.

L'utilisation des agents mobiles n'est pas une première, des études précédentes présentent l'utilisation des agents mobiles : pour combler le manque d'évolution des IDS surveillant des réseaux multi sites en phase d'expansion [47]. Pour apporter un second niveau de mobilité [5], la mobilité logicielle pose un problème dès lors que l'on souhaite utiliser un service précis ou pour favoriser, dans une certaine mesure, la capacité de croissance d'un système et le passage à l'échelle [14]…

Notre approche de surveillance est basée sur le concept d'une communauté d'agents mobiles collaborant entre eux pour collecter les différents événements d'un système d'information, les analyser et agir, si nécessaire, en fonction d'une stratégie de l'analyse.

Dans ce chapitre nous allons décrire les grandes lignes, de notre démarche. Dans un premier temps, nous allons présenter un exemple type de système d'information sur lequel nous nous appuyons pour définir notre approche. Cette application type est utilisée au cours de notre approche formelle de la surveillance. Elle a de plus été le support pour évaluer les différentes implémentations d'agents mobiles que nous avons sélectionnées au chapitre I. Ensuite, nous allons décrire les besoins de notre approche de surveillance basée sur une communauté d'agents mobiles.

II.1. Application type

La surveillance logicielle recouvre plusieurs aspects que nous abordons dans ce document. Afin de disposer d'une approche cohérente, nous nous basons sur une structure type constitué d'un système d'information et d'applications représentatives réagissant par événements ou par flux de données formatées. Toutefois, ces besoins sont applicables sur n'importe quel système d'information capable de communiquer avec des composants extérieurs par l'intermédiaire d'une interface applicative (API), ou encore mieux de fournir des journaux d'activités. Ce qui est fort recommandable.

Notre exemple est composé de cinq machines (cf. Figure 1) que nous voulons surveiller. Pour chacune d'elle, nous allons collecter les journaux d'activités du système d'exploitation, en plus de collecter des informations spécifiques aux services de chaque machine.

Nous considérons cet exemple comme faisant référence à un réseau simplifié d'une petite ou moyenne entreprise (PME). En réalité, un réseau d'entreprise est composé de plusieurs machines, cette différence par rapport à notre système se situe dans le nombre des postes utilisateurs (postes clients), généralement, ils ont la même nomenclature.

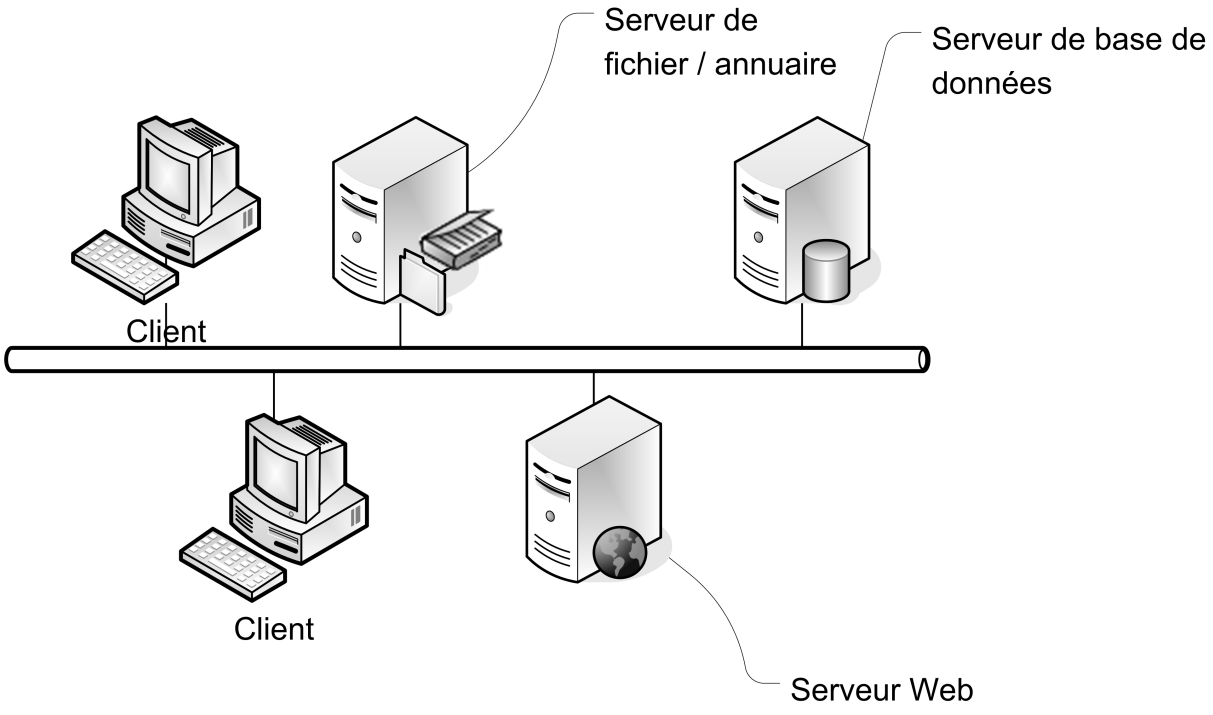

Figure 1 Architecture d'un système d'information

Pour l'architecture, nous avons choisi une architecture Windows car cette dernière représente environ 90 % de la part du marché mondial et aux alentours de 91,5 % pour les postes de travail des professionnels français (source E-soft) malgré les critiques qui fusent envers Microsoft et sa politique de sécurité. Il faut rappeler que notre approche n'est définie que pour Windows mais elle doit pouvoir s'adapter à n'importe quels autres environnements.

En ce qui concerne les différents services, nous avons opté pour des solutions libres ayant une grande pénétration du marché mondiale. Par rapport au serveur Web, nous utilisons le Serveur Apache, ce dernier représente 73,80 % des serveurs Web (source Netcraft). Quant au serveur de base de données, nous utilisons MySQL, un SGBD utilisé par de nombreuses entreprises et services comme Wikipédia, Google, Yahoo!, YouTube, Adobe, Airbus, Alstom, Crédit agricole, Linden Lab (Second Life), RATP, URSSAF, AFP, Reuters, BBC News, Leader Price, Système U, Cap Gemini, Ernst & Young, Alcatel-Lucent, etc. Récemment, Sun Microsystems a acheté MySQL.

II.1.1. Serveur WEB

Un serveur Web est un service permettant à des internautes de consulter des sites à l'aide d'un navigateur, d'utiliser des applications Web ou plus généralement recevoir une information choisie. Dans notre cas, nous avons choisi Apache HTTP Server. Communément appelé Apache, ce serveur de référence sauvegarde toutes ses activités dans un fichier Log sous un format commun (CLF). Le but n'est pas seulement de pouvoir analyser des fichiers standards mais aussi tous les journaux d'activités à condition qu'ils soient dans un format texte homogène, c'est-à-dire le même format dans tout le fichier.

La surveillance du serveur web va nous permettre, entre autres, de détecter les requêtes malveillantes issues d'un hôte pour pouvoir les bloquer à l'avenir, connaître la liste des pages introuvables pour l'envoyer au "Webmaster" et de déceler un comportement anormal. Exemple : un même utilisateur connecté à un espace privé à partir de deux machines différentes et dans un intervalle de temps très réduit.

II.1.2. Serveur de base de données

Un serveur de base de données est un ensemble structuré et organisé permettant le stockage de grandes quantités d'informations et d'en faciliter l'exploitation. Dans notre cas, nous

avons choisi le serveur de base de données MySQL afin d'en surveiller ses performances et anticiper les problèmes potentiels avant qu'ils n'aient un impact sur le système d'information. Périodiquement, des informations du serveur sont extraites afin d'en faire l'analyse telles que : la capacité des tables, le nombre des requêtes par minute/type, le nombre de connexion en cours…

Contrairement au serveur web, cette collecte va se faire par l'intermédiaire de l'API client en exécutant des requêtes SQL spécifiques au serveur en question. Par exemple :

- *SHOW STATUS* fournit toutes les informations du serveur. D'ailleurs, elles filtrent en spécifiant le nom des variables ou un modèle pour un ensemble de variables par thème.
- *SHOW TABLE STATUS [FROM db_name]* fournit toutes les informations pour chaque table de la base active ou celle fournie en paramètre.
- *SHOW PROCESSLIST* fournit tous les processus en cours ainsi que leurs statuts.

L'utilisation de l'API pour collecter les informations n'est pas indispensable. Nous devons pouvoir le faire comme pour le serveur web Apache malgré la différence du format de stockage de leurs journaux d'activités et la complexité de l'analyse de ces derniers.

D'ores et déjà, il apparaît nécessaire d'adopter le contexte des agents et d'isoler le format d'échange.

II.1.3. Serveur d'annuaire et de fichiers

Ce serveur est multifonction, il dispose, d'une part, d'un service d'annuaire regroupant les informations des utilisateurs (nom, adresse, coordonnées, etc.). Ce service est utile, entre autres, à l'authentification des utilisateurs, et s'appuie sur le protocole LDAP ayant une grande notoriété dans l'interrogation et la modification des services d'annuaire. Dans notre cas, nous utilisons Active Directory (AD) qui est la mise en œuvre par Microsoft des services d'annuaire.

D'autre part, le serveur sert à stocker l'ensemble des données de l'entreprise. Généralement, ces données sont des fichiers et répertoires qui peuvent être partagés par un ou plusieurs groupes d'utilisateurs.

Nous désirons, pour ce serveur, collecter toutes les données relatives à l'identification des utilisateurs (connexion et déconnexion) ainsi que toutes les opérations liées à l'accès aux

données (Parcours, Liste, Création, Écriture, Suppression, Autorisation et Appropriation) en s'appuyant sur l'audit du système d'exploitation.

II.1.4. Postes clients

Ce sont des postes de travail disposant d'un client léger par exemple Internet Explorer ou autre navigateur, permettant de se connecter aux différentes applications Web hébergées sur le serveur Apache. En plus, ces postes disposent d'une suite bureautique avec licence d'utilisation et des applications utilitaires pour accomplir les tâches quotidiennes. Généralement, ces postes ont la même nomenclature.

Pour les postes clients, nous allons collecter les journaux des événements Windows. D'autres informations s'avèrent utiles telles que la présence de l'utilisateur, la validité de licence, etc.

II.2. Nos besoins

Nos besoins portent sur cinq aspects essentiels, à savoir : la mobilité, l'exécution des tâches, la communication des agents, la gestion des références dynamiquement et le suivi des agents.

II.2.1. La mobilité

Notre approche s'appuie sur des agents mobiles, il est donc nécessaire de recourir à des composants ou plates-formes mobiles offrant dynamiquement la possibilité de fournir des agents, aptes à se déplacer d'une machine à l'autre et à s'adapter à leur environnement d'exécution. Leur but est de réaliser localement certaines tâches qui varient d'une machine à l'autre.

Il est vrai qu'un agent mobile est capable de se déplacer d'une machine à l'autre, mais cette opération ne peut pas se faire sans la présence d'un composant pour l'accueillir. En effet, même les virus, pour se propager discrètement, ont besoin d'un service vulnérable ou d'un programme ou encore de l'action de l'utilisateur par exemple : ouvrir un fichier infecté. Pour cela, nous déployons un service par machine. Ce service ou agent immobile a la charge d'accueillir et d'autoriser l'agent mobile dans un nouveau contexte. Cet agent local peut également aider l'agent mobile en lui fournissant des informations relatives à

l'environnement local pour qu'ils puissent exécuter ses tâches une fois qu'il se trouve sur la machine en question. Ce sera le cas pour les formats des données consultées.

II.2.2. L'exécution des tâches

La plupart du temps, le déplacement de l'agent mobile est initié par lui-même ou par un autre acteur (un autre agent mobile) dans le but de réaliser une tâche bien définie sur une machine. Dans le cadre de notre approche, l'agent mobile se déplace pour collecter les données. Le terme collecte peut paraître restrictif, il faut dire exécution d'une tâche de collecte. Car, l'agent mobile doit être capable d'accomplir d'autres tâches en réaction à un stimulus afin de réagir devant une situation particulière. Ces tâches peuvent être l'exécution d'un script, d'un programme ou encore d'une commande système. Nous pouvons, également, considérer d'autres actions réalisées par l'agent comme des tâches, par exemple: la création d'un agent mobile, la notification d'un changement, d'une modification de l'environnement… Pour que les agents puissent être autonomes l'ensemble de ces tâches doit s'exécuter d'une façon asynchrone.

II.2.2.1. Priorités des tâches

Généralement, un agent mobile peut se déplacer sur plusieurs machines et exécuter une ou plusieurs tâches par machine. En plus nous pouvons considérer que certaines tâches soient plus prioritaires que d'autres. L'agent mobile doit être indépendant et capable de redéfinir son parcours en fonction de ses priorités de façon à s'adapter à la politique des systèmes d'information. Cette dernière est souvent différente d'un système à l'autre. Nous pouvons définir une politique de manière à considérer qu'une tâche de contrôle soit prioritaire par rapport à une tâche de collecte.

Nous pouvons éventuellement faire abstraction des priorités en créant un agent par type de collecte. Mais dans le cas où deux tâches du même type doivent être exécutés, il est nécessaire de fournir des priorités si un seul agent est présent. En pratique, ce problème se pose essentiellement pour les tâches de contrôle où il est nécessaire d'en exécuter une avant l'autre. En revanche pour les tâches de collecte, l'agent doit terminer son parcours pour revenir à son hôte initial. Sauf si l'utilisateur décide que, pour certaines tâches de collecte, l'agent doit retourner et rapporter le résultat de la collecte avant de finir son tracé.

L'idée est d'utiliser une approche de programmation par contraintes (CSP) [48] en se basant sur des critères comme le type de la tâche, la catégorie du service, ou bien la priorité

de la machine et/ou la tâche… Pour mieux optimiser le chemin et dans le cas où l'agent dispose de plusieurs choix, il sera libre (autonome) de choisir son tracé.

II.2.2.2. Problématique des tâches de collecte

La collecte des données s'avère une tâche délicate surtout pour des ressources disposant d'une taille de données importante. Pour des fichiers de petite taille ou des fichiers d'activités rotatives le problème ne se pose pas. En effet, ce problème n'est pas lié à l'algorithme ou au temps nécessaire au traitement mais plutôt au transport des données, qui devient quasi impossible à partir d'une certaine taille. C'est l'inconvénient de la mobilité par rapport aux échanges basés sur des flux de communication. Nous pouvons éventuellement réduire les risques tout en gardant la mobilité afin de bénéficier de tous ses avantages en incluant la notion de filtres. L'agent mobile doit filtrer les données jugées inutiles au moment de la collecte, ce qui permet de réduire sensiblement la taille. Cette opération implique que nous devons fournir les filtres nécessaires à l'agent pour chaque tâche de collecte au moment de sa création. En plus, l'agent doit disposer d'un système de compression pour qu'il compresse les données pendant la collecte ou les décompresse à la livraison. Malheureusement, ces deux solutions ne font que repousser une limite. D'où la nécessité de découper les résultats pour faciliter leurs transports. L'agent mobile doit être capable d'évaluer la taille des données en fonction des tâches de collecte restantes ou prendre en considération un seuil prédéfini pour scinder les données puis faire plusieurs livraisons (aller-retour). Mieux, il peut faire appel à d'autres agents mobiles sur le réseau pour l'aider, ou demander la création d'un ou de plusieurs agents mobiles avec l'unique tâche de récupération des résultats.

II.2.3. La communication des agents

Un agent mobile doit être capable de communiquer avec d'autres agents mobiles dans le même réseau pour qu'ils puissent collaborer ensemble, échanger des informations nécessaires à leurs fonctionnements et se notifier entre eux. Cette communication peut se faire en s'appuyant sur un standard de langage de communication d'agents (Agent Communication Language : ACL) [30] le plus utilisé des standards est celui de la Foundation for Intelligent Physical Agents (FIPA), une organisation destinée à établir des standards afin de favoriser l'interopérabilité des applications, des services et des

équipements informatiques. L'organisation est dissoute depuis 2005 et remplacée par un comité des standards de l'IEEE.

Nous pouvons utiliser Knowledge Query and Manipulation Language (KQML) [49] un langage de haut niveau de communication entre agents. Il est indépendant de la syntaxe et de l'ontologie des messages, du mécanisme de transport et du langage de codage des messages.

Tous les deux (FIPA et KQML) sont fondés sur la théorie Acte de langage [50] avancée par Searle en 1960 et augmentée par Winograd et Flores dans les années 70. Cette théorie définit un ensemble de performatives et leur signification. Le contenu du performative n'est pas normalisé, mais varie de système à un autre.

Pour inciter des agents à se comprendre ils doivent non seulement parler la même langue, mais avoir une ontologie commune. Les messages sont des actions ou des actes communicatifs, car ils sont prévus pour effectuer une certaine action en vertu de l'envoi.

Notre but n'est pas de s'orienter vers une norme standard mais de fournir un moyen de communication pour que les agents se comprennent entre eux. Ceci dit, si c'est le cas, c'est un atout supplémentaire pour notre approche.

II.2.4. La gestion des références

Parfois, une machine ou son agent local s'arrête pour une raison inexpliquée ou inopinée, cela n'est pas dramatique lorsqu'il s'agit de l'indisponibilité d'une seule machine. Si ce sont plusieurs machines sur un tronçon de réseau important, il faut, dans ce cas-là, un moyen pour notifier l'agent mobile de l'indisponibilité des machines en question de manière à ce qu'il adapte son parcours. Le but est d'éviter un retard lié à des tests parfois inefficaces. L'idéal est d'avoir au moins un annuaire où tous les agents locaux s'enregistrent pour formuler le désir d'accueillir des agents mobiles et de se désabonner automatiquement lorsqu'ils sont indisponibles.

II.2.5. Le suivi des agents

Comme tout composant informatique, un agent mobile n'est pas épargné par un problème pondant son parcours, il peut s'arrêter brusquement sans pouvoir notifier les autres agents. Ce problème peut se produire, mais ne doit pas être fréquent, et il faudra l'anticiper pour le

gérer (proactif). D'un autre côté, le "non retour" d'un agent, censé collecter des données puis les transmettre à un autre agent initiateur de la collecte, peut être considéré comme un problème. Souvent, le temps du parcours d'un agent dépend du nombre des machines à visiter et des tâches à effectuer. Cette durée est compliquée à évaluer quand il s'agit de tâches de collecte, nécessitant un temps proportionnel à la taille des données à collecter.

La difficulté est de savoir si l'agent s'est arrêté ou s'il poursuit son parcours, d'où la nécessité d'avoir un mécanisme de suivi de l'agent qui nous permet de tracer les déplacements d'un agent et de le surveiller pour savoir s'il adhère toujours à la communauté. De plus, lorsque l'ordre des sites parcourus est un facteur de convergence, il est utile de s'assurer que l'ordre réel de traversée des sites est celui escompté.

Si, à un moment donné, l'agent mobile s'arrête, un autre doit être créé avec les mêmes paramètres pour reprendre le même état (données et paramètres) que l'ancien agent mobile et terminer son parcours à partir de la dernière étape connue. Les gestionnaires de références jouent dans ce cas le rôle de sauvegarde d'état stable.

II.3. Limites des plates formes existantes

L'enjeu de l'étude des différentes plates-formes présentées dans le premier chapitre était de déterminer celle qui est la plus appropriée à notre approche mobile et celle que nous pouvons utiliser dans la surveillance des logiciels. Seulement, aucune ne couvre totalement nos besoins, et ceci malgré leurs caractéristiques nombreuses. En revanche, cette étude comparative basée sur l'architecture type de la section précédente, nous a permis d'approfondir notre compréhension de la mobilité et avec le recul, nous avons pu aborder d'autres aspects que nous pouvons utiliser pour décrire notre modèle d'agent mobile.

II.3.1. Frameworks actuels

Le point commun entre les différentes plateformes étudiées est qu'elles sont pour la plupart basées sur l'environnement Java. Comme nous l'avons dit précédemment, le langage Java reste le plus adapté pour l'implémentation des plateformes d'agents mobiles, son seul inconvénient est la sécurité. Ce dernier est considéré comme une contrainte majeure, ce qui pourrait expliquer la non industrialisation des applications à base d'agents mobiles, cela

malgré les avantages qu'elles peuvent apporter. En effet, Java permet de gérer la sécurité uniquement au niveau local et en aucun cas les privilèges peuvent être changés dynamiquement avec l'arrivée d'un agent mobile.

D'autre part, toutes les plateformes étudiées à l'exception de la plateforme JADE ont une activité industrielle très réduite ou quasi nulle, par exemple les aglets offrent une très grande simplicité d'implémentation mais ne suscitent pas l'engouement attendu, d'ailleurs les publications concernant spécifiquement les aglets, semblent s'être arrêtées en 1998.

Un dernier point concerne la difficulté d'utilisation des plateformes d'agents mobiles : d'une manière générale, la plupart des travaux de mobilité se sont focalisés sur l'aspect technique [14] ce qui est le cas des frameworks que nous avons présenté. Ces derniers sont plus orientés outils de développement et ne disposent d'aucune approche formelle permettant de décrire une plateforme d'agent mobile précise et indépendante de toute implémentation.

Les plateformes que nous avons étudié sont avant tout faites pour des développeurs et non des concepteurs d'applications à base d'agents. De par la nature de nos travaux nous souhaitons que nos développements soient pilotés par une phase de conception rigoureuse.

Aussi, dans les points suivants nous décrivons nos besoins pour utiliser une approche formelle.

II.3.2. Besoins de piloter un projet par les spécifications

L'emploi de méthodes de spécification formelle est d'une grande importance pour le développement de systèmes logiciels et tout particulièrement pour la conception et la vérification de systèmes dits sécuritaires ou critiques (par exemple ceux impliquant la vie humaine). Le fait que les spécifications formelles permettent une description non ambiguë, précise, complète et indépendante de toute implémentation conduit à une compréhension approfondie du système à développer, et beaucoup d'erreurs sont ainsi évitées. De plus, de telles spécifications sont à l'origine des phases de vérifications et de validations, ce qui est un des points qui concourent à leur utilisation de plus en plus fréquente. Les méthodes formelles permettent aussi de dériver automatiquement des prototypes à partir des spécifications.

Un certain nombre de besoins peut être associé aux spécifications. Un formalisme de spécification offre toujours un compromis entre ces besoins qui dépendent du domaine des systèmes à spécifier. La formalisation est le fait qu'une sémantique formelle soit clairement et proprement définie pour le formalisme.

Il existe différents types de sémantique [51]. Elle pourra être opérationnelle (comme les systèmes à base d'états et de transitions), dé-notationnelle ou axiomatique. Elle peut d'autre part être exécutable [52, 53] (comme les systèmes à base de réécriture [54, 55,56]) et ainsi autoriser une animation des spécifications. Une sémantique opérationnelle nous permet de dériver une représentation proche d'un comportement sur machine.

Les propriétés de non ambiguïté et de précision sont inhérentes à l'aspect formel des spécifications (elles permettent aussi concrètement de contrôler que la spécification vérifie une ou plusieurs propriétés). Certaines notations, pourtant couramment utilisées (tel qu'UML [57]) ne présentent qu'un très faible degré de formalisation. Les possibilités de validation de ces spécifications sont limitées, voire inexistantes. Plus grave, un spécifieur n'est absolument pas assuré que celui qui lira sa spécification en comprendra le sens voulu sans avoir à lui fournir un glossaire détaillé des termes utilisés.

L'expressivité est associée aux notations (à la fois textuelles et graphiques) des spécifications qui doivent être suffisantes pour spécifier un ensemble précis du système. Différents aspects pourront être pris en compte par des notations différentes. Il n'existe pas de formalisme universel, et augmenter l'expressivité d'un formalisme risque de réduire sa lisibilité et par là même son utilisation.

L'abstraction est une propriété importante puisque des formalismes présentant un plus grand degré d'abstraction conduiront à des spécifications plus concises et plus aisées à valider. Un faible niveau d'abstraction peut ainsi conduire par exemple à une explosion du nombre d'états d'une spécification à base de système états/transitions et rendre impossible sa validation. L'abstraction aide le spécifieur à ne s'attacher qu'aux points importants par rapport à un problème ou à une propriété donnée. Cependant, l'implémentation (par exemple pour faire un prototype) d'une spécification écrite à l'aide d'un formalisme très abstrait peut présenter d'autres difficultés.

Nous avons besoin d'un langage formel qui permette de décrire les notions d'agents, de mobilité et de communication. De plus, les notions d'architectures doivent pouvoir être décrites afin de définir la localité d'un agent et la portée des données qui y sont déclarées.

La lisibilité induit la façon dont les spécifieurs pourront utiliser un formalisme avec ou sans un grand degré d'expertise. Cet aspect est souvent lié à la proximité des concepts de spécification et des concepts maîtrisés par le spécifieur (comme les concepts ensemblistes sous-jacents à Z [58] ou B [59] par exemple) ou à l'existence de notations graphiques formellement bien définies.

L'expressivité et la lisibilité sont en quelque sorte opposées : les formalismes très expressifs comme les spécifications algébriques [55], les logiques d'ordre supérieur [60, 61, 62], les algèbres de processus [63, 64] et les calculs [65] peuvent être considérés comme peu lisibles et par là même réservés à des spécialistes.

Les spécifications ne peuvent pas être totalement graphiques, et ce pour des raisons d'expressivité, mais tout formalisme devrait utiliser des notations graphiques communément acceptées (et formalisées) lorsque cela est possible. Un système dual avec une notation graphique (avec quelques composants textuels, utilisés par exemple dans les compositions) et une contrepartie textuelle associée semble être la meilleure approche. Les formalismes totalement graphiques comme UML [57] ou totalement textuels comme le calcul des systèmes communicants (CCS) [63], Communicating Sequential Processes (CSP) [64] ou π-calculus [65] ne sont pas pratiques pour différentes raisons : soit au niveau formel (une jolie notation graphique sans sémantique formelle et donc sans possibilité de vérification), soit au niveau de la lisibilité lorsqu'il s'agit de spécifications de taille importante. Certains systèmes utilisent les deux types de notations (SDL [66], Raise [67], Argos [68], CPN [69]), d'autres autorisent la dérivation d'une représentation graphique à partir de la totalité ou d'une partie d'une représentation textuelle (Estelle [70], LOTOS [71]).

Notre domaine d'intérêt est la construction d'application à base d'agents mobiles pour l'observation voire le contrôle d'autres applications telle que notre application type (cf. Figure 1). Notre besoin d'expression porte donc sur la définition d'agents, l'échange de données, la notion de site ou de localité où sont déployés des agents. Enfin pour observer une application, il faut accéder à des ressources locales à un site. Le langage de spécification doit permettre d'exprimer les droits d'accès à ces ressources. La précision des

droits peut d'ailleurs être soulignée. Ainsi, l'accès à une ressource utile à une application métier peut être à l'origine de perturbation. La spécification doit pouvoir exprimer que ces perturbations sont minimales ; cela signifie lister les opérations sur ces ressources, leur auteur et éventuellement leur format.

Bien entendu, d'autres exigences peuvent s'ajouter mais alors la lisibilité y perdait beaucoup, ainsi que d'autres opérations de synthèse.

II.3.3. Calculs d'agents mobiles

La programmation mobile et répartie est un sujet d'étude récent qui a déjà donné naissance à une multitude de calculs. Ces calculs d'agents mobiles, c'est à dire ces procédés de transformation d'expressions formelles, ont chacun leurs objectifs. Certains sont attachés aux aspects sécurité telle que SMA Calculus qui contient des primitives cryptographiques [72], d'autres utilisent une sémantique particulière associé à une notation formelle afin de se focaliser sur les aspects migratoires [73].

Le calcul des ambients [74] en est un autre exemple. D'autres, comme le Join Calcul distribué [75], se préoccupent d'une programmation répartie bien plus transparente, où la distribution n'interfère pas ou peu avec le comportement des agents.

La notion même de calcul pour la mobilité et la répartition implique immédiatement une notion de localité. Ces localités peuvent accueillir les agents, leurs frontières peuvent être variables et ne pas correspondre à des machines sur lesquelles les agents migrent et s'exécutent. Cette distinction n'est plus si claire si l'on souhaite également modéliser des machines pouvant se déplacer, que ce soit physiquement, comme des assistants personnels ou des ordinateurs portables. Ceci nous amène à considérer une seule sorte de localité et à structurer les localités de manière hiérarchique, sous forme d'un arbre. De nombreux calculs ont en effet fait ce choix. La mobilité d'un agent correspond alors à la migration d'un sous-arbre de localités.

Une deuxième notion cruciale pour les calculs distribués est la notion de communication et de synchronisation, permettant de modéliser les interactions entre agents. Cette notion se traduit habituellement soit par des rendez-vous, comme dans CCS ou le π-calcul, soit par des messages asynchrones comme dans le Join Calcul. L'envoi des messages entre agents peut se faire soit directement, soit utiliser un troisième agent qui transporte le message et le

libère à sa destination, comme dans le calcul des ambients. Ce choix est stratégique et a une influence directe sur les implémentations possibles.

Dans tous les cas, un des aspects principaux de la conception d'un calcul d'agents mobiles est la spécification de l'interaction entre localité d'une part et migration et communication de l'autre.

II.3.4. Conclusion

Le concept de localité présenté dans le calcul des ambients souffre de plusieurs limitations et notamment, ne permet pas de modéliser, ou alors de manière détournée, des aspects liés aux pannes, à la reconfiguration et à la protection. Ainsi, nous voulons pouvoir exprimer le fait qu'une localité peut disparaître et éventuellement modéliser différents types de défaillance. Concernant la protection, on désire pouvoir exercer un contrôle fin des interactions entre deux localités, à la manière de Winskel [76, 77], afin par exemple d'exécuter du code potentiellement non fiable dans un environnement sécurisé.

Finalement, on veut pouvoir reconfigurer la hiérarchie de localités du système de manière simple. Par reconfigurer, on entend détruire, stopper, réactiver, déplacer et dupliquer des localités. Une reconfiguration peut représenter aussi bien une mise à jour de logiciel, qu'une migration de processus ou encore le déploiement d'application. La reconfiguration est par ailleurs fondamentale pour la programmation par composant [78, 79].

C'est pour ces raisons que nous préférons au calcul des ambients, le π-calcul. Il exprime la mobilité en tant qu'un changement des chemins de communications, tandis que le calcul ambient, a des concepts explicites d'endroit. D'autant plus que le π-calcul est utilisé au sein de notre équipe de recherche, ce qui nous permet de garder une continuité et une cohérence dans l'ensemble de nos travaux. Nous disposons également d'un outil HOPiTool [80] offrant la possibilité de validation des systèmes mobiles d'agents conçus à l'aide de Higher Order π-calculus.

III. Approche formelle de la mobilité

Milner et d'autres auteurs [65, 81] ont proposé le π-calcul comme un langage formel pour spécifier et analyser des processus mobiles en 1989. Jugeant par la quantité et la qualité du travail dans le secteur des calculs de processus mobiles depuis lors, nous pouvons avancer que le π-calcul est devenu la manière établie pour décrire formellement le parallélisme, l'interaction et la mobilité. Dans le π-calcul, les processus se synchronisent et échangent des noms ; ces noms peuvent être employés comme liens pour la transmission d'autres noms, permettant le reconfiguration de processus dynamique. Comme Milner l'avait cité [82], la mobilité des processus est déterminée par les liens disponibles entre eux.

Dans ce chapitre nous décrivons π-calcul puis nous présentons formellement l'architecture logicielle de notre système d'agent mobile. Dans un deuxième temps nous décrivons Distributed π-calcul, une extension de π-calcul afin de présenter formellement l'architecture matérielle de notre système d'agents mobiles.

III.1. π-calcul

Le π-calcul est un modèle mathématique de processus. La notation et la définition de ce modèle a été introduite par Milner [63] puis proposée par Parrow [83]. Le π-calcul vise à définir que tout terme bien formé dénote un processus. L'abstraction fondamentale est qu'on ne s'intéresse au comportement d'un processus qu'à travers un certain nombre de points d'interaction également appelés canaux. La synchronisation et la communication sont alors exprimées par des lois de composition internes et externes. Il faut noter que tant en π-calcul qu'en CCS, l'ancêtre de π-calcul, nous ne faisons aucune hypothèse sur les vitesses d'exécution relatives des différents processus, qui sont donc présumés progresser chacun à leur rythme.

Il y a plusieurs extensions de π-calcul. Trois d'entre elles sont : le monadic π-calcul est la forme de base et permet seulement de transmettre un seul nom de canal par message. Le polyadic π-calcul est une extension du monadic permettant la transmission d'un tuple de

noms de canal par message simple. Le higher-order π-calcul permet la transmission des agents dans un message, et il peut modéliser ainsi la mobilité plus directement.

III.1.1. Le monadic π-calcul

Nous considérons :

- un ensemble infini de noms *N*, notés typiquement $a, b, c, \ldots, x, y, z$ qui fonctionneront tous comme des portes de communications, des variables et des données.
- un ensemble de termes, nous les nommerons également processus ou agent, noté typiquement A, chacun avec une arité non négatif fixe.

Les agents, notés typiquement $P, Q, R, \ldots$ sont définis dans le tableau suivant (cf. Tableau 4). Selon ce dernier, nous listons les différents comportements de l'agent :

1. Le préfixe d'émission $\bar{a}x.P$ peut transmettre le nom x via le nom a et continue comme P
2. le préfixe de réception $a(x).P$ peut recevoir n'importe quel nom via a et continue comme P où le nom reçu remplacera x. Par exemple, $a(x).\bar{b}x.0$ peut recevoir n'importe quel nom via a, transmettre le nom reçu via b et devient inactive tandis que $a(x).\bar{x}b.0$ peut recevoir n'importe quel nom via a, transmettre b via le nom reçu et devient inactive.
3. le préfixe silencieux $\tau.P$ représente un agent qui peut changer en P sans interaction visible avec l'environnement.
4. L'agent 0 n'effectue aucune action, c'est-à-dire qu'il n'interagit pas avec son environnement.
5. La somme $P + Q$ représente un agent qui peut contenir P ou Q. Par exemple, $a(x).\bar{x}y.0 + \bar{b}z.0$ dispose de deux possibilités : pour recevoir un nom par l'intermédiaire de a, et envoyer z par l'intermédiaire du b. Si la première possibilité est exercée et u est le nom reçu par l'intermédiaire de a, alors la suite est $\bar{u}y.0$; la possibilité pour envoyer z par l'intermédiaire de b est perdue. Si, la deuxième possibilité est déclenchée,

alors la suite est 0, et la possibilité de recevoir par l'intermédiaire de a est perdue.

6. La composition parallèle $P|Q$ représente le comportement combiné de P et de Q exécutés en parallèle, où P et Q peuvent procéder indépendamment et agir l'un sur l'autre par l'intermédiaire des noms partagés. Par exemple, $\left(a(x).\bar{x}y.0 + \bar{b}z\right)|\bar{a}u.0$ a quatre possibilités : pour recevoir un nom par l'intermédiaire de a, envoyer z par l'intermédiaire de b, envoyer u par l'intermédiaire de a, et évoluer invisiblement comme un effet d'une interaction entre ses composants par l'intermédiaire du nom partagé a.
7. L'égalité $[x = y]P$ peut évoluer comme P si x et y sont le même nom, et peut faire rien autrement. Par exemple, $a(x).[x = y]\bar{x}z.0$ peut recevoir un nom par l'intermédiaire de a, puis envoyer z par l'intermédiaire de ce nom juste si ce nom est y ; s'il n'est pas, le processus ne peut plus rien faire, c.-à-d. il est bloqué.
8. La restriction $(vx).P$ se comporte comme P mais la portée du nom x est limitée à P. Les composants ne peuvent utiliser x pour agir l'un sur l'autre entre eux mais pas avec d'autres processus. Par exemple, $(vx)\left(\left(a(x).\bar{x}y.0 + \bar{b}z.0\right) \mid \bar{a}u.0\right)$ a seulement deux possibilités : pour envoyer z par l'intermédiaire de b, et évoluer invisiblement comme effet d'une interaction entre ses composants par l'intermédiaire de a. La portée d'une restriction peut changer en raison de l'interaction entre les agents.
9. La réplication $!P$ peut être considérée comme une composition infinie $P \mid P \mid \ldots$ ou, d'une manière équivalente, un agent satisfaisant l'équation $!P = P \mid !P = !P \mid P$. La réplication de P est l'opérateur qui permet d'exprimer des comportements infinis. Par exemple, $!a(x).!\bar{b}y.0$ peut recevoir des noms par l'intermédiaire de a à plusieurs reprises, et peut envoyer par l'intermédiaire de b n'importe quel nom qu'il reçoit.
10. L'identifiant $A(y_1, \ldots, y_n)$, où n est l'arité de A, a une définition $A(x_1, \ldots, x_n) \stackrel{def}{\equiv} P$ où x_i doit être par paires distinctes, et l'intuition est que

$A(y_1,\ldots,y_n)$ se comporte comme P avec y_i remplaçant x_i pour chaque i. Une définition peut être considérée comme une déclaration d'agent, $x_1,\ldots,x_n$ en tant que paramètres formels, et la marque $A(y_1,\ldots,y_n)$ comme invocation avec les paramètres effectifs $y_1,\ldots,y_n$.

Tableau 4 La syntaxe de π-calcul

Prefixes	$\alpha ::=$	$\bar{\alpha}x$	Output
		$a(x)$	Input
		τ	Silent
Agents	$P ::=$	0	Nil
		$\alpha.P$	Prefix
		$P + P$	Sum
		$P \mid P$	Parallel
		$[x = y]P$	Match
		$(\nu x)P$	Restriction
		$!P$	Replication
		$A(y_1,\ldots,y_n)$	Identifier
Definitions		$A(x_1,\ldots,x_n)$	$(where\ i \neq j \Rightarrow x_i \neq x_j)$

Définitions :

Liaison [84] : Il y a trois opérateurs pour l'usage de variables liées

- le préfixe de réception $a(x)$
- la restriction $(\nu x)P$
- L'identifiant $A(y_1,\ldots,y_n)$

En π-calcul, une variable est liée à un ν. Une variable liée a une portée et cette portée est locale ; de plus on peut renommer une variable sans changer la signification globale de l'expression entière de l'agent où elle figure. Une occurrence d'un nom dans un agent est libre si elle n'est pas liée.

Définissons les noms libres $fn(P)$ et les noms liés $bn(P)$ d'un agent. Cette définition est fonction de la structure des termes :

$$bn(a(x)) = \{x\} \qquad fn(a(x)) = \{a\}$$

$$bn(\bar{a}x) = 0 \qquad fn(\bar{a}x) = \{a, x\}$$

Par exemple, une application de cette définition.

$$fn\left(\left(\bar{a}x.0+\bar{b}y.0\right)\middle|\ \bar{c}u.0\right)=\{a,x,b,y,c,u\}$$

Et

$$fn\left(\nu a\left(\left(a(x).\bar{x}y.0+\bar{b}u.0\right)\middle|\ \nu u\bar{a}u.0\right)\right)=\{y,b,u\}$$

Substitution [84] : une substitution est une fonction des noms vers d'autres. On écrit x/y pour la substitution qui remplace y par x et tous les autres noms. En général $\{x_1\ldots x_n/y_1\ldots y_n\}$ là où y_i est par paire distinct, pour une fonction qui remplace chaque y_i par x_i .

Par exemple :

$\bar{u}a\mid u(x).P(x)$ avec $P(y)=\bar{u}y.0$ après la communication et la substitution le processus P devient $P(a)=\bar{a}y.0$

Sémantique opérationnelle : une sémantique opérationnelle est très utile pour calculer une évolution d'un terme. La sémantique opérationnelle de π-calcul est donnée par un système de transition étiqueté, là où les transitions sont du type $P\xrightarrow{\alpha}Q$ pour la somme d'un ensemble d'actions notés par α .

Il y a trois notations pour les transitions : le τ l'étape silencieuse, l'action de réception $a(x)$ et d'émission $\bar{a}x$;

L'action de réception : $a(x).P\xrightarrow{\bar{a}(u)}P\{u/x\}$ signifie que si le nom u est transmis par le canal a alors le processus $a(x).P$ attend pour un nom de processus ou de canal sur a , le reçoit et effectue la substitution de x par u , il se comporte alors comme P , où toutes les occurrences de x sont remplacées par u .

Les règles de communication sont les plus spécifiques pour un système de mobilité. L'interaction entre deux processus est donnée par les règles Com_1 et Com_2 de communication [84]:

$$Com_1:\ \frac{P\xrightarrow{\bar{a}x}P',Q\xrightarrow{a(y)}Q'}{P\mid Q\xrightarrow{\tau}P'\mid Q'\{x/y\}}$$

$$Com_2 : \quad \frac{P \xrightarrow{a(y)} P', Q \xrightarrow{\bar{a}x} Q'}{P \mid Q \xrightarrow{\tau} P'\{x/y\} \mid Q'}$$

Une action d'émission permet à P de devenir P' depuis P, l'action correspondante de réception fait que Q devient Q' puis parallèlement P et Q deviennent P' et Q' en parallèle.

La notation d'une sémantique de transition est donnée dans le tableau ci-dessous :

Tableau 5 La sémantique opérationnelle.

STRUCT	$\frac{P' \equiv P, P \xrightarrow{\alpha} Q, Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}$
PREFIX	$\overline{\alpha.P \xrightarrow{\alpha} P}$
SUM	$\frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$
MATCH	$\frac{P \xrightarrow{\alpha} P'}{[x = x] P \xrightarrow{\alpha} P}$
PAR	$\frac{P \xrightarrow{\alpha} P', bn(\alpha) \cap fn(Q) = \phi}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$
COM	$\frac{P \xrightarrow{a(y)} P', Q \xrightarrow{\bar{a}x} Q'}{P \mid Q \xrightarrow{\tau} P'\{x/y\} \mid Q'}$
RES	$\frac{P \xrightarrow{\alpha} P', x \notin \alpha}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}$

Pour une explication détaillée des règles de la sémantique opérationnelle, il est possible de se rapporter à [83].

Ce langage formel contient la base des notations qui nous permettent de spécifier notre architecture type d'agents mobiles. Ainsi un serveur d'agents mobiles fournit des agents et les configure dans le but d'effectuer une prochaine mission. Pour cette configuration d'un agent mobile, nous décrivons un terme *Configurateur* qui fournit à un agent mobile le nom des sites à visiter via un transfert d'information sur un canal *c* commun aux termes

Configurateur et *MobileAgent.* (Terme qui correspond à l'agent mobile, défini ultérieurement)

$$Configurateur(c) \triangleq (vs_1, s_2, s_3)\,\bar{c}s_1.\bar{c}s_2.\bar{c}s_3.....0$$

Il apparaît clairement que cette notation n'est pas assez riche pour écrire une spécification réelle. Un seul nom peut être émis à la fois et si plusieurs informations concernant un même site, elles doivent être émises en séquence. Des sous séquences de communication doivent être considérée comme atomique. Des tuples d'informations doivent être émis en une communication. Aussi, nous avons choisi une extension naturelle du monadic appelé le polyadic π-calcul.

III.1.2. Le polyadic π-calcul

Nous décrivons le π-calcul polyadique synchrone du premier ordre [85]. Les notations sont le prolongement naturel du tableau précédant (cf. Tableau 4). La syntaxe retenue est la suivante. Elle est plus riche que la précédente.

$$P ::= 0 \mid \rho.P \mid P|P \mid P + P \mid [x = y]P \mid [x \neq y]P \mid (v\vec{u})P \mid A(\vec{u})$$

Où les préfixes ρ ou actions sont définis de la manière suivante :

$$\rho ::= \tau \mid \alpha(\vec{u}) \mid \bar{\alpha}\langle\vec{u}\rangle$$

et où l'abstraction de processus doit faire l'objet de déclarations de la forme :

$$A(\vec{u}) \overset{def}{=} P$$

$\vec{u}$ dénotant une liste, éventuellement vide, de variables également appelées noms. Nous donnons à la suite une sémantique informelle du polyadic π-calcul :

- Le processus 0 ne fait rien, c'est-à-dire qu'il n'interagit pas avec son environnement. Il dénote la terminaison normale d'un processus mais aussi des cas plus pathologiques comme celui d'un processus dont l'évolution interne est bloquée. Enfin, 0 étant la seule constante de base de l'algèbre, elle permet la construction, de façon inductive, des autres processus.
- Le processus $\rho.P$ fait intervenir l'opérateur de préfixage : il s'agit d'une loi de composition externe impliquant une action ρ et un processus P. Cet opérateur est le seul qui exprime la séquentialité en π-calcul. Ainsi, $\rho.P$ exécute l'action ρ puis

se comporte comme P. L'interaction avec l'environnement : $\alpha(\vec{a})$ ou $\overline{\alpha}\langle\vec{a}\rangle$ qui expriment respectivement la réception et l'émission d'une liste de variables sur le canal α. D'un point de vue strictement temporel, une interaction est semblable à un mécanisme de type rendez-vous tel qu'il a été défini en CSP [64] : un processus qui entend exécuter une action ne peut le faire que s'il existe au même moment une contrepartie dans l'environnement. Nous entendons par là une offre d'interaction duale sur le même canal, c'est-à-dire une émission dans le cas d'une réception et réciproquement.

- $P|Q$ dénote un couple de processus qui s'exécutent en parallèle. La signification est semblable à celle du monadic π-calcul même si la notation des communications a évoluée. Ainsi, une synchronisation se produit et si la liste des noms en commun n'est pas vide, celle-ci est unidirectionnelle. Ainsi, dans le cas où : $\overline{\alpha}\langle\vec{a}_i\rangle.P|\alpha(\vec{b}_i) \xrightarrow{\tau} P|Q\{\vec{a}_i/\vec{b}_i\}$ on substitue à chaque occurrence libre du marqueur de place b_i dans le processus récepteur la variable correspondante a_i. La transition est étiquetée par τ dans la mesure où on considère qu'il s'agit d'un opération interne à l'ensemble des deux processus. De plus, le terme : $\overline{\alpha}\langle\beta\rangle.P|\alpha(a).\overline{a}\langle x\rangle.Q$ se réduit-il, en l'absence d'autres processus dans l'environnement, à : $P|\overline{\beta}\langle x\rangle.Q\{\beta/a\}$ Cette propriété est appelée mobilité : son objectif est d'autoriser la reconfiguration dynamique de la topologie des systèmes.
- Le processus $P+Q$ dénote le choix indéterministe entre les processus P et Q. Ces derniers ne peuvent être exécutés tous les deux. Ils leur sont notamment impossibles d'interagir l'un avec l'autre. Le choix de celui qui est exécuté est effectué en fonction des offres d'interaction présentes dans l'environnement. En l'absence d'opérateur de "matching", il s'agit donc d'un choix externe, c'est-à-dire effectué par l'environnement. Par exemple, en l'absence d'autres processus dans l'environnement et si α, β et γ sont tous différents, le processus : $\alpha(a_1, a_2).P + \overline{\beta}\langle b\rangle.Q \mid \beta(c).R + \gamma(d).S$ ne peut évoluer que vers : $Q \mid R\{b/c\}$ si plusieurs possibilités d'interaction sont simultanément éligibles, le choix est effectué de façon indéterministe.

- L'opérateur de matching permet de réaliser l'exécution conditionnelle d'un processus. Ainsi, si $a = b$, $[a = b]P$ se comporte comme P, sinon il est bloqué, c'est-à-dire qu'il se comporte comme 0. L'opérateur de "mismatching" a une sémantique symétrique : si $a \neq b$, $[a \neq b]P$ se comporte comme P, sinon il est bloqué. La combinaison de ces opérateurs avec le choix indéterministe permet de réaliser des choix internes équivalents à des énoncés conditionnels.
- $(v\vec{a})P$ dénote un processus dans lequel les variables de la liste $\vec{a}$ ont une portée limitée à P, c'est-à-dire qu'elles sont inconnues dans les autres processus : on parle alors de variables privées. Une des conséquences de cette situation est que P ne peut communiquer avec l'environnement en utilisant ces canaux. Si l'un d'entre eux est émis sur un autre canal, sa portée est étendue au processus récepteur : ce phénomène est désigné sous le terme de "scope extrusion". Les éventuels conflits d'homonymie entre des variables privées exportées et des variables locales sont résolus par le biais d'alpha conversions. Par exemple : $(va)(\overline{\beta}\langle a\rangle.P) \mid \beta(x).\overline{a}\langle x\rangle.Q$ requiert une alpha conversion avant l'exportation de a de façon à préserver la différence entre les variables a appartenant à chacun des processus définis de part et d'autre de l'opérateur "|". On effectue donc l'alpha conversion (a' est obligatoirement un nom libre) préalablement à l'émission : $(va')(\overline{\beta}\langle a'\rangle.P\{a'/a\}) \mid \beta(x).\overline{a}\langle x\rangle.Q$ puis la réduction de terme avec extension de la portée de a' : $(va')(P\{a'/a\} \mid \overline{a}\langle a'\rangle.Q\{a'/x\})$
- la définition d'une abstraction de processus $A(\vec{u}) \stackrel{def}{=} P$ lie un identifiant de processus A et une liste de variables $\vec{u}$ à un comportement de processus P. Par substitution de paramètres effectifs à ses paramètres formels, on crée un processus instance de P. Ce mécanisme de définition d'abstractions autorise de plus la récursivité et permet donc de créer des comportements de processus infinis. Il est également le seul outil dont on dispose en π-calcul pour constituer des structures de données.

Grâce à cette expressivité accrue notre terme *Configurateur* peut avoir une définition plus riche où pour chaque site une action à faire peut être fournie. Ainsi l'agent mobile configuré saura quoi faire sur chacun des sites visités.

$$Configurateur(c) \stackrel{\Delta}{=} (\nu\ s_1, s_2, s_3, a_1, a_2, a_3)\ \ \bar{c}\langle a_1, s_1\rangle.$$
$$\bar{c}\langle a_2, s_2\rangle.$$
$$\bar{c}\langle a_3, s_3\rangle.$$
$$....$$
$$0$$

De ce fait des connaissances sont transmises à l'agent mobile. La portée des noms $\vec{a}$ et $\vec{s}$ a donc évoluée pour être connues de l'agent mobile seul : scope extension

Cette expression est toutefois limitée et n'autorise pas d'actions elles mêmes paramétrées. Par exemple, si la configuration de l'agent mobile a pour objectif de numéroter les sites parcourus de façon incrémentale, le polyadic π-calcul oblige la spécification à préparer sa numération de manière extensive lors de la définition du terme *Configurateur(c)*. Ainsi le site s_1 sera désigné comme étant le numéro 1 avant même le début du parcours. Il pourrait être plus adapté que le numéro du site soit évalué lors du parcours effectif. De ce fait, si un site d'accueil n'est pas disponible ou défaillant, la numérotation pourra se faire en conservant une séquence continue. Cette expressivité est possible pour l'utilisateur d'une extension naturelle du polyadic π-calcul, nommé : Higher order π-calcul.

III.1.3. Le higher-order π-calcul

Le higher-order π-calcul (HOπ) est une extension de π-calcul du premier ordre, introduit par D.Sangiorgi [86]. Ce calcul enrichit le π-calcul avec une communication évoluée explicite. Le Hoπ permet non seulement d'émettre des noms, mais également des termes d'ordre supérieur. La syntaxe de Hoπ-Calcul est une extension de la syntaxe de π-calcul du premier ordre soit : $a, b, \ldots, x, y, z$ un ensemble de nom et $P, Q, \ldots$ un ensemble de processus.

La syntaxe de HOπ-Calcul est une extension de la syntaxe π-calcul [65,81] :

$$P ::= \bar{x}\langle K\rangle.P \mid x(U).P \mid P|Q \mid \tau.P \mid \nu x P \mid P+Q \mid [x=y]P \mid [x\neq y]P \mid !P \mid 0$$

Avec K est un agent ou nom, U est une variable ou nom et :

- 0 est le processus qui n'interagit pas avec son environnement. Il dénote la terminaison normale d'un processus mais aussi des cas plus pathologiques comme celui d'un processus dont l'évolution interne est bloquée.

- $\bar{x}\langle K\rangle.P$ peut transmettre un nom ou un processus K via le nom x et continue comme P.
- $x(U).P$ peut recevoir n'importe quel nom ou variable U et continue comme P où le nom reçu se substitue à U.
- La composition $P|Q$, les deux composants peuvent procéder indépendamment et interagir par l'intermédiaire de noms et variables partagées, comme dans le cas du polyadique π-calcul.
- $\tau.P$ est une action invisible de P. Elle peut être une action interne ou processus interne.
- νxP est la restriction, elle signifie que la portée du nom x est limitée à P, de façon semblable au polyadic π-calcul.
- la somme $P+Q$, les processus P ou Q peuvent agir l'un ou l'autre avec d'autres processus.
- L'égalité $[a=b].P$ dénote l'activation d'un processus qui est choisi par d'autres processus en fonction de la condition $([a=b])$
- La différence $[a\neq b].P$ permet de compléter la définition précédente.

Les extensions de la sémantique opérationnelle sont fournies par Saingiorgi [86] :

$$ComHO_1: \quad \frac{P\xrightarrow{\bar{x}K}P', Q\xrightarrow{x(U)}Q'}{P|Q\xrightarrow{\tau}P'|Q'\{K/U\}}$$

$$ComHO_2: \quad \frac{P\xrightarrow{x(U)}P', Q\xrightarrow{\bar{x}K}Q'}{P|Q\xrightarrow{\tau}P'\{K/U\}|Q'}$$

Pour les deux règles de la sémantique opérationnelle il est nécessaire de noter que le nombre de paramètres de U est identique à celle de K [87]: $Arity_{A,P}(U)=Arity_{A,P}(K)$ avec A est un ensemble d'actions $x,y,\ldots$ et P est un ensemble d'agents $P,Q,\ldots$. Cette contrainte peut être enrichie par un contrôle des signatures entre celle attendue par U et celle proposée par K. L'ajout de ce type de contrôle [88] nous permet d'employer la surcharge dans la définition d'agent

III.2. Architecture logicielle d'un système d'agent mobile

Au regard de ce que nous avons mentionné précédemment et en s'appuyant sur notre application type (cf. § II.1). Nous déployons un agent local nommé *AgentHost* par machine, cet agent assurera la migration d'un agent mobile et aussi lui accordera l'accès aux ressources locales afin de lui garantir l'exécution des tâches. Nous déployons aussi un serveur d'agents mobiles (cf. Figure 2). Ce service est considéré comme le cœur de notre système d'agent mobile (SAM). Ce service est composé de deux agents que nous nommerons *AgentServer* et *AgentDirectory*. Ces deux agents sont agents mobiles, c'est-à-dire qu'ils sont capables de se déplacer d'un serveur vers l'autre afin d'assurer une disponibilité élevée dans le cas d'une défaillance matérielle. Aussi, ces deux composants peuvent résider sur deux machines distinctes.

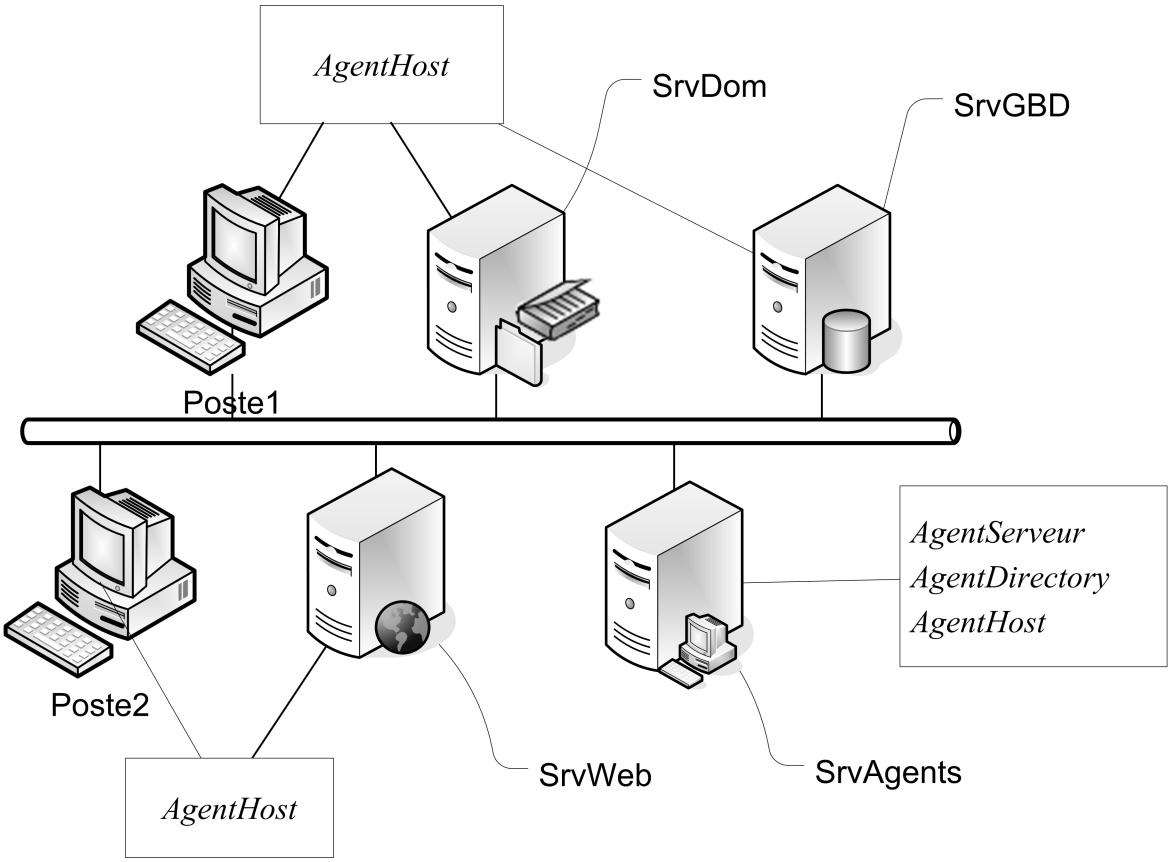

Figure 2 Architecture de l'application type

L'*AgentServer* garantit la gestion des agents mobiles (création, migration…) et analyse leurs résultats de collecte. L'*AgentDirectory* gère les références des différents agents du

SAM. Il peut être considéré comme un annuaire qui supporte les opérations de recherche, de publication et de retrait.

La figure 2 présente aussi les noms des machines de notre architecture tels que *Poste1*, *SrvDom*, etc. dont une explication est donnée dans le chapitre précédent. Les noms des agents figurent encadrés avec un lien sur la localité initiale.

III.2.1. Création d'un agent mobile

A la création, un agent mobile possède une ou plusieurs tâches à effectuer. Ces dernières sont des traitements à réaliser sur un site d'arrivée de l'agent mobile.

Dans la suite, Site est l'ensemble de tous les sites où un agent mobile peut se déplacer par exemple : $s_1, s_2, s_3, \ldots$ Une tâche est un couple traitement, site.

Un traitement est une séquence d'actions à réaliser sur un site. Dans la suite, Action est l'ensemble de toutes les actions effectuées par un agent mobile : par exemple : $a_1, a_2, a_3, \ldots$

Notre architecture type présente six machines (ou sites) susceptibles de recevoir un agent, nous considérons dans la suite des agents mobiles capables de faire une mission où une action est réalisée sur chaque site dans un ordre à choisir.

Lors de la création d'un agent mobile, l'*AgentServer* crée deux processus :

1. l'agent mobile,
2. le configurateur : celui-ci est chargé de fournir la mission à faire c'est-à-dire un ensemble de traitements.

Dans notre spécification, cette information est émise à destination de l'agent mobile créé et de lui seul. Elle peut être lue ou extraite depuis une source de type base de données ou un autre forme de présentation.

$$Configurateur(c) \stackrel{\Delta}{=} (\nu a_1, a_2, a_3, a_4, a_5, s_1, s_2, s_3, s_4, s_5, s_0, register)$$
$$\bar{c}\langle a_1, s_1 \rangle .$$
$$\bar{c}\langle a_2, s_2 \rangle .$$
$$\bar{c}\langle a_3, s_3 \rangle .$$
$$\bar{c}\langle a_4, s_4 \rangle .$$
$$\bar{c}\langle a_5, s_5 \rangle .$$
$$\bar{c}\langle registre, s_0 \rangle .$$
$$0$$

Équation 1 Configurateur

Au niveau du serveur d'agents :

$$AgentServer(register, unregister) \stackrel{\Delta}{=} (\nu\ init_1, aGroup, execute, request)$$
$$($$
$$Configurateur(init_1) \mid$$
$$MobileAgent(aGroup, init_1, execute, register, request)$$
$$)$$

Équation 2 AgentServer

Au niveau de l'agent mobile : il reçoit l'ensemble de sa mission et le mémorise, c'est l'étape d'initialisation comme le montre la figure suivante (cf. Figure 3). La dernière action *register* signifie que l'agent mobile devra terminer sa configuration pour s'enregistrer auprès de *l'AgentDirectory* sur le site s_0.

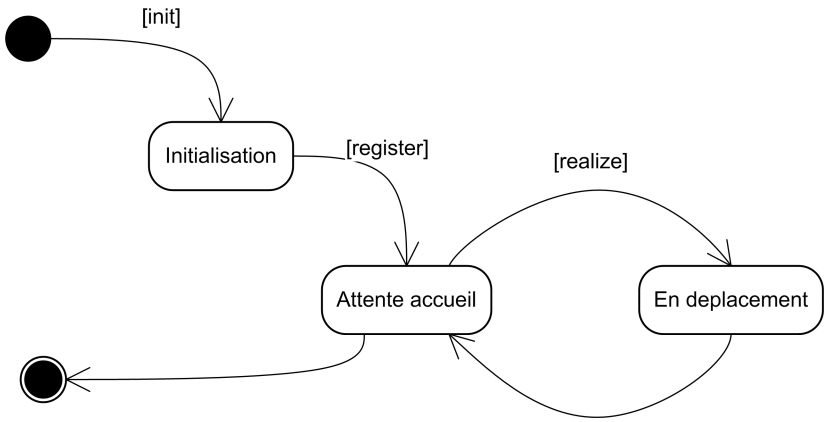

Figure 3 Les étapes de l'agent mobile

$$
\begin{aligned}
&MobileAgent(aGroup, init, execute, register, request) \triangleq (\nu\ input, reset, channel, inputIdle, resetIdle)\\
&\quad (\\
&\quad\ (\ init(a,s).\\
&\quad\quad (\\
&\quad\quad\ [a = registre]\,\overline{register}\langle aGroup, MobileAgent, execute\rangle .\\
&\quad\quad\ Realizer(init, aGroup, execute, request) +\\
&\quad\quad\ (\\
&\quad\quad\quad Mission(input, reset, channel, inputIdle) \mid\\
&\quad\quad\quad Scheduler(input, resetIdle, channel, inputIdle) \mid\\
&\quad\quad\quad \overline{input}\langle a, s\rangle\\
&\quad\quad\ )\\
&\quad\quad)\\
&\quad\ )\\
&\quad \mid execute(ch).\overline{channel}\langle ch\rangle\\
&\quad).\\
&\quad MobileAgent(aGroup, init, execute, register, request)
\end{aligned}
$$

Équation 3 MobileAgent

Le premier membre de la composition parallèle sert à la réception de la configuration sur *init*.

A la réception d'une action *a* sur le site *s* l'agent mobile crée deux processus, l'un pour conserver les actions dans le but de les effectuer lors de la première demande (c'est le processus *Mission*) et un second processus si l'ensemble des actions est à effectuer à plusieurs reprises : c'est le *Scheduler*.

$$
\begin{aligned}
Mission(input, reset, channel, inputIdle) \triangleq (&input(a,s).channel(ch).\overline{ch}\langle a, s\rangle.\overline{inputIdle}\langle a, s\rangle\\
&Mission(input, reset, channel, inputIdle)\\
&)\\
&+ reset.0
\end{aligned}
$$

Équation 4 Mission

$$Scheduler(input, resetIdle, channel, inputIdle) \stackrel{\Delta}{=} (inputIdle(a,s).channel(ch).\overline{ch}\langle a,s\rangle.\overline{input}(a,s) . Scheduler(input, resetIdle, channel, inputIdle)) + resetIdle.0$$

Équation 5 Scheduler

Lorsque l'agent est configuré, il s'enregistre dans l'annuaire (*AgentDirectory*) via le canal *register* et passe dans l'état "Attente accueil" (cf. Figure 3) Il est donc prêt à effectuer sa mission. Celle-ci débutera avec l'évaluation du terme *Realizer*.

De ce fait notre système au niveau le plus haut peut se décrire comme suit :

$$System \stackrel{\Delta}{=} \nu(register, unregister, request, response) \; AgentServer(register, unregister) \mid AgentDirectory(register, unregister, request, response)$$

Équation 6 System

Il faut spécifier l'*AgentDirectory* qui permet de conserver les références des agents mobiles créés par le serveur d'agent mais aussi celles des agents d'accueil. L'ensemble des références d'agents mobiles ou d'agents d'accueils doit être accessible pour pouvoir être utilisé dans le système : c'est le rôle de l'*AgentDirectory*. L'*AgentDirectory* enregistre les références via le canal *register* de portée globale au système.

$$
\begin{aligned}
&AgentDirectory(register, unregister, request, response) \triangleq (\nu\ save, reset, echange, saveNext, resetIdle)\\
&\quad (\\
&\quad\quad register(groupe, agent, port).\\
&\quad\quad (\\
&\quad\quad\quad (\\
&\quad\quad\quad\quad AgentMemory(save, reset, exchange, saveNext) \mid\\
&\quad\quad\quad\quad IdleAgentMemory(save, resetIdle, echange, saveNext) \mid\\
&\quad\quad\quad\quad \overline{save}\langle groupe, agent, port\rangle.\overline{response}\\
&\quad\quad\quad)\\
&\quad\quad\quad + unregister(groupe, agent, port).reset.\overline{resetIdle}.\overline{response}\\
&\quad\quad)\\
&\quad\quad \mid\\
&\quad\quad request(group, agent, ch).\overline{exchange}\langle groupe, agent, ch\rangle\\
&\quad).\\
&AgentDirectory(register, unregister, request, response)
\end{aligned}
$$

Équation 7 AgentDirectory

L'enregistrement d'un agent mobile ou d'un agent host dans l'annuaire, signifie la fourniture d'un triplet d'information *groupe, agent* et *port* qui représentent :

- *groupe* est le nom de la communauté à laquelle appartient l'agent
- *agent* est le nom de l'agent lui-même
- *port* représente le port de communication de l'agent. C'est ce nom qui sera utile pour le déclenchement de sa mission.

L'annuaire décrit précédemment n'est pas nécessairement unique, même si cela est le cas dans notre exemple. Cet agent dispose d'une mémoire pour conserver les références (terme *AgentMemory*). Ce terme n'est connu que d'*AgentDirectory* afin qu'aucune perturbation ne puisse avoir lieu et ainsi influer les missions des agents en cours.

Lorsque l'enregistrement est effectué, le terme *Realizer* permet le lancement de l'agent. Ce terme est défini plus loin pour des raisons de notation.

$$AgentMemory(save, reset, exchange, saveNext) \stackrel{\Delta}{=}$$
$$(\ save(group, agent, port).$$
$$exchange(gr, ag, ch)[group = gr\ AND\ agent = ag]\overline{ch}\langle group, agent, port\rangle.$$
$$\overline{saveNext}\langle group, agent, port\rangle.$$
$$AgentMemory(save, reset, exchange, saveNext)$$
$$+ reset.0\ )$$

$$IdleAgentMemory(save, resetIdle, exchange, saveNext) \stackrel{\Delta}{=}$$
$$(\ saveNext(group, agent, port).$$
$$exchange(gr, ag, ch)[group = gr\ AND\ agent = ag]\overline{ch}\langle group, agent, port\rangle\ .$$
$$\overline{save}\langle group, agent, port\rangle.$$
$$IdleAgentMemory(save, resetIdle, exchange, saveNext)$$
$$+ resetIdle.0\ )$$

Équation 8 AgentMemory

Ces deux termes permettent de conserver les références et de pouvoir les fournir lors d'une demande sur le canal *exchange*.

Si plusieurs *AgentDirectory*, sont présents dans le système, il est capital d'assurer leur cohérence. La composition parallèle du terme *AgentDirecotry* doit lors être enrichie afin d'ajouter une opérande dédiée à la synchronisation de référence. Le but est de ne pas avoir deux références différentes pour un même agent.

La création d'agents nécessite de pouvoir recueillir les agents en fin de mission. C'est aussi le rôle de l'agent *AgentServer*.

III.2.2. Migration d'un agent mobile

Lorsque la configuration est terminée, l'agent s'est enregistré auprès de l'annuaire, il peut donc commencer sa mission et se déplacer sur le premier site. Ceci ne peut se faire que si l'agent *AgentHost* s'est lui-même enregistré dans l'*AgentDirectory*. Il faut commencer par la spécification d'un *AgentHost*.

La figure 2 présente six agents d'accueil qui ont lors de leur enregistrement un comportement similaire. Au niveau le plus haut, notre système évolue, il s'écrit désormais comme suit :

$$\begin{aligned} System \stackrel{\Delta}{=} & \nu\,(registre, unregistre) \\ & \mid SrvAgents(registre, unregistre) \\ & \mid Poste_1(registre, unregistre) \\ & \mid Poste_2(registre, unregistre) \\ & \mid SrvDom(registre, unregistre) \\ & \mid SrvWeb(registre, unregistre) \\ & \mid SrvSGBD(registre, unregistre) \end{aligned}$$

Équation 9 System version 2

Avec

$$\begin{aligned} & SrvAgents(registre, unregistre) \stackrel{\Delta}{=} \\ & \quad \mid AgentServer(registre, unregistre) \\ & \quad \mid AgentDirectory(registre, unregistre, request, response) \\ & \quad \mid AgentHost(registre, unregistre) \end{aligned}$$

Équation 10 SrvAgents

Les noms pris pour ces nouveaux agents correspondent aux machines que nous avons choisies (cf. Figure 2). Enfin, la définition de chacun d'eux comprend le lancement d'au moins un *AgentHost*. Cette expression devient plus complexe car cela notation ne permet pas de faire la séparation entre site et agent.

Chacun des six agents d'accueil débute par un appel à l'annuaire *AgentDirectory* par exemple pour $Poste_1$:

$$\overline{register}\langle userGroup, Poste_1, P1\rangle$$

où *P1* est le port de communication de $Poste_1$ et *register* et le canal permettant de faire le lien avec *AgentDirectory*.

Il est nécessaire de faire évoluer la spécification de l'agent mobile afin qu'il débute sa mission après son enregistrement. Cela signifie, accéder au premier traitement à effectuer, rechercher le site d'accueil dans l'annuaire (*AgentDirectory*) pour connaître le port

d'écoute de ce site. Puis, il suffit d'utiliser ce port pour effectuer une migration ou émission de l'agent à destination de l'agent d'accueil (*AgentHost*) de ce site.

$$[a = registre]\overline{register}\langle aGroup, MobileAgent, execute\rangle.Realizer(init, aGroup, execute, request)$$

$$\begin{aligned} Realizer(init, gr, execute, request) \triangleq & (\nu\ work)\,\overline{execute}\langle work\rangle \\ & .work(a, s) \\ & .\overline{request}\langle gr, s, resp\rangle \\ & .resp(group, ag, port) \\ & .\overline{port}\langle MobileAgent(group, int, execute, registre, request)\rangle \\ & .0 \end{aligned}$$

Équation 11Realizer

Le terme *Realizer* définit en premier un canal local afin de recevoir sur le canal *work* le traitement à faire et le lieu de migration. La communication sur *request* permet de connaître des informations sur le site, en particulier son port d'écoute. Celui-ci a été fourni à *AgentDirectory* lors de l'enregistrement de *AgentHost,* auprès de l'annuaire.

Les fonctionnalités principales de l'*AgentHost* sont :

1. S'enregistrer auprès *d'AgentDirectory*
2. Réceptionner un agent mobile et le lancer après avoir mis à jour sa localité sur l'*AgentDirectory*.

$$\begin{aligned} AgentHost(chan, register, unregister) \triangleq & (\nu\ port, work)\,chan(g, name). \\ & \overline{register}\langle g, name, port\rangle. \\ & !\,port(MA(gr, init, execute, reg, req)). \\ & \overline{reg}\langle gr, MA, execute\rangle \end{aligned}$$

Équation 12 AgentHost

La réception sur le port nommé *port* impose une signature sur l'agent reçu mais n'exige aucune contrainte sur les aptitudes de cet agent. Seule, une analyse de la définition de l'agent le permettrait. Le nom *reg* est le canal qui permet à l'agent de s'enregistrer. Il dépend nécessairement du serveur qui l'a crée. Cette définition *d'AgentHost* montre que

lors de la phase de déploiement sur chaque site, il est utile de configurer cet agent d'accueil. Prenons comme exemple le site nommé *SrvWeb* (cf. Figure 2).

$$SrvWeb(register, unregister) \triangleq (\nu\ chan)(Init(chan) \mid AgentHost(chan, register, unregister))$$

Équation 13 SrvWeb

Le terme *Init* correspond à la configuration locale de l'*AgentHost*. Cela signifie la fourniture d'information utile pour localiser le terme *AgentHost* avant son étape d'enregistrement auprès *d'AgentDirectory*.

$$Init(chan) \triangleq (\nu\ gn, Srvweb)\overline{chan}\langle gn, SrvWeb\rangle.0$$

Équation 14 Init

Une telle déclaration s'applique aisément au cinq autres définitions de serveur : *SrvDom*, *SrvSGBD*, *SrvAgents*, $Poste_1$, $Poste_2$. Celle-ci permet de fournir un nom à chaque *AgentHost* dépendant du site, ainsi que le nom de la communication à la quelle il appartient.

III.2.3. Exécution d'un tâche

Une tâche est considérée comme définie par un couple (traitement, site). Celui-ci décrit les actions faites par un agent mobile après son arrivée via un *AgentHost*. Pour un site donné, un traitement consiste en une séquence d'actions, chacune d'elle est décrite en HOπ-Calcul. Ainsi, les actions sont fréquemment des accès aux ressources locales au site (fichier, capteur, ressource matérielle, etc.). D'où un premier problème de gestion de droit et d'autorisation. Il apparaît évident qu'un contrôle de ces actions doit être effectué à l'arrivée de l'agent mobile par l'*AgentHost.* Ce besoin d'expression est difficilement comblé par Hoπ-Calcul car cela amène à mélanger des spécifications métier avec le contrôle d'autorisation. Dans un premier temps, nous allons considérer ce contrôle comme étant réalisé avec succès. Enfin de section, nous abordons les moyens d'exprimer ce contrôle.

Ainsi, pour effectuer un traitement comportant la lecture d'un fichier local, nous procédons de la manière suivante :

La gestion de l'accès en lecture du fichier est faite par un agent

$$FileReader(read) \stackrel{\Delta}{=} (\nu\, u)\, \overline{read}\langle u\rangle.\tau.FileReader(read)$$

Équation 15 FileReader

Dans cet exemple le flot de données est considéré comme non fini, l'action invisible τ représente l'accès interne à la ressource.

Le canal *read* est considéré comme local au site. Pour accéder à ces données l'agent mobile va exécuter un traitement comprenant en particulier des lectures sur le canal *read*. Cette action doit impérativement se réaliser sur le site où se trouve l'agent *FileReader*.

La définition du terme *SrvWeb* évolue afin de prendre en compte la gestion des ressources locales.

$$\begin{aligned} SrvWeb(register, unregister) \stackrel{\Delta}{=} (\nu\, chan, read) \\ (\; Init(chan) \mid \\ \quad AgentHost(chan, register, unregister) \mid \\ \quad FileReader(read) \\ ) \end{aligned}$$

Équation 16 SrvWeb version 2

Le terme *FileReader* joue le rôle de gestionnaire d'une ressource : l'accès à un fichier via *read*. Si d'autres ressources entre en jeu, il faut enrichir la définition de *SrvWeb* par la mise en parallèle d'autres termes.

Soit la tâche

$$FileReaderTask(cRead) = cRead(value).\tau.FileReaderTask(cRead)$$

Ce terme correspond au traitement fourni par *Configurateur* sur le site de gestion des agents mobiles : *SrvAgents.* Le terme *FileReaderTask* est la séquence d'actions que devra réalisées l'agent mobile sur le site *SrvWeb*.

$$\begin{aligned} Configurateur(c) \stackrel{\Delta}{=} (\nu\, SrvWeb, SrvAgents, read)\, \bar{c}\langle FileReaderTask(read), SrvWeb\rangle . \\ \bar{c}\langle registre, SrvAgents\rangle . \\ 0 \end{aligned}$$

Équation 17 Configurateur version 2

Le *Configurateur* attribue cette tâche à l'agent mobile. Celle-ci sera exécutée à l'arrivée de l'agent mobile via *l'AgentHost* du site considéré.

De même il faut faire évoluer la spécification de *l'AgentHost* pour lancer la tâche importée par l'agent mobile

$$
\begin{aligned}
AgentHost(chan, register, unregister) \triangleq (\nu\ port, work)\big(chan(g, name).\overline{register}\langle g, name, port\rangle\big). \\
!(\ port(MA(gr, init, execute, reg, req)). \\
\overline{reg}\langle gr, MA, execute\rangle . \\
(\\
MA(gr, init, execute, reg, req) \mid \\
\overline{execute}\langle work\rangle . work(a, s).[s = name]a \\
) \\
)
\end{aligned}
$$

Équation 18 AgentHost version 2

A l'analyse de ce travail, il apparaît que l'ensemble les notions d'agents de sites et de ressources est considéré de manière identique. La notion de localité absente du HOπ-Calcul oblige à construire des spécifications denses où toutes les notations sont interdépendantes.

Pour améliorer en lisibilité et en évolutivité nos spécifications, il est utile que nous puissions exprimer qu'un canal est local à un site donné. De plus, il est important de contrôler les besoins d'un agent sur un site tel que $Poste_1$. Si cet agent souhaite accéder à un canal non disponible sur $Poste_1$ alors il n'est pas utile d'autoriser son arrivée. Ainsi, l'enregistrement d'un agent dans l'annuaire devient plus riche. Son nom et son groupe ne sont plus suffisants. Il faut y ajouter ses besoins en terme de ressources accordées. De même un *AgentHost* doit fournir ses aptitudes. Ainsi la mobilité sera davantage contrôlée. Un agent mobile pourra migrer si ses besoins peuvent être satisfaits sur le site d'accueil. Ce besoin d'expressivité est comblé par plusieurs extensions de HOπ-Calcul. Les travaux de Matthew Hennessy ont montré une compatibilité avec les travaux existants de HOπ-Calcul. Nous avons choisi ce calcul pour la description d'architecture physique de notre application type. Notre objectif est désormais de disposer de deux niveaux de spécification : une première écrite en HOπ-Calcul pour la description logicielle et une seconde en Dpi-Calcul pour la description matérielle.

III.3. DpiCalcul

Les architectures évolutives sont répandues dans de nombreux domaines d'activités, depuis les applications Web au calcul numérique. La gestion de telle architecture au cours de son évolution est en revanche moins souvent abordée. Cette gestion est pourtant indispensable dans l'écriture de systèmes auto adaptatifs [89] ou auto correcteurs [90].

Cette volonté de séparer la description de l'architecture n'est pas nouvelle et des langages dédiés existent tels que Architecture Description Language (ADL) [91]. Une description ADL permet ainsi de schématiser une vue globale de l'application de la même manière qu'un plan d'architecte modélise une maison en décrivant les différentes pièces de celle-ci et la manière dont elles sont disposées et communiquent entre elles. Mais le langage de notre choix devait disposer d'un pouvoir d'expression suffisamment riche pour que les perturbations dues à son emploi se justifient en particulier sur la notion d'accès aux ressources locales où la signature d'une opération ne peut être suffisante.

III.3.1. Description d'architecture

L'approche formelle, pour la description d'architecture dynamique, impose la spécification de la structure précise de l'architecture du système ainsi que les possibilités de reconfigurations.

Ces approches formelles peuvent se classer en quatre catégories : celles qui se basent sur un ou plusieurs graphes, celles qui sont fondées sur une logique dédiée, celles qui ont pour base une algèbre de processus ou encore des approches exotiques.

L'utilisation de graphes et des grammaires de graphes semble naturelle pour spécifier une architecture dynamique. De nombreux travaux portent d'ailleurs sur l'emploi de règles de réécriture de graphes pour spécifier les reconfigurations possibles au sein d'une architecture logicielle [92, 93, 94] ou encore la définition de transformations en extensions [95].

D'autres travaux ajoutent à la définition du graphe initial une spécification algébrique pour chacune des reconfigurations envisagées [96]. En revanche, toutes ces évolutions sont définies de manière statique et limitent ainsi l'intérêt de ces approches.

Des approches basées sur des logiques permettent de construire des preuves par rapport aux propriétés de reconfiguration d'architectures. Dans ce contexte, le résultat de

l'architecture après reconfiguration n'est pas défini mais c'est à l'évolution de cette architecture que l'on s'intéresse avec la recherche d'évolutions compatibles [97, 98, 99].

D'autres approches se démarquent par leur aspect plus opérationnel. Ainsi, le projet C2SADEL est basé sur le langage ADL qui permet de construire un graphe implicite où chaque nœud définit un ensemble de contraintes par rapport à ses voisins directs [90, 100].

Les algèbres de processus sont plus communément utilisées pour l'analyse de systèmes répartis. La notion de base est le processus, spécifié dans une algèbre (CCS, CSP ou autre) et le calcul mis en place pour faire des preuves de propriétés. L'emploi de π-calcul pour la description d'architecture a déjà été validé avec le projet LEDA [82,101], mais l'emploi de polyadique pi calcul est insuffisant pour énoncer toutes les propriétés de localité nécessaires à la description d'un cas réel. Des extensions ont même été définies à CCS afin de pouvoir s'adapter à ce domaine, c'est le cas de l'approche PiLar [102]. De nombreux calculs pour la mobilité ont vu le jour ces dernières années : Ambient Calcul, Kell Calcul [103], mais une des premières approches réellement abouties pour des descriptions matérielles est sans contexte Dpi calcul de M. Hennessy suivi d'une évolution telle que le langage SafeDi.

Les besoins de description étant croissants, des langages plus riches sont apparus avec en premier Dpi pour Distributed π-calcul [104,105]. Si la syntaxe nécessite un investissement important (des ensembles de noms sont présupposés connus, etc), l'ensemble des besoins architecturaux peut être rigoureusement décrit. Depuis la notion de site, jusqu'à la notion de port local ou externe, toute l'architecture initiale d'un système évolutif peut s'exprimer. Un système est composé d'un ensemble de localités sans hiérarchie spatiale et chaque localité peut contenir un nombre arbitraire d'agents. Ces agents reprennent la syntaxe du pi calcul polyadic classique auquel s'ajoute l'action *goto* qui permet de faire migrer un processus. De plus, contrairement aux calculs de sites mobiles, si $site_1[P]$ signifie que le site $site_1$ contient l'agent P, ce terme n'interdit pas à $site_1$ de contenir aussi d'autres agents. Ainsi, les termes $site_1[P] \mid site_1[Q]$ et les termes $site_1[P \mid Q]$ sont équivalents. Une des conséquences de ce point de vue est qu'un nom ne peut désigner qu'une seule et unique localité : tous les agents P qui apparaissent comme $site_1[P]$ sont composés en parallèle dans la même localité $site_1$.

Des études ont d'ailleurs été menées pour les besoins propres aux agents mobiles [106] ; d'autres sur la reconfiguration de grille de calcul, dans le but de partager des ressources de calcul [107].

L'utilisation d'une extension du pi calcul d'ordre supérieur, pour les descriptions de bas niveau, est une aide importante pour l'étude des comportements des agents. De plus, l'auteur des spécifications n'est pas confronté à deux langages de spécification totalement différents mais uniquement à un langage : le pi calcul, et à une extension.

III.3.2. Spécification d'une architecture type

Afin de décrire les aspects matériels ou bas niveau, nous nous sommes orientés vers l'emploi du langage Dpi calcul conçu par M. Hennessy [108]. Cette fois, la notion de site de migration est prise en compte en ajoutant la notion d'aptitude ou d'autorisation. Il est alors possible d'écrire des expressions sur les besoins de migration pour satisfaire une mission, autrement dit effectuer un certain parcours sur un réseau. Poursuivons notre précédente étude (§ III.2) où un agent mobile doit parcourir une séquence de sites afin de leur fournir un numéro unique de parcours. La figure architecture de l'application type (cf. Figure 2) montre les machines prises en compte avec leur nom de site. *SRVAgents* est le site où l'agent *AgentServeur* crée un agent mobile et le publie via *AgentDirecotry*. L'ensemble de nos descriptions précédentes nous amène à un premier placement :

$$SrvAgents\left[AgentServer \mid AgentDirectory \mid AgentHost\right]$$

AgentHost est un agent qui permet entre autre l'accueil des agents mobiles de retour de mission.

La description d'un algorithme de numérotation de sites particuliers, afin de passer seulement une fois par chacun des nœuds physiques, peut s'exprimer à l'aide de ce calcul. En revanche, l'activité précise entre l'agent *AgentServer* et l'agent *AgentDirectory*, ou plutôt le métier propre de l'agent *AgentHost,* sera spécifié à l'aide du HOπ-calcul.

Comme extension du langage pi calcul, Dpi calcul repose sur une notion de localités, ou de sites, de sorte que tous les processus sont annotés par la localité dans laquelle ils s'exécutent :

$Poste_1[AgentHost]$: signifie que l'agent *AgentHost* est physiquement placé sur le site nommé $Poste_1$.

$Poste_1[AgentHost_1] \,||\, SrvAgents[AgentServer \mid AgentDirectory \mid AgentHost_0]$: au niveau d'un réseau plus complexe, il est possible d'exprimer la mise en parallèle des processus localisés.

Dpi est un langage d'ordre supérieur, basé sur le π-calcul, qui permet d'exprimer le comportement des agents mobiles qui se déplacent entre les sites d'un réseau réparti. Si la syntaxe de ce langage est plus complexe, utilisée conjointement avec une spécification de type polyadic π-calcul, ce langage permet une description des scénarii de déplacements des agents sur un réseau de sites de réception d'agents.

À chaque instant, un des processus d'un système peut réaliser une étape de calcul interne ou bien interagir : une interaction met en jeu deux processus qui décident de se synchroniser et d'échanger un message. Pour avoir lieu, cette communication suppose qu'il existe un canal sur lequel un des deux processus émet un message tandis que l'autre processus écoute.

Puisque la synchronisation entre deux processus est une opération complexe, Dpi calcul impose que les communications soient locales, c'est-à-dire que les canaux sur lesquels elles s'effectuent soient eux aussi localisés et ne soient accessibles qu'aux processus s'exécutant dans leur localité. Ainsi, dans l'expression suivante :

$$SrvAgents[AgentHost_0] \,||\, (new\ a\ C)(Poste_1[AgentHost_1] \,||\, SrvAgents[AgentServer \mid AgentDirectory])$$

Les agents nommés *AgentServer*, *AgentDirectory* et $AgentHost_0$ s'exécutent sur le site nommé *SrvAgents*, alors que l'agent $AgentHost_1$ s'exécute sur le site nommé $Poste_1$. Les agents $AgentHost_1$, *AgentServer* et *AgentDirectory* partagent le même canal logique privé *a* de type *C*. Dans cet exemple, *AgentServer*, *AgentDirectory*, $AgentHost_1$ et $AgentHost_0$ sont des agents traditionnels comme l'exprime Robin Milner dans ses travaux de présentation de π-calcul d'ordre supérieur. Ils peuvent recevoir des informations et en émettre sur des canaux. Le type de ces canaux indique la nature des valeurs véhiculées.

Cette approche est totalement dynamique et un agent peut décider de migrer sur un site déjà connu ou nouvellement pris en compte. Ainsi :

$$Poste_2\left[\left(new\ Poste_1 : Site\right)\ with\ MobileAgent\ in\ a!\left\langle Poste_2\right\rangle \mid e!\left\langle Poste_2\right\rangle\right]$$

Cette expression décrit qu'à partir d'un site, nommé $Poste_2$, un nouveau site, nommé $Poste_1$ est pris en compte. Le code de l'agent, nommé *MobileAgent*, est placé sur ce nouveau site et le nom de ce site est communiqué sur les canaux *a* et *e*. Dans cette expression, les canaux *a* et *e* sont déclarés à un niveau supérieur autorisant leur utilisation dans cette expression.

Dans une telle spécification, il est important de déclarer initialement tous les sites connus avec les services qui y sont disponibles, afin de ne gérer les reconfigurations que lorsque cela est indispensable. De manière générale, cette déclaration de services correspond au type du site considéré telle que l'expression suivante :

$$Poste_1 = loc\left[c : w\left\langle MobileAgent\right\rangle,\ f : r\left\langle AgentHost\right\rangle, b : w\left\langle AgentHost\right\rangle\right]$$

L'identificateur *Poste₁* correspond à un site physique où trois services sont disponibles.

L'émission sur un canal *c* est considérée comme un service utilisé par le *MobileAgent*, ainsi que la réception sur le canal *f* pour l'*AgentHost*. De plus, le service offert sur le canal *b* permet la remontée d'informations à destination depuis l'agent *AgentHost* vers un autre agent présent sur *Poste₁*. Cette propriété peut être affinée en précisant la nature des données émises. De telles contraintes de types peuvent parfois aboutir à des incohérences.

De manière identique aux sites, les canaux entre sites supportent un typage qui permet de limiter l'expressivité du message échangé. Ainsi, supposons la définition du canal *c* depuis *Poste₁* conjointement à la précédente définition du type du site *Poste₁*:

$$c : w\left\langle loc\left[f : r\left\langle AgentHost\right\rangle\right]\right\rangle$$

Tout d'abord cette expression est cohérente avec la déclaration du type de *Poste₁*, où le canal *c* était annoncé comme étant utilisé en émission par le *MobileAgent*. Ce qui signifie, que lorsque cet agent aura émigré depuis *Poste₂* vers *Poste₁* il pourra disposer de ce service. De plus, cette déclaration ajoute que la nature de la donnée transportée sur *c* doit disposer d'un service *f* en lecture. Dans l'expression précédente, le receveur de l'information est mentionné, ce sera l'*AgentHost*. Une telle expression joue le rôle de

contrainte sur le message émis. Dans notre cas, nous pouvons, à titre d'exemple, utiliser le canal *c* depuis *Poste₁* de la manière suivante :

$$c!\langle Poste_1 \rangle$$

Dans cette expression, les agents receveurs de l'information *Poste₁* depuis le canal *c*, doivent bien entendu être aptes, sur leur site à recevoir sur un canal f (site où une déclaration duale de *c* a été faite). Enfin, cette information reçue leur permet d'utiliser le service f de *Poste₁*.

Une autre notation porte sur la migration d'agents en fonction des sites à parcourir. Par exemple, lorsque le *MobileAgent* souhaite se déplacer depuis *Poste₂* vers *Poste₁*, l'expression initiale de ce déplacement mentionne le canal support de ce déplacement $(goto_f)$. L'évaluation d'une telle expression revient à utiliser le service f de *Poste₁*, avec comme message l'agent en cours de déplacement. Cette approche est reprise dans les travaux de M. Hennessy sur une évolution de Dpi calcul nommé SafeDPi [107].

$$Poste_2[goto_f\ Poste_1.MobileAgent] \mid Poste_1[AgentHost] \rightarrow Poste_1[f!\langle MobileAgent \rangle \mid AgentHost]$$

Un agent doit désigner un canal f vers sa destination (ici le site nommé *Poste₁*) pour effectuer sa migration. Comme le montre la règle d'évaluation ci-dessus, cette expression revient alors à évaluer l'agent sur le site de destination. Un agent qui souhaite migrer ne pourra s'évaluer tant que le site de destination n'acceptera pas cette communication. Cela signifie que le site receveur doit supporter un terme, nommé ici *AgentHost*, dont l'évaluation permet d'aboutir à :

$$Poste_1[f?(x).x.AgentHost]$$

Ce qui signifie que le site, nommé *Poste₁*, peut contrôler les arrivées des agents mobiles qui souhaitent lui rendre visite. Dans cette description, la répartition d'agent est ainsi dirigée par l'emploi d'une communication d'ordre supérieur. En outre, les canaux utilisés peuvent être typés afin de contrôler davantage les importations d'agents. Le typage des canaux est complété par la définition même des agents émigrants.

Quand un agent migre, il peut être instancié à la réception grâce à des valeurs de type adéquat. Ceci procure davantage de contrôle et assure ainsi une meilleure gestion des agents. Ce cas d'instanciation, dès la réception par le site receveur, est d'ailleurs le plus fréquent, mais une facette de cet agent n'est pas encore prise en compte : les ressources locales utilisées.

III.3.3. Définition de type d'élément architectural

Pour limiter les accès aux ressources locales par des agents mobiles, la notion d'agent typé est introduite. Celle-ci permet de décrire les besoins des agents au niveau local, c'est-à-dire par rapport au site qui l'accueille. Ces ressources incluent aussi les canaux utilisés par l'agent à partir de son arrivée sur le site [107]. Une expression de typage d'agent peut s'écrire sous la forme :

$$AgentType[f : MobileAgentType@Poste_1, f : MobileAgentType@Poste_2]$$

Avec le typage de l'agent mobile :

$$MobileAgentType[ch : Integer@Poste_1, ch : Integer@Poste_2, \ldots]$$

Un agent de ce type peut ainsi utiliser au plus les canaux *f* placés sur le site *Poste₁* et *Poste₂* avec la possibilité de véhiculer respectivement les données décrites par le type *MobileAgentType,* c'est-à-dire de l'information lue sur *Poste*. Cette notation permet de différencier les canaux locaux par le mot clé *here* telle que l'entrée suivante: $f : AgentType@here$. Bien entendu, cette définition de type est incomplète car elle nécessite d'ajouter les autres canaux utilisés. Une telle expression de type offre de nombreuses possibilités de contrôle lorsqu'elle apparaît dans la déclaration de canal entre deux sites. Lors de l'arrivée d'un agent, par exemple, considérons la déclaration du service d'importation d'agent vu depuis le site d'accueil *Poste₁* qui n'accepte que des agents de type *MobileAgentType*:

$$PortType_{Poste_1}(f : r\langle MobileAgentType\rangle \rightarrow process[g_1 : r\langle C_1\rangle@here, warn_1 : w\langle W_1\rangle@Poste_1])$$

PortType représente la stratégie d'importation de l'agent *MobileAgent.* Ainsi un tel agent arrivant sur ce site peut seulement être instancié à partir d'un canal *f* qui autorise l'importation d'agent de type *MobileAgentType*. En outre, l'agent ainsi construit (membre droit de la transition), est seulement autorisé à lire sur un canal local g_1 des données de type C_1 et écrire sur un canal particulier nommé $warn_1$ spécifique au site *Poste₁*. Dans

notre exemple, cela signifie qu'un agent local, par exemple un *AgentHost* du $Poste_1$, peut recevoir un tel agent.

Tandis qu'avec une définition d'agent comme suit, les contraintes sont attribuées différemment:

$$PortType_{site_1}\left(f : r\langle MobileAgentType\rangle @\, Poste_1 \rightarrow process\left[g_1 : r\langle C_1\rangle @\, here, warn_1 : w\langle W_1\rangle @\, Poste_1\right]\right)$$

Le site $Poste_1$ peut recevoir un agent mobile de type *AgentMobileType* uniquement sur ce canal localisé sur $Poste_1$. Une fois migré, cet agent pourra lire des données de type C_1 depuis un canal g_1 local mais aussi écrire sur $warn_1$. Une telle déclaration d'agent importé n'est pas obligatoirement utile qu'au seul $Poste_1$, mais il peut aussi être employé sur d'autres sites afin de partager une même aptitude.

Dans les exemples précédents, les définitions d'importation d'agents mobiles et de sites d'accueil étaient faites conjointement. Il peut être utile de définir les notions séparément comme cela est le cas pour un serveur d'agents mobiles avec un canal qui permet l'arrivée d'un agent d'un type donné. Des contraintes sur les types permettent de limiter les sites appartenant à l'ensemble Site de tous les sites déjà définis :

$$AgentServer_{SrvAgents}(site : Site)PortType\Big(\; f : r\langle MobileAgentType\rangle @\, site \rightarrow process\left[g : r\langle C\rangle @\, here, warn : w\langle W\rangle @\, Poste_1\right] \;\Big)$$

Cette définition d'agent est paramétrée par le site, où sont utilisées des données de type C. Le serveur gère des agents qui peuvent écrire des informations de type W sur un canal *warn* sur les sites choisis par le receveur de l'agent. Quel que soit le site de réception, il sera possible pour l'agent importé de pouvoir accéder à une ressource locale en lecture, nommée g. Cette contrainte peut être vue comme une aptitude que doivent posséder les sites d'accueil.

Cette définition de type paramétré est essentielle dès que l'on traite un cas d'étude où figure au moins un serveur d'agents mobiles. L'inconvénient premier réside dans la contrainte forte imposée par le serveur. En effet, un client qui souhaite émettre des données correspondant au type paramétré, doit respecter la contrainte imposée par le serveur. Ainsi de nouveaux besoins peuvent être exportés par le serveur vers ses clients et cela peut amener une reconfiguration de l'ensemble des définitions d'agents. Ces contraintes ont

pour but de ne pas frauder auprès du serveur d'agents et fournir de ce fait, une protection plus grande. Il ne faut pas se tromper sur leur rôle dans une spécification dédiée aux aspects matériels.

Cette rapide présentation d'une description matérielle fait apparaître plusieurs aspects. Tout d'abord la difficulté de rédaction des spécifications d'architecture dues en partie à la richesse du langage employé mais aussi à l'étendue des caractéristiques à modéliser. En effet, ce rapide cas d'étude montre qu'il est essentiel de séparer les connaissances architectures des connaissances métiers propres. Nous avons choisi de décrire l'architecture par l'emploi d'une algèbre de processus adaptée à ce domaine pour l'aspect composable avec l'approche logicielle mais aussi pour l'expression de la mobilité.

Le cadre des descriptions faites avec Dpi calcul est large car l'architecture décrite comporte clairement deux aspects : matériel et architecture. Les aspects matériels montrent la disposition physique des sites qui composent le système et la répartition des agents sur ces sites. Les ressources matérielles sont caractérisées par certaines aptitudes ou contraintes. Les matériels sont connectés entre eux, à l'aide de supports de communication souvent appelés port. La nature des lignes de communication et leurs caractéristiques peuvent être précisées.

Les descriptions de sites peuvent montrer des instances d'agents (un agent précis tel qu'un *AgentHost*), ou des classes d'agents (tel que *MobileAgentType*).

Les aspects architecture décrits avec Dpi calcul portent sur l'architecture fonctionnelle qui doit être mise en place sur les sites. La séparation est ainsi établie avec l'architecture technique.

L'architecture fonctionnelle est en rapport avec le domaine auquel elle s'applique. En d'autre terme, elle a pour objectif d'aider à réaliser le problème posé. Alors que l'architecture technique porte sur la manière d'implémenter l'architecture fonctionnelle. Ce sera en partie le sujet de la section suivante.

Notre exemple précédent porte sur la collecte d'informations sous la surveillance d'un observateur unique. Il est clair que cette situation est fréquente dans le domaine des algorithmes de contrôle. Des patterns d'architecture doivent pouvoir être mis en place rapidement dès le début d'un projet. Le langage formel Dpi calcul permet de définir de tels patterns et ainsi mettre en place une architecture qui sera utilisée dans plusieurs autres spécifications logicielles. Notre description des types d'agents hôte et serveur illustre cet

aspect avec la définition d'interface de comportement imposée par le serveur sur les agents mobiles.

A la suite de ce constat, il semble important d'insister de nouveau sur le rôle des spécifications d'architecture faites en Dpi calcul, afin de cerner avec précision ce qui doit figurer dans une telle description et surtout ce qui ne doit pas y être. De nombreux travaux tentent de mettre en place une algèbre idéale permettant de modéliser tout un système informatique. Il est évident de devoir dissocier les aspects pour une meilleure lisibilité et surtout une gestion plus précise des spécifications dans le temps. Dans la section suivante nous allons décrire notre architecture matérielle support des collectes par agents mobiles.

III.4. Etude de cas d'architecture

Grâce au langage Dpi calcul, nous disposons d'une notation nous permettant de séparer l'architecture matérielle de la partie logicielle. Autrement dit, nous utilisons le langage Hoπ-Calcul pour définir les aspects logiciels de chaque agent, tandis que le langage Dpi calcul nous servira pour définir l'architecture type. (cf. Figure 2)

III.4.1. Architecture matérielle

Notre architecture comporte six sites dont les noms sont présentés sur la figure 2. Une définition de cette architecture est de la forme suivante :

$$\begin{aligned} Architecture \stackrel{\Delta}{=} & \; Poste_1[AgentHost \mid NTEventReader] \\ & \mid Poste_2[AgentHost \mid NTEventReader] \\ & \mid SrvDom[AgentHost \mid NTEventReader \mid NTPropertiesReader] \\ & \mid SrvWeb[AgentHost \mid NTEventReader \mid NTPropertiesReader] \\ & \mid SrvSGBD[AgentHost \mid NTEventReader \mid MySQLStatus] \\ & \mid SrvAgents[AgentHost \mid AgentDirectory \mid AgentServer] \end{aligned}$$

Équation 19 Architecture

Cette définition met en évidence un placement initial, et offre une spécification entre localité et agents. Chacun des agents est défini par une expression Hoπ-Calcul. De nouveaux agents apparaissent sur cette spécification tels que *NTEventReader* ou

MySQLStatus, ils représentent des agents immobiles sur un site donné aptes à fournir une information. Telle que les événement NT ou des propriétés associées au SGBD considéré.

En plus de cette expression globale, nous pouvons déclarer les services disponibles par site. Ainsi, le terme suivant :

$$SrvWeb = loc\begin{bmatrix} web_1 : w\langle MobileAgent\rangle, execute : w\langle AgentHost\rangle, \\ read : r\langle MobileAgent\rangle, register : w\langle AgentHost\rangle \end{bmatrix}$$

Signifie que *SrvWeb* est un site physique disposant des services :

- *web₁* : canal utilisé pour un agent *MobileAgent* en écriture, pour l'importation d'agent mobile.
- *execute* : canal utilisé pour un agent *AgentHost* pour déclencher l'exécution des traitements par l'agent mobile.
- *register* : canal utilisé pour un agent *AgentHost* pour mettre à jour la localité de l'agent mobile reçu par *l'AgentHost*.
- *read* : service local fourni par *FileReader* pour que *MobileAgent* puisse lire le fichier.

De même pour *SrvAgents.*

$$\begin{aligned} SrvAgents = loc[\ & agents : w\langle MobileAgent\rangle, execute : w\langle AgentHost\rangle, \\ & register : w\langle AgentHost\rangle, request : w\langle MobileAgent\rangle, \\ & response : w\langle MobileAgent\rangle, register : w\langle MobileAgent\rangle\ ] \end{aligned}$$

Bien entendu, tous les sites doivent posséder une telle description qui est insérée à cet endroit pour ne pas nuire à la compréhension de l'ensemble de notre démarche. Conjointement, nous déclarons aussi les services mis à disposition. Cette partie n'est pas indispensable mais elle permet de lever des conflits lorsque les spécifications sont écrites à plusieurs auteurs. Elle fournit une déclaration de plus à croiser avec la déclaration de site. Par exemple pour le service web, utile pour l'importation d'agents mobile sur *SrvWeb.*

$$web_1 : w\langle loc[execute : w\langle AgentHost\rangle, register : w\langle AgentHost\rangle]\rangle$$

Cette déclaration stipule que le service web, est déclaré accessible en écriture sur des sites où il existe 2 services ou places :

- *execute* : en écriture par un agent *AgentHost*
- *register* : en écriture par un agent *AgentHost*

Cette déclaration est cohérente avec la déclaration de *SrvWeb*. De même, les acteurs services importants peuvent posséder une déclaration propre. L'étape suivante consiste à définir les types des agents placés sur cette architecture.

III.4.2. Type d'agent

Nous allons exprimer les besoins des agents ce qui permettra ensuite d'effectuer un croisement avec les services offerts pour les sites qui les accueillent. Dans le cas d'un agent mobile, ses besoins sont dépendants des traitements à effectuer. Ce contrôle n'aurait pas été fait en section III.2.3 pour des questions de lisibilité. Grâce au langage Dpi calcul, nous ôtons ce contrôle de la spécification de l'agent mobile pour le placer dans sa déclaration de type.

$$\begin{aligned} MobileAgentType[\ & event_1 : String @ Poste_1, event_2 : String @ Poste_2, \\ & event_3 : String @ SrvDom, prpos_1 : Struct @ SrvDom, \\ & read_1 : String @ SrvWeb, event_4 : String @ SrvWeb, \\ & event_5 : String @ SrvSGBD, status_1 : Binary @ SrvSGBD, \\ & execute_1 : Channel @ here\ ] \end{aligned}$$

Cette expression stipule que tout agent de type *MobileTypeAgent* a besoin d'un service $event_1$ sur le site $Poste_1$ pour échanger des données de type texte ainsi que $event_2$, $event_3$,… De même *props* est un service qu'utilise un agent de ce type pour lire des données de type *Struct.* De même un service *execute* doit être disponible pour lancer les traitements.

Une fois ce type d'agents déclarés, il est possible de typer les services des sites de façon plus générale. Ainsi, revenons sur la déclaration de la localité *SrvWeb*.

$$\begin{aligned} SrvWeb = loc[\ & web_1 : w\langle MobileAgentType\rangle, execute : w\langle AgentHostType\rangle, \\ & read : r\langle MobileAgentType\rangle, register : w\langle AgentHostType\rangle\ ] \end{aligned}$$

L'ensemble des déclarations de site peut être revu dans ce sens.

III.4.3. Migration d'agents entre sites

Nous rappelons que toute communication s'effectue localement à un site. Le langage Dpi calcul exprime le transfert d'information entre deux sites pour la migration d'un agent contenant cette information. Dans notre cas d'étude, les agents mobiles sont créés sur le site *SrvAgents* et migrent vers les autres sites de l'architecture type. Nous avons expliqué précédemment la création des agents mobiles et leur configurateur. Nous allons désormais

exprimer le trajet d'un agent mobile sur l'architecture type de façon séparée de la spécification de l'agent mobile. Ainsi le terme suivant est une partie de notre architecture.

$$SrvAgents[AgentHost \mid AgentDirectory \mid AgentServer] \mid SrvWeb[AgentHost]$$

Par les déclarations des deux sections précédentes *AgentHost* de *SrvWeb* accueille l'agent mobile crée via web_1. La configuration de l'agent mobile apporte à cet agent la première localité de migration (première tâche à effectuer). Donc le terme précédent devient :

$$SrvAgents[AgentHost \mid AgentDirectory \mid Configurateur \mid MobileAgent] \mid SrvWeb[AgentHost]$$

A la fin de sa configuration, l'agent mobile s'enregistre via l'*AgentDirectory*, puis le terme *Realizer* permet de lancer sa mission. Ce terme est transformé par l'emploi de Dpi calcul

$$\begin{aligned} Realizer(init, gr, execute, request) \stackrel{\Delta}{=} & (\nu\ work)\,\overline{execute}(work) \\ & .work(a, s) \\ & .\overline{request}\langle gr, s, resp\rangle \\ & .resp(group, ag, port) \\ & .goto_{port}\ s.MobileAgent \end{aligned}$$

Équation 20 Realizer

Au niveau plus global de notre spécification :

$$SrvAgents[AgentHost \mid AgentDirectory \mid goto_{web_1}\ SrvWeb.MobileAgent] \mid SrvWeb[AgentHost]$$

$$SrvAgents[AgentHost \mid AgentDirectory] \mid SrvWeb[web_1!\langle MobileAgent\rangle \mid AgentHost]$$

La migration est donc orientée par l'agent migrant. L'accueil est déclaré par une réplication, c'est-à-dire que le site est toujours susceptible d'accueillir un tel agent. Afin d'utiliser la déclaration de type précédent, il est possible de généraliser cette expression.

Si la migration d'un agent mobile est effective entre les sites *SrvAgents* et *SrvWeb*, il demeure délicat de fournir dans un document tel que celui-ci la spécification dans son ensemble. Si nos besoins en spécifications formelles énoncées au paragraphe III.3.2 sont couverts par notre approche hybride, la gestion de spécification de grande taille justifie le besoin d'outil de conception formelle. Des travaux sont en cours au LCAL pour enrichir l'outil HOPitool afin de gérer des spécifications écrite en Dpi calcul. Il est certain que l'usage de cet outil sera une aide de poids pour le développement de notre approche de

conception. Celle-ci pilote notre démarche projet jusqu'à la phase de réalisation que nous abordons au chapitre suivant.

IV. Mobilité par patterns interposés

La définition d'une approche mobile doit être robuste, simple, sécurisée et compréhensible. En conséquence, pour répondre à notre besoin de surveillance, nous avons décidé, dans un premier lieu, de mettre en place notre modèle de conception d'agents mobiles dont une partie a été décrite au chapitre III. Nous avons fourni au chapitre précédant, une analyse formelle de notre solution ; il nous faut désormais avancer dans notre démarche et nous intéresser à la conception avancée.

Ainsi, nous souhaitons construire un socle réutilisable pour notre approche de la mobilité avec comme application directe un outil de surveillance logicielle.

Dans ce chapitre nous présentons le rôle d'un modèle de conception puis notre modèle comportemental d'agents mobiles. Dans une troisième partie nous définirons notre mécanisme de communication d'agent mobile. Pour terminer, nous décrirons la technologie Jini, celle utilisée pour l'implémentation de notre modèle de conception.

IV.1. Le rôle des modèles de conception

Les modèles de conception ou design pattern sont des outils de conception avancée dont l'usage est très répandu de nos jours. Effectivement, c'est un moyen d'accroître la vitesse de construction d'une maquette tout en assurant de bonnes propriétés à la solution a terme : par exemple la compositionnalité.

La première présentation structurée de modèles de conception a été publiée par Gamma, Barres, Johnson, et Vlissides [109], elle fait référence encore aujourd'hui.

Nous travaillons sur des applications ayant des propriétés récurrentes à savoir la mobilité dans un réseau. Ce sujet a beaucoup d'applications mais peu, parmi elles, ont une approche systématique. Notre expérience accentue quelques aspects liés à ce sujet :

- Comment un agent peut se déplacer d'un hôte à l'autre.
- Comment un agent peut effectuer une tâche et sauvegarder les résultats.

La mobilité est une propriété inhérente à un agent depuis sa création, jusqu'à sa mort. Elle est utile tant au long de la mission que doit remplir l'agent. Aussi, nous avons choisi de

définir un premier pattern de conception. Ce n'est pas une première approche parce que dans une étude précédente, D. Bonura, R. Culmone, et M. Angeletti ont défini un modèle pour un agent mobile réactif qui peut être appliqué à une grande base de données [110]. D'un autre coté, D. Deugo, F. Oppacher, J. Kuester, et I. Von Otte ont présenté une approche pour concevoir et appliquer des modèles indépendamment des plates-formes spécifiques en tenant compte de leurs spécificités [111]. Ces auteurs ont montré qu'une vue proportionnée indépendante de la plate-forme peut être construite afin d'être employée comme guide et d'utiliser le modèle dans différentes plates-formes. Notre approche reprend certains aspects bas niveau tel que l'éventualité qu'un agent se déplace sur un ensemble d'hôtes tout en exécutant des tâches ou en faisant appel à d'autres agents. Cependant, nous avons les mêmes contraintes à propos de la portabilité comme contrainte essentielle de notre solution.

Afin de construire un pattern de conception, il est de coutume de collecter les caractéristiques vues au travers de plusieurs applications et d'en abstraire les notions communes. Enfin cette démarche se termine par une illustration graphique ou opératoire.

Les thèmes principaux sont : détecter la similitude des applications, résumer et unifier la technique employée de ces applications, formaliser cette technique, et enfin analyser son utilisation (où, quand, qui, comment, et pourquoi).

IV.2. Modèle comportemental d'agents mobiles

Au sein de notre groupe de travail, nous avons modélisé et réalisé plusieurs applications à base d'agents mobiles dont une application de surveillance des données utilisant une base de données [112] où un agent collecte toute modification ayant atteint aux protocoles étudiés. Les déplacements sont implémentés en utilisant la technologie Jini, un projet Sun, basé sur la plateforme Java, permet le développement des services distants. Cette technologie va être décrite ultérieurement car elle est utilisée pour l'implémentation de notre approche d'agents mobiles.

Parmi les projets réalisés, nous avons également écrit une implémentation du protocole Service Location Protocol (SLP) [113], il fournit un cadre extensible pour la découverte et le choix des services sur réseau. Dans les spécifications de ce protocole, la recherche des services est obtenue en utilisant un agent mobile plutôt qu'un découvreur mobile. Le but

est de découvrir où le service est défini. Dans ce contexte, nous avons aussi utilisé, la même technologie précitée.

Une autre expérience porte sur la gestion des déplacements d'un robot d'une pièce à l'autre [114]. Nous avons réalisé, encore une fois, une implémentation pour ce robot. Sa capacité à se déplacer dans les salles peut être comparée aux études précédentes, excepté le fait que des objets locaux peuvent être exportés d'une pièce à l'autre. Ceci signifie que ces objets locaux peuvent devenir mobiles. Nous avons employé la même approche pour le robot et les meubles exportés de la salle.

De ces études et réalisation nous avons affiné nos travaux, analysé nos erreurs et collecté les atouts. L'étude et analyse de nos anciens travaux, la formalisation et le perfectionnement d'une technique sont des aspects qui sont intimement liés. Comme nous l'avons mentionné, il est capital de faire abstraction des aspects trop techniques. Il parait évident que la technologie évolue vite, les modèles de conception que nous souhaitons construire doivent accepter le changement de technologie afin de ne pas devenir des modèles de programmation.

IV.2.1. Design Pattern DMA

Les travaux que nous avons trouvés et/ou conçus ont différents niveaux d'application et différentes origines. D'ailleurs, les méthodes de conception d'agents ne sont pas uniquement utilisées par des experts mais également par des utilisateurs qui ne maîtrisent pas avec précision les concepts complexes et tous leurs détails. Heureusement, les modèles de conception sont une technique remarquable pour enseigner et réutiliser des conceptions. Ils ont une manière unique de transmettre l'expérience, le développement et leurs limites.

Nous avons utilisé nos designs patterns lors de la conception avancée de notre application type, présentée chapitre III section 1. Cette application est un bon candidat pour l'utilisation de par ses caractéristiques réactives, dynamiques et pour les besoins de communication.

Cette section présente le design pattern principal nommé Distributed Mission Assignment for Mobile Agents (DMA) il met en place la structure de notre solution (Figure 5). À un niveau plus précis, il compose deux design patterns plus élémentaires : le premier pour la communication et le second pour la gestion des missions.

La figure 4 montre l'architecture physique et le besoin d'un agent local sur chaque site receveur pour l'importation et la poursuite de la mission de l'agent mobile. Comme nous l'avons spécifié (chapitre 3 Equation 12) l'*AgentHost* offre ce service d'accueil

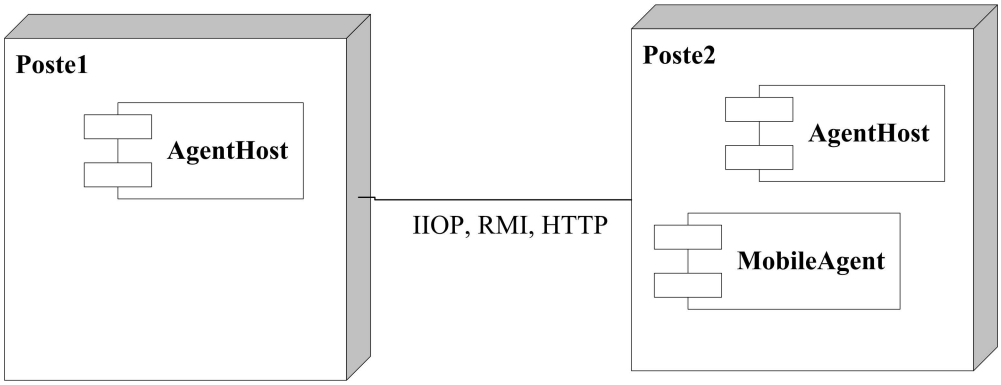

Figure 4 Diagramme de déploiement d'un agent mobile

Quand l'agent mobile désire se déplacer, il contacte l'*AgentHost* pour qu'il l'importe vers le site en question. Le diagramme de composants ci-dessus décrit cette situation pour un agent se déplaçant du site Poste 2 vers le site Poste 1.

Le diagramme de classe suivant (cf. Figure 5) met en évidence les principales entités de notre modèle. Il est structuré en deux packages nommés *tasks* et *mobile* (cf. Figure 6) : le premier package regroupe les différentes tâches qu'un agent peut effectuer. Ces tâches définissent les missions de l'agent en terme de séquence d'actions, cela peut être *FileReader* (Equation 15) pour lire le contenu des fichiers plats, *NTProperties* pour recueillir les données du système d'exploitation Windows (mémoire, taille disque, charge CPU...), etc.

La classe *Agent* est celle qui définit les propriétés et le comportement d'un agent à un niveau plus élevé. La classe *Host* et *Mobile* dérivent de la classe *Agent* pour définir de nouveaux attributs et de nouveaux comportements. Ainsi nous pouvons faire la différence entre un agent local et un agent mobile malgré leurs propriétés communes. L'interface IAgent déclare toutes les méthodes distantes pouvant être invoquées par le client en l'occurrence un agent. Cette interface permet de détecter les similitudes parmi des classes sans forcer artificiellement leurs rapports, déclarer des méthodes (opérations) pour qu'une ou plusieurs classes ou services puissent implémenter et indiquer un objet sans fournir sa classe. Elle est implémentée par les deux classes Host et Mobile et définie précédemment dans le chapitre III comme étant *AgentHost* (Equation 18) et *MobileAgent* (Equation 3).

La dernière classe *Properties* représente les informations d'un agent (son hôte, son groupe, son nom…)

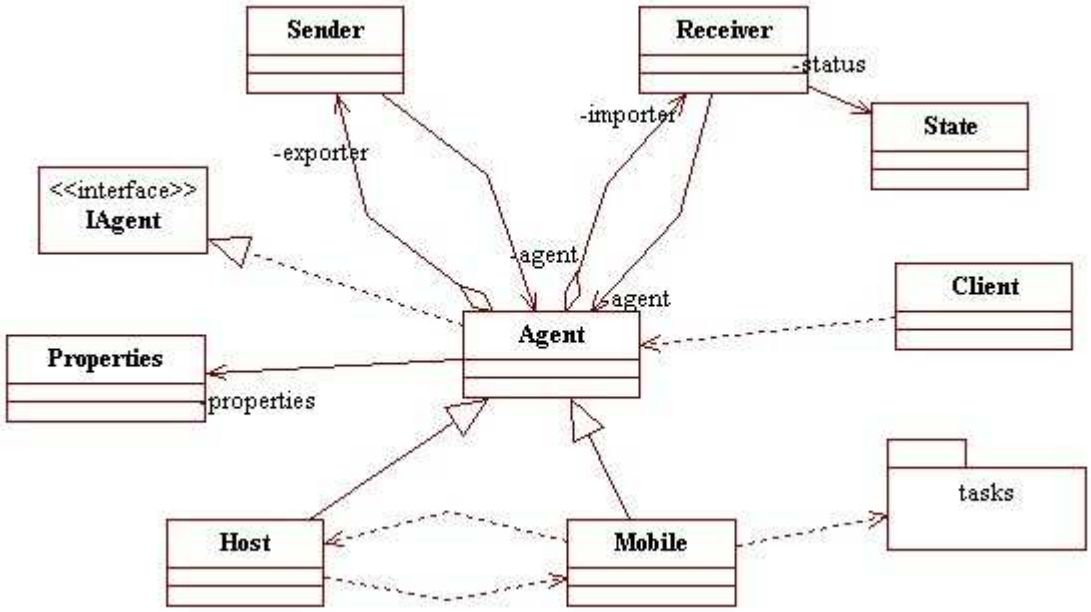

Figure 5 Structure du design pattern DMA

Le deuxième package représente le cycle de vie d'un agent en deux niveaux :

- L'état de création jusqu'à ce que cet agent soit disponible pour les hôtes,
- La durée de vie durant laquelle cet agent est contacté ou retourné par un hôte,

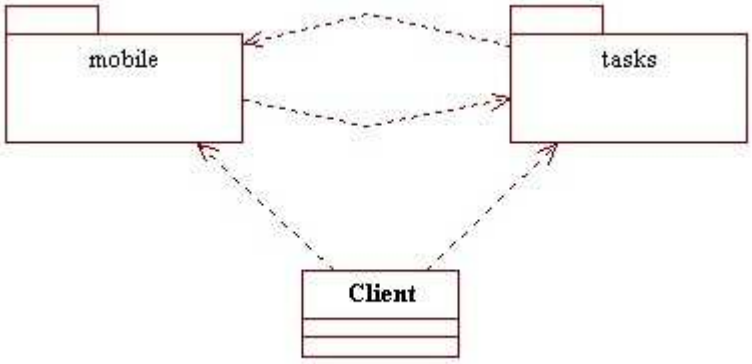

Figure 6 Diagramme de package

Ce découpage décrit un contexte qui ressemble à une personne dans une société qui attend un ordre de mission pour pouvoir se déplacer et l'effectuer. Une fois la mission terminée, la personne devient opérationnelle pour une nouvelle mission. Cette métaphore n'est pas

entièrement significative si nous ne mentionnons pas que l'expérience acquise par la personne en question sera utile pour les prochaines missions.

Donc, nous avons choisi la stratégie de la mobilité suivante :

- Chaque hôte désirant recevoir un agent doit révéler son intention.
- Chaque hôte souhaitant se déplacer doit révéler son habilité.
- Si un agent quitte un hôte pour se déplacer, il doit éventuellement annoncer son départ.

Ainsi, le deuxième package *mobile* est structuré de telle sorte que les agents peuvent changer leur comportement en fonction de leur état (design pattern State). Ce qui nous ramène à une agrégation de deux entités essentielles pour le cycle de vie de l'agent : *Sender,* un service de migration responsable de l'exportation d'un agent vers un autre hôte et *Receiver* procèdent en deux états (classe *State*) :

- *Waiting* quand il attend pour la réception des agents.
- *Working* pour aider un agent à exécuter une tache en local

Ce modèle comportemental permet à l'utilisateur d'avoir une approche systématique de la mobilité pour toutes ses applications. Naturellement, il garantit une meilleure lisibilité pour l'utilisateur.

IV.2.2. Création d'un agent mobile

Nous avons défini dans le diagramme de classe précédant, une classe générique *client* pour garder un niveau d'abstraction élevé. Cette classe interagit avec le modèle pour pouvoir créer des agents locaux et des agents mobiles. Pour les agents locaux, la classe *client* peut être assimilée à un programme de déploiement, alors que pour les agents mobiles, c'est une instance de l'*AgentServer*. Dans les deux cas, une fois les agents mobiles créés, ils doivent s'enregistrer auprès de l'*AgentDirecotry*.

Le diagramme de classe suivant (cf. Figure 7), représente la structure des deux agents *AgentServer* (Server) et *AgentDirectory* (Directory) que nous avons conçu par rapport à notre analyse formelle du chapitre III. En effet, pour l'*AgentDirecotry* (Equation 7) nous avons inclus le comportement de l'*AgentMemory* (Equation 8) dans celui de l'*AgentDirecotry* puisque dans une approche conceptuelle, contrairement à une approche formelle, nous pouvons représenter une mémoire par une collection d'éléments.

Nous avons également, inclus le *Configurateur* (Equation 1) dans l'*AgentServer* pour garder une cohérence conceptuelle. Notons que le seul composant qui peut créer une séquence de missions, afin de la transmettre à *MobileAgent* est l'*AgentServer*.

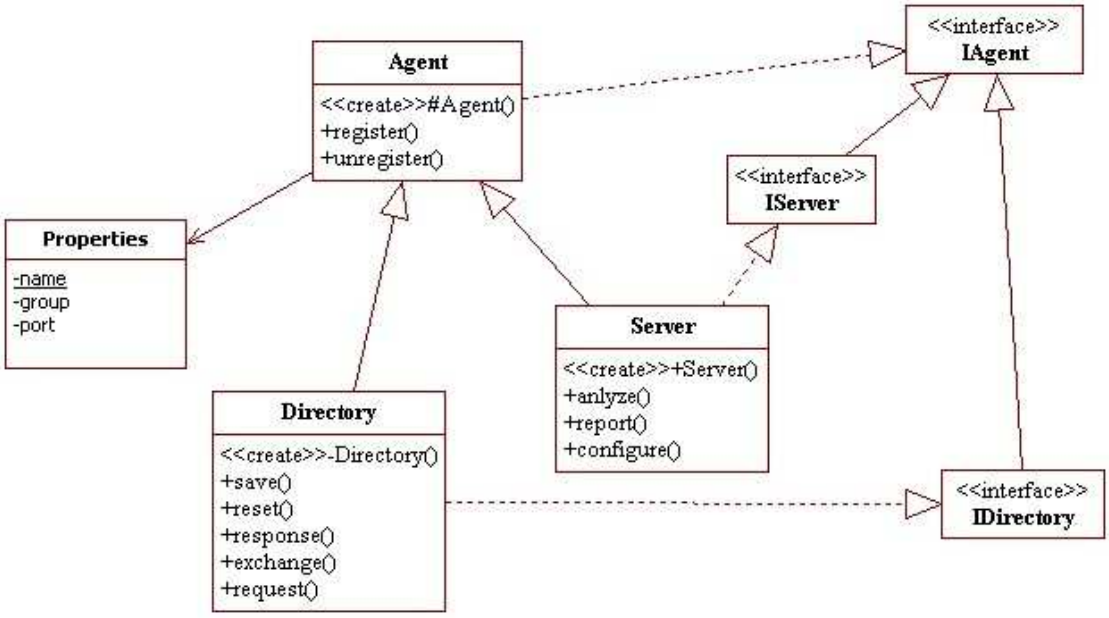

Figure 7 Diagramme de classe de l'*AgentServer* et l'*AgentDirectory*

Contrairement aux deux agents *Host* et *Mobile*, les deux agents *Server* et *Directory* ont un comportement distinct même s'ils sont des agents mobiles. C'est pour cette raison que chacun d'eux doit implémenter sa propre interface IServer et IDerectory. Ces derniers héritent de *IAgent* définit précédemment, cela implique qu'elle hérite des méthodes distantes d'un agent.

Le diagramme de séquence suivant (cf. Figure 8), décrit l'étape de création d'un agent mobile. Cette étape est une partie du cycle de vie de l'agent, les autres parties sont abordées dans les points suivants.

Dans un premier temps, l'*AgentServer* construit la liste de route de l'agent mobile cette route est constituée de plusieurs associations sites et missions, comme cela a été défini dans notre analyse formelle. Généralement, ces informations sont disponibles dans une unité de stockage gérée par l'utilisateur. Dans un second temps, l'*AgentServer* crée un agent mobile en lui fournissant sa feuille de route, son nom ainsi que son groupe. Enfin, l'agent mobile s'auto publie dans l'*AgentDirectory* pour qu'il soit visible par les autres agents de la communauté.

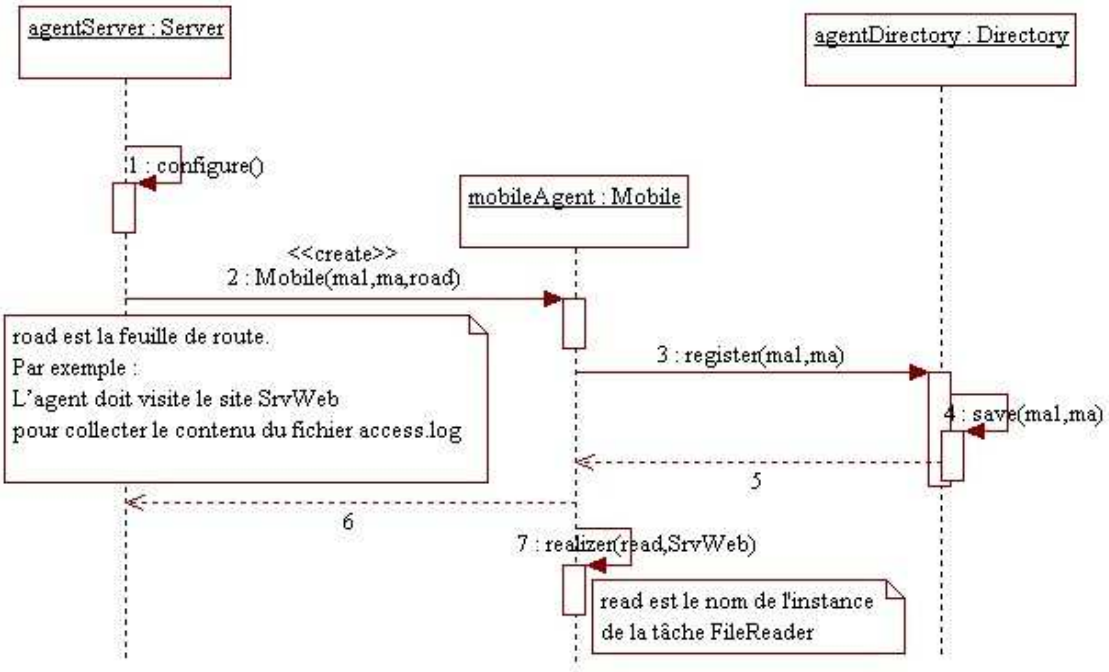

Figure 8 Diagramme de séquence de création de *MobileAgent*

Une fois enregistré, l'agent mobile débute son parcours pour se déplacer vers le premier site de sa feuille de route, ce qui constitue la deuxième partie de son cycle de vie.

IV.2.3. Le déplacement de l'agent à travers les hôtes.

Dans notre analyse formelle, un *AgentHost* est considéré comme un agent qui attend le contact d'un agent mobile pour l'importer vers son site d'accueil, il peut également l'aider afin de l'exporter vers une autre destination. Ces deux activités de l'*AgentHost* sont gérées par les entités *Sender* et *Receiver* (cf. Figure 5) qui doivent s'exécuter en simultanée.

Cependant pour que le *MobileAgent* contacte *l'AgentHost*, ce dernier doit s'enregistrer auprès de *l'AgentDirectory* au même titre que les autres agents afin qu'il soit connu par la communauté.

Le diagramme de séquence suivant (cf. Figure 8) résume le déplacement d'un agent mobile. Une fois que l'*AgentHost* s'est enregistré, le *MobileAgent* contact l'*AgentDirectory* pour récupérer le canal de communication de l'*AgentHost* puis le contact. La sous entité *Receiver* de l'*AgentHost* récupère le *MobileAgent* ensuite elle notifie l'*AgentDirectory* par un changement de site puis elle relance l'activité du *MobileAgent* afin qu'il poursuive ses missions.

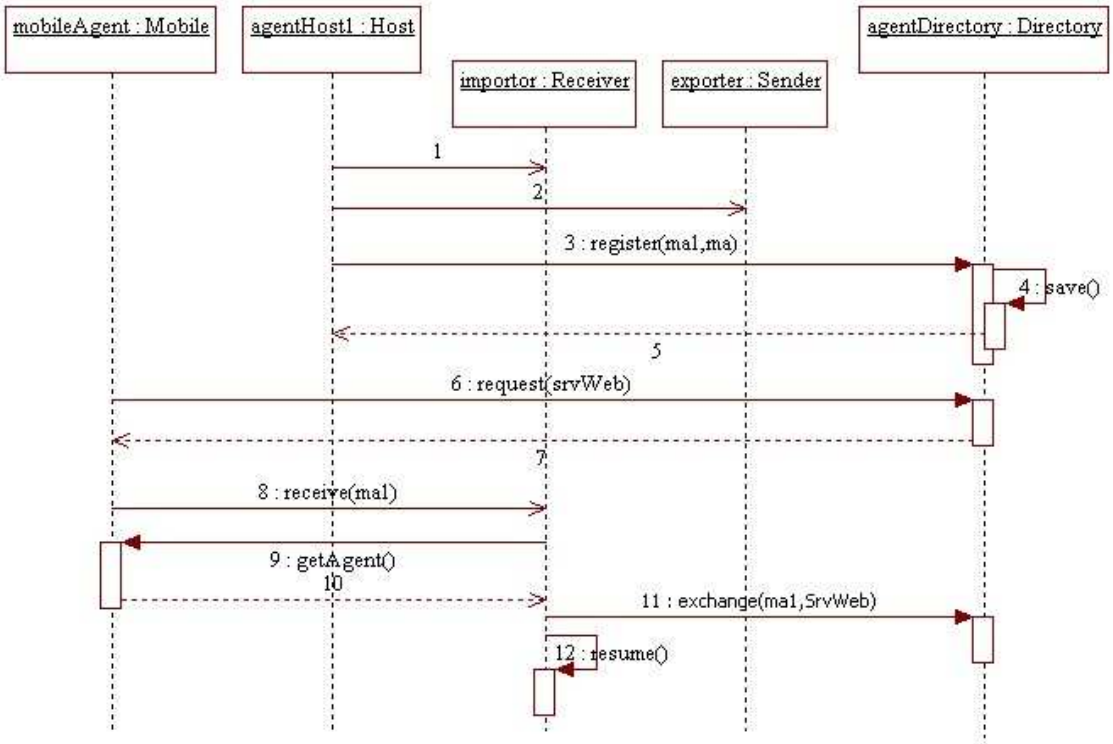

Figure 9 Diagramme de séquence de la migration de MobileAgent

Une fois le *MobileAgent* démarré, il exécute les actions sur son site. C'est la troisième étape de son cycle de vie.

Pendant la période d'attente de l'*AgentHost* d'une demande d'importation son état est "*Waiting"*. Cet état est changé par la requête d'importation qui le change en "*Working"*. Le diagramme d'état suivant modélise le changement d'état de l'*AgentHost*.

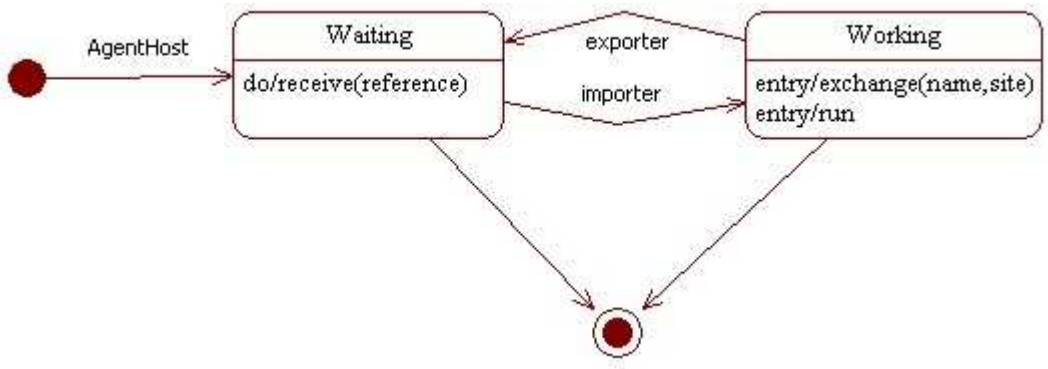

Figure 10 Diagramme d'état de transition de l'AgentHost

L'AgentHost peut atteindre l'état final dans deux cas, soit par une action de l'utilisateur ou par l'arrêt de la machine où il est déployé.

IV.2.4. L'exécution d'une tâche

Comme nous l'avons décrit dans l'étape de déplacement de l'agent, une fois que *MobileAgent* change de site, il est relancé par *l'AgentHost* pour qu'il continue ses actions c'est-à-dire effectuer l'ensemble des tâches programmées sur le site d'arrivée.

Initialement, les tâches sont transmises au *MobileAgent* lors de sa création par l'*AgentServer*. Généralement, elle sont liées à des ressources, par exemple : la tâche *FileRead* (Equation 15) doit accéder à un fichier pour collecter son contenu sauf que le fichier est une ressource distante, de ce fait le *MobileAgent* ne peut disposer que du nom et du lecteur approprié. Donc, dès qu'il arrive sur le site, il demande à *l'AgentHost* l'accès à la ressource. C'est pour cette raison que nous avons fait la distinction entre les tâches et les ressources.

Cette différenciation est présentée dans le diagramme de classe suivant, le *MobileAgent* fait référence aux deux tâches *FileReader* et *OSEventReader* qui dépendent respectivement des sources *FileSource* et *OSEventSource* accessibles par *l'AgentHost*.

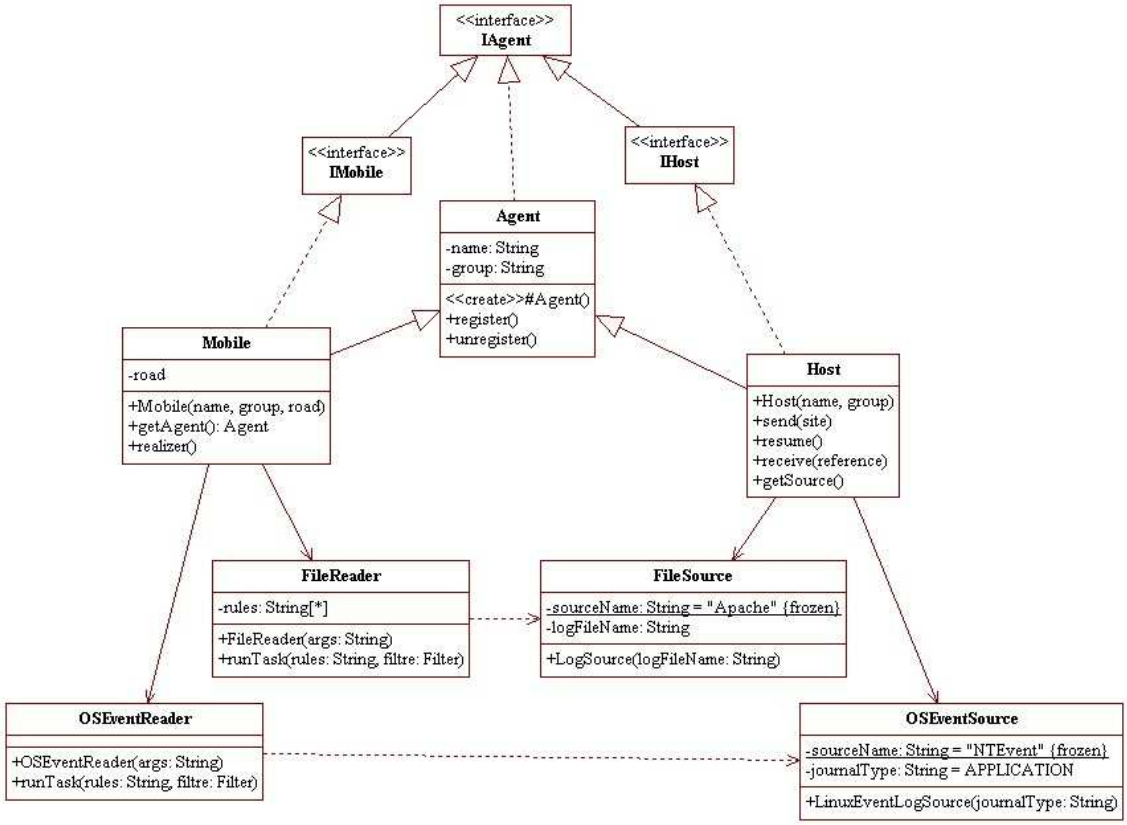

Figure 11 Diagramme de classe de l'AgentHost et le MobileAgent

Le diagramme de séquence suivant résume la collaboration entre les deux agents pour l'exécution d'une tâche. Cette opération, définie dans le bloc *loop*, se répète autant de fois qu'il y a de tâches à effectuer sur le site. Une fois que toutes les tâches sont effectuées, le *MobileAgent* demande au *Sender* de le migrer vers un nouveau site.

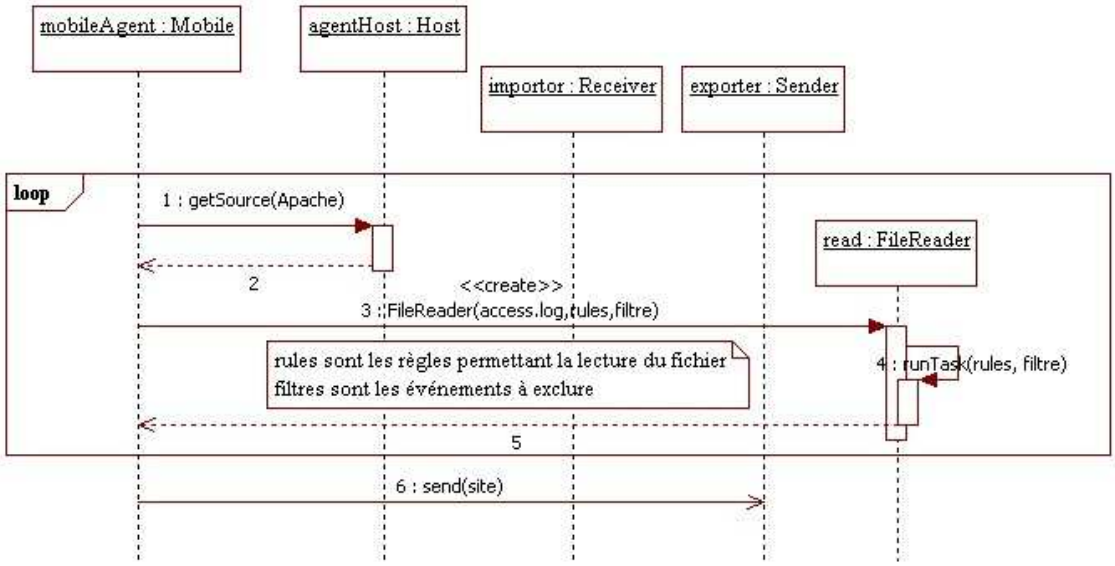

Figure 12 Diagramme de séquence de l'exécution d'une tâche

Dans le diagramme de classe précédant (cf. Figure 11) nous illustrons un scénario à deux tâches et nous nous sommes appuyés sur le modèle de conception Abstract Factory [115] qui permet la création d'objets regroupés en familles sans devoir connaître les classes concrètes destinées à la création de ces objets. C'est une manière élégante et facile pour créer des nouvelles tâches sans avoir à modifier l'existant. Le diagramme de classe suivant présente une évolution du digramme de classe de l'*AgentHost* et le *MobileAgent* incluant la nouvelle structure des tâches.

Dans ce nouveau diagramme, nous séparons la définition des tâches en deux catégories car un agent mobile doit pouvoir exécuter des tâches de collecte ainsi que des tâches d'actions.

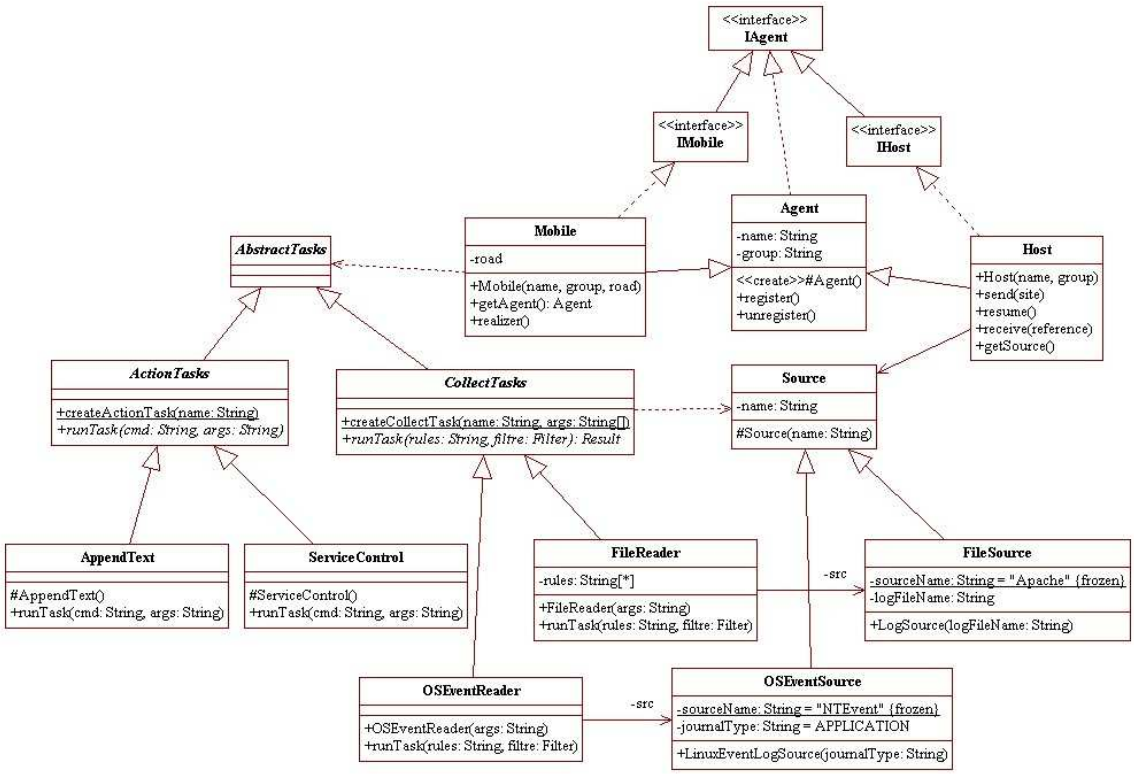

Figure 13 Diagramme de classe des tâches

Quand l'agent mobile a terminé son parcours, il revient à son site initial, celui de l'*AgentServer*. Cette dernière étape définit la fin de la mission de l'*AgentMobile*.

En conclusion, l'agent est une entité mobile se déplaçant parmi divers sites. Cet agent n'est pas intelligent, mais il possède la propriété communicative, car il révèle des informations dès son arrivée sur chaque site. Métaphoriquement, cet agent "parle", mais ne peut pas "écouter". La capacité de recevoir des données dont il a besoin, devient essentielle. Cette capacité implique non seulement le problème de performance mais également les contraintes logistiques. Ceci peut être résolu par la conception d'un modèle de communication.

IV.2.5. Besoin de communiquer

Les agents n'ont pas seulement besoin de se déplacer, mais aussi de communiquer avec d'autres sources d'information ou d'autres agents. Le système le plus courant de transmission des messages pour les agents est Knowledge Query Manipulation Language (KQML) [49]. Comme nous l'avons noté précédemment, KQML est un mécanisme de communication parce qu'il permet des formes beaucoup plus complexes d'interactions que

des mécanismes de question/réponse. Il permet à des agents de communiquer en utilisant un ensemble de messages riches. Il est aussi capable de communiquer les attitudes de l'information plutôt que des données et leurs faits. Les systèmes de passage des messages comme KQML n'ont pas besoin de la mobilité. Ils peuvent simplement passer un message qui sera livré par un mécanisme de transport. Ceci constitue une contrainte principale de l'immobilité du récepteur.

Il est difficile de réaliser des groupes de communication pour les agents stationnaires dans un modèle client/serveur. Cette réalisation est encore plus difficile pour une communauté d'agents mobiles dans un contexte mobile.

Par exemple deux agents communiquent, l'un d'eux désire se déplacer alors que les deux agents veulent toujours collaborer pour la réalisation de certaines tâches. Ils devront être en mesure de garder un moyen de reconnecter leurs voies de transmission après la migration.

L'infrastructure fondamentale de mobilité devrait maintenir ces voies de transmission. Cette infrastructure fournit une couche d'abstraction sur le mécanisme fondamental de communication pour faire ce travail. C'est l'idée de notre toile de communication.

IV.3. Évolution de communication

La communication d'un agent est un dispositif important qui a été ajouté à notre modèle d'agent mobile. Au début, la communication entre les agents mobiles a semblé moins importante que la migration. Mais, l'introduction d'échange de données dans un contexte mobile est bouleversante.

Deux approches principales peuvent être employées :

- L'utilisation d'une adresse de réception (Postmaster), où chaque agent mobile a sa propre référence unique. Ce choix est proche d'un modèle de communication entre les agents immobiles parce que l'expéditeur peut se déplacer après avoir exporté ses données et la réception est faite par une boîte aux lettres immobile.
- Une utilisation d'un messager mobile. Puisqu'un agent mobile connaît sa feuille de route ou une partie, il peut créer ou préparer un messager avant de quitter l'hôte. Le rôle de celui-ci est de propager un type de message d'un hôte à l'autre (par exemple, le prochain hôte sur la feuille de route).

Cette solution est une association directe entre la mobilité et la communication. Ce choix pourrait être considéré comme peu puissant, parce qu'une émission peut être réalisée par

une séquence de communication. Mais, son premier atout est de prendre en considération des événements pendant la communication. La migration de l'agent récepteur est un événement et l'échange des données pourrait être un autre. Mais, les deux ne sont pas identiques si le récepteur est présent ou non à un endroit précis. Notre solution peut donc traduire cette situation.

Les autres atouts sont le reportage ou la diffusion des données à cause de l'événement local. La localité a un impact direct sur la communication. Le messager peut envoyer un premier récépissé à l'expéditeur pour confirmer la réception du message. Au final, l'agent mobile notifiera que le message a bien été utilisé. Nous avons ici deux événements distincts pouvant être employés au lieu d'un seul en utilisant la première solution.

Notre toile de communication a plusieurs composants qui sont détaillés ci-dessous : couche de messager, expéditeur d'événement, boîte aux lettres comportementale locale.

IV.3.1. Messenger Layer

Deux mécanismes sont employés lorsqu'un agent mobile quitte un agent hôte : l'utilisation des messagers et celle des rôles.

Pour l'utilisation des messages nous considérons un messager comme un agent mobile en attente d'un message entrant. Sa tâche est de propager ce dernier jusqu'à une prochaine étape (vers le vrai récepteur du message ou à un autre messager relais).

Puisqu'un messager n'est pas suffisant dans certains contextes, nous avons ainsi ajouté des rôles. Un messager a des rôles et ne réagira que si le message entrant possède lui-même un rôle appartenant à un ensemble de rôles qui lui est propre. La figure suivante montre une structure des messagers, appelés *Msn*, quand un agent mobile quitte son hôte.

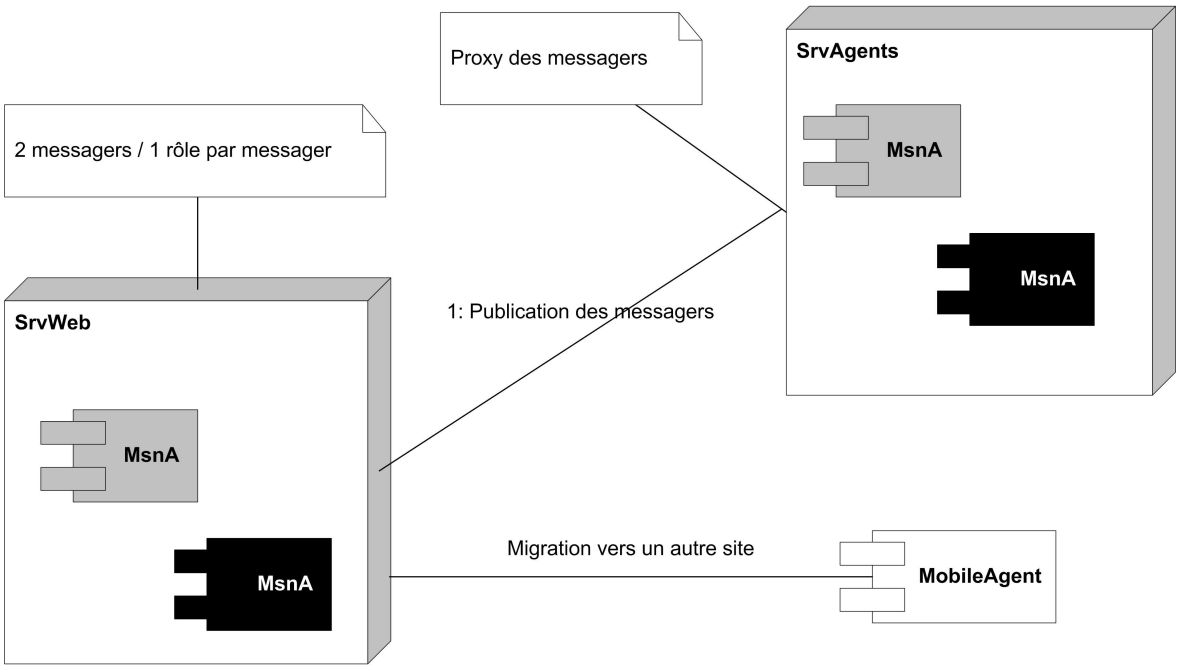

Figure 14 Création d'un "Messenger" avant la migration

Quand un message entrant arrive, trois cas de figures sont pris en considération :

- L'agent récepteur est présent et traite lui-même la réception,
- L'agent récepteur n'est pas présent contrairement à ses messagers. Mais, il y a un messager qui a le bon rôle. Quand l'agent mobile migre, son nouvel emplacement est déjà connu par la construction du messager. En raison de sa nature son déplacement signifie la création de son clone. Il y aura toujours ce genre de messager sur l'hôte jusqu'à ce que l'agent mobile initial décide de l'arrêter.
- Le récepteur n'est pas présent contrairement à ses messagers. Et aucun messager n'a le bon rôle. De ce fait, le message est perdu.

La gestion des messagers est gérée par l'agent récepteur. Il les crée en fonction de ses besoins. Ces nouveaux services sont enregistrés dans un registre local. L'agent récepteur pourrait les détruire en cas de besoin. Par exemple, un agent attend des données, quand le message entrant est lu par l'agent (via son messager), il peut annuler le service en détruisant le messager associé.

IV.3.2. Event Dispacher

La section précédente décrit la première étape d'échange de données. Dans cette section, nous expliquerons la migration du messager et le processus pour obtenir le message.

Quand un messager arrive sur un hôte, il possède le message pour l'agent mobile. Le déplacement du messager s'effectue de la même façon qu'un agent mobile. La nouveauté, c'est la création de l'événement du messager sur l'hôte. Cet événement engendre trois possibilités :

- L'agent receveur est présent sur l'hôte alors il donne le message et s'arrête. Aussi, l'agent mobile peut décider des conséquences de sa politique du messager.
- L'agent receveur n'est pas présent sur l'hôte en question contrairement à ses messagers. Il y a un messager qui possède le bon rôle. Ce qui veut dire que le messager local connaît la prochaine destination de l'agent mobile. Pour respecter le modèle de conception, le messager local configure les messages entrants puis se déplace à la destination suivante.
- L'agent receveur n'est pas présent sur l'hôte contrairement à ses messagers. Et aucun messager ne possède le bon rôle. De ce fait le message est perdu et le messager s'arrête.

L'utilisation du registre local est importante, elle évite toute confusion entre les services de messager et réduit les temps de recherche.

L'utilisation du messager permet aux utilisateurs de surveiller et de reporter des flux de messages. Les messages peuvent être augmentés ou filtrés en fonction du chemin utilisé par le messager.

Dans un système d'agents mobiles, certains agents peuvent migrer plusieurs fois dans le même contexte. Il est essentiel d'estimer qu'il y aura des conflits entre les messagers antérieurs. D'ailleurs, nous pouvons décider qu'un message ne suive pas le même chemin que l'agent receveur. Il est plus intéressant de délivrer les messages le plus rapidement possible. C'est pour cette raison que nous avons décidé de gérer un nouveau type d'événement appelé agent mobile *GainedEvent*. Cet événement se produit quand un agent host accepte un agent mobile.

La gestion de cet événement permet au concepteur de l'agent mobile de choisir entre deux stratégies :

- Quand l'agent mobile arrive sur un hôte, ses messagers précédents sont arrêtés, parce qu'il est présent pour la réception.
- Les messagers continuent à réagir comme auparavant.

D’autre part, quand l’agent mobile quitte l’agent host encore une fois, un autre événement appelé agent mobile *LostEvent* se produit. Ainsi, la gestion des événements permet la mise à jour de la politique des messagers. De nouveau, deux cas peuvent être définis :

- Quand l’agent mobile quitte un agent host, les messagers ne seront pas créés puisque les anciens messagers sont déjà présents.
- Quand l’agent mobile quitte un agent host, de nouveaux messagers sont créés car les anciens sont stoppés par l’événement *GainedEvent*. Une autre raison peut être plus complexe : l’agent receveur désire faire la distinction entre les messagers de la précédente et actuelle migration. Il est possible d’assigner une priorité plus élevée aux anciens messagers.

La figure suivante résume les différents cas :

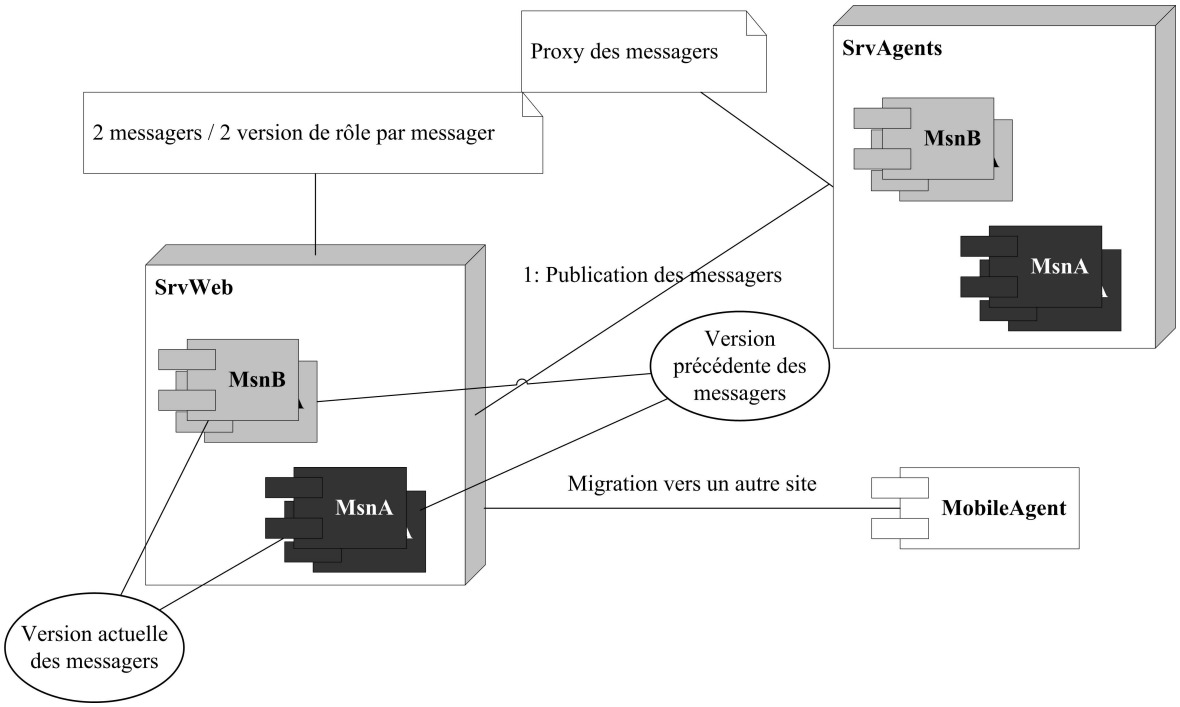

Figure 15 La deuxième migration de l'agent mobile

Cette observation est utile pour le contrôle de l'algorithme où une propriété locale doit être trouvée mais la stabilité doit être établie (par exemple distributed termination detection"). Ce genre de contexte se produit avec des algorithmes numériques où quelques calculs sont croissants et convergent vers une solution, par la méthode FDTD [116].

La dernière étape est la réception du message initial et la mise à jour de la politique du messager.

IV.3.3. Behavioral mailbox

L'étape de réception est un événement principal parce qu'elle implique deux changements. Un premier porte sur l'état de l'agent mobile lui-même et un second porte sur la chaîne des messagers qui constitue le lien faible entre l'expéditeur et le récepteur.

D'abord, la réception d'un message est le résultat d'une communication asynchrone. Ce message a un rôle spécifique qui implique une activité locale pour l'agent récepteur. Mais plusieurs messages peuvent arriver de façon indépendante les uns des autres. C'est pour cette raison que nous avons défini pour chaque agent mobile receveur une boîte aux lettres. Cette structure est divisée en plusieurs cas, un cas par rôle. Les messages sont triés et un agent peut lire le message noir avant le gris. (cf. Figure 9)

Ensuite, la stratégie du messager peut être changée. Très souvent, un messager a été créé seulement pour un seul message. Habituellement, il est possible de deviner le nombre de messages qu'un messager doit propager, particulièrement lorsqu'ils ne sont pas supprimés quand l'agent mobile revient sur un hôte où se trouvent ses messages. Un premier regard à ce contexte implique que le concepteur du système d'agent mobile doit décider d'une manœuvre à l'étape de réception. Nous avons défini deux stratégies distinctes :

- Le message reçu était dernier. Puis, tous les messagers créés ne sont plus utiles. Il est nécessaire d'arrêter les messagers qui ont le rôle précis et de modifier ceux qui possèdent ce rôle. Pour faire cette action, l'historique des agents d'accueil doit être sauvegardé par l'agent mobile pendant la migration.
- Le message reçu est un élément de la séquence des messages. L'agent mobile peut décider de mettre à jour la configuration des messagers qui ont été créés par son déplacement. Cette action peut être faite en employant l'historique de l'agent de l'hôte (contrôlé par l'agent mobile). La nouvelle configuration des messagers contient l'endroit précis de l'agent mobile. Très souvent, cette étape est couplée avec l'arrêt des messagers qui sont intermédiaires dans une chaîne de responsabilité. Ce qui signifie que cette stratégie peut être plus complexe parce que sa conception doit être appropriée.

La structure composée de l'agent mobile est devenue plus complexe depuis la première approche. La figure ci-dessous présente toutes les parties d'un agent mobile avec leurs responsabilités.

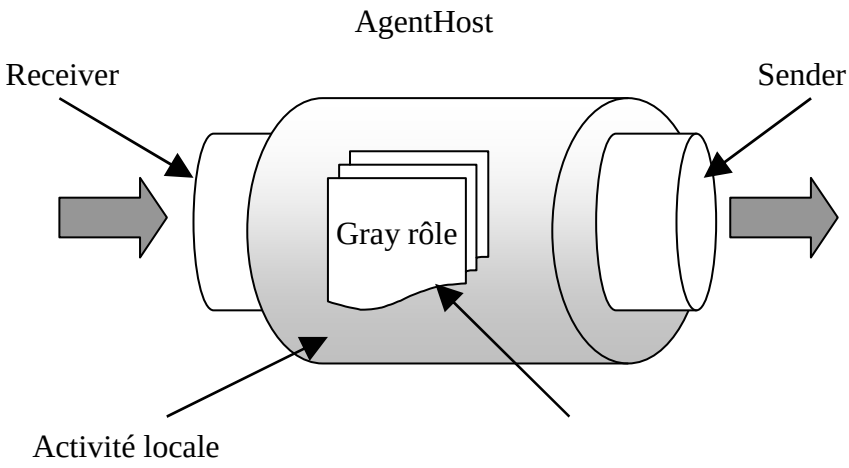

Figure 16 Les quatre activités concourantes de l'agent mobile

La structure de l'agent hôte a changé également parce qu'il écoute les événements survenus par l'agent mobile. Pour résumer cette section, nous soulignons que le dispositif de notre approche est composé de plusieurs couches où tous les agents peuvent se déplacer d'un agent vers un autre. Un messager est employé non seulement pour un agent mobile mais il pourrait être utile pour un autre messager se déplaçant avec son agent hôte.

IV.4. Implémentation de notre approche mobile.

Pour l'implémentation du modèle d'agent mobile précité nous avons choisi la technologie Java plus répandue pour ce type de code. En plus depuis 1998, Sun a définit une spécification [16] basée sur Java permettant l'implémentation des systèmes distribués correspondant à notre approche.

A ce jour, plusieurs implémentations de cette spécification sont disponibles : JSC and Seven, Servicehost, Rio, Harvester, H2O et CoBRA. L'ensemble de ces implémentations ont été présentées récemment par Svetozar Misljencevic [117].

Dans notre cas, nous avons opté pour l'implémentation du constructeur parce qu'elle a déjà été utilisée par notre équipe pour d'autres travaux et aussi pour faciliter son utilisation. Cette implémentation nommée Jini, porte le même nom que la spécification.

Jini est une architecture réseau destinée à la construction de systèmes distribués sous la forme de services coopérants modulaires [118], introduite par Sun dans l'idée de fédérer un

groupe d'utilisateurs et des ressources en faisant abstraction de toute dépendance à l'égard des systèmes d'exploitation. Ceci en considérant les périphériques et les logiciels comme des objets ou une combinaison d'objets indépendants pouvant communiquer. Ces ressources matérielles ou logicielles peuvent être connectées à un réseau pour se déclarer et fonctionner dès qu'elles sont branchées. Chaque client qui désire les utiliser peut les localiser et faire appel à elles afin d'exécuter certaines tâches. Le but de cette technologie, où la responsabilité a été transférée à Apache sous le nom de projet River, est de rendre un réseau plus dynamique pour mieux refléter la nature dynamique du travail d'équipe (collaboratif) en donnant les droits d'ajout ou de suppression des services d'une manière flexible.

Il est à noter que l'équipe Jini de Sun Microsystems a toujours expliqué que Jini n'est pas un acronyme. Autrement dit le terme Jini ne signifie pas par exemple "Java Intelligent Network Interface" ou "Java INIside", il se suffit à lui-même à tel point que l'équipe disait que "Jini Is Not Initials"

IV.4.1. Les composants de Jini

Les composants de Jini sont une extension réseau de l'infrastructure, du modèle de programmation et des services de la technologie Java (cf. Tableau 6)

- L'infrastructure est l'ensemble de composants qui permet d'établir un système fédéré de Jini. Ceci inclut le service de recherche ou de localisation "Lookup Service" ainsi que le protocole d'enregistrement et de découverte (Discovery/Join Proocol)
- Le modèle de programmation est un ensemble d'interfaces qui permet la construction des services fiables incluant ceux qui font partie de l'infrastructure et ceux qui se joignent à la fédération. Ce modèle peut intégrer la notion de bail et les événements distants. Il peut compter sur la possibilité de Java pour déplacer des objets à travers les machines virtuelles (JVM).
- Les services sont des entités dans la fédération. Ils utilisent l'infrastructure pour la communication, la découverte et l'annonce de leur présence.

La séparation entre ces catégories, qui s'appuie sur la distinction, Java n'est pas toujours claire puisque les composants de chaque catégorie sont interdépendants.

Tableau 6 Les composants Jini

	Infrastructure	***Modèle de programmation***	***Services***
Base Java	Java VM RMI Java Security	Java APIs Java Beans …	JNDI Entreprise Beans …
Java + Jini	Discovery/join Distributed Security Lookup	Leasing Transcations Events	Printing Transacation Manager …

Dans notre approche, tous les agents sont considérés comme des composants Jini pour bénéficier de l'infrastructure, les modèles de programmation et aussi les services. Cette considération va leur permettre d'établir une fédération d'agents ce qui rejoint la notion de la communauté.

IV.4.1.1. Services

La notion de service est le concept le plus important dans l'architecture Jini. C'est son utilisation qui permet au client d'exécuter des traitements, utiliser une unité de stockage et communiquer… Le système Jini se compose de services qui peuvent être regroupés pour exécuter des tâches particulières. Un service peut utiliser un autre service. Autrement dit, un service peut être le client d'un autre service tout en utilisant son propre service. La nature dynamique du système Jini permet à des services de s'enregistrer ou de se retirer d'une fédération de services à tout moment selon la demande, le besoin, ou les conditions du groupe de travail qui les utilise [119].

Un service Jini est défini par une interface Java. Et peut être identifié par cette même interface. Il peut être implémenté de différentes façons. Toutefois, le client et les différentes implémentations semblent identiques parce qu'ils implémentent la même interface. D'un point de vue global, le fournisseur du service doit implémenter l'interface du service, enregistrer l'instance du service dans le service de recherche (Lookup Service) pour qu'il soit visible dans le réseau et gérer le cycle de vie du service en restant en attente. Ceci est similaire à la création d'un objet distant basée sur l'appel des méthodes distantes (RMI). En réalité Jini utilise RMI comme protocole de communication. Quant aux

transmissions de données, elles se font à travers un système de couches comme le montre le schéma suivant.

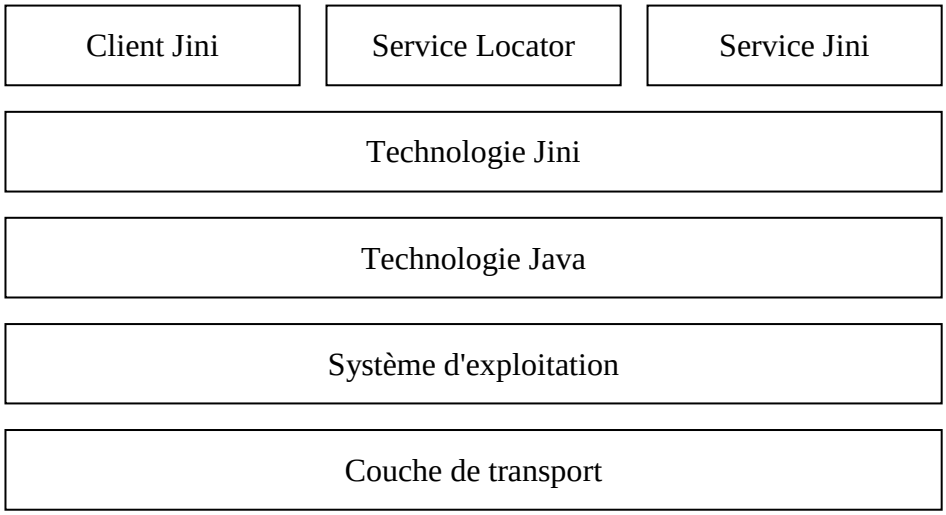

Figure 17 Architecture de Jini

L'interface d'un service doit hériter de l'interface *Remote* qui sert à créer le lien avec le système Jini et à identifier les interfaces dont les méthodes peuvent être appelées depuis une machine virtuelle distante (soit sur la même machine, soit sur une machine différente).

Il faut noter que tous les agents ainsi que les tâches sont des services Jini. Le diagramme de classe suivant présente le lien entre notre modèle et le système Jini.

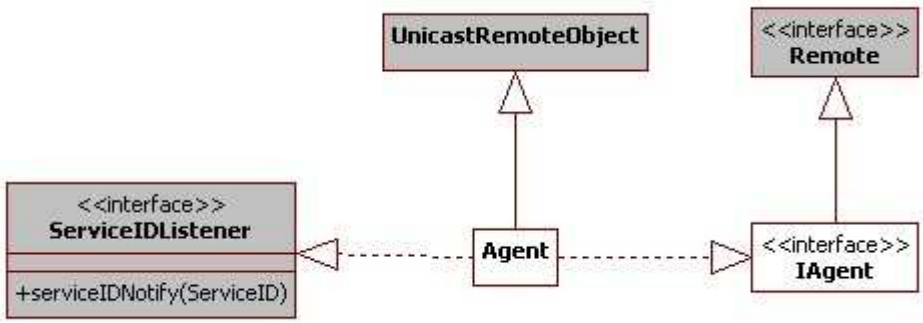

Figure 18 Diagramme de classe du service Jini

L'instance du service peut être sérialisée et transmise au client. Seulement, dans la plupart des cas, cette dernière dépend des ressources locales (ex. CPU, fichier, imprimante…). C'est pour cette raison qu'elle doit rester active sur la JVM locale. Certains pensent que c'est un problème. Mais, il se résout en introduisant le modèle de structure "Proxy" [120], un modèle très utilisé pour la gestion d'objets distribués (ex : RMI). L'idée étant de

communiquer avec des objets distants sans que le client fasse de différence entre un accès local et un accès distant.

IV.4.1.2. Lookup service

Une fois le service créé, il faudra le publier pour qu'il soit visible sur le réseau. Contrairement à une application client/serveur où le client accède directement au serveur, Jini est une architecture intermédiaire (Broker architecture). Une telle architecture est plus appropriée pour les environnements dynamiques comme la fédération de Jini où un service et un client adhèrent et se désabonnent fréquemment. Le service de découverte est centralisé via le "Lookup Service" mais par la suite la communication entre le client et le service est directe.

Le "Lookup Service" est le mécanisme central de la fédération de Jini. Il fournit le point de contact entre le fournisseur de service et le client.

Pour localiser le "Lookup Service", le client et le fournisseur du service utilisent le mécanisme de découverte. Cette découverte peut se faire de deux façons différentes : si l'emplacement du Lookup service est connu, alors le client utilise "Unicast discovery" pour se connecter directement. Dans le cas contraire, il envoie une requête "Multicast discovery" pour rechercher dans le réseau un ou plusieurs "Lookup Service".

Quand une requête est envoyée au "Lookup service", il retourne un objet de type "Registrar" au client. Cet objet a le rôle d'un "Proxy". Et, toutes les requêtes destinées au "Lookup service" passent par lui.

Il faut savoir qu'avec le "Lookup Service", nous ne pouvons réaliser que deux opérations en fonction du rôle :

Adhésion (Join) : le fournisseur de service peut adhérer à une fédération de services enregistrée dans un "Lookup Service". Cette opération est effectuée par l'enregistrement du service ou son "Proxy" et certains attributs décrivant le service (Ex : nom du service, description, version…). Ces informations sont stockées dans le "Lookup Service". Le processeur d'enregistrement se termine par le retour d'un objet "ServiceRegistration" qui peut être utilisé pour modifier les attributs du service.

Recherche (Lookup) : un client récupère le service ou son Proxy en utilisant le mécanisme de recherche du "Lookup Service". Dans un premier temps, il construit un modèle basé sur l'interface du service et divers attributs pour identifier le service voulu.

Quand le "Lookup Service" reçoit le modèle, il recherche dans tous les services enregistrés puis retourne le ou les services qui correspondent au modèle.

En plus de l'adhésion et de la recherche, le "Lookup Service" peut également notifier le client si un service venait d'adhérer ou de se désabonner de la fédération via le mécanisme des événements distants.

Le schéma suivant montre en détail le déroulement de ce processus.

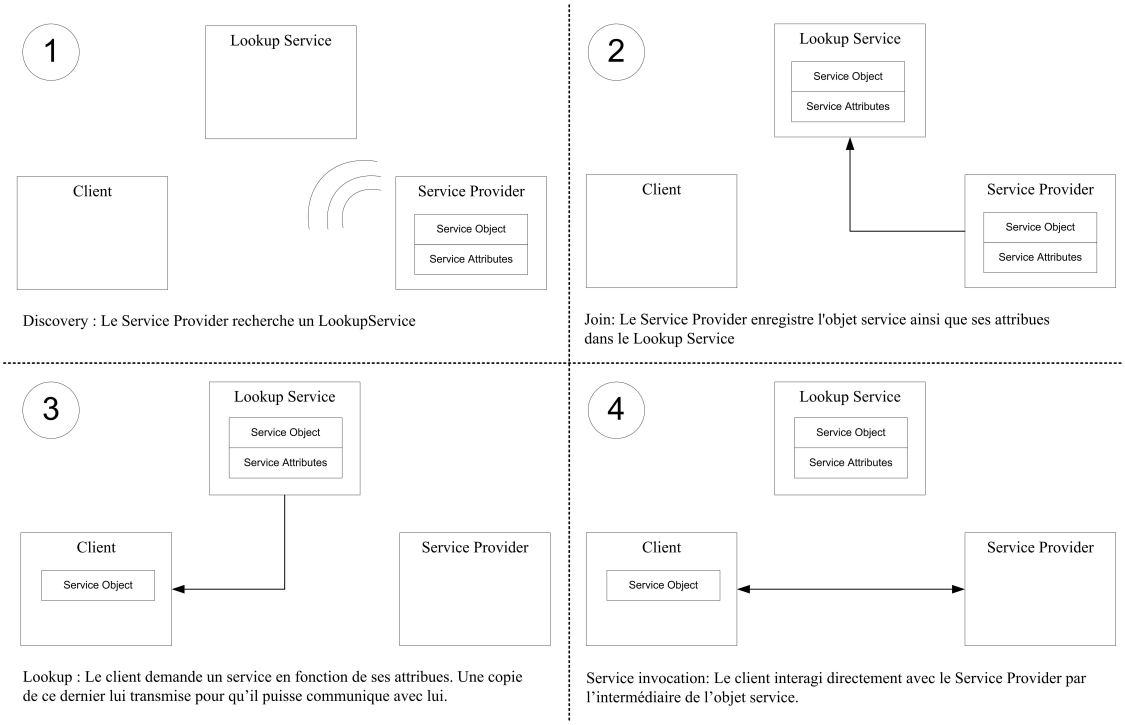

Figure 19 Protocoles de découverte et de recherche

Les agents implémentent l'interface *DiscoveryListener*, comme le montre le diagramme de classe suivant, pour disposer du mécanisme de découverte. Chaque agent, lors de sa création, crée son propre service de découverte puis il se publie dans ce dernier.

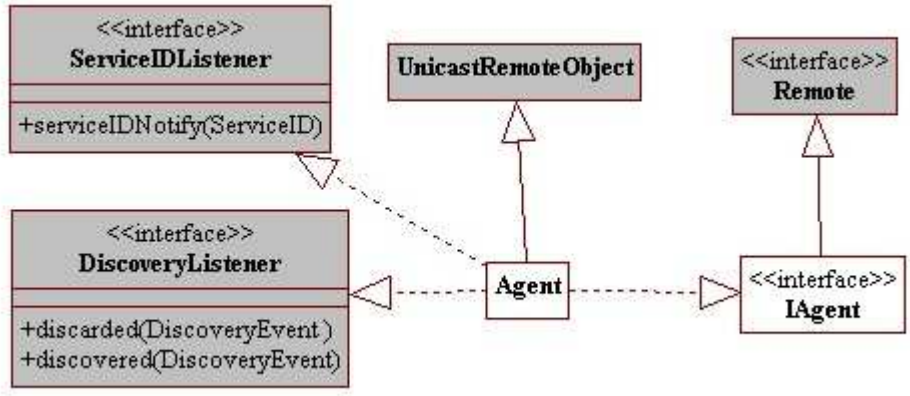

Figure 20 Diagramme de classe du service de découverte

IV.4.1.3. Bail

Le mécanisme du bail (Leasing) est utilisé en deux applications pour donner un accès aux ressources durant une période convenue.

La plupart des services dans une fédération Jini utilisent le bail pour autoriser l'accès. Chaque bail est négocié entre l'utilisateur et le fournisseur du service. Le service est sollicité pour une période, son accès est accordé pour la même période, vraisemblablement cette période doit être prise en compte. Si le bail n'est pas renouvelé lorsqu'il expire cela s'explique par le fait que la ressource n'est plus utilisée, qu'une défaillance s'est produite, ou que le renouvellement n'est pas permis. Alors le client ou le fournisseur peuvent conclure que la ressource doit être libérée.

En plus du contrôle d'accès au service, le bail peut notifier au service que le client est toujours actif.

Dans notre modèle, la notion de bail n'est pas abordée mais elle est importante dans le cas où la stratégie de l'application type impose des créneaux horaires pour la visite d'un site. Par exemple les collectes sur le *SrvWeb* peuvent se faire entre 01 h 00 et 05 h 00, pour cela l'*AgentHost* du *SrvWeb* doit adhérer à son propre service de découverte pour une durée déterminée.

Dans Jini, le "Lookup Service" retourne un bail quand le service est enregistré. Ainsi, le fournisseur du service doit renouveler le bail, ce qui fait que le service reste adhérent de la fédération. Hormis l'utilisation de "Lookup service", les baux sont aussi utilisés pour les

événements distants. L'adhèrent reçoit un événement à chaque fois que le bail est renouvelé.

IV.4.1.4. Transaction

Les transactions sont des fonctionnalités nécessaires voir primordiales pour des opérations distribuées. Fréquemment, deux objets ou plus doivent synchroniser le changement d'un état, s'il a eu lieu. Ce changement peut se produire dans plusieurs situations, telles que la mise à jour d'une information, où une partie des objets peuvent la faire au même temps que d'autres sont incapables de l'effectuer, en conséquent le résultat final sera incohérent.

La théorie de transactions est souvent associée à la propriété ACID :

- **Atomicité (Atomicity) :** Toutes les opérations d'une transaction doivent avoir lieu, ou non.
- **Cohérence (Consistency) :** L'accomplissement d'une transaction doit laisser les participants de cette dernière dans un état logique. Par exemple, une donnée doit rester identique.
- **Isolation :** Les activités d'une transaction ne doivent affecter aucune autre transaction.
- **Durabilité (Durability) :** Les résultats d'une transaction doivent être persistants.

Les transactions utilisent un protocole de validation (commit protocol) biphasé. Ceci implique que les participants à une transaction soient invités à voter pour cette dernière. Si tous les participants sont d'accord alors la transaction est validée, dans le cas contraire elle est annulée pour l'ensemble des participants. Il incombe aux clients et aux services dans une transaction d'observer les propriétés ACID s'ils le désirent. Jini fournit essentiellement le mécanisme biphasé, et lègue les règles de signification aux participants de la transaction.

Les participants à une transaction peuvent disparaître et il en est de même pour le gestionnaire des transactions. Dès lors, les transactions sont contrôlées par un bail, qui expirera à moins qu'il soit remplacé.

Chaque agent doit pouvoir participer à une transaction dans le cadre de leurs activités afin de s'assurer que les données partagées ou informations échangées sont valides. Cela implique l'implémentation de l'interface *TransactionParticipant*, comme le montre le diagramme de classe suivant, pour utiliser le mécanisme Jini gérant les transactions.

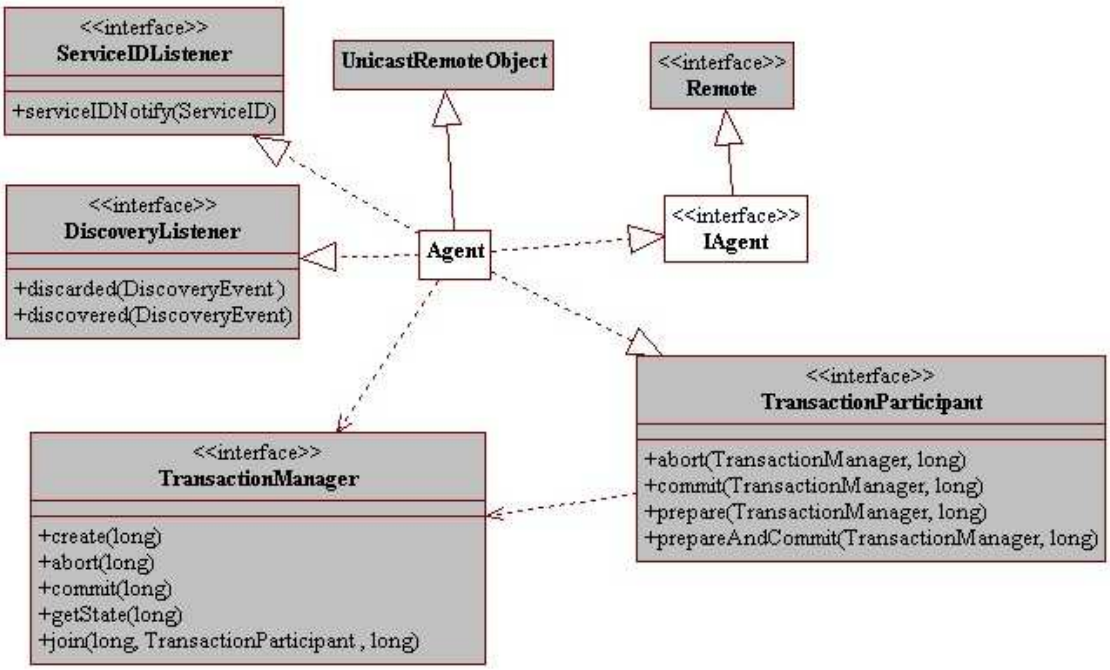

Figure 21 Diagramme de classe des transactions

IV.4.1.5. Remote Events

Le mécanisme des événements est utilisé en liaison asynchrone dans Jini. C'est un mode crucial à l'utilisation efficace de la technologie. Il est intéressant et précieux pour considérer la nature distribuée des événements. C'est pour cette raison que le modèle existant de délégation d'événements n'est pas utilisé.

Un objet Jini peut s'intéresser aux changements d'un autre objet Jini. Il désire aussi écouter quelques changements. Un objet peut autoriser d'autres objets à s'intéresser à lui et les notifier à chaque changement. Plusieurs modèles d'événements existent, notamment le modèle de délégation en Java, pour répondre à ce besoin. Toutefois, la nature réseau de Jini et la considération de la nature distribuée des événements ont mené vers un modèle particulier, différent du modèle standard pour les raisons suivantes :

- Le réseau n'est pas fiable alors la délivrance des messages ne peut pas être garantie. D'autant plus que les méthodes synchrones, utilisées dans le modèle d'événement Java, nécessitent une réponse. Ce qui pourrait ne pas fonctionner.
- La latence du réseau peut engendrer des décalages de temps entre la propagation de l'événement et sa notification. Alors l'état de l'objet risque d'être incohérent d'un écouteur à l'autre.

- Le veilleur distant (Listener) peut disparaître à cause de son bail au moment de l'événement.

Jini utilise un seul type d'événement qui se résume en une simple classe *RemoteEvent* (cf. Figure 22). Cette classe contient le strict minimum d'informations pour réduire les échanges réseau afin d'éviter la latence et la surcharge. L'objectif de l'utilisation des événements est de notifier des applications ayant pour but de monitorer l'activité des agents. Nous pouvons également, utiliser les événements pour que les agents se notifient entre eux pour certains changements d'état, l'inconvénient est que notre modèle sera lié à une technologie chose qui n'est pas souhaitée d'autant plus la classes *RemoteEvent* ne permet pas des échanges riches.

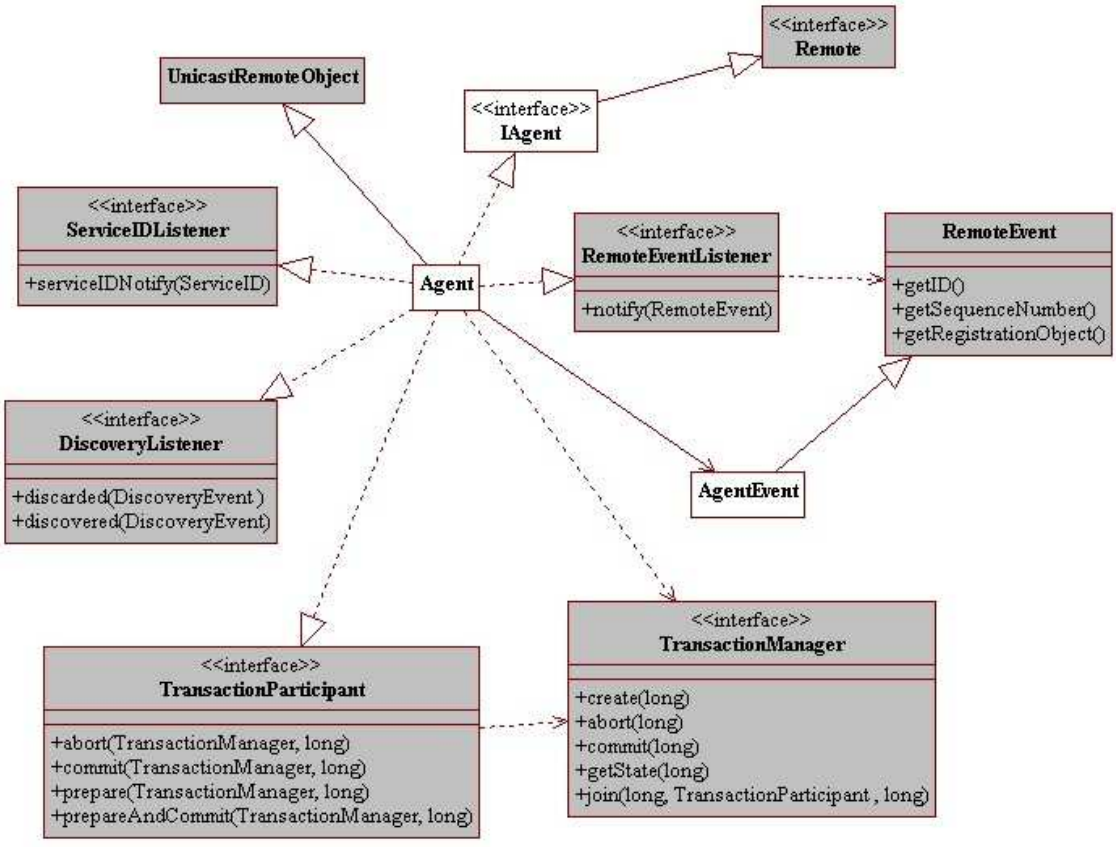

Figure 22 Diagramme de classes des événements

Jini ne spécifie pas comment le client doit s'enregistrer pour recevoir les événements distants. Le développeur du service doit prévoir une méthode pour écouter l'adhésion. Néanmoins, Jini ne spécifie pas une classe qui puisse être utilisée comme type de retour de la méthode d'écoute.

IV.4.1.6. Communication Protocol

Jini fournit une grande capacité de communication entre les différents composants de son architecture incluant des protocoles ouverts et finis, et définissant un certain nombre d'entre eux. Dans les versions précédentes de Jini (version 1), Jini utilise le protocole Java Remote Method Protocol (JRMP), lui-même basé sur TCP/IP comme protocole de communication. Le JRPM est une implémentation Java de RMI [121] mais son concept est différent de celui de Jini. D'où le développement de Jini Extensible Remote invocation (JERI) qui permet une importante flexibilité et résout quelques lacunes de RMI. En voici quelques exemples :

- Le RMI force la génération des souches (Stubs) au moment de la compilation. Contrairement à JERI qui le génère automatiquement quand un objet est exporté. Ainsi, il réduit le temps d'exécution.
- JRMP utilise seulement les connexions TCP, JERI offre d'autres possibilités.
- Un ramasse-miettes distribué (DGC) plus flexible que celui de RMI.
- Une personnalisation beaucoup plus grande. Un développeur peut utiliser sa propre implémentation du protocole de transport ou disposer d'autres couches de JERI.

IV.4.1.7. Configuration

Pour construire et maintenir les services, plusieurs paramètres sont nécessaires (ex. attributs du service, l'url du chargement dynamique…). Ces paramètres sont fréquemment spécifiés en dur dans le code source. Ce qui implique une recompilation à chaque fois que nous les changeons. Pour cette raison, nous pouvons utiliser la classe Configuration qui permet de spécifier les paramètres à l'exécution.

Une configuration peut être utilisée de différentes façons. Le développeur peut, par exemple, implémenter sa propre configuration, pour rechercher un objet de configuration d'une base de données ou par l'intermédiaire d'un canal bloqué. Mais, la plupart du temps, nous utilisons la classe *ConfigurationFile* pour spécifier la configuration dans un fichier. La syntaxe de ce dernier est basée sur le langage Java avec un minimum d'instructions sans itérations ni conditions :

- Affectation de variables
- Création d'objets

- Les méthodes statiques à invoquer.

Un développeur est libre de décider quelles propriétés peuvent être stockées dans un fichier de configuration.

Le code ci-dessous montre un exemple d'un fichier de configuration du Lookup Service. Il permet de définir les paramètres nécessaires pour construire une instance *NonActivatableServiceDescriptor*. Les paramètres représentent respectivement l'*url* du chargement dynamique, le fichier des privilèges, le *classpath* du service, le nom de la classe, les arguments de configuration du serveur qui peuvent être aussi un autre fichier de configuration.

```
import com.sun.jini.start.NonActivatableServiceDescriptor;
import com.sun.jini.start.ServiceDescriptor;
import com.sun.jini.config.ConfigUtil;
com.sun.jini.start {
   private static policy = "priv.policy";
   private static host = ConfigUtil.getHostName();
   private static port = "80";
   private static jskdl = " http://" + host + ":" + port + "/jsk-dl.jar";
   serviceDescriptors = new ServiceDescriptor[]{
      new NonActivatableServiceDescriptor(
         "http://" + host + ":" + port + "/reggie-dl.jar" + jskdl,
         policy,
         "C:\\jini2_1\\lib\\reggie.jar",
         "com.sun.jini.reggie.TransientRegistrarImpl",
         new String[] { "transient-reggie.config" })
   };
}
```

IV.4.2. La sécurité

La sécurité joue un rôle très important dans les systèmes distribués qui sont vulnérables en vue de leur accessibilité via les réseaux. Dans le cas de Jini, le chargement dynamique des classes et des objets mobiles introduit de nouveaux risques. D'autant plus que la sécurité de base de Jini est gérée par la sécurité Java qui consiste, grâce au Security Manager, à refuser tout chargement de code distant ou exécution d'opération jugée non fiable si nous ne disposons pas de privilèges nécessaires. Généralement, ces privilèges peuvent être accordés sous la forme d'un fichier dit de politique de sécurité, mais cela n'est pas

suffisant. Dans le cas où nous autorisons le chargement d'un code distant qui risque d'être malveillant, pour un système d'environnement dynamique, tel que Jini, où les services sont découverts dynamiquement et leurs codes téléchargés puis exécutés.

Le plus grand risque de la sécurité dans un système basé sur Jini est le téléchargement et l'exécution potentielle d'un code malveillant provenant d'un hôte distant. Pour pallier à cela, nous pouvons définir une politique de sécurité afin de spécifier les hôtes distants de confiance ou des signatures numériques de confiance. Ce modèle de sécurité n'est pas suffisant pour un environnement dynamique, tel que Jini, où des services sont dynamiquement découverts et leur code est téléchargé puis exécuté. Si toutes les permissions de sécurité doivent être spécifiées dans un dossier de politique, ceci présente des restrictions pour un système dynamique. Néanmoins, ces risques peuvent être décentralisés [122] par différents services et/ou mécanismes comme l'authentification, l'intégrité, la confidentialité, la contrainte de sécurité et l'attribution dynamique des droits.

IV.4.2.1. Authentification

Ce système d'authentification consiste à vérifier l'identité de l'individu avant de lui donner les droits d'accès tout en tenant compte qu'un individu peut avoir plusieurs rôles. Un sujet s'authentifie à un service en utilisant des informations pour être reconnus en tant qu'un *Principal*. Par exemple, un nom d'utilisateur est un attribue d'un *Principal* et le mot de passe sert à la vérification. Ce concept est connu sous le nom de Java Authentication and Authorization Service (JAAS) [123] une API standard Java qui augmente son modèle de sécurité en permettant de gérer des identifications et les droits associés, par rôles, au niveau du client et du serveur d'application. JAAS apporte une solution souple et pratique pour sécuriser des composants J2EE, que ce soit des EJB ou des applications Web. Cependant, des solutions alternatives existent, basées sur les mêmes principes de configuration externe, avec le même fonctionnement modulaire. Des solutions comme Acegi Security [124] peuvent offrir une souplesse plus importante.

IV.4.2.2. Intégrité

Durant le transport, les classes et les données risquent d'être altérées par une entité malveillante. Selon Robert William Scheifler[3], ce problème peut être réglé en intégrant un

[3] Robert William Scheifler est un informaticien né en 1954. Nous lui devons le développement du système X windows. Il a été aussi l'architecte de Jini à Sun Microsystem.

mécanisme d'intégrité [125]. Généralement, si nous utilisons le déploiement standard de Jini, la majorité utilise le protocole *http* pour charger les classes. Le problème avec cette pratique, c'est l'impossibilité au protocole de garantir l'intégrité. De ce fait, si quelqu'un attaque le serveur, il peut, par exemple, remplacer un service par un autre malveillant qui porte le même nom. Il est imaginable qu'un tiers puisse intercepter le trafic et modifier les données.

Une manière possible de télécharger un code intègre est d'utiliser le protocole HTTPS mais cela a quelques inconvénients. Parce que nous sommes obligés de garder le code crypté, le but est d'assurer l'intégrité et non pas de restreindre l'utilisation. D'autant plus que nous pouvons toujours récupérer la clé privée.

Une autre alternative, jugée plus sûre, est d'utiliser un nouveau schéma d'url nommé HTTPMD, disponible depuis la version 2 de Jini. Ce nouveau schéma, issu du projet Davis de Sun Microsystem, consiste à inclure dans l'url http une empreinte numérique (Message Digest) des données à charger. Quand le client charge un fichier, il compare son empreinte avec celle référencée dans l'url. C'est un schéma conçu spécialement pour les fichiers JAR et ne prend pas en compte les répertoires. Car il n'y a aucun moyen connu, à ce jour, pour déterminer leur empreinte.

L'avantage d'utiliser HTTPMD est que ce dernier utilise le protocole HTTP fournit par un simple serveur web sans extensions ou contraintes de cryptage par rapport au protocole HTTPS. En plus, les informations ne sont pas cryptées. Ce qui n'a pas d'impact sur les performances.

IV.4.2.3. Confidentialité

Ce mécanisme permet de s'assurer que l'information est accessible seulement à ceux autorisés à avoir un accès. Typiquement, un chiffrage est utilisé pour s'assurer que ses messages ne peuvent pas être lus par d'autres personnes.

Dans le système Jini, nous pouvons également décrire des contraintes de sécurité [126], basées sur les précédents mécanismes (authentification, confidentialité et intégrité) pour décrire le genre de sécurité du réseau souhaitée.

IV.4.2.4. Contrainte de sécurité

Les contraintes sont injectées dans le Proxy par le client avant de le recevoir et par le service après son exportation. Par exemple, quand un client télécharge le Proxy, le service

peut restreindre les invocations des méthodes à certains clients. Le service peut également exiger que le client doive s'identifier avant d'invoquer des méthodes, si le client refuse, alors l'accès au Proxy lui sera refusé. De la même manière, le client peut placer des contraintes dans le Proxy : il peut exiger une authentification avant d'utiliser le service, ou il peut insister à ce que le Proxy garantisse l'intégrité des données qu'il doit transmettre au service.

Les contraintes d'invocation spécifient des contraintes sur le comportement. Elles ne spécifient pas comment une contrainte est implémentée, mais de quelle contrainte il s'agit. Les contraintes d'invocation sont divisées en deux parties :

- **Conditions** : sont des contraintes qui doivent absolument être satisfaisantes.
- **Préférences** : sont des contraintes qui devraient être satisfaisantes. Prenons l'exemple de la contrainte de préférence "anonymat" : un client désire rester anonyme mais si le serveur exige une authentification alors la contrainte d'anonymat ne sera pas satisfaisante.

IV.4.2.5. Attribution dynamique des droits

Le modèle de sécurité standard de Java accorde des permissions à un ensemble de codes (code base) ou à des URL, mais parfois l'emplacement des classes des services distants est inconnu jusqu'à ce que le client les utilise. Nous devons alors, accorder de façon dynamique des permissions, sauf que le modèle de sécurité Java ne peut pas satisfaire cette demande car il ne peut pas garantir l'attribution des permissions. Par ailleurs, ces dernières sont accordées pour protéger un domaine constitué de trois composants:

- L'acteur principal qui exécute le code.
- Le code source.
- Le chargeur de classes (class loader).

Dans un contexte mobile, il est nécessaire d'attribuer dynamiquement des permissions. Ainsi l'utilisation de Jini nous permet-elle de le faire [127].

Dans ce dernier point, nous avons présenté la technologie JINI et les mécanismes de sécurité choisis pour l'implémentation de notre approche mobile. Cette présentation nous a permis de mettre en évidence certains aspects utilisés pour l'application de notre prototype en terme de surveillance que nous développons dans le chapitre suivant.

V. Stratégie de surveillance

Dans ce chapitre, nous présentons l'application de notre approche mobile à la surveillance logicielle. Pour cela, nous définissons dans l'ordre l'architecture de l'approche, la stratégie pour analyser les données collectées, les interactions entre les agents de la communauté et la structure des données de l'approche. Enfin, pour terminer nous exposons les résultats et les limites de notre travail.

V.1. Architecture

Notre solution est basée, principalement, sur cinq composants dont quatre mobiles. Les composants mobiles sont conçus à partir du même modèle défini précédemment dans le chapitre IV. Certes, ils ont des structures identiques puisqu'ils héritent tous de la classe Agent (cf. Figure 23) néanmoins ils ont différentes couches en fonction de leur domaine fonctionnel. Les noms de ces agents ont été changés par rapport à notre modèle pour garder une cohérence avec leur domaine d'activité.

Le premier composant, nommé *AgentCollector*, est dévoué à réaliser deux types de tâches : la collecte des données, sa tâche initiale d'où son nom. Et des tâches d'actions, une évolution apportée au cours de nos travaux. Cet agent est l'équivalent du *MobileAgent*.

Le deuxième composant *AgentAnalyzer* est considéré comme le cerveau du système parce qu'il analyse les données collectées par le premier et lui transmet les tâches d'actions à effectuer. En réalité il n'est pas intelligent et son processus d'analyse est basé sur des règles définies par l'utilisateur. Ce processus va être détaillé dans le point V.3. Cet agent est similaire à *l'AgentServer*.

Le troisième composant est *AgentDirectory*, c'est un annuaire regroupant les informations de localisation de l'ensemble des composants de la solution (y compris les autres *AgentDirectory*).

Il faut savoir que ces trois composants sont totalement indépendants les uns des autres. Puisque l'exécution de l'un d'eux peut se faire sans la présence des autres. Seulement les trois sont très connexes et sont essentiels tout le long du cycle de vie du processus.

Le quatrième composant nommé *AgentMessenger* assure la communication entres les autres composants, sa principale tâche consiste à transmettre des messages.

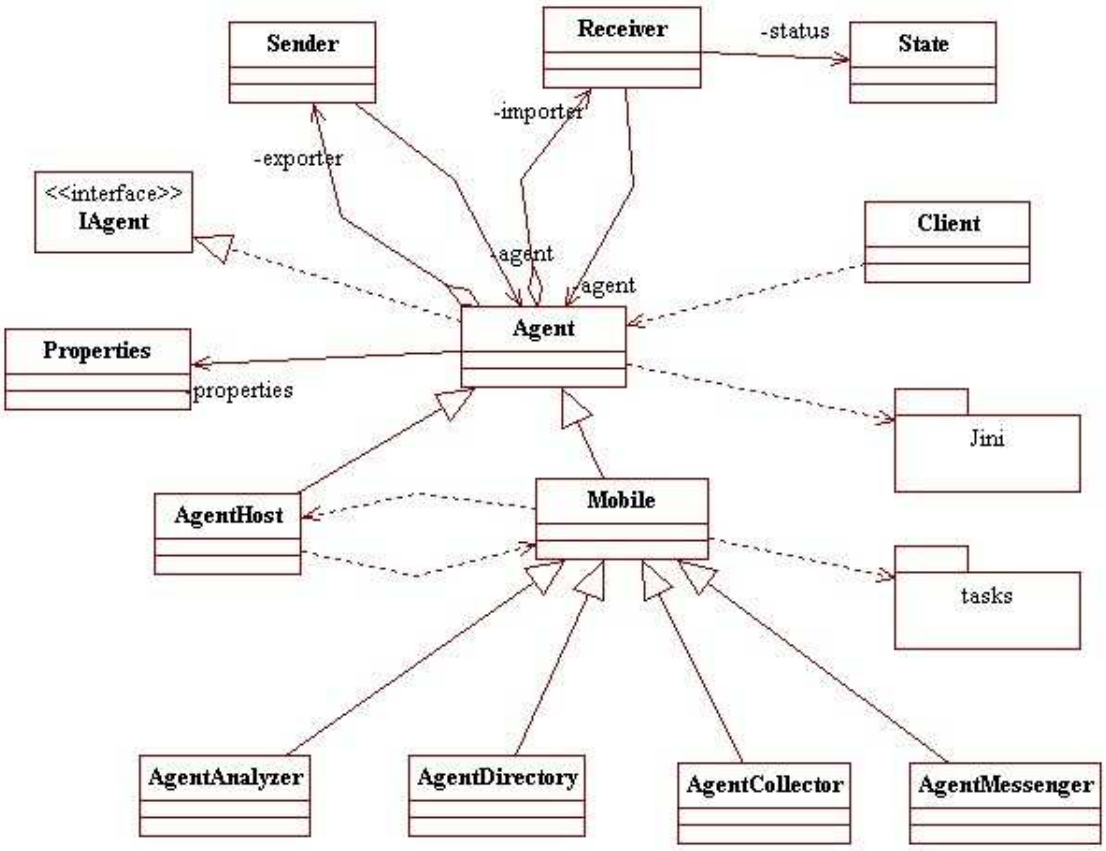

Figure 23 Diagramme de classe de la surveillance logicielle

Initialement, les composants peuvent communiquer sans recourir à un intermédiaire mais l'idée d'intégrer un composant tiers est pertinente. Nous estimons qu'un composant qui veut transmettre une information (configurations, taches, données…) à un autre n'a pas à se déplacer vers la machine de destination pour le faire ni à le transmettre directement sur le réseau. D'autant plus ce composant peut intégrer un algorithme de cryptage, par exemple, au lieu que ce soit tous les autres composants qui implémentent le même algorithme chose qui rend leur évolution délicate.

Il faut reconnaître que dans certains cas où les agents doivent communiquer avec des services Jini, tel que le *ServiceRegistrar*, ne pouvant pas recevoir des agents, cette communication s'effectue d'une manière standard.

Pour terminer, le dernier composant *AgentHost* est un service unique par machine. Il peut être assimilé, en terme de programmation Java, à un *ClassLoader* parce qu'il gère le chargement des agents mobiles. En plus il possède un service de localisation pour la gestion d'accès à l'annuaire lors de son déplacement.

Enfin, il faudra noter que l'ensemble des composants de la solution sont des services Jini.

V.1.1. AgentCollector

Le composant *AgentCollector* est composé de trois couches interdépendantes. Ces couches sont : mobilité, activité locale et communication.

V.1.1.1. Couche mobilité

Cette couche est commune à tous les agents mobiles, elle gère la création de l'agent et leurs déplacements au sein du réseau. Cette gestion comporte trois parties :

Configuration du service : C'est une implémentation du service Configuration de Jini. Elle permet la lecture des paramètres nécessaires à la création de l'agent mobile. Initialement un service Jini possède des informations génériques liées à son identité (nom, constructeur, modèle, version, numéro de série, fournisseur…), à son export dans le réseau et à sa création. Ces informations peuvent être enrichies par d'autres informations pour adapter le service à notre besoin. Dans notre cas nous allons ajouter des informations nécessaires à la création automatique du LookupService (description du service) et ajouter la notion de groupes pour pouvoir faire la différence entre les divers types d'agents.

Toutes ces informations sont stockées dans des fichiers de configuration Jini et peuvent être modifiées en cours d'exécution des agents via une interface graphique.

Localisation du service : Cette partie de la couche mobilité gère la création automatique du service LookupService pour qu'il s'auto publie dans ce dernier. Ce service contient deux protocoles : de découverte (discovery process) et d'adhésion (join process) qui permettent respectivement de découvrir un service Jini et de s'enregistrer dans une fédération de services Jini. Pour qu'un agent soit visible sur le réseau il doit obligatoirement joindre à un LookupService et pour que les autres agents puissent le contacter ils doivent le rechercher dans le même service précité.

Comportement mobile : Cette partie gère la création du service de l'agent mobile et l'implémentation des méthodes distantes de ce dernier. En plus elle est composée de deux sous processus, comme cela a été précédemment défini dans notre approche mobile. Un premier sous processus Receiver qui permet de réceptionner les AgentMessenger. Le second sous processus Sender gère la migration de l'agent vers un autre hôte.

V.1.1.2. Couche d'activité locale

Cette couche varie d'un agent mobile à un autre en fonction de son domaine fonctionnel. Pour le *AgentCollector*, elle permet d'effectuer les traitements locaux. Ces traitements consistent à gérer l'interaction entre les différentes couches, l'exécution des tâches reçues transmises par *AgentAnalyzer* via le *AgentMessenger* et la transmission des résultats des tâches effectuées.

Pour que l'exécution des tâches soit dynamique et sécurisée, les tâches ainsi que les droits nécessaires à leur exécution ne sont pas préalablement connus. En effet, les tâches sont des services Jini publiés dans le *AgentDirectory* et comme pour tout service Jini, ils peuvent être configurés en leur fournissant des fichiers de privilèges utiles à leur fonctionnement. Les privilèges sont associés à la tâche et attribués dynamiquement par le mécanisme Jini.

V.1.1.3. Couche de communication

Cette couche est commune aux agents : *AgentCollector*, *AgentAnalyzer* et *AgentDirectory*. Elle permet de gérer les échanges d'informations entre eux. Ces informations peuvent être : une demande d'exécution d'une tâche, la transmission du résultat partiel ou complet d'une tâche et la notification de changement de localisation en cas de déplacement de l'agent. Pour le *AgentCollector*, il peut s'enregistrer auprès de l'agent annuaire et le notifier de sa migration vers un autre hôte. Il peut, également, envoyer les résultats des tâches à l'agent *AgentAnalyzer*.

L'enregistrement auprès de l'*AgentDirectory* se fait d'une manière synchrone, de ce fait cette opération est réalisée par un appel distant, sinon les échanges d'informations sont opérés par le *AgentMessenger*. Chaque agent mobile désirant communiquer avec l'autre doit créer un *AgentMessenger* en lui fournissant l'information à transmettre ainsi que les informations relatives à la localisation du destinataire. Par ailleurs cette couche gère l'interprétation des messages encapsulés dans les agents messagers quand ils attendent leurs destinations.

V.1.2. AgentAnalyzer

Le composant *AgentAnalyzer* est composé de quatre couches, dont trois couches identiques à celle de l'agent collecteur avec certaines différences pour la couche d'activité locale et la couche de communication. La nouvelle couche de ce composant est la couche report.

V.1.2.1. Couche communication

La différence avec la couche de l'agent collecteur est que l'agent analyseur envoie à l'agent collecteur des tâches à effectuer, et non les résultats des tâches. En ce qui concerne l'enregistrement, la notification et l'interprétation des messages reçus, ils restent inchangés.

V.1.2.2. Couche d'activité locale

Pour le *AgentAnalyzer*, cette couche permet d'effectuer les traitements locaux. Ces traitements consistent à gérer l'interaction entre les différentes couches, la construction des tâches à transmettre à l'agent collecteur via l'agent messager, la sauvegarde des données reçues et leurs analyses. La stratégie d'analyse et les règles définissant la politique de l'agent analyseur seront détaillées dans le chapitre suivant.

V.1.2.3. Couche de report

Cette couche est un système de notification multi canal qui permet de transmettre des informations à l'utilisateur. Ces informations sont des événements, erreurs ou des alertes pouvant être envoyés à l'utilisateur par mail, message réseau, SMS ou tout simplement sauvegardés dans un support pour une prochaine consultation.

V.1.3. AgentHost

Ce composant immobile est unique par machine et indispensable. Il fournit l'environnement Jini pour accueillir les agents : analyseur, collecteur et annuaire. Si l'un de ces composants veut se déplacer d'un hôte à un autre, il le contacte pour en faire la demande puis il s'occupe de leur importation vers l'hôte où il s'exécute. En aucun cas, il n'interfère dans la mobilité, les tâches locales ou dans la communication des composants qu'il accueille.

Il comporte, également, un service d'écoute qui permet de découvrir dans un réseau la présence des services de localisation des agents annuaires. S'il n'est enregistré dans aucun agent annuaire et que ce dernier vient d'être créé, il est aussitôt notifié par un événement distant. Cette notification va lui permettre de récupérer la référence de l'agent annuaire et de s'enregistrer auprès de lui afin que le hôte, où il s'exécute, soit considéré comme disponible à accueillir des agents mobiles.

D’autre part ce service sert de référentiel aux autres composants mobiles (à l’exception de l’agent annuaire) dans le but de leur fournir des informations exactes sur l’emplacement de l’agent annuaire s’ils désirent le contacter.

V.1.4. AgentDirectory

L’agent annuaire est composé de quatre couches : couche monitoring, couche de communication et couche mobilité. Cette dernière couche est identique à celle des deux premiers agents.

V.1.4.1. Couche d'annuaire

Cette couche peut être assimilée à une couche d’activité locale parce qu’elle gère le domaine fonctionnel de l’agent annuaire. Elle permet de gérer la synchronisation et le référencement des informations de localisation des agents mobiles dans le réseau à l’exception de l’agent messager. Cette gestion consiste à enregistrer, modifier et/ou supprimer.

- ***Enregistrement*** : les agents mobiles doivent s’enregistrer auprès de l’agent annuaire pour collaborer au sein de la communauté. Dans un réseau nous pouvons éventuellement disposer de plusieurs agents annuaires. En conséquence, pour qu’ils puissent collaborer entre eux et échanger les informations de localisation des différents composants, ils doivent se reconnaître mutuellement. Enfin, l’échange entre les différents agents annuaires ne s’opère pas par un traitement de synchronisation mais par des événements distants (cf. § Remote Events). Lors de chaque opération de gestion (enregistrement, modification ou suppression) un événement distant est créé puis propagé aux autres agents annuaires pour qu’ils mettent à jour leurs informations.
- ***Modification*** : quand un composant mobil migre vers un nouvel hôte, il doit notifier son nouveau emplacement à l’agent annuaire par l’envoi d’un agent messager. Dans le cas où c’est un agent annuaire qui se déplace, son nouvel emplacement est notifié uniquement aux autres agents annuaires par un événement distant.
- ***Suppression*** : l’agent annuaire surveille, par un système d'écoute, les différents composants enregistrés auprès de lui. Quand l’un d’eux s’arrête, généralement cela

se produit quand l'hôte - où se trouve l'agent - s'arrête, alors l'agent annuaire le supprime de son référentiel puis notifie les autres agents toujours par propagation d'un événement distant.

V.1.4.2. Couche communication

La couche communication de cet agent intègre un système d'écoute comme pour l'agent hôte afin d'écouter les autres agents annuaires et de s'enregistrer auprès deux. En plus elle se charge, d'une part, de gérer le mécanisme d'événements distants (cf § Jini Remote Events). Cette gestion repose sur l'enregistrement des écouteurs (*Listener*), la création et la propagation des événements (*RemoteEvent*). D'autre part, elle fournit les méthodes distantes nécessaires pour retrouver les références des différents composants enregistrés dans l'annuaire.

Enfin, cette couche permet l'interprétation des messages de notification de changement de localisation reçus par les agents analyseurs et les agents collecteurs.

V.1.5. AgentMessenger

L'agent messager comme les autres agents mobiles dispose d'une couche mobilité identique à celle des autres. Mais lorsque ce composant veut se rendre sur un hôte afin de transmettre son message, il ne passe pas par l'agent hôte, ce qui est le cas pour les autres agents mobiles. Il va plutôt passer par l'agent qui désire contacter un agent collecteur, un agent analyseur ou un agent annuaire. Ce choix de passer directement par l'agent est motivé, d'une part, par la présence sur une machine de plusieurs agents, pouvant être du même type. D'autre part, nous évitons d'inclure un intermédiaire qui engendre la demande de la référence de localisation de l'agent hôte, surtout que cet intermédiaire risque de ralentir la transmission du message.

Il faut également noter que ce composant, ne notifie pas l'agent annuaire de son déplacement parce qu'il est considéré comme un agent transitoire : il est créé par un agent A pour atteindre un agent B et sera détruit par ce dernier. Dans le cas ou il ne peut pas atteindre l'agent destination il retourne vers l'agent expéditeur et sera détruit.

Le message encapsulé dans l'agent messager n'est pas un message formaté, définissant la tâche ou l'opération à effectuer, mais une instance d'une classe dérivée de la classe Message. Cette partie est détaillée dans le point V.5.4.

Pour terminer l'activité locale de ce composant, qui ne se résume pas seulement à transmettre des messages, mais intègre les fonctionnalités suivantes :

- Cryptage: cryptage et décryptage des résultats.
- Compression : compression et décompression des résultats.
- Déboguage : sauvegarder la trace de l'ensemble des opérations entre agents.

V.2. Vue d'ensemble

Ce point résume la description des fonctionnalités, la mobilité ainsi que le type et le mode de communication de l'ensemble des composants de l'approche (cf. Tableau 7). Il présente, également, un schéma (cf. Figure 24) représentant le mode d'interaction des différents composants.

Tableau 7 Vue d'ensemble des composants

Composant	***Fonctionnalité***	***Mobile***	***Communication***	
			Type	***Mode***
Analyzer	- Stratégie du déploiement - Analyse des données	Oui	- Appel distant - Messenger	- Synchrone - Asynchrone
Collector	Exécution des tâches	Oui	- Appel distant - Messenger	- Synchrone - Asynchrone
Directory	Annuaire des agents mobiles	Oui	- Événement distant	- Synchrone - Asynchrone
Host	- Migration des agents - Référentiel	Non	- Local - Événement distant	- Synchrone - Asynchrone
Messenger	Porteur	Oui	- Local	- Synchrone - Asynchrone

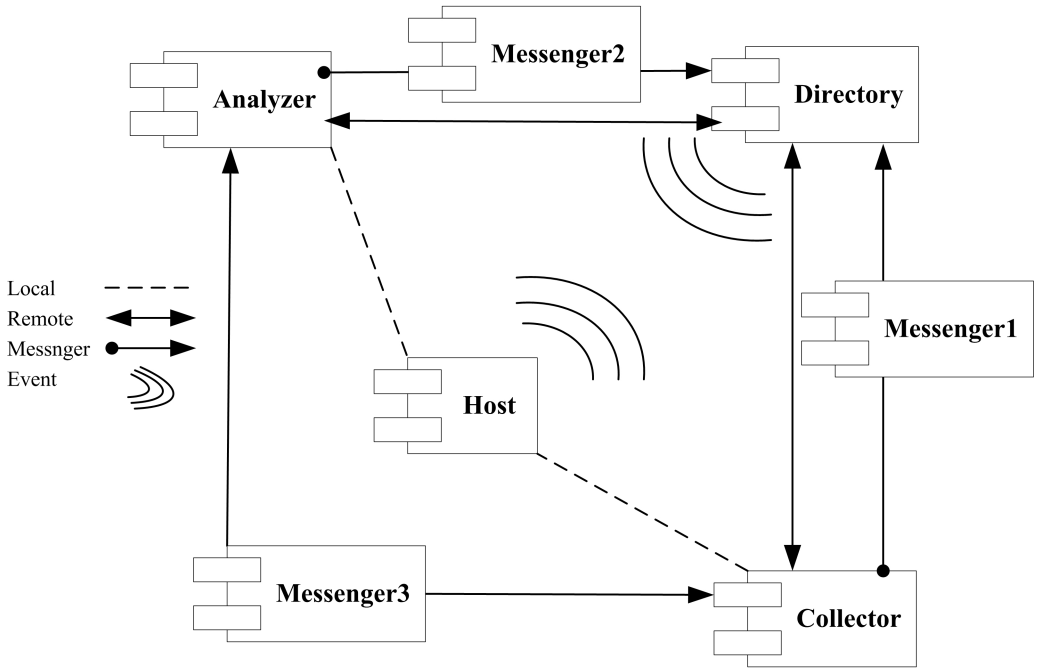

Figure 24 Mode de communication entre les composants

V.3. Stratégie d'analyse

La subtilité de la couche d'analyse est d'éprouver la façon optimale pour analyser les données collectées. Nous avons étudié des ébauches préalables pour trouver un moyen efficace.

Dans des études faites par T. Abbes, A. Bouhoula and M. Rusinowitch, ils présentent une approche pour analyser le trafic du réseau en utilisant une organisation intelligente de règle de détection [128]. Les mêmes auteurs utilisent la corrélation équilibrée dans un processus de détection d'intrusions [129], cette approche peut s'appliquer à d'autres secteurs si nous considérons qu'ils ont uniquement un seul type de source de données.

Nous nous sommes penchés sur l'intelligence artificielle (AI) et plus précisément sur l'intelligence faible artificielle [130], une approche pragmatique qui cherche à établir des systèmes de plus en plus autonomes afin de réduire le coût de la supervision. Cette approche est comparable à une reconstitution de l'intelligence humaine qui permet à une machine d'évoluer en passant par un apprentissage. Cependant, il lui est difficile de prendre rapidement en compte les besoins variables des diverses sources de données. En outre, nous voulons donner un certain degré de liberté aux utilisateurs afin qu'ils puissent ajuster les grandes lignes des événements anormaux et déterminer un seuil avant de qualifier ces événements comme anormaux. Une démarche qui se développe de plus en

plus dans des systèmes de sécurité informatique comme Prelude un IDS ou IBM Tivoli Access Manager for Operating Systems un système de contrôle d'accès commandé par des règles.

Dans une autre étude réalisée par C. Flack et M.J. Atallah [131], une méthode d'analyse grammaticale a été définie pour reformater les fichiers log afin de détecter les anomalies. Cependant, une anomalie n'est pas forcement détectable qu'à partir d'un seul fichier.

Toutes ces idées sont intéressantes et prometteuses, malheureusement elles ne peuvent pas pendre en charge un système complètement dynamique (en termes de sources de données et de tâches) d'une part. D'autre part elles sont toutes tournées vers la détection d'intrusion ; mais dans notre travail nous voulons intégrer également le concept de la maintenance qui représente une charge de travail importante et parfois contraignante.

Finalement nous nous sommes orientés vers un concept de corrélation. Ce n'est pas une première approche. Différentes études l'ont déjà invoqué ou utilisé, particulièrement dans l'analyse des données arithmétiques. Ces dernières temps, cette démarche est incluse, notamment, dans les moteurs d'intelligence artificielle des systèmes de détection d'intrusion.

Dans notre solution, nous définissons des modèles de corrélation dans la couche d'analyse pouvant s'appliquer avec succès à n'importe quelle source de données, ils permettent à des utilisateurs de spécifier des activités de corrélation et de les contrôler selon l'environnement du réseau.

Le concept nommé Event Correlation Engine (ECE), permet à l'utilisateur de définir des règles de corrélation pour utiliser des réactions appropriées dans le cas où l'événement est anormal. Dans le contexte de l'ECE, la corrélation relate un événement à un ou plusieurs autres événements; elle recherche parmi les données existantes des informations pour les corréler avec les informations de l'événement déclencheur. Elle examine les événements qui se sont déjà produits pour satisfaire les conditions de corrélation.

L'utilisateur spécifie le comportement de la corrélation dans le moteur de corrélation (ECE) à travers un modèle de corrélation. Ce modèle est composé : d'un déclencheur de corrélation (Correlation Trigger), d'un ou de plusieurs corrélateurs (Correlators) et une conséquence si la corrélation est vérifiée :

- **Trigger** : spécifie les caractéristiques de l'événement qui provoque l'exécution de la corrélation.

- **Correlator** : définit la nature de la corrélation. Il informe comment l'événement déclencheur doit s'apparenter avec les autres événements.
- **Consequent** : c'est l'action résultante si la corrélation est vérifiée.

Par exemple, une corrélation peut se produire dans le cas suivant : si nous obtenons un événement d'accès avec succès à une ressource donnée, nous cherchons si un événement d'ouverture de session a eu lieu et s'il provient du même hôte que l'événement d'accès. Si ce n'est pas le cas, nous pouvons dire que le premier événement est anormal.

V.4. La communauté d'agents mobiles

Pour mieux comprendre la collaboration entre les différents acteurs nous procéderons en quatre étapes :

V.4.1. Phase d'administration

Cette fonctionnalité nous permet, via une interface graphique, d'administrer l'ensemble des services Jini, les composants enregistrés dans l'annuaire et les listes des hôtes concernés par la surveillance. Cette administration consiste à connaître leurs emplacements ainsi que leurs états. Nous pouvons également, les arrêter, les déplacer d'un hôte vers un autre, changer leurs propriétés, etc.

Nous considérons cette fonctionnalité comme une étape essentielle pour le bon fonctionnement du processus de surveillance car elle nous permet de gérer l'ensemble des paramètres de ce dernier, définir les modèles de corrélation de la stratégie d'analyse, manager les ressources de données, les tâches (collectes et actions) ainsi que leurs associations.

Cette étape, dite aussi de paramétrage, sera expliquée au fur et à mesure tout au long des autres étapes où elle se produit.

V.4.2. Phase d'initialisation

Cette phase, comme son nom l'indique, décrit les opérations d'initialisation des composants. Au démarrage de chaque agent, il crée son service de localisation pour qu'il puisse être contacté par les autres composants. Puis il s'enregistre auprès d'un annuaire.

L'agent annuaire et l'agent analyseur s'appuient sur un système d'écoute pour s'auto découvrir. La différence entre ces deux composants est que l'agent hôte s'ajoute en tant qu'écouteur afin qu'ils puissent être notifiés par des événements distants pour recevoir la référence de l'agent annuaire quand ce dernier change d'emplacement.

Un agent hôte dispose du nom d'un agent annuaire préférentiel. Dès lors qu'il trouve un service annuaire appartenant au groupe *Directory* ayant un attribut correspondant au nom prédéfini, il ajoute son écouteur. Dans le cas où il ne le trouve pas, il s'enregistre auprès du premier annuaire localisé, à défaut il continue son processus d'écoute.

La référence récupérée par l'agent hôte sera disponible pour les autres agents mobiles, qui arrivent sur le hôte et qui désirent contacter l'agent annuaire.

Le diagramme de séquence suivant (cf. Figure 25) schématise l'étape d'initialisation d'un agent hôte.

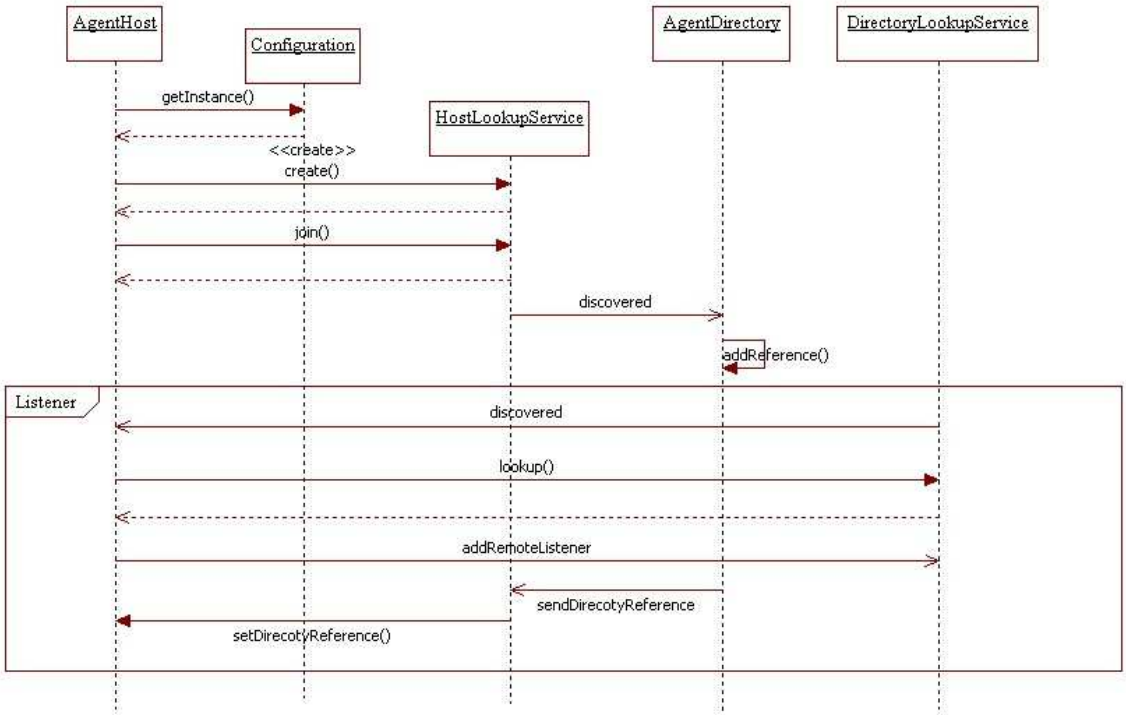

Figure 25 Diagramme de séquence de l'étape d'initialisation d'un agent hôte

L'agent analyseur et l'agent collecteur peuvent être créés par l'utilisateur, via l'interface d'administration, l'ordre de leur exécution n'est pas important et peut être en différé. Dans le cas où l'agent analyseur ne trouve aucun agent collecteur dans le réseau il peut créer autant d'agents collecteurs qu'il le souhaite en fonction de ses besoins. Ce nombre dépend

de la stratégie d'analyse. La différence entre la création d'un agent collecteur par l'agent analyseur ou via l'interface est que cette dernière méthode permet à l'utilisateur de spécifier le hôte initial de l'agent.

Les deux types d'agent (analyseur et collecteur) s'appuient sur le service de localisation du hôte où ils s'exécutent pour récupérer la référence de l'annuaire afin de s'enregistrer auprès de lui. Le diagramme suivant (cf. Figure 26) présente l'étape d'initialisation d'un agent collecteur.

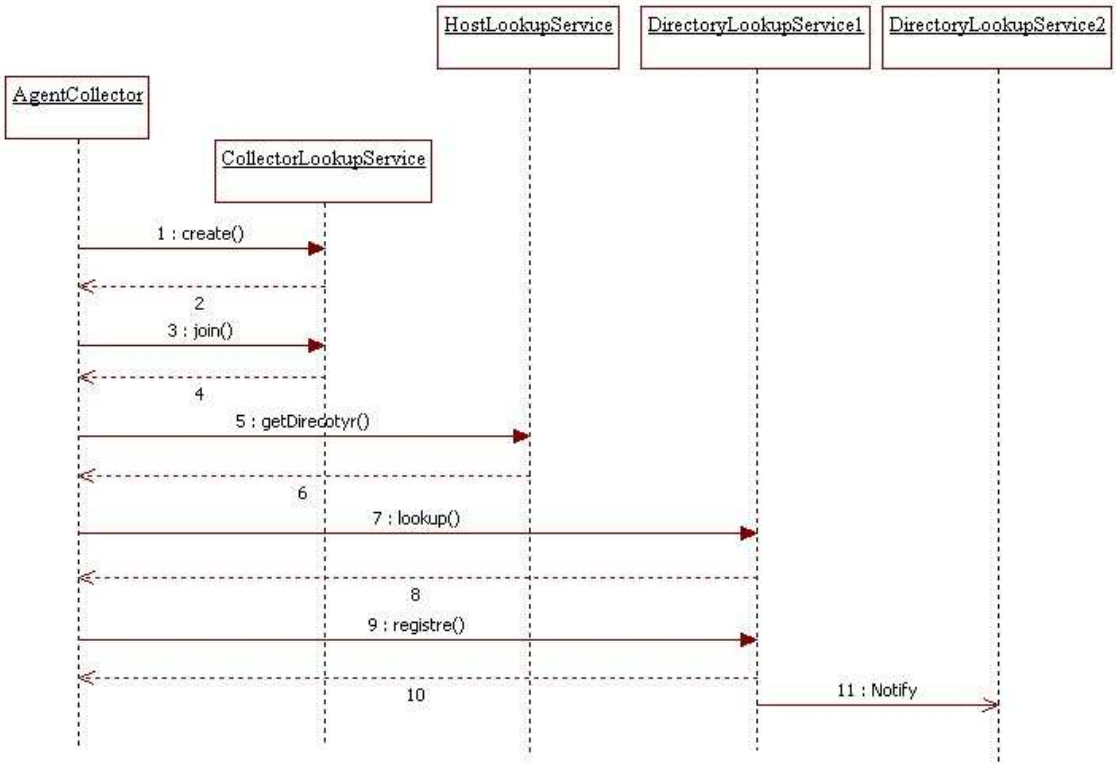

Figure 26 Diagramme de séquence de l'étape d'initialisation d'un agent Collecteur

La distinction entre les deux est que, d'une part, l'annuaire génère et associe une clé chiffrée pour chaque collecteur qu'il lui transmet lors de son enregistrement. Cette clé servira pour tous les échanges entre l'agent analyseur et l'agent collecteur dans le but de s'assurer de l'authenticité de chaque agent comme le montre le schéma suivant (cf. Figure 27). D'autre part, l'agent analyseur charge la liste des tâches définies par l'utilisateur pour créer les services Jini correspondants à ces dernières et les publie dans son propre service de localisation.

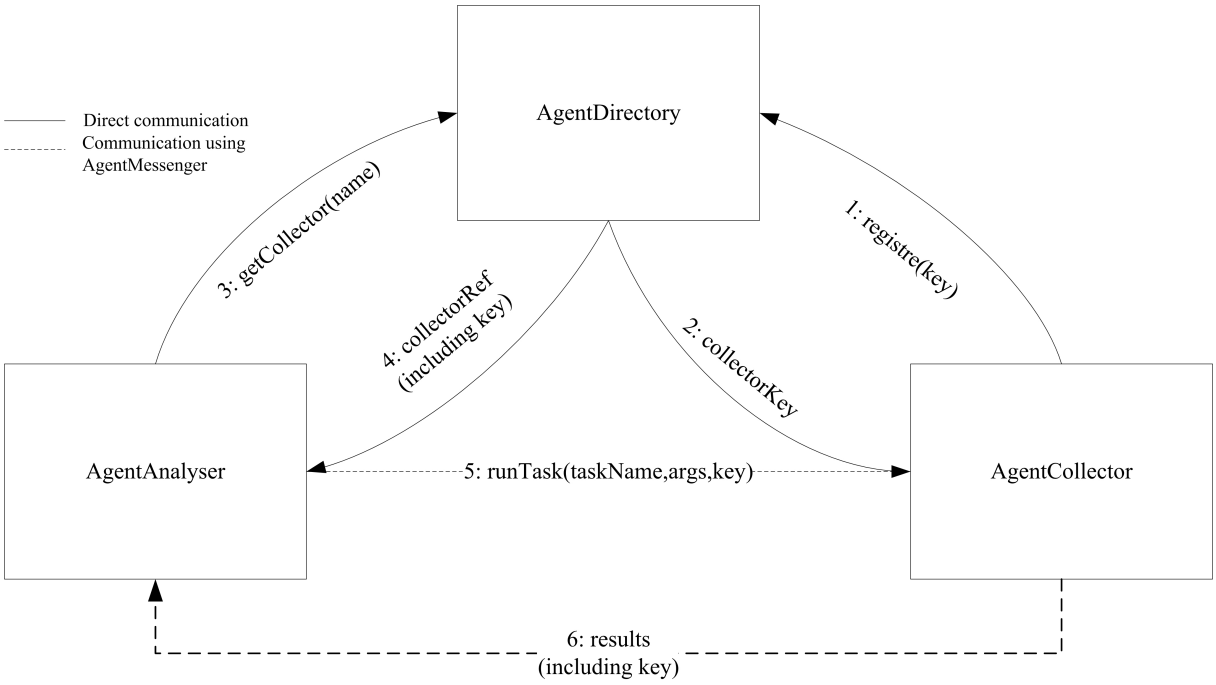

Figure 27 Mécanisme d'échange de clé

V.4.3. Phase de collecte

Régulièrement, en fonction des paramètres définis par l'utilisateur via l'interface d'administration, l'agent analyseur recherche la liste des hôtes auprès de l'annuaire. Cette liste correspond à la liste des agents hôtes enregistrés c'est-à-dire les machines qui sont capables de recevoir des agents mobiles car nous pouvons imaginer qu'il peut y avoir des machines sur les réseaux non surveillés. L'agent analyseur recherche également la liste des agents collecteurs enregistrés y compris leurs clés d'échange et charge aussi les informations relatives aux tâches à effectuer par l'hôte. Sur un hôte donnée un agent collecteur peut exécuter plusieurs tâches. Ces informations, définies par l'utilisateur via l'interface d'administration, sont des associations entre les hôtes et les tâches.

Pour chaque hôte, l'utilisateur définit un ordre et les priorités des tâches par hôte. Le poids est une valeur comprise entre 0 et 1. Cette dernière valeur signifie que le résultat de la tâche en question doit être transmis après sa collecte et non à la fin du parcours de l'agent collecteur.

Il faut noter qu'une tâche est un service définissant le traitement et les opérations nécessaires à la réalisation de cette dernière, la source de données où nous allons collecter les données, les règles pour formater les informations collectées et les filtres pour collecter

les informations jugées utiles par l'utilisateur. Une définition plus précise de ces informations sera abordée dans le chapitre suivant.

Le schéma suivant (cf. Figure 28), présente sommairement un exemple de combinaisons entre les hôtes et les tâches. Pour chaque hôte, nous pouvons définir une ou plusieurs tâches, et pour chaque tâche nous définissons une priorité. Par exemple, auprès du serveur Web nommé *SrvWeb* nous allons initier trois tâches : la collecte du fichier Log du service Apache, cette tâche possède un priorité élevée (1), la collecte du journal des événements des erreurs de Windows et la collecte du journal des événements des applications uniquement pour les événements relatifs à la source Apache Service.

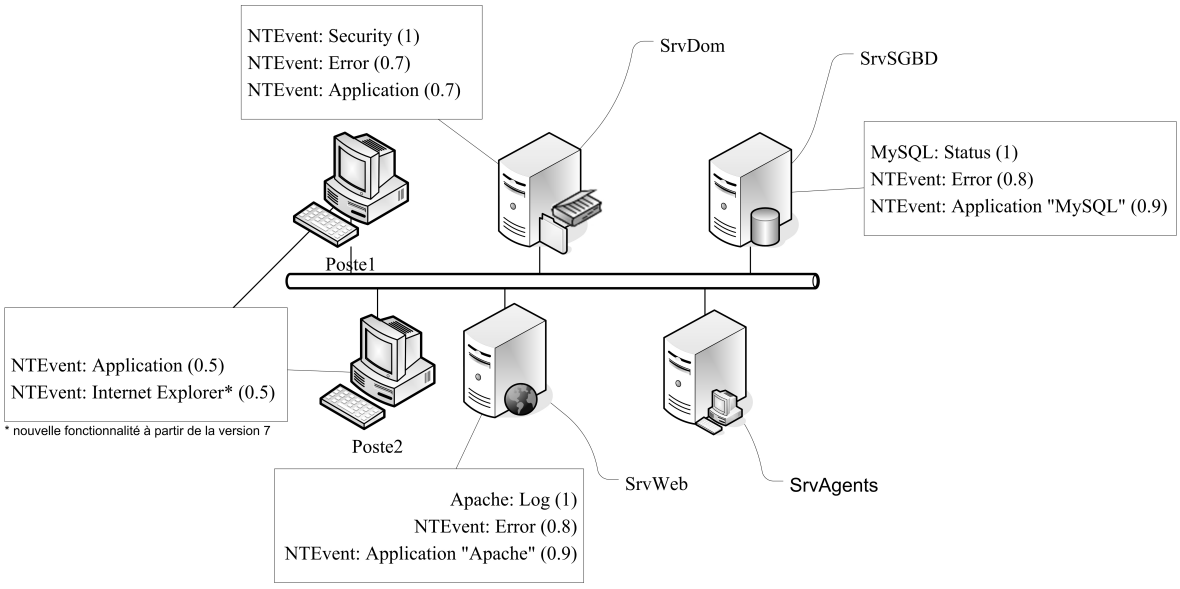

Figure 28 Exemple de tâches à effectuer

L'ensemble des données chargées permet à l'agent analyseur de construire des piles de tâches à transmettre aux agents collecteurs, généralement une pile par *AgentCollector*. Cette construction s'opère à l'aide d'un système de résolution de contraintes par exemple Java Constraint Library (JCL) [132]. La programmation de contrainte consiste à résoudre des problèmes combinatoires modélisés par un problème de satisfaction de contrainte (CSP). Clairement, un CSP est défini par un triplet (X, D, C) comme :

- **Variables**: X = {X1, X2,..., Xn} est l'ensemble de variables du problème
- **Domaine**: D est une fonction qui associe chaque variable Xi à son domaine D (Xi), autrement dit, les valeurs possibles qui peuvent être affectées à Xi.

- **Contraintes** : C = {C1, C2,..., Ck} est l'ensemble des contraintes. Chaque contrainte Cj est une relation entre un sous-ensemble de variables qui définit la limite de chaque domaine.

Dans notre cas, les variables sont considérées comme les tâches, le domaine est la liste des hôtes et les contraintes sont les poids des tâches (un poids distinct par pile) ainsi que l'ordre des hôtes et des tâches. Inlassablement, le CSP retourne à l'agent analyseur une solution unique qui satisfait l'ensemble des contraintes précitées. Le cas où il ne retourne aucune solution est fort improbable parce qu'il est géré en amont, l'utilisateur ne peut pas sauvegarder une stratégie de collecte jugée incohérente. Un contrôle systématique est effectué lors des mises à jour des informations d'associations des tâches et des hôtes.

La solution retournée par le CSP est un ensemble de piles, une pile par agent collecteur. Ce nombre correspond à la plus grande somme des tâches du même poids. L'exemple suivant (cf. Tableau 8) donne une solution de l'exemple exposé précédemment (cf. Figure 28).

Tableau 8 Exemple de solution de stratégie de déplacement (parcours et tâche)

	SrvDom	***SrvSGBD***	***SrvWeb***	***Poste1***	***Post2***
Tâche1	1	1	1	0.5	0.5
Tâche2	0.7	0.9	0.9	0.5	0.5
Tâche3	0.7	0.8	0.8		

⇩

Pile1	SrvDom /Tâche1 (1)	SrvSGBD /Tâche2 (0.9)	SrvSGBD /Tâche3 (0.8)	SrvDom /Tâche2 (0.7)	Poste1/Tâche1 (0.5)
Pile2	SrvSGBD /Tâche1 (1)	*SrvWeb* /Tâche2 (0.9)	*SrvWeb* /Tâche3 (0.8)	SrvDom /Tâche3 (0.7)	Poste1/Tâche2 (0.5)
Pile3	*SrvWeb* /Tâche1 (1)	Poste2/Tâche1 (0.5)			
Pile4	Poste2/Tâche2 (0.5)				

Il faut noter que, d'une part le poids des tâches n'est pas général à l'ensemble des hôtes mais à l'ordre de l'exécution des tâches pour un hôte et par un collecteur donné. D'autre part, le poids ne peut pas être considéré comme un ordre d'arrivée des résultats des tâches parce que cet ordre dépend fortement de la taille des données à collecter par tâche.

La figure suivante (cf. Figure 29) résume la première étape de la phase de collecte.

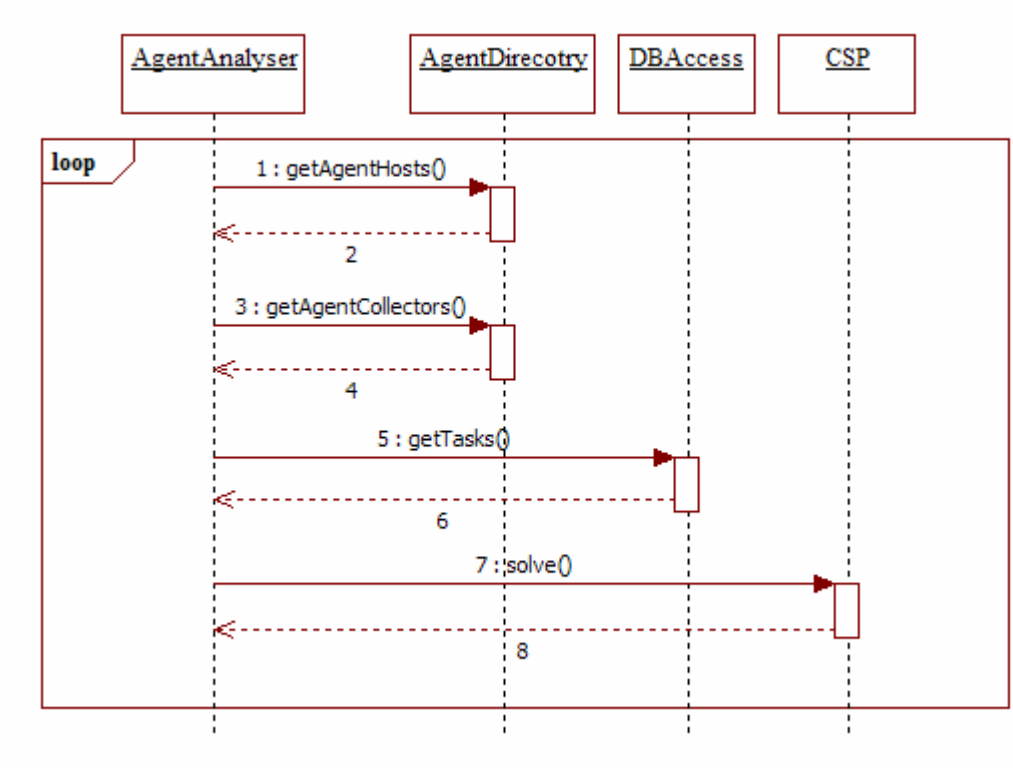

Figure 29 Initialisation de la phase de collecte

L'agent analyseur associe pour chaque pile de tâches un agent collecteur. Dans le cas où, il n'y a pas autant d'agents collecteurs que de piles, l'agent analyseur crée autant d'agents collecteurs nécessaires. Puis pour chaque agent collecteur, il crée un agent messager et lui transmet un message de collecte incluant la liste des hôtes à parcourir, des tâches à effectuer par hôte et la clé d'échange. L'agent messager s'exporte vers le hôte où l'agent s'exécute afin de lui délivrer le message.

Dès que l'agent collecteur reçoit le message, il se déplace vers le premier hôte de la liste puis exécute les tâches et formate les informations collectées. Il répète les mêmes opérations jusqu'à ce qu'il n'y ait plus aucun hôte à parcourir. À chaque déplacement, l'agent collecteur crée un agent messager en lui intégrant un message de notification de localisation pour le transmettre à l'agent annuaire afin que ce dernier puisse mettre à jour ses informations. Lorsque l'ensemble des tâches est effectué, le collecteur crée un agent messager et lui transmet un message de résultat incluant l'ensemble des données collectées et la clé d'échange. À son tour l'agent messager compresse les données et de la même manière qu'il transmet les tâches à l'agent collecter, il transmet les résultats à l'agent

analyseur. Cette opération s'effectue après la décompression des données lorsque l'agent messager arrive sur l'hôte de l'agent analyseur.

Lors de la réception des résultats des collectes, l'agent analyseur stocke les données reçues dans une base de données consacrée à cet effet, puis dépile sa pile de tâches. Dans le cas où certaines tâches n'ont pas pu être exécutées, l'agent analyseur va soit les empiler à nouveau pour pouvoir les exécuter au prochain cycle, soit il notifiera l'utilisateur. Ce choix est définit par l'utilisateur via l'interface graphique.

La figure suivante résume la deuxième étape de la phase de collecte

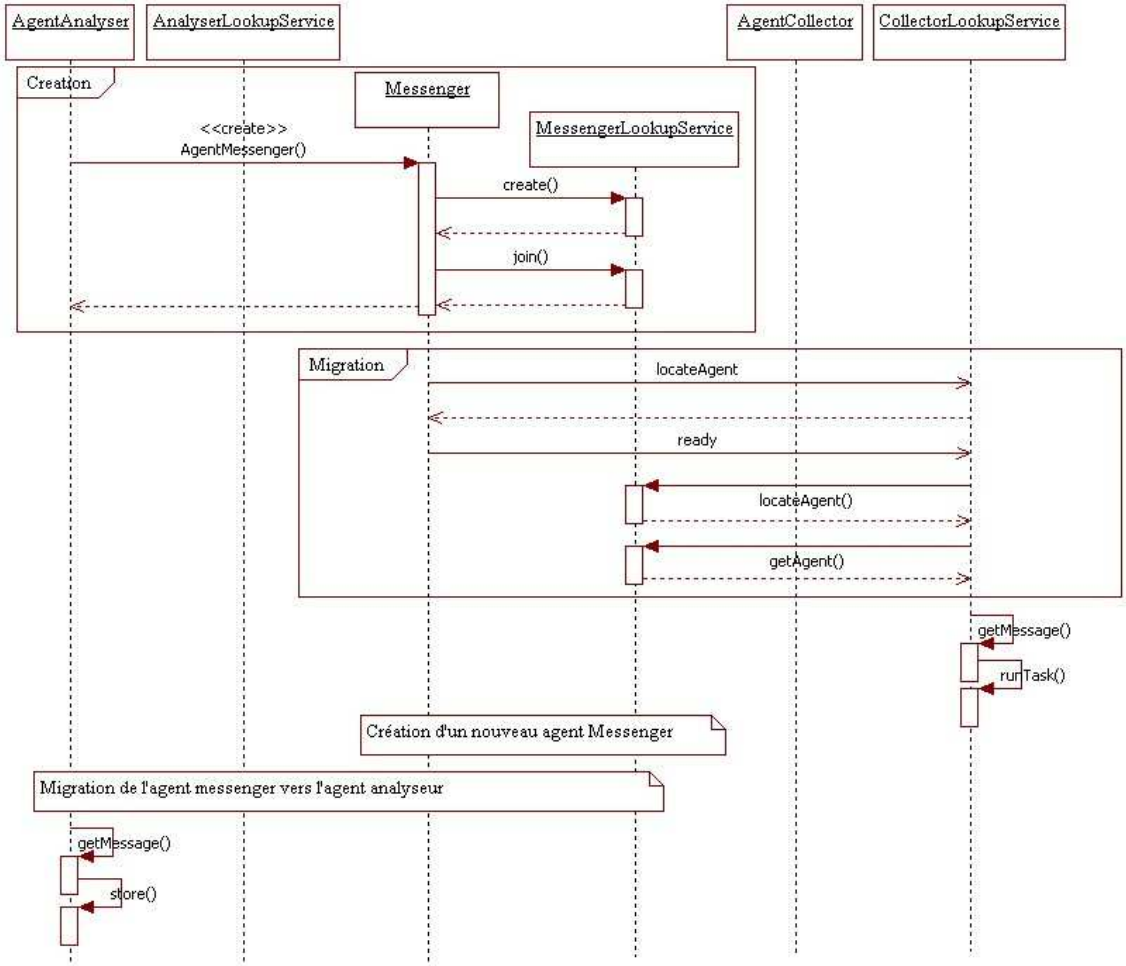

Figure 30 Transmission et exécution des tâches de collecte

La difficulté de cette phase de collecte se situe dans la taille des données collectées. Habituellement ces données sont stockées dans la mémoire de l'agent collecteur et ce dernier les transporte avec lui d'un hôte à l'autre. Mais est-il capable de gérer et de véhiculer des données de taille très importante ? Qu'elle sera sa limite ?

Il est difficile de répondre à ces questions, parce que le problème ne concerne pas seulement l'agent collecteur. La limite peut poser problème aussi à l'agent messager qui transporte des données et l'agent analyseur qui les reçoit. L'idée, pour résoudre cette difficulté, est de définir une limite de taille : quand elle est atteinte l'agent collecteur arrête temporairement la collecte et transmet des résultats partiels de la tâche interrompue, ainsi que les résultats des collectes précédentes, déjà effectuées, à l'agent analyseur de la même manière qu'il envoie les résultats. Lorsqu'il s'agit d'une collecte à partir d'un seul hôte, nous pouvons dire que la limite dépend de la plus petite taille maximale qui pourra être allouée à la machine virtuelle de l'un des trois composants (collecteur, messager et analyseur). Si l'agent collecteur doit parcourir plusieurs hôtes, cette taille dépendra des données déjà collectées et des données restantes à collecter. Cette limite reste difficile à estimer dans l'état actuel. Nous pouvons imaginer que pour certaines tâches, telle que la collecte à partir des fichiers, l'agent collecteur est capable d'estimer la taille en se basant sur la taille du fichier. Seulement, ceci n'est pas valable à l'ensemble des tâches, notamment, la collecte des données à partir d'une base de données. D'autant plus que le problème des données futures à collecter, dans le même parcours, persiste.

Pour pallier à cela, nous définissons un seuil paramétrable par l'utilisateur, il faut admettre que c'est une façon simple pour détourner le problème. Néanmoins, nous avons réalisé des tests pour se rapprocher d'une limite plus ou moins exacte. Ces tests ont été réalisés à l'aide du prototype que nous avons développé, pour tester notre approche mobile et dans des conditions identiques à un environnement de production. Pour l'ensemble des tests, nous avons tenté de collecter les données d'un fichier log d'un serveur Apache, contenant 1 000 000 de lignes et d'une taille approximative de 120 Mo.

Dans un premier temps, nous avons réalisé l'opération avec une JVM d'une taille de 64 Mo (taille par défaut), puis nous avons augmenté cette dernière jusqu'à 256 Mo. Puis, nous avons relevé le temps pour chaque 5 Mo de données collectées. Nous avons intégré la notion de temps pour avoir des points de référence. Le but de ces tests n'est pas de calculer le temps du traitement de la collecte mais plutôt la limite des données.

Le graphe suivant (cf. Figure 31) donne les résultats de ces tests. Nous constatons que le ratio de la taille mémoire de la JVM et la taille des données collectées reste identique, il est d'environ **0,3**.

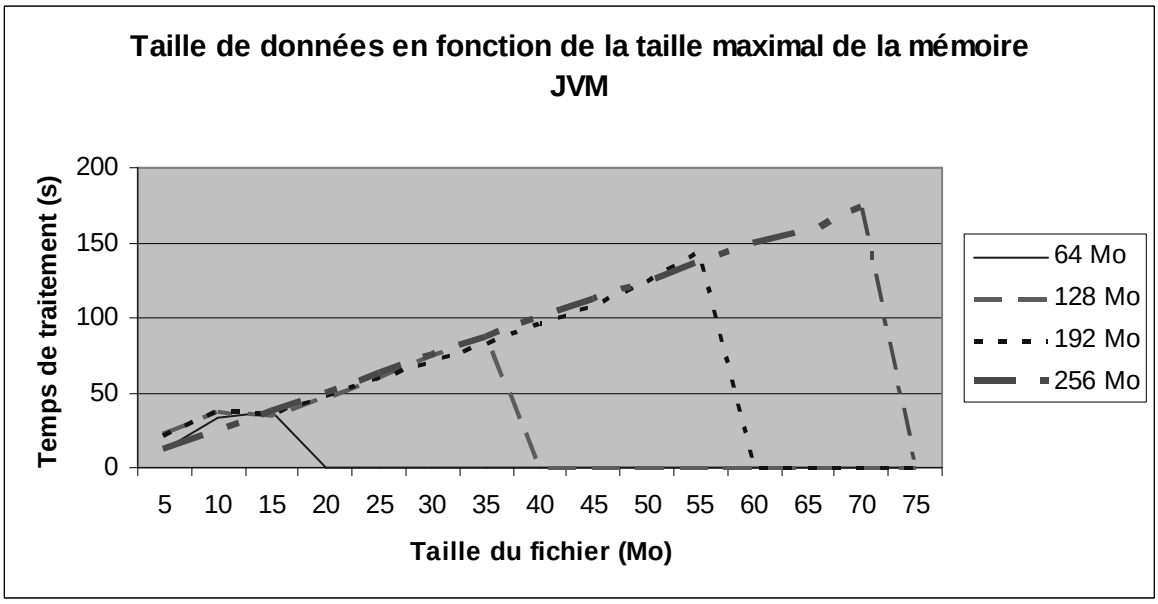

Figure 31 Résultats d'expérimentation pour déterminer la taille

Ce ratio est d'environ 30 % de la plus petite taille de mémoire maximale allouée par les JVM. Ce pourcentage peut être retenu pour définir la taille de la limite. Cette valeur est transmise à l'agent collecteur au même titre que les tâches, elle va lui permettre de contrôler sa taille, d'arrêter sa collecte et de transmettre le résultat à l'agent analyseur puis de libérer sa mémoire, à l'aide du Garbage Collector, avant de reprendre ses traitements. Ce résultat n'est pas considéré comme cohérent, vis-à-vis de l'agent analyseur. Puisqu'il n'est pas sûr que l'agent collecteur soit capable de compléter ces tâches. Certes pour certaines sources de données, où la tâche a été finalisée, le résultat sera traité mais les tâches incomplètes seront mises en attente. En effet, quand l'agent collecteur veut transmettre des résultats partiels à l'agent analyseur, il crée un agent messager en lui incluant un message de début de transaction et à chaque fois qu'il veut lui envoyer le reste des résultats partiels, il transmet les données avec un message de transaction. La dernière partie est transmise à l'agent analyseur avec le message de fin de transaction ainsi que le nombre des parties transmises pour que l'agent analyseur puisse valider la transaction et sauvegarder les données reçues pour les analyser. Il faut noter que les données partielles ne sont pas stockées dans la mémoire de l'agent analyseur mais sérialisées dans des fichiers. La figure suivante montre les séquences d'un exemple du parcours utilisant des transactions.

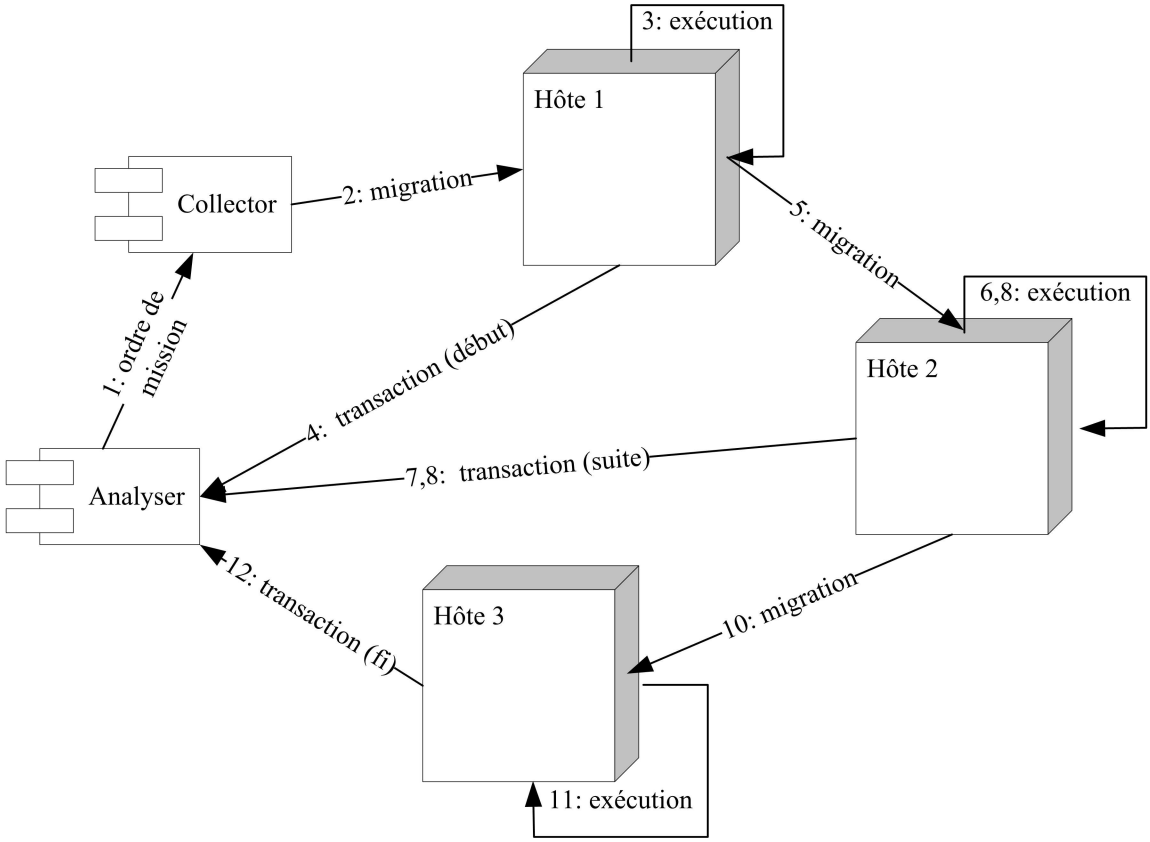

Figure 32 Exemple d'un parcours utilisant des transactions

Le service gérant les transactions est un service Jini créé et publié au lancement de l'agent analyseur, le seul composant qui interagit et communique avec lui. Ce service permet de créer, gérer et valider des transactions. Il permet, également, d'organiser les baux des transactions. Ces baux, sont nécessaires pour annuler une transaction, si l'agent collecteur ne finalise pas une transaction, elle sera automatiquement annulée une fois son bail arrivé à expiration.

Il est essentiel de préciser qu'une transaction peut être initiée par un seul agent collecteur et pour une tâche donnée. Dans le cas où le traitement d'une tâche n'est pas séquentiel puisqu'un agent collecteur collabore avec d'autres agents collecteurs pour paralléliser le traitement de la tâche en question, il devient difficile de gérer les transactions. Seul le premier agent collecteur peut transmettre les données et ainsi créer une transaction, ceci complique le processus d'échange des données entre les différents acteurs du traitement.

Cette contrainte s'applique également à la limite de taille lorsqu'il s'agit d'un traitement parallèle effectué par plusieurs agents.

V.4.4. Phase d'analyse

Cette dernière phase consiste à analyser les données collectées. L'agent analyseur charge, continuellement, les modèles de corrélation définis par l'utilisateur et pour chaque règle il vérifie les conditions du déclencheur. Si l'une de ces dernières est satisfaite, les règles de corrélation sont alors activées. En fonction de l'intervalle définit dans le modèle, l'analyseur continue à vérifier les conditions du corrélateur. Une fois la limite dépassée, il construit l'action à entreprendre, cette dernière est également définie sous forme de conséquence dans le modèle. Le mécanisme nécessaire au traitement de la corrélation est défini dans la partie suivante (cf. § Modèle de corrélation).

Nous distinguons deux types de conséquences, locales et distantes :

- **Locales** : ce sont des conséquences considérées comme un simple report d'information, elles sont créées en fonction de leurs paramètres puis transmises à la couche report pour les réaliser.
- **distantes** : ce sont des tâches d'actions. L'agent analyseur crée la tâche en fonction des paramètres définis dans le modèle de corrélation, puis crée un agent messager en lui intégrant un message de tâche d'action et lui fournit la tâche à transmettre à l'agent collecteur. Ce dernier sera chargé de se déplacer, si cela est nécessaire, à l'hôte en question afin d'exécuter la tâche. De la même façon que pour les tâches de collecte, l'agent collecteur transmet le résultat de cette tâche d'action (Échec, Succès..) à l'agent analyseur via un agent messenger.

V.5. Structure des données

V.5.1. Événement (Event)

Comme nous l'avons exposé précédemment, l'ensemble des données collectées sont formatées en un format unique pour faciliter leur analyse et pour réduire sensiblement leur taille. Ce format est définit par une entité abstraite *Event*, cette classe a été conçue pour prendre en charge les différentes sources de données. Nous pouvons dire, vulgairement, que c'est une synthèse de l'ensemble des formats de sources de données existants et à venir.

Un événement est structuré en six attributs :

- **Host (Hôte)** : le nom de l'hôte où l'événement s'est produit.
- **Date** : la date (jour et heure) où l'événement s'est produit.
- **Origine (Source)** : le nom de l'application, le programme ou le matériel générant l'événement.
- **Destination (Target)** : la finalité de l'événement.
- **Action** : l'opération de l'événement.
- **Custom** : zone diverse pour personnaliser l'événement en fonction de la source de données.

Pour mieux comprendre ces attributs, le tableau suivant (cf. Tableau 9) montre deux exemples de données en provenance d'un ficher d'activités Apache (cf. exemple ci-après) et un journal d'événement Windows NT (cf. Figure 33)

Tableau 9 Exemple des événements

Attribue	***Apache Access Log***	***NT Event Log***
Host	192.168.100.130	GIMBE004
Date	1246113040	1214419988
Source	ApacheAccessLog	NTEventApplication
Target	/members	Symantec Antivirus
Action	200	Information
Custom	user	Analyse Terminée : Menaces : 0 Eléments analysés : 24043 Fichiers/Dossiers/Lecteurs omis : 0

Exemple Apache Access Log:

192.168.100.130 - user - [27/Jun/2009:16:30:40 +0200] "GET /members HTTP/1.1" 200 322 "-" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022)"

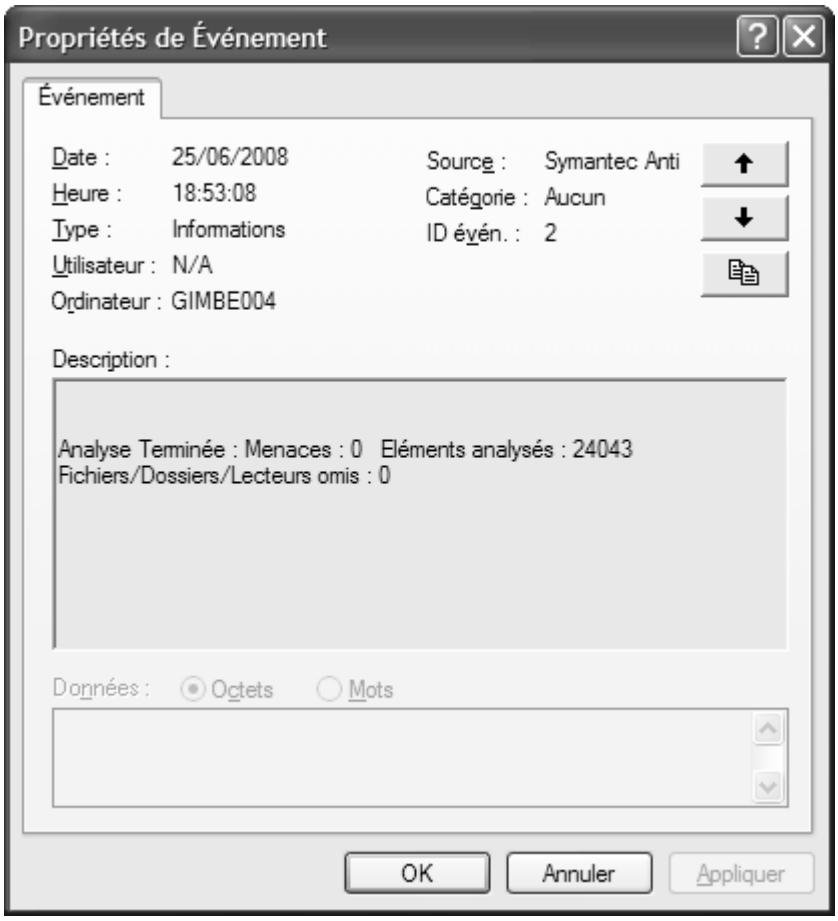

Figure 33 Exemple d'événement Windows

V.5.2. Tâches (Tasks)

Les tâches d'action permettent l'exécution d'un script, une ligne de commande ou un programme afin de contrôler un service ou de mettre à jour une application. Ces tâches sont moins contraignantes que les tâches de collecte parce qu'elles n'ont pas une forte dépendance à l'environnement d'exécution.

Les tâches de collecte, quant à elles, reposent sur les sources de données où elles collectent les informations. Ces sources peuvent être un fichier plat, une API, une base de données ou une interface réseau. Elles peuvent être classées par type, certains types ont un format homogène. Pour les fichiers plats, leurs formats varient d'une source à l'autre. Pour gérer cette distinction, la tâche de collecte doit décrire, en plus des règles permettant de formater les événements, les informations des sources de données. Ces informations sont nécessaires pour établir un lien afin de collecter les données. Ces règles sont représentées sous forme d'une expression régulière. Nous associons à chaque attribut d'événement un modèle qui sera utilisé pour trouver son correspondant dans les données collectées.

Le tableau suivant représente un exemple de règles de collecte pouvant être appliquées au format standard d'un log Apache.

Exemple Apache Access Log:		
192.168.100.130 - user - [27/Jun/2009:16:30:40 +0200] "GET /members HTTP/1.1" 200 322 "-" "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.648; .NET CLR 3.5.21022)"		
Attribue	***Règles***	***Résultat***
Host	\b(?:\d{1,3}\.){3}\d{1,3}\b	192.168.100.130
Date	\d{2}\/[A-Za-z]{3}\/\d{4}(\:\d{2}){3}\s(\+|\-)\d{4}	1246113040
Source	La source ne dépend pas des éléments collectés, elle correspond au nom de la source liée à la tâche.	
Target	\"[A-Z]{3,7}\s(\/\S+|\/)	/members
Action	\"\s(\d{3})	200
Custom	\-\s(\w*)\s\-	user

Cette forte association et la diversification des sources de données impliquent l'utilisation du modèle de conception Abstract Factory (cf. Figure 34) comme nous l'avons cité précédemment.

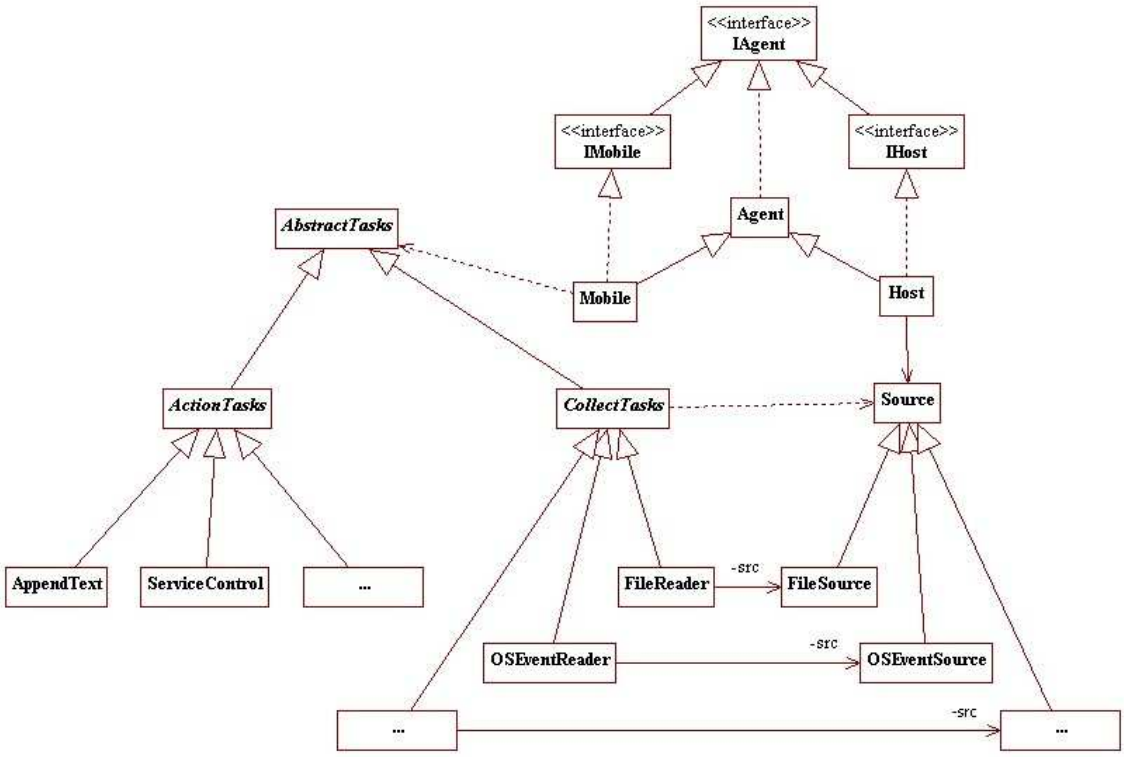

Figure 34 Diagramme de classe des tâches

Le code Java suivant présente le modèle sur lequel nous devons nous appuyer pour développer une nouvelle tâche de collecte.

```
public class TemplateCollectTask extends CollectTasks {
        private TemplateSource src;
        private String[] rules=new String[6];
        private Filter[] filtres;
        //Others properties
        protected TemplateCollectTask(String[] args){
                src = new TemplateSource(args[0]);
                //Others initialization
        }
        @Override
        public Result runTask(String[] rules, Filter[] filtres) {
                this.rules=rules;
                this.filtres=filtres;
                Vector<Event> result = new Vector<Event> ();
                //Collect and Formatting treatment
                return new CollectResult(0, result);
        }
```

```
}
class TemplateSource extends Source {
        //Others properties
        public TemplateSource(String name) {
                super(name);
        }
}
```

En ce qui concerne les tâches d'action nous avons défini deux types de tâches, elles correspondent aux conséquences de la corrélation. Les tâches locales qui interagissent avec la couche report. Les tâches distantes, qui doivent s'exécuter sur des machines distantes, sont créées de la même façon qu'une tâche de collecte même si elles ne disposent pas des informations relatives aux sources de données. Néanmoins, les actions sont fortement spécifiques et ne devraient dans aucun cas être généralisées. Par exemple : une tâche d'automatisation de mise à jour de logiciel est parfois spécifique à la version précédente.

V.5.3. Résultats (Result)

De la même façon que les tâches, nous avons défini deux types de résultats, chaque type se réfère à un type de tâche. Le résultat d'action *ActionResult* représente un message de sortie issu de l'exécution de l'action. Le résultat de collecte *CollectResult* représente un groupe d'événements collectés pour une source donnée. Ces événements sont stockés dans une base de données, dès leur réception par l'agent analyseur, pour les analyser. Une analyse structurée est organisée sous le concept de la convergence. Cela signifie qu'il est essentiel d'extraire le plus petit ensemble de résultats possibles. De la même manière, nous employons un dispositif discriminatoire afin de garder des interprétations significatives.

Pour les deux types de résultats, nous disposons d'un code d'erreur qui permet à l'agent analyseur de savoir si l'action s'est bien déroulée ou pas. Dans le dernier cas, et en fonction du code, l'agent analyseur doit reporter l'échec de l'exécution de la tâche ou l'empiler à nouveau pour une autre tentative.

Généralement, pour l'ensemble des tâches le code 0 correspond à une réussite. Sinon les raisons d'échec sont des codes qui varient d'une tâche à l'autre, ils correspondent soit aux codes définis par le développeur quand il s'agit des tâches natives Java, souvent ces codes

s'accordent aux exceptions Java et peuvent survenir au moment du traitement. Soit aux codes générés par l'application pour les tâches d'action ou par des API pour les tâches de collecte.

Voici un exemple de codes pour deux tâches de collecte :

- Tâche de collecte des données d'un fichier log apache (*LogParser*), le traitement de cette dernière est natif.
- Tâche de collecte du journal d'événements Windows (*NTEventLogReader*) réalisée par l'intermédiaire de l'API Servertec Foundation Classes. SFC est une collection de classes de Java et de bibliothèques natives permettant aux développeurs d'avoir accès à des fonctionnalités de l'environnement Windows, pour celles-ci les codes correspondent aux codes erreurs de Windows (MSDN System Error Codes)

Code	***LogParser***	***NTEventLogReader***
0	Tache réalisée avec succès	Tache réalisée avec succès
1	Le fichier spécifié est introuvable	
2	Problème d'entrée/sortie lors du traitement	Problème d'entrée/sortie lors du traitement
1500		Le fichier journal d'événements est endommagé.
1501		Le fichier journal d'événements n'ayant pu être ouvert, le service d'enregistrement des événements n'a pas démarré.
1502		Le fichier journal d'événements est plein.
1503		Le fichier journal d'événements a changé pendant les opérations de lecture.

V.5.4. Messages

Nous avons défini une hiérarchie de classes de plus en plus spécialisées, pour déterminer plusieurs types de messages, une classe par type. Comme le montre le diagramme suivant (cf. Figure 35). La définition de ces classes est similaire à celle des messages KQML sans performative, c'est-à-dire seul le niveau de communication et le niveau de contenu sont décrits.

Le niveau de communication définit dans la classe mère *Message* représente la référence de l'agent expéditeur (*Sender*) et de l'agent destinataire du message (*Receiver*).

Le niveau de contenu est un nombre variable d'attributs dépendants du type d'informations à transmettre (tâche de collecte, tâche d'action, récupération des résultats, notification de

localisation).

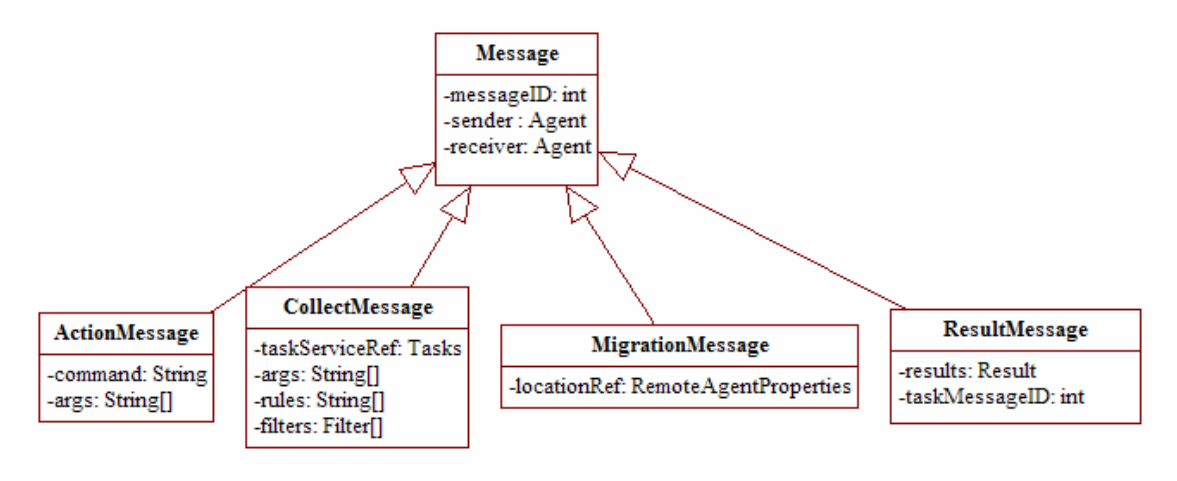

Figure 35 La structure des Messages

V.5.5. Modèle de corrélation (ECE Template)

Comme nous l'avons dit précédemment les événements collectés sont analysés en utilisant des modèles de corrélation. Ces modèles sont gérés par l'utilisateur et sauvegardés dans une base de données. Quand un agent analyseur désire analyser les événements en attente, il charge l'ensemble des modèles de corrélation disponibles sous forme d'une structure XML valide, puis les transforme en requêtes SQL afin de les exécuter.

L'exemple suivant, montre un modèle de corrélation pour la source de données Apache (*ApacheLogSource)* : si l'accès est autorisé à une section sécurisée ("/members"), nous vérifions s'il existe le même événement pour le même utilisateur connecté à cette section mais à partir d'un autre hôte.

```
<?xml version="1.0" encoding="UTF-8"?>
<templates xsi:noNamespaceSchemaLocation="ECE.xsd">
<template id="1" text="Apahce multiple user access">
        <trigger source="Apache Log">
            <conditions>
                <eq order="1" match="target" value="/members"/>
                <eq order="2" match="action" value="200"/>
            </conditions>
        </trigger>
        <correlator source="Apache Access Log" seconds="3600" limit="1">
            <conditions>
```

```
                    <eq order="1" match="custom" value="custom"/>
                    <ne order="2" match="host" value="host"/>
                </conditions>
            </correlator>
            <consequent type="remote" name="AppendText">
                <arguments>
                    <argument order="1" value=".htaccess"/>
                    <argument order="2" name="deny from" match="trigger"
value="host"/>
                    <argument order="3" name="deny from"
match="correlator" value="host"/>
                </arguments>
            </consequent>
        …
</templates>
```

La transformation des modèles en requêtes SQL est réalisée à l'aide d'une transformation XSL (XSLT) comme le montre le schéma suivant. Le langage XSLT (spécification W3C) est équipé pour traiter des XML complexes aussi rapidement et facilement que simplement. Mais quand la tâche devient trop coûteuse en terme de temps, il est également possible d'employer XSLTC pour produire une ou plusieurs classes de Java (que nous appellerons "translets"). Toutes ces technologies et leurs avantages nous aident à améliorer notre stratégie pour automatiser notre analyse et pouvoir réagir rapidement.

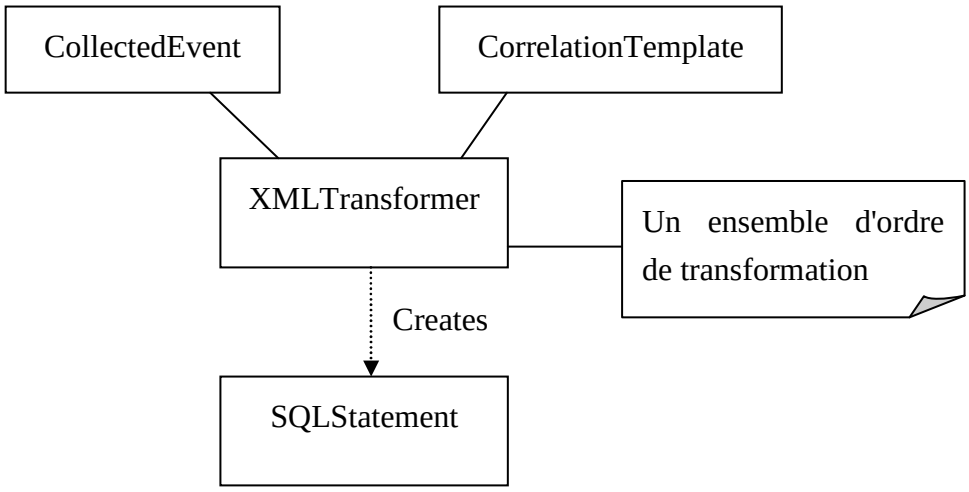

Figure 36 Transformation des règles de corrélations

Pour le modèle de corrélation précité, la transformation XSL générera une requête SQL (voir exemple ci-dessous), en fonction des résultats de la requête, si le nombre des événements obtenus est supérieur ou égal à la limite définie, alors l'agent analyseur construit une nouvelle tâche d'action en utilisant les paramètres de la conséquence définie dans le modèle. Dans notre étude de cas, cette action ajoutera deux lignes au fichier de configuration Apache (".htacces") pour définir une nouvelle règle du répertoire ("/members") et ainsi restreindre l'accès aux deux hôtes qui accèdent en même temps à ce service.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output version="1.0" encoding="UTF-8" indent="no" omit-xml-declaration="no"/>
    <xsl:template match="/">
        <sql>
            <xsl:for-each select="//template">
                <request>
                    <xsl:text>SELECT Count(TG.idEvent) AS countEvent FROM events AS TG, events AS CO WHERE </xsl:text>
                    <xsl:text>TG.source="</xsl:text>
                    <xsl:value-of select="trigger/@source"/>
                    <xsl:text>" AND CO.source="</xsl:text>
                    <xsl:value-of select="correlator/@source"/>
                    <xsl:text>"</xsl:text>
                    <xsl:for-each select="trigger/conditions/eq">
                        <xsl:text> AND TG.</xsl:text>
                        <xsl:value-of select="@match"/>
                        <xsl:text>="</xsl:text>
                        <xsl:value-of select="@value"/>
                        <xsl:text>" </xsl:text>
                    </xsl:for-each>
                    <xsl:for-each select="trigger/conditions/ne">
                        <xsl:text>AND TG.</xsl:text>
                        <xsl:value-of select="@match"/>
                        <xsl:text>!="</xsl:text>
                        <xsl:value-of select="@value"/>
                        <xsl:text>" </xsl:text>
                    </xsl:for-each>
                    <xsl:for-each select="correlator/conditions/eq">
```

```
                    <xsl:text> AND CO.</xsl:text>
                    <xsl:value-of select="@match"/>
                    <xsl:text>=TG.</xsl:text>
                    <xsl:value-of select="@value"/>

                </xsl:for-each>
                <xsl:for-each select="correlator/conditions/ne">
                    <xsl:text> AND CO.</xsl:text>
                    <xsl:value-of select="@match"/>
                    <xsl:text>!=TG.</xsl:text>
                    <xsl:value-of select="@value"/>
                </xsl:for-each>
                <xsl:if test="correlator/@seconds">
                    <xsl:text>  AND  (CO.date  between
(TG.date-</xsl:text>
                    <xsl:value-of
select="correlator/@seconds"/>
                    <xsl:text>) AND TG.date)</xsl:text>
                </xsl:if>
            </request>
        </xsl:for-each>
    </sql>
  </xsl:template>
</xsl:stylesheet>
```

V.6. L'évolution de l'approche de surveillance

Notre démarche scientifique de cette approche de surveillance est cohérente puisqu'il y a eu une phase de conception puis une phase de validation. Nous avons testé à l'aide d'un prototype l'ensemble des aspects avancés, de l'approche de mobilité jusqu'à l'approche de surveillance en passant par la corrélation. D'ailleurs la première version du prototype axé uniquement sur la mobilité a été cité et utilisé dans les travaux de Andrea BARBU [80].

Cette phase de validation nous a permis de mettre en avant certaines contraintes auxquelles il faudra répondre avant d'entamer le développement d'une implémentation perspicace et libre d'utilisation.

La problématique actuelle de cette approche de surveillance est liée à la taille des données à collecter, nous pensons que sur un réseau large de fortes activités, les données sont d'une taille importante. A ce jour un agent transporte les données collectées, s'il s'agit d'une ressource disposant de plusieurs données nécessitant plusieurs allers-retours qui retarde le processus d'analyse de ces dernières. Ce retard rend l'approche de moins au moins réactive. Aujourd'hui nous posons la question suivante : est ce qu'il est essentiel que l'agent transporte des données ?

Cette propriété de l'agent mobile est jugée importante, en plus elle constitue un critère de comparaison. Certes l'agent doit être capable de transporter des données nécessaires à son fonctionnement (Message, parcours, stratégies…) mais peut-être pas les données qu'il collecte. Il faut admettre que cette problématique est spécifique à une approche basée sur des agents mobiles, s'il s'agissait d'une approche basée sur des échanges classiques comme des flux réseau le problème ne se poserait pas.

Actuellement, nous travaillons sur ce sujet pour définir d'autres canaux pour transporter les données au lieu de définir des tailles limitées, nous devons fournir à l'agent collecteur un critère sous forme de règles ; par exemple, pour choisir le mode adéquat pour transmettre les données qu'il collecte. Ce mode peut être un transfert FTP, par échange réseau (flux) ou simplement via un messager si la donnée est jugée moins importante. La résolution de cette problématique va remettre en cause la démarche des transactions, ainsi toutes les tâches de collecte peuvent être exécutées en parallèle sans se soucier de quel agent collecteur transmettra les données.

Nous pouvons envisager, comme pour la spécification MASIF, deux environnements : un pour le contrôle (la mobilité des agents) et un autre pour le transfert des données. Ce nouveau réseau ou environnement n'est plus ou moins qu'une extension de la JVM de chaque agent mobile. Cette extension peut s'opérer par l'utilisation de l'architecture TerraCotta [133]. Cette dernière est pensée pour minimiser le dialogue réseaux, garantir l'absence de perte d'objet, fournir un control maximum, tout en assurant l'obtention d'une haute disponibilité et l'évitement des interruptions de service (SPOF).

TerraCotta dépasse les solutions classiques de clustering fonctionnant pour les applications JEE, soit en s'appuyant sur JMS pour synchroniser différentes instances d'une application tournant sur différentes JVM, ou sur des solutions de cache distribué du type JCache (JSR 107) [134]. De façon générale, ces solutions passent par la sérialisation. TerraCotta propose une approche différente, le clustering est proposé directement au niveau JVM.

L’architecture de TerraCotta présente les caractéristiques suivantes (cf. Figure 37) :

- Client Nodes : Chaque Noeud Client correspond à un noeud dans le cluster. Les noeuds clients s’exécutent dans une JVM standard. TerraCotta est installée dans la JVM par le biais des librairies TerraCotta chargées au démarrage de la JVM.
- Terracotta Server Cluster : Terracotta Server Cluster fournit l’intelligence pour la mise en cluster. Chaque serveur au sein du cluster est un processus 100% Java. L’implémentation actuelle de TerraCotta opère dans le mode Actif/Passif, avec un serveur actif et un ou plusieurs serveurs passifs.
- Storage : Le serveur TerraCotta utilise le stockage sur disque pour différentes raisons :
    - Virtual Heap storage : si la mémoire, pour les objets, est pleine alors les objets sont paginés sur disque
    - Lock Arbiter : pour s’assurer de ne pas rencontrer le problème classique du "split-brain", TerraCotta s’appuie sur l’infrastructure disque pour fournir un verrou.
    - Shared Storage : pour passer les objets de l’état actif à passif, les objets doivent persister sur disque (des objets persistants).

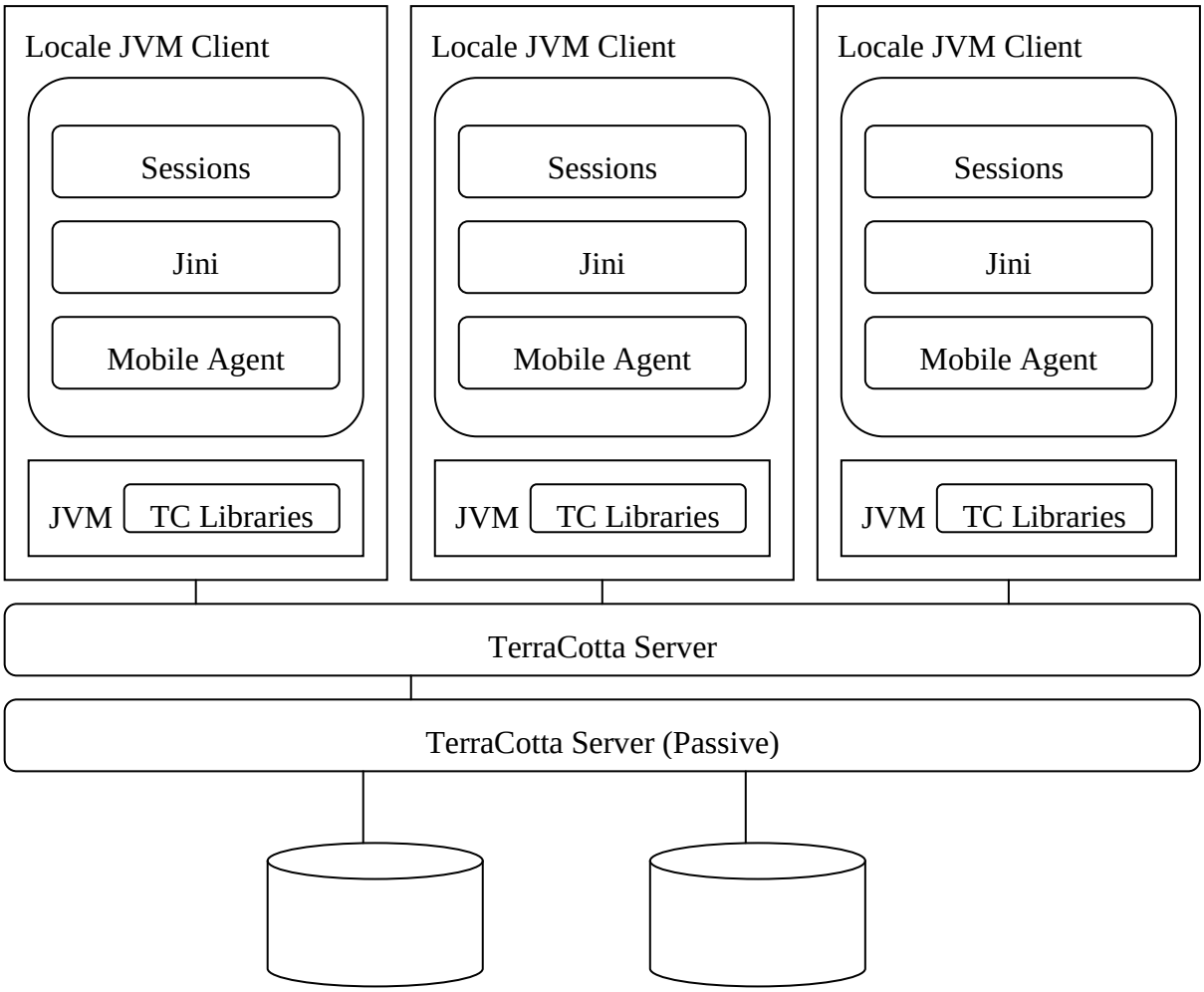

Figure 37 Terracotta architecture

D'autres améliorations de notre approche de surveillance sont à prévoir. Dans un premier temps nous devons revoir la stratégie de la surveillance, englobant le déplacement, le comportement et l'autonomie, trois aspects très liés. Aujourd'hui, les agents collecteurs opèrent à la demande d'un agent analyseur qui, à son tour, s'appuie beaucoup sur les paramètres définis par l'utilisateur et utilise des règles de base prédéfinies comme la création des agents collecteurs en cas de nécessité. D'ailleurs quand ces agents collecteurs sont créés et une fois leurs missions respectives terminées, ils restent inactifs. Par conséquent, ce sont des ressources perdues. Aussi, les agents collecteurs s'exportent vers des hôtes via l'agent hôte qui assure leurs migrations. Sauf que ce dernier n'a pas forcement le choix, il agit machinalement sans pouvoir interdire des déplacements quand la situation ne le permet pas.

Dans un deuxième et dernier temps, nous examinons l'idée d'avoir un agent mobile "caméléon" à la place de quatre agents. Ce caméléon va intégrer les domaines fonctionnels

de tous les agents mobiles et s'adapter en fonction de l'environnement (la présence d'autres agents mobiles). Certes la taille de cet agent sera plus importante. Seulement, il pourra pallier au problème d'interruption du processus de surveillance durant l'absence d'un composant névralgique. Aujourd'hui, dans un réseau si aucun agent annuaire ou analyseur n'est disponible alors la surveillance ne peut pas se faire.

Nous pensons que ces améliorations peuvent se résoudre conjointement par la mise en place d'un système de règles, tel le cas de la plateforme JADE qui utilise JESS, un moteur de règles pour intégrer une approche de logique ou de raisonnement. Nous pouvons également, se tourner vers un outil ou une approche de système expert d'aide à la décision.

Conclusion

Nos études ont abouti à mettre en place une architecture logicielle permettant la circulation d'agents mobiles au sein d'une communauté. Cette dernière a un rôle structurant pour l'activité réalisée. Nous avons montré dans une première partie l'importance des choix de concept afin de construire des agents mobiles autonomes et adaptables à leur contexte d'exécution. Nous avons souligné le rôle de certains choix techniques dans cette approche.

Notre démarche scientifique n'aurait pas été satisfaisante sans une validation de ces choix par l'expérimentation. En effet, comment juger de la pertinence de résultats sans les mettre en situation réelle. Ce besoin de valider nos résultats, a d'abord été satisfait dans un cadre de surveillance logicielle. Nous avons pu montrer qu'une stratégie de surveillance a besoin de changement par construction. De plus, nous avons répondu à cette exigence par un prototype permettant l'observation du comportement d'une application type. Nos premiers résultats soulignent d'ailleurs les limites de l'architecture de cette application.

Enfin, nous nous sommes intéressés au problème de la sécurité logicielle, domaine où les changements sont continus car les attaques sont toujours nouvelles et pour lesquelles les acteurs appliqués doivent être élaborés en relation étroite avec leur caractère. Notre approche de la mobilité s'est concrétisée grâce à un nouveau prototype qui nous a permis de montrer la diminution du trafic des données et surtout l'aptitude à faire évoluer une politique de sécurité de manière totalement dynamique.

Il n'en demeure pas moins que ces résultats ne représentent que les débuts de notre travail. En effet, le concept de code mobile sur un réseau de machine est stratégique en génie logiciel. Le besoin d'adaptation au changement est valable pour la plupart des logiciels. Ainsi nous nous intéressons à l'introduction de la mobilité dans des applications par l'intermédiaire de "container de code mobile". De même nos travaux nous ont permis d'appréhender la notion de communauté d'agents mobiles avec le besoin d'échange au sein d'une communauté. Cette notion n'est pas seulement un concept déclaratif, mais possède un rôle important au cours de l'exécution. Nous souhaitons développer cette notion pour gérer davantage la mobilité d'agent par rapport aux frontières d'une communauté et ainsi

pouvoir définir une application comme fonction d'échange entre communautés. C'est le cas de l'application de surveillance où une première communauté d'agents s'intéresse à la collecte dès qu'une autre porte sur l'analyse. Il n'y a pas échange d'agents entre les deux, mais un échange de données. Les frontières de ces communautés peuvent être communes et imperméables à la mobilité.

Enfin, la mobilité ne résout pas que les problèmes de surveillance logicielle. Nous pouvons envisager d'autres domaines tels que l'informatique nomade où il est alors question d'exporter des agents sur des périphériques de proximité (PDA, décodeur numériques..). Ce qui constituera de nouvelles voies pour nos futurs travaux.

Bibliographie

[1] Milojicic, D. "Mobile Agent Applications," *IEEE Concurrency* (7:3), 1999, pp. 80-90.

[2] Lange, D. B. & Oshima, M. Seven good reasons for mobile agents *Commun. ACM, ACM,* 1999*, 42*, 88-89

[3] Fuggetta, A.; Picco, G. P. & Vigna, G. Understanding Code Mobility IEEE Trans. Softw. Eng., IEEE Press, 1998, 24, 342-361

[4] Thorn, T. Programming languages for mobile code ACM Comput. Surv., ACM, 1997, 29, 213-239

[5] CROS, C. C. D. Agents Mobiles Coop´erants pour les Environnements Dynamiques Institut National Polytechnique de Toulouse, 2005

[6] Knabe, F. An Overview of Mobile Agent Programming Selected papers from the 5th LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages, Springer-Verlag, 1997, 100-115

[7] Lieberherr, K. J. & Holland, I. M. Assuring Good Style for Object-Oriented Programs IEEE Softw., IEEE Computer Society Press, 1989, 6, 38-48

[8] Dömel, P. Mobile Telescript Agents and the Web Computer Conference, IEEE International, IEEE Computer Society, 1996, 0, 52

[9] Cardelli, L. A language with distributed scope POPL '95: Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM, 1995, 286-297

[10] Baumann, J.; Hohl, F.; Rothermel, K. & Stra, M. Mole -- Concepts of a mobile agent system World Wide Web, Kluwer Academic Publishers, 1998, 1, 123-137

[11] Kurotsuchi, B. T. The wonders of Java object serialization Crossroads, ACM, 1997, 4, 3-8

[12] Suri, N.; Bradshaw, J. M.; Breedy, M. R.; Groth, P. T.; Hill, G. A.; Jeffers, R.; Mitrovich, T. S.; Pouliot, B. R. & Smith, D. S. NOMADS: toward a strong and safe mobile agent system AGENTS '00: Proceedings of the fourth

international conference on Autonomous agents, ACM, 2000, 163-164

[13] Binder, W.; Hulaas, J. G. & Villazón, A. Portable resource control in Java OOPSLA '01: Proceedings of the 16th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, ACM, 2001, 139-155

[14] Rouvais, S. Utilisation d'agents mobiles pour la construction de services distribués Université Rennes 1, 2002

[15] Loulergue, F. Développement d'applications avec Objective CAML by E. Chailloux, P. Manoury and B. Pagano, O'Reilley, 2003 J. Funct. Program., Cambridge University Press, 2004, 14, 592-594

[16] Waldo, J. Arnold, K. (ed.) The Jini Specifications Addison-Wesley Longman Publishing Co., Inc., 2000

[17] Poslad, S. & Charlton, P. Standardizing agent interoperability: the FIPA approach Springer-Verlag New York, Inc., 2001, 98-117

[18] Milojičić, D.; Breugst, M.; Busse, I.; Campbell, J.; Covaci, S.; Friedman, B.; Kosaka, K.; Lange, D.; Ono, K.; Oshima, M.; Tham, C.; Virdhagriswaran, S. & White, J. MASIF, the OMG Mobile Agent System Interoperability Facility ACM Press/Addison-Wesley Publishing Co., 1999, 628-641

[19] Magedanz, T. Activities Agents Technology in Europe ACTS Activities, 1999

[20] Tsalgatidou, A. & Pilioura, T. An Overview of Standards and Related Technology in Web Services Distrib. Parallel Databases, Kluwer Academic Publishers, 2002, 12, 135-162

[21] Zhang, J.; Wang, Y. & Varadharajan, V. Mobile Agent and Web Service Integration Security Architecture SOCA '07: Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications, IEEE Computer Society, 2007, 172-179

[22] Mourlin, F. La mobilité : une autre approche logicielle Université Paris 12 - Val de marne, 2008

[23] Wong, D.; Paciorek, N.; Walsh, T.; DiCelie, J.; Young, M. & Peet, B. Concordia: An Infrastructure for Collaborating Mobile Agents MA '97: Proceedings of the First International Workshop on Mobile Agents, Springer-

Verlag, 1997, 86-97

[24] White, J. E. Telescript technology: mobile agent ACM Press/Addison-Wesley Publishing Co., 1999, 460-493

[25] Bäumer, C. & Magedanz, T. Grasshopper - A Mobile Agent Platform for Active Telecommunication IATA '99: Proceedings of the Third International Workshop on Intelligent Agents for Telecommunication Applications, Springer-Verlag, 1999, 19-32

[26] Milojičić, D. S.; LaForge, W. & Chauhan, D. Mobile objects and agents (MOA) COOTS'98: Proceedings of the 4th conference on USENIX Conference on Object-Oriented Technologies and Systems, USENIX Association, 1998, 13-13

[27] Tai, H. & Kosaka, K. The Aglets project Commun. ACM, ACM, 1999, 42, 100-101

[28] Lange, D. B. & Mitsuru, O. Programming and Deploying Java Mobile Agents Aglets Addison-Wesley Longman Publishing Co., Inc., 1998

[29] Bellifemine, F.; Poggi, A. & Rimassa, G. JADE: a FIPA2000 compliant agent development environment AGENTS '01: Proceedings of the fifth international conference on Autonomous agents, ACM, 2001, 216-217

[30] Ametller, J., R. S. & Borrel, J. Agent migration over FIPA ACL messages Proceedings of Fifth International Workshop on Mobile Agents for Telecommunication Applications, 2003, 210-219

[31] Hill, E. F. Jess in Action: Java Rule-Based Systems Manning Publications Co., 2003

[32] Picco, G. P.; Murphy, A. L. & Roman, G.-C. LIME: Linda meets mobility ICSE '99: Proceedings of the 21st international conference on Software engineering, ACM, 1999, 368-377

[33] LIME: A Middleware for Physical and Logical Mobility ICDCS '01: Proceedings of the The 21st International Conference on Distributed Computing Systems, IEEE Computer Society, 2001, 524

[34] Ohsuga, A.; Nagai, Y.; Irie, Y.; Hattori, M. & Honiden, S. Plangent: An Approach To Making Mobile Agents Intelligent IEEE Internet Computing,

IEEE Educational Activities Department, 1997, 1, 50-57

[35] Jérémy Buisson, Françoise André, J.-L. P. A negotiation-based approach of consistency for dynamic adaptation I R I S A, 2005

[36] Sudmann, N. P. TACOMA - fundamental abstractions supporting agent computing in a distributed environment 1996

[37] Demers, A.; Greene, D.; Hauser, C.; Irish, W.; Larson, J.; Shenker, S.; Sturgis, H.; Swinehart, D. & Terry, D. Epidemic algorithms for replicated database maintenance PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing, ACM, 1987, 1-12

[38] Bernich, M. & Mourlin, F. Mobile agent communication scheme Systems and Networks Communication, International Conference on, IEEE Computer Society, 2006, 0, 6

[39] Bernichi, M. & Mourlin, F. Software management based on mobile agents ICSNC '07: Proceedings of the Second International Conference on Systems and Networks Communications, IEEE Computer Society, 2007, 64

[40] Guimaraes, M. & Murray, M. Overview of intrusion detection and intrusion prevention InfoSecCD '08: Proceedings of the 5th annual conference on Information security curriculum development, ACM, 2008, 44-46

[41] Zanero, S. & Savaresi, S. M. Unsupervised learning techniques for an intrusion detection system SAC '04: Proceedings of the 2004 ACM symposium on Applied computing, ACM, 2004, 412-419

[42] Beale, J. Caswell (ed.) Snort 2.1 Intrusion Detection, Second Edition Syngress, 2004, 751

[43] Pillou, J.-F. Dunod (ed.) Sécurite Informatique 2005

[44] Kuri, J.; Navarro, G.; Mé, L. & Heye, L. A Pattern Matching Based Filter for Audit Reduction and Fast Detection of Potential Intrusions RAID '00: Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection, Springer-Verlag, 2000, 17-27

[45] IBM Tivoli Composite Application Manager V6.0 Family: Installation, Configuration, And Basic Usage Vervante, 2006

[46] Hagimont, D. & Ismail, L. A performance evaluation of the mobile agent paradigm OOPSLA '99: Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, ACM, 1999, 306-313

[47] Ganame, A. K. Conception d'un systéme de collecte et d'analyse d'événements de séecurité distribué pour les réseaux multi-sites Université de Franche-Comté, 2007

[48] Yokoo, M. & Hirayama, K. Algorithms for Distributed Constraint Satisfaction: A Review Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers, 2000, 3, 185-207

[49] Finin, T.; Labrou, Y. & Mayfield, J. KQML as an agent communication language MIT Press, 1997, 291-316

[50] Searle, J. The Construction of Social Reality. New York: The Free Press, 1995

[51] Winskel, G. Ltd, M. P. (ed.) The Formal Semantics of Programming Languages, An Introduction 1993

[52] Fuchs, N. E. Specifications are (preferably) executable Softw. Eng. J., Michael Faraday House, 1992, 7, 323-334

[53] I. J. HAYES, C. B. J. Specifications are not Necessary Executable Softw. Eng. J., Michael Faraday House, 1998, 4, 330–338

[54] S. GARLAND, J. G. An overview of LP, the Larch Prover In Proc. of the Third International Conference on Rewriting Techniques and Applications, 1989, 355, 137–151

[55] Wirsing, M. Algebraic specification MIT Press, 1990, 675-788

[56] Meseguer, J. Rewriting Logic as a Semantic Framework for Concurrency: a Progress Report CONCUR '96: Proceedings of the 7th International Conference on Concurrency Theory, Springer-Verlag, 1996, 331-372

[57] Medvidovic, N.; Rosenblum, D. S.; Redmiles, D. F. & Robbins, J. E. Modeling software architectures in the Unified Modeling Language ACM Trans. Softw. Eng. Methodol., ACM, 2002, 11, 2-57

[58] Spivey, J. M. The Z notation: a reference manual Prentice-Hall, Inc., 1989

[59] Abrial, J.-R. The B-Book, assigning programs to meaning Cambridge University Press, 1996

[60] Jourdan, M. & Maraninchi, F. A Modular State/Transition Approach for programming realtime systems In ACM Worshop on Language, Compiler, and Tool Support for Real-Time Systems, Mouriel.Jourdan@imag.fr, 1994

[61] Owre, S.; Rajan, S.; Rushby, J. M.; Shankar, N. & Srivas, M. K. PVS: Combining Specification, Proof Checking, and Model Checking CAV '96: Proceedings of the 8th International Conference on Computer Aided Verification, Springer-Verlag, 1996, 411-414

[62] Filliâtre, J.-C.; Herbelin, H.; Barras, B.; Barras, B.; Boutin, S.; Giménez, E.; Boutin, S.; Huet, G.; Muñoz, C.; Cornes, C.; Cornes, C.; Courant, J.; Courant, J.; Murthy, C.; Murthy, C.; Parent, C.; Parent, C.; mohring, C. P.; mohring, C. P.; Saibi, A.; Saibi, A.; Werner, B. & Werner, B. The Coq Proof Assistant - Reference Manual Version 6.2.4 1999

[63] Milner, R. Communication and concurrency Prentice-Hall, Inc., 1989

[64] Brookes, S. D.; Hoare, C. A. R. & Roscoe, A. W. A Theory of Communicating Sequential Processes J. ACM, ACM, 1984, 31, 560-599

[65] Milner, R.; Parrow, J. & Walker, D. A calculus of mobile processes, I Inf. Comput., Academic Press, Inc., 1992, 100, 1-40

[66] J. ELLSBERGER, D. H. e. A. S. SDL : Formal Object-oriented Language for Communicating Systems. Prentice-Hall, 1997

[67] George, C.; Haxthausen, A. E.; Hughes, S.; Milne, R.; Prehn, S. & Pedersen, J. S. The RAISE Development Method Prentice Hall Int., 1995

[68] Gordon, M. J. C. & Melham, T. F. Introduction to HOL: a theorem proving environment for higher order logic Cambridge University Press, 1993

[69] Jensen, K. Coloured Petri nets: basic concepts, analysis methods and practical use, volume 3 Springer-Verlag New York, Inc., 1997

[70] Budkowski, S. & Dembinski, P. An introduction to Estelle: a specification language for distributed systems Comput. Netw. ISDN Syst., Elsevier

Science Publishers B. V., 1987, 14, 3-23

[71] LOTOS: A Formal Description Technique based on the Temporal Ordering of Observational Behaviour 1989

[72] Peng, R.; He, K. & Zhong, X. SMA Calculus " A Secure Mobile Agent Calculus CIT '04: Proceedings of the The Fourth International Conference on Computer and Information Technology, IEEE Computer Society, 2004, 516-521

[73] Fournet, C.; Gonthier, G.; Lévy, J.-J.; Maranget, L. & Rémy, D. A Calculus of Mobile Agents CONCUR '96: Proceedings of the 7th International Conference on Concurrency Theory, Springer-Verlag, 1996, 406-421

[74] Cardelli, L. & Gordon, A. D. Mobile Ambients FoSSaCS '98: Proceedings of the First International Conference on Foundations of Software Science and Computation Structure, Springer-Verlag, 1998, 140-155

[75] Cardelli, L. & Gordon, A. D. Types for mobile ambients POPL '99: Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM, 1999, 79-92

[76] Sewell, P. & Vitek, J. Secure composition of untrusted code: box &x03C0;, wrappers, and causality types J. Comput. Secur., IOS Press, 2003, 11, 135-187

[77] Bugliesi, M.; Castagna, G. & Crafa, S. Boxed Ambients TACS '01: Proceedings of the 4th International Symposium on Theoretical Aspects of Computer Software, Springer-Verlag, 2001, 38-63

[78] Leavens, G. T. & Sitaraman, M. (ed.) Foundations of component-based systems Cambridge University Press, 2000

[79] Medvidovic, N. & Taylor, R. N. A Classification and Comparison Framework for Software Architecture Description Languages IEEE Trans. Softw. Eng., IEEE Press, 2000, 26, 70-93

[80] Barbu, A. A Mobile application modeled by a formal approach University of Paris12, 2004

[81] Milner, R.; Parrow, J. & Walker, D. A calculus of mobile processes, II Inf. Comput., Academic Press, Inc., 1992, 100, 41-77

[82] Jeffrey, A. Communicating and Mobile Systems: the π-calculus by Robin Milner, Cambridge University Press, 1999 J. Funct. Program., Cambridge University Press, 2001, 11, 433-436

[83] Bergstra, J. A. Ponse, A. & Smolka, S. A. (ed.) Handbook of Process Algebra Elsevier Science Inc., 2001

[84] Sangiorgi, D. & Walker, D. PI-Calculus: A Theory of Mobile Processes Cambridge University Press, 2001

[85] Milner, R. The Polyadic pi-Calculus: a Tutorial Logic and Algebra of Specification, 1991

[86] Sangiorgi, D. From pi-Calculus to Higher-Order pi-Calculus - and Back TAPSOFT '93: Proceedings of the International Joint Conference CAAP/FASE on Theory and Practice of Software Development, Springer-Verlag, 1993, 151-166

[87] Barbu, A. & Mourlin, F. A Higher Order pi-Calculus Specification for a Mobile Agent in JINI LACL, University of Paris12, 2005

[88] Dam, M. Proof systems for &x03C0;-calculus logics Kluwer Academic Publishers, 2003, 145-212

[89] Schmerl, B. & Garlan, D. Exploiting architectural design knowledge to support self-repairing systems SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering, ACM, 2002, 241-248

[90] Oreizy, P.; Gorlick, M. M.; Taylor, R. N.; Heimbigner, D.; Johnson, G.; Medvidovic, N.; Quilici, A.; Rosenblum, D. S. & Wolf, A. L. An Architecture-Based Approach to Self-Adaptive Software IEEE Intelligent Systems, IEEE Educational Activities Department, 1999, 14, 54-62

[91] Medvidovic, N. ADLs and dynamic architecture changes Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops, ACM, 1996, 24-27

[92] Le Métayer, D. Software architecture styles as graph grammars SIGSOFT '96: Proceedings of the 4th ACM SIGSOFT symposium on Foundations of

software engineering, ACM, 1996, 15-23

[93] Le Métayer, D. Describing Software Architecture Styles Using Graph Grammars IEEE Trans. Softw. Eng., IEEE Press, 1998, 24, 521-533

[94] Hirsch, D.; Inverardi, P. & Montanari, U. Graph grammars and constraint solving for software architecture styles ISAW '98: Proceedings of the third international workshop on Software architecture, ACM, 1998, 69-72

[95] Taentzer, G.; Goedicke, M. & Meyer, T. Dynamic Accommodation of Change: Automated Architecture Configuration of Distributed Systems ASE '99: Proceedings of the 14th IEEE international conference on Automated software engineering, IEEE Computer Society, 1999, 287

[96] Wermelinger, M. & Fiadeiro, J. L. Algebraic software architecture reconfiguration ESEC/FSE-7: Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering, Springer-Verlag, 1999, 393-409

[97] Aguirre, N. & Maibaum, T. A Temporal Logic Approach to the Specification of Reconfigurable Component-Based Systems ASE '02: Proceedings of the 17th IEEE international conference on Automated software engineering, IEEE Computer Society, 2002, 271

[98] Endler, M. & Wei, J. Programming generic dynamic reconfigurations for distributed applications Fraunhofer Publica (Germany), 1992

[99] de Paula, V. C.; Justo, G. R. R. & Cunha, P. R. F. Specifying and Verifying Reconfigurable Software Architectures PDSE '00: Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems, IEEE Computer Society, 2000, 21

[100] Oreizy, P. & Taylor, R. On the Role of Software Architectures in Runtime System Reconfiguration CDS '98: Proceedings of the International Conference on Configurable Distributed Systems, IEEE Computer Society, 1998, 61

[101] Canal, C.; Pimentel, E. & Troya, J. M. Specification and Refinement of Dynamic Software Architectures WICSA1: Proceedings of the TC2 First

Working IFIP Conference on Software Architecture (WICSA1), Kluwer, B.V., 1999, 107-126

[102] Cuesta, C. E.; de la Fuente, P. & Barrio-Solárzano, M. Dynamic coordination architecture through the use of reflection SAC '01: Proceedings of the 2001 ACM symposium on Applied computing, ACM, 2001, 134-140

[103] Schmitt, A. & Stefani, J. The Kell Calculus: A Family of Higher-Order Distributed Process Calculi Global Computing, Springer, 2004, 3267

[104] Hennessy, M.; Merro, M. & Rathke, J. Towards a behavioural theory of access and mobility control in distributed systems Theor. Comput. Sci., Elsevier Science Publishers Ltd., 2004, 322, 615-669

[105] Hennessy, M. & Riely, J. Resource access control in systems of mobile agents Inf. Comput., Academic Press, Inc., 2002, 173, 82-120

[106] Yoshida, N. Channel dependent types for higher-order mobile processes POPL '04: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM, 2004, 147-160

[107] Feng, Z.; Yin, J.; He, Z.; Liu, X. & Dong, J. A Novel Architecture for Realizing Grid Workflow Using Pi-Calculus Technology APWeb, 2006, 800-805

[108] Hennessy, M.; Rathke, J. & Yoshida, N. safeDpi: a language for controlling mobile code In Proc. FOSSACS, LNCS 2987, 2003

[109] Gamma, E.; Helm, R.; Johnson, R. & Vlissides, J. Design patterns: elements of reusable object-oriented software Addison-Wesley Longman Publishing Co., Inc., 1995

[110] Diego Bonura, R. C. & Angeletti, M. A Pattern for reactive mobile agent in Genome database annotation Agents in Bioinformatics, University of Camerino, 2002

[111] D. Deugo, F. Oppacher, J. K. & Otte, I. V. Patterns as a Means for Intelligent Software Engineering Patterns as a Means for Intelligent Software Engineering In Proceedings of The International Conference on Artificial Intelligence, CSREA Press, 1999, 605–611

[112] Mourlin, Fabrice Skaita, J. Mobile agent for the database management using

Jini 6th International Conference on Applied Informatics, 2004

[113] Barbu, A. SLP-Modeling using Higher Order Pi-Calculus 2001

[114] A. Barbu, F. M. A Higher Order Pi-Calculus Specification of a Mobile Agent in Jini Fourth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, SNPD'03, 2003

[115] Sunyé, G.; Guennec, A. L. & Jézéquel, J.-M. Design Patterns Application in UML ECOOP '00: Proceedings of the 14th European Conference on Object-Oriented Programming, Springer-Verlag, 2000, 44-62

[116] Namiki, T. Numerical simulation of antennas using three-dimensional finite-difference time-domain method HPC-ASIA '97: Proceedings of the High-Performance Computing on the Information Superhighway, HPC-Asia '97, IEEE Computer Society, 1997, 437

[117] Misljencevic, S. Jini Service Container Universiteit Antwerpen, 2006

[118] Newmarch, J. Foundations of Jini 2 Programming Apress, 2006

[119] Flora, C. D.; Cotroneo, D.; Flora, C. D. & Russo, S. A Jini Framework for Distributed Service Flexibility Proceedings. 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, 2002, 109 – 116

[120] Ledru, P. Smart proxies for Jini services SIGPLAN Not., ACM, 2002, 37, 57-61

[121] Koutsogiannakis, G.; Savva, M. & Chang, J. M. Performance studies of remote method invocation in Java PCC '02: Proceedings of the Performance, Computing, and Communications Conference, 2002. on 21st IEEE International, IEEE Computer Society, 2002, 1-8

[122] Eronen, P. & Nikander, P. Decentralized Jini Security In Proceedings of the Network and Distributed System Security Symposium, 2001

[123] Lai, C. & Gong, L. User authentication and authorization in the java(tm) platform In ACSAC '99: Proceedings of the 15th Annual Computer Security Applications Conference, IEEE Computer Society, 1999, 285

[124] Johnson, R.; Hoeller, J.; Arendsen, A.; Risberg, T. & Kopylenko, D.

Professional Java Development with the Spring Framework Wrox Press Ltd., 2005

[125] Venners, B. Object Integrity A Conversation with Bob Scheifler, Part III, 2002

[126] Venners, B. Security Constraints A Conversation with Bob Scheifler, Part IV, 2002

[127] Venners, B. Dynamic permission grants A Conversation with Bob Scheifler, Part IV, 2002

[128] Abbes, T.; Bouhoula, A. & Rusinowitch, M. Protocol Analysis in Intrusion Detection Using Decision Tree ITCC '04: Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04) Volume 2, IEEE Computer Society, 2004, 404

[129] Abbes, T.; Bouhoula, A. & Rusinowitch, M. On the use of weighted correlation in intrusion detection process. Annals of Telecommunications, 2004, 59, 9- 10

[130] S.J. Russell, P. N. & L., Miclet, F. P (ed.) Intelligence Artificielle 2006

[131] Flack, C. & Atallah, M. J. Better Logging through Formality RAID '00: Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection, Springer-Verlag, 2000, 1-16

[132] Torrens, M.; Weigel, R. & Faltings, B. Constraint-based Algorithms in Java: the JCL Working Notes of the Swiss Workshop on Collaborative and Distributed Systems, 1997

[133] Liebmann, E. & Dustdar, S. Adaptive data dissemination and caching for edge service architectures built with the J2EE SAC '04: Proceedings of the 2004 ACM symposium on Applied computing, ACM, 2004, 1717-1724

[134] Ekaterina, P.-F. & Divitini, M. Collaborative virtual environments for supporting learning communities: an experience of use GROUP '03: Proceedings of the 2003 international ACM SIGGROUP conference on Supporting group work, ACM, 2003, 58-67

Table des équations

Table des figures

Liste des tableaux

Glossaire

ACC	Agent communication Channel
AAFID	Autonomous Agents for Intrusion Detection
ACL	Agent Communication Language
AD	Active Directory
Aglets	Agile Applets
AMS	Agent Management System
API	Application Programming Interface
CAE	Correlation Autonomous Engine
CCS	Calcul des Systèmes Communicants
CDF	Common Data Format
CIC	Comité d'Intérêt Commun
CLF	Common Log Format
CORBA	Common Object Request Broker Architecture
CPN	Coloured Petri Nets
CSP	Constraint Satisfaction Problem
CSP	Communicating Sequential Processes
DAE	Distributed Agent Environment
DF	Director Facilitor
DGC	Distributed Garbage Collector
DSM	Date Source Mediator
ECE	Event Correlation Engine
EJB	Enterprise JavaBeans
FIPA	Foundation for Intelligent Physical Agents
FTP	File Tranport Protocl
FTS	Federation Tuple Space
GPL	General Public License
GUI	Graphical User Interface
HIDS	Host Intrusion Detection System
HTTP	HyperText Transfer Protocol
IDMEF	Intrusion Detection Message Exchange Format
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IEEE	Institute of Electrical and Electronics Engineers

IIOP	Internet Inter-ORB Protocol
ITS	Interface Tuple Space
JAAS	Java Authentication and Authorization Service
JADE	Java Agent DEvelopment Framework
JCL	Java Constraint Library
JEE	Java Enterprise Edition
JERI	Jini Extensible Remote invocation
JESS	Java Expert System Shell
JMS	Java Message
JRMP	Java Remote Method Protocol
JSP	Java Server Page
JVM	Java Virtual Machine
KQML	Knowledge Query and Manipulation Language
LACL	Laboratoire d'Algorithmique, Complexité et Logique
LAN	Locale Aera Network
LDAP	Lightweight Directory Access Protocol
LGPL	Lesser General Public License
LIME	Linda In Mobile Environnement
MASIF	Mobile Agent System Interoperability Interoperability
MSDN	Jini Extensible Remote Invocation
NIDS	Network Intrusion Detection System
OMG	Object Management Group
PCRE	Perl Compatible Regular Expressions
PDA	Personal Digital Assistant
PHP	PHP: Hypertext Preprocessor
PME	Petites et Moyennes Entreprises
RMA	Remote Management Agent
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SDL	Specification and Description Language
SFC	Servertec Foundation Classes
SGBD	Système de gestion de base de données
SMA	System Multi Agents
SMS	Short Message Service
SPOF	Single Point Of Failure
SQL	Structured Query Language

SSL	Secure Socket Layer
TACOMA	Troms And COrnell Moving Agents
UML	Unified Modeling Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language
XSL	Extensible stylesheet language
XSLT	Extensible Stylesheet Language Transformations
XSLTC	Extensible Stylesheet Language Transformations Compiler

Résumé

Les agents mobiles peuvent physiquement migrer à travers un réseau informatique dans le but d'effectuer des tâches sur des machines, ayant la capacité de leur fournir un support d'exécution. Ces agents sont considérés comme composants autonomes, une propriété qui leur permet de s'adapter à des environnements dynamiques à l'échelle d'un réseau large.

Ils peuvent également échanger des informations entre eux afin de collaborer au sein de leur groupe, nous parlerons ainsi d'une communauté d'agents mobiles. Nous avons développé ce concept de communauté, en se référant aux recherches et aux études précédentes pour définir un nouveau modèle comportemental d'agent mobile.

Ce modèle est utilisé pour répondre aux besoins de la surveillance logicielle. Celle ci consiste à collecter des événements à partir de plusieurs sources de données (Log, événements système…) en vue de leur analyse pour pouvoir détecter des événements anormaux. Cette démarche de surveillance s'appuie sur plusieurs types d'agents mobiles issus du même modèle. Chaque type d'agent gère un domaine fonctionnel précis. L'ensemble des ces agents constitue une communauté pouvant collaborer avec différentes autres communautés lorsqu'il existe plusieurs sites à surveiller.

Les résultats de cette approche nous ont permis d'évoquer les limites liées à la taille des données collectées, ce qui nous amène à de nouvelles perspectives de recherche et à penser un agent mobile "idéal".

Enfin, nous nous intéressons également à l'application de la communauté d'agent mobile pour les systèmes de détection d'intrusion et la remontée d'anomalie.

Mots-clés : Agent Mobile, Communauté, Surveillance logicielle, Communication, Collecte des données, Jini, Modèle de corrélation, IDS.

Abstract

Mobile agents can physically travel across a network, and perform tasks on machines, that provide agent hosting capability. These agents are autonomous; this property allows them to adapt themselves on a dynamic environment in a large network. Also, they can exchange information and data in order to collaborate within their group; in this case we can talk about community of mobile agents. We refer to previous studies and research to develop this concept of community by defining a new behavioural pattern of mobile agent. This pattern is used in monitoring software approach which consist of collecting events from various data sources (log file, OS events…) and analyse them to detect abnormal events. This approach is based on different kind of mobile agents, each kind manages some features. Whole of those mobile agents constitute a community which collaborate with other communities if there are a several sites to supervise.

The results of this approach allow us to evoke some limits related to size of collected data. This limit pushes us to have a new possibility of research and probably define an ideal mobile agent.

Lastly, we illustrate our mobile approach with results about intrusion detection system application to retrieve anomalies.

Keywords: Mobile agent Mobile, Community, Software Monitoring, Communication, Data Collecting, Jini, Correlation Pattern, IDS