

Remerciements

Je voudrai exprimer ma profonde reconnaissance pour mes directeurs de thèse, Monsieur **Mohamed JMAIEL**, Professeur à l'Ecole Nationale d'Ingénieurs de Sfax et Directeur de l'unité de Recherche en Développement et Contrôle d'Applications Distribuées (ReDCAD), et Monsieur **Mohamed MOSBAH**, Professeur à l'Institut Polytechnique de Bordeaux (ENSEIRB-MATMECA), pour la confiance qu'ils m'ont accordée en me déléguant ce travail. Ce travail a grandement profité de leurs encadrements, de leurs grandes compétences et de leurs qualités humaines.

Je tiens à remercier vivement Monsieur **Ahmed HADJ KACEM**, Maître de Conférences à la Faculté des Sciences Economiques et de Gestion de Sfax, pour son encadrement tout au long de ce travail. Je lui suis extrêmement reconnaissante pour sa constante disponibilité ainsi que pour la rigueur de raisonnement scientifique qui m'a permis de structurer mes idées. Sûrement les principaux résultats présentés dans cette thèse sont issus de ces discussions et de ces conseils d'ordre méthodologique, technique et rédactionnel.

J'adresse mes remerciements à Monsieur **Faiez GARGOURI**, Professeur à l'Institut Supérieur d'Informatique et de Multimédia de Sfax, pour avoir bien voulu me faire l'honneur de présider le jury de ma thèse.

J'adresse mes sincères remerciements à Monsieur **Frédéric CUPPENS**, Professeur à l'École Nationale Supérieure de Télécommunications de Bretagne, ainsi qu'à Monsieur **Maher KHEMAKHEM**, Maître de conférences à l'Institut Supérieur de Gestion de Sousse, pour l'intérêt qu'ils ont porté à mon travail en acceptant d'être rapporteurs de cette thèse.

Je tiens à remercier les étudiants de mastère qui, au cours de leurs projets, se sont intéressés à ma problématique de thèse et m'ont apporté du sang neuf et des idées nouvelles. Je cite notamment Mohamed Tounsi et Housseem Aloulou.

Je tiens à remercier également tous les membres de notre équipe de recherche pour leur sympathie et l'ambiance chaleureuse qu'ils donnent lors de nos réunions. Je remercie particulièrement Slim Kallel, Imen Loulou et Amira Regayeg pour les coopérations que nous avons eues et les informations que nous avons échangées.

Table des matières

Introduction Générale	1
Problématique	2
Objectifs	3
Contributions et Organisation de la thèse	4
1 Technologie des agents mobiles	7
1.1 Du Modèle Client-Serveur au Modèle Agent Mobile	7
1.1.1 Client-Serveur	8
1.1.2 Evaluation distante	8
1.1.3 Code à la demande	8
1.1.4 Agent Mobile	9
1.2 Agents Mobiles : Une nouvelle approche de conception	9
1.2.1 Définition	9
1.2.2 Qualités de l'agent mobile	10
1.2.3 Environnement d'exécution des agents mobiles	11
1.3 Efforts de standardisation	11
1.3.1 MASIF	11
1.3.2 FIPA	13
1.3.3 Synthèse	15
1.4 Modélisation des systèmes à base d'agents mobiles	15
1.4.1 Modélisation structurelle	16
1.4.2 Modélisation comportementale	18
1.4.3 Synthèse	19
1.5 Conclusion	20
2 Sécurité & Mobilité : État de l'art	21
2.1 Politiques de sécurité	22
2.1.1 Définitions et Fondements	22
2.1.2 Modélisation de la sécurité	23
2.1.2.1 Modèles de contrôle d'accès	23
2.1.2.2 Modèles de contrôle de flux	24
2.1.2.3 Modèles de contrôle d'usage	24
2.1.3 Approches de spécification des politiques de sécurité	25
2.1.3.1 Spécification à travers un langage métier	25

2.1.3.2	Spécification à base de règles	26
2.1.3.3	Spécification à base de logique	28
2.1.3.4	Synthèse	29
2.1.4	Mécanismes d'imposition des politiques de sécurité	29
2.2	Sécurité des systèmes à base d'agents mobiles	30
2.2.1	Besoins de sécurité	30
2.2.2	Solutions de sécurité adaptées	32
2.2.3	Solutions de sécurité spécifiques	33
2.3	Discussion : Une nouvelle perspective de sécurité	34
2.4	Conclusion	35
3	Spécification Formelle des Systèmes à base d'Agents Mobiles	37
3.1	La spécification formelle : langage & outil de preuve	38
3.1.1	La Notation Z	38
3.1.2	Z/EVES : outil de preuve formelle	39
3.2	Modélisation conceptuelle des systèmes à base d'agents mobiles	40
3.2.1	Cycle de vie de l'Agent Mobile	40
3.2.2	Taxonomie commune : Spécification	41
3.2.2.1	Système d'exécution d'agents mobiles	41
3.2.2.2	Agent de Service	42
3.2.2.3	Agent Mobile	43
3.3	Conclusion	46
4	Cadre formel pour la sécurité des systèmes à base d'agents mobiles	47
4.1	Cadre de spécification	48
4.1.1	Politique de sécurité : Spécification	48
4.1.2	Politique de sécurité : Raffinement	50
4.2	Cadre de vérification	52
4.2.1	Preuve formelle de la consistance des spécifications	53
4.2.2	Preuve formelle de la consistance intra-politique	53
4.3	Cadre de reconfiguration	58
4.4	Cadre d'adaptation	60
4.4.1	Identification et Spécification des différents cas d'incohérences	61
4.4.2	Spécification de la politique d'adaptabilité	64
4.5	Conclusion	70
5	Approche formelle pour la spécification et la vérification des attaques	71
5.1	Taxonomie des attaques	72
5.1.1	Attaques contre l'authentification	72
5.1.2	Attaques contre la confidentialité	72
5.1.3	Attaques contre l'intégrité	73
5.1.4	Attaques contre la disponibilité	73
5.2	Approche proposée	74
5.3	Prévention de l'attaque <i>DoS</i>	74

5.3.1	Spécification formelle de l'attaque <i>DoS</i>	74
5.3.2	Consistance des spécifications de l'attaque <i>DoS</i>	79
5.3.3	Preuve formelle pour la prévention contre l'attaque <i>DoS</i>	80
5.4	Conclusion	81
6	Déploiement des politiques de sécurité	83
6.1	Programmation Orientée Aspect	84
6.1.1	Terminologie de la POA	85
6.1.1.1	Aspect	85
6.1.1.2	Coupe	85
6.1.1.3	Code advice	86
6.1.2	Tissage d'aspects	86
6.1.2.1	Tissage statique	86
6.1.2.2	Tissage dynamique	87
6.2	RDyMASS : Approche pour l'imposition des politiques de sécurité	88
6.2.1	Étape 1 : Définition d'un template d'aspects de sécurité	88
6.2.2	Étape 2 : Génération des aspects de sécurité	91
6.2.3	Étape 3 : Tissage d'aspects	91
6.3	Choix d'implémentation de RDyMASS	91
6.3.1	Choix du tisseur d'aspect	91
6.3.1.1	Terminologie et Syntaxe de JBoss AOP	93
6.3.2	Choix de la plateforme de développement des agents mobiles	95
6.3.2.1	plateforme Aglets	95
6.4	Implémentation de RDyMASS	96
6.4.1	Template d'aspect de sécurité spécifique pour JBoss AOP	97
6.4.2	Générateur d'aspect de sécurité selon JBoss AOP	98
6.4.2.1	Fonctionnalités du générateur de code	99
6.4.3	Etude de cas	100
6.4.3.1	Préoccupations fonctionnelles de l'étude de cas	100
6.4.3.2	Imposition d'un exemple de règle de sécurité	101
6.5	Conclusion	102
	Conclusion Générale	103
	Bibliographie	105

Table des figures

1.1	Cycle de vie de l'agent mobile de FIPA	13
4.1	Théorème associé à la spécification de <i>Contradictory</i>	55
4.2	Traces de preuve du théorème <i>verif_consistency</i>	57
6.1	Impact des aspects sur le code global d'une application	84
6.2	Aperçu générale sur l'approche RDyMASS	89
6.3	Cas d'implémentation de l'approche RDyMASS	97
6.4	Onglet d'ajout d'une nouvelle règle de sécurité	99
6.5	Architecture adoptée pour le commerce électronique	101
6.6	Spécification formelle d'une instance de règle de sécurité	102

Introduction Générale

Désormais la mutation du paysage informatique vers les systèmes distribués n'envisage plus le fonctionnement d'un ordinateur seul sans qu'il interagisse ou coopère avec d'autres ordinateurs à travers un réseau et éventuellement avec une connexion temporaire, voire mobile. Ces systèmes doivent être conçus de manière à répondre de plus en plus à de nouvelles exigences. Ceci se traduit généralement par des impératifs de dynamique et de mobilité. Ainsi, une intense activité de recherche continue à stimuler la communauté en vue d'identifier et de résoudre les problèmes fondamentaux qui surgissent suite à l'intégration de la mobilité en particulier ceux liés à la reconfiguration dynamique et à la sécurité des interactions.

Les systèmes multi-agents ont été mis au point dans un but d'apporter des solutions à ces problèmes sous-jacents. Ils ont ensuite évolué pour devenir une méthode de gestion de la mobilité. L'intérêt d'aborder la mobilité selon l'approche agent réside dans la compatibilité et les convergences entre les paradigmes agent et mobilité.

Cette nouvelle technique a montré son adéquation au développement des applications industrielles dans la gestion des réseaux distribués, les grilles de calcul, les services WEB intelligents, le commerce électronique, les applications multimédias, etc.

La puissance de la technologie des agents mobiles dans la résolution des problèmes complexes résulte du fait que les agents, grâce à leur autonomie, mobilité et adaptabilité, peuvent réaliser leurs buts d'une manière flexible en se servant d'une interaction locale et/ou distante avec les autres agents sur le réseau. Cependant, cette flexibilité soulève des difficultés dans le développement des systèmes à base d'agents mobiles quant à la sécurité du système, la migration des agents et leurs communications, etc.

Notamment, le problème de sécurité constitue un frein à l'expansion de cette technologie. La sécurité des systèmes à base d'agents mobiles est double : elle vise, d'une part, la protection des agents mobiles et, d'autre part, la protection des systèmes d'accueil des agents mobiles. En effet lorsqu'un agent se déplace, il est crucial de s'assurer qu'il sera exécuté correctement en toute sécurité sur le nouveau système visité. De même, il est crucial de rassurer le système d'agents de manière qu'il n'y aura aucun risque d'accueillir un nouvel agent mobile.

La spécification d'une politique de sécurité, pour chacun de ces deux types d'entité, revient à identifier leurs besoins en termes de sécurité et à définir, par la suite, les règlements

à entreprendre pour atteindre le niveau de sécurité visé. L'ensemble de ces réglementations constitue la politique de sécurité.

Dans le contexte des agents mobiles, ces politiques peuvent être sujettes à des changements qui reflètent l'émergence continue de nouvelles menaces de sécurité et l'aspect dynamique des systèmes à base d'agents mobiles. En effet, suite à une migration, l'agent doit s'adapter dynamiquement à la politique de sécurité du système visité.

Problématique

La technologie des agents mobiles représente une progression constante de l'autonomie d'exécution. En particulier, cette propriété a augmenté le degré de complexité des agents. Bien que la notion d'agent mobile soit en cours de standardisation (FIPA, MASIF), nous avons remarqué l'absence d'un fondement formel qui élimine les éventuelles ambiguïtés et imprécisions dans la description des concepts fondamentaux qui caractérisent les systèmes à base d'agents mobiles et dissocie les différents niveaux d'abstraction de ces descriptions. Ainsi, la diversité des notions de base, d'une part, et la complexité des concepts liés aux agents mobiles, d'autre part, impliquent une difficulté à concevoir et à développer un système à base d'agents mobiles.

Plusieurs travaux dans la littérature ont été focalisés sur la modélisation des systèmes à base d'agents mobiles. Ces travaux ont pris des formes très diverses, traitant différents niveaux d'abstraction (structurelle vs. comportementale) et utilisant des notations très variées (semi-formelle vs. formelle). La plupart des travaux [MG01, CPKL06, HLG08] qui ont étudié l'aspect structurel des systèmes à base d'agents mobiles, utilisent des notations graphiques et semi-formelles et par conséquent ils ne permettent pas d'établir des raisonnements et de vérifier des propriétés structurelles du système. En plus, ces travaux sont marqués par une large omission des concepts liés à la sécurité du système.

Quant aux travaux qui ont tiré profit des notations mathématiques [Smi04] pour pouvoir raisonner et vérifier des propriétés, proposent une spécification dépourvue de tous les concepts nécessaires pour exprimer l'autonomie de l'agent et s'intéressent plutôt à modéliser la mobilité de l'agent tout en négligeant les aspects liés à la sécurité.

La deuxième classe des travaux de modélisation, met l'accent sur l'aspect dynamique des agents mobiles tout en décrivant leurs comportements dans un environnement distribué. Plusieurs travaux [SEM04, KWT04, LHG06] s'intéressent à décrire le comportement des agents mobiles en utilisant les diagrammes d'UML et en apportant des extensions à ces diagrammes. Malgré la simplicité et l'expressivité des notations graphiques utilisées, ces travaux souffrent d'un manque de rigueur de raisonnement sur des propriétés du système. En plus, ces travaux s'intéressent uniquement à modéliser les aspects liés à la mobilité de l'agent sans se préoccuper par les concepts liés à la sécurité. De même, les travaux qui ont profité de la puissance des notations mathématiques, s'intéressent à modéliser les protocoles de communication et de migration des agents. En plus, ils souffrent d'un

manque de couverture des concepts qui décrivent les systèmes à base d'agents mobiles.

Ainsi, nous retenons que la majorité des travaux focalisés sur la modélisation des systèmes à base d'agents mobiles, s'intéressent à exprimer la mobilité des agents sans toutefois accorder une importance aux problèmes de sécurité. Par conséquent, il est nécessaire d'explorer davantage la technologie des agents mobiles au regard des problèmes de sécurité.

Les efforts entrepris pour répondre aux problèmes de sécurité dans le contexte des agents mobiles, ont fait apparaître différentes approches de spécification de politique de sécurité. Les travaux dans ce cadre n'ont pas traité les différentes préoccupations de la sécurité. En effet, la majorité des travaux [Bry06, DRF03, MSD01] s'intéressent à spécifier des politiques pour contrôler le comportement des agents et leurs accès aux ressources. En plus, ces travaux sont marqués par un manque de couverture des concepts liés à la définition d'un système à base d'agents mobiles. Par exemple, la représentation de l'agent mobile est généralement limitée par un simple objet dépourvue de tous les concepts nécessaires pour exprimer son autonomie et son aspect cognitif (tels que les croyances, les connaissances et les compétences).

Ce manque d'investigation, s'explique par la double complexité liée, d'une part, à la richesse et la variété des concepts d'expression des politiques de sécurité et, d'autre part, à la richesse des concepts qui décrivent un système à base d'agents mobiles.

Les politiques de sécurité pour les systèmes à base d'agents mobiles s'inscrivent dans un contexte dynamique. En fait, l'émergence continue de nouvelles menaces de sécurité et le besoin d'adaptation de l'agent aux exigences de sécurité d'un nouveau système visité nécessitent une reconfiguration dynamique des politiques de sécurité. Cependant, la plupart des travaux [DRF03, MSD01], dans ce cadre, s'intéressent à la spécification des politiques statiques et non configurables. D'autres travaux, ont considéré l'aspect dynamique à un niveau d'implémentation [UE06] ce qui rend difficile la vérification des propriétés de la nouvelle politique après reconfiguration. L'enjeu est alors de faire évoluer la politique en préservant ses propriétés relatives à la consistance et à la préservation du niveau de sécurité visé.

Objectifs

Il est largement acquis que la spécification formelle est une phase primordiale dans toute démarche de conception et d'implémentation de systèmes complexes. En effet, cette phase écarte toute ambiguïté et imprécision dans la description du système et permet de raisonner rigoureusement en termes de propriétés et de comportements. Compte tenu de l'hétérogénéité des composants d'un système à base d'agents mobiles, la complexité de ses mécanismes de contrôle et le manque d'un consensus au sujet de ses concepts fondamentaux, la spécification formelle devient de plus en plus pesante afin de maîtriser cette complexité.

La mobilité de l'agent pose évidemment des problèmes de sécurité. Étant donné la variété des préoccupations de sécurité dans les systèmes d'agents mobiles, il est nécessaire d'utiliser un formalisme qui offre suffisamment d'expressivité pour spécifier les politiques de sécurité aussi bien au niveau statique qu'au niveau dynamique. De plus, il est essentiel que ce formalisme soit supporté par des outils de preuves automatiques pour la vérification des propriétés désirées et la validation des spécifications.

Ainsi, l'objectif de nos travaux de recherche consistent à proposer une approche pour la spécification, la vérification et le déploiement des politiques de sécurité dynamiques dans les systèmes à base d'agents mobiles. Nous proposons d'exploiter la puissance des techniques formelles afin de garantir une spécification cohérente et valide aussi bien au niveau statique qu'au niveau dynamique. En plus, nous profitons des techniques formelles et nous appliquons une approche qui permet de générer, automatiquement, à partir d'une spécification fiable des politiques, le code de sécurité permettant leur imposition.

Contributions et Organisation de la thèse

Les synthèses faites et les réflexions développées dans le cadre de ce travail de thèse ont fait l'objet de six chapitres répartis comme suit

Dans le premier chapitre, nous avons commencé par introduire le contexte d'application des agents mobiles afin d'appréhender la complexité liée à ce domaine. Ensuite, nous avons synthétisé les principaux travaux focalisés sur la modélisation des systèmes à base d'agents mobiles.

Le deuxième chapitre présente un état de l'art des principaux travaux liés à la sécurité des systèmes distribués afin de montrer à quel niveau ils peuvent être appliqués aux problèmes spécifiques aux systèmes à base d'agents mobiles. Ensuite, nous avons étudié les travaux liés à la sécurité des systèmes à base d'agents mobiles pour exhiber leurs limites en regard de leurs avantages.

Les contributions développées dans le cadre de cette thèse ont tiré profit du pouvoir d'expression de la notation Z, son aptitude de modéliser le système à différents niveaux d'abstraction et la facilité de raisonner rigoureusement par le biais de son démonstrateur de théorème Z/EVES. En effet, nous avons proposé, dans le troisième chapitre, une première contribution qui consiste à modéliser conceptuellement les systèmes à base d'agents mobiles selon la notation Z. Cette modélisation constitue un référentiel dans les différentes phases de développement d'un système à base d'agents mobiles. Elle spécifie d'une manière concise et précise les différents concepts liés aux systèmes d'agents mobiles, unifie leurs représentations et définit les relations entre eux.

Notre deuxième contribution, qui a fait l'objet du quatrième chapitre, consiste à définir

un cadre formel pour la sécurité des systèmes à base d'agents mobiles et qui porte sur la spécification, la vérification, la reconfiguration et l'adaptabilité.

Le cadre de spécification propose une définition explicite et générique des politiques de sécurité. Il peut être enrichi par des concepts liés à un ou plusieurs modèles de sécurité. Pour éviter toute anomalie capable de réduire la performance de la politique, nous associons au cadre de spécification un cadre de vérification pour prouver formellement la consistance des spécifications proposées aussi bien que la consistance des règles de sécurité intra politique. Pour répondre aux changements dynamiques des exigences de sécurité dans les systèmes à base d'agents mobiles, nous proposons un troisième cadre pour la reconfiguration des politiques de sécurité. Enfin, nous proposons un cadre qui exprime l'adaptabilité de l'agent aux nouvelles exigences de sécurité du système visité et rendre, par conséquent, sa politique cohérente avec celle du système.

Notre troisième contribution, définie au niveau du chapitre 5, a porté sur la définition d'une approche formelle pour la prévention des attaques dans un système à base d'agents mobiles. Cette approche consiste à définir une bibliothèque d'attaques susceptibles de se présenter dans un système d'agents mobiles et de définir, sur cette base, les actions à entreprendre pour éviter ces attaques. Cette solution jouera un rôle vital pour vérifier la préservation du niveau de sécurité après une opération de reconfiguration.

Dans le dernier chapitre, notre contribution a porté sur la définition d'un cadre opérationnel pour le déploiement des politiques de sécurité dans les systèmes à base d'agents mobiles. Ce cadre tire profit du cadre théorique, que nous avons défini, et applique une approche de génération de code correspondant à une spécification fiable des politiques de sécurité. Cette approche utilise le paradigme de la programmation orientée aspect de ce qu'il offre de modularité et de séparation entre les préoccupations fonctionnelles et les préoccupations techniques d'une application.

Nous avons illustré cette approche par une expérimentation pour sécuriser des transactions de commerce électronique. En fait, nous avons adopté le tisseur JBoss AOP pour tisser dynamiquement des règles de sécurité dans une application développée sous Aglets.

Enfin, ce document s'achève par une conclusion générale ainsi que les perspectives que nous avons tracées pour poursuivre ce travail.

1

Technologie des agents mobiles

L'émergence des réseaux informatiques à grande échelle a donné naissance à de nombreuses applications réparties. Ces applications exigent une forte interaction entre différentes entités distribuées sur le réseau et qui partagent les mêmes ressources et les mêmes buts. Plusieurs modèles d'exécution distribués pour ces applications ont été proposés dans la littérature.

Nous commençons, dans ce chapitre, par présenter brièvement les caractéristiques de quelques modèles d'exécution distribuée utilisés pour la mise en œuvre d'une application répartie. Nous nous focaliserons sur la description du modèle d'agent mobile, qui propose une solution facilitant le développement des applications réparties sur des réseaux à grande échelle. Cette description permet d'appréhender la complexité de ce domaine qui est exprimée par la variété des aspects à considérer dans un tel système.

Pour gérer cette complexité, plusieurs travaux de modélisation ont été proposés. Nous présenterons, dans ce chapitre, une classification des travaux existants. Cette classification nous permet de préciser les points forts et les points faibles des différents modèles proposés.

1.1 Du Modèle Client-Serveur au Modèle Agent Mobile

Nous présentons, dans cette section, quatre principaux modèles pour le développement des application distribuées. Nous finissons par situer le modèle des agents mobiles par rapport aux autres modèles et montrer son adéquation aux réseaux à grande échelle.

1.1.1 Client-Serveur

Le modèle client-serveur [Lef94] était le modèle le plus utilisé pour le développement des application distribuées. Selon ce modèle lorsque qu'un client a besoin d'un service disponible sur un nœud distant du réseau (serveur) il envoie une requête au serveur pour lui exécuter le service en question et lui renvoyer la réponse correspondante.

En effet, l'inconvénient majeur de ce modèle c'est que tous les composants du système (ressource nécessaire pour l'exécution, code à exécuter et état d'exécution du code) sont stationnaires de manière qu'ils ne peuvent interagir que d'une manière distante ce qui augmente le trafic sur le réseau et exige une connexion permanente entre le client et le serveur. Pour remédier à ces problèmes, différentes formes de changements ont été apportés au modèle client-serveur afin de supporter une mobilité au niveau des composants du système et minimiser par conséquent les interactions distantes. Nous distinguons ainsi, trois nouveaux modèles d'exécution distribuée : l'évaluation distante, le code à la demande et les agents mobiles.

1.1.2 Evaluation distante

Le modèle d'évaluation distante [SG90] s'applique dans le cas où l'application cliente dispose d'un savoir faire (code) qui nécessite, pour son exécution, des ressources localisées sur un nœud distant du réseau (serveur). Dans ce cas le client envoie au serveur un code à exécuter. Le serveur se préoccupe de l'exécution de ce code en utilisant ses propre ressources. Une fois le serveur termine l'exécution du code, les résultats obtenus seront transmis au client.

Ainsi et en comparaison avec le modèle client-serveur classique, le modèle d'évaluation distante permet la mobilité du code afin qu'il se rapproche des ressources et s'exécute sans avoir une connexion permanente entre le client et le serveur.

1.1.3 Code à la demande

Par rapport au modèle d'évaluation distante, les rôles dans le modèle de code à la demande seront permutés entre le client et le serveur. En effet, le client dispose d'un ensemble de ressources nécessaires à la réalisation d'un service bien déterminé, mais il ne dispose pas du savoir faire de ce service. De ce fait le client interagit avec le serveur afin de demander le code nécessaire pour exécuter un service donné. Le serveur transmet alors le code au client et l'exécution s'effectua sur la machine cliente.

1.1.4 Agent Mobile

Le modèle d'agent mobile [BI02] offre plus de souplesse par rapport aux deux derniers modèles présentés, dans la mesure où il accepte la mobilité du savoir faire du client vers plusieurs nœuds du réseau afin d'interagir localement avec plusieurs ressources et/ou savoirs-faire éparpillés sur le réseau. En effet, l'agent se déplace d'un nœud à un autre avec un ensemble d'actions à exécuter (formant le code de l'agent) et l'état de son exécution. Une fois l'agent (code client) termine son exécution il doit revenir à son emplacement de départ ou simplement lui renvoyer les résultats obtenus.

Les facultés offertes par le modèle d'agent mobile, permet d'améliorer les performances des applications distribuées qui exigent une forte interaction entre les différentes entités du réseau. En effet, l'agent mobile est capable d'interagir constamment avec les différentes entités du système sans avoir besoin d'une connexion permanente entre les différents nœuds du réseau. En fait, il aura besoin d'une connexion entre deux nœuds uniquement lors de la migration de l'agent.

En plus, la mobilité de l'agent peut être considérée comme une solution afin de décharger la machine cliente et déléguer l'exécution de certaines tâches sur d'autres machines du réseau.

1.2 Agents Mobiles : Une nouvelle approche de conception

Après avoir introduit le concept d'agent mobile, nous nous intéressons, dans cette section, à donner une description plus exhaustive à l'agent mobile, tout en décrivant ses propriétés et les services nécessaires pour son exécution.

1.2.1 Définition

Nous commençons par rappeler brièvement la définition de l'agent stationnaire puis nous précisons comment ce concept a évolué et a encapsulé la mobilité.

Un agent stationnaire [Fer95] est une entité physique ou virtuelle qui est capable d'agir dans un environnement ; qui peut communiquer directement avec d'autres agents ; qui est menue d'un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voir de survie, qu'elle cherche à optimiser) ; qui possède des ressources propres ; qui est capable de percevoir (mais de manière limitée) son environnement ; qui ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune) ; qui possède des compétences et offre des services ; qui peut éventuellement se reproduire et dont le comportement tend à satisfaire ses objectifs, en tenant compte des ressources et des compétences dont elle dispose, et en fonction de sa

perception, de ses représentations et des communications qu'elle reçoit.

Pour rendre ce concept plus adéquat aux besoins des réseaux à grande échelle et à l'informatique nomade, l'aspect 'mobilité' s'est alors associé à l'approche agent pour définir un nouveau concept nommé 'agent mobile'. D'ailleurs, ce concept est destiné à la mise en œuvre d'applications dont les performances varient en fonction de la disponibilité et de la qualité des services et des ressources, ainsi que du volume des données échangées [LAP04].

L'utilisation diversifiée du concept *agent mobile* dans plusieurs domaines d'application, a donné lieu à de très nombreuses définitions. Une définition générale pourrait être la suivante : un agent mobile est un agent qui peut accomplir les propriétés d'un agent stationnaire. En plus, il est capable de se déplacer au cours de son exécution, pour se rapprocher des ressources distantes.

1.2.2 Qualités de l'agent mobile

Pour pouvoir évoluer convenablement dans un système distribué, l'agent mobile doit satisfaire un certain nombre de propriétés. Principalement, il doit être :

- *Autonome* : un agent mobile doit être capable de prendre des décisions afin d'améliorer l'exécution de sa mission. Ainsi, il est capable de se déplacer d'un système à un autre afin de se rapprocher des ressources et/ou des services dont il a besoin. De plus, nous supposons qu'un agent soit capable de se cloner afin de lancer en parallèle l'exécution de ses tâches sur différents systèmes.
- *Communicant* : un agent mobile doit avoir la faculté de communiquer avec les autres agents du système (agents locaux ou distants), afin d'échanger des informations et bénéficier des connaissances et du savoir-faire des autres agents.
En pratique, la mobilité ne se substitue pas aux capacités de communication des agents mais elle les complète (la communication distante, moins coûteuse dans certains cas, reste possible).
- *Sécurisé* : à sa migration, l'agent doit se protéger contre les systèmes d'accueil malveillants. Sans protection, le système d'accueil peut altérer ou détruire les informations sensibles de l'agent tels que les résultats partiels de son exécution.
- *Adaptable* : la mobilité de l'agent nécessite impérativement l'attitude d'adaptation chez l'agent. En effet, pour pouvoir évoluer convenablement entre différents systèmes hétérogènes l'agent doit être capable de s'adapter dynamiquement aux variations de son contexte d'exécution.

Bien évidemment, cette liste est loin d'être exhaustive. Elle peut être étendue par d'autres propriétés telles que la perception, la coopération, négociation, etc.

1.2.3 Environnement d'exécution des agents mobiles

L'environnement d'exécution des agents mobiles est une infrastructure offrant un certain nombre de services de contrôle et de qualité pour assurer, convenablement, l'exécution des agents mobiles. Couramment cette infrastructure permet : la création d'un nouveau agent, l'enregistrement et la réception des agents mobiles entrants, l'activation du code de l'agent, la migration des agents vers d'autres systèmes, la communication (distante ou locale) entre les agents, l'accès aux ressources/services locaux du système, la localisation des agents mobiles, leur sécurité, etc.

Actuellement, plusieurs plateformes de développement d'agents mobiles ont été commercialisées. L'ensemble des services offerts varie d'une plateforme à une autre. Cette variété est due à plusieurs facteurs, notamment le domaine d'application et le modèle d'agent adopté (i.e. réactif ou proactif).

1.3 Efforts de standardisation

La prolifération des plateformes d'exécution d'agents mobiles est marquée par une large incompatibilité entre différents environnements d'exécution d'agents mobiles. En effet, chaque plateforme est conçue par rapport à une architecture et un modèle d'agent particuliers ou par rapport à un domaine d'application spécifique. De ce fait, l'agent ne peut pas migrer vers une machine qui exécute un système différent de son système actuel. Il s'est avéré alors nécessaire de proposer une standardisation des concepts et des fonctionnalités minimales des plateformes d'agents mobiles afin de leur assurer un haut niveau d'interopérabilité.

Dans ce cadre, deux travaux de recherche se sont imposés : la norme MASIF [MBB⁺98] (Mobile Agent System Interoperability Facilities Specification) et la norme FIPA [FfIPA02] (Foundation for Intelligent Physical Agents).

Dans ce qui suit, nous présentons un aperçu général sur les fondements de ces deux normes.

1.3.1 MASIF

MASIF [MBB⁺98] est un standard, pour les systèmes à base d'agents mobiles, qui a été spécifié par l'Object Management Group (OMG). L'objectif principal de ce travail était de proposer un ensemble de définitions et d'interfaces afin d'avoir un niveau d'interopérabilité entre différents agents mobiles. MASIF ne standardise pas les opérations locales des agents telles que l'interprétation de l'agent, sa sérialisation et son exécution. Plutôt, MASIF standardise :

- La gestion des agents par la définition des opérations pour créer un agent, suspendre son exécution, reprendre et terminer son activité.

- Le transfert des agents.
- Le nommage des agents et des systèmes d'accueil des agents par le fait de standardiser la syntaxe et la sémantique des noms.
- Les types des systèmes d'accueil et la syntaxe de leur localisation afin de vérifier si le type du système supporte l'accueil de l'agent.

La norme MASIF exige une infrastructure basée principalement sur les concepts suivants :

- Les *agents* sont des programmes autonomes qui agissent pour le compte d'une personne ou d'une organisation. Ces dernières expriment l'autorité de l'agent.
- Les agents sont hébergés et s'exécutent sur des *places*. Une place est un contexte défini dans un système d'agents (*eng. agent system*).

La place source et la place destination d'un agent mobile peuvent résider dans le même système d'agents.

- Un système d'agents est une plateforme qui peut créer, interpréter, exécuter, transférer et arrêter l'agent.

De même que l'agent, un système d'agents est associé à une autorité qui identifie la personne ou l'organisation pour laquelle il agit.

- Un type de système d'agents (*eng. agent system type*) décrit le profil de l'agent. Un agent se déplace d'une place à une autre si son profil (type du système d'agents, langage et méthode de sérialisation) est reconnu par le système d'agents destination.
- Les systèmes d'agents de même autorité seront regroupés dans une *région*. Ce concept assure la scalabilité.

Cette infrastructure est développée par l'adoption de deux interfaces CORBA : MAFAgentSystem et MAFFinder. Ces deux interfaces ont été définies au niveau du système d'accueil des agents et non pas au niveau de l'agent.

L'interface MAFAgentSystem définit les opérations pour la gestion du cycle de vie de l'agent telles que la création, la suspension, la reprise et la terminaison de l'agent. Aussi, cette interface assure le transfert et la réception des classes d'agents mobiles. La deuxième interface MAFFinder se préoccupe de l'enregistrement et la localisation des agents, des places et des systèmes d'agents.

La norme MASIF est fondée sur des services CORBA à savoir le service de : nommage, cycle de vie, externalisation (sériation) et sécurité. Le service de sécurité définit principalement des fonctions permettant : l'authentification du client pour une création distante des agents, l'authentification mutuelle des systèmes d'accueil d'agents, l'authentification et délégation des agents. Les agents mobiles et les systèmes d'accueil disposent d'une ou plusieurs politiques de sécurité qui gouvernent leurs activités. Cette dernière est composée d'un ensemble de règles pour limiter ou accorder des capacités/accès à l'agent et définir des limites de consommation de ressource. En fait, une politique est définie en fonction de l'authenticité des parties communicantes, la classe d'agent, l'autorité de l'agent, et/ou plusieurs autres facteurs.

1.3.2 FIPA

FIPA est une organisation de standardisation fondée en 1996 à Genève (Suisse) dont l'objectif est de spécifier des standards logiciels pour assurer l'interopérabilité entre les agents et les applications à base d'agents.

Les spécifications FIPA sont réparties en cinq catégories :

- Les applications : représentent des exemples de domaines d'application sur lesquels peuvent être déployer des agents FIPA.
- Les architectures abstraites : ces spécifications s'intéressent à la définition des entités abstraites nécessaires pour le développement d'un environnement d'agents.
- La gestion des agents : ces spécifications traitent le contrôle et la gestion des agents dans/entre les plateformes d'agents.
- Le transport des messages d'agents : ces spécifications traitent le transport et la représentation des messages à travers différents protocoles réseaux.
- La communication des agents : ces spécifications traitent les messages d'ACL (Agent Communication Language), les protocoles d'échange de message, etc.

FIPA a publié plusieurs documents qui présentent les spécifications pour ces différents catégories. Ces spécifications ont passé par plusieurs versions FIPA97, FIPA98 et FIPA2000. Dans notre étude nous nous sommes basés sur la version 2000 et plus particulièrement au document intitulé "FIPA Agent Management Support for Mobility Specification" qui s'intéresse à la spécification des besoins et des technologies permettant à l'agent de tenir avantage de la mobilité. Principalement dans ce contexte, les efforts ont été investis dans la spécification de :

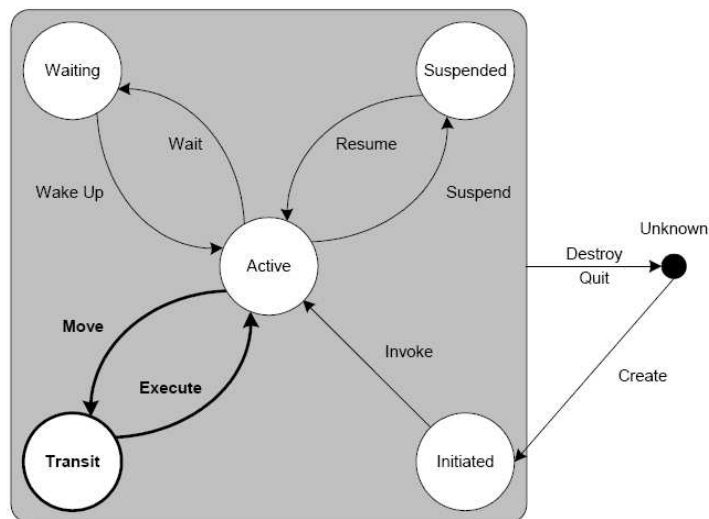


FIG. 1.1 – Cycle de vie de l'agent mobile de FIPA

- *Protocoles de mobilité* : un nombre de protocoles pour couvrir les différentes formes de mobilité de l'agent. Ils s'adressent, spécialement à la migration, au clonage et à l'invocation de l'agent. En fait chaque protocole représente graphiquement les actions et les réactions des entités impliquées (agents et plateformes d'agents).
- *Cycle de vie de la mobilité* : il étend le cycle de vie de l'agent stationnaire par l'ajout d'un nouvel état ('Transit') et deux nouvelles actions pour atteindre/partir de cet état ('Move' et 'Execute'). Ce cycle de vie est illustré par la Figure 1.1.

L'action 'Move' permet de faire passer l'agent dans un état transitoire. Cette action sera initiée par l'agent, par contre l'action 'execute' sera initiée par le système d'exécution d'agents. L'action 'execute' permet de faire sortir l'agent de l'état transitoire et active son exécution. Pour passer à l'état 'Transit', l'agent mobile doit initié l'exécution d'un protocole de mobilité pour aller à un nouveau système.

- *Ontologie de mobilité de l'agent* : elle présente une description textuelle d'un ensemble d'objects et de fonctions pour assurer la mobilité de l'agent.

Tout objet est décrit par un ensemble de paramètres, dont chacun est représenté par une description (qui présente une description textuelle détaillée de la sémantique de ce paramètre), une présence (mandatory/optional : qui indique si le paramètre est obligatoire ou facultatif), un type (qui indique le type des valeurs de ce paramètre : Integer, Word, String, URL, Term, Set ou Sequence) et des valeurs réservées (représentent une liste de valeurs qui peuvent être supportées par ce paramètre).

D'une manière similaire, une fonction est décrite d'un symbole (qui l'identifie), un type d'agent (qui supporte cette fonction), une description (description textuelle de la sémantique de cette fonction), un domaine (le domaine auquel est définie une fonction), un range, et une arité (indique le nombre des arguments de cette fonction)

Les problèmes liés à la sécurité, dans un contexte mobile, n'ont pas été traités ni dans cette spécification ni dans ses implémentations tel que FIPA-OS (FIPA Open Source) [SP00]. Pour ce faire, une tentative de définition d'architecture de sécurité a été proposée dans [ZKI01]. Cette architecture définit une structure permettant d'implémenter deux services de sécurité : un service de communication sécurisée et un autre pour une exécution sécurisée. Le premier service fait face à l'écoute ou l'interférence du réseau extérieur. Le deuxième service protège les ressources du serveur d'exécution et les services de l'agent contre les accès non-autorisés.

Actuellement, plusieurs plateformes d'agents mobiles supportent le standard de FIPA, telles que JADE [BPR99] et ZEUS [NNLC99]. De même, la norme MASIF a été largement utilisée dans le développement des plateformes d'agents mobiles. Grasshopper [BBCM99] a été la première plateforme conforme au standard MASIF.

La complémentarité des aspects définis par MASIF et FIPA a incité plusieurs chercheurs à définir des plateformes conformes aux deux normes [KW03].

1.3.3 Synthèse

Ces efforts de standardisation ont apporté un gain considérable pour la communauté des agents mobiles. Toutefois, ces travaux présentent des limites. Les deux standards présentent une description informelle des concepts (les entités, les états, les services, les protocoles, etc.) nécessaires pour assurer convenablement la migration des agents. En effet, la norme MASIF offre une description textuelle des concepts (Agent, Stationary Agent, Mobile Agent, Agent State, Agent Location, Agent System, Place, etc.) et quelques représentations structurelles afin de faire apparaître les liens entre ces concepts. De même, la norme FIPA a développé une description textuelle des concepts liés au cycle de vie de l'agent mobile et aux protocoles de mobilité de l'agent. Cette description est suivie par une représentation graphique qui illustre l'évolution des états de l'agent et les différentes actions qui peuvent être exécutées par l'agent.

En plus nous avons remarqué, dans ces travaux, que dans la même description différents niveaux d'abstractions sont souvent mêlés. Par exemple, dans FIPA l'ontologie de mobilité de l'agent présente l'ensemble d'objets nécessaires pour assurer la mobilité de l'agent. Cette présentation débute par une description abstraite de chacun des paramètres d'un objet pour finir par la définition du type de données de ce paramètre (Integer, Word, String, URL) ainsi que ses valeurs possibles.

De même, le modèle conceptuel de la norme MASIF présente à la fois une définition abstraite des composants d'un système à base d'agents mobiles, et des détails liés à l'implémentation tels que le processus de sérialisation/désérialisation des agents et le 'codebase' qui spécifie la localisation des classes utilisées par un agent qui peut être soit un 'agent system' ou un objet non-CORBA.

Pour pallier à ces limites et approcher la complexité des systèmes à base d'agents mobiles, il est nécessaire d'utiliser dans leurs descriptions un formalisme bien fondé. Ce dernier doit être défini par trois éléments essentiels [Har08] : une syntaxe abstraite, une sémantique précise et une syntaxe concrète. La syntaxe abstraite définit les concepts de base du formalisme. La sémantique exprime les interprétations possibles à ces concepts. Et enfin, la syntaxe concrète définit le type de notation qui sera utilisé (textuelle, graphique, formelle ou mixte), et donne à chaque concept abstrait une représentation concrète dans cette notation.

1.4 Modélisation des systèmes à base d'agents mobiles

Dans la littérature, de nombreux formalismes ont été appliqués pour la modélisation des systèmes à base d'agents mobiles. Ces modélisations ont pris des formes très diverses, partant d'une description statique et structurelle du système jusqu'à une représentation formelle et rigoureuse des changements comportementaux du système.

En effet, un système peut être modélisé à différents niveaux d'abstraction (structurelle vs. comportementale) en utilisant des notations très variées (semi-formelle vs. formelle).

Suivant ces deux axes, nous proposons une classification des travaux focalisés sur la modélisation des systèmes à base d'agents mobiles et nous donnons un bref aperçu de ces travaux afin de souligner leurs apports et leurs limites.

1.4.1 Modélisation structurelle

La classification suivant le niveau d'abstraction nous permet de distinguer deux grandes classes de travaux de modélisation : des travaux qui étudient l'aspect structurel des systèmes d'agents mobiles et d'autres qui s'intéressent à l'aspect comportemental. Une modélisation structurelle, doit mettre en évidence la structure générale du système, la structure de chacun de ses composants, ainsi que les différents types de liens entre eux. D'après l'étude que nous avons menée dans la littérature, nous avons remarqué qu'il existe peu de travaux qui s'intéressent à définir de manière explicite un modèle d'agent mobile. Tout de même, nous distinguons des travaux qui utilisent une notation semi-formelle et d'autres utilisent une notation formelle.

Dans [MG01], les auteurs ont fait naître un ADL (Architecture Description Language) qui permet, d'une part, de modéliser l'infrastructure qui peut supporter l'exécution des agents mobiles conformément au standard OMG-MASIF et, d'autre part, de définir les éléments nécessaires pour la représentation des mécanismes de distribution tels que : l'accès, la persistance, la migration des agents et leur localisation.

Cet ADL a été défini comme étant un profile UML nommé MASIF-DESIGN. Il spécifie les concepts de '*Region*', '*CoreAgency*' par des sous-systèmes stéréotypés. Le concept de '*AgentSystem*' est modélisé par un nœud stéréotypé. En fait, chaque '*AgentSystem*' est formé d'un '*CoreAgency*' et un ou plusieurs '*Places*'. Le '*AgentSystem*' agit comme un conteneur pour exécuter les agents en utilisant les fonctionnalités offertes par le *CoreAgency*. Une *Place* est un contexte, défini au sein d'un '*AgentSystem*', dans lequel s'exécute un agent. Il a été modélisée par un package. Les opérations qui offrent des services aux agents localisés dans une '*Place*' sont définies dans une interface appelée *SpecialPlaceInterface*. Enfin l'*Agent* est modélisé par un composant stéréotypé. Les opérations nécessaires pour l'exécution et la migration des agents sont définies dans une interface appelée *AgentsOperations*.

Dans ce travail, la modélisation se rapporte proprement à l'infrastructure exigée par la norme MASIF. Elle spécifie, de manière abstraite, les composants d'un système à base d'agents mobiles et définit, de manière claire, les relations entre eux. Cette modélisation représente un support d'aide considérable pour les concepteurs en se basant sur la notation UML. Néanmoins, elle ne permet pas d'établir des raisonnements et de vérifier des propriétés structurelles du système. En plus nous avons remarqué une large omission des concepts liés à la sécurité du système.

Le travail de [CPKL06], propose une spécification non-formelle d'une ontologie pour la modélisation des systèmes à base d'agents mobiles. Les concepts qui ont été identifiés comme fondamentaux pour la mobilité des agents sont : *role*, *agent*, *home*, *itinerary*, *visit*,

task, *location*, *migration constraint*, et *resource*. Un itinéraire spécifie l'ordre des visites qu'un agent doit effectuer afin d'accomplir ces objectifs. Ainsi, un itinéraire pourrait être associé à une tâche ou à un but. En plus un itinéraire est associé à des contraintes de migration qui déterminent l'ordre de visites et pourquoi et quand un itinéraire particulier est choisi.

Les relations entre ces concepts ont été représentés graphiquement selon le modèle d'Entité-Association. Ces liens sont caractérisés par des cardinalités et une étiquette.

La définition d'une ontologie pour la modélisation des systèmes à base d'agents mobiles est certainement un travail intéressant, dans le sens où elle identifie et spécifie de manière explicite les concepts fondamentaux pour la mobilité des agents. Néanmoins, nous notons que cette ontologie est dépourvue de tout concepts quant à la sécurité des agents mobiles et leurs systèmes d'accueil. En plus, les concepts définis dans cette ontologie ont été représentés de manière graphique sans permettre de raisonner sur des propriétés ayant trait à la structure du système.

Dans le travail de [HLG08], les auteurs se sont intéressés à étendre les diagrammes de classes de UML [RV02] et de AUML [BMO01] pour spécifier l'aspect statique et structurel des agents mobiles. Cette extension a porté sur la définition de trois nouveaux diagrammes :

- Diagramme d'environnement : spécifie les entités fondamentales dans une application à base d'agents mobiles selon la norme MASIF. Ces entités ont été modélisées par des stéréotypes de classes de UML/AUML ou des stéréotypes de paquetages. En plus, les relations entre ces entités ont été modélisées par différents stéréotypes de dépendances ou d'associations.
- Diagramme d'agent mobile : offre une représentation riche de la structure interne d'un agent mobile ainsi que ses propriétés. Cette représentation vient compléter la structure d'une classe d'agent AUML par de nouveaux stéréotypes d'attributs.
À ce niveau, nous avons souligné un effort modeste qui consiste à identifier les paramètres nécessaires pour la sécurité de l'agent mobile. Cependant, les paramètres identifiés ne sont pas suffisantes pour modéliser les différents aspects liés à la sécurité des agents mobiles. Ils s'intéressent uniquement à l'authentification des agents mobiles. En plus ces paramètres n'ont pas été modélisés de manière explicite.
- Diagramme d'itinéraire : représente une extension du diagramme de classe de UML afin de présenter la vue statique du modèle d'itinéraire d'un agent mobile.

Des contraintes supplémentaires sur les concepts évoqués dans ces diagrammes ont été exprimées en langage OCL (Object Constraint Language). Toutefois, les auteurs ne présentent aucune démarche de vérification des propriétés structurelles du système.

Dans l'objectif de pouvoir raisonner, spécifier et vérifier les propriétés d'un système à base d'agents mobiles, certains travaux ont tiré profit des avantages des notations mathématiques.

Le travail de G. Smith dans [Smi04] s'intéresse à définir un cadre pour la modélisation des systèmes mobiles selon la notation Object-Z. En particulier, plusieurs extensions ont été apportées à la spécification du modèle client-serveur pour l'appliquer au paradigme

d'agent mobile. L'agent mobile est alors modélisé par un simple objet comme c'est défini par Object-Z. Une telle spécification est dépourvue de tous les concepts nécessaires pour exprimer l'autonomie de l'agent. En plus, ce cadre s'intéresse uniquement à modéliser la mobilité de l'agent tout en négligeant les aspects liés à la sécurité.

1.4.2 Modélisation comportementale

La classe des travaux centrés sur la modélisation comportementale des agents mobiles met l'accent sur comment ils évoluent et interagissent dans un environnement distribué. Plusieurs travaux se sont intéressés à décrire le comportement des agents mobiles en utilisant les diagrammes d'UML. Par exemple, dans [KJ05], les auteurs utilisent les diagrammes de séquence d'UML pour modéliser la mobilité des agents en s'appuyant sur trois concepts clés à savoir : l'emplacement actuel de l'agent, le trajet de mobilité de l'agent et l'emplacement de création de l'agent. Quatre variantes de modélisation ont été proposées dans ce travail. Ces propositions sont marquées par l'introduction de nouveaux stéréotypes à savoir 'agent', 'at' et 'move'.

Plusieurs autres travaux se sont focalisés sur l'adaptation de la notation UML afin de supporter la modélisation comportementale des systèmes à base d'agents mobiles [SEM04, KWT04]. Récemment le travail de [LHG06] introduit par le nouveau langage proposé, MA-UML, des éléments de modélisation pour décrire l'aspect dynamique des agents mobiles. En effet MA-UML propose des extensions pour le diagramme d'activités de UML, le diagramme d'état-transition de UML et le diagramme de séquence de AUML. Le concept de 'localisation' a été introduit au niveau du diagramme d'activités afin d'établir le lien entre une tâche et l'emplacement de son exécution. Une autre contribution a porté sur l'extension du diagramme d'état-transition afin de supporter la modélisation du plan de voyage dynamique des agents. De même, le diagramme de séquence de AUML s'est étendu pour supporter la spécification des différentes interactions qui peuvent se présenter entre les agents mobiles, les agents stationnaires, les places (emplacement d'exécution des agents mobiles) et les ressources.

MA-UML montre certainement l'intérêt de l'utilisation d'une notation graphique qui est à la fois simple, intuitive et expressive. Néanmoins, cette notation semi-formelle souffre toujours de problèmes d'ambiguïtés, d'imprécisions et de manque de rigueur, ce qui rend difficile le raisonnement sur un certaines propriétés. En plus, MA-UML s'intéresse uniquement à modéliser les aspects liés à la mobilité de l'agent sans donné importance aux concepts liés à la sécurité.

Une deuxième classe de travaux s'intéresse à profiter de la puissance et de la richesse des notations mathématiques pour modéliser l'aspect comportemental des agents mobiles. Principalement, les méthodes destinées à l'aspect comportemental sont les algèbres de processus et les systèmes de transitions. Ces méthodes permettent, d'une part, d'avoir une meilleure compréhension des concepts et de lever toute ambiguïté quant à leur description, et de raisonner rigoureusement des propriétés comportementales. Ainsi, il y aura une

vision cohérente du système et une dissociation entre la spécification de haut niveau du système et les détails d'implémentation des agents mobiles.

Dans les méthodes de modèle de processus, un système est décrit en termes d'entités distinctes appelées de manière générique 'processus' où l'accent est mis sur les interactions entre les processus plutôt que sur leurs structures. Initialement, cette classe de méthodes a été particulièrement adaptée à la modélisation des systèmes concurrents, distribués ou encore à la description des protocoles. Puis, plusieurs travaux de recherche ont fait évoluer cette classe en introduisant des notions pour exprimer la mobilité. Dans cette classe, nous trouvons les algèbres de processus comme le π -calcul et les systèmes de transition tels que les réseaux de Petri de haut niveau.

Les algèbres de processus ont été largement utilisées par la communauté des agents mobiles. De ce fait, plusieurs calculs de processus focalisés sur la représentation de la mobilité et de la concurrence des processus ou des agents sont apparus. Nous citons le *join calcul*, le calcul des *ambients* [SZ98], le *SEAL calcul* [VC99], etc. Tous ces langages permettent de représenter des processus concurrents, qui peuvent communiquer et migrer dans un environnement distribué. D'une manière générale, la notion de localité (*Domain*) dans un calcul de processus implique directement que ce langage exprime la mobilité des processus, en plus de leur concurrence. Ces localités peuvent représenter soit des agents, soit des machines sur lesquelles les agents migrent et s'exécutent. Les localités peuvent être structurées de manière hiérarchique, sous forme d'un arbre. La mobilité d'un agent correspond alors à la migration d'un sous-arbre de localités.

Peu de travaux sur les agents mobiles s'intéressent à l'aspect sécurité. Nous citons à titre d'exemple les ambients qui représentent des domaines administratifs et contrôlent leurs accès. Le π -calcul d'ordre supérieur proposé dans [Tho89] s'intéresse aux problèmes de sécurité et plus particulièrement au contrôle d'accès par la spécification des ressources accessibles par chaque nouveau processus reçu. De même, dans *spi-calcul* [AG97], la sécurité joue un rôle très important. Ce calcul a été conçu pour la description et l'analyse des protocoles de cryptographie.

En plus des algèbres de processus, un autre courant de travaux fondés sur l'adoption des réseaux de Petri s'est imposé. En fait, plusieurs extensions ont été apportées aux réseaux de Petri classiques (place/transition) afin de pouvoir exprimer la mobilité. Par exemple, dans les réseaux de Petri mobiles [AB96], la mobilité est exprimée par l'utilisation des jetons (représentant les noms des places) et les jetons d'entrées d'une transition pour spécifier les destinations.

D'autres travaux adoptent les réseaux de Petri de haut niveau [XD00] afin de modéliser le comportement des agents dans un contexte mobile.

1.4.3 Synthèse

Les travaux formels focalisés sur la description des agents mobiles ne possèdent pas les mêmes visions certes, néanmoins, ils cherchent tous à pouvoir offrir aux développeurs la

possibilité de raisonner sur certaines propriétés. Cependant, nous avons décelé quelques limites.

En fait, la plupart de ces travaux sont orientés plutôt comportement et ceci par la modélisation des protocoles de communication et de migration des agents. En outre, les modélisations basées sur les algèbres de processus (e.g. [VC99] et [SZ98]) souffrent généralement d'un manque de couverture des concepts qui décrivent un système à base d'agents mobiles (e.g. les services, les ressources, etc). Il y a même des concepts différents (agent mobile et système d'agents) qui ont été modélisés par la même représentation. Nous avons remarqué, également, plusieurs de ces travaux limitent la représentation de l'agent mobile à un simple objet ou processus. Une telle représentation est dépourvue de tous les concepts nécessaires pour exprimer l'autonomie et l'aspect cognitif de l'agent (telles que les croyances, les connaissances et les compétences).

Nous avons aussi remarqué essentiellement que la majorité de ces travaux s'intéressent à modéliser la mobilité des agents sans toutefois accorder une importance aux problèmes de la sécurité. D'autres, se limitent à l'étude du problème de contrôle d'accès [Tho89] ou encore le problème de confidentialité [AG97].

Les travaux qui ont donné plus d'ampleur à l'aspect cognitif de l'agent, présentent une modélisation structurelle des systèmes à base d'agents mobiles en utilisant une notation semi-formelle qui ne permet pas de raisonner sur des propriétés du système. En plus, la plupart de ces travaux n'ont pas modélisé les concepts liés à la sécurité de l'agent et la sécurité du système.

1.5 Conclusion

Le but de ce chapitre était de dresser un état de l'art de la technologie des *agents mobiles*, tout en essayant de dégager les enjeux et les problèmes sous-jacents. Les synthèses faites sur les différents travaux étudiés, nous a permis de remarquer qu'il est nécessaire de faire recours aux techniques formelles afin de modéliser rigoureusement les systèmes à base d'agents mobiles. Ces techniques permettent de dissocier entre la spécification de haut niveau du système et les détails d'implémentation des agents mobiles.

En plus, cette modélisation doit tenir compte des différents aspects liés à la sécurité des systèmes à base d'agents mobiles. A cet effet, nous présentons dans le chapitre qui suit, un état de l'art des travaux qui se rapportent à la sécurité de tels systèmes afin de relever leurs apports et leurs limites, d'une part et dégager les concepts et les aspects fondamentaux, d'autre part.

2

Sécurité & Mobilité : État de l'art

La technologie des agents mobiles apporte des gains considérables pour plusieurs domaines d'application. Néanmoins, la mobilité des agents pose évidemment des problèmes de sécurité. En effet, lorsqu'un agent se déplace, il est crucial de s'assurer que l'agent sera exécuté correctement et en toute sécurité sur le nouveau système visité. De même, il est crucial de rassurer le système d'accueil qu'il n'y aura aucun risque d'héberger un nouvel agent.

Dans cet état de l'art, nous nous intéressons à deux courants de recherche : nous commençons par l'étude des travaux liés à la sécurité des systèmes distribués afin de montrer à quel niveau ils peuvent être appliqués aux problèmes spécifiques aux systèmes à base d'agents mobiles. Ensuite, nous étudions les travaux liés à la sécurité des systèmes à base d'agents mobiles pour exhiber leurs limites en regard de leurs avantages.

2.1 Politiques de sécurité

Depuis l'apparition des systèmes informatiques distribués, la sécurité des données manipulées est de plus en plus un domaine au cœur des préoccupations.

Pour assurer les besoins de sécurité d'un système informatique, deux tendances de recherches peuvent être distinguées dans la littérature. La première cherche à proposer des techniques pour détecter et contrecarrer les attaques. Il existe deux classes de techniques de protection [AB04, JK99] : les techniques de prévention et les techniques de détection. La deuxième tendance, s'intéresse à exprimer les besoins de sécurité par la spécification des règlements qui régissent la façon de protéger les informations et les ressources au sein d'un système informatique. L'ensemble de ces règlements constitue la politique de sécurité du système.

Les travaux focalisés sur la définition de techniques de protection restent au delà des attentes des utilisateurs, notamment dans le cadre des systèmes à base d'agents mobiles. En effet, ces systèmes nécessitent une structure de sécurité flexible qui s'adapte aux changements dynamiques des besoins de sécurité des agents mobiles et leurs systèmes d'exécution.

Notre but, dans cette section, est de définir le concept de politique de sécurité et de présenter ses principaux fondements. Dans une deuxième étape, nous apportons un intérêt sur les efforts de modélisation des politiques de sécurité et nous menons une discussion sur les différentes approches de spécification de ces politiques, ainsi que les mécanismes de leur imposition.

2.1.1 Définitions et Fondements

Dans la littérature, différentes définitions ont été proposées pour les politiques de sécurité. Toutefois, ces définitions partagent le même objectif qui consiste à éliminer les risques d'attaque d'un système et satisfaire ses besoins de sécurité.

Selon F.B. Schneider [Sch00], une politique de sécurité définit l'exécution (action) qui, pour une raison ou une autre, est considérée inacceptable. Une telle définition peut concerner différents types de politiques tels que : le contrôle d'accès, le contrôle de flux d'information ou la disponibilité.

Pour A.S. Oliveira [Oli08] la politique de sécurité, d'un système donné, permet d'évaluer les données dans son état actuel de façon à indiquer quelles sont les transitions possibles, à partir de cet état, qui mènent à d'autres états qui peuvent être considérés comme sûrs.

Quant à A. Abou Kalem [Kal03], il définit une politique de sécurité comme un dispositif nécessaire pour renforcer la sécurité des systèmes qui implique d'empêcher la réalisation d'opérations illégitimes contribuant à mettre en défaut les propriétés de confidentialité, d'intégrité et de disponibilité, mais aussi de garantir la possibilité de réaliser les opérations

légitimes dans le système.

A. Contes définit, dans [Con05] la notion de politique de sécurité par un ensemble de propriétés de sécurité qui doivent être satisfaites par le système et un schéma d'autorisations qui représente les règles permettant de modifier l'état de protection d'un système.

Le standard ITSEC (Information Technology Security Evaluation Criteria) [CdCE91], définit une politique de sécurité par l'ensemble des lois, règles et pratiques qui régissent la façon dont l'information sensible et les autres ressources sont gérées, protégées et distribuées à l'intérieur d'un système spécifique.

Sous cette variété de définitions se cache une large complexité liée à la manière d'exprimer la politique de sécurité afin d'être suffisamment compréhensible et explicite et de permettre la représentation des différents besoins de sécurité d'un système. En plus, il est essentiel de vérifier à la spécification d'une politique un certain nombre de propriétés (e.g. terminaison, complétude, cohérence, etc) afin d'assurer convenablement son application. De même, il est fondamental de vérifier la concordance d'une politique avec les besoins de sécurité requis.

Par ailleurs, il faut mettre en avant le besoin de flexibilité de la politique afin de répondre aux changements dynamiques des exigences de sécurité d'une application, au cours de son exécution. De ce fait, une délicatesse émerge aussi au niveau de l'imposition de la politique et sa maintenance.

2.1.2 Modélisation de la sécurité

Le but d'un modèle de sécurité consiste à définir un ensemble de concepts afin de structurer l'expression des politiques de sécurité.

Plusieurs modèles de sécurité ont été proposés dans la littérature. L'étude que nous avons menée, nous a permis de distinguer principalement trois classes de modèles de sécurité [CCB06] : les modèles de contrôle d'accès, les modèles de contrôle de flux et les modèles de contrôle d'usage. Nous présentons, dans ce qui suit, chacune de ces classes.

2.1.2.1 Modèles de contrôle d'accès

Plusieurs modèles de contrôle d'accès ont été proposés, tels que RBAC [FSG⁺01], TBAC [TS97], OrBAC [KBB⁺03]. Chacun de ces modèles propose de structurer les politiques de sécurité autour d'un ensemble de concepts. Par exemple, le modèle RBAC repose principalement sur le concept de *rôle* et OrBAC repose sur le concept d'*organisation*.

Initialement, les modèles de contrôle d'accès expriment uniquement les autorisations. Pour mieux faciliter l'expression des règles de contrôles d'accès, de nouveaux modèles offrent la possibilité d'exprimer les interdictions. Ces modèles posent le problème de conflit entre les autorisations et les interdictions. En effet, plusieurs stratégies de

résolution de conflits ont été proposées dans la littérature.

D'autres modèles offrent l'expression des contextes d'application des règles de sécurité pour permettre l'activation dynamique des autorisations. Dans cette perspective, plusieurs types de contexte peuvent être identifiés, tel que le contexte temporel [Dam02].

2.1.2.2 Modèles de contrôle de flux

Les modèles de contrôle d'accès ne sont plus suffisants au regard des différents besoins de sécurité. En fait, ces modèles ne permettent pas de contrôler les flux d'information entre les objets du système.

Bell-LaPadula [BP76] représente le premier modèle de contrôle de flux. Principalement, ce modèle définit différents niveaux de sécurité, pour mieux gérer l'accès au système. Ces niveaux de sécurité seront affectés entre les différents sujets (entités actives qui peuvent exécuter des opérations) et les différents objets (entités passives de sauvegarde) du système. En effet, l'état d'un système est défini par un quadruplet formé de : l'état courant (mode d'accès de l'objet), les permissions d'accès (Matrice associant les objets aux sujets en indiquant les droits qui s'y appliquent), le niveau de sécurité de l'objet (Top Secret, Secret, Confidential, Unclassified) et la hiérarchie de l'objet. Les propriétés de sécurité seront exprimées en fonction des valeurs de ces attributs.

Le modèle Bell-LaPadula constitue la base de tous les autres modèles de contrôle de flux. Parmi ses dérivés, le modèle Biba [Bib77] qui utilise une modélisation similaire afin d'assurer l'intégrité des données.

2.1.2.3 Modèles de contrôle d'usage

Les nouvelles applications de nos jours tels que les DRM (Digital Right Management), exigent la satisfaction d'un ensemble de conditions avant et pendant l'exécution d'une action. Ces exigences sont au-delà des modèles de contrôle d'accès qui vérifient la satisfaction des conditions, uniquement, avant d'autoriser l'exécution d'une action.

Partant de ce fait, une nouvelle tendance de travaux apparaît pour le contrôle d'usage des ressources [SC05]. En effet, cette classe de modèles de sécurité exige une réévaluation continue des conditions d'usage, au fur et à mesure de l'évolution du système, jusqu'à la fin de l'exécution de l'action.

J.Park et R. Sandhu proposent dans [PS04] une famille de modèles $UCON_{ABC}$ pour le contrôle d'usage ($UCON$) qui intègre des Autorisations (A), obligations (B), et Conditions (C). Les autorisations évaluent les attributs du sujet (demandeur d'accès), les attributs de l'objet (ressource/service demandé) et le type du droit demandé pour une décision d'usage (exécution d'action).

Les autorisations peuvent être soit des pré-autorisations (preA) ou des autorisations en

cours (onA). Le PreA s'exécute avant de bénéficier du droit d'accès tandis que onA s'exécute (continuellement ou périodiquement) au moment de l'accès.

Les obligations vérifient si le sujet respecte des exigences d'exécution avant ou pendant un exercice d'usage. De même, les obligations peuvent être des pré-obligations (preB) ou des obligations en cours (onB).

Pour les conditions, elles représentent des facteurs de décision orientés environnement ou système. Ces conditions sont complètement différentes des autorisations du fait qu'elles évaluent l'état de l'environnement ou l'état du système sans avoir aucune relation avec les attributs du sujet et de l'objet.

2.1.3 Approches de spécification des politiques de sécurité

Dans cette section, nous avons porté un intérêt particulier aux travaux qui tournent autour de la spécification des politiques de sécurité dans le cadre des systèmes distribués.

Ces travaux peuvent être structurés en trois grandes classes d'approches [Dam02] : spécification à travers un langage métier ou un langage dédié (Domain-Specific Language : DSL), spécification à base de règles (rule-based specification) et spécification à base de logique (logic-based specification).

2.1.3.1 Spécification à travers un langage métier

Les langages métiers permettent la représentation explicite des concepts relatifs à un domaine d'application particulier. Dans le cadre des politiques de sécurité, plusieurs travaux de spécification ont proposé des langages métiers mettant en relief les concepts fondamentaux de la sécurité des systèmes distribués, en utilisant des notations très variées. Nous présentons, dans ce qui suit, quelques travaux des plus célèbres langages métiers.

Ponder

Ponder [DDLS01] est un langage déclaratif et abstrait, dédié pour la spécification des politiques de sécurité et de gestion dans le cadre des systèmes distribués. Il supporte la spécification de plusieurs types de politique à savoir les politiques de : autorisation, filtrage d'information, restriction, délégation, et obligation. Chacune de ces politiques est décrite par un ensemble d'attributs qui varie selon le type de la politique.

De façon commune, toute politique s'applique sur des sujets (subjects) pour contrôler leur comportement à l'intention d'exécuter une action (Action) sur un objet (Target) donné. Les objets du système ainsi que les sujets peuvent être regroupés en domaines. Ce qui facilite la spécification des systèmes à grande échelle, possédant des milliers d'objets.

Toute politique est valide sous certaines conditions. En effet, différents types de contraintes ont été identifiés : principalement nous distinguons des contraintes qui évaluent les valeurs des attributs de la politique (e.g. l'état du sujet/target) et d'autres contraintes temporelles qui spécifient la période de validité de la politique. Ces contraintes ont été exprimées par le langage OCL (Object Constraint Language).

En plus le langage Ponder inclut la définition de méta-politiques, pour gérer les conflits susceptibles de se présenter entre politiques.

Une méta-politique définit pour un ensemble de politiques les contraintes qui refusent l'exécution simultanée de politiques conflictuelles. De même une méta-politique est exprimée par le langage OCL.

SPL (Security Policy language)

SPL [RZFG01] est un langage qui exprime les concepts de permission, d'interdiction et d'obligation. Il est composé de quatre blocs : entités, ensembles, règles et politiques. Les entités sont de deux types : interne ou externe au service de sécurité. Les entités manipulées par le SPL (e.g. ensemble ou politique) sont internes au SPL par contre les utilisateurs et les fichiers représentent des entités externes.

Les entités peuvent être classifiées dans des ensembles afin de rendre la politique plus compacte et éliminer la répétition d'une même règle qui s'applique pour différentes entités.

Une règle exprime des contraintes définies entre les entités et les ensembles. La politique de sécurité est composée d'un ensemble de règles.

Étant donné que SPL supporte des politiques mixtes (positive et négative), différentes priorités peuvent être assignées aux règles d'une politique afin de résoudre les formes possibles de conflits.

SPL est un langage basé sur les événements dont le but de chaque règle est de décider si l'événement est permis ou refusé.

Le langage SPL peut exprimer trois différents types de contraintes : contraintes à base d'historique (history based constraint), contraintes d'obligation et des contraintes in-variantes. Les politiques à base d'historique s'expriment par une dépendance avec les événements passés (PastEvents), alors que les politiques d'obligation s'expriment par une dépendance avec des événements futures (FutureEvents).

2.1.3.2 Spécification à base de règles

La spécification des politiques à base de règles peut être exprimée de différentes manières. Principalement, nous avons distingué les règles réactives et les règles productives. Une règle réactive appelée aussi règle ECA (Event-Condition-Action) réagit en réponse à un événement perçu (E) et détermine en fonction de l'évaluation d'une condition (C), l'action

(A) adéquate à exécuter.

Alors qu'une règle productive est de la forme "Si C alors A" (if C then A), où C est une condition qui doit être vérifiée pour pouvoir exécuter l'action A.

La spécification des politiques à base de règles présente plusieurs avantages [ABB⁺07]. D'une part, les règles possèdent une sémantique précise et relativement simple. D'autre part, la spécification à base de règles permet d'intégrer de manière cohérente différents types de politique de sécurité dans une seule spécification. En plus, il existe une forte tendance à formuler les politiques de sécurité sous forme de règles.

PDL (Policy Description Language)

PDL [LBN99] est un exemple de langage de description de politique à base de règles. La définition de la politique avec PDL se fait en deux étapes. En premier lieu, le gestionnaire de la politique doit identifier : 1) l'ensemble des événements que le système peut contrôler, 2) l'ensemble des actions qui peuvent être invoquées par la politique et 3) l'ensemble des fonctions qui évaluent l'état de l'environnement. Ensuite, le gestionnaire combine ces trois ensembles pour définir la politique du système.

Les événements sont divisés en deux ensembles : des événements pré-définis par le système (system defined event) et d'autres définis par la politique (policy defined event).

En fait, les politiques selon PDL sont décrites par une collection de propositions (axiomes) de deux types différents : des règles de la politique et des événements définis par la politique. Les règles de la politique sont de la forme ECA :

event causes action if condition

Alors que les événements définis par la politique sont de la forme suivante :

event triggers pde(ml = tl, ... , mk = tk) if condition

où *event* est un événement déclencheur d'un autre événement *pde* (policy defined event) sous contrainte que la *condition* soit satisfaite.

Le langage PDL a été étendu par la définition d'un cadre formel pour la détection et la résolution de conflits [CLN00]. Une politique génère un conflit lorsque il existe en sortie deux actions qui ne doivent pas s'exercer en même temps. Pour éviter ces conflits, des contraintes d'action doivent être spécifiées par l'administrateur de la politique. Une contrainte d'action est définie de la forme :

never $a_1 \cdots a_n$ if C (formellement : $\forall \neg(a_1 \wedge \cdots \wedge a_n \wedge C)$)

2.1.3.3 Spécification à base de logique

Avec la complexité des systèmes d'information de nos jours et la variété de leurs besoins de sécurité, il est nécessaire d'exprimer les politiques de sécurité dans un langage muni d'une syntaxe claire et d'une sémantique précise afin d'éliminer toutes formes d'ambiguïté et vérifier des propriétés fondamentales pour une mise en œuvre correcte.

De ce fait, les formalismes logiques (e.g. logique de premier ordre, logique déontique et logique modale) ont prouvé leur adéquation pour spécifier les politiques de sécurité du fait qu'ils disposent d'une sémantique claire, non ambiguë et expressive, permettant de raisonner rigoureusement en terme de propriétés. En plus, ils permettent la spécification des politiques à un niveau d'abstraction élevé qui offre plus de simplicité et de clarté.

ASL (Authorization Specification Language)

Le langage ASL [SJ97] est un langage logique pour la spécification des politiques de contrôle d'accès. En effet, plusieurs constantes, variables et prédicats ont été définis pour exprimer différents concepts (e.g. objects, actions, types, users, groups, roles, etc.) ainsi que les relations entre eux (*cando*, *decando*, *do*, *grant*, *done*, *active*, etc.).

Par exemple '*cando(o,s,a)*', est un prédicat ternaire qui spécifie les autorisations (*a*) assignées à un utilisateur/groupe/role (*s*) au sujet de l'utilisation de l'objet (*o*). Ces autorisations peuvent être positives ou négatives. Le prédicat '*decando(o, s, a)*', est aussi un prédicat ternaire qui représente des autorisations dérivées par le système en utilisant des règles logiques d'inférence.

En effet, le langage ASL est composé de plusieurs types de règle, tels que les règles de dérivation, qui permettent la dérivation implicite des autorisations de celles explicitement spécifiées par le prédicat '*cando*'. Par exemple, la règle suivante :

$$decando(file1, s, +read) \leftarrow cando(file1, s', +read) \ \& \ in(s, s')$$

dérive une autorisation positive pour le sujet '*s*' de lire le fichier '*file1*' s'il existe un groupe '*s'*' auquel il appartient '*s*' et ce groupe '*s'*' bénéficie d'une autorisation positive de lire le fichier '*file1*'.

Les règles de résolution consistent à déterminer comment les conflits entre les autorisations doivent être résolus. Par exemple :

$$do(file2, s, +a) \leftarrow \neg decando(file2, s, -a)$$

Cette règle stipule que le sujet '*s*' peut accéder au fichier '*file2*' s'il n'a pas une autorisation négative quant au même objet '*file2*'.

Les règles de contrôle d'accès contrôlent les décisions de la politique ; et les règles d'intégrité seront utilisées pour exprimer différents types de contraintes relatives à la spécification et l'utilisation des autorisations.

2.1.3.4 Synthèse

Bien que les approches focalisées sur la définition d'un langage métier offrent une définition explicite aux concepts liés à la sécurité, il est nécessaire d'utiliser un formalisme logique, muni d'une syntaxe claire et d'une sémantique précise, afin de pouvoir raisonner sur des propriétés liées à la cohérence de la politique et sa concordance avec les besoins de sécurité requis.

En plus, les approches à base de logique abordent le problème à un niveau d'abstraction élevé, ce qui permet de mieux appréhender la complexité liée à la sécurité des systèmes distribués. Cependant, ces approches exigent une bonne connaissance autour des fondements mathématiques et logiques.

Quant aux approches à base de règles, elles montrent une simplicité d'expression. Cependant, les politiques sont généralement spécifiées à un niveau d'abstraction moins élevé. En plus, l'expression des politiques par un ensemble de règles ne constitue pas une base solide sur laquelle on peut raisonner rigoureusement et établir des preuves. Il est alors indispensable d'appliquer, en plus, des fondements mathématiques.

2.1.4 Mécanismes d'imposition des politiques de sécurité

Bien évidemment, après la spécification d'une politique de sécurité il faut se préoccuper de sa mise en œuvre et d'en choisir le mécanisme d'imposition adéquat.

Nous avons noté d'après la littérature [Tal07], l'existence de trois classes de mécanismes d'imposition des politiques de sécurité à savoir : l'analyse statique, le monitoring de l'exécution, et la réécriture de programme. Chacun de ces mécanismes prouve ses avantages selon le type de l'application.

– **Analyse Statique** : Les analyses statiques [CM04] consistent à déterminer le comportement possible d'un programme avant de commencer son exécution. En effet, de tels mécanismes opèrent strictement dans un temps fini avant de commencer l'exécution du programme non digne de confiance. Après analyse, les programmes (sous-programmes) qui peuvent générer un comportement inacceptable seront rejetés et ils ne seront jamais exécutés.

Parmi les exemples utilisant ce mécanisme nous citons : Static type-checkers de la machine virtuelle de Java [LY99], le langage JFlow [Mye99] et les scanners de virus [Nac97].

– **Monitoring de l'exécution** : Le mécanisme de monitoring de l'exécution [TTD06] contrôle le comportement du programme au moment de son exécution et intervient lorsque une imminente violation de la politique est détectée. Cette intervention peut, simplement, interrompre l'exécution de la méthode/action, qui a violé la politique. La forme d'intervention la plus puissante consiste à insérer des actions ou omettre

de dangereuses actions au sujet du programme contrôlé. Cette forme d'intervention correspond au mécanisme de monitoring de l'exécution basé sur la réécriture.

- **Réécriture du programme :** La Réécriture du programme [Ham06] est une alternative entre l'analyse statique et le monitoring de l'exécution. Comme son nom l'indique, le mécanisme de réécriture de programme consiste à modifier, dans un temps fini, le code des programmes non digne de confiance avant de commencer leurs exécution. Bien évidemment, l'exécution du programme qui résulte de la réécriture du programme doit être conforme à l'exécution originale du programme.

Pour aborder convenablement le problème de sécurité dans les systèmes à base d'agents mobiles, il est nécessaire de déployer un mécanisme qui respecte l'aspect dynamique des exigences de sécurité d'une application, au cours de son exécution. En plus, ce mécanisme doit maintenir un haut niveau de disponibilité des applications pour pouvoir modifier la politique de sécurité sans arrêter l'exécution de l'application.

De ce fait, le mécanisme d'analyse statique et celui de la réécriture de programme, ne représentent pas le mécanisme approprié pour l'imposition des politiques de sécurité dans les systèmes à base d'agents mobiles du fait qu'ils opèrent avant l'exécution du programme et par conséquent ils ne s'alignent pas avec l'aspect dynamique de la sécurité.

Contrairement, le mécanisme de monitoring de l'exécution basé sur la réécriture montre bien son adéquation, du fait qu'il contrôle le comportement du programme au moment de son exécution. En plus, à la détection d'une violation, ce mécanisme peut intervenir par la modification du code de l'application, tout en maintenant sa disponibilité.

2.2 Sécurité des systèmes à base d'agents mobiles

Dans cette section, nous passons à introduire les besoins de sécurité des systèmes à base d'agents mobiles. Par la suite, nous proposons, dans ce cadre, une classification des travaux focalisés sur la définition d'un cadre de sécurité pour les systèmes à base d'agents mobiles, toute en soulignant leurs limites en regard de leurs avantages.

2.2.1 Besoins de sécurité

Les attaques qui se produisent dans un système à base d'agents mobiles peuvent avoir différentes sources et différentes cibles. Effectivement, il existe quatre classes d'attaques [Jan00, BR05] : deux classes dont la source est un agent mobile malveillant et les deux autres sont provoquées par un système malhonnête.

En effet, un agent malveillant peut attaquer les systèmes visités comme il peut attaquer d'autres agents mobiles avec lesquelles il réside dans le même système.

De l'autre côté, un système malveillant peut attaquer les agents mobiles visiteurs comme

il peut attaquer d'autres systèmes d'agents.

Une description détaillée de ces différentes classes sera présentée dans la Section 5.1.

Pour contrecarrer les différentes formes d'attaques dans un système informatique, il est nécessaire de mettre en œuvre des mécanismes permettant de garantir les besoins de sécurité telles que l'*authenticité*, la *confidentialité*, l'*intégrité* et la *disponibilité*.

Dans ce qui suit, nous allons développer ces besoins de sécurité dans le cadre des systèmes à base d'agents mobiles [BR05].

- *Authenticité* : l'authentification assure la reconnaissance sûre de l'identité. Elle consiste à prouver que l'identité déclarée par une entité est bien la sienne. En fait, un agent mobile doit s'authentifier sur chaque système d'agents visité et par conséquent, un système d'agents est alors capable de décider s'il s'agit d'un agent de confiance. Dans le même ordre d'idée, l'agent mobile doit être capable d'authentifier le système d'agents.
- *Confidentialité* : la confidentialité assure qu'une information ne peut être lue que par les entités autorisées. La confidentialité couvre les données stockées, les paramètres de traitement, et toute information transmise. L'agent mobile, ainsi que le système d'agents doivent protéger leurs informations privées contre les accès non autorisés.
- *Intégrité* : il est essentiel de vérifier l'intégrité des données par le fait de s'assurer qu'une donnée n'a pas été modifiée par une entité non autorisée. Par exemple, dans le cadre des agents mobiles, l'itinéraire de l'agent est une donnée qui demande une protection contre toutes formes d'altération.
- *Disponibilité* : la disponibilité assure d'une part l'utilisation convenable et non abusive des services et/ou des ressources du système. D'autre part, cette propriété assure l'accessibilité aux ressources et/ou services tant qu'il s'agit d'un agent autorisé.

Pour assurer ces besoins de sécurité, plusieurs travaux de la littérature tournent autour de la spécification des politiques de sécurité dans le contexte de mobilité.

En fait, l'étude que nous avons menée sur ces travaux, nous a permis de distinguer deux tendances de recherche : la première tendance tire profit des travaux focalisés sur la spécification des politiques de sécurité pour les systèmes distribués en vue de les adapter à une infrastructure à base d'agents mobiles. La deuxième tendance, se focalise sur la définition d'un cadre spécifique pour la sécurité des systèmes à base d'agents mobiles.

2.2.2 Solutions de sécurité adaptées

Nous synthétisons dans ce qui suit quelques travaux, de référence, qui proposent des solutions adaptées pour la sécurité des systèmes à base d'agents mobiles.

Le premier exemple est celui de [DRF03], qui a utilisé le langage SPL [RZFG01] pour définir des politiques d'autorisation qui contrôlent les actions des agents mobiles. L'originalité de ce travail, réside dans le contrôle de l'agent en fonction de l'historique de son comportement. Néanmoins, nous avons noté plusieurs limites pour ce travail. D'une part, cette solution de sécurité contrôle uniquement le comportement des agents mobiles afin de mieux protéger les systèmes d'exécution des agents (host) contre les agents malveillants et de protéger les agents mobiles contre les attaques d'autres agents mobiles. De ce fait, aucune importance n'a été accordée à la protection des agents mobiles contre les hôtes malveillants et la protection des hôtes contre les autres systèmes malveillants.

En plus, la représentation de l'agent mobile était restreinte à un objet décrit par un nom, un propriétaire, un hôte de création, un temps de création et un ensemble des hôtes visités. De même, le concept 'host' a été défini par un simple attribut de type 'string' dans la description de l'agent mobile.

Par ailleurs, ce travail s'intéresse à la description des politiques d'un point de vue statique, sans prendre en considération l'aspect dynamique des besoins de sécurité dans les systèmes à base d'agents mobiles.

Un autre exemple est celui de [MSD01], où les auteurs exploitent le langage Ponder [DDLS01] pour exprimer des politiques de sécurité qui contrôlent le comportement des agents mobiles.

De même, cette solution de sécurité ne traite pas les différentes préoccupations de sécurité dans les systèmes à base d'agents mobiles. Elle contrôle uniquement le comportement des agents mobiles sans contrôler les systèmes d'agents et comment ils doivent se comporter avec un agent visiteur. En fait, chacun des agents mobiles et des systèmes d'agents doit avoir séparément une politique de sécurité qui contrôle son comportement avec les autres entités du système et contrôle l'accès de ses ressources et/ou de ses services.

En plus, nous remarquons que ce travail n'exhibe pas un détail assez complet des concepts qui émergent de la combinaison de la mobilité avec la sécurité. En fait, suite à une migration l'agent doit s'adapter au comportement et aux exigences de sécurité du nouveau système visité. Cette adaptation peut agir sur la définition de la politique de l'agent mobile. Cependant, le changement dynamique de la politique de l'agent et son adaptation au nouveau système visité n'a pas été pris en considération dans ce travail.

2.2.3 Solutions de sécurité spécifiques

La deuxième classe de travaux s'intéresse plutôt à définir un cadre de sécurité spécifique pour la protection des systèmes à base d'agents mobiles.

Le travail de S. Ugurlu et al. [UE06], offre un haut niveau de flexibilité pour la spécification des politiques de sécurité. Nous avons souligné pour ce travail, une large préoccupation aux concepts indispensables au développement d'un système d'agents mobiles sécurisé. En effet, deux types de politiques ont été supportés : politique du hôte (Host Policy) et politique de l'agent mobile (Agent Policy).

La politique d'un hôte protège les ressources locales d'un hôte contre les actions non autorisées. Quant à la politique de l'agent, elle détermine l'aptitude de l'agent de réaliser des demandes (actions) sur des hôtes distants. En plus, ce type de politique sert à protéger l'agent contre les hôtes malveillants ou d'autres agents malicieux.

Chacune de ces politiques est définie par un nombre de règles de type ECA. Les règles de la politique d'un hôte tiennent compte de deux critères : l'emplacement de création de l'agent (agent owner) et l'emplacement duquel il provient (agent source).

La plateforme SECMAP (Secure Mobile Agent Platform), qui implémente ce cadre de sécurité, permet de manipuler dynamiquement le contenu des politiques des agents. Il s'agit d'une interface graphique qui permet à l'agent créateur de contrôler ses agents et de modifier manuellement leur politiques afin qu'ils s'adaptent aux nouveaux besoins de sécurité du système.

En dépit de cette richesse de concepts relative à la sécurité des systèmes à base d'agents mobiles, nous avons décelé quelques limites. D'une part, la définition de ce cadre de sécurité ne se repose pas sur des fondements mathématiques afin d'apporter une représentation formelle des politiques de sécurité et établir sur sa base des raisonnements rigoureux pour vérifier la consistance de la politique et sa concordance avec les besoins de sécurité du système. En plus, avec ces politiques il est possible d'exprimer uniquement des règles de contrôle d'accès sans toutefois exprimer des règles d'obligation pour le contrôle d'usage. D'autre part, l'aspect dynamique des politiques de sécurité a été considéré seulement à un niveau d'implémentation. Par la suite, il devient difficile et même impossible de vérifier des propriétés sur la nouvelle politique obtenue.

C. Bryce dans ses travaux [Bry06] propose un cadre de sécurité pour les systèmes à base d'agents mobiles qui supporte la spécification des politiques de sécurité pour l'agent mobile et pour le système d'exécution (host/place). La spécification de ce cadre a été précédée par la définition d'un cadre conceptuel qui exhibe les principaux concepts liés à la structure du système, et ceux liés aux différentes étapes du cycle de vie de l'agent (i.e. sa création, sa communication, sa migration, jusqu'à sa terminaison).

Sur la base de ces concepts, la politique de sécurité d'un agent (agent mobile/hôte) peut être exprimée.

Une politique de sécurité contrôle, uniquement, les droits d'accès d'un agent. En fait, les

droits sont rassemblés dans des groupes et l'agent doit être associé à un groupe afin de bénéficier des droits de ce groupe.

L'agent peut avoir des niveaux de confiance différents à chaque hôte. Par conséquent il doit être adaptable à l'environnement auquel il s'exécute et répondre au niveau de confiance du hôte visité. Cette adaptation est exprimée par le changement du groupe de l'agent.

D'une part, ce travail ne couvre pas les différentes préoccupations de la sécurité des systèmes à base d'agents mobiles. Il est limité au contrôle d'accès. La politique est constituée d'un ensemble de règles de la forme événement-action. En plus, nous avons noté l'absence d'une description claire et précise des différents types d'événements déclencheur d'une règle et par analogie les actions qui peuvent être remplies.

D'autre part, l'adaptabilité de l'agent qui est exprimée par un changement de groupe doit tenir compte du niveau de confiance accordé par le hôte visité. Aucune relation n'a été définie entre le niveau de confiance de l'agent et les droits correspondants.

En plus, suite à un changement de groupe et en fonction du niveau de confiance accordé, l'agent doit vérifier s'il est capable de poursuivre son exécution sur le nouveau hôte. Pour pouvoir vérifier ce type de propriété, il est nécessaire d'appliquer des fondements logiques dans la spécification de la politique, chose qui était complètement absente dans ce travail.

2.3 Discussion : Une nouvelle perspective de sécurité

Les travaux effectués sur la spécification des politiques de sécurité dans le cadre des systèmes à base d'agents mobiles sont certainement intéressants, cependant nous avons dégagé des limites.

En fait, les travaux qui proposent des solutions de sécurité adaptées sont marqués par un manque de couverture des concepts liés à la définition d'un système à base d'agents mobiles tels que le concept *hôte*, *système d'agents*, *service*, etc. Le concept *agent mobile* est généralement modélisé par un simple objet sans toutefois le définir par rapport à une vue globale du système.

En plus, ces travaux ne s'intéressent pas aux différentes préoccupations de la sécurité dans un système à base d'agents mobiles. En effet, la majorité des travaux s'intéressent à spécifier des politiques de sécurité pour contrôler le comportement des agents mobiles et leurs accès aux ressources.

Quant aux travaux qui proposent des solutions de sécurité spécifiques pour les systèmes à base d'agents mobiles, ils donnent plus d'importance aux concepts liés à la mobilité. Néanmoins, les spécifications proposées ne reposent pas sur des techniques formelles.

L'aspect dynamique qui caractérise les systèmes à base d'agents mobiles et l'émergence continue de nouvelles menaces de sécurité reflètent l'évolution dynamique des besoins de sécurité de tels systèmes. Cependant, nous avons remarqué que la plupart des travaux s'intéressent à la spécification des politiques à un niveau statique. Ainsi, l'aspect dyna-

mique est peu considéré.

Le travail effectué par [UE06] permet de manipuler dynamiquement le contenu des politiques des agents, mais ceci étant défini à un niveau d'implémentation. Nous notons ainsi l'absence de travaux qui spécifient rigoureusement les opérations de reconfiguration des politiques de sécurité afin de pouvoir raisonner sur la consistance de la politique.

Par conséquent, il est crucial de définir une structure de reconfiguration qui adapte les politiques aux exigences dynamiques du système.

Nous pouvons expliquer ces limites par la double complexité liée, d'une part, à la richesse et la variété des concepts d'expression des politiques de sécurité et, d'autre part, à la richesse des concepts qui décrivent un système à base d'agents mobiles.

En conclusion, nous notons que le problème de sécurité dans les systèmes à base d'agents mobiles reste encore ouvert et dépasse largement la problématique de mobilité des agents.

2.4 Conclusion

Le but de ce chapitre était de dresser un état de l'art des travaux effectués sur la spécification des politiques de sécurité dans le cadre des systèmes distribués et dans le cadre des systèmes à base d'agents mobiles.

Les synthèses faites sur les différents travaux étudiés, nous a permis de tirer les conclusions suivantes : 1) La variété des préoccupations de sécurité d'une manière générale et particulièrement dans les systèmes à base d'agents mobiles, demande un appui sur des techniques formelles, qui offrent suffisamment de flexibilité et d'expressivité, afin de spécifier rigoureusement les politiques de sécurité aussi bien au niveau statique qu'au niveau dynamique. 2) Après avoir spécifié la politique de sécurité et vérifié certaines propriétés attendues, il est nécessaire de déployer un mécanisme qui respecte l'aspect dynamique des exigences de sécurité des systèmes à base d'agents mobiles. Le mécanisme de monitoring de l'exécution basé sur la réécriture a bien montré son adéquation.

Ainsi, nous proposons dans ce travail de thèse, d'exploiter la puissance des techniques formelles afin de définir un cadre logique qui modélise rigoureusement les systèmes à base d'agents mobiles et nous permet de raisonner sur leurs propriétés de sécurité. Ce cadre doit tenir compte de l'évolution dynamique des besoins de sécurité de tels systèmes et le besoin d'adaptabilité de l'agent suite à une migration.

Notamment le choix du langage de spécification dépend du niveau d'abstraction à aborder. Pour mieux maîtriser la complexité des systèmes d'une façon générale, il est recommandé d'adopter un niveau d'abstraction élevé qui écarte tout détail inutile vis-à-vis de l'ensemble des propriétés attendues. De ce fait, nous choisissons la notation Z comme langage de spécification orientée modèle qui permet de modéliser le système aussi bien au niveau statique qu'au niveau dynamique. Nous utilisons l'outil de preuve automatique Z/EVES pour analyser les spécifications et vérifier les propriétés du système.

3

Spécification Formelle des Systèmes à base d'Agents Mobiles

La diversité et la complexité des concepts de base qui caractérisent les systèmes à base d'agents mobiles impliquent une difficulté à comprendre et à concevoir de tels systèmes. La définition d'un modèle conceptuel s'avère alors nécessaire pour maîtriser cette complexité et proposer un consensus au sujet de leurs concepts fondamentaux.

La spécification du système en langage naturel peut donner lieu à différentes interprétations. Les spécifications formelles ont remédié à ce problème par l'utilisation d'un langage formel basé sur une syntaxe et une sémantique précisent.

Nous proposons dans ce qui suit une modélisation qui spécifie rigoureusement les concepts liés à la structure des systèmes à base d'agents, unifie leurs représentations et définit les relations entre eux. Nous adoptons, pour ce faire, la notation Z comme langage de spécification.

Z est une notation standard assez expressive qui offre une approche rigoureuse pour la structuration et la composition des spécifications. Elle est soutenue par plusieurs outils de preuve, tels que Z/EVES [Saa97, MS97], qui permettent une vérification syntaxique et sémantique aussi bien que la preuve de théorèmes.

Nous commençons par présenter le langage Z et son système de preuve Z/EVES. Dans une deuxième étape, nous modélisons à un haut niveau d'abstraction, selon la notation Z, les concepts qui nous paraissent essentiels pour la conception et le développement d'un système à base d'agents mobiles.

3.1 La spécification formelle : langage & outil de preuve

Tout langage formel de spécification est basé sur des concepts mathématiques bien établis. Ces concepts constituent la syntaxe et la sémantique du langage. Dans ce qui suit nous commençons par présenter les concepts de base de la notation Z.

3.1.1 La Notation Z

La notation Z, comme présentée dans [WD96], est un langage orienté modèle basé sur la théorie des ensembles et la logique de premier ordre. En plus de son langage mathématique, la notation Z est distinguée par son langage de schéma qui fournit une description structurée des états du système et les opérations potentielles sous lesquelles l'état du système peut changer. Par conséquent, une spécification en Z peut décrire les aspects statique et dynamique du système. En plus, le langage Z permet la modélisation du système à plusieurs niveaux d'abstraction, partant du niveau le plus abstrait allant à des niveaux plus concrets.

L'aspect statique est spécifié par un schéma d'état (*state schema*) qui se compose de deux parties :

- Partie *déclarative* : elle est essentielle dans la description d'un schéma d'état. Dans cette partie, on peut déclarer différentes structures de données composées telles que : ensemble, séquence, bag, types libres, etc ...
- Partie *prédicative* : qui contraigne les valeurs des variables déclarées.

Il y a deux formes pour représenter un schéma d'état : horizontale ou verticale. Le schéma suivant est présenté verticalement :

<i>Schema</i>
<i>Declaration</i>
<i>Predicate_i, ..., Predicate_j</i>

Les liens entre les composants du système, peuvent être décrits formellement soit par des relations soit par des fonctions. De même les relations et les fonctions pourraient être composées d'une partie déclarative et une partie prédicative.

L'aspect dynamique, décrit la variation des états du système par la spécification des opérations potentielles sur un état. Le schéma Δ *Schema* spécifie les relations entre l'état avant l'opération (*Schema*) et l'état après l'opération (*Schema'*).

Δ <i>Schema</i>
<i>Schema</i>
<i>Schema'</i>
<i>Predicate_m, ..., Predicate_n</i>

En Z une opération peut impliquer des paramètres d'entrée (input) et éventuellement des paramètres de sortie (output). Ces paramètres seront insérés dans la partie déclarative du schéma d'opération. Une variable qui se termine par un point d'interrogation (?) représente une entrée. Une variable qui se termine par une marque d'exclamation (!) représente un résultat (output). La partie prédicative établit le rapport entre les valeurs de ces paramètres et les états du système avant et après l'opération. $\Delta Schema$ représente l'état du système avant et après l'opération.

<i>Operation</i> $\Delta Schema$ <i>i?</i> : <i>Input</i> <i>o!</i> : <i>Output</i> ⋮ ...
--

Afin de vérifier certaines propriétés, une spécification Z peut faire l'objet de preuves formelles qui peuvent être assistées ou automatisées par des moteurs de preuve de théorèmes. Dans la section suivante, nous allons présenter brièvement le démonstrateur de théorèmes Z/EVES. Ce démonstrateur a été adopté dans la preuve des propriétés des systèmes à base d'agents mobiles.

3.1.2 Z/EVES : outil de preuve formelle

Z/EVES [Saa97, MS97] est un outil pour analyser les spécifications Z. Il peut être utilisé pour l'analyse, le contrôle de type (type checking), le contrôle de domaine (domain checking), le calcul des préconditions (precondition calculation), la preuve des raffinements et la preuve des théorèmes.

Ce logiciel conjoint la force du démonstrateur automatique de théorèmes EVES avec la syntaxe du langage de spécification Z. Ainsi Z/EVES peut accepter soit des paragraphes Z ou des commandes Z/EVES.

Z/EVES peut fonctionner en deux modes indépendants : mode textuel ou mode graphique. L'utilisation de l'interface graphique est plus conviviale. Cette interface a été adoptée dans nos preuves.

Le principe de Z/EVES fonctionne selon la méthode du but fixé [Bol01]. Cette méthode consiste à prouver le prédicat appelé but (eng. goal). En fonction de sa complexité, un but courant peut être subdivisé en plusieurs sous-buts dont l'ensemble constitue le but global. La preuve d'un but peut se faire par plusieurs commandes de preuve. Chaque étape (dont on applique une commande de preuve) transforme le but en un nouveau but équivalent. Une fois le but est transformé au prédicat *true*, la preuve est alors terminée.

Les étapes de preuve dans Z/EVES se font avec deux classes de commandes : des com-

mandes utilisées de façon manuelle et d'autres utilisées de façon automatique. Parmi les commandes manuelles, nous mentionnons principalement les commandes *apply* et *use*. La commande *apply* permet d'appliquer un théorème au but. Ce théorème doit être une règle de réécriture. La commande *use* permet de rajouter un théorème dans les hypothèses du but courant.

Quant aux commandes utilisant les théorèmes et les définitions de façon automatique nous mentionnons les commandes *Simplify*, *Rewrite* et *Reduce*. Ces commandes de réduction utilisent uniquement les théorèmes en état actif (eng. enabled). Les théorèmes en état inactif (eng. disabled) peuvent être définis dans des prédicats étiquetés (eng. labelled predicates).

Le principe des commandes de réduction consiste à parcourir le but courant, rassembler les suppositions et appliquer des réductions sur les prédicats et les expressions du but courant. Dans le parcours du but, chaque formule ou sous-formule du but courant est examinée. Elle peut être remplacée par une formule équivalente et considérée comme plus simple.

Tirons profit du pouvoir d'expression du langage Z, son aptitude à modéliser le système à différents niveaux d'abstraction et la facilité de raisonner rigoureusement par le biais de son démonstrateur de théorème Z/EVES, nous proposons selon la notation Z une modélisation conceptuelle des systèmes à base d'agents mobiles [LHKJ05, LHKJM06]. Cette modélisation constitue un référentiel dans les différentes phases de développement d'un système à base d'agents mobiles. Elle spécifie d'une manière concise et précise les différentes entités qui constituent un système à base d'agents mobiles ainsi que les liens entre elles.

3.2 Modélisation conceptuelle des systèmes à base d'agents mobiles

Pour mieux comprendre les systèmes à base d'agents mobiles, nous devons passer en revue les différentes étapes du cycle de vie de l'agent mobile. Nous dégagerons par conséquent, les concepts fondamentaux pour mieux assurer l'évolution de l'agent dans un système distribué. Ces concepts seront spécifiés formellement selon la notation Z.

3.2.1 Cycle de vie de l'Agent Mobile

Le cycle de vie d'un agent mobile désigne toutes les étapes de progression de l'agent, depuis sa création jusqu'à sa terminaison. La création d'un agent mobile peut être initiée soit par un agent utilisateur (eng. owner) [TTK⁺99, LZ01] soit par un autre agent mobile (eng. parent agent) [PXB03]. À la création de l'agent un identifiant unique est assigné, ainsi qu'un état initial pour commencer immédiatement l'exécution de ses tâches. La par-

ticularité des agents mobiles réside dans le fait qu'ils ont l'aptitude de choisir eux-mêmes de se déplacer entre différents systèmes afin d'accomplir au mieux leurs tâches et travailler localement sur les ressources et les services du nouveau système. Dans la plupart des cas, l'agent choisit le système qui lui offre plus de services et de ressources nécessaires pour son exécution. Dès que l'agent décide de migrer, le système qui le supporte arrête son exécution, enregistre son état d'exécution et le transfère (code, données et éventuellement état d'exécution) au nouvel emplacement.

Le système d'accueil de l'agent, examine son profil afin de se protéger contre les agents malveillants. Une fois l'agent est authentifié, il peut alors poursuivre son exécution tout en profitant d'un accès local aux ressources et/ou services autorisés. L'agent mobile peut aussi interagir avec les agents déployés dans ce système.

À la fin de son exécution, l'agent doit retourner, éventuellement, à son emplacement d'origine avec les résultats finaux sinon il doit envoyer ces résultats à son propriétaire à travers une application externe tel que l'e-mail. Une fois la durée de vie de l'agent est expirée, il sera alors détruit.

3.2.2 Taxonomie commune : Spécification

Un système à base d'agents mobiles ne se réduit pas à un système informatique centralisé, il est composé d'un ensemble de machines inter-connectées appelées hôtes (eng. host). Plusieurs définitions ont été proposées pour ce concept. Ces définitions dépendent du point de vue de leurs auteurs et du domaine d'application de l'agent.

D'après l'étude de plusieurs systèmes [YHWL03, PS97, BHRS98], nous retenons qu'un *hôte* est une machine capable d'héberger un ou plusieurs systèmes d'exécution d'agents mobiles. Il constitue généralement un noeud dans le réseau. Une machine hôte est décrite par un nom et un ensemble de ressources informatiques (caractéristiques matérielles).

Soit [*Propriety*] l'ensemble des propriétés, et soit [*CResource*] l'ensemble des ressources informatiques telles que CPU, mémoire, ressources de réseau, etc. Formellement, nous décrivons un hôte par le schéma suivant :

Host

Name_H : Propriety

Resources : \mathbb{P}_1 CResource

3.2.2.1 Système d'exécution d'agents mobiles

Le système d'exécution d'agents mobiles (eng. Agent System) est défini par l'environnement dans lequel l'agent mobile peut évoluer. Cet environnement offre les fonctionnalités de base pour l'exécution des agents mobiles. En effet, il assure : la création et l'initiation de l'agent mobile, l'accueil des agents visiteurs, la communication (locale et distante) entre les agents, l'accès aux ressources, la migration de l'agent, etc. Ces services de

contrôle et d'autres d'applications seront assurés par des agents de services (eng. Service Agent). Ainsi un Agent System (*AgS*) est décrit par :

- un nom qu'il l'identifie.
- une localisation : le hôte sur lequel il est installé,
- un ensemble de services offerts : un service peut être accompli par un ou plusieurs agents de service, et
- un ensemble de ressources mises à sa disposition : cet ensemble peut évoluer en fonction des besoins des agents visiteurs. Lorsque l'agent quitte le système, certaines ressources seront libérées.

Pour protéger les ressources et les services contre les accès non autorisés, l'AgS doit mettre en place une politique de sécurité. Une spécification abstraite des politiques de sécurité sera présentée dans le chapitre suivant.

3.2.2.2 Agent de Service

Un agent de service (*SAg*) est un agent stationnaire qui peut coopérer avec d'autres agents afin de satisfaire le service demandé par un agent visiteur. Nous avons proposé dans [LHKJ04b, LHKJ04a] une spécification pour l'agent stationnaire. Il est décrit par son état mental (*MentalState*) qui regroupe l'ensemble de ses propriétés. Ces dernières sont classées en trois sous-ensembles : les compétences (*Capability* == $\mathbb{P} \text{Propriety}$), les connaissances (*knowledge*) et les croyances (*Belief*). Formellement un agent stationnaire est décrit par le schéma suivant :

$\begin{array}{l} \textit{MentalState} \\ \textit{capability} : \mathbb{F}_1 \textit{Capability} \\ \textit{knowledge} : \mathbb{F} \textit{Propriety} \\ \textit{belief} : \mathbb{F} \textit{Propriety} \end{array}$	$\begin{array}{l} \textit{Agent} \\ \textit{MentalState} \\ \textit{belief} \cup \textit{knowledge} \neq \{\} \end{array}$
--	--

En fonction de la complexité du service requis, un ou plusieurs agents vont exercer le service en question. En effet un service peut avoir deux représentations différentes : soit une tâche atomique qui peut être accomplie par un seul *SAg*, soit une représentation plus complexe (désignée dans [LHKJ04b] par le concept 'objectif commun' : *CommonObjective*). Dans ce cas, un service sera le résultat de la coopération de plusieurs *SAg*. Ces agents coopèrent selon le modèle de coopération présenté dans [LHKJ04b, LHKJ04a].

Soit $\textit{Service} ::= S\langle\langle \textit{Task} \rangle\rangle \mid Sc\langle\langle \textit{CommonObjective} \rangle\rangle$.

La fonction $\textit{Assume_T}$ détermine les agents qui peuvent assumer une tâche t .

$$\mid \textit{Assume_T} : \textit{Task} \rightarrow \mathbb{F} \textit{Agent}$$

La fonction $\textit{Assume_Oc}$ détermine le groupe d'agents qui peuvent assumer un objectif commun Oc .

$$\mid \textit{Assume_Oc} : \textit{CommonObjective} \rightarrow \mathbb{F}(\mathbb{F}_1 \textit{Agent})$$

Dans la partie prédicative du schéma *AgentSystem* nous vérifions que les agents définis dans un système sont capables d'accomplir les services de ce système. De plus, nous vérifions, avec le dernier prédicat, que les ressources mises à la disposition d'un système d'agents doivent appartenir à l'ensemble des ressources de la machine hôte dans laquelle il est localisé.

<i>AgentSystem</i>
<i>Name_AgS</i> : <i>Propriety</i>
<i>Location_AgS</i> : <i>Propriety</i>
<i>Reserved_res</i> : \mathbb{F}_1 <i>CResource</i>
<i>AgentS</i> : \mathbb{F}_1 <i>Agent</i>
<i>Services</i> : \mathbb{F} <i>Service</i>
$\forall s : \textit{Service} ; t : \textit{Task} \mid s = St \wedge s \in \textit{Services} \bullet$
$\exists a : \textit{Agent} \mid a \in \textit{AgentS} \bullet a \in \textit{Assume_Tt}$
$\forall s : \textit{Service} ; Oc : \textit{CommonObjective} \mid s = ScOc \wedge s \in \textit{Services}$
$\bullet \exists Ga : \mathbb{P}_1 \textit{Agent} \mid Ga \subseteq \textit{AgentS} \bullet Ga \in \textit{Assume_OcOc}$
$\forall h : \textit{Host} \mid h.\textit{Name_H} = \textit{Location_AgS} \bullet \textit{Reserved_res} \subseteq h.\textit{Resources}$

3.2.2.3 Agent Mobile

Un agent mobile (*MAg*) est généralement défini par une entité logicielle capable, en cours de son exécution, de migrer d'un emplacement à un autre afin de se rapprocher des ressources et des services nécessaires pour accomplir convenablement son but.

Ainsi nous revenons sur la spécification de l'agent stationnaire et l'étendre par un certain nombre d'attributs nécessaires pour exprimer la mobilité de l'agent.

Pour pouvoir localiser l'agent mobile pendant ses migrations, ce dernier doit être identifié par un nom unique qui sera défini lors de sa création. L'agent mobile agit en fonction de ses compétences et de ses connaissances afin de réaliser les tâches qui lui sont affectées. Il définit sa nouvelle localisation en fonction de ses besoins en terme de ressources et de services et en fonction de sa vue partielle des connexions inter-hôtes. L'ensemble des systèmes visités par l'agent constitue son itinéraire. En plus, un agent mobile doit mémoriser son emplacement de création pour pouvoir y revenir dès qu'il termine son exécution. Formellement, un agent mobile est spécifié par le schéma *MobileAgent*.

Nous distinguons, dans la partie déclarative, une inclusion simple du schéma *Agent*. Ceci exprime le fait qu'un agent mobile hérite toutes les propriétés d'un agent stationnaire. En plus nous déclarons les propriétés :

- identité de l'agent mobile (*Identity_MAg*),
- localisation de création de l'agent (*CLocation_MAg*),
- localisation courante de l'agent (*HLocation_MAg*),
- durée de vie de l'agent (*TTL_MAg*),
- itinéraire de l'agent (*Itinerary_MAg*),

- vue partielle de l'agent (*Partial_View*),
- ensemble des tâches affectées à l'agent (*Actions*), et
- mission de l'agent pour chaque localisation visitée (*Mission*).

Les quatre premières propriétés citées doivent appartenir à l'ensemble des connaissances de l'agent. Cette contrainte est exprimée par [C1].

<i>MobileAgent</i>	
<i>Agent</i>	
<i>Identity_MAg</i> : <i>Propriety</i>	
<i>CLocation_MAg</i> : <i>Propriety</i>	
<i>HLocation_MAg</i> : <i>Propriety</i>	
<i>TTL_MAg</i> : <i>Propriety</i>	
<i>Itinerary_MAg</i> : seq <i>Propriety</i>	
<i>Partial_View</i> : $\mathbb{F}(\text{Propriety} \leftrightarrow \text{Propriety})$	
<i>Actions</i> : $\mathbb{F}_1 \text{ Task}$	
<i>Mission</i> : $\mathbb{F}(\text{AgentSystem} \leftrightarrow \mathbb{F} \text{ Task})$	
$\{ \text{Identity_MAg}, \text{CLocation_MAg}, \text{HLocation_MAg}, \text{TTL_MAg} \} \subset \text{knowledge}$	
	[C1]
$\forall \text{As} : \text{AgentSystem}; T : \text{Task} \mid (\text{As}, \{T\}) \in \text{Mission} \bullet T \in \text{Actions}$	
	[C2]

Nous vérifions avec la contrainte [C2] que la mission de l'agent, réalisée au sein d'une localisation visitée, fait partie de l'ensemble des tâches qui lui sont affectées à son initialisation.

Formellement, un système à base d'agents mobiles sera décrit par le schéma *MobileAgentSystem*. Une telle description, caractérise un système à base d'agents mobiles (MbAS) par quatre éléments :

- l'ensemble des hôtes définis dans le MbAS,
- l'ensemble des interconnexions entre ces hôtes,
- les systèmes d'exécution d'agents logés dans ces hôtes, et
- les agents mobiles initiés sur ces systèmes d'exécution.

<i>MobileAgentSystem</i>	
$Host_system : \mathbb{F}_1 Host$	
$Connection_host : \mathbb{F}(Propriety \leftrightarrow Propriety)$	
$AgS_system : \mathbb{F} AgentSystem$	
$MAG_system : \mathbb{F} MobileAgent$	
$\#AgS_system \geq 2$	[C3]
$\forall g : AgentSystem \mid g \in AgS_system$ • $\exists h : Host \mid h \in Host_system \bullet g.Location_AgS = h.Name_H$	[C4]
$\forall g_1, g_2 : AgentSystem \mid g_1 \in AgS_system \wedge g_2 \in AgS_system$ • $g_1.Name_AgS \neq g_2.Name_AgS$	[C5]
$\forall a_1, a_2 : MobileAgent \mid a_1 \in MAG_system \wedge a_2 \in MAG_system$ • $a_1.Identity_MAG \neq a_2.Identity_MAG$	[C6]
$\forall a : MobileAgent \mid a \in MAG_system \bullet \exists g : AgentSystem \mid g \in AgS_system$ • $a.CLocation_MAG = g.Name_AgS$	[C7]
$\forall a : MobileAgent \mid a \in MAG_system \bullet \exists g : AgentSystem \mid g \in AgS_system$ • $a.HLocation_MAG = g.Name_AgS$	[C8]
$\forall a : MobileAgent \mid a \in MAG_system$ • $\exists g : AgentSystem \mid g \in AgS_system$ • $\forall i : \mathbb{N} \mid 1 \leq i \leq \#a.Itinerary_MAG$ • $g.Name_AgS \in \{a.Itinerary_MAGi\}$	[C9]
$\forall p_1, p_2 : Propriety \mid \{(p_1, p_2)\} \in Connection_host$ • $\exists h_1, h_2 : Host \mid h_1 \in Host_system \wedge h_2 \in Host_system$ • $p_1 = h_1.Name_H \wedge p_2 = h_2.Name_H$	[C10]
$\forall a : MobileAgent \mid a \in MAG_system \bullet a.Partial_View \subseteq Connection_host$	[C11]

Dans la partie prédicative, nous vérifions un certain nombre de contraintes étiquetées [Ci] :

- C3 vérifie que la cardinalité des systèmes supportant l'exécution des agents mobiles est supérieure strictement à 1.
- C4 vérifie que l'ensemble des AgS déclarés sont hébergés dans les hôtes constituant le MbAS.
- C5 et C6 vérifient respectivement l'unicité du nom d'un AgS et l'unicité de l'identité d'un MAG.
- C7 stipule que l'emplacement de création d'un agent mobile du système doit être dans un des AgS qui constitue le MbAS.
- C8 et C9 vérifient que la migration d'un agent mobile se fait toujours entre les AgS relatifs à un même MbAS.
- C10 montre que les connexions entre hôtes sont modélisées par l'ensemble des relations définies entre les noms des hôtes du système.
- C11 vérifie que la vue partielle d'un agent mobile fait partie de l'ensemble des connexions entre les hôtes d'un même système.

3.3 Conclusion

La modélisation générique des systèmes à base d'agents mobiles est encore un domaine de recherche ouvert. Nous avons proposé dans ce chapitre une tentative de modélisation conceptuelle et générique qui couvre les différents concepts nécessaires pour la définition d'un système à base d'agents mobiles, ainsi que les liens entre eux.

Cette modélisation étudie uniquement la dimension structurelle des systèmes à base d'agents mobiles. La dimension dynamique sera développée dans le chapitre suivant par la définition d'un cadre formel pour exprimer la sécurité des systèmes à base d'agents mobiles pour les deux dimensions.

4

Cadre formel pour la sécurité des systèmes à base d'agents mobiles

La sécurité dans les systèmes à base d'agents mobiles est double : elle vise la protection des agents mobiles d'une part, et la protection des systèmes d'accueil des agents, d'autre part. Les solutions proposées pour la sécurité des systèmes distribués s'avèrent insuffisantes. De plus, il n'y a aucune solution qui traite les différentes préoccupations de sécurité dans les systèmes à base d'agents mobiles. Pour atteindre les exigences de sécurité, notre modèle d'agent mobile [LHKJ05], présenté dans le chapitre précédent, doit supporter la spécification de diverses politiques de sécurité afin de contrôler le comportement des entités du système (*MobileAgent* et *AgentSystem*) et de les protéger contre les différents types d'attaques.

Nous proposons dans ce chapitre un cadre formel pour la sécurité des systèmes à base d'agents mobiles et qui porte sur la spécification, la vérification, la reconfiguration et l'adaptabilité [LHKJM08, MMM09]. Le cadre de spécification propose une définition explicite et générique des politiques de sécurité. Il peut être enrichi par des concepts liés à un ou plusieurs modèles de sécurité. À titre d'illustration, nous présentons un enrichissement basé sur les concepts du modèle RBAC [FSG⁺01].

Plusieurs cas d'inconsistance peuvent se présenter entre les règles de sécurité d'une même politique [RZFG00, CCB06]. Pour éviter toute anomalie capable de réduire la performance de la politique, nous devons vérifier la consistance d'une telle politique. Ainsi, nous associons au cadre de spécification un cadre de vérification pour prouver formellement la consistance des spécifications proposées ainsi que la consistance des règles de sécurité intra politique.

Pour répondre aux changements dynamiques des exigences de sécurité et l'émergence continue de nouvelles menaces de sécurité, nous proposons un troisième cadre pour la reconfiguration des politiques de sécurité.

Enfin, avec le cadre d'adaptabilité, nous exprimons la mobilité de l'agent par son adaptation aux nouvelles exigences de sécurité du système visité.

4.1 Cadre de spécification

La définition d'une politique de sécurité est une étape cruciale pour la mise en œuvre effective de la sécurité au sein d'un système informatique donné. Deux types d'entités sont concernés par la définition des politiques de sécurité dans les systèmes à base d'agents mobiles : les agents mobiles et les systèmes qui supportent leur exécution.

Dans ce qui suit, nous proposons une spécification formelle de l'ensemble des lois et des règlements qui régissent la façon de gérer et de protéger les ressources sensibles au sein d'un système à base d'agents mobiles. La spécification de ces réglementations sera décrite par une politique de sécurité.

Cette spécification sera définie en concordance avec les concepts liés à la modélisation des systèmes à base d'agents mobiles, proposés dans le chapitre précédent.

4.1.1 Politique de sécurité : Spécification

Différentes classes de politiques de sécurité peuvent être distinguées dans la littérature à savoir les politiques de contrôle d'accès, les politiques de disponibilité et les politiques des flux d'informations [Sch00].

- Contrôle d'accès : consiste à limiter l'ensemble des opérations réalisées au sujet d'un objet. Cette classe peut être exprimée, uniquement, par des politiques d'autorisation. Pour donner plus de flexibilité d'expression, récemment, plusieurs travaux de recherche proposent les politiques d'interdiction pour exprimer le contrôle d'accès [CCB06].
- Flux d'information : le contrôle d'accès est insuffisant pour régler la propagation et le flux des informations entre les entités [Zda04]. En effet ce type de politique consiste à définir un niveau de sécurité pour les informations. Il s'agit en fait de réduire les inférences qui peuvent se faire suite à la perception du comportement du système. De même, cette classe peut être exprimée par des politiques d'autorisation et/ou d'interdiction.
- Disponibilité : ce type de politique assure l'utilisation convenable et non abusive des services et des ressources du système. En d'autres termes, elle évite toute forme de déni de service et de ressources. Cette classe peut être exprimée soit par des politiques d'interdiction soit par des politiques d'obligation [CCBR06].

Ainsi, nous retenons de cette brève description, trois types de base pour exprimer les différentes classes de politiques de sécurité, à savoir : l'autorisation, l'interdiction et

l'obligation.

Dans le but de proposer une définition complète des politiques de sécurité, nous définissons trois constructeurs pour exprimer l'autorisation (*Auth*), l'interdiction (*Prohb*) et l'obligation (*Oblig*). Formellement, nous présentons ces constructeurs par le type libre *SConstruct*.

$$SConstruct ::= Auth \mid Prohb \mid Oblig$$

Dans le cadre des systèmes à base d'agents mobiles deux types d'entités seront concernés par la définition de ces types de politiques de sécurité : les agents mobiles et les systèmes qui supportent leur exécution. Nous rassemblons, ces deux classes d'entités en une seule classe, dénommée *SEntity*, qui sera décrite formellement comme suit :

$$SEntity ::= MAg\langle\langle MobileAgent \rangle\rangle \mid AgS\langle\langle AgentSystem \rangle\rangle$$

Ces entités cherchent à protéger un certain nombre d'objets sensibles (eng. secure object). Un objet à protéger dénoté par *SObject*, peut être une donnée (propriété), un service ou une ressource : $SObject ::= D\langle\langle Data \rangle\rangle \mid Sr\langle\langle Service \rangle\rangle \mid Rs\langle\langle CResource \rangle\rangle$.

Nous spécifions une politique de sécurité par un ensemble fini de règles de sécurité. Cette définition apporte plus de simplicité et de précision dans l'expression des politiques de sécurité. En plus, elle permet d'intégrer, de manière cohérente, différents types de politiques de sécurité dans une même structure unifiée. Nous définissons, formellement, une règle de sécurité par le schéma suivant :

<i>SRule</i>	
<i>Name</i> : Propriety	
<i>Type</i> : <i>SConstruct</i>	
<i>Interested</i> : <i>SEntity</i>	
<i>RSubject</i> : $\mathbb{F}_1 SEntity$	
<i>Target</i> : $\mathbb{F} SObject$	
<i>Context</i> : Condition	
<i>Actions</i> : $\mathbb{F}_1 Action$	
$\forall r : CResource \mid Target = \{Rsr\}$	
<ul style="list-style-type: none"> • ($Type = Auth \vee Type = Prohb$) 	
<ul style="list-style-type: none"> $\wedge (\exists s_1 : AgentSystem \bullet (Interested = AgSs_1$ 	[C12]
<ul style="list-style-type: none"> $\wedge r \in s_1.Reserved_res)$ 	
<ul style="list-style-type: none"> $\wedge \neg (\exists s_2 : AgentSystem \bullet AgSs_2 \in RSubject)$ 	
$\forall sc : Service \mid Target = \{Srsc\}$	
<ul style="list-style-type: none"> • ($Type = Auth \vee Type = Prohb$) 	
<ul style="list-style-type: none"> $\wedge (\exists s_1 : AgentSystem \bullet (Interested = AgSs_1$ 	[C13]
<ul style="list-style-type: none"> $\wedge sc \in s_1.Services)$ 	
<ul style="list-style-type: none"> $\wedge \neg (\exists s_2 : AgentSystem \bullet AgSs_2 \in RSubject)$ 	
$Type = Oblig \Rightarrow \{Context\} \neq \emptyset$	[C14]

Dans la partie supérieure du schéma, nous déclarons :

- Type : le type de la règle de sécurité qui peut être soit une autorisation, soit une interdiction ou une obligation.
- Interested : l'entité concernée par la règle de sécurité. Elle peut être soit un agent mobile soit un système d'agents.
- RSubject : le sujet de la règle de sécurité. Il est défini par un ensemble non vide d'entités sur lesquelles on va appliquer la règle de sécurité.
- Target : l'ensemble des objets à protéger. Cet ensemble peut être vide.
- Context : le contexte d'application de la règle de sécurité. Il désigne les contraintes qui limitent l'applicabilité de la règle.
- Actions : l'ensemble des opérations ou des actions imposées par la règle de sécurité afin d'atteindre le comportement désiré.

La spécification d'une règle de sécurité doit satisfaire les contraintes définies dans la partie prédicative : C12 précise que si l'objet de sécurité d'une règle est une ressource informatique, alors l'entité concernée par la règle (*Interested*) doit être un *AgS* et, par conséquent, l'entité sujet de sécurité sera un *MAG*. En plus nous vérifions par C12 que l'*AgS* dénoté par *Interested* peut contrôler uniquement l'accès à ses propres ressources. De même, C13 précise que lorsque l'objet de sécurité (*Target*) est un service, alors cette règle concerne un *AgS* qui peut protéger uniquement ses propres services contre les agents mobiles malveillants. Pour s'aligner avec les travaux de recherche les plus récents, portant sur la spécification des politiques d'obligation, nous associons à une obligation une contrainte contextuelle qui précise l'événement de son déclenchement. Cette supposition est exprimée par la contrainte C14.

Formellement une politique de sécurité est modélisée selon le schéma *SPolicy*. Nous déclarons dans la première partie de ce schéma : le sujet de la politique (i.e. l'entité concernée par la mise en place de cette politique) et l'ensemble des règles de sécurité qui constituent la politique. Ces règles concernent la même entité ; sujet de la politique. Deux règles qui appartiennent à la même politique ne doivent jamais avoir le même nom. Nous spécifions, formellement, les deux contraintes citées, respectivement par les prédicats étiquetés C15 et C16.

<i>SPolicy</i>	
<i>Subject</i> : <i>SEntity</i>	
<i>Rules</i> : $\mathbb{F} \text{SRule}$	
$\forall r : \text{SRule} \mid r \in \text{Rules} \bullet r.\text{Interested} = \text{Subject}$	[C15]
$\forall a, b : \text{SRule} \mid a \in \text{Rules} \wedge b \in \text{Rules} \wedge a \neq b \bullet a.\text{Name} \neq b.\text{Name}$	[C16]

4.1.2 Politique de sécurité : Raffinement

Plusieurs modèles de sécurité ont été proposés dans la littérature. Ces modèles peuvent être regroupés en trois classes [CCB06] : contrôle d'accès, contrôle de flux et contrôle

d'usage. Il est intéressant d'intégrer plusieurs modèles dans un même système de protection afin d'améliorer le niveau de sécurité et satisfaire différents besoins de sécurité. Par exemple, dans [ACBC07] les auteurs intègrent un modèle de contrôle d'accès avec un autre modèle de contrôle de flux.

Dans la section 4.1.1, nous avons proposé une définition générique qui supporte la modélisation de différents types de politique de sécurité. L'expressivité offerte par la notation Z , permet d'aller en profondeur dans une spécification formelle et de la raffiner par l'ajout de certaines informations. En effet, nous pouvons enrichir la spécification de la politique de sécurité, initialement proposée, par plusieurs concepts en rapport avec un ou plusieurs modèles de sécurité tels que RBAC [FSG⁺01], OrBAC [KBB⁺03], DTE [BSS⁺95], etc.

Dans ce qui suit, nous étalons la spécification de la politique de sécurité par des concepts relatifs au modèle RBAC.

Le modèle RBAC comporte six concepts de base : utilisateur, rôle, session, autorisation, opération, et objet. Les relations entre ces concepts sont définies comme suit : à un utilisateur on peut assigner un ou plusieurs rôles, un même rôle peut être joué par plusieurs utilisateurs. Les autorisations seront assignées aux rôles, et par conséquent un utilisateur peut acquérir une autorisation par le fait de jouer le(les) rôle(s) approprié(s). Un utilisateur peut activer dans une session un ou plusieurs rôles nécessaires pour exécuter une tâche donnée. Ces relations ont été spécifiées formellement, dans le schéma *RBAC_Policy*, respectivement par les fonctions suivantes : *assign_role*, *entity_role*, *role_permission* et *entity_sessions*. Conformément à la terminologie des systèmes à base d'agents mobiles, nous avons substitué le terme *utilisateur* par le terme *agent mobile* système d'agents dénoté par *SEntity*.

Nous vérifions avec les contraintes C17, ..., C20 le domaine (*dom*) et l'image (*ran*) de chacune des fonctions déclarées. Par exemple la contrainte C17 définit le domaine de la fonction *assign_role* par l'ensemble des entités de type *SEntity* constituant les *RSubject* de la politique *SPolicy*. La contrainte C18 définit le domaine de la fonction *entity_role* par l'ensemble d'images (*range*) de la fonction *assign_role*.

Selon le modèle RBAC, on peut exprimer uniquement des autorisations. En effet, nous modélisons cette supposition par la contrainte C21. La dernière contrainte C22 vérifie que si deux entités différentes jouent un même rôle (*r*) et une permission (*p*₁) est attribuée à ce rôle alors nécessairement il existe, selon la définition générique donnée avec *SPolicy*, deux règles de sécurité qui correspondent à la permission (*p*₁).

Cette spécification peut faire l'objet de plusieurs autres raffinements, par le fait de lui associer les concepts liés à la hiérarchie des rôles et la séparation des pouvoirs.

<i>RBAC_Policy</i>	
<i>S</i> Policy	
$assign_role : SEntity \rightarrow \mathbb{F}_1 Role$	
$entity_role : Role \rightarrow \mathbb{F} SEntity$	
$role_permission : Role \rightarrow \mathbb{F} SObject \times \mathbb{F}_1 SAction \times Condition$	
$entity_sessions : SEntity \times Session \rightarrow \mathbb{F}_1 Role$	
$dom\ assign_role$	[C17]
$= \{s : SEntity \mid \forall r : SRule \mid r \in Rules \bullet s \in r.RSubject\}$	
$dom\ entity_role \in ran\ assign_role$	[C18]
$\forall r : Role \bullet entity_roler = \{s : SEntity \mid r \in assign_roles\}$	
$dom\ role_permission = dom\ entity_role$	[C19]
$dom\ entity_sessions$	[C20]
$= \{s : SEntity \mid \forall r : SRule \mid r \in Rules \bullet s \in r.RSubject\} \times Session$	
$\forall r : SRule \mid r \in Rules \bullet r.Type = Auth$	[C21]
$\forall e_1, e_2 : SEntity ; r : Role ; so : \mathbb{F} SObject ;$	
$sa : \mathbb{F}_1 SAction ; c : Condition$	
$\mid r \in assign_role(e_1) \wedge r \in assign_role(e_2)$	
$\wedge role_permission(r) = (so, sa, c)$	
$\bullet \exists r_1, r_2 : SRule \mid r_1 \in Rules \wedge r_2 \in Rules \wedge r_1 \neq r_2$	
$\bullet r_1.Type = r_2.Type = Auth \wedge r_1.Target = r_2.Target = so$	[C22]
$\wedge r_1.Context = r_2.Context = c$	
$\wedge r_1.Actions = r_2.Actions = sa$	
$\wedge e_1 \in r_1.RSubject \wedge e_2 \in r_2.RSubject$	

4.2 Cadre de vérification

Le cadre de spécification, proposé dans la section précédente, est non testé. En fait, il ne présente aucune preuve de consistance entre les prédicats définis. De plus, les différents cas d'inconsistance susceptibles de survenir dans la définition d'une politique de sécurité n'ont pas été traités.

Pour apporter plus de complétude et de consistance aux spécifications proposées et améliorer par conséquent la qualité de mise en place des politiques de sécurité, nous donnons un intérêt considérable aux preuves formelles à deux niveaux différents : prouver la consistance entre les prédicats d'une spécification donnée et prouver la consistance entre les règles intra politique.

Il est à noter que dans ce travail, nous n'avons pas porté une attention particulière à la définition d'une stratégie de résolution des éventuels cas d'inconsistances d'une politique.

4.2.1 Preuve formelle de la consistance des spécifications

Plusieurs outils, tels que Z/EVES [Saa97] et Isabelle-HOL [KSW96], offrent des fonctions de preuves syntaxique et sémantique des spécifications Z, mais ils ne vérifient pas la consistance des contraintes spécifiées dans la partie prédictive.

Afin de vérifier la non-contradiction entre les prédicats d'un schéma donné, il faut montrer l'existence, au moins, d'un état initial de ce schéma. Nous prouvons cette propriété par le théorème de l'initialisation [WD96].

Supposons que *State* décrit l'état du système, et que *StateInit* décrit un état initial de ce système. Une fois nous prouvons que : $\exists State \bullet StateInit$ alors nous démontrons qu'un état initial existe. D'où les contraintes de ce schéma d'état ne sont pas contradictoires.

Dans notre contexte, nous procédons à la vérification de la consistance du schéma *SRule* comme suit. Prenons l'exemple d'un système d'agents 'ss' qui autorise l'agent mobile 'mm' d'exécuter sous la condition 'c' l'ensemble des actions 'i' sur la ressource 'rr'. Nous décrivons cet état initial par le schéma *SRule_Init*.

<i>SRule_Init</i>
<i>SRule</i>
<i>Name</i> = R1
<i>Interested</i> = AgS ss
<i>RSubject</i> = {MAg mm}
<i>Target</i> = {Rs rr}
<i>Type</i> = Auth
<i>Context</i> = c
<i>Actions</i> = {i}

Le théorème d'initialisation *Consistency_SRule* nous a permis de prouver, avec l'outil Z/EVES, la consistance du schéma *SRule*.

$$\left| \begin{array}{c} Y \\ Y \end{array} \right| \text{theorem } Consistency_SRule \\ \exists SRule \bullet SRule_Init$$

4.2.2 Preuve formelle de la consistance intra-politique

Plusieurs cas d'inconsistance peuvent se présenter entre les règles de sécurité d'une même politique. Ces cas peuvent être divisés en trois classes [CCB06] : les conflits,

la redondance et le masquage. Les conflits sont à leur tour subdivisés en deux grandes classes [LS99] : les conflits de modalité et les conflits spécifiques à l'application. Pour éviter la deuxième sous-classe de conflits, des contraintes supplémentaires résultant de ces conflits doivent s'associer à la spécification de la politique ou devraient être spécifiées comme une méta-politique [DDL01]. Dans notre structure de vérification, cette classe n'a pas été considérée.

Une anomalie de masquage (shadowing) se produit lorsqu'il existe une règle qui s'applique toujours avant une règle moins prioritaire. Ce problème ne se présente pas dans notre structure, du fait que nous n'affectons pas un ordre de priorité entre les règles de sécurité.

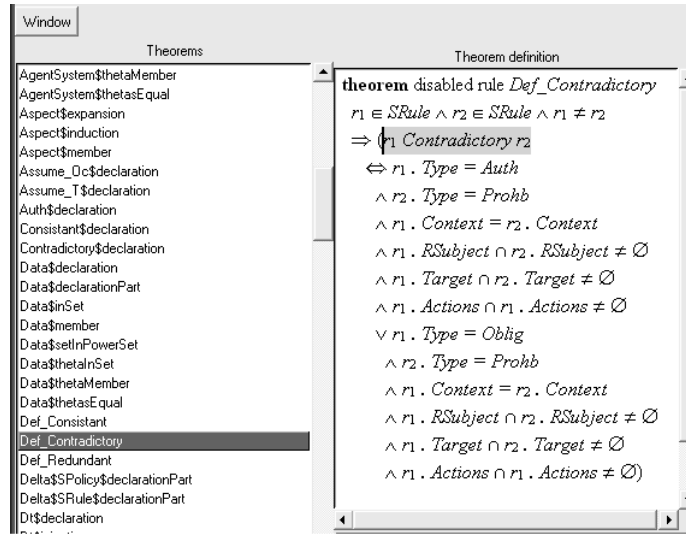
La contradiction et la redondance entre les règles de sécurité peuvent réduire la performance de la politique et même la rendre inefficace. En effet, il est indispensable d'associer à la spécification des politiques de sécurité, une structure qui vérifie leur consistance par le fait de prouver la non présence de conflits de modalités ni de redondances entre les règles intra politique.

Partant de la spécification des politiques de sécurité, nous avons distingué trois modalités différentes à savoir : l'autorisation, l'interdiction et l'obligation. Deux types de conflits peuvent se présenter entre ces modalités :

- Conflit entre une règle d'autorisation et une règle d'interdiction. Ces deux règles s'appliquent aux mêmes sujets, objets, actions et contexte. Il s'agit, en fait, d'une action autorisée et qui est prohibée par une règle d'interdiction.
- Conflit entre une règle d'obligation et une règle d'interdiction. Ces deux règles se rapportent aux mêmes sujets, objets, actions et contexte. Il s'agit, en fait, d'une règle qui exige l'exécution d'une action prohibée par une autre règle.

Pour décrire les relations qui peuvent exister entre deux ou plusieurs règles de sécurité, nous définissons trois relations : la relation unaire *Consistent* et deux relations binaires *Contradictory* et *Redundant*.

En Z, une opération, peut être de type *inrel* ou de type *prerel*. Avec l'opération *inrel*, nous exprimons une relation unaire. Alors que l'opération *prerel* exprime une relation binaire. Dans la spécification de la relation *Contradictory*, nous attachons à sa partie prédicative un label («*disabled rule Def_Contradictory*») auquel correspond un théorème (Fig. 4.1) de type règle de réécriture qui sera utilisée dans les preuves de consistance d'une politique de sécurité.

FIG. 4.1 – Théorème associé à la spécification de *Contradictory*

Ce théorème stipule qu’une règle de sécurité r_1 est contradictoire avec une deuxième règle r_2 , s’il existe une contradiction entre les modalités des deux règles (i.e. *Auth* contre *Prohb* ou *Olig* contre *Prohb*) et un chevauchement quadruple entre leurs sujets (*RSubject*), cibles (*Target*), actions (*Actions*) et contexte (*Context*).

syntax *Contradictory inrel Contradictory*

$_Contradictory_ : \text{SRule} \leftrightarrow \text{SRule}$
$\langle\langle \text{disabled rule Def_Contradictory} \rangle\rangle \forall r_1, r_2 : \text{SRule} \mid r_1 \neq r_2$
<ul style="list-style-type: none"> • $r_1 \text{ Contradictory } r_2$
$\Leftrightarrow (r_1.Type = \text{Auth} \wedge r_2.Type = \text{Prohb}$
$r_1.Context = r_2.Context$
$r_1.RSubject \cap r_2.RSubject \neq \emptyset$
$r_1.Target \cap r_2.Target \neq \emptyset$
$r_1.Actions \cap r_2.Actions \neq \emptyset)$
$\vee (r_1.Type = \text{Oblig} \wedge r_2.Type = \text{Prohb}$
$r_1.Context = r_2.Context$
$r_1.RSubject \cap r_2.RSubject \neq \emptyset$
$r_1.Target \cap r_2.Target \neq \emptyset$
$r_1.Actions \cap r_2.Actions \neq \emptyset)$

De la même manière, nous attachons à la partie prédicative de la relation *Redundant* le label ($\langle\langle \text{disabled rule Def_Redundant} \rangle\rangle$) auquel correspond un théorème de type règle

de réécriture. Ce prédicat stipule qu'une règle de sécurité r_1 est contradictoire avec une deuxième règle r_2 , s'ils sont de la même modalité et du même contexte d'application et l'ensemble des sujets (*RSubject*), des cibles (objets à protéger) et des actions de la règle r_1 sont respectivement incluses dans l'ensemble des sujets, des cibles et des actions de la règle r_2 .

syntax *Redundant inrel Redundant*

$\underline{\textit{Redundant}__} : \textit{SRule} \leftrightarrow \textit{SRule}$
$\langle\langle \textit{disabled rule Def_Redundant} \rangle\rangle \forall r_1, r_2 : \textit{SRule} \mid r_1 \neq r_2$ <ul style="list-style-type: none"> • $r_1 \textit{Redundant } r_2$ $\Leftrightarrow r_1.\textit{Type} = r_2.\textit{Type}$ $\wedge r_1.\textit{Context} = r_2.\textit{Context}$ $\wedge r_1.\textit{RSubject} \subseteq r_2.\textit{RSubject}$ $\wedge r_1.\textit{Target} \subseteq r_2.\textit{Target}$ $\wedge r_1.\textit{Actions} \subseteq r_2.\textit{Actions}$

Afin de prouver la consistance d'une politique donnée, nous devons vérifier l'inexistence de contradictions entre les règles de la politique en question et l'absence de règles redondantes. Sur cette base, nous proposons une définition rigoureuse de la consistance des politiques de sécurité, donnée par le théorème étiqueté *Def_Consistent*.

syntax *Consistent prerel Consistent*

$\underline{\textit{Consistent}__} : \mathbb{P} \textit{SPolicy}$
$\langle\langle \textit{disabled rule Def_Consistent} \rangle\rangle \forall p : \textit{SPolicy}$ <ul style="list-style-type: none"> • $\textit{Consistent } p$ $\Leftrightarrow (\forall a, b : \textit{SRule}$ <ul style="list-style-type: none"> $\mid a \in p.\textit{Rules} \wedge b \in p.\textit{Rules} \wedge a \neq b$ • $\neg a \textit{Redundant } b \wedge \neg a \textit{Contradictory } b)$

Pour illustrer notre cadre de vérification, nous prenons l'exemple d'une politique dénotée *Test_Policy*, composée de trois règles de sécurité X, Y et Z, définies comme suit :

```

m1, m2, m3 : MobileAgent
X, Y, Z : SRule
Test_Policy : SPolicy
c1, c2 : Condition
A1, A2, A3 : Action
CR1, CR2, CR3 : CResource

Test_Policy.Rules = {X, Y, Z}
X ≠ Y ∧ X ≠ Z ∧ Y ≠ Z
m1 ≠ m2 ∧ m3 ≠ m1 ∧ m2 ≠ m3
Y.RSubject = {MAgm1} ∧ Z.RSubject = {MAgm2}
X.RSubject = {MAgm3}
Y.Type = Auth ∧ Z.Type = Prohb ∧ X.Type = Oblig
Y.Target = {RsCR1} ∧ X.Target = {RsCR2} ∧ Z.Target = {RsCR3}
CR1 ≠ CR2 ∧ CR1 ≠ CR3 ∧ CR2 ≠ CR3
Y.Context = Z.Context = c1 ∧ X.Context = c2 ∧ c1 ≠ c2
Y.Actions = {A1} ∧ X.Actions = {A2} ∧ Z.Actions = {A3}
A1 ≠ A2 ∧ A1 ≠ A3 ∧ A2 ≠ A3

```

Pour prouver la consistance de *Test_Policy*, il faut démontrer le théorème suivant :

theorem *verif_consistency*
Consistent Test_Policy

```

Edit Window << < > >> Abort
Run Proof Script for verif_consistency
Y use axiom$18
Y prove
Y use Def_Contradictory[r1 := Z, r2 := Y]
Y use axiom$18
Y prove
Y use Def_Contradictory[r1 := Y, r2 := Z]
Y use axiom$18
Y prove
Y with normalization simplify
----- (action point) -----
Reduction Cases Quantifiers Normal Forms Equality
Formula
true
-
```

FIG. 4.2 – Traces de preuve du théorème *verif_consistency*

Le but est alors de prouver *ConsistentTest_Policy*. La preuve de ce but implique l'utilisation de la définition de la relation *Consistent* et par conséquent les définitions des relations *Contradictory* et *Redundant*. La fenêtre de preuve correspondante (eng. *proof window*), donnée par la figure 4.2, montre les traces de preuve (eng. *proof scripts*) qui font référence aux théorèmes associés aux parties prédicatives des relations *Consistent*, *Contradictory* et *Redundant*. Après avoir exécuté cette liste de commandes de preuve, nous avons obtenu le prédicat *true*. Ainsi, le théorème est prouvé et la politique '*Test_Policy*' est, par conséquent, consistante.

4.3 Cadre de reconfiguration

De nos jours, les systèmes d'information se retrouvent face à une émergence continuelle de nouvelles menaces de sécurité. Ce qui implique l'évolution dynamique des besoins de sécurité du système. Il est alors crucial de définir une structure de reconfiguration qui ajuste les politiques de sécurité aux nouvelles exigences du système.

Notre objectif dans la définition d'une structure de reconfiguration est double. Il consiste à : spécifier les opérations élémentaires de reconfiguration des politiques de sécurité et définir les théorèmes nécessaires pour maintenir la consistance de la politique après reconfiguration.

Pour s'accommoder aux nouvelles exigences de sécurité, nous aurons besoin de modifier la politique par la création, suppression et/ou modification d'une ou de plusieurs règles de sécurité. À l'ajout d'une nouvelle règle, nous aurons besoin de vérifier sa cohérence avec les autres règles de la politique. Le même type de vérification doit se faire à la modification d'une règle.

Formellement, nous spécifions une opération de reconfiguration par un schéma d'opération Z . La partie prédicative de ce schéma établit le lien entre les valeurs d'entrée de l'opération avec les états avant et après l'opération de reconfiguration.

Pour garantir l'exécution convenable d'une opération donnée, nous devons vérifier la satisfaction d'un certain nombre de pré-conditions. Dans notre cas, nous cherchons à préserver la propriété de consistance des politiques après reconfiguration. Cette contrainte sera vérifiée par la preuve d'un théorème qui précise les pré-conditions qui doivent être remplies à chaque opération de reconfiguration.

Nous spécifions une opération de suppression d'une règle de sécurité par le schéma suivant :

$\begin{array}{l} \text{Supp} \\ \Delta SPolicy \\ r? : SRule \end{array}$
$\begin{array}{l} Rules' = Rules \setminus \{r?\} \\ Subject' = Subject \end{array}$

Le théorème *SupplIsHonest* vérifie que la règle à supprimer ($r?$) est déjà présente dans la politique à reconfigurer. Après l'exécution de cette opération, $r?$ sera enlevée de la définition initiale de la politique.

Y	Y	theorem <i>SupplIsHonest</i> $\forall SPolicy; r?: SRule$ $ r? \in Rules \wedge (\forall a: SRule a \in Rules \cdot a.Name \neq r?.Name) \cdot \text{pre } Supp$
---	---	--

L'expression des pré-conditions de l'opération d'ajout, sera plus compliquée. Il s'agit de vérifier la consistance de la nouvelle règle de sécurité avec chacune des règles définies initialement dans la politique. De même, en modifiant la valeur des attributs d'une règle de sécurité nous devons satisfaire la même pré-condition de consistance.

Formellement, le schéma d'opération *AddSRule* stipule que la règle à ajouter $r?$ ne doit pas faire partie de la politique sujet de reconfiguration. Après l'exécution de cette opération, $r?$ sera insérée dans la configuration initiale de la politique.

<i>AddSRule</i>	$\Delta SPolicy$	$r?: SRule$
		$r? \notin Rules$ $Rules' = Rules \cup \{r?\}$ $Subject' = Subject$

Le théorème prouvé *AddSRuleIsHonest*, vérifie la préservation de la propriété de consistance, après avoir exécuté une opération d'ajout. En fait, ce théorème vérifie qu'il n'existe ni une redondance ni une contradiction entre la règle à ajouter $r?$ et l'ensemble des règles de la configuration initiale.

Y	Y	theorem <i>AddSRuleIsHonest</i> $\forall SPolicy; r?: SRule$ $ r? \notin Rules$ $\wedge (\forall r1, r2: SRule r1 \in Rules \cup \{r?\} \wedge r2 \in Rules \cup \{r?\} \wedge r1 \neq r2$ $\quad \cdot r1.Name \neq r2.Name)$ $\wedge (\forall r: SRule r \in Rules \cup \{r?\} \cdot r.Interested = Subject)$ $\wedge (\forall r: SRule r \in Rules \cdot (\neg r.Redundant\ r? \wedge \neg r.Contradictory\ r?))$ $\cdot \text{pre } AddSRule$
---	---	--

De même, nous maintenons la consistance suite à une opération de modification. À titre d'illustration, nous spécifions avec le schéma *Modif_Context* une opération qui consiste à modifier le contexte d'application d'une règle. Le théorème prouvé *ModifIsHonest* stipule que ce changement de contexte ne doit pas affecter la consistance de la politique.

<i>Modif_Context</i>		
$\Delta SRule$	Y Y	theorem <i>Modif_IsHonest</i> $\forall SRule; SPolicy; f?: Condition$ $ (Type = Oblig \Rightarrow \{f?\} \neq \emptyset)$ $\wedge (\forall rx: SRule \mid rx \in Rules$ <ul style="list-style-type: none"> • $(\exists r: SRule \mid r \in Rules \wedge r. Context = f? \wedge rx \neq r$ <ul style="list-style-type: none"> • $(\neg rx Contradictory r$ $\wedge \neg rx Redundant r$ $\wedge \neg r Redundant rx))) \cdot \text{pre } Modif_Context$
$f?: Condition$		
$Target' = Target$		
$Name' = Name$		
$Type' = Type$		
$Interested' = Interested$		
$RSubject' = RSubject$		
$Context' = f?$		
$Actions' = Actions$		

4.4 Cadre d'adaptation

Les systèmes à base d'agents mobiles sont caractérisés par un aspect très dynamique. Ceci est dû, principalement, à la migration des agents vers plusieurs systèmes avec des comportements et des politiques de sécurité différents.

En effet, à la visite d'un nouveau système, l'agent doit s'adapter dynamiquement au comportement de son système d'exécution. Ce système exige notamment un comportement, lié à des propriétés de sécurité, qui est exprimé par un ensemble de règles de sécurité. À ce niveau, l'agent doit adapter sa politique de sécurité afin de la rendre cohérente avec la politique du système.

Dans [KBC02], Ketfi A. et al. identifient quatre raisons d'adaptation (statique ou dynamique) d'une application :

- Adaptation correctionnelle : l'application en cours d'exécution ne se comporte pas correctement. La solution est d'identifier la partie (le composant) défectueuse et de la remplacer par une nouvelle version supposée correcte.
- Adaptation évolutive : au moment du développement de l'application, certaines fonctionnalités ne sont pas prises en compte. Avec l'évolution des besoins de l'utilisateur, l'application doit être étendue avec de nouvelles fonctionnalités.
- Adaptation perfective : l'objectif de ce type d'adaptation est d'améliorer les performances de l'application. Dans le cadre des politiques de sécurité l'objectif est d'améliorer la qualité de sécurité de l'application.

- Adaptation adaptative : même si l'application s'exécute correctement, parfois son environnement d'exécution comme le système d'exploitation, les composants matériels ou d'autres applications ou données dont elle dépend changent. Dans ce cas, l'application est adaptée en réponse aux changements affectant son environnement d'exécution.

L'adaptation évolutive a fait l'objet du cadre de reconfiguration (proposé dans la Section 4.3), dans la mesure où l'agent/système d'agents doit renforcer sa politique pour lutter contre les nouvelles formes d'attaque.

Dans la suite de cette section, nous nous intéressons uniquement au type d'adaptation adaptative, étant donné que l'agent doit s'adapter au changement de son système d'exécution.

Pour définir un cadre qui supporte l'adaptabilité des politiques de sécurité d'un agent mobile, nous commençons par identifier les différents cas d'incohérence susceptibles de se présenter entre la politique d'un agent mobile et la politique d'un système d'accueil. Ensuite, nous exprimons une politique d'adaptabilité par un ensemble de règles de la forme ECA (Événement, Condition, Action).

4.4.1 Identification et Spécification des différents cas d'incohérences

D'une manière générale, la sécurité distribuée se caractérise par l'existence de plusieurs politiques de sécurité desquelles peut résulter une politique de sécurité globale incohérente [Con05]. Typiquement, ce problème d'incohérence se présente dans le cadre des systèmes à base d'agents mobiles. En effet, de tels systèmes sont constitués d'un ensemble d'agents mobiles et de systèmes d'exécution d'agents dont chacun d'eux a ses propres contraintes de sécurité et met en œuvre une politique de sécurité spécifique à ses besoins.

Initialement, à sa création, un agent mobile est doté d'une politique de sécurité qui s'accorde avec la politique de son système de création. À sa migration, l'agent risque de rencontrer des problèmes d'incohérence avec les nouvelles contraintes de sécurité du système visité. Bien évidemment, ces incohérences ne se rapportent pas à des règles de contrôle d'accès, étant donné que la politique de sécurité d'un agent (respectivement d'un système d'agents) contrôle uniquement les ressources de l'agent (respectivement du système d'agents) en question.

Nous avons distingué, ainsi, trois formes d'incohérences qui peuvent survenir entre la politique d'un agent mobile et celle d'un système d'agents. Nous présentons pour chaque forme, les éléments d'incohérence et un cas d'exemple.

- Incohérence entre une règle d'interdiction (côté agent) et une règle d'obligation (côté système d'agents) : il s'agit, en fait, d'une règle de l'agent qui interdit l'exécution d'une action exigée par le système d'exécution. Ces deux règles se rapportent à une même action pour un même contexte d'application, ou bien ils se rapportent à une même action avec le contexte d'application de la règle d'interdiction reflète l'état

d'exécution de l'agent.

À titre d'exemple, nous citons le cas d'un agent qui dispose d'une règle de sécurité qui lui interdit d'exposer l'historique de ses parcours (eng. path history) aux systèmes non dignes de confiance par conséquent l'agent doit crypter son itinéraire avant de migrer vers un système. De l'autre côté, le système visité impose à chaque nouveau agent entrant de présenter l'historique de son parcours. Ces deux règles se contredisent incohérentes.

- Incohérence entre une règle d'obligation (côté agent) et une règle d'interdiction (côté système d'agents) : il s'agit, en fait, d'une règle de l'agent qui exige l'exécution d'une action interdite par le système d'agents. Par conséquent, les deux règles soit qu'ils se rapportent à une même action pour un même contexte d'application, ou bien ils se rapportent à une même action avec le contexte d'application de la règle d'interdiction reflète l'état du système.

Nous citons comme exemple, le cas d'une règle de sécurité qui oblige l'agent de se cloner avant de migrer vers un système particulier. Tandis que le système interdit les agents entrants de se cloner plus qu'une fois. De ce fait, dans le cas où l'agent a effectué auparavant une opération de clonage, le système va lui interdire de l'effectuer une autre fois.

- Incohérence entre une règle d'autorisation (côté agent) et une règle d'interdiction (côté système d'agents). Pour ce cas d'incohérence, l'agent est autorisé d'effectuer une action prohibée par le système. De même, les deux règles soit qu'ils se rapportent à une même action et un même contexte d'application, ou bien ils se rapportent à une même action et dont le contexte d'application de la règle d'interdiction reflète l'état du système.

À titre d'exemple, nous citons le cas d'un agent qui dispose d'une règle qui lui autorise de migrer vers un système spécifique (AgS1). Par contre, le système d'exécution de cet agent interdit ses agents d'aller sur le système AgS1.

Bien entendu, dans le cas où il s'agit d'une action interdite par l'agent, mais qui est autorisée du côté du système d'agents, aucune forme d'incohérence ne se présente.

Formellement, nous avons modélisé l'incohérence de deux règles de sécurité par la relation binaire *Incoherent*. Dans la spécification de cette relation, nous avons attaché à sa partie prédicative un label ($\langle\langle \textit{disabled rule Def_Incoherent}\rangle\rangle$) auquel correspond un théorème de type règle de réécriture qui sera utilisé dans les preuves d'incohérence de deux règles de sécurité.

Ce théorème stipule qu'une règle de sécurité r_1 est incohérente avec une deuxième règle r_2 , s'ils vérifient un certain nombre de contraintes notés C23, . . . , C28 :

- La contrainte C23 précise que l'incohérence ne se rapporte pas à des règles de contrôle d'accès. Par conséquent, l'ensemble des objets contrôlés par les règles r_1 et r_2 est vide.
- La contrainte C24 précise que les deux règles doivent contrôler des actions en commun.
- La contrainte C25 stipule que les deux règles doivent s'appliquer sur des entités communes.

- Une fois les contraintes C23, . . . , C25 sont satisfaites, il faut vérifier C26 ou C27 :
 - La contrainte C26, spécifie que les deux règles doivent être de type ‘Auth’ ou ‘Oblig’ (côté agent) et ‘Prohb’ (côté système d’agents) et dans ce cas, soit que les deux règles s’appliquent pour un même contexte d’application, soit que le contexte d’application de la règle d’interdiction reflète l’état du système.
 - La contrainte C27, spécifie que les deux règles doivent être de type ‘Prohb’ (côté agent) et ‘Oblig’ (côté système d’agents) et dans ce cas soit que les deux règles s’appliquent pour un même contexte d’application, soit que le contexte d’application de la règle d’interdiction reflète l’état d’exécution de l’agent.

syntax *Incoherent inrel Incoherent*

$\underline{\text{Incoherent}_- : SRule \leftrightarrow SRule}$
$\langle\langle \text{disabled rule Def_Incoherent} \rangle\rangle$
$\forall r_1, r_2 : SRule ; a : MobileAgent ; s : AgentSystem$
$ r_1.Interested = MAga \wedge r_2.Interested = AgSs$
$\bullet r_1 \text{ Incoherent } r_2$
$\Leftrightarrow r_1.Target = r_2.Target = \emptyset \quad [C23]$
$\wedge r_1.Actions \cap r_2.Actions \neq \emptyset \quad [C24]$
$\wedge r_1.RSubject \cap r_2.RSubject \neq \emptyset \quad [C25]$
$\wedge ((r_1.Type = Auth \vee r_1.Type = Oblig) \wedge r_2.Type = Prohb \wedge (r_1.Context = r_2.Context \vee r_2.Context \in s.Estate)) \quad [C26]$
$\vee r_1.Type = Prohb \wedge r_2.Type = Oblig \wedge (r_1.Context = r_2.Context \vee r_1.Context \in a.Estate)) \quad [C27]$

Pour spécifier l’état d’exécution d’un agent mobile (respectivement un système d’agents), nous avons repris la définition du schéma *MobileAgent* (respectivement *AgentSystem*) pour définir un nouveau attribut ‘Estate’, étiqueté C28 (respectivement C29). L’état d’exécution (*Estate*) d’un agent/système d’agents est défini par l’ensemble des conditions d’exécution de ce dernier.

<i>MobileAgent</i>	<i>AgentSystem</i>
<i>Agent</i> <i>Identity_MAg</i> : <i>Propriety</i> <i>CLocation_MAg</i> : <i>Propriety</i> <i>HLocation_MAg</i> : <i>Propriety</i> <i>TTL_MAg</i> : <i>Propriety</i> <i>Itinerary_MAg</i> : seq <i>Propriety</i> <i>Actions</i> : \mathbb{F}_1 <i>Task</i> <i>Mission</i> : $\mathbb{F}(\text{AgentSystem} \leftrightarrow \mathbb{F} \text{Task})$ <i>Estate</i> : $\mathbb{F} \text{Condition}$	<i>Name_AgS</i> : <i>Propriety</i> <i>Location_AgS</i> : <i>Propriety</i> <i>Reserved_res</i> : \mathbb{F}_1 <i>CResource</i> <i>AgentS</i> : \mathbb{F}_1 <i>Agent</i> <i>Services</i> : \mathbb{F} <i>Service</i> <i>Estate</i> : $\mathbb{F} \text{Condition}$
[C28]	[C29]
...	...

4.4.2 Spécification de la politique d'adaptabilité

Pour remédier à ces cas d'incohérence, l'agent doit adapter/ajuster sa politique de sécurité afin de la rendre cohérente avec les exigences de son nouveau système d'exécution. Cette adaptation doit obéir à des règles qui vont gouverner le comportement de l'agent à la détection d'un cas d'incohérence. L'ensemble de ces règles constitue la politique d'adaptabilité de l'agent mobile.

Les règles de type ECA (Événement, Condition, Action) [Bou95] permettent de rendre un système actif (ou réactif), de manière à ce qu'il sera capable de détecter des situations et de réagir sans intervention de l'utilisateur. Par conséquent, la détection d'un événement entraîne l'exécution d'une action lorsqu'une condition est satisfaite. Ce type de règles offre une structure flexible pour exprimer l'évolution d'un système.

Partant de ce fait, nous adoptons les règles ECA pour exprimer l'adaptabilité des politiques des agents mobiles.

Les événements déclencheurs du processus d'adaptabilité des agents mobiles, correspondent aux différents cas d'incohérence identifiés, auparavant, dans la Section 4.4.1. Suite à un événement, l'agent doit évaluer des conditions selon lesquelles il peut déterminer l'action adéquate à entreprendre.

Admettant qu'on peut avoir dans une politique de sécurité des règles rigides, qui sont d'une valeur vitale pour la sécurité de l'agent/système d'agents, et d'autres moins rigides. En plus, les trois différents types d'une règle de sécurité (e.i. autorisation, interdiction et obligation) n'ont pas, généralement, le même degré d'importance quant au bon fonctionnement de l'agent/système d'agents. En effet, il est plus risqué d'omettre une règle de type interdiction/obligation que d'omettre une règle d'autorisation.

Par conséquent, l'agent est en mesure de choisir l'action d'adaptabilité adéquate, en fonction du type de sa règle de sécurité sujette d'incohérence et son niveau de flexibilité.

Nous avons, ainsi, repris la définition du schéma '*SRule*', pour supporter la modélisation du niveau de flexibilité d'une règle de sécurité. Cette propriété sera exprimée par le nouveau attribut '*Flexibility*', étiqueté **C30**.

Soit le type libre *flex*, modélise les deux niveaux de flexibilité d'une règle : *hard* représente le niveau rigide, et *soft* désigne le niveau le moins rigide.

<i>SRule</i>	
<i>Name</i> : <i>Propriety</i>	
<i>Type</i> : <i>SConstruct</i>	
<i>Interested</i> : <i>SEntity</i>	
<i>RSubject</i> : \mathbb{F}_1 <i>SEntity</i>	
<i>Target</i> : \mathbb{F} <i>SObject</i>	<i>flex</i> ::= <i>hard</i> <i>soft</i>
<i>Context</i> : <i>Condition</i>	
<i>Actions</i> : \mathbb{F}_1 <i>Action</i>	
<i>Flexibility</i> : <i>flex</i>	[C30]
...	

En fonction de l'évaluation du type et du niveau de flexibilité de la règle sujette d'incohérence, l'agent peut choisir entre ces trois actions d'adaptabilité :

- Obéit aux exigences du système et désactive, par conséquent, sa règle de sécurité sujette d'incohérence (*Ac1*)
- Ajuste les conditions d'applicabilité de sa règle de sécurité afin d'éliminer la forme d'incohérence (*Ac2*)
- Refuse les contraintes de sécurité et quitte le système (*Ac3*)

Avant d'en choisir une de ces actions, l'agent peut lancer des négociations avec le système d'exécution afin de converger vers une solution qui satisfait au maximum ses besoins de sécurité. Après négociation, l'agent doit opter à une des trois actions d'adaptabilité déjà citées.

Formellement, le type libre *Decision* ::= *Remove* | *Modify* | *Quit* exprime les trois éventuelles décisions, qui correspondent respectivement aux actions notées *Ac1*, *Ac2* et *Ac3*.

Formellement, nous spécifions l'action de négociation par le schéma *negotiate* avec :

- $a_1?$ et $a_2?$ représentent les deux entités impliquées dans le processus de négociation. Il s'agit respectivement d'un agent mobile et de son système d'exécution. Nous vérifions avec la contrainte **C31**, que l'agent négocie avec le système auquel est actuellement localisé.
- $a_3?$ et $a_4?$ se rapportent respectivement aux contraintes de sécurité de l'agent mobile ($a_1?$) et de son système d'exécution ($a_2?$). La contrainte **C32** stipule une incohérence entre les deux règles.
- $d!$ représente la décision prise par l'agent, après négociation.

<i>negociate</i>	
$a_1? : SEntity$	
$a_2? : SEntity$	
$a_3? : SRule$	
$a_4? : SRule$	
$decision! : Decision$	
<hr/>	
$\exists a : MobileAgent ; s : AgentSystem \mid a_1? = MAga \wedge a_2? = AgSs$	[C31]
• $a.CLocation_MAg = s.Location_AgS$	
$a_3?.Interested = a_1?$	
$a_4?.Interested = a_2?$	
$a_3? \text{ Incoherent } a_4?$	[C32]

Le processus de négociation est basé sur un échange d'informations entre l'agent mobile et le système d'exécution afin de converger vers une solution qui concilie leurs intérêts. Un tel mécanisme, doit faire intervenir la génération de propositions, l'évaluation de propositions et l'engagement en cas d'accord. Dans ce travail de thèse nous n'avons pas porté une attention particulière à la modélisation de ce processus de négociation.

Dans ce qui suit, nous allons développer l'action d'adaptabilité adéquate pour chaque paire mettant en jeu le type (*Type*) et la flexibilité (*Flexibility*) d'une règle de sécurité.

– **Règle (côté agent) de type Autorisation :**

- Règle flexible : l'agent obéit aux exigences du système.
- Règle rigide : l'agent procède à une négociation avec le système. Après négociation : soit que l'agent ajuste les conditions d'applicabilité de sa règle de sécurité afin d'éliminer la forme d'incohérence, soit dans le cas pire l'agent refuse les nouvelles restrictions et quitte le système.

– **Règle (côté agent) de type Interdiction/Obligation :**

- Règle flexible : l'agent procède à une négociation avec le système. Après négociation : soit que l'agent obéit aux exigences du système, ou il ajuste les conditions d'applicabilité de sa règle de sécurité afin d'éliminer la forme d'incohérence.
- Règle rigide : de même l'agent procède à une négociation avec le système. Après négociation : soit qu'il accepte de modifier les conditions d'applicabilité de sa règle de sécurité, ou il refuse les nouvelles exigences de sécurité et quitte le système.

Formellement, nous spécifions chaque action d'adaptabilité par un schéma d'opération Z, qui exprime l'évolution de l'état de l'agent (avant et après adaptation) en fonction des valeurs d'entrée de l'action.

Le schéma d'opération *Adapt_Case1* modélise la première forme d'adaptation 'A_{C1}'. En effet, ce schéma stipule les trois cas pour lesquels l'agent décide de désactiver sa règle de sécurité. Après l'exécution de cette opération, la règle $a?$ sera supprimée de la configuration initiale de la politique de l'agent. Bien évidemment, cette suppression est momentanée

jusqu'à la fin de l'exécution de l'agent sur le système en question.

<p><i>Adapt_Case1</i></p> <p>$\Delta SPolicy$</p> <p>$a? : SRule$</p> <p>$b? : SRule$</p> <hr/> <p>$\exists t : MobileAgent ; s : AgentSystem \mid Subject = MAgt \wedge b?.Interested = AgSs$</p> <ul style="list-style-type: none"> • $a? \in Rules \wedge a?.Incoherentb?$ <ul style="list-style-type: none"> $\wedge a?.Type = Auth \wedge a?.Flexibility = soft \wedge b?.Type = Prohb$ $\vee (a?.Type = Prohb \wedge a?.Flexibility = soft \wedge b?.Type = Oblig$ <ul style="list-style-type: none"> $\wedge negotiate[a_1? := Subject, a_2? := b?.Interested,$ <ul style="list-style-type: none"> $a_3? := a?, a_4? := b?, decision! := Remove])$ $\vee (a?.Type = Oblig \wedge a?.Flexibility = soft \wedge b?.Type = Prohb$ <ul style="list-style-type: none"> $\wedge negotiate[a_1? := Subject, a_2? := b?.Interested,$ <ul style="list-style-type: none"> $a_3? := a?, a_4? := b?, decision! := Remove])$ <p>$\Rightarrow Supp[r? := a?]$</p> <p>$Subject' = Subject$</p>
--

Le schéma d'opération *Adapt_Case2* modélise la deuxième forme d'adaptation 'Ac2'. Ce schéma stipule les trois cas pour lesquels l'agent décide, après négociation, de modifier le contexte d'applicabilité de sa règle de sécurité (sujette d'incohérence) afin de parachever son exécution sur le même système.

Dans le cas où la règle du système d'agents ($b?$) est de type interdiction, il faut que le nouveau contexte d'applicabilité de la règle $a?$ soit différent du contexte d'applicabilité de $b?$ et ne reflète pas l'état du système. Cette contrainte est spécifiée par C33. De même, si la règle de l'agent ($a?$) est de type interdiction, il faut que son nouveau contexte d'applicabilité soit différent du contexte d'applicabilité de la règle $b?$ et ne reflète pas l'état d'exécution de l'agent. Cette contrainte est spécifiée par C34.

<i>Adapt_Case2</i>	
$\Delta SPolicy$	
$a? : SRule$	
$b? : SRule$	
$\exists t : MobileAgent ; s : AgentSystem \mid Subject = MAgt \wedge b?.Interested = AgSs$	
<ul style="list-style-type: none"> • $a? \in Rules \wedge a?Incoherentb?$ <ul style="list-style-type: none"> $\wedge (a?.Type = Auth \wedge a?.Flexibility = hard \wedge b?.Type = Prohb$ $\wedge negotiate[a_1? := Subject, a_2? := b?.Interested,$ $a_3? := a?, a_4? := b?, decision! := Modify]$ $\vee a?.Type = Oblig \wedge b?.Type = Prohb$ $\wedge negotiate[a_1? := Subject, a_2? := b?.Interested,$ $a_3? := a?, a_4? := b?, decision! := Modify]$ $\Rightarrow (\exists c : Condition \mid c \neq a?.Context$ $\wedge (c \neq b?.Context \wedge c \notin s.Estate)$ [C33] • $Modify_Context[r? := a?, f? := c])$ $\vee (a?.Type = Prohb \wedge b?.Type = Oblig$ $\wedge negotiate[a_1? := Subject, a_2? := b?.Interested,$ $a_3? := a?, a_4? := b?, decision! := Modify]$ $\Rightarrow (\exists c2 : Condition \mid c2 \neq a?.Context$ $\wedge (c2 \neq b?.Context \wedge c2 \notin t.Estate)$ [C34] • $Modify_Context[r? := a?, f? := c2])$ 	
$Subject' = Subject$	

Lorsque l'agent décide de quitter le système il doit déterminer le nouveau système sur lequel il va s'installer. Formellement, l'action de quitter un système et migrer vers un autre est spécifiée par le schéma d'opération *withdraw_system*. Ce schéma stipule qu'il y aura un changement dans les propriétés de l'agent mobile. Exactement, la contrainte C36 précise que le changement concerne l'attribut '*HLocation_MAg*' qui désigne l'emplacement actuel de l'agent. La contrainte C35 vérifie que la nouvelle destination est différente de l'ancien emplacement de l'agent.

<i>withdraw_system</i>	
$\Delta MobileAgent$	
$l? : Propriety$	
$l? \neq HLocation_MAg$	[C35]
$Identity_MAg' = Identity_MAg$	
$CLocation_MAg' = CLocation_MAg$	
$HLocation_MAg' = l?$	[C36]
$Actions' = Actions$	

Le schéma d'opération *Adapt_Case3* modélise la troisième forme d'adaptation '*A_{C3}*' par

la migration de l'agent vers un nouveau emplacement sans modifier, certainement, sa politique de sécurité. En effet, ce schéma stipule les trois cas, déjà identifiés, pour lesquels l'agent décide de quitter le système.

<i>Adapt_Case3</i>	
$\exists SPolicy$	
$\Delta MobileAgent$	
$a? : SRule$	
$b? : SRule$	
$\exists t : MobileAgent ; s : AgentSystem \mid Subject = MAgt \wedge b?.Interested = AgSs$	
<ul style="list-style-type: none"> • $a? \in Rules \wedge a?Incoherentb?$ <ul style="list-style-type: none"> $\wedge a?.Type = Auth \wedge a?.Flexibility = hard \wedge b?.Type = Prohb$ $\wedge negotiate[a_1? := Subject, a_2? := b?.Interested,$ <ul style="list-style-type: none"> $a_3? := a?, a_4? := b?, decision! := Quit]$ $\vee a?.Type = Oblig \wedge a?.Flexibility = hard \wedge b?.Type = Prohb$ <ul style="list-style-type: none"> $\wedge negotiate[a_1? := Subject, a_2? := b?.Interested,$ <ul style="list-style-type: none"> $a_3? := a?, a_4? := b?, decision! := Quit]$ $\vee a?.Type = Prohb \wedge a?.Flexibility = hard \wedge b?.Type = Oblig$ <ul style="list-style-type: none"> $\wedge negotiate[a_1? := Subject, a_2? := b?.Interested,$ <ul style="list-style-type: none"> $a_3? := a?, a_4? := b?, decision! := Quit]$ 	
$\Rightarrow (\exists h : Propriety \bullet withdraw_system[l? := h])$	

Après la spécification formelle des différentes actions d'adaptabilité, nous spécifions la politique d'adaptabilité d'un agent mobile par le schéma d'opération *APolicy* où :

<i>APolicy</i>	
$\Delta SPolicy$	
$\Delta MobileAgent$	
$p2? : SPolicy$	
$\exists_1 x, y : SRule \mid x \in Rules \wedge y \in p2?.Rules \wedge xIncoherenty$	[Event]
<ul style="list-style-type: none"> • $Adapt_Case1[a? := x, b? := y]$ $\vee Adapt_Case2[a? := x, b? := y]$ $\vee Adapt_Case3[a? := x, b? := y]$ 	[Actions]

- $\Delta SPolicy$: indique que l'adaptation peut engendrer des changements au niveau de la politique de sécurité de l'agent.
- $\Delta MobileAgent$: indique que l'adaptation peut engendrer des changements au niveau des propriétés de l'agent.
- $p2?$: paramètre d'entrée qui désigne la politique (côté système), sujette d'incohérence.
- **Event** : l'événement déclencheur de l'adaptabilité qui désigne l'existence de deux règles incohérentes.
- **Actions** : désigne les trois actions d'adaptabilité possibles. Forcement, leur application dépend des conditions exprimées dans la définition de chacune d'elles.

4.5 Conclusion

Nous avons proposé, dans ce chapitre, un cadre pour la sécurité des systèmes à base d'agents mobiles. Ce cadre traite aussi bien l'aspect statique que l'aspect dynamique. L'aspect statique est lié à la spécification formelle, selon la notation Z , des politiques de sécurité dans les systèmes à base d'agents mobiles. L'aspect dynamique, s'intéresse à définir l'ensemble des opérations élémentaires de reconfiguration d'une politique de sécurité. En plus, nous avons abordé à ce niveau le problème d'adaptabilité des agents mobiles suite à leur migration vers de nouveaux systèmes avec des besoins de sécurité divers. La deuxième contribution dans ce travail, consiste à proposer un cadre de vérification formelle afin de rendre les spécifications proposées plus complètes et plus consistantes.

Lors de la définition d'une politique, il faut avoir le moyen de mesurer son niveau de sécurité en terme d'attaques contrecarrés. Nous proposons dans le chapitre suivant, une approche formelle qui vérifie la prévention des attaques vis-à-vis d'une politique de sécurité.

5

Approche formelle pour la spécification et la vérification des attaques

Le niveau de sécurité, attendu par un système particulier, varie d'une application à une autre. Ceci est exprimé par l'aptitude de prévention contre des attaques.

Les politiques de sécurité, offrent une structure expressive et homogène qui spécifie les différents règlements nécessaires pour contrecarrer plusieurs types d'attaques et protéger, par conséquent, les informations et les ressources d'un système donné.

Pour répondre à l'évolution dynamique des exigences de sécurité dans les systèmes à base d'agents mobiles, un besoin d'adaptation des politiques de sécurité s'impose.

Le cadre de reconfiguration, proposé dans le chapitre précédent, spécifie les opérations de reconfiguration d'une politique de sécurité et définit un ensemble de théorèmes nécessaires pour maintenir la consistance de la politique après reconfiguration. Cependant, nous ne vérifions pas la préservation du même niveau de sécurité après reconfiguration.

Nous proposons, dans ce chapitre, une approche qui permet de vérifier formellement la prévention des attaques vis-à-vis d'une politique de sécurité donnée [LTHK⁺07].

Pour illustrer notre approche, nous étudions le cas de prévention de l'attaque déni de service.

5.1 Taxonomie des attaques

Dans la littérature, nous distinguons plusieurs critères pour classer les attaques qui peuvent se produire dans un système à base d'agents mobiles. Les classifications les plus reconnues sont celles basées sur les effets négatifs des exigences de sécurité présentées dans [BC02, MT06], et celles basées sur la source et la cible d'une attaque [JK99, BR05]. Dans ce qui suit, nous intégrons ces deux critères de classification pour présenter un panorama d'attaques, organisé en trois grandes classes. Étant donné le nombre important de types d'attaques, nous présentons pour chaque classe les attaques les plus courantes.

5.1.1 Attaques contre l'authentification

L'authentification consiste à vérifier l'identité d'une entité informatique, afin de lui autoriser l'accès à un ensemble de ressources et de services. Dans le cas des systèmes à base d'agents mobiles, chacun des agents mobiles malveillants et des systèmes d'accueil hostiles cherchent à faire camoufler leur identité réelle pour avoir plus de privilèges. Le type d'attaque le plus fréquent contre l'authentification est celui de la mascarade (eng. masquerade). Trois cas peuvent se présenter :

1. Agent Mobile **attaque** Système d'agents : un agent mobile malicieux peut prétendre l'identité d'un autre agent autorisé afin d'acquérir des accès à des ressources et à des services, originellement, non assignés. En fait, l'agent déguisé peut :
 - violer la confidentialité et l'intégrité des données du système visité,
 - endommager les ressources du système,
 - nuire la confiance et la réputation de l'agent légitime.
2. Agent Mobile **attaque** Agent Mobile : pour ce cas d'attaque, l'agent malveillant prétend l'identité d'un autre agent afin de tromper l'agent avec qui il communique. En effet, la confidentialité des informations sensibles de l'agent crédule sera violée et la réputation de l'agent légitime se dégrade.
3. Système d'agents **attaque** Agent Mobile : un système malveillant peut se déguiser en tant que système fiable afin d'inciter les agents mobiles de le visiter et d'extraire par la suite leurs informations sensibles. Par conséquent, la réputation du système d'agents légitime sera compromise.

5.1.2 Attaques contre la confidentialité

Les entités malveillantes, pour cette classe d'attaques, tentent d'extraire les informations sensibles sans la permission de leur propriétaire. En fait, il existe, principalement, deux formes d'attaque :

- Accès direct à la mémoire de l'agent,

- Enregistrement et analyse des messages de l'agent.

Dans cette classe, l'espionnage (eng. eavesdropping) est l'attaque la plus fréquente. Le seul cas qui se présente est celui du système d'agents qui espionne l'agent mobile. Il s'agit de contrôler les communications de l'agent afin de révéler ses secrets. Ces informations collectées seront exploitées aux propres intérêts du système malintentionné, sans porter connaissance à l'agent attaqué.

5.1.3 Attaques contre l'intégrité

Cette classe se présente suite à une altération des informations de l'agent mobile ou suite à une modification du contenu de ses communications.

Pour pouvoir s'exécuter, l'agent mobile doit exposer ses propres données au système d'accueil. Ce dernier peut profiter de cette aptitude et lancer des attaques de type altération. En fait, un système malveillant peut modifier les informations de l'agent mobile en exerçant des opérations de suppression, de modification et/ou d'insertion sur son code, ses données ou son état.

5.1.4 Attaques contre la disponibilité

Les attaques contre la disponibilité, cherchent principalement à rendre les services et les ressources autorisés non disponibles. Cette classe est couramment constituée d'attaques de déni de service (eng. Denial of Service : DoS). Dans le cadre des systèmes à base d'agents mobiles, trois formes de déni de service peuvent se présenter :

1. Agent Mobile **attaque** Système d'Agents : dans ce cas, l'objectif de l'agent malveillant consiste à rendre le système d'accueil incapable de répondre aux requêtes des autres agents visiteurs. Il s'agit de consommer d'une façon excessive et illégale ses ressources (mémoire, réseau, bande passante, CPU, etc.) ainsi que ses services. Par conséquent, les performances du système se dégradent.
2. Agent Mobile **attaque** Agent Mobile : un agent mobile peut attaquer un autre agent mobile en lui transmettant intentionnellement des informations incorrectes ou inutiles afin de perturber son comportement et retarder, par conséquent, son exécution par rapport au délai prévu.
3. Système d'Agents **attaque** Agent Mobile : ce type d'attaque se présente principalement, par le fait d'ignorer les requêtes d'un agent autorisé et le placer dans une file d'attente. Cette ignorance, rend l'agent incapable d'achever convenablement sa mission dans les délais prévus. De plus, un système malveillant peut bombarder l'agent par des informations inutiles ou des tâches supplémentaires qui ne sont en aucun rapport avec la mission de l'agent.

D'après cette étude, nous avons noté la quasi absence de travaux qui spécifient et unifient la représentation des différents types d'attaque. Cependant, l'évaluation du niveau de sécurité d'un système en fonction de l'ensemble des attaques combattues, nécessite une compréhension profonde ainsi qu'une représentation explicite de chaque forme d'attaque. En plus, il est indispensable d'utiliser les techniques formelles afin de raisonner rigoureusement pour déterminer le niveau de sécurité d'un système et prouver sa prévention contre ses attaques.

5.2 Approche proposée

Il est largement acquis que la vérification des propriétés à un niveau d'abstraction proche de l'implémentation, est une tâche difficile voir même impossible. De ce fait, nous proposons une approche qui consiste à définir, à un niveau d'abstraction élevé, un ensemble de mesures nécessaires pour déterminer le niveau de sécurité fourni par une politique de sécurité. Cette approche est déclinée en deux étapes :

La première étape, consiste à représenter formellement une bibliothèque d'attaques susceptibles de se présenter dans un système à base d'agents mobiles. Il s'agit de :

- extraire les concepts clés qui caractérisent les différents types d'attaques.
- spécifier formellement, selon la notation Z , chaque type d'attaque d'une manière concise et complète.

Cette spécification doit s'aligner avec les concepts définis pour la modélisation des systèmes à base d'agents mobiles et leurs politiques de sécurité.

Dans une deuxième étape, il s'agit de vérifier formellement l'aptitude d'une politique de sécurité à s'opposer contre un type d'attaque bien déterminé. Ainsi, nous faisons recours aux spécifications des attaques pour définir un ensemble de théorèmes qui précisent les conditions sous lesquelles une politique de sécurité est capable de se prévenir contre un type d'attaque redouté.

Nous illustrons cette approche par la prévention de l'attaque DoS. Une spécification formelle de ce type d'attaque ainsi que le théorème de sa prévention seront détaillées dans la suite de ce chapitre.

5.3 Prévention de l'attaque *DoS*

5.3.1 Spécification formelle de l'attaque *DoS*

Nous avons distingué dans la section 5.1.4 trois formes de déni de service. Le large écart entre ces trois formes nous a amené à spécifier chacune par un schéma Z , à part.

Admettant que toute attaque, dans les systèmes à base d'agents mobiles, est déclenchée par une action bien/mal intentionnée du MAg/AgS. Nous décrivons une action (*Action*) par un nom '*name*', un opérateur '*operator*' qui peut être soit un agent mobile (MAg) soit un système d'agents (AgS), un ensemble de tâches '*tasks*' et éventuellement des entrées '*input*' et des sorties '*output*'. Formellement, nous spécifions une action par le schéma suivant :

Action

name : *Attribute*
operator : *SEntity*
tasks : \mathbb{F} *Task*
input : \mathbb{F} *SObject*
output : \mathbb{F} *Data*

Agent Mobile (*attaque*) Système d'Agents : DoS_AS

Pour cette forme de déni de service, un agent malveillant a tendance à consommer d'une façon excessive les ressources/services du système visité. Pour bien gérer ces ressources/services (*SObject*), il est indispensable d'associer à l'autorisation d'accès à un *SObject* des valeurs limites d'accès. Plusieurs types de contraintes d'accès peuvent exister, par exemple : durée d'accès à la mémoire, nombre de pages à tirer pour une imprimante/jour, etc.

Nous spécifions formellement la relation entre un *SObject* et ses contraintes d'accès (*Constraint*) par la fonction suivante :

$$| \text{own} : \text{SObject} \rightarrow \mathbb{F} \text{Constraint}$$

Pour chaque *Constraint/SObject*, le système doit définir une valeur maximale de consommation, afin de bien gérer leur disponibilité entre les différents agents visiteurs. Cette relation est décrite par la fonction (*maccept_value*) :

$$\left| \begin{array}{l} \text{maccept_value} : \text{AgentSystem} \times \text{MobileAgent} \times \\ \text{SObject} \times \text{Constraint} \rightarrow \text{Value} \\ \hline \forall S : \text{SObject} ; C : \text{Constraint} ; As : \text{AgentSystem} ; Ma : \text{MobileA} \\ | (As, Ma, S, C) \in \text{dom}(\text{maccept_value}) \bullet C \in \text{own}(S) \end{array} \right.$$

Ainsi, la spécification relative à l'attaque *DoS_AS* sera représentée comme suit :

<p><i>DoS_AS</i></p> <p><i>attacker</i> : <i>MobileAgent</i></p> <p><i>target</i> : <i>AgentSystem</i></p> <hr/> <p>$\exists t : Task ; so : SObject ; c : Constraint ; ac : Action ;$ $v : Value ; x : Condition$ $so \in appartient(target)$ $\wedge (target, attacker, so, c) \in \text{dom } maccept_value$ $\wedge execute_value(attacker, so, t) = (c, v)$ $\bullet x = (v, greater, maccept_value(target, attacker, so, c))$ $\quad \wedge so \in ac.input$ $\quad \wedge t \in ac.tasks$ $\quad \wedge ac.operator = MAg(attacker)$</p>

Dans la partie supérieure de ce schéma nous déclarons l'entité attaquante (*attacker*) qui est de type agent mobile et le système victime (cible d'attaque) dénoté par l'attribut (*target*). La partie prédicative stipule que nous ne pouvons parler d'attaque DoS (MAg contre AgS) que si à l'exécution d'une tâche *t* sur l'objet *so*, l'agent opérateur consomme une quantité considérable de *so* qui dépasse la valeur maximale fixée par le système cible ($maccept_value(target, so, c)$). La valeur actuelle de consommation de l'objet *so* est définie par la fonction *execute_value* :

<p>$execute_value : MobileAgent \times SObject \times Task \rightarrow Constraint \times Value$</p> <hr/> <p>$\forall S : SObject ; C : Constraint ; A : MobileAgent ; T : Task ; As : AgentSystem$ $(A, S, T) \in \text{dom}(execute_value)$ $\bullet C \in own(S)$</p>
--

La fonction (*appartient*) détermine l'ensemble des ressources mises à la disposition d'un système d'agents.

<p>$appartient : AgentSystem \rightarrow \mathbb{F} SObject$</p>

Agent Mobile (**attaque**) Agent Mobile : *DoS_AA*

Les agents mobiles sont généralement des agents concurrents dont chacun cherche à répondre, aux mieux, à la requête de son propriétaire. Seuls les agents créés sur la même localisation sont non concurrents.

Les agents mobiles peuvent interagir avec différentes formes d'envoi de messages. En effet, un agent malveillant peut profiter de ce type d'interaction pour dénier les services des autres agents mobiles. Il vise en fait, à saturer l'agent victime par des messages incorrectes ou inutiles afin de perturber son exécution. Ainsi, trois scénarios possibles d'attaque *DoS_AA* peuvent se présenter :

- L'agent malveillant envoie des messages inutiles qui contiennent des informations non demandées par l'agent cible d'attaque.

- L'agent malveillant submerge l'agent victime par l'envoi du même message plusieurs fois ; ce nombre dépasse alors la capacité de réception de l'agent cible. Ainsi nous ajoutons dans la définition de l'agent mobile, la propriété *cap_MAg* qui désigne la capacité maximale de réception du même message.
- L'agent malveillant peut répondre aux requêtes de l'agent sujet d'attaque par des informations non sûres et qui ne font pas partie de sa propre base de connaissances.

Pour modéliser le type d'attaque *DoS_AA*, nous avons défini les trois fonctions suivantes :

- La fonction *send* retourne le contenu des messages envoyés par l'agent mobile vers un autre agent.

$$\mid \text{ send} : \text{MobileAgent} \times \text{MobileAgent} \rightarrow \mathbb{F} \text{Data}$$

- La fonction *request* qui détermine les données demandées par un agent mobile.

$$\mid \text{ request} : \text{MobileAgent} \times \text{MobileAgent} \rightarrow \mathbb{F} \text{Data}$$

- La fonction *act_receive* qui compte le nombre de réceptions du même message.

$$\mid \text{ act_receive} : \text{MobileAgent} \times \text{MobileAgent} \times \text{Data} \rightarrow \mathbb{N}$$

Formellement l'attaque *DoS_AA* est alors spécifiée par le schéma suivant :

<i>DoS_AA</i>	
<i>attacker</i> : <i>MobileAgent</i>	
<i>target</i> : <i>MobileAgent</i>	
<i>attacker</i> ≠ <i>target</i>	[C37]
<i>attacker.CLocation_MAg</i> ≠ <i>target.CLocation_MAg</i>	[C38]
∃ <i>d</i> : <i>Data</i> <i>d</i> ∈ <i>send(attacker, target)</i>	[C39]
• <i>d</i> ∉ <i>request(target, attacker)</i>	
∨ <i>act_receive(target, attacker, d)</i> > <i>target.cap_MAg</i>	

Dans la partie prédicative, nous précisons avec la contrainte C37 que l'attaquant (*attacker*) doit être différent de l'agent victime (*target*). En plus, nous vérifions avec la contrainte C38 qu'un agent n'attaque jamais un autre agent mobile de même localisation source.

Enfin, la contrainte C39, spécifie que l'action qui peut causer un *DoS_AA* peut être soit l'envoi d'une donnée (information) non demandée par l'agent *target* soit l'envoi excessif du même message.

Système d'Agents (attaque) Agent Mobile : DoS_SA

Il existe plusieurs formes d'attaque *DoS_SA*. En fait, un système d'agents peut attribuer à l'agent visiteur de nouvelles tâches à exécuter et qui n'ont aucune relation avec la mission principale de l'agent. De plus, il peut ignorer les demandes sollicitées par

l'agent visiteur. Dans ce cas, l'agent victime va demander l'exécution de la même tâche n fois sans aucune réponse.

Pour se prévenir contre ce type de scénario d'attaque, l'agent mobile doit fixer un nombre limite de demandes d'exécution.

Le schéma Z suivant décrit formellement l'attaque DOS_SA . Il s'agit d'un attaquant de type système d'agents et d'une cible de type agent mobile.

DOS_SA	
$attacker : AgentSystem$	
$target : MobileAgent$	
$\exists t : Task \mid t \in execute(target, attacker)$	[C40]
<ul style="list-style-type: none"> • $(attacker, t) \notin target.Mission$ 	
$\vee (\exists a : Action ; t : Task ; so : SObject ; ans : answer_execute$	[C41]
<ul style="list-style-type: none"> $\mid ans = (target, attacker, t, so) \wedge t \in a.tasks$ • $(\exists r : SRule$ <li style="padding-left: 40px;">$\mid r.Type = Auth \wedge AgS(attacker) = r.Interested$ <li style="padding-left: 40px;">$\wedge MAg(target) \in r.RSubject$ <li style="padding-left: 40px;">$\wedge so \in r.Target \wedge a \in r.Actions$ • $nbr_answer(ans_1) > 3)$ 	
$\vee (\exists a : Action ; t : Task ; so : SObject ; ans : answer_execute$	[C42]
<ul style="list-style-type: none"> $\mid ans = (target, attacker, t, so) \wedge t \in a.tasks$ • $\neg(\exists r : SRule$ <li style="padding-left: 40px;">$\mid r.Type = Prohb \wedge AgS(attacker) = r.Interested$ <li style="padding-left: 40px;">$\wedge MAg(target) \in r.RSubject$ <li style="padding-left: 40px;">$\wedge so \in r.Target \wedge a \in r.Actions$ • $nbr_answer(ans_1) > 3)$ 	

Nous modélisons, avec la contrainte donnée dans la partie prédictive, les deux scénarios possibles de DOS_SA . La première partie de cette contrainte (C40), spécifie que si l'agent visiteur exécute une tâche qui n'appartient pas à sa mission, alors il s'agit d'une tâche attribuée par le système d'exécution actuel et qui désigne une intention d'attaque.

Soit la fonction axiomatique $execute$ qui détermine l'ensemble des tâches exécutées par un agent mobile durant sa visite à un système donné.

$execute : MobileAgent \times AgentSystem \rightarrow \mathbb{F}_1 Task$
<ul style="list-style-type: none"> $\vee m : MobileAgent ; a : AgentSystem \mid (m, a) \in \text{dom } execute$ • $m.HLocation_MAg = a.Name_AgS$

Nous vérifions, dans la partie prédictive de cette fonction, que l'agent est actuellement localisé dans ce système.

Les deux dernières parties (C41 et C42) modélisent une intention d'attaque DOS_SA par

le fait d'ignorer la demande d'exécution de l'agent bien qu'il s'agit d'un agent autorisé. Soit la fonction axiomatique $nbr_request$ qui détermine le nombre d'appels d'exécution de la même tâche.

$$| \quad nbr_request : MobileAgent \times AgentSystem \times Task \times \mathbb{F} SObject \rightarrow \mathbb{N}$$

Dans la spécification de l'attaque DOS_SA, nous avons supposé qu'une demande d'exécution est ignorée si le nombre de ses appels dépasse trois. La possession, d'un agent mobile, d'un droit d'exécuter une tâche bien déterminée est modélisée par l'existence d'une règle de sécurité qui autorise l'exécution de cette tâche ou l'absence d'une règle qui prohibe l'exécution de la tâche par cet agent.

Le pouvoir expressif offert par la notation Z permet d'étudier en profondeur les spécifications produites. Ainsi, nous pouvons prouver aisément la consistance des spécifications fournies et vérifier la préservation des politiques de sécurité contre les différents types d'attaques.

5.3.2 Consistance des spécifications de l'attaque DoS

Vérifier la consistance d'une spécification revient à prouver qu'au moins un état existe [WD96]. Dans Z, si nous traitons un schéma qui décrit l'état du système, il s'agirait alors de fournir un schéma qui l'instancierait.

Nous nous limitons dans cette section à prouver la consistance de la spécification du schéma DoS_AS . Pour ce faire, nous procédons comme suit. Supposons un état initial constitué d'un attaquant A1 et une cible d'attaque S1.

A1 et S1 sont définis comme suit :

$$\begin{array}{l}
 A1 : MobileAgent \\
 S1 : AgentSystem \\
 so1 : SObject \\
 t1 : Task \\
 c1 : Constraint \\
 AA : Action \\
 v1 : Value \\
 x : Condition \\
 \hline
 so1 \in appartientS1 \\
 (S1, A1, so1, c1) \in dom\ maccept_value \\
 execute_value(A1, so1, t1) = (c1, v1) \\
 x = (v1, greater, maccept_value(S1, A1, so1, c1)) \\
 so1 \in AA.input \\
 t1 \in AA.tasks \\
 AA.operator = MAgA1
 \end{array}$$

Un état initial de DoS_AS est alors décrit par le schéma DoS_AS_int :

DoS_AS_int
DoS_AS
$attacker = A1$
$target = S1$

Nous pouvons prouver la consistance du schéma DoS_AS par le théorème d'initialisation suivant :

theorem Consistency_DoS_AS

$\exists DoS_AS \bullet DoS_AS_int$

5.3.3 Preuve formelle pour la prévention contre l'attaque DoS

La dernière étape de notre approche de prévention consiste à proposer, à un haut niveau d'abstraction, un ensemble de théorèmes qui décrivent les conditions sous lesquelles une politique de sécurité est capable de se prévenir contre les différents types d'attaques. L'idée consiste à définir une opération de type *inrel* entre une politique de sécurité et un type d'attaque donné. Nous attachons à la définition de cette relation un théorème de type règle de réécriture. Cette règle sera utilisée dans le processus de preuve.

Nous nous contentons, dans cette section, de présenter le théorème de prévention de l'attaque DoS_AS . En se basant sur la spécification de l'attaque DoS_AS , nous définissons la relation *Prevent* entre une politique de sécurité et l'attaque DoS_AS comme suit :

syntax *Prevent inrel Prevent*

$$\underline{\text{Prevent}} : \text{SPolicy} \leftrightarrow \text{DoS_AS}$$

$$\langle\langle \text{disabled rule Def_Prevent} \rangle\rangle \forall \text{attack} : \text{DoS_AS}; \text{policy} : \text{SPolicy}$$

- $\text{policy Prevent attack} \Leftrightarrow$
 - $(\forall m : \text{MobileAgent}; as : \text{AgentSystem}; t : \text{Task};$
 - $so : \text{SObject}; c : \text{Constraint}; ac : \text{Action};$
 - $V1, V2 : \text{Value}; cc : \text{Condition}$
 - $| \text{attack.attacker} = m \wedge \text{attack.target} = as$
 - $\wedge so \in \text{appartientas} \wedge (as, m, so, c) \in \text{dom maccept_value}$
 - $\wedge \text{execute_value}(m, so, t) = (c, V1)$
 - $\wedge \text{maccept_value}(as, m, so, c) = V2$
 - $\wedge cc = (V1, \text{greater}, V2)$
 - $\wedge so \in ac.\text{input} \wedge t \in ac.\text{tasks} \wedge ac.\text{operator} = \text{MAgm}$
 - $(\exists r : \text{SRule} \mid r \in \text{policy.Rules}$
 - $r.Type = \text{Prohb}$
 - $\wedge \text{MAgm} \in r.RSubject$
 - $\wedge \text{AgSas} = r.Interested$
 - $\wedge so \in r.Target$
 - $\wedge r.Actions = \{ac\}$
 - $\wedge r.Context = (V1, \text{greater}, V2))$

Nous attachons à la partie prédicative de la relation *Prevent* un label ($\langle\langle \text{disabled rule Def_Prevent} \rangle\rangle$) auquel correspond un théorème qui sera utilisé pour prouver la prévention d'une politique de sécurité contre un *DoS_AS*.

La règle de réécriture *disabled rule Def_Prevent*, montre qu'une politique de sécurité est capable de se prévenir contre l'attaque *DoS_AS* si elle dispose d'une règle de sécurité qui interdit les actions qui peuvent causer une consommation excessive des ressources. En effet, l'expression $r.Context = (V1, \text{greater}, V2)$ signifie que l'agent attaquant, en exécutant l'action *ac*, va consommer la ressource *so* d'une valeur *V1* qui dépasse celle autorisée par le système *as*. Dans ce cas, l'exécution de l'action *ac* devra être prohibée.

5.4 Conclusion

Dans ce chapitre nous avons proposé une approche formelle pour la prévention des attaques dans un système à base d'agents mobiles. L'idée consiste à spécifier formellement une bibliothèque des attaques susceptibles de se présenter dans un système à base d'agents mobiles. Sur cette base un ensemble de théorèmes seront défini afin de décrire les actions à entreprendre pour éviter ces attaques.

Cette approche est une solution prometteuse pour déterminer le niveau de sécurité offert par une politique donnée. De plus, cette solution jouera un rôle vital pour vérifier la préservation du niveau de sécurité après une opération de reconfiguration.

Nous avons illustré cette approche par l'exemple de prévention de l'attaque DoS dans le cadre des systèmes à base d'agents mobiles.

Les résultats théoriques auxquels nous sommes parvenus sont sujets d'un raffinement successif en vue d'une implémentation des politiques de sécurité relatives à des types d'attaques redoutés.

Nous proposons dans le chapitre suivant un cadre opérationnel de déploiement des politiques de sécurité.

6

Déploiement des politiques de sécurité

Comme dernière contribution, dans ce travail de thèse, nous visons de définir un cadre opérationnel pour le déploiement des politiques de sécurité dans les systèmes à base d'agents mobiles.

Pour déployer les politiques d'une manière fiable, il est nécessaire d'utiliser les techniques formelles qui permettent un raisonnement rigoureux quant à la sécurité des systèmes complexes [KCM⁺09]. Pour ce faire, nous profitons du cadre théorique que nous avons déjà défini, et nous appliquons une approche pour générer automatiquement à partir d'une spécification fiable des politiques de sécurité, le code correspondant.

Un générateur de code nécessite en entrée une spécification qui comporte les données à implémenter, et en plus un ensemble de règles selon lesquelles seront traduites ces données pour obtenir le code approprié. Ces règles seront définies en fonction du paradigme utilisé et du langage de programmation choisi. Dans le cadre de ce travail, nous nous intéressons à la programmation orientée aspect (POA) de ce qu'elle offre de modularité et de séparation entre les préoccupations fonctionnelles et les préoccupations techniques d'une application. De ce fait, nous commençons par présenter les principes et les fondements de la POA afin de montrer son utilisation avantageuse pour l'implémentation des propriétés de sécurité. Par la suite, nous proposons l'utilisation de ce paradigme dans une approche d'imposition des politiques de sécurité. Cette approche sera implémentée, ensuite, selon la syntaxe d'un tisseur d'aspect dynamique. Nous achevons ce chapitre par une expérimentation sur un cas d'étude.

6.1 Programmation Orientée Aspect

La POA est un paradigme qui a été défini au centre de recherche de Xerox à Palo Alto au milieu des années 1990 [KLM⁺97]. En fait, ce nouveau paradigme ne remet pas en cause les autres paradigmes de programmation, comme l'approche procédurale ou l'approche objet, mais les étends en offrant des mécanismes complémentaires afin de faciliter la réutilisation et la maintenance des logiciels.

D'une manière générale, pour appréhender la complexité d'une application et mieux assurer sa maintenance, il est nécessaire de décomposer le problème en petits sous-problèmes qui peuvent être traités et résolus séparément. Ensuite, les solutions partielles peuvent être combinées pour avoir une solution complète du problème initial [Mar02].

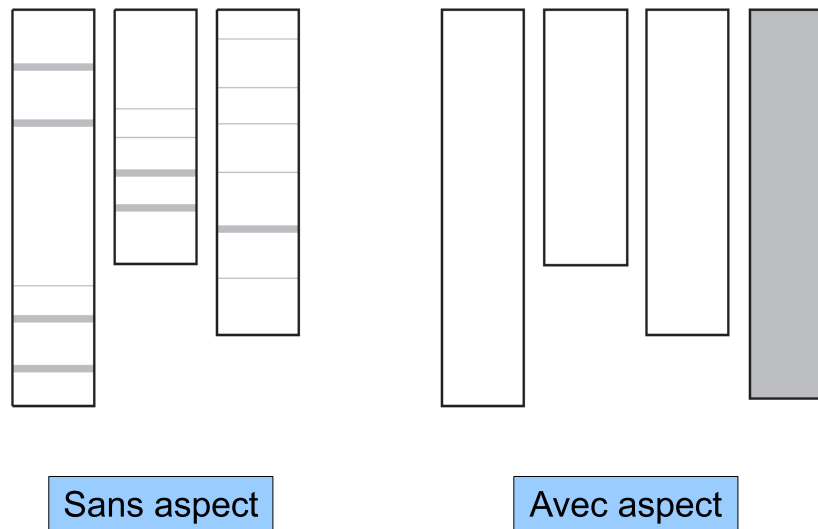


FIG. 6.1 – Impact des aspects sur le code global d'une application

Sur la base de cette hypothèse plusieurs travaux de recherche ont été focalisés sur l'identification des différentes préoccupations d'une application telles que par exemple les préoccupations liées à : la synchronisation, la tolérance aux pannes, la persistance, la sécurité (contrôle d'accès, authentification, ...), etc. Une classification plus abstraite distingue deux types de préoccupations [Mar02] : les préoccupations fonctionnelles (métiers) qui correspondent au coeur du métier de l'application et les préoccupations non-fonctionnelles (techniques) qui correspondent souvent aux exigences non-fonctionnelles du système. La séparation entre ces deux types de préoccupations transverses permet de les rendre indépendantes et par conséquent améliore la modularité des applications [KLM⁺97].

La POA est fondée sur ce principe de séparation de préoccupations transverses. En effet, en POA une application est formée de *classes* et d'*aspects*. Le code transverse lié aux préoccupations non fonctionnelles est modularisé sous forme d'aspects. Ces aspects

seront, par la suite tissés au code fonctionnel afin de former une application complète. La Figure 6.1 tirée de [RP05] illustre bien l'impact des aspects sur le code global d'une application.

La partie gauche de la figure représente une application composée de trois classes dont les lignes horizontales représentent une partie du code qui correspond à une fonctionnalité transverse, telle que la sécurité. Le code relatif à cette fonctionnalité est dispersé sur les trois classes de l'application. En appliquant les fondements de la POA, nous obtenons pour la même application une nouvelle répartition formée de trois classes et un aspect (le rectangle représenté dans l'extrémité droite). En fait, cet aspect encapsule tout le code relatif à la dite fonctionnalité transverse.

6.1.1 Terminologie de la POA

Dans cette section nous présentons les concepts de base de la POA. À coup sûr nous commençons par la définition du concept d'*aspect*, ensuite nous abordons les notions connexes de coupe, de point de jonction, de code advice, et de tissage. Ces concepts seront présentés indépendamment de tout langage d'implémentation.

6.1.1.1 Aspect

En programmation orienté objet (POO), le code d'une méthode est localisée dans une classe, alors que les appels de cette dernière se retrouvent dispersés dans différentes classes. Cette dispersion rend la maintenance et l'évolution du code difficile.

La POA apporte une solution à ce problème de dispersion et propose une deuxième unité de décomposition : l'*aspect*. Ainsi, en POA le programme est décomposé en aspects et classes. Les classes implémentent le code métier d'une application et les aspects implémentent les fonctionnalités techniques qui s'avèrent dispersées dans le code de l'application. Un aspect est constitué de deux parties : la coupe et le code advice.

6.1.1.2 Coupe

Une coupe (eng. pointcut) permet de spécifier l'endroit(s) où elle va s'appliquer la fonctionnalité transverse de l'aspect associé. Ceci est définie par un ou plusieurs points de jonction (eng. joinpoints). Chaque point de jonction représente un point, dans le flot d'exécution d'un programme, au niveau duquel sera greffé le code d'une fonctionnalité transverse (eng. code advice).

Il existe différents types de point de jonction : il peut être l'appel d'une méthode (call), l'exécution d'une méthode (execute), la lecture/l'écriture d'une variable (set/get), etc.

La coupe est une expression logique construite avec les opérateurs *AND*, *OR* et *NOT* pour

relier les différents points de jonction qui le composent.

Une coupe peut concerner un ou plusieurs aspects à la fois et un aspect peut être associé à plusieurs coupes. En effet, un point de jonction peut appartenir à plusieurs coupes d'un même aspect ou d'aspects différents.

6.1.1.3 Code advice

Un code advice est un fragment de code qui sera inséré au niveau des points de jonction. Il désigne le comportement d'un aspect, c-à-d la manière selon laquelle une fonctionnalité transversale va être intégrée.

Un aspect peut comporter un ou plusieurs codes advices. Chaque advice associé à une coupe, désigne un comportement particulier de son aspect. Il existe, principalement, trois types de code advice :

- *before* : le code sera exécuté avant la coupe.
- *after* : le code sera exécuté après la coupe.
- *around* : une partie du code sera exécuté avant la coupe et une autre sera exécuté après. En fait, il existe dans les langages de la POA une instruction nommée *proceed* qui permet de revenir à l'exécution du programme principale et exécuter les points de jonction. En effet, la partie du code qui se trouve avant l'instruction *proceed* va être exécutée avant le point de jonction et l'autre sera exécutée juste après.

6.1.2 Tissage d'aspects

Pour obtenir une application étendue qui intègre, à la fois, le code métier (décrit par des classes) avec les fonctionnalités techniques (décrites par des aspects), il faut utiliser un outil de composition des aspects et des classes. Cet outil s'appelle *tisseur d'aspect* (eng. aspect weaver).

Le processus d'intégration des aspects dans le code métier s'appelle le *tissage* (eng. weaving). Ce processus peut être effectué à la compilation ou à l'exécution de l'application. Par conséquent, il existe deux types de tisseurs d'aspects : les *tisseurs statiques* qui effectuent l'opération de tissage avant de commencer l'exécution de l'application et les *tisseurs dynamiques* qui effectuent l'opération de tissage au cours de l'exécution. Un autre type de tisseur peut être découlé de cette classification ; ce sont les tisseurs polyvalents qui peuvent assurer le tissage dynamique et statique.

6.1.2.1 Tissage statique

Un tisseur statique est un outil qui effectue l'opération de tissage avant de commencer l'exécution du programme. Ce tisseur prend en entrée un ensemble de classes et un ensemble d'aspects pour fournir en sortie une application complète qui satisfait un certain

nombre d'exigences techniques.

Les classes fournies en entrée peuvent être soit du code source et dans ce cas il s'agit d'un tissage au moment de la compilation (*compile-time weaving*) soit du code intermédiaire et dans ce cas il s'agit d'un tisseur de bytecode (*Bytecode Weaver*). Le deuxième type de tissage est plus intéressant vu qu'il accepte de tisser les aspects avec des applications dont le code source n'est pas disponible. En plus, l'analyse d'un code intermédiaire est moins complexe que l'analyse d'un code source qui peut être muni d'un certain nombre d'erreurs syntaxiques.

6.1.2.2 Tissage dynamique

Le tissage dynamique se fait au moment de l'exécution du code (eng. at run-time). En effet, avec ce type de tissage les aspects et les classes de l'application doivent être préalablement compilés.

L'avantage majeur d'une telle méthode de tissage, est que les aspects et les classes de l'application peuvent être manipulés d'une manière indépendante les uns des autres. Cela veut dire que les aspects et les classes sont tissés sans pour autant interdire la possibilité d'interagir individuellement sur les aspects. Cette propriété permet ainsi d'assurer l'adaptabilité (par activation ou désactivation) des aspects déjà existants en fonction des besoins de l'application ou du programmeur. De ce fait, l'utilisation d'un tisseur dynamique est un choix naturel pour l'implémentation des systèmes adaptables qui exigent une forte contrainte de disponibilité [GS05].

Il est à noter que les tisseurs dynamiques actuels ne sont pas en mesure de répondre aux différentes questions de reconfiguration de l'application en particulier la modification des aspects au moment de l'exécution de l'application [GS05, GB06]. Différents travaux proposent des solutions à ces questions, par l'utilisation d'autres technologies en plus des tisseurs dynamiques, à savoir les intergiciels (eng. middleware) et les plateformes orientée service, le principe de réflexion [Mae87, TH96], la technologie des frames [Bas96], etc. Ces technologies ne sont pas encore satisfaisantes car elles ont apporté des failles au niveau du code et elles ont alourdi l'exécution de l'application. En plus, leur consommation excessive de la mémoire, peut dégrader les performances du système.

6.2 RDyMASS : Approche pour l'imposition des politiques de sécurité

Les approches classiques de mise en place d'un système de sécurité, consistent à revenir sur le code de l'application pour disséminer manuellement le code correspondant aux propriétés de sécurité. L'adoption d'une telle approche représente une tâche lourde pour le programmeur et surtout dans un contexte dynamique qui exige continuellement des changements dans la définition de la politique de sécurité. En plus, ce principe d'intégration manuelle ne garantit pas une conformité avec la spécification proposée ; Le code qui implémente les propriétés de sécurité peut contenir des inconsistances qui ne se présentent pas dans la spécification formelle. Ces limites sont accentuées par l'absence de modularité du programme.

La POA surmonte les limites des approches classiques de programmation, et assure une meilleure qualité du code, réutilisation et maintenance des applications à travers sa nouvelle dimension de modularisation nommée *Aspect*.

Pour aborder convenablement le problème de la mise en place d'un système de sécurité dans les applications à base d'agents mobiles, il est nécessaire d'utiliser un mécanisme d'imposition de politiques de sécurité qui maintient un haut niveau de disponibilité de ces applications, et qui respecte l'aspect dynamique de leurs besoins de sécurité.

Le mécanisme de monitoring de l'exécution basé sur la réécriture (présenté dans la section 2.1.4) remplit ses exigences, en plus son principe de fonctionnement s'aligne avec les fondements de la POA dynamique [RP05], qui permet de tisser et dé-tisser les aspects au moment de l'exécution de l'application.

Ainsi, nous proposons d'intégrer le paradigme de la POA dans un cadre opérationnel pour l'imposition dynamique des politiques de sécurité, qui opère selon le mécanisme de monitoring de l'exécution basé sur la réécriture.

Pour tirer profit des méthodes formelles qui permettent un raisonnement rigoureux quant à la sécurité et nous garantissent une consistance au niveau des politiques, nous nous basons dans la définition du cadre opérationnel d'imposition des politiques de sécurité sur le cadre théorique présenté dans le chapitre 4. En fait, nous proposons une approche [AMKHK10] qui génère un code aspect qui correspond à une spécification fiable et prouvée de politiques de sécurité.

Notre approche, baptisée RDyMASS (*Reliable and Dynamic Mobile Agent System's Security*) [AMKHK10], consiste en trois étapes schématisées par la Figure 6.2. Nous détaillons dans ce qui suit chacune de ses étapes et les relations entre elles.

6.2.1 Étape 1 : Définition d'un template d'aspects de sécurité

Pour réduire progressivement l'écart entre la représentation formelle d'une politique de sécurité et son équivalent en terme d'aspects, nous proposons la définition d'un template

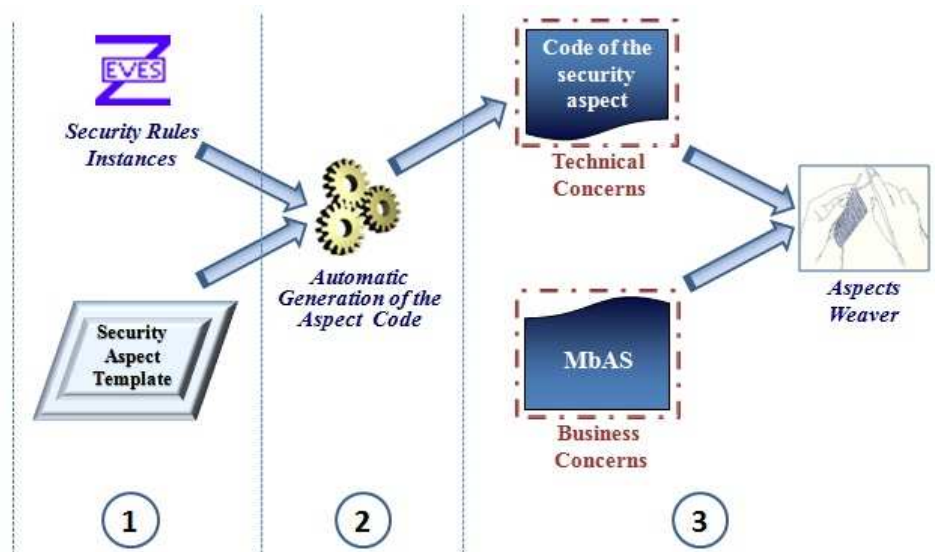


FIG. 6.2 – Aperçu générale sur l'approche RDyMASS

pour les aspects de sécurité. Nous abordons dans la définition de ce template, deux niveaux d'abstraction :

- Le premier niveau est générique. Il présente à un haut niveau d'abstraction l'ossature d'un aspect de sécurité indépendamment de la syntaxe imposée par un tisseur d'aspects. Ce modèle sera défini en concordance avec la spécification proposée pour une règle de sécurité.
- Le deuxième niveau est spécifique. Il se base sur le modèle de définition des aspects défini au niveau générique, et la terminologie adoptée par un tisseur spécifique. Ainsi, nous obtenons à ce niveau une représentation des aspects de sécurité proche du code à générer.

Avant de choisir le tisseur d'aspect qui sera utilisé dans l'étape d'expérimentation, nous allons présenter, dans cette sous-section, uniquement le *template d'aspect générique*. Dans la section 6.4.1, nous développerons le template d'aspect spécifique à la syntaxe du tisseur choisi.

L'imposition d'une politique de sécurité revient à imposer l'ensemble de ses règles de sécurité. Un même aspect peut encapsuler le code correspondant à une ou plusieurs règles de sécurité. Afin de simplifier la structure d'un aspect, pour assurer une meilleure adaptabilité, et agir séparément sur les aspects et par conséquent sur les règles de sécurité, nous avons choisi de définir un seul aspect par instance de règle de sécurité.

Comme c'est présenté dans le listing 6.1, le template d'aspect générique est composé de deux principales parties : la coupe et le code advice.

La coupe de l'aspect générée (lignes 1-2) intercepte l'exécution de l'agent mobile c-à-d ses méthodes qui correspondent à l'attribut *Actions* pour une instance de règle de sécurité.

La coupe est alors formée d'un ou de plusieurs points de jonction, dont chacun est composé de l'appel d'une ou de plusieurs méthodes correspondantes à l'attribut *Actions* de la classe désignant le *RSubject*.

```

1 pointcut pointcut_name = execution(public * RSubject's class ->
2 the corresponding method of the action attribute of the RSubject's class)
3
4 [P1] -----
5 //Checking the constraints presented in the Context signature
6 if(verifConstraint-1) {...}
7 ...
8 if(verifConstraint-n) {...}
9
10 [P2] -----
11 if((All constraints are verified && Type of SRule is ``Prohb")
12    || (There is one constraint not verified && Type of SRule is ``Auth"))
13    // prohibit the execution of the corresponding method
14    // launch of an exception
15
16 [P3] -----
17 else if((There is one constraint not verified && Type of SRule is ``Prohb")
18    || (All constraints are verified && Type of SRule is ``Auth"))
19    // implementation of the method that triggered the safety rule
20    proceed(parameter)
21    // update of the system

```

Listing 6.1 – Template Générique d'Aspect de Sécurité

La deuxième partie concerne le code advice de l'aspect (lignes 5-21). Ce code vérifie les contraintes spécifiées dans une instance de règle de sécurité. Si les contraintes sont bien vérifiées, l'aspect exécute l'action de l'agent mobile, et puis il met à jour l'état du système. Sinon, l'aspect interdit l'exécution de l'action considérée.

Dans notre template, nous utilisons l'advice de type *around*. Nous décomposons un advice en trois sous-parties. La première partie étiquetée [P1] permet de vérifier les contraintes définies dans l'attribut *Context* d'une instance de règle de sécurité. Deux cas peuvent se présenter.

Il aura une transition vers la partie étiquetée [P2], si les contraintes dans partie [P1] sont vérifiées et la règle de sécurité est de type interdiction, ou lorsque il existe au moins une contrainte non vérifiée et la règle de sécurité est de type autorisation (i.e. l'agent mobile n'est pas autorisé à exécuter l'action désirée). Dans ce cas, la demande est refusée et une exception sera levée.

Dans le deuxième cas, il y aura une transition vers la partie étiquetée [P3]. Cette transition est réalisée lorsque toutes les contraintes sont vérifiées et le type de la règle de sécurité est autorisation, ou lorsque il existe au moins une contrainte non vérifiée et le type de la règle de sécurité est interdiction. Dans ce cas, la demande de l'agent sera acceptée et la méthode qui a déclenché la règle de sécurité sera exécutée. Par la suite, une éventuelle mise à jour de l'état global du système aura lieu.

6.2.2 Étape 2 : Génération des aspects de sécurité

La seconde étape de notre approche consiste à la génération automatique du code d'aspects de sécurité. Cette génération se fait en fonction de la syntaxe du tisseur adopté. Contrairement au template d'aspect générique, qui définit la structure de l'aspect indépendamment du langage de programmation, le template du niveau spécifique raffine cette structure en fonction du langage de programmation adopté (i.e. le tisseur d'aspects utilisé). En effet, le template d'aspect spécifique sera utilisé par le programmeur pour implémenter le code correspondant du générateur. Ce générateur applique la structure proposée par le template spécifique d'aspect et prend en entrée des spécifications d'instances de règles de sécurité afin de générer le code aspect correspondant. Ces spécifications sont importées de Z/EVES après avoir prouvé leur consistance.

6.2.3 Étape 3 : Tissage d'aspects

Une fois le code aspect est généré, il faut faire tisser cet aspect dans le code fonctionnel qui correspond à une application à base d'agents mobiles non sécurisée. Le code fonctionnel de l'application doit être implémenté en concordance avec les concepts évoqués dans le cadre formel pour la modélisation des MbAS (Section 3.2). Dans notre approche, nous utiliserons le tissage dynamique afin de supporter le changement dynamique des besoins de sécurité d'une application par l'activation et la désactivation des aspects de sécurité.

6.3 Choix d'implémentation de RDyMASS

Dans la section précédente (section 6.2), nous avons présenté l'approche RDyMASS indépendamment de tout choix du langage de programmation. En effet, RDyMASS peut être instantiée sur différents contextes pratiques relatifs au tisseur d'aspect adopté. En plus, RDyMASS peut s'appliquer pour imposer les politiques de sécurité dans divers plateformes d'agent mobile.

Dans ce qui suit, nous allons développer nos choix d'implémentation relatifs au tisseur d'aspect et à la plateforme de développement des agents mobiles.

6.3.1 Choix du tisseur d'aspect

Un tisseur d'aspect est un outil qui permet de mettre en œuvre les concepts fondamentaux de la POA (section 6.1.1) dans un langage de programmation. En effet, un tisseur d'aspect consiste à intégrer en une seule application des classes et des aspects, qui implémentent respectivement les fonctionnalités métiers et les fonctionnalités techniques d'une application. Dès lors, plusieurs outils de la POA ont été proposés pour différents langages de

programmation tels que Java, C#, C++, Smalltalk, etc. Nous citons à titre d'exemple les tisseurs AspectJ, JAC, JBoss AOP, AspectC++, Eos, AspectS.

Dans notre travail, nous nous intéressons aux tisseurs dynamiques développés autour de Java, étant donné que la majorité des environnements de développement des applications à base d'agents mobiles (i.e. les fonctionnalités métiers) utilisent Java.

Dans la littérature, plusieurs travaux s'intéressent à comparer les tisseurs d'aspect, selon différents critères. Évidemment chaque tisseur possède des avantages (qualités) et des inconvénients (défauts). Principalement, nous nous sommes basés sur les critères suivants, en fonction desquels nous avons choisi l'outil JBoss AOP pour implémenter notre approche RDyMASS.

1. Puissance d'expression du langage du tisseur : un bon tisseur doit disposer d'une bibliothèque riche en fonctionnalités et supporter différentes formes de tissage du code (exemple : le tissage avant/après/autour l'appel d'une méthode, l'exécution d'une méthode, etc).
2. Langage de développement des aspects : il est recommandé aux programmeurs d'utiliser un tisseur sans apprendre un nouveau langage de programmation et sans acquérir de nouveaux outils de programmation.
3. Niveau d'adhérence entre les aspects et le tisseur utilisé : il est très bénéfique d'avoir des aspects portables qui peuvent être tissés par différents outils de POA. C'est le cas du cartel de l'*AOP Alliance* qui a pour but de simplifier la réutilisation des aspects en disposant d'une API commune.
4. Niveau d'intégration du tisseur dans un environnement de développement : un tisseur d'aspect peut être utilisé soit d'une manière autonome soit en dépendance avec un serveur d'application.
5. Importance de l'utilisation d'un tisseur polyvalent : un tisseur polyvalent assure statiquement et dynamiquement le tissage des aspects. Par conséquent il permet d'optimiser le temps global de tissage des aspects en fonction de leur type.

JBoss AOP est un framework pour la programmation orientée aspect [RP05, Kha06]. Il a été réalisé par une équipe dirigée par *Bill Bruke* puis ses droits ont été achetés en 2004 par la société RedHat.

JBoss AOP est un tisseur Open Source, distribué gratuitement et membre de l'*AOP Alliance*. Il peut être soit utilisé dans sa version autonome (stand-alone) en se servant d'un plug-in, soit étroitement intégré au serveur d'application JBoss Application Server (JBoss AS). En plus, JBoss AOP possède un lanceur orienté aspect séparé sous Eclipse qui permet de simplifier la tâche du programmeur par l'utilisation d'une interface graphique.

JBoss AOP dispose d'un ensemble d'aspects pré-définis, cependant certains d'entre eux ne sont pas exploitables en dehors de JBoss AS.

Les aspects avec JBoss AOP, sont écrits en Java pur sans extension syntaxique mais nécessite l'utilisation d'une API propre à JBoss AOP.

Un aspect se compose obligatoirement de deux fichiers : le premier contient le code d'un advice, appelé intercepteur (interceptor). Le deuxième est un fichier *.xml*. Ce fichier permet de configurer les coupes de l'application et d'assurer le *HotDeployment* des aspects

au cours de l'exécution. L'utilisation du langage de balisage XML est très bénéfique puisqu'il permet à son code d'être interprété par d'autres langages de programmation. En plus, l'utilisation du fichier de configuration XML rend le tissage avec cet outil très puissant pour une durée de temps très courte grâce à l'exploitation des balises des coupes ou du mécanisme d'annotation du code Java.

Admettant qu'on peut avoir dans une politique de sécurité des règles rigides, qui sont d'une valeur vitale pour la sécurité de l'agent/système d'agents et elles ne doivent jamais être retirées, et d'autres moins rigides. Les règles de type rigide doivent être intégrées avec un tisseur statique pour éviter le risque de leur désactivation. Quant autres règles il est nécessaire d'utiliser un tisseur dynamique. JBoss AOP répond bien à cette exigence du fait qu'il assure statiquement et dynamiquement le tissage des aspects.

Nous explicitons dans ce qui suit la syntaxe et la terminologie de JBoss AOP. Nous appliquerons cette syntaxe dans la définition du template d'aspect de sécurité spécifique à JBoss AOP et la mise en place du générateur d'aspects.

6.3.1.1 Terminologie et Syntaxe de JBoss AOP

Un aspect écrit avec JBoss AOP [RP05] est composé principalement de deux fichiers :

- Le premier est un fichier de configuration XML qui définit une coupe. Les coupes d'une même application seront définies dans un seul fichier XML. Le nom de ce fichier est obligatoirement *jboss-aop.xml*.

La balise principale de ce fichier est `< aop >`. Les définitions de coupe sont comprises entre les balises `< aop >` et `< /aop >`. Il est possible d'utiliser, selon le besoin, d'autres sous balises pour la définition des coupes, à savoir les balises *bind* et *prepare*.

- Le second est un fichier Java qui fournit le code *Advice* associé à la coupe. Il est appelé dans la terminologie de JBoss AOP un intercepteur plutôt que de code advice. Le code d'un intercepteur est fourni dans une classe qui doit implémenter l'interface *org.jboss.aop.Interceptor*. Cette interface définit deux méthodes, *getName*, qui retourne le nom de l'intercepteur, et *invoke*, qui doit fournir le code avant/après.

Un intercepteur possède la structure suivante :

```

1 import org.jboss.aop.advice.Interceptor;
2 import org.jboss.aop.joinpoint.Invocation;
3 public class Interceptor\_Name implements Interceptor
4 {
5     public String getName()
6     {
7         return ``Interceptor\_Name";
8     }
9     public Object invoke(Invocation invocation) throws Throwable
10    {
11        //- - - - Partie before de l'intercepteur - - - -
12        //***** Invocation des méthodes du pointcut*****
13        Object rsp = invocation.invokeNext();
14        //- - - - - Partie after de l'intercepteur - - - -
15    }
16 }
```

La méthode *invoke* contient deux parties principales qui sont séparées par la ligne de code (ligne 13) permettant l’invocation de la méthode qui déclenche l’intercepteur. La première partie de la méthode *invoke* concerne les traitements à réaliser avant l’exécution de la méthode précisée dans le point de jonction. Elle peut contenir aussi des conditions pour décider de l’exécution ou non de la méthode.

La deuxième partie (partie *after*) contient les instructions pour la mise à jour de l’état globale du système. Ces deux parties sont séparées par l’invocation de la méthode liée à l’intercepteur.

JBoss AOP : tissage statique

La balise `< bind >` est utilisée pour le tissage des aspects au moment de la compilation. Elle permet de lier l’*advice* d’un intercepteur et le point de jonction. Elle a la forme suivante :

```
1 <bind pointcut="execution(join_points)">
2 <interceptor class="Interceptor_Name"/>
3 </bind>
```

JBoss AOP dans sa version statique est fourni avec un plugin Eclipse pour gérer les intercepteurs d’une application à travers une interface graphique. Cette intégration nous permet non seulement de simplifier le démarrage de projets contenant des aspects, mais elle va jusqu’à nous montrer en temps réel sur quelles zones sont greffés les aspects du projet, ceci à travers un “Aspect Manager” qui offre une représentation graphique du fichier ‘jboss-aop.xml’.

JBoss AOP : tissage dynamique

Pour utiliser JBoss AOP dans sa version dynamique, il faut commencer par la configuration des arguments de la MVJ pour supporter le HotSwapping (l’échange à chaud). L’activation du HotSwap permet d’instrumenter le bytecode des intercepteurs durant l’exécution.

Une fois la MVJ est configurée pour l’échange à chaud (eng. HotSwapping), il est possible d’utiliser la technique du déploiement à chaud (eng. HotDeployment) qui permet de modifier, au cours d’exécution, les relations (eng. bindings) entre une coupe et les intercepteurs associés. En fait, il est possible de retirer des relations ou de déployer d’autres relations avec des intercepteurs qui sont déjà chargés dans la MVJ.

La balise `< prepare >` permet d’assurer le déploiement à chaud des aspects au cours de l’exécution de l’application. L’instrumentation avec *prepare* peut se faire à travers un code XML (`< prepareexpr = “execution(join_points)” / >`) ou bien l’annotation (`@Prepare`). Cette dernière nécessite l’importation de la bibliothèque *org.jboss.aop.Prepare*.

JBoss AOP propose une API pour l'ajout/suppression des relations (eng. bindings) entre les advices au cours de l'exécution.

6.3.2 Choix de la plateforme de développement des agents mobiles

Au niveau de la troisième étape de l'approche RDyMASS, nous devons adopter une plateforme de déploiement d'agents mobiles pour implémenter les préoccupations fonctionnelles d'une application non sécurisée. Bien évidemment, cette plateforme doit avoir un minimum de concordance avec les concepts évoqués dans notre cadre de spécification des systèmes à base d'agents mobiles, défini dans la section 3.2. En plus, cette plateforme doit répondre à certain nombre de critères à savoir :

- La disponibilité, la popularité, la bonne documentation, et la simplicité d'utilisation.
- Les standards supportés : comme CORBA, FIPA, MASIF et XML permettent une implémentation plus aisée du système cible.
- Les standards de la technologie agent : comme KQML, KIF et ACL peuvent être très utiles dans le développement.
- Le langage de développement des agents : le langage utilisé affecte la méthode d'implémentation et les utilisations futures.
- Les protocoles de communication : qui peuvent être utilisés par le système pour la communication entre les plateformes comme RMI, CORBA ou JINI.

Le niveau de sécurité, n'a pas été pris comme un point déterminant parmi les critères de choix.

En fonction de cette liste de critères, nous avons choisi l'adoption de la plateforme Aglets [LM98, DBL97].

6.3.2.1 plateforme Aglets

Aglets Software Development Kit (ASDK) est un environnement pour la programmation des agents mobiles en Java. Ces derniers sont des objets Java qui peuvent se déplacer d'un hôte à l'autre dans les réseaux d'Internet en utilisant le protocole ATP (Agent Transfer Protocol).

Aglets, présente plusieurs avantages par rapport à d'autres plateformes. D'une part sa simplicité d'utilisation et sa légèreté. En plus, Aglets est disponible gratuitement en open source. Elle est bien documentée, implémente le standard MASIF, dispose d'une interface graphique qui facilite la gestion des agents, etc. D'autre part, nous retrouvons plusieurs concepts en communs entre Aglets et notre cadre de spécification des systèmes à base d'agents mobiles.

Le cycle de vie d'un agent Aglet commence par sa création. Au cours de son cycle de vie, un agent Aglet peut être transféré d'un site à un autre, il peut être cloné, supprimé, activé ou désactivé pour une durée de temps bien déterminée. La manipulation des agents Aglets (ajout, suppression, clonage, etc.) est assurée grâce à l'utilisation du serveur Tahiti

livré lors du téléchargement de ASDK et présenté sous forme d'une interface graphique utilisateur.

Le modèle appliqué pour développer un système à base d'agents mobiles sous Aglets est défini comme suit :

- Aglet : est un objet Java mobile qui s'exécute dans un thread indépendant. Il est caractérisé par une *mobilité forte* dont il transporte avec lui son code et son état d'exécution. Un Aglet est connu par un identificateur unique afin être répertorié sur le net.
- Proxy : il fournit des Aglets et sert de bouclier contre les tentatives d'accès directs. Il permet de fournir une transparence à l'emplacement d'un Aglet.
- Contexte : c'est l'environnement d'exécution des Aglets. Il fournit les ressources nécessaires pour l'exécution des agents Aglets. Un Aglet peut consulter les informations contenues dans son contexte d'exécution et de communiquer avec son environnement grâce à l'interface *AgletContext*.
- Hôte : Plusieurs contextes peuvent être regroupés dans un même hôte ou machine. Il est généralement représenté par un noeud dans un réseau.

Dans notre étude de cas, nous visons de développer les préoccupations métiers avec la plateforme Aglets, puis d'imposer les préoccupations techniques (politiques de sécurité) sous forme d'aspects.

D'après les recherches menées, nous avons remarqué l'absence d'une plateforme qui effectue le déploiement des agents mobiles selon la POA, en occurrence la plateforme Aglets. Il est, alors, nécessaire de configurer le *serveur Tahiti de Aglets* afin de supporter l'exécution d'un code aspect tissé avec *JBoss AOP*. Pour ce faire, et dans le cadre d'un travail de Master, nous avons opté d'utiliser Eclipse en tant qu'un environnement de développement intégré pour assurer cette configuration. Nous avons commencé par configurer séparément JBoss AOP et Aglets sous Eclipse et puis de définir une configuration d'exécution afin de tourner les deux plateformes (Aglets et JBoss AOP) simultanément.

6.4 Implémentation de RDyMASS

Dans ce qui suit, nous allons expérimenter l'approche RDyMASS, en fonction des choix d'implémentation effectués. En effet, nous définissons dans une première étape un template spécifique d'aspect selon la structure d'un aspect de JBoss AOP. Dans une deuxième étape, nous développons un générateur de code aspect selon la syntaxe de JBoss AOP. Les préoccupations fonctionnelles seront développées sous la plateforme Aglets. Finalement, les préoccupations techniques et fonctionnelles seront tissées en utilisant JBoss AOP.

La Figure 6.3 illustre les différentes étapes de cette expérimentation.

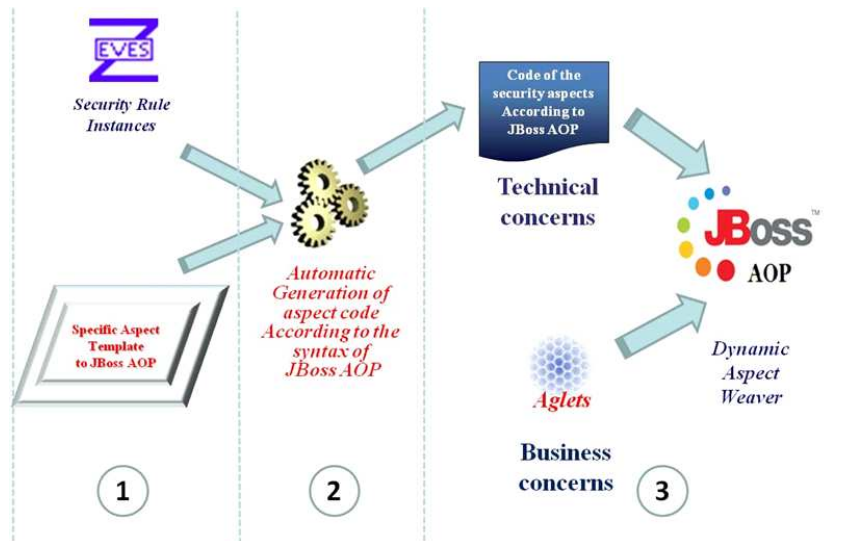


FIG. 6.3 – Cas d’implémentation de l’approche RDyMASS

6.4.1 Template d’aspect de sécurité spécifique pour JBoss AOP

Après avoir présenté la syntaxe et la terminologie du tisseur choisi pour l’implémentation de RDyMASS, nous passons à l’élaboration du template d’aspect de sécurité selon la syntaxe de JBoss AOP. Ce template spécifique à JBoss AOP respecte la même structure du template générique qui a été défini dans la section 6.2.1. Il est composé de trois parties :

- *La coupe* : elle contient les mêmes champs que le template générique. L’exception réside par le fait d’utiliser le mot clé *prepare* à la place de *pointcut*. Ainsi, la coupe spécifique à JBoss AOP est défini comme suit :

```

1 <aop>
2   <prepare expr="execution" (public * RSubject's class ->
3     the corresponding method to the Actions attribute of the RSubject's class)>
4 </aop>

```

Listing 6.2 – Template d’aspect spécifique pour JBoss AOP : partie de la coupe

- *L’intercepteur* : il est illustré par le listing suivant. Pour montrer l’avancement de l’exécution de l’intercepteur, nous décomposons ce template en trois parties annotées [P1’], [P2’], and [P3’] quiinstancient, selon la syntaxe de JBoss AOP, respectivement les parties [P1], [P2], and [P3] du template générique.

```

1 import org.jboss.aop.advice.Interceptor;
2 import org.jboss.aop.joinpoint.Invocation;
3
4 public class Interceptor_Name implements Interceptor {
5   public Object invoke(Invocation invocation) throws Throwable {
6

```

```

7  [P1'] ----- Before part of the Interceptor -----
8  /* Verification of constraints corresponding to the context attribute
9   {Use of a Boolean constraint trait that takes the value "true"
10  if the constraints are verified and false otherwise} */
11
12 [P2']----- Around part of the Interceptor -----
13  if(trait== true){
14      System.out.println("You are authorized to accomplish the request");
15
16      /* Invocation of the method corresponding to the action */
17      Interceptor[] inter = new Interceptor[1];
18      Class dynaclass = Thread.currentThread().getContextClassLoader()
19                      .loadClass(Resource_Class);
20      Interceptor NewInterc = (Interceptor)dynaclass.newInstance();
21      inter[0] = NewInterc;
22
23      Object rsp = invocation.invokeNext(); // proceed in JBoss AOP
24
25 [P3']----- After part of the Interceptor -----
26      /* Updating of the overall system state */
27      return null;
28  }
29  else{
30      System.out.println("You aren't authorized to accomplish the request");
31      return null;
32  }
33  }
34 }

```

Listing 6.3 – Template d’aspect spécifique pour JBoss AOP : partie code de l’intercepteur

- *Les relations (binding code)* : permettent la connexion entre l’étiquette *prepare* et l’intercepteur. Le listing suivant présente la structure du “binding code”. Ce code permet, en premier lieu, de créer une instance de la classe *AdviceBinding* dont le paramètre encapsule la définition d’un point de jonction (ligne 1). Dans la ligne 4, l’intercepteur sera ajouté via le nom de sa classe. Ce dernier sera tissé dans le code global de l’application par l’ajout d’un binding “addBinding” (comme présenté dans la ligne 5) ou dé-tissé suite à la suppression d’un binding “removeBinding” (comme présenté dans la ligne 6).

```

1  AdviceBinding binding_Name = new AdviceBinding("execution(public
2  *com.ibm.awb.launcher.Main -> Method_Name)", null);
3  binding_Name.setName("Name");
4  binding_Name.addInterceptor(Package.Interceptor_Name.class);
5  AspectManager.instance().addBinding(binding_Name);
6  AspectManager.instance().removeBinding(binding_Name);

```

Listing 6.4 – Template d’aspect spécifique pour JBoss AOP : partie du code de binding

6.4.2 Générateur d’aspect de sécurité selon JBoss AOP

La deuxième étape de notre approche RDyMASS consiste à définir, selon JBoss AOP, un générateur d’aspects de sécurité. Ce générateur se base, évidemment, sur le Template d’aspects de sécurité spécifique à JBoss AOP pour transformer une instance de politique

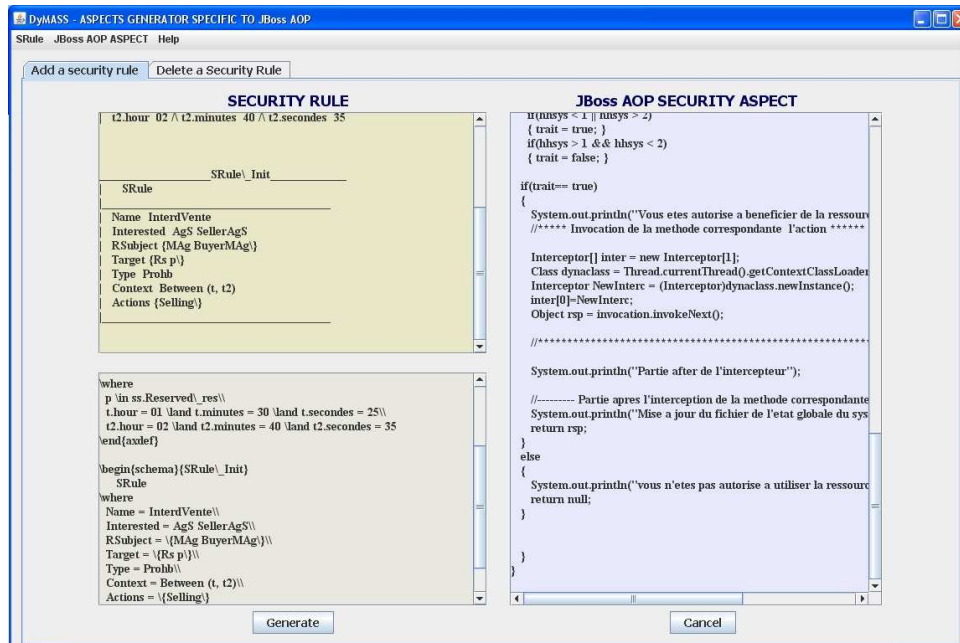


FIG. 6.4 – Onglet d’ajout d’une nouvelle règle de sécurité

de sécurité en un code aspect qui sera tissé dans le code fonctionnel d’une application. Nous présentons, dans cette section, les fonctionnalités de notre générateur, ainsi qu’une brève description de ses interfaces de manipulation.

6.4.2.1 Fonctionnalités du générateur de code

Principalement notre générateur, vise de générer à partir d’une spécification formelle des règles de sécurité d’une politique, le code aspect correspondant. Une fois le code est généré, cet aspect sera automatiquement intégré dans le code des préoccupations techniques d’une application. En plus de la génération automatique de code de sécurité, nous proposons une interface qui permet de supprimer une règle déjà existante. La manipulation des interfaces d’ajout/suppression de règle de sécurité, est explicité dans un utilitaire d’aide à l’utilisation de notre générateur.

En résumé, notre générateur est composé de deux interfaces graphiques (Graphical User Interface, GUI) :

- Une interface principale pour la génération de code.
- Un utilitaire d’aide à l’utilisation du générateur.

L’interface principale pour la génération de code est composée de deux onglets ; l’un pour l’ajout d’un nouveau aspect (voir figure 6.4) et l’autre pour la suppression d’un aspect déjà existant. Cette interface contient, en plus, une barre de menu organisée comme suit :

- Menu *SRule* : il renferme deux commandes. La commande *Z/EVES* permet de lancer

l'outil Z/EVES pour éditer et prouver la consistance d'une instance de règle de sécurité à imposer. Cet outil permet d'exporter les instances de règles de sécurité dans un fichier en format '.tex'. À partir de l'interface de notre générateur, ce fichier '.tex' peut être interpellé par la commande '*Ouvrir*'. En fait, cette commande permet d'importer la règle de sécurité et d'afficher son contenu selon deux formats différents : format '.tex' (en haut, à gauche de l'onglet) et format '.zev' (en bas, à gauche de l'onglet).

- Menu *Aspect en JBoss AOP* : ses commandes permettent d'enregistrer l'aspect généré dans un emplacement qui sera désigné par l'utilisateur.
- Menu *Help* : il englobe les commandes '*Rubrique d'aide*' et '*A propos*'.

Il est à noter que notre générateur se limite à la génération des règles de types autorisation/interdiction avec des contraintes temporelles. Nous distinguons quatre types de contraintes temporelles : *before*, *after*, *between* et *at*.

6.4.3 Etude de cas

Pour illustrer l'approche RDyMASS proposée, nous présentons dans ce qui suit un cas d'exemple qui consiste à imposer des règles pour sécuriser des transactions de commerce électronique dans une application à base d'agents mobiles.

Nous présentons en détail l'étude de cas des transactions électroniques sur Internet et finir de présenter un exemple d'imposition de règle de sécurité dans cette étude de cas.

6.4.3.1 Préoccupations fonctionnelles de l'étude de cas

De nos jours, le développement des applications à base d'agents mobiles reste en continue évolution et métamorphose. Parmi les domaines les plus développés, nous pouvons citer à titre d'exemple : la recherche d'informations dispersées sur le réseau, les systèmes de veille dans les réseaux, le commerce électronique, la télé-communication, etc.

Les agents mobiles prennent une place importante dans les applications de commerce électronique (CE) vu qu'ils puissent, grâce à leurs intelligence et leurs mobilité, de réaliser un certain nombre de tâches d'une manière automatique et plus rapide. En fait, deux classes d'agents peuvent intervenir : les agents acheteurs (coté client) pour retrouver facilement l'information et mieux comparer les prix et les offres. Quant au agents vendeurs (coté fournisseur) permettent de collecter, facilement, les informations concernant les besoins du marché, afin de mieux répondre aux requêtes des clients.

La plénitude de ce domaine est conditionnée par la sécurité des différents partenaires (coté client et coté vendeur). Nous proposons, dans cette section, d'expérimenter notre approche RDyMASS pour sécuriser des transactions de CE. Nous commençons par présenter l'architecture [ANY01] que nous l'avons adopté pour implémenter les préoccupations fonctionnelles d'une application de CE. Ensuite, nous proposons l'imposition, selon l'approche RDyMASS, d'une règle pour sécuriser une transaction de vente.

L'architecture adoptée (voir Figure 6.5) est constituée de deux modules majeurs : un sous-

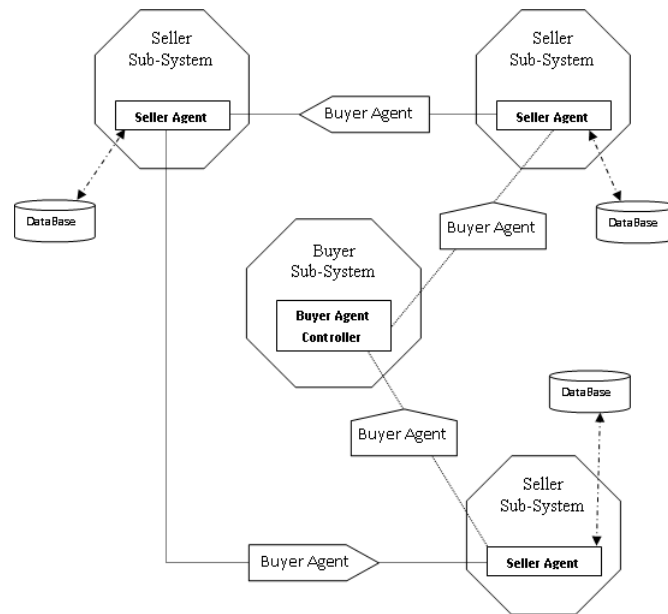


FIG. 6.5 – Architecture adoptée pour le commerce électronique

système de vente (eng. selling sub-system) et un sous-système d'achat (eng. buying sub-system). Nous concevons ces deux modules, selon notre modèle d'agent mobile, par deux systèmes d'agents (AgS) respectivement dénotés par *Seller_AgS* et *Buyer_AgS*.

Le premier sous-système est constitué d'agents de service de vente (eng. selling agent) qui représentent les services et les biens offerts par le système de vente (*Seller_AgS*). Le deuxième est composé d'agents stationnaires qui permettent de lancer la création d'un ou de plusieurs agents acheteurs (eng. buyer agent) pour satisfaire ou répondre à la requête de l'utilisateur. Un agent acheteur noté par *buyer_MAg* sera envoyé sur le réseau pour visiter plusieurs *Seller_AgS* et trouver l'offre d'achat la plus intéressante.

Sous la plateforme Aglets chaque système d'agents *Buyer_AgS/Seller_AgS* sera défini par une instance de serveur Tahiti.

6.4.3.2 Imposition d'un exemple de règle de sécurité

Dans le domaine du CE, différents types d'attaques peuvent se présenter lors d'une transaction électronique. Pour faire face à ces attaques, différentes règles de sécurité doivent être imposées que ce soit coté vendeur ou coté client. Par exemple, il est nécessaire d'imposer une règle de sécurité qui interdit toute opération de vente au moment de l'édition de l'inventaire quotidien de la société. Cette règle sera spécifié par le schéma *Z SRule_Init*. Ce schéma représente une instance du schéma *SRule* défini dans notre cadre formel de sécurité (Section 4.1.1).

Cette instance interdit l'agent vendeur (*Seller_AgS*) d'effectuer l'opération de vente à un

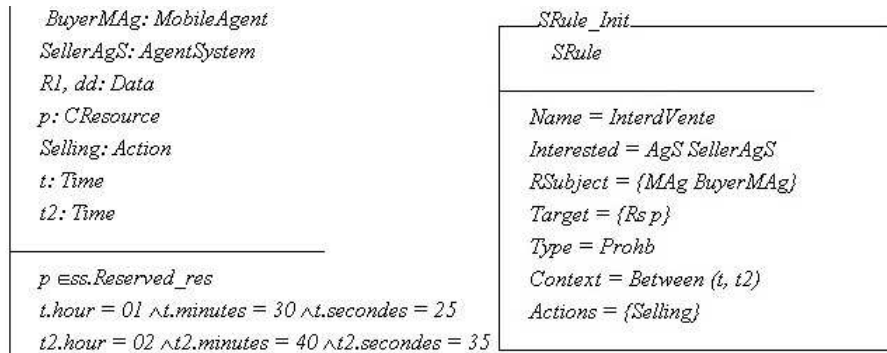


FIG. 6.6 – Spécification formelle d’une instance de règle de sécurité

agent acheteur *Buyer_MAg* entre ‘01h 30mn 25s’ et ‘02h 40mn 35s’ (le moment d’édition de l’inventaire). Pour imposer cette règle de sécurité, nous avons utilisé notre générateur d’aspect (voir Figure 6.4). En effet, à chaque fois l’agent *Buyer_MAg* veut acheter un produit du *Seller_AgS*, l’interceptor vérifie la date du système pour vérifier si l’agent acheteur est autorisé d’acheter le produit à ce moment ou non. Dans le cas positif, il y’aura un binding entre le code métier de vente de produit et l’intercepteur. Sinon, il n’y aura pas le binding ce qui signifie qu’il n’y aura pas de vente et le système se limite à informer l’agent *Buyer_MAg* qu’il n’est pas capable d’acheter le produit maintenant.

6.5 Conclusion

Plusieurs travaux de la littérature ont proposé des approches par aspects pour l’imposition des politiques de sécurité. Certaines approches utilisent la modélisation orientée aspect pour imposer des politiques de sécurité. Généralement, ils utilisent UML en tant que langage de modélisation étendu par les concepts d’aspect et de tissage [GRF02, MTL⁺09]. D’autres approches s’intéressent aux politiques de sécurité seulement à un niveau d’implémentation [RPW06, SK10].

Pour ce qui est de notre approche de déploiement des politiques de sécurité, nous avons proposé de tirer profit d’un cadre théorique pour la spécification et la vérification des politiques de sécurité et nous avons défini une démarche de génération de code aspect à partir d’une spécification fiable et prouvée de politiques de sécurité.

Principalement notre approche consiste en trois étapes : d’abord la définition d’un template d’aspect de sécurité selon deux niveaux d’abstraction (générique/spécifique). Ensuite la génération automatique des aspects de sécurité. Finalement, le tissage dynamique des aspects de sécurité avec le code fonctionnel d’une application à base d’agent mobile. Cette approche a été expérimenté pour sécuriser des transactions de commerce électronique. En effet, nous avons adopté JBoss AOP pour tisser dynamiquement des règles de sécurité dans une application développée sous Aglets.

Conclusion Générale

Le besoin de sécurité s'est considérablement accru avec l'émergence du modèle d'agent mobile. Ainsi, dans le cadre de cette thèse, notre vision de la mobilité met l'accent non pas sur les problèmes de migration des agents en cours d'exécution, mais sur les problèmes liés à la sécurité des agents mobiles des systèmes d'accueil d'agents. En effet, nous avons proposé, dans ce cadre, une approche formelle pour la spécification, la vérification et le déploiement des politiques de sécurité dynamiques.

Afin de lever la complexité liée à la sécurité des systèmes d'agents mobiles, nous avons adopté un niveau d'abstraction élevé qui écarte tout détail d'implémentation.

Notre étude a porté sur deux aspects complémentaires : l'aspect statique et l'aspect dynamique. L'aspect statique est lié à la spécification formelle des politiques de sécurité dans les systèmes à base d'agents mobiles. Afin d'obtenir une vue objective des systèmes, nous avons proposé un cadre formel qui permet de spécifier rigoureusement les concepts liés aux systèmes d'agents mobiles, unifie leurs représentations et définit les relations entre eux. L'aspect dynamique, s'intéresse d'une part à définir formellement l'ensemble des opérations élémentaires de reconfiguration d'une politique de sécurité et de définir, d'autre part, un cadre qui exprime l'adaptabilité de l'agent aux nouvelles exigences de sécurité du système visité et rendre, par conséquent, sa politique cohérente avec celle du système.

Nous avons porté un intérêt considérable à la vérification formelle. Au niveau statique, nous avons profité du système de preuve Z/EVES pour prouver la consistance des spécifications proposées. En outre, pour éviter toute anomalie capable de réduire la performance de la politique, nous vérifions la consistance entre les règles d'une politique. Au niveau dynamique, d'une part, nous avons défini les théorèmes nécessaires pour maintenir la consistance de la politique après une opération de reconfiguration. D'autre part, nous avons proposé un cadre qui permet de vérifier formellement la prévention des attaques et la préservation du niveau de sécurité d'une politique après reconfiguration.

Nous avons défini un cadre opérationnel pour le déploiement des politiques de sécurité dans les systèmes à base d'agents mobiles. Ce cadre tire profit du cadre théorique, que nous avons défini, et applique une approche de génération de code correspondant à une spécification fiable des politiques de sécurité. Cette approche utilise le paradigme de la

programmation orientée aspect qui se caractérise par la modularité et la dichotomie entre les préoccupations fonctionnelles et les préoccupations techniques d'une application.

L'effort de modélisation, résultat de ces travaux de thèse, constitue un référentiel dans les différentes phases de développement des systèmes à base d'agents mobiles. En effet, une suite logique de notre travail consiste à développer, sur la base de ce référentiel, une démarche de développement des systèmes à base d'agents mobiles sécurisés.

Dans cette perspective, il est essentiel de tirer profit de l'utilisation des méthodes formelles afin de pouvoir raisonner rigoureusement et vérifier la satisfaction des propriétés jugées nécessaires dans la conception des systèmes à base d'agents mobiles et sécurisés. En plus, il est essentiel d'adopter un processus systématique et incrémental qui consiste à raffiner progressivement la spécification abstraite des besoins du système en vue de dériver une conception plus concrète. Ce processus doit être enrichi par des aides méthodologiques qui permettent de guider le concepteur et de vérifier avec des obligations de preuve, à chaque passage d'étape, la préservation des propriétés exprimées dans la spécification initiale.

Afin de maîtriser la complexité liée à la sécurité des systèmes à base d'agents mobiles, la démarche proposée doit couvrir les différentes phases de développement, allant de la modélisation, la spécification, la validation, jusqu'à la génération de code. L'approche dirigée par les modèles (MDE : Model Driven Engineering) permet la génération automatique de code à partir d'une modélisation abstraite du système. Elle s'appuie, en fait, sur la définition de plusieurs modèles à différents niveaux d'abstraction et des transformations permettant le passage d'un niveau d'abstraction vers un autre plus détaillé.

Étant donné les avantages de l'approche MDE et ceux des méthodes formelles, nous proposons de les utiliser conjointement en deux niveaux complémentaires : un niveau semi-formel et un niveau formel. Le niveau semi-formel applique l'approche MDE en utilisant des notations graphique afin d'assurer la conception et l'implémentation des politiques de sécurité dans les systèmes à base d'agents mobiles. Au niveau formel, il s'agit d'appliquer un processus de raffinement incrémental pour la conception des systèmes à base d'agents mobiles. Ainsi, nous serons amenés à la traduction des modèles du premier niveau dans le formalisme du deuxième niveau afin de vérifier que chacun de ces modèles respecte les propriétés prouvées formellement dans le processus de conception.

Bibliographie

- [AB96] A. Asperti and N. Busi. Mobile Petri Nets. Technical report, University of Bologna, 1996.
- [AB04] M. Alfalayleh and L. Brankovic. An overview of security issues and techniques in mobile agents. In *Proceedings of the 8th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security*, pages 59–78, University of Salford, UK, September 2004.
- [ABB⁺07] G. Antoniou, M. Baldoni, P. A. Bonatti, W. Nejdl, and D. Olmedilla. Rule-based policy specification. *Secure Data Management in Decentralized Systems*, 33 :169–216, May 2007.
- [ACBC07] S. Ayed, N. Cuppens-Boulahia, and F. Cuppens. An integrated model for access control and information flow requirements. In *Proceedings of the 12th Annual Asian Computing Science Conference Focusing on Computer and Network Security (ASIAN '07)*, volume 4846/2008 of LNCS, pages 111–125, Doha, Qatar, 2007. Springer Berlin/Heidelberg.
- [AG97] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols : The Spi Calculus. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pages 36–47, Zurich, Switzerland, April 1997.
- [AMKHK10] H. Aloulou, M. Loulou, S. Kallel, and A. Hadj-Kacem. RDyMASS : Reliable and dynamic enforcement of security policies for mobile agent systems. In *International Workshop on autonomous and spontaneous security (SETOP 2009) at the 14th European Symposium on Research in Computer Security (ESORICS 2009)*, volume 5939 of LNCS, pages 237–252, Saint Malo, Brittany, France, 2010. Springer-Verlag.
- [ANY01] I. Stewart A. Nguyen and X. Yang. A mobile agent model : Applications for e-commerce. In *Proceedings of the 7th Australian World Wide Web Conference*, pages 247–258, Coffs Harbour, Australia, 2001.
- [Bas96] P. G. Bassett. *FRAMING Software REUSE : Lessons From The Real World*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.
- [BBCM99] C. Bäumer, M. Breugst, S. Choy, and T. Magedanz. Grasshopper - a universal agent platform based on OMG MASIF and FIPA standards. In *Proceedings of the First International Workshop on Mobile Agents for Telecom-*

- munication Applications (MATA'99)*, pages 1–18, Ottawa, Canada, October 1999.
- [BC02] E. Bierman and E. Cloete. Classification of malicious host threats in mobile agent computing. In *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT '02)*, pages 141–148, Port Elizabeth, South Africa, September 2002.
- [BHRS98] J. Baumann, F. Hohl, K. Rothermel, and M. Straßer. Mole - concepts of a mobile agent system. *World Wide Web*, 1(3) :123–137, 1998.
- [BI02] G. Bernard and L. Ismail. Apport des agents mobiles à l'exécution répartie. *Technique et Science Informatiques (RSTI-TSI)*, 21(6) :771–796, 2002.
- [Bib77] K. J. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153, MITRE Corporation, April 1977.
- [BMO01] B. Bauer, J. P. Müller, and J. Odell. Agent UML : A formalism for specifying multiagent software systems. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, 11(3) :207–230, June 2001.
- [Bol01] C. Bolduc. Les démonstrateurs automatiques de théorèmes. Technical Report DIUL-RR-0104, Département d'informatique, Université Laval, Août 2001.
- [Bou95] F. Bounaas. *Gestion de l'évolution dans les bases de connaissances : une approche par les règles*. PhD thesis, Institut National Polytechnique De Grenoble, Octobre 1995.
- [BP76] D. Bell and L. La Padula. Secure computer systems : Unified exposition and multics interpretation. Technical Report MTR-2997, MITRE Corporation, July 1976.
- [BPR99] F. Bellifemine, A. Poggi, and G. Rimassa. JADE : A FIPA-compliant agent framework. In *Proceedings of 4th International Conference and Exhibition on the Practical Applications of Intelligent Agents and Multi-Agent Systems (PAAM '99)*, pages 97–108, London, UK, April 1999.
- [BR05] P. Braun and W. Rossak. *Mobile agents : Basic concepts, mobility models and the tracy toolkit*. Morgan Kaufmann/Elsevier and dpunkt.verlag, USA, 2005.
- [Bry06] C. Bryce. A security framework for a mobile agent system. In *Proceedings of the 6th European Symposium on Research in Computer Security (ESORICS 2000)*, volume 1895/2000 of LNCS, pages 273–290, Toulouse, France, 2006. Springer Berlin/Heidelberg.
- [BSS⁺95] L. Badger, D. F. Sterne, D. L. Sherman, K. M. Walker, and S. A. Haghghat. Practical domain and type enforcement for UNIX. In *Proceedings of the*

- IEEE Symposium on Security and Privacy*, pages 66–77, Oakland, CA, USA, 8-10 May 1995. IEEE Computer Society.
- [CCB06] F. Cuppens and N. Cuppens-Boulahia. Les modèles de sécurité. *Sécurité des systèmes d'information (Traité IC2, série Réseaux et télécoms)*, Juin 2006.
- [CCBR06] F. Cuppens, N. Cuppens-Boulahia, and T. Ramard. Availability enforcement by obligations and aspects identification. In *Proceedings of the First International Conference on Availability, Reliability and Security (ARES'06)*, pages 229–239, Austria, April 2006. IEEE Computer Society.
- [CdCE91] Commission des Communautés Européennes. Critères d'évaluation de la sécurité des systèmes informatiques, version 1.2, juin 1991.
- [CLN00] J. Chomicki, J. Lobo, and S. Naqvi. A logic programming approach to conflict resolution in policy management. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR'2000)*, pages 121–132, Colorado, USA, 2000. Morgan Kaufman.
- [CM04] B. Chess and G. McGraw. Static analysis for security. *IEEE Security & Privacy*, 2(6) :76–79, 2004.
- [Con05] A. Contes. *Une architecture de sécurité hiérarchique, adaptable et dynamique pour la grille*. PhD thesis, Université de Nice-Sophia Antipolis, Faculté des Sciences, September 2005.
- [CPKL06] M. B. Chhetri, R. Price, S. Krishnaswamy, and S. W. Loke. Ontology-based agent mobility modelling. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, page 45a, Kauai, 2006. IEEE Computer Society.
- [Dam02] N. C. Damianou. *A policy framework for management of distributed systems*. PhD thesis, Imperial College, London University, February 2002.
- [DBL97] G. Karjoth-K. Kosaka D. B. Lange, M. Oshima. Aglets : Programming mobile agents in Java. In *Proceedings of the International Conference on WorldWide Computing and Its Applications (WWCA '97)*, volume 1274/1997 of *LNCS*, pages 253–266, Tsukuba, Japan, 1997. Springer Berlin/Heidelberg.
- [DDLS01] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks (POLICY '01)*, volume 1995 of *LNCS*, pages 18–38, Bristol, UK, January 2001. Springer-Verlag.
- [DRF03] P. Dias, C. Ribeiro, and P. Ferreira. Enforcing history-based security policies in mobile agent systems. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY '03)*, pages 231–234, Lake Como, Italy, 4-6 June 2003. IEEE Computer Society.

- [Fer95] J. Ferber. *Les Systèmes multi-agents : Vers une intelligence collective*. InterEditions, 1995.
- [FfIPA02] Foundation for Intelligent Physical Agents. FIPA Agent Management Support for Mobility Specification, document number dc00087c. Technical report, Geneva, Switzerland, May 2002.
- [FSG⁺01] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3) :224–274, August 2001.
- [GB06] P. Greenwood and L. Blair. A Framework for policy driven auto-adaptive systems using dynamic framed aspects. *Transactions on Aspect Oriented Software Development II*, 4242/2006 :30–65, November 2006.
- [GRF02] G. Georg, I. Ray, and R. France. Using aspects to design a secure system. In *Proceedings of the Eighth International Conference on Engineering of Complex Computer Systems (ICECCS '02)*, page 117. IEEE Computer Society, 2002.
- [GS05] W. Gilani and O. Spinczyk. Dynamic aspect weaver family for family-based adaptable systems. In *NetObjectDays (NODe '05)*, Lecture Notes in Informatics, pages 94–109, Erfurt, Germany, September 2005. German Society of Informatics.
- [Ham06] K. W. Hamlen. *Security policy enforcement by automated program-rewriting*. PhD thesis, Cornell University, August 2006.
- [Har08] C. Hardebolle. *Composition de modèles pour la modélisation multi-paradigme du comportement des systèmes*. PhD thesis, Université Paris-Sud XI, UFR Scientifique d'Orsay, Paris, France, Novembre 2008.
- [HLG08] H. Hachicha, A. Loukil, and K. Ghedira. MAMT : an environment for modeling and implementing mobile agents. In *Proceedings of the 6th International Workshop From Agent Theory to Agent Implementation, AT2AI-6 at the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Estoril, Portugal (EU), May 2008.
- [Jan00] W. A. Jansen. Countermeasures for mobile agent security. *Computer Communications*, 23(17) :1667–1676, 2000.
- [JK99] W. Jansen and T. Karygiannis. NIST Special Publication 800-19- Mobile agent security. Technical report, National Institute of Standards and Technology, October 1999.
- [Kal03] A. A. Kalam. *Modèles et politiques de sécurité pour les domaines de la santé et des affaires sociales*. PhD thesis, Institut National Polytechnique de Toulouse, Décembre 2003.
- [KBB⁺03] A. A. Kalam, R. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access

- Control. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)*, Como, Italie, June 2003.
- [KBC02] A. Ketfi, N. Belkhatir, and P. Y. Cunin. Adaptation dynamique : concepts et expérimentations. In *Proceedings of the 15th International Conference on Software and Systems Engineering and their Applications (ICSSEA'02)*, page 8 p, Paris, France, December 2002.
- [KCM⁺09] Slim Kallel, Anis Charfi, Mira Mezini, Mohamed Jmaiel, and Karl Klose. From Formal Access Control Policies to Runtime Enforcement Aspects. In *Proceedings of the International Symposium on Engineering Secure Software and Systems*, pages 16–31. Springer, Fevrier 2009.
- [Kha06] K. Khan. JBOSSAOP : Framework for Organizing Cross Cutting Concerns. <http://jboss.org/jbossaop/>, 2006.
- [KJ05] M. Kusek and G. Jezic. Modeling agent mobility with UML sequence diagram. *Agent-Oriented Software Engineering TFG, AL3*, February 2005.
- [KLM⁺97] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of 11th European Conference on Object-Oriented Programming*, volume 1241/1997 of LNCS, pages 220–242, 1997.
- [KSW96] R. Kolyang, T. Santen, and B. Wolff. A structure preserving encoding of Z in Isabelle-Hol. In *Proceedings of 9th International Conference on Theorem Proving in Higher Order Logics*, volume 1125/1996 of LNCS, pages 283–298, Turku, Finland, 1996. Springer-Verlag.
- [KW03] S. Kaffille and G. Wirtz. Integrating MASIF and FIPA Standards for Agent System Interoperability. In *Proceedings of The 7th International Conference on Software Engineering and Applications (SEA'03)*, pages 798–806, Marina del Rey, USA, 03-05 November 2003.
- [KWT04] M. Kang, L. Wang, and K. Taguchi. Modelling mobile agent applications in UML2.0 activity diagrams. In *Proceedings of the 3th International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS'04), W16L Workshop - 26th International Conference on Software Engineering*, pages 104–111, Edinburgh, Scotland, UK, May 2004.
- [LAP04] S. Leriche, J.-P. Arcangeli, and M. Pantel. Agents mobiles adaptables pour les systèmes d'information pair à pair hétérogènes et répartis. In *Proceedings de la 4 ème Conférence Internationale sur les NOuvelles TEchnologies de la REpartition (NOTERE '04)*, pages 29–43, Saidia, Maroc, Juin 2004.
- [LBN99] J. Lobo, R. Bhatia, and S. Naqvi. A policy description language. In *Proceedings of the National Conference of the American Association for Artificial Intelligence (AAAI'99)*, pages 291–298, Orlando, Juin 1999.

- [Lef94] A. Lefebvre. *L'architecture client-serveur : aspects techniques, enjeux stratégiques*. Number 2. 1994.
- [LHG06] A. Loukil, H. Hachicha, and K. Ghedira. A proposed approach to model and to implement mobile agents. *International Journal of Computer Science and Network Security (IJCSNS)*, 6(3B) :125–129, March 2006.
- [LHKJ04a] M. Loulou, A. Hadj-Kacem, and M. Jmaiel. Agent et SMA : Modélisation de la coopération et de la négociation selon la notation Z. In *Proceedings of the 8th Maghrebian Conference on Software Engineering and Artificial Intelligence*, pages 85–98, Sousse, Tunisia, May 2004.
- [LHKJ04b] M. Loulou, A. Hadj-Kacem, and M. Jmaiel. Formalization of cooperation in MAS : Towards a generic conceptual model. In *Proceedings of the IX Ibero-American Conference on Artificial Intelligence (IBERAMIA '04)*, volume 3315 of *Lecture Notes in Artificial Intelligence*, pages 43–52, Puebla, Mexique, 2004. Springer-Verlag.
- [LHKJ05] M. Loulou, A. Hadj-Kacem, and M. Jmaiel. A formal model for mobile agent systems using Z. In *Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'05)*, Cairo, Egypt, January 2005.
- [LHKJM06] M. Loulou, A. Hadj-Kacem, M. Jmaiel, and M. Mosbah. A conceptual model for secure mobile agent systems. In *Proceedings of the IEEE International Conference on Computational Intelligence and Security ((CIS'2006))*, pages 524–527, Guangzhou, China, November 2006. IEEE.
- [LHKJM08] M. Loulou, A. Hadj-Kacem, M. Jmaiel, and M. Mosbah. A formal security framework for mobile agent systems : Specification and verification. In *Proceedings of the 3rd International Conference on Risks and Security of Internet and Systems*, pages 69–76, Tozeur, Tunisia, 2008.
- [LM98] D. B. Lange and O. Mitsuru. *Programming and Deploying Java Mobile Agents Aglets*. Addison-Wesley, Boston, MA, USA, 1998.
- [LS99] E. C. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 25(6) :852–869, 1999.
- [LTHK⁺07] M. Loulou, M. Tounsi, A. Hadj-Kacem, M. Jmaiel, and M. Mosbah. A formal approach to prevent attacks on mobile agent systems. In *Proceedings of the International Conference on Emerging Security Information, Systems, and Technologies (SECURWARE '07)*, pages 42–47, 2007.
- [LY99] T. Lindholm and F. Yellin. *Java Virtual Machine Specification*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, second edition, 1999.

- [LZ01] W. Li and M. Zhang. MAT : a mobile agent system for supporting autonomous mobile agents. *Journal of Research and Practice in Information Technology*, 33(3) :211–227, 2001.
- [Mae87] P. Maes. Concepts and experiments in computational reflection. In *Proceedings of the International Conference on Object-oriented programming systems, languages and applications (OOPSLA '87)*, pages 147–155, Orlando, Florida, United States, 1987. ACM.
- [Mar02] R. Marvie. *Séparation des préoccupations et méta-modélisation pour environnements de manipulation d'architectures logicielles à base de composants*. PhD thesis, Université des Sciences et Technologies de Lille, décembre 2002.
- [MBB⁺98] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. MASIF : The OMG Mobile Agent System Interoperability Facility. *Personal and Ubiquitous Computing*, 2(2) :117–129, June 1998.
- [MG01] F. Muscutariu and M.P. Gervais. On the modeling of mobile agent-based systems. In *Proceedings of the 3th International Workshop on Mobile Agents for Telecommunication Applications (MATA'01)*, volume 2164/2001 of LNCS, pages 219–233, Montreal, Canada, 2001. Springer Berlin/Heidelberg.
- [MMM09] M.Loulou, M.Jmaiel, and M.Mosbah. Dynamic security framework for mobile agent systems : specification, verification and enforcement. *International Journal of Information and Computer Security*, 3(3/4) :321–336, 2009.
- [MS97] I. Meisels and M. Saaltink. The Z/EVES Reference Manual. Technical report TR-97-5493-03d, ORA Canada, 267 Richmond Road, Suite 100, Ottawa, Ontario K1Z 6X3, Canada, September 1997.
- [MSD01] R. Montanari, C. Stefanelli, and N. Dulay. Flexible security policies for mobile agent systems. *Microprocessors and Microsystems*, 25(2) :93–99, 20 April 2001.
- [MT06] L. Ma and J. J. P. Tsai. *Security Modeling and Analysis of Mobile Agent Systems*, volume 5 of *Electrical and computer engineering*. Imperial College Press, University of Illinois at Chicago, USA, 2006.
- [MTL⁺09] D. Mouheb, C. Talhi, V ; Lima, M. Debbabi, L. Wang, and M. Pourzandi. Weaving security aspects into uml 2.0 design models. In *Proceedings of the 13th workshop on Aspect-oriented modeling (AOM '09)*, pages 7–12, Charlottesville, Virginia, USA, 2009. ACM.
- [Mye99] A. C. Myers. JFlow : practical mostly-static information flow control. In *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL'99)*, pages 228–241, San Antonio, Texas, United States, 1999. ACM.

- [Nac97] C. Nachenberg. Computer virus-antivirus coevolution. *Communications of the ACM*, 40(1) :46–51, 1997.
- [NNLC99] H. S. Nwana, D. T. Ndumu, L. C. Lee, and J. C. Collis. Zeus : A toolkit for building distributed multi-agent systems. *Applied Artificial Intelligence Journal*, 13(1) :129–185, January 1999.
- [Oli08] A. S. Oliveir. *Réécriture et modularité pour les politiques de sécurité*. PhD thesis, Université Henri Poincaré, Mars 2008.
- [PS97] H. Peine and T. Stolpmann. The architecture of the Ara platform for mobile agents. In *Proceedings of the First International Workshop on Mobile Agents (MA'97)*, volume 1219 of *LNCS*, pages 50–61, Berlin, Germany, April 1997. Springer-Verlag.
- [PS04] J. Park and R. Sandhu. The UCONABC usage control model. *ACM Transactions on Information and System Security (TISSEC)*, 7(1) :128–174, 2004.
- [PXB03] S. Pears, J. Xu, and C. Boldyreff. Mobile agent fault tolerance for information retrieval applications : An exception handling approach. In *Proceedings of the The Sixth International Symposium on Autonomous Decentralized Systems (ISADS'03)*, page 115, Pisa, Italy, April 2003. IEEE Computer Society.
- [RP05] J.-P. Retaillé R. Pawlak, L. Seinturier. *Foundations of AOP for J2EE Development*. Apress, Berkely, CA, USA, September 2005.
- [RPW06] R. Ramachandran, D. J. Pearce, and I. Welch. AspectJ for multilevel security. In *Proceedings of the 5th AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS)*, pages 13–17, Bonn, Germany, March 2006.
- [RV02] P. Roques and F. Vallée. *UML en action*. second edition, November 2002.
- [RZFG00] C. Ribeiro, A. Zuquete, P. Ferreira, and P. Guedes. Security policy consistency. In *Proceedings of the First Workshop on Rule-Based Constraint Reasoning and Programming at the First International Conference on Computational Logic (CL2000)*, Imperial College, London, England, July 2000.
- [RZFG01] C. Ribeiro, A. Zúquete, P. Ferreira, and P. Guedes. SPL : An access control language for security policies with complex constraints. In *Network and Distributed System Security Symposium (NDSS'01)*, pages 89–107, San Diego, California, February 2001.
- [Saa97] M. Saaltink. The Z/EVES system. In *Proceedings of the 10th International Conference of Z Users on The Z Formal Specification Notation*, volume 1212 of *LNCS*, pages 72–85, Reading, UK, April 1997. Springer-Verlag.
- [SC05] T. Sans and F. Cuppens. Nouvelles problématiques de contrôle d'usage dans les systèmes d'information. In *Proceedings de Atelier de sécurité des systèmes d'informations lors de INFORSID*, Grenoble, France, Mai 2005. RSM - Dépt. Réseaux, Sécurité et Multimédia (Institut Télécom Bretagne).

- [Sch00] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1) :30–50, February 2000.
- [SEM04] K. Saleh and C. El-Morr. M-UML : an extension to UML for the modeling of mobile agent-based software systems. *Information and Software Technology*, 46(4) :219–227, 2004.
- [SG90] J. W. Stamos and D. K. Gifford. Remote evaluation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(4) :537–564, 1990.
- [SJ97] V. S. Subrahmanian S. Jajodia, P. Samarati. A logical language for expressing authorizations. In *Proceedings of the Symposium on Security and Privacy*, pages 31–42, Oakland, CA, USA, May 1997. IEEE Press.
- [SK10] K. Sirbi and P. J. Kulkarni. Stronger enforcement of security using aop and spring aop. *CoRR*, abs/1006.4550, 2010.
- [Smi04] G. Smith. A formal framework for modelling and analysing mobile systems. In *Proceedings of the 27th Australasian conference on Computer science (ACSC'04)*, pages 193–202, Dunedin, New Zealand, 2004. Australian Computer Society, Inc.
- [SP00] R. Hadingham S. Poslad, P. Buckle. The FIPA-OS agent platform : Open Source for Open Standards. In *Proceedings of the 5th International Conference and Exhibition on the Practical Application of Intelligent Agents and MultiAgents*, pages 355–368, Manchester, UK, April 2000.
- [SZ98] P. Staski and A. Zaslavsky. Expressing dynamics of mobile agent systems using ambient calculus. In *Proceedings of the 9th International Workshop on Database and Expert Systems Applications (DEXA'98)*, pages 434–439, Vienna, Austria, 26-28 August 1998. IEEE Computer Society.
- [Tal07] C. Talhi. *Memory-Constrained Security Enforcement*. Phd thesis, Faculty of Science and Engineering, University of Laval, Quebec, Canada, April 2007.
- [TH96] A. T. Teije and F. V. Harmelen. Using reflection techniques for flexible problem solving (with examples from diagnosis). *Future Generation Computer Systems*, 12(2-3) :217–234, 1996.
- [Tho89] B. Thomsen. A calculus of higher order communicating systems. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL'89)*, pages 143–154, Austin, Texas, United States, 1989. ACM.
- [TS97] R. Thomas and R. Sandhu. Task-based authorization controls (TBAC) : A family of models for active and enterprise-oriented authorization management. In *Proceedings of the 11th IFIP Working Conference on Database Security*, Lake Tahoe, California, USA, 1997.
- [TTD06] C. Talhi, N. Tawbi, and M. Debbabi. Execution monitoring enforcement for limited-memory systems. In *Proceedings of the 2006 International*

- Conference on Privacy, Security and Trust (PST '06)*, pages 1–12, Markham, Ontario, Canada, 2006. ACM.
- [TTK⁺99] A. R. Tripathi, R. Tripathi, N. M. Karnik, R. D. Singh, A. Tanvir, J. Eberhard, and A. Prakash. Development of mobile agent applications with ajanta. Technical Report MN 55455, Department of Computer Science, University of Minnesota, 1999.
- [UE06] S. Ugurlu and N. Erdogan. A flexible policy architecture for mobile agents. In *Proceedings of the 32nd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '06)*, pages 538–547, Merín, Czech Republic, 21–27 January 2006.
- [VC99] J. Vitek and G. Castagna. Seal : A framework for secure mobile computations. In *Proceedings of the Workshop on Internet Programming Languages (ICCL'98)*, volume 1686/1999 of LNCS, pages 47–77, Chicago, IL, USA, 1999. Springer-Verlag.
- [WD96] J. Woodcock and J. Davies. *Using Z : Specification, Refinement and Proof*. International Thomson Computer Press, Upper Saddle River, NJ, USA, 1996.
- [XD00] D. Xu and Y. Deng. Modeling mobile agent systems with high level Petri Nets. In *Proceedings of the International Conference on Systems, Man, and Cybernetics (SMC'00)*, volume 5, pages 3177–3182, Nashville, Tennessee, USA, October 2000. IEEE computer society.
- [YHWL03] H.-H. Huang Y.-H. Wang, C.-L. Wang and W.-H. Lo. Applying mobile agents to e-business. *Tamkang Journal of Science and Engineering*, 6(3) :159–172, 2003.
- [Zda04] S. Zdancewic. Challenges in information-flow security. In *Proceedings of the First International Workshop on Programming Language Interference and Dependence*, Verona, Italy, 25 August 2004.
- [ZKI01] M. Zhang, A. Karmouch, and R. Impey. Towards a secure agent platform based on FIPA. In *Proceedings of the 3th International Workshop on Mobile Agents for Telecommunication Applications (MATA '01)*, volume 2164/2001 of LNCS, pages 277–289, Montreal, Canada, 2001. Springer Berlin/Heidelberg.

Liste de mes publications

Publications dans des revues internationales

M. Loulou, M. Jmaiel, and M. Mosbah. Dynamic security framework for mobile agent systems : specification, verification and enforcement. *International Journal of Information and Computer Security*, 3(3/4) : 321-336, 2009.

Publications dans des notes de lecture

H. Aloulou, M.Loulou, S. Kallel, and A. Hadj Kacem. Rdymass : Reliable and dynamic enforcement of security policies for mobile agent systems. In *International workshop on autonomous and spontaneous security (SETOP '09) at the 14th European Symposium on Research in Computer Security (ESORICS 2009)*, volume LNCS 5939, pages 237-252, Saint Malo, Brittany, France, 2010. Springer-Verlag.

M. Loulou, A. Hadj-Kacem, and M. Jmaiel. Formalization of cooperation in MAS : Towards a generic conceptual model. In *Proceedings of the IX Ibero-American Conference on Artificial Intelligence (IBERAMIA '04)*, volume 3315 of *Lecture Notes in Artificial Intelligence*, pages 43-52, Puebla , Mexique, 2004. Springer-Verlag.

Publications dans des conférences internationales

M. Loulou, A. Hadj-Kacem, M. Jmaiel, and M. Mosbah. A formal security framework for mobile agent systems : Specification and verification. In *Proceedings of the 3rd International Conference on Risks and Security of Internet and Systems(CRiSIS '08)*, pages 69-76, Tozeur, Tunisia, 2008.

M. Loulou, M. Tounsi, A. Hadj-Kacem, M. Jmaiel, and M. Mosbah. A formal approach to prevent attacks on mobile agent systems. In *Proceedings of the International Conference on Emerging Security Information, Systems, and Technologies (SECURWARE '07)*, pages 42-47. IEEE Computer Society, 2007.

M. Loulou, A. Hadj-Kacem, M. Jmaiel, and M. Mosbah. A conceptual model for secure mobile agent systems. In *Proceedings of the IEEE International Conference on Computational Intelligence and Security (CIS '06)*, pages 524-527, Guangzhou, China, November 2006. IEEE.

M. Loulou, A. Hadj-Kacem, and M. Jmaiel. A formal model for mobile agent systems using Z. In Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Applications (AICCSA '05), Cairo, Egypt, January 2005.

M. Loulou, A. Hadj-Kacem, and M. Jmaiel. Agent et SMA : Modélisation de la coopération et de la négociation selon la notation Z. In Proceedings of the Eighth Maghrebian Conference on Software Engineering and Artificial Intelligence, pages 85-98, Sousse, Tunisia, May 2004.