République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Larbi Ben M'Hidi-Oum El Bouaghi
Faculté des Sciences Exactes et Sciences de la Nature et de la Vie
Département de Mathématiques et d'Informatique

N° D'ordre :……

Série :……..

MEMOIRE

Pour l'obtention du diplôme de Magister en Informatique

OPTION

Intelligence Artificielle et Imagerie

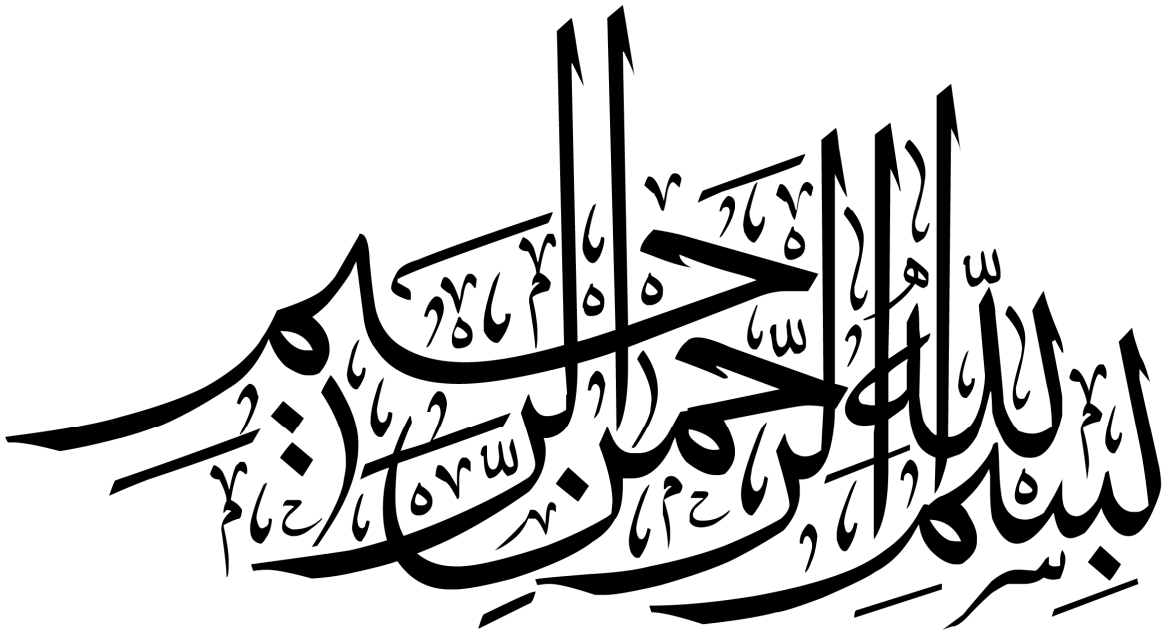
Présenté par

Kalache Ayyoub

Contrôle de la Réorganisation dans les SMA : Une Approche basée sur le paradigme Aspect

DEVANT LE JURY COMPOSE DE

Mr. Nini Brahim	Maitre de conférences	Univ. d'O.E.B.	**Président**
Mr. Farid Mokhati	Maitre de conférences	Univ. d'O.E.B.	**Rapporteur**
Mr. Boutekkouk Fateh	Maitre de conférences	Univ. d'O.E.B.	**Examinateur**
Mr. Kazar Okba	Professeur	Univ. de Biskra	**Examinateur**
Mr. Maamri Ramdane	Professeur	Univ. de Constantine	**Examinateur**


بسم الله الرحمن الرحيم

Contrôle De La Réorganisation Dans Les SMA

Une Approche Basée Sur Le Paradigme Aspect

Dédicaces

« Louange à Dieu, le tout puissant »

A ceux qu'ont attendu avec patience les fruits de leur éducation : mes très chers parents,

Témoignage d'affectation et de grande reconnaissance,

Que Dieu les garde pour moi,

À ma sœur et mes trois frères,

À mon oncle Mohamed Ali,

À mes cousant Mohamed Salah, et Nadjib

A tout ma famille,

À mes amis Zinou, Khalid, Mustapha, Mohamed , Salim,

À Mohamed.S et Ayyoub,

À tous mes amis qui sont proche de mon cœur

et dont je n'ai pas cité leurs noms

À tous ceux qui m'ont connu, soutenu et aimé

À tous ceux qui me sont chers

À TOUS, je dédie ce travail en leur adressant tous mes sentiments

D'affection et de considération !

K.Ayyoub

Remerciements

***E**n tout premier lieu je remercie Allah le tout puissant, à la sagesse et au savoir infini « gloire à Toi ! nous n'avons de savoir que ce que tu nous as appris. Certes c'est toi l'omniscient, le sage. » (Sourate al-baquarah verset 32).*

***J**e tiens à remercier particulièrement mon encadreur monsieur MOKHATI Farid et mon co-encadreur BADRI Mourad de m'avoir proposés ce sujet et avoir dirigés mes travaux, pour leurs constantes disponibilités, pour les conseils qu'ils n'ont cessés de me prodiguer et enfin pour leurs encouragements.*

***J**e voudrais également remercier les membres de jury qui m'ont fait l'honneur de bien vouloir juger ce travail et d'y apporter leurs critiques constructives et leurs valeureuses remarques.*

***M**es reconnaissances vont également à l'ensemble de mes enseignants, pour le savoir inestimable qu'ils m'ont transmis tout au long de mon cursus.*

***M**erci à toutes les personnes dont l'amitié m'a apporté les moments de réconfort et distraction nécessaires lors du déroulement d'un tel projet, notamment mes chers amis Zinou, Khalid, Mohamed et aussi Salim.*

***S**on oublier de remercier mes amis d'études et tous ceux qui m'ont aidé de près ou loin dans ce projet*

***E**nfin, remercier mes parents serait se répéter citer leur affection serait un pléonasme parfois, pour exprimer plus que ce que j'ai envie de dire j'ai rien retrouvé que : « Ô mon Seigneur, fais-leur, a tous deux miséricorde comme ils m'ont élevé tout petit » (Sourate al-Isar verset 24).*

K.Ayyoub

Abstract:

Reorganization in Multi-Agent Systems plays a crucial role in the dynamic adaptation of the structure and the behavior of organizations. In order to ensure consistency of the resulting organization, the reorganization process has to be controlled. This dissertation proposes a novel approach to controlling the reorganization process of MAS, which are specified and implemented using the framework OMACS (Organizational Model for Adaptive Computational Systems). The proposed control process is accomplished using the framework MOP (Monitoring Oriented Programming) for supporting the verification of some reorganizational properties. The proposed approach, supported by a software tool that we developed, is illustrated using a concrete case study.

Keywords: MAS, Organization, Control, Adaptation, OMACS, JavaMOP, RunTime Monitoring.

Résumé:

La Réorganisation dans les systèmes multi-agents joue un rôle crucial dans l'adaptation dynamique de la structure et du comportement des organisations. Afin d'assurer la cohérence de l'organisation résultante, le processus de réorganisation doit être contrôlé. Dans ce mémoire, nous proposons une nouvelle approche pour contrôler le processus de réorganisation des SMA, qui sont spécifiés et implémentés à l'aide de Framework OMACS (Organizational Model for Adaptive Computational Systems). Le processus de contrôle proposé est réalisé en utilisant le Framework MOP (Monitoring Oriented Programming) pour supporter la vérification de certaines propriétés de réorganisation. L'approche proposée, est soutenue par un outil logiciel, que nous avons développé, est illustrée par une étude de cas concrète.

Mots-clés: SMA, organisation, Contrôle, Adaptation, OMACS, JavaMOP, Moniteur.

ملـخـص

إعادة التنظيم في النظم متعددة العملاء يلعب دورا اساسيا في التكيف الديناميكي لهيكلة وسلوكيات هذا النوع من النظم . لضمان صحة و اتساق المنظمة الناتجة عن هذا التكيف، لابد من التحكم من عمليات إعادة التنظيم .نقترح في هذه الأطروحة مقاربة جديدة للمراقبة و للتحكم في عملية إعادة تنظيم النظم متعددة العملاء ، والتي تم انشائها و تطويرها باستخدام النموذج OMACS (نموذج التنظيمي لنظم الحاسوبية المتكيف). عملية التحكم المقترحة تعتمد على الوسط MOP (مراقبة الموجهة للبرمجة) من اجل مراقبة التحقق من خصائص السلوكية لنظام خلال عملية إعادة التنظيم . الطريقة المقترحة مدعمة بأداة برمجية قمنا بتطويرها وقد تم تبين ناجعتها بعد استخدامها في حالة عملية.

الكلمات المفتاحية : ,نظام متعدد العملاء, منظمة، التحكم، التكيف، OMACS ، JavaMOP ، مراقبة و وقت العمل.

Table des Matières

Table des Figures

Table des Tableaux

Introduction Générale

1. Cadre et motivation

Ces dernières années, les Systèmes Multi-Agent (SMA) sont de plus en plus sollicités pour la mise en œuvre des systèmes complexes, flexibles et fiables capables d'opérer dans des environnements ouverts, dynamiques et incertains. Face à ces nouvelles exigences, les SMA ont évolués en intégrant d'autres aspects tels que les aspects organisationnels, sociaux, normatifs, …etc. Aujourd'hui, on parle de la séparation agent/organisation (SMA, centré organisation) où l'organisation (une entité complexe qui a ses propres objectifs) est définie indépendamment de l'agent (Ferber et al., 2004; Picard et al., 2009). Avec ce point de vue centré organisation, le comportement global des SMA est lié à leur structure organisationnelle est non pas aux comportements individuels des agents; ce qui a permis d'augmenter le degré d'ouverture des SMA et d'hétérogénéité des Agents (Dignum, 2009).

La notion d'ouverture des SMA exige que ceux-ci doivent être capables de prendre en considération la nature dynamique de l'environnement; d'être plus flexible, afin de s'adapter avec les changements qui peuvent survenir soit dans leur environnement, ou bien dans les objectifs du système (dû au changement d'environnement). L'adaptation exprime la capacité du système de changer son organisation, de se réorganiser face aux certains évènements afin qu'il puisse garder son efficacité et continuer d'exister (Argente et al., 2013; Dignum, 2009; Esparcia and Argente, 2012).

Plusieurs travaux sur les SMA adaptatifs ont vu le jour, qu'ils soient sur l'adaptation auto-organisationnelle, ou sur l'adaptation réorganisationnelle, mais un problème se pose toujours : l'assurance de la qualité et de l'efficacité de la nouvelle organisation (Dignum et al., 2004). En effet, le dynamisme qui se crée au sein du système (dû aux changements d'organisation) rend le comportement de ce dernier très difficile à prévoir (parfois impossible).

Dans la littérature, peu de travaux ont abordé le problème de contrôle de la réorganisation (Guessoum et al., 2004; Harmon and DeLoach, 2008). Généralement, les techniques de réorganisation existantes ne se basent pas sur l'autonomie d'organisation et d'agent (agent organisateur) comme moyen de construction dynamique de nouvelles organisations (organisation auto-constructive), car la difficulté d'élaborer et de contrôler ce genre de réorganisation a été toujours une tâche fastidieuse (nécessite un niveau de raisonnement macro chez les agents de l'organisation).Pour ces raisons, la plupart des travaux de réorganisation proposés ont contourné le problème de contrôle dynamique de la

réorganisation en proposant soient des processus de réorganisation statiques, exerçants des comportements prédéfinis et connus à l'avance, soient des processus de réorganisation dynamiques, mais spécifiques (se focalisent sur une seule manière de changement d'organisation) produisant dynamiquement des changements à simples effets, plus ou moins prédictibles et faciles à les maitriser (Artikis et al., 2009; Hoogendoorn and Treur, 2009; Mahmoud et al., 2013; De Paz et al., 2014; Wang and Liang, 2006; Weyns et al., 2010).

En revanche, les travaux qui traitent directement le contrôle dynamique de la réorganisation proposent soient des solutions élaborées spécifiquement pour accompagner un processus de réorganisation bien déterminé (impossible de les généraliser à d'autres problèmes de réorganisation), soient des solutions générales basées sur la définition, dans la phase de spécification de processus de réorganisation, des contraintes et lois comportementales à respecter lors de l'implémentation du processus de réorganisation (Harmon and DeLoach, 2008). Cependant, cette dernière solution présente un inconvénient majeur; l'absence d'un moyen de vérification et d'assurance de conformité de processus de réorganisation à la spécification de contraintes et lois établie.

Dans ce contexte, nous proposons dans ce mémoire, une nouvelle approche (Kalache et al., 2014) de contrôle des processus de réorganisation dynamiques des systèmes multi-agents qui sont spécifiés et implémentés par le Framework OMACS (DeLoach, 2009). Le processus de contrôle proposé est établi en utilisant le Framework JavaMOP (Monitoring Oriented Programming)(Meredith et al., 2012). Ce Framework va nous permettre de spécifier (hors ligne) , de vérifier dynamiquement (en exécution) certaines propriétés réorganisationnelles de système et ainsi de corriger (si nécessaire) son comportement dynamiquement (en cour d'exécution).

2. Organisation du Mémoire

Le reste de ce mémoire est organisé en quatre chapitres :

Le premier chapitre présente les notions de base sur les SMA organisationnels, les modèles proposés, les processus de la réorganisation et les mécanismes du contrôle de la réorganisation. À la fin de ce chapitre, nous citons les différents critères nécessaires pour la proposition des mécanismes de réorganisation fiable.

Dans le deuxième chapitre, nous présentons le modèle et le Framework organisationnel OMACS que nous avons choisi pour la réalisation des SMA

organisationnels. Nous discutons également, dans ce chapitre, les limites de ce Framework et les lacunes de son processus de développement des SMA réorganisationnels.

Nous consacrons le troisième chapitre à la présentation de notre approche du contrôle de la réorganisation dans les SMA adaptatifs. Une approche basée sur l'amélioration de Framework OMACS par l'intégration du Framework JavaMOP dans le processus de spécification des contraintes organisationnelles et réorganisationnelles.

Le quatrième chapitre présente l'outil que nous avons développé pour supporter notre approche, ainsi un exemple illustrant l'application de cette approche .

Chapiter I : Contrôle d'adaptation Des SMA Organisationnels

1. Introduction

Depuis leur apparition, les SMA ont été considérés comme des sociétés d'individus autonomes, capables d'interagir et coordonner pour réaliser leurs objectifs. Les premiers travaux ont concentré sur les aspects de l'autonomie et de la dynamique des agents pour résoudre les problèmes complexes et distribués en adoptant des approches à base de comportements émergents. Malgré leurs efficacités à résoudre ce genre de problèmes, les SMA souffrent du problème de contrôle de comportement et de régulation de l'autonomie des agents (Wooldridge et al., 2000). Pour résoudre cet inconvénient, un point de vue social, inspiré de la théorie d'organisation a été adopté pour la spécification et l'implémentation des SMA. On parle des SMA organisés selon une structure organisationnelle définie explicitement (organisation) capable de supporter les activités des agents et contraindre l'autonomie des agents tout en gardant leurs efficacités et un certain niveau de contrôle sur leur comportement (Dignum, 2009; Ferber et al., 2004).

Les SMA sont des entités dynamiques qui peuvent changer leur environnement, et peuvent aussi être affectées par son dynamisme. Pour qu'ils puissent garder leur efficacité, ou au moins survivre ; les SMA doivent s'adapter avec cette dynamique environnementale, en changeant leur organisation vers une autre plus adéquate aux nouvelles conditions d'opération. Cette dynamique doit être maitrisée et contrôlée afin d'assurer que les changements effectués sur l'organisation n'affectent pas l'efficacité du système.

Dans ce chapitre, nous allons essayer de mettre la lumière sur l'aspect évolutionnaire des SMA organisationnels. Nous commencerons par présenter les aspects sociaux dans les SMA et l'utilisation du point vu macro (organisation) pour la spécification et l'implémentation des SMA. Ensuite, nous mettrons plus de lumière sur la notion d'adaptation dans ces systèmes où nous décrirons les différents types, stratégies, et mécanismes de la mise en œuvre d'adaptation et nous terminerons ce chapitre avec une étude comparative des différentes approches proposées et les travaux réalisés pour le contrôle du comportement dynamique des SMA organisationnels.

2. L'Organisation : un terme multidisciplinaire !

Selon Larousse[1] le terme Organisation désigne :

- L'action d'organiser, de structurer d'arranger d'aménager.

[1] www.larousse.fr

- La manière dont quelque chose se trouve structuré, agencé ; la structure elle-même.

En réalité, le terme organisation est un terme multidisciplinaire. Sa définition diffère d'un domaine à un autre :

- La sociologie considère l'organisation comme étant : « *l'agencement de relations entre composants ou individus qui produit une unité complexe ou système, doté de qualités inconnues au niveau des composants ou individus. L'organisation lie de façon inter relationnelle des éléments ou événements ou individus divers qui dès lors deviennent les composants d'un tout. Elle assure la solidarité et solidité relative à ces liaisons, donc assure au système, une certaine possibilité de durée en dépit des perturbations aléatoires. L'organisation donc : transforme, produit, maintient.* » (Morin, 1977)

- En biologie : « *organisation (ensemble de relations conduisant à des transformations de forme donnée) est l'élément qui définit une unité vivante indépendamment de sa structure, de la matérialité au sein de laquelle cette organisation est incorporée* » (Varela, 1989).

- En Sciences de Gestion :« *une organisation est un ensemble de moyens structurés constituant une unité de coordination ayant des frontières identifiables, fonctionnant en continu en vue d'atteindre un ensemble d'objectifs partagés par les membres participants* » (Robbins, 1987).

En revanche, en informatique on parle plutôt d'organisation virtuelle:

- Selon (Desanctis and Monge, 1998) une organisation virtuelle est définie comme étant « *a collection of geographically distributed, functionally and/or culturally diverse entities that are linked by electronic forms of communication and rely on lateral, dynamic relationships for coordination* »

- Tandis que (Foster et al., 2001) l'ont décrit comme « *a set of individuals and/or institutions defined by a sharing of computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering* »

3. L'organisation dans Systèmes multi-agent

Dès leur apparition aux années 80, les SMA ont été décrits comme étant des organisations ou sociétés d'agents *« a set of agents that interact together to coordinate their behavior and often cooperate to achieve some collective goal »* (Ferber et al., 2004). Mais en réalité, le terme organisation d'agents (Organisation) est utilisé dans le littérateur SMA pour désigner deux notions complètement différentes et généralement incompatibles. La première définit l'organisation comme étant un processus (faire organiser un ensemble d'individus) tandis que la deuxième définit l'organisation comme une entité qui a une existence propre (définie explicitement et indépendamment des individus), des exigences et des objectifs. Ce qui en résulte deux points de vue des SMA très différentes l'un de l'autre : SMA centré Agents (ACMAS : Agent Centered Multi-Agent Systems) et le SMA centré Organisation (OCMAS : Organization Centered Multi-Agent systems) (Ferber et al., 2004).

- **Les SMA Centrés Agent (ACMAS) :**

Cette classe aborde la notion de SMA d'un point de vue informatique distribuée & intelligence artificielle, où l'organisation n'existe qu'en tant que phénomène émergent observable (Dignum, 2009). Les travaux ACMAS (Balbo et al., 2002; Bernon et al., 2005; Ferber, 1999; Jennings, 1999; Rao and Georgeff, 1992; Varela and Bourgine, 1992; Wooldridge et al., 2000) ont concentré sur le niveau micro des SMA, exactement sur l'autonomie et l'état mental d'agent (sur les états des agents et les relations entre ces états) comme un moyen d'explication du comportement global observé. Généralement, les SMA développés selon cette approche souffrent de certains inconvénients (Dignum, 2009; Ferber et al., 2004; Jennings, 2000):

- Les patterns et les résultats d'interaction sont de nature imprédictible.
- La prévision de comportement global du système à partir de ceux de ses composants (agents) est très difficile (parfois impossible, car généralement, c'est un comportement émergeant) dû à l'absence d'un contrôle centralisé et d'une structure qui impose aux agents leur comportement.
- Les systèmes développés sont des systèmes fermés, avec des agents homogènes (même type : d'architecture, coordination et comportement).

- **Les SMA Centrés Organisation (OCMAS) :**

Les problèmes rencontrés avec l'approche ACMAS ne peuvent pas être résolus au niveau d'agents (micro), car leurs résolutions nécessitent un niveau d'abstraction supérieur (macro). Selon Jenning dans (Wooldridge et al., 2000), la définition d'un niveau social (organisationnel) est indispensable, non seulement pour réduire la complexité de système et gérer son comportement, mais plutôt pour bien décrire et comprendre le problème envisagé.

À partir de ce point de vue centré sociologie et théorie d'organisation, les SMA ne sont plus décrits en termes des états mentaux d'agents, mais plutôt en termes des concepts organisationnels (groupe, rôle, protocole d'interaction, activité, norme…), reliant le comportement global observé à la structure d'organisation. Par conséquent, l'organisation n'est plus considérée comme un processus, mais un ensemble de mécanismes d'ordre social règlementant l'autonomie des acteurs (agent jouant un rôle) pour réaliser l'objectif global, et permettant d'avoir un certain niveau d'ordre et prédictibilité dans l'environnement.

3.1. Définitions de l'organisation d'agents

Cette métaphore inspirée de l'organisation humaine a donné lieu à plusieurs définitions pour l'organisation d'agents :

- Selon (Coutinho et al., 2009) l'organisation est une société d'entités autonomes. chaque entité est capable d'interagir avec d'autres entités internes (appartenant à la même société) ou externes (appartenant à l'environnement de société) par :
  - ✓ des actions matérielles (avec des entités non autonomes) telles que : production, consommation... etc.
  - ✓ des actions sociales (avec des entités autonomes) telles que : communication, coordination, autorité, délégation, contrôle… etc.
- « *Une entité complexe ou une multitude d'agents interagissent dans un environnement bien structuré pour réaliser certains objectifs globaux* » (Dignum, 2009).
- *« L'organisation virtuelle est une entité sociale composée d'un ensemble d'agents qui exercent différentes fonctionnalités, et qui sont structurés par de modèles de communication et une topologie spécifique, selon un ensemble de normes afin d'achever les objectifs globaux de l'organisation »*(Argente et al., 2011).

Dans le cadre de notre mémoire, nous adoptons la définition de (Esparcia and Argente, 2012) :

« Une *Organisation virtuelle est un ensemble d'agents qui ont besoin de suivre la structure de l'organisation, c.-à-d. ils doivent être distribués selon une topologie donnée, ils doivent jouer un ensemble de rôles (qui indiquent les fonctionnalités que les sont capables de faire), et ils doivent suivre un ensemble donné de normes. En outre, l'Organisation virtuelle est munie d'une fonctionnalité, c'est-à-dire, un ensemble de services (qui sont construits par des tâches), décrivant ce que l'organisation est capable de faire. Toutes les entités qui peuplent l'organisation sont nécessaires pour aider l'organisation d'atteindre ses objectifs (aussi connus comme objectifs organisationnels) »*

3.2. Modèle Organisationnel

Un modèle d'organisation est un Framework conceptuel doté d'une syntaxe de spécification permettant de décrire une spécification organisationnelle de SMA (Coutinho et al., 2009). Une spécification organisationnelle est une formalisation (indépendante de composant) structurelle et fonctionnelle de l'organisation d'agents. Cette formalisation est établie dans deux perspectives complémentaires: statique (description indépendante de l'aspect temps) et cinétique (dynamique; l'aspect temps est pris en considération lors de la description). Généralement un modèle organisationnel permet de capter les dimensions (Coutinho et al., 2009):

- Structurelle : la description de l'aspect structurel et statique (invariant dans le temps) d'organisation. Par la définition des éléments constituants cette organisation (agents ou rôles), ainsi que les relations d'influence qui peuvent exister entre ces éléments (groupe, communication, autorité….).

- Fonctionnelle : représente les objectifs d'organisation à réaliser. Par la spécification des buts et sous buts à réaliser par les agents d'organisation.

- Interactionnelle : décrit dans un point de vue temporel, les actions et les interactions attendues des éléments de l'organisation (définit dans la structure) pour réaliser les buts (définis dans la dimension fonction)

- Normative : permet de contraindre (en termes de lois, normes et contraintes d'ordre social) la relation ternaire qui existe entre les trois dimensions précédentes : structure (statique), interaction (cinétique), fonction (statique).

- Environnementale : représente l'environnement d'organisation ; les entités autonomes (agents, d'autres organisations) ou non autonomes peuplant cet environnement et qui peuvent affecter et/ou être affectées par les activités d'organisation.

- Ontologique : permet de construire la conceptualisation générale de domaine d'application, partagé par les différents agents de l'organisation. Cette conceptualisation permet de maintenir une cohérence dans les activités d'organisation.

- Évaluative : permet de mesurer la performance de la spécification établie (structure, norme… etc.) dans la réalisation des objectifs de l'organisation.

- Évolutive : l'organisation doit être capable de s'adapter avec les changements qui peuvent survenir dans son environnement, cette dimension permet de décrire les changements d'organisation (structure, normes, buts…) issus d'une telle adaptation.

La liste des dimensions décrites ci-dessus représente une description générique des dimensions de modélisation qui peuvent être captées (pas obligatoirement toutes) dans un modèle organisationnel. Les quatre premières dimensions sont les dimensions de base que l'on trouve presque dans tous les modèles organisationnels proposés, en revanche, les restes sont des dimensions complémentaires moins importantes et peu utilisées dans la littérature OCMAS (Coutinho et al., 2009).

L'adoption du point de vue organisationnel par plusieurs chercheurs dans les SMA a donné lieu à plusieurs propositions de modèles organisationnels, nous pouvons citer entre autres: AGR (Ferber et al., 2004), TAEMS (Decker, 1996; Lesser et al., 2004), MOISE+ (Hübner et al., 2002), ISLANDER (Esteva et al., 2002), OperA (Dignum, 2004), OMNI(Dignum et al., 2005), OMACS(Deloach et al., 2008),AUML(Parunak and Odell, 2002) ,ODML (Horling and Lesser, 2005), STEAM (Tambe et al., 1999), et MAS-ML (da Silva et al., 2004) . Chacun d'eux se diffère par les dimensions organisationnelles qu'il capte (structurelle, fonctionnelle, normatives… etc.), la sémantique de la spécification qu'il utilise

et le type des SMA qu'il cible (fermé, ouvert… etc.). La section suivante présente les modèles organisationnels les plus connus.

3.2.1. AGR (Agent, Group, Rôle)

AGR (Ferber et al., 2004) représente l'un des premiers modèles organisationnels proposés dans la littérature. C'est une version évoluée du modèle ALAADIN (Ferber and Gutknecht, 1998) . Il est construit autour de trois concepts principaux *Agent, Rôle, Groupe* (Figure 1.1) :

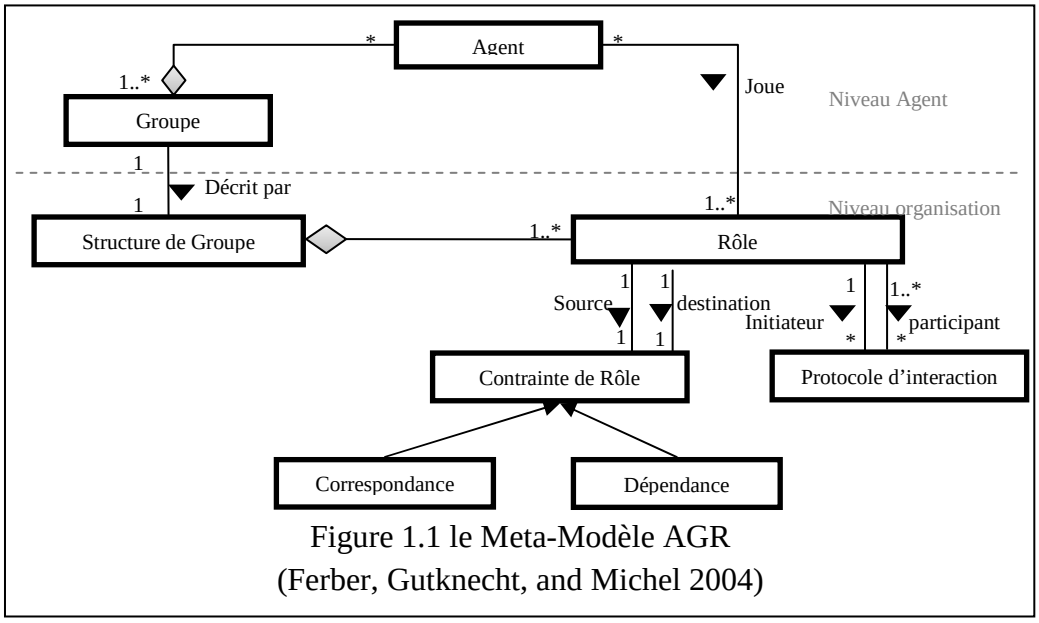


Figure 1.1 le Meta-Modèle AGR
(Ferber, Gutknecht, and Michel 2004)

- Agent : une entité autonome, communicante, qui peut jouer un ou plusieurs rôles dans un ou plusieurs groupes.

- Groupe : Un groupe est l'unité de décomposition d'organisation. C'est une agrégation des agents ayant des caractéristiques communes (activités). Un groupe définit un ou plusieurs rôles à jouer par les agents, ceux-ci ne peuvent communiquer que s'ils appartiennent au même groupe.

- Rôle : décrit une abstraction de: une fonction, une position dans l'organisation ou un comportement attendu de l'agent qu'il y jouera. Un rôle peut être décrit par : un attribut de cardinalité (le nombre maximal des agents pouvant jouer ce rôle) et deux types de relation inter-rôle : (1) *Correspondance* : une correspondance « c » entre deux rôles A et B ($A - c \rightarrow B$) indique que si un agent joue le rôle A, il va jouer automatiquement le rôle B. (2) *Dépendance* : une dépendance « d » entre deux rôles A et B ($A - b \rightarrow B$) défini l'ordre dans la d'adoption des rôles ; avant qu'un agent

puisse jouer le rôle B il doit jouer au premier lieu le rôle A. AGR exige qu'un rôle n'appartienne qu'à un seul groupe, et il ne peut pas être joué que par les agents de ce groupe.

Un autre élément essentiel de la spécification AGR est la possibilité de représenter les liens d'interactions (protocole) entre les rôles, par contre la spécification de logique de protocole d'interaction n'est pas prise en charge par AGR.

L'inconvénient majeur de AGR c'est qu'il n'offre qu'une vue structurelle minimale pour l'organisation de SMA, il ne supporte pas : ni la spécification fonctionnelle ni l'interactionnelle (des aspects laissés ouverts) (Coutinho et al., 2009). Il y a d'autres extensions de AGR comme AGRE (Ferber et al., 2005) .

3.2.2. MOISE$^+$ (Model of Organization for multI-agent SystEms)

C'est un modèle organisationnel qui supporte la spécification explicite de trois dimensions : structurelle, fonctionnelle et déontique (Hübner et al., 2002) :

- Structurelle : elle capte les relations statiques qui existent entre agents, en terme de relations inter-rôles et relation inter-groupes. Les relations inter-rôles sont : *lien de communication* (décrivent la possibilité d'échanger les messages entre deux rôles), *connaissance* (Acquaintance : l'agent jouant le rôle source de cette relation peut récupérer des informations sur l'agent jouant le rôle objet), *Autorité* (relation d'autorité, pouvoir, responsabilité entre rôles) et relation *compatibilité* (l'agent qui a joué le rôle source peut jouer le rôle objet). Tandis que les relations inter-groupes sont de mêmes types que les relations inter—rôles (même sémantique), mais d'un point de vue plus général (entre les rôles de deux groupes).

- Fonctionnelle: décrit les objectifs globaux de l'organisation à l'aide d'un arbre de décomposition de but globale à un ensemble de plans. Chaque plan est une exécution séquentielle, alternative ou parallèle d'un ensemble des buts élémentaires ou d'autres sous plans. Les buts élémentaires sont affectés aux agents sous forme des missions à réaliser.

- Déontique : les deux dimensions précédentes sont liées entre elles par un ensemble de règles déontologiques (obligation, permissions et prohibition) sur les rôles (décrit dans la structure) et les missions à réaliser (définies dans la dimension fonctionnelle).

Une extension de MOISE^{+} a été proposée MOISE^{Inst} (Gâteau et al., 2005), qui vise l'aspect institutionnel dans les SMA (organisation d'agent règlementé par un ensemble de normes et lois institutionnelles) en intégrant la spécification normative .

3.2.3. OPERA (Organizations PER Agents)

C'est plus qu'un simple modèle organisationnel, c'est un Framework de modélisation des organisations d'agent (Dignum, 2004), constitué de trois modèles interdépendants : modèle d'organisation, modèle social et modèle d'interaction.

Le modèle d'organisation (MO): définit les caractéristiques de l'organisation à l'aide de quatre structures : sociale, interaction, normative, et de communication.

- La structure sociale : définit les objectifs et les rôles de l'organisation ainsi les types de coordination qui peuvent exister dans l'organisation.
- La structure d'interaction : décrit les différentes scènes d'interaction possibles entre les rôles d'organisation.
- La structure normative : définit les normes règlementant les activités à l'intérieur d'organisation (comportement d'agent, interaction entre agents)
- La structure de communication : définit l'ontologie utilisée par l'organisation pour la description des concepts du domaine et l'allocution de communication.

Le modèle social (MS) : l'adoption des rôles par les agents est contrainte par un ensemble des contrats sociaux. Un contrat social définit les conditions que l'agent doit satisfaire avant d'assumer un rôle (exigences) et les règles qu'il doit suivre après avoir pris le rôle (ses responsabilités).

Le modèle d'interaction (MI) : permet de décrire la réalisation des scènes d'interaction (définis dans MO) par des agents jouant des rôles selon les contrats sociaux (définis dans MS).

3.2.4. OMACS (Organization Model for Adaptive Computational Systems)

Ce modèle a été proposé (Deloach et al., 2008) pour la spécification de l'organisation dans les SMA adaptatifs. Il permet de définir et de capter les connaissances concernant la structure organisationnelle du système et les capacités de ses agents. Ces connaissances seront utilisées par l'organisation, afin de s'adapter aux changements qui puissent survenir

dans son environnement ou dans les capacités des agents. OMACS définit l'organisation par: agents, rôles, buts, capacités, contraintes organisationnelles (Policies), et l'ensemble des affectations courantes (agent, rôle, but). Chaque rôle est défini pour réaliser un ou plusieurs buts d'organisation, l'affectation d'agent au rôle est contrainte par la possession de l'agent les capacités exigées par le rôle.

Dans le cadre de réalisation de l'objectif global du système, l'organisation OMACS est censée trouver les meilleures affectations (agent, rôle, but) possible permettant la réalisation des buts de manière efficace et optimale. OMACS définit aussi un ensemble de contraintes organisationnelles pour contraindre le comportement global de l'organisation (en tenant compte de l'adaptation) et le comportement de l'agent.

Dans la littérature, plusieurs autres modèles existent. En effet, la liste des modèles organisationnels proposés est longue, chacun d'eux se diffère par les dimensions captées et la sémantique de la spécification qu'il utilise. La Table 1.1 représente une description comparative des différents modèles organisationnels les plus connus (selon la dimension de spécification supportée; « + » la dimension est prise en charge, « - » sinon). Cette Table est prise de (Coutinho et al., 2009), c'est l'un des rares travaux qui ont étudié et évalué les modèles organisationnels proposés .

	Dimensions Modélisées							
Modèle	Structure	interaction	Fonction	Norme	Environnement	Évolution	Évaluation	Ontologie
AGR	+	+	-	-	-	-	-	-
TEAMS	-	-	+	-	+	-	+	-
MOISE+	+	-	+	+	-	+	-	-
ISLANDER	+	+	-	+	-	-	-	+
OPERA	+	+	+	+	-	-	-	+
AGRE	+	+	-	-	+	-	-	-
MOISE ^{inst}	+	-	+	+	-	+	-	-
ODML	+	-	-	-	-	-	+	-
STEAM	+	-	+	-	-	-	-	-
MAS-ML	+	+	+	+	+	-	-	-

Table 1.1 Comparaison des différents modèles Organisationnels
(Coutinho et al., 2009)

4. L'adaptation

L'une des raisons derrière la définition d'organisation (micro ou macro) des SMA est d'avoir une structure stable, capable de supporter la coordination et l'interaction entre agents afin qu'ils puissent réaliser leurs objectifs de manière efficace (Dignum, 2009). Cependant, le système et son environnement ne sont pas stables : la population d'agents peut changer

(migration), les objectifs peuvent changer, la nature dynamique et l'incertitude de l'environnement… etc. Ceci peut altérer l'efficacité du système. À cet effet, le système doit être assez flexible pour qu'il puisse s'adapter, prendre en considération et tirer davantage (si possible) de ces changements.

De manière générale, la notion d'adaptation SMA se réfère à la capacité d'un système à changer son organisation, par une autre plus adéquate aux nouvelles conditions d'opération, dans le but de garder son efficacité, ou simplement survivre. En d'autres termes, l'adaptation SMA est une adaptation organisationnelle où le type et le niveau (micro ou macro) de changement d'organisation dépendent fortement de point de vue utilisé pour décrire celle-ci (ACMAS ou OCMAS). On distingue deux façons d'adaptation SMA :

- **Auto-organisation** (ACMAS) : « *est un processus endogène (par les agents appartenant au système), ascendant (émergeant), concernant les systèmes dans lesquels seules des informations et représentations locales sont manipulées par les agents inconscients de l'état de l'organisation dans sa globalité. Afin d'adapter le système à la pression environnementale en modifiant indirectement l'organisation, donc en changeant directement la configuration du système (topologie, voisinages, influences, différenciation), ou l'environnement du système, par des interactions et propagations locales, en évitant le biais de modèles prédéfinis* » (Picard et al., 2009).

- **Réorganisation** (OCMAS) : « *est un processus endogène ou exogène (par une entité, humain ou agent, hors du système), concernant les systèmes dans lesquels l'organisation est explicitement manipulée au travers de spécifications, des contraintes ou autres moyens, afin d'assurer un comportement global adéquat, lorsque l'organisation n'est pas adaptée. Les agents étant conscients de l'organisation, ils sont capables de manipuler des primitives afin de modifier leur environnement social. Ce processus peut-être à la fois initié par une entité externe au système ou par les agents eux-mêmes, en raisonnant directement sur l'organisation (rôles, spécification organisationnelle) et sur les schémas de coopération (dépendances, engagements, pouvoirs)* » (Picard et al., 2009).

Dans ce mémoire nous nous sommes intéressés au processus d'adaptation dans les SMA organisationnels dont l'aspect organisationnel est celui d'OCMAS. Nous utilisons dans le reste de ce mémoire le terme d'adaptation pour se référer au processus de réorganisation de SMA.

4.1. Déclencheurs de Réorganisation :

Le processus de changement d'organisation (réorganisation) se déclenche à cause de la présence d'une ou plusieurs forces stimulatrices. Selon (Argente et al., 2013; Esparcia and Argente, 2012) ces forces peuvent être soit d'origine interne ou externe à l'organisation:

Les forces externes: ce sont les déclencheurs provenant de l'environnement d'organisation, généralement la source de ces déclencheurs peut être d'autres organisations, agents, ou entité non autonome peuplant ce même environnement. Parmi ces forces on a (Argente et al., 2013) :

- Obtention de ressource : si une organisation ne parvient pas à obtenir des ressources nécessaires, elle doit s'adapter avec une telle situation (ex. changer la façon d'obtention) pour qu'elle puisse survivre (Aldrich, 1999)

- La demande de marché : dans une organisation productrice, les demandes de produits et services peuvent changer avec le temps. Par conséquent, l'organisation doit s'adapter avec la cadence et le type des produits demandés (Aldrich, 2008).

- Généralisation : certaines organisations qui sont spécialisées dans une gamme limitée de produits ou services et qui sont incapables de réaliser ses objectifs (bénéfices) peuvent survivre en se généralisant, c'est-à-dire en offrant des produits et des services plus généraux (Esparcia and Argente, 2012);

- la détérioration : une organisation peut être affectée par les changements environnementaux qui peuvent rendre ses objectifs, produits ou services obsolètes, inadéquats, perdre leur sens… etc. (Aldrich, 2008).

- Changements technologiques : une organisation productrice doit adopter de nouvelles technologies pour améliorer sa productivité à l'intérieur du marché (environnement) où elle opère (Barnett and Carroll, 1995).

- Compétence: organisations doit s'adapter pour acquérir de nouvelles compétences, fournies par d'autres organisations (généralement ayant le même objectif) (Barnett and Carroll, 1995).

- Les caractéristiques démographiques : les organisations ouvertes sont souvent soumises à des questions de contrôle et de gestion des changements continues de

l'ensemble d'agents peuplant l'organisation et son environnement. L'organisation doit s'adapter pour gérer et contrôler ce dynamisme d'une manière efficace (McShane et al., 2003)

- Les lois et règlements : non seulement l'organisation est soumise à des lois et règles internes (organisationnelles), mais il peut y avoir aussi des lois externes provenant de l'environnement (environnementales) qui pourraient influer sur le comportement de l'organisation (Barnett and Carroll, 1995).

Les forces interne : ce sont les déclencheurs provenant de l'intérieur de l'organisation elle-même (changement d'objectif, exigence…) parmi ces forces on a :

- Croissance : lorsque la population d'une organisation croît, il est nécessaire de changer la structure d'organisation (ex., une organisation plus hiérarchisée) pour prendre en considération les nouvelles relations (statique et/ou dynamique) qui peuvent se développer (Aldrich, 2008).

- Pouvoir et facteurs politiques : Les membres les plus puissants de l'organisation (jouant des rôles dirigeants) peuvent avoir des objectifs différents à des agents dans un niveau hiérarchique inférieur, et qui peuvent différer même des objectifs de l'organisation. L'organisation doit s'assurer (par la restructuration, la réaffectation des rôles..) que les agents n'imposent pas leurs objectifs au détriment de ceux de l'organisation (Aldrich, 2008).

- Réalisation d'objectifs: Il y a certaines organisations qui disparaissent après avoir atteint leurs objectifs. Cependant, d'autres organisations cherchent de nouveaux objectifs à atteindre pour survivre.

- Ressources humaines: Les gestionnaires de l'organisation doivent contrôler que leurs agents sont engagés avec l'organisation, offrant un comportement adéquat et que leur performance est acceptable (Argente et al., 2013).

- Gestionnaires de comportement : Les conflits de travail entre les agents et leurs superviseurs au sein des organisations sont des facteurs importants de changement. Si un agent subordonné n'est pas d'accord avec son superviseur, ce dernier pourrait demander de nouvelles tâches à développer à l'intérieur de l'organisation (Argente et al., 2013).

- Restrictions économiques : certaines organisations exercent leurs activités pour maximiser leur performance et leur bénéfice tout en consommant le moins possible de ressources. Si une organisation détecte qu'elle a consommé trop de ressources, elle peut s'adapter en améliorant son rapport production/consommation (Argente et al., 2013).

- Fusion et acquisition d'organisations : l'une des forces internes qui entraînent le changement organisationnel est la fusion de deux ou plusieurs organisations, ou l'acquisition d'une organisation par un autre, ce qui conduit à la reconstruction d'organisation plus grande (Argente et al., 2013).

- Crise: Si une organisation est en crise en raison d'une baisse de son efficacité, elle doit modifier ses éléments structurels et/ou fonctionnels pour améliorer son efficacité (Esparcia and Argente, 2012).

À noter que les déclencheurs d'adaptation sont spécifiques au domaine d'application, ils se diffèrent d'un système à un autre. Les forces décrites ci-dessus représentent une description générique des quelques déclencheurs (interne et externe) potentiels d'adaptation.

4.2. Type de changement

Après la manifestation d'une force déclencheuse d'adaptation, l'organisation peut décider de lancer un processus de réorganisation. Le type et le résultat de ce processus peuvent se différer selon : le type de déclencheur, le moment, la nature d'organisation elle-même (certaines organisations sont plus vulnérables au changement que d'autres)… etc. Selon (Dignum, 2009; Dignum et al., 2004) les résultats d'un tel processus peut-être des changements structurels et/ou comportementaux :

Changement structurel : le résultat d'un processus de réorganisation peut être la modification d'un ou plusieurs éléments structurels d'organisation tels que : rôle, norme, topologie, dépendance, interaction… etc. Dans ce type de changement, on peut distinguer entre deux façons d'application de changement :

- Auto-structuration (auto-organisation (Dignum, 2009)) : une variation dynamique de société émergente, en agissant sur le niveau local (structure locale) pour changer l'organisation globale. Dans ce cas, la réorganisation se fait par le changement d'interaction entre agents et relations locales d'agent (position).

- Restructuration (adaptation structurelle (Dignum, 2009)) : la réorganisation Top-Down se fait par l'ajout, suppression ou modification explicite d'un ou plusieurs éléments structurels.

Changement comportemental : dans cette situation la structure organisationnelle reste invariante, les changements se font au niveau d'aspects comportementaux tels que le rôle joué par un agent (réaffectation des rôles), changement de protocole d'interaction (l'abstraction d'interaction spécifiée dans la structure est invariante).

Les changements comportementaux sont généralement temporels et n'affecte pas les activités futures de l'organisation, tandis que, les changements structurels sont des changements à long terme ce qui nécessite un niveau de raisonnement social élevé au niveau de l'organisation, pour pouvoir raisonner sur : le comportement actuel, le comportement désiré, la différence entre eux, et les conséquences de ces changements… etc. (Dignum, 2009)

4.3. Le processus de réorganisation :

Un processus de réorganisation peut être mis en œuvre de façon centralisée par une seule entité responsable ou décentralisée par plusieurs entités, et qui peut être soit (Picard et al., 2009):

- *Endogène :* accompli par un agent organisationnel particulier (centralisé) ou les agents d'organisation eux-mêmes (approche décentralisée et coordonnée);
- *exogène* : accompli par un utilisateur de système, un agent ou autre système externe à l'organisation concernée.

De manière générale, un processus de réorganisation est articulé autour de 4 phases principales (Alberola Oltra, 2013; Alberola et al., 2011):

- **Surveillance**: la phase de surveillance de comportement d'organisation et d'environnement pour la détection des situations critiques, dans lesquelles l'organisation actuelle n'est plus adéquate aux conditions d'opération actuelle (changement d'environnement) ou aux exigences d'organisation (performance actuelle n'est pas satisfaite, changement d'objectif). En d'autres termes, cette phase vise à détecter la manifestation d'une ou plusieurs forces déclencheuses (internes ou externes) de réorganisation.

- **Conception:** une fois, la réorganisation est nécessaire, l'entité responsable de la réorganisation tente d'élaborer un ensemble d'organisations alternatives, en analysant les conditions de déclenchement de la réorganisation, l'organisation actuelle et l'exigence du système (état désiré). La conception de l'ensemble d'organisations alternatives peut être faite : soit à partir d'une recherche dans une bibliothèque d'organisation prédéfinie à l'avance ou bien par un processus de construction d'organisation à la demande (Hübner et al., 2004), basée sur le modèle de spécification d'organisation et un ensemble d'outils et d'heuristiques prédéfinies.

- **La sélection et l'implémentation** : cette phase consiste à sélectionner et forcer l'implémentation d'une seule solution parmi ceux élaborés dans la phase précédente. Les problèmes majeurs de cette phase sont la définition des critères de choix (une solution parmi plusieurs), et la minimisation de coût d'implémentation de la nouvelle organisation (conséquence d'un tel changement sur l'activité courante et future des agents et du système).

- **L'évaluation :** cette phase est souvent négligée dans la plupart des processus de réorganisation. Elle est exécutée par l'entité responsable sur l'adaptation après le déploiement de la solution sélectionnée, elle vise à évaluer la qualité de la nouvelle organisation ainsi à vérifier si le coût de réorganisation est acceptable par rapport à l'efficacité de cette organisation. Ces informations et d'autres seront collectées pour évaluer la stratégie globale de réorganisation, améliorer les futures réorganisations et aussi pour trouver une organisation alternative lorsque celle déployée ne repend pas à l'exigence du système (Alberola Oltra, 2013).

4.4. Mise en œuvre de processus de réorganisation:

Nous avons présenté dans les sections précédentes les déclencheurs de réorganisation, ces déclencheurs peuvent survenir dans n'importe quel moment du cycle de vie d'une organisation. Les stimules et les réactions après un déclencheur peuvent se différer d'une organisation à une autre, d'un état à un autre dans une même organisation. Le problème qui se pose est : qu'est l'acteur initiateur de la réorganisation, et qu'elle est le degré de la réorganisation ; qui peut varier d'une simple réorganisation d'instance interactive, à un changement drastique de la structure organisationnelle. D'un point de vue de la théorie organisationnelle, la manœuvre d'adaptation peut-être soit (Dignum, 2009):

- *réactive* : qui répond automatiquement à l'évènement déclencheur d'adaptation par un processus de réorganisation prédéfini (défini à l'avance pour faire face à un changement imprédictible dans le futur). Ce genre de stratégies est réalisé par des simples agents organisationnels capables de sentir et réagir vite aux évènements déclencheurs. Généralement, l'approche réactive génère des changements (plus ou moins profonds) portant sur les éléments structurels de l'organisation.

- *Proactive*: contrairement à la stratégie réactive, celle-ci tente de faire des ajustements et tirer davantage de l'évènement produit (si possible). Ce type de stratégies nécessite un mécanisme de raisonnement (implémenter par entité responsable sur la réorganisation) sur l'état courant d'organisation, l'état désiré et les actions à engager, pour déterminer le processus d'adaptation adéquat. Généralement, ce genre de stratégies donne lieu à des changements comportementaux, ou à des changements structurels à simple effet (pas de changement drastique tel que la redéfinition des rôles, relations inter-rôles, interactions).

Selon cette classification, la décision d'initier un processus de réorganisation peut être statique ou dynamique : dans le premier cas (*les réorganisations statiques*), le processus se déclenche automatiquement lors de la présence de certains évènements (critères prédéfinis lors de la spécification de l'organisation). Le processus est statique, il est défini à l'avance (planifié et exprimé) par le concepteur, il comporte seulement deux phases : surveillance et implémentation (Picard et al., 2009). Il est issu d'une phase d'évaluation et prédiction de comportement d'organisation (lors de la conception de système), à l'aide d'outils et des techniques tels que le modèle checking (DeLoach and Kolesnikov, 2006), de prototypage rapide ou de simulation, comme dans IODA (Kubera et al., 2008). Tandis que dans le second cas (*la réorganisation dynamique*), la réorganisation est une conséquence du fonctionnement du système; l'organisation est modifiée au cours de son fonctionnement. Le système ne sait pas quand l'organisation changera, mais il connait les conditions déclencheuses de la réorganisation. Selon (Hübner et al., 2004; Picard et al., 2009), les processus de réorganisation dynamiques peuvent être soient :

- Réorganisation contrôlée (top Down): ici le processus est conduit et contrôlé par une entité responsable (endogène ou exogène). Dès qu'elle détecte que l'organisation n'est plus adéquate (la manifestation d'une ou plusieurs conditions déclencheuses), elle lance la phase de conception soit par la recherche dans une bibliothèque

d'organisation prédéfinie ou bien par un processus de construction spécifique. Ensuite elle force l'implémentation de la nouvelle organisation (structurelle et/ou comportementale)

- Réorganisation émergente (bottom-up) : ce processus est caractérisé par l'absence d'un contrôle explicite sur l'organisation, la réorganisation est dirigée par un ou plusieurs agents d'organisation (endogène) raisonnants et agissants au niveau local pour changer l'organisation globale. Après la détection locale de l'inadéquation d'organisation actuelle, l'agent détecteur exécute une phase de réparation qui consiste à générer et/ou à sélectionner et à exécuter une ou plusieurs actions réparatrices (qu'il les juge nécessaires). Ces actions consistent à changer sa position dans l'organisation (topologie) ou à changer sa spécialisation comportementale (Drogoul et al., 1995). Habituellement, ce type de réorganisation est utilisé pour la mise en œuvre de la réorganisation auto-structurée.

5. Contrôle de la réorganisation

Depuis leur apparition aux années 80, les SMA ont connu un grand succès dans la résolution des problèmes distribués et complexes, ils ont été de plus en plus sollicités pour développer des systèmes dynamiques, plus compliqués, capables d'opérer dans des environnements ouverts, incertains et critiques. Ces systèmes exercent des comportements, de plus en plus complexes, dynamiques et difficiles (voire impossible) à prévoir. Pour maîtriser cette dynamique comportementale, nous étions amenés à contraindre l'autonomie des agents ainsi que leurs interactions en leurs imposant des comportements prédéfinis ou en définissant des contraintes et règles comportementales à respecter ou à éviter (situations indésirables). Ces règles seront soient codées dans le comportement d'agent ou bien formalisées comme des connaissances sur lesquelles l'agent raisonne (Chopinaud et al., 2005). Le problème de ce genre de solutions « sont très rigoureuses », ce qui limite la marge de flexibilité du système et sa capacité d'adapter son comportement pour faire face aux changements environnementaux.

En réalité, pour maitriser les comportements des SMA tout en gardant leurs efficacités, nous sommes obligés à trouver un compromis entre deux problèmes qui sont, dans ce contexte, plus ou moins opposés: l'adaptation et la vérification. D'une part la vérification nécessite d'avoir un comportement prévisible, ce qui exige de diminuer l'autonomie d'agent et du système et, d'autre part, l'adaptation s'appuie sur l'autonomie du

système pour qu'il soit flexible et capable de changer son comportement afin de faire face aux changements environnementaux (Picard et al., 2009) .

L'intégration du niveau social dans la description des SMA (OCMAS) a permis d'avoir un compromis entre l'adaptation et la maitrise du comportement. Car d'une part, l'organisation est un mécanisme d'ordre social permettant de guider les activités des agents et leurs interactions à l'intérieur de l'organisation (rôle, norme, lois, protocole d'interaction…), tout en préservant leurs autonomies (libre de choisir comment : jouer les rôles et accomplir les tâches associées) et, d'autre part, l'adaptation est garantie par la capacité du système de changer son organisation (réorganisation) de manière autonome. En effet, le comportement réorganisationnel de l'organisation ajoute un autre niveau de complexité, dans le comportement global du système, qui doit être pris en considération lors du contrôle de ce dernier; on doit s'assurer que la réorganisation ne diverge pas le comportement global du système vers des situations indésirables.

Le contrôle d'un processus de réorganisation doit s'assurer de la satisfaction de deux propriétés essentielles : *Timeliness* et *resiliency* (rependre au bon moment et de maniéré correcte) (Dignum, 2009) :

- *Timeliness* : correspond au timing d'initiation de la réorganisation; répondre au moment approprié (à ne pas confondre avec la vitesse). En effet, si les changements se produisent trop souvent et rapidement, la prédictibilité et la stabilité du système se diminuent ce qui peut alterner l'efficacité du système (pas de structure stable pour supporter et gérer les activités dans le système). En revanche, si les changements se produisent lentement et tardivement, cela va diminuer la flexibilité du système et le rendre rigide et incapable de suivre le dynamisme environnemental.

- *Resiliency:* décrit la capacité du processus de réorganisation à amener le système rapidement à un état (organisation), stable, durable et en consistance avec la spécification, les normes et les objectifs de l'organisation.

5.1. Positionnement des travaux sur la réorganisation :

Nous présentons dans cette section quelques travaux établis sur la réorganisation, d'un point vu satisfaction des deux propriétés *Timeliness* et *Resiliency.* Ces deux notions de *Timeliness* et *Resiliency* peuvent avoir de différentes appréciations, selon les domaines d'application. Il y en a ceux qui cherchent une réponse rapide applicable même si elle n'est

pas optimale (généralement les adaptations réactives), tandis que pour d'autres, l'optimalité de la solution est un facteur prioritaire par rapport à la rapidité de réponse (les adaptations proactives)(Dignum, 2009).

Le Timeliness : la satisfaction de cette propriété revient à la capacité de la phase de surveillance (de réorganisation) d'initier le processus de réorganisation au bon moment (non pas trop tôt ou trop tard). Généralement, cette propriété est vérifiée et assurée par la spécification préalable (hors ligne) des conditions des déclenchements de la réorganisation. Ensuite, ces conditions seront implémentées dans les entités responsables sur la surveillance soit : (1) sous forme d'évènements à surveiller (en exécution) tels que : le changement d'environnement (Stone and Veloso, 1999), agent incapable de jouer le rôle qui lui est assigné, l'arrivée d'une nouvelle activité où l'organisation actuelle n'est pas adéquate (So and Durfee, 1993; Le Strugeon et al., 1996), détection d'une faute dans les activités d'organisation (Horling et al., 2001)…etc. ; ou bien (2) sous forme des propriétés à vérifier (en exécution) telles que : l'utilité et la performance d'organisation (Bou et al., 2007; Campos et al., 2011; Carvalho et al., 2006; Kota et al., 2012), la capacité d'agent (DeLoach, 2009)…etc.

Resiliency : la satisfaction de cette propriété est la tâche difficile dans le contrôle du comportement adaptatif, car on doit s'assurer de l'habilité du processus de réorganisation à produire des changements :

- Adéquates : aux nouvelles conditions d'opération marquées par le(s) évènement (s) déclencheur (s).
- Correctes: une nouvelle organisation consistante avec la spécification, les normes, les objectifs organisationnels et les exigences du système.

Pour le premier point, la vérification d'adéquation est réalisée par la spécialisation de processus de réorganisations ; généralement, les travaux de réorganisation SMA proposés se focalisent sur un seul type de changement spécifique. Le principe de réorganisation est défini lors de la conception de système : soit pour des changements comportementaux tels que la réaffectation dynamique des rôles (Hexmoor, 2010; Hoogendoorn and Treur, 2009; Nair et al., 2003; De Paz et al., 2014) ou structurels (auto structuration)(Kamboj and Decker, 2007; Wang and Liang, 2006; Weyns et al., 2010) tels que : le changement

dynamique de la spécification des protocoles d'interaction (Artikis et al., 2009) et l'adaptation dynamique des normes(Mahmoud et al., 2013)

En revanche, le contrôle de consistance se diffère selon le mode de mise en œuvre de processus de réorganisation (statique ou dynamique) :

La réorganisation statique : le contrôle de consistance ne pose pas de problème, car l'exactitude de la réorganisation est assurée par le concepteur de système (hors ligne). Le processus de réorganisation ne comporte pas une phase de conception dynamique de réorganisation, il implémente seulement les changements prédéfinis.

La réorganisation dynamique: le contrôle de consistance dans ce genre de processus se diffère selon la façon d'application des changements :

- Réorganisation émergente : la réorganisation émergente (OCMAS) est similaire à l'auto-adaptation dans ACMAS. La seule différence est dans la nature des actions engagées dans l'adaptation de système; dans la première adaptation, ce sont des actions locales d'ordre social (macro), en revanche dans la deuxième ce sont des actions d'ordre individuel (micro). Pour cela, généralement, le contrôle de cette réorganisation recourt à l'ajustement des techniques définies pour contrôler l'émergence dans l'auto-adaptation (ACMAS) tels que la modélisation et l'apprentissage dynamique de l'adaptation organisationnelle (Abdallah and Lesser, 2007; Guessoum et al., 2004) et le contrôle distribué de l'adaptation émergente (Chopinaud et al., 2005; Kota et al., 2012).

- Réorganisation contrôlée : ce genre de changement est plus difficile que soit pour sa mise en œuvre ou pour son contrôle, car on parle d'un niveau de raisonnement macro qui nécessite des agents intelligents capables de raisonner sur l'aspect social de l'organisation: rôles, groupes, normes… etc. Généralement, les travaux proposés pour ce genre de réorganisation sont basés sur des changements à simples effets (prédictible) ciblant un seul aspect d'organisation à la fois : changement comportemental par la réaffectation prédictible des agents aux rôles, changement de nombre d'agents participant aux activités d'organisation, changement d'aspect normatif règlementant les activités dans l'organisation (par exemple par l'activation/ désactivations de normes sociales)… etc. Rares sont les travaux ayant traité le problème d'adaptation contrôlée (réorganisation contrôlée) : il y a ceux qui se basent essentiellement sur l'interaction humaine pour guider le processus de réorganisation

(Carvalho et al., 2006), d'autres se sont basés sur les approches formelles; langage et logique formelle pour représenter, évaluer et évoluer l'organisation (Dignum, 2004), (Dignum and Dignum, 2009, 2012), (Aldewereld et al., 2008).

En réalité, le problème de contrôle de réorganisation (exactement la réorganisation dynamique) n'a pas sollicité beaucoup d'intérêt dans la communauté OCMAS, car généralement, les techniques de la réorganisation ne se basent pas sur l'autonomie d'organisation et d'agent (agent organisateur) comme un moyen de construction dynamique de nouvelle organisation (organisation auto-constructive), dû à la difficulté d'élaborer et de contrôler ce genre de système. Pour cela, la plupart des processus de réorganisation proposés produisent des comportements plus ou moins prédictibles.

En revanche, deux travaux intéressants ont tenté de traiter le problème de la réorganisation contrôlée, il s'agit de MOISE$^+$ et OMACS:

MOISE$^+$: en se basant sur le modèle organisationnel MOISE$^+$, (Hübner et al., 2004) ont proposés un processus de réorganisation exogène, implémenté par une autre organisation (maitresse). Cette organisation est responsable sur l'identification de situations conflictuelles et sur sa réparation dans l'organisation esclave (à changer). L'organisation maitresse est constituée des rôles suivants :

- Un manager : le responsable sur le processus de l'adaptation de l'organisation objet (esclave), il a la permission de changer l'organisation, de gérer toutes les activités des autres rôles et de contrôler le processus complet de la réorganisation.
- Historien: garde l'information concernant l'historique entier du comportement de l'organisation à changer.
- Moniteur : il est chargé d'identifier les situations nécessitantes la réorganisation.
- Expert d'adaptation: responsable sur l'identification du problème courant de l'organisation et de proposer la solution de réorganisation (fonctionnelle ou organisationnelle), en utilisant les informations fournies par l'historien, moniteur, et le manager.

L'approche de réorganisation MOISE$^+$ offre une grande flexibilité d'application, elle peut être utilisée dans plusieurs domaines d'applications (différentes techniques de réorganisation peuvent être mises en fonction du domaine) (Alberola Oltra, 2013). (Hübner et al., 2004) ont

proposé une approche générique pour la réalisation d'un processus de la réorganisation exogène, plutôt qu'une solution appropriée au problème de réorganisation contrôlée, car les phases de réorganisation sont laissées aux concepteurs du système pour qu'ils les implémentent de manière correcte au niveau des agents responsables d'accomplir les rôles : manager, historien, moniteur, expert d'adaptation.

OMACS : c'est l'un des meilleurs travaux sur les SMA adaptatifs. Il propose un processus de réorganisation contrôlé basé sur le modèle OMACS (Organisation Model for Adaptive Computational Systems) (Deloach et al., 2008). Ce modèle permet de capter et de fournir aux organisations les informations nécessaires sur son comportement pour qu'elles puissent s'adapter aux changements de l'environnement (dégradation des capacités d'agents) ou changement interne du système (changement des objectifs). Pour le contrôle de la réorganisation, OMACS offre la possibilité de spécifier (hors ligne) des contraintes comportementales et réorganisationnelles, pour contrôler et guider le processus de réorganisation (en exécution) (Harmon and DeLoach, 2008).

6. Conclusion

Nous avons présenté dans ce chapitre les systèmes multi-agent organisationnels et les différents concepts relatifs à leur développement. En particulier, nous avons mis l'accent sur les différents modèles organisationnels existants dans la littérature ainsi que sur les différents mécanismes et techniques utilisées pour contrôler et développer des SMA flexibles, capables de s'adapter aux changements qui peuvent surgir dans leurs environnements. Parmi les modèles organisationnels présentés, nous avons opté pour le modèle OMACS pour les avantages qu'il procure. Dans le chapitre qui suit, nous présenterons avec plus de détails le modèle OMACS pour le développement des SMA organisationnels et adaptatifs.

Chapiter II : Framework OMACS

1. Introduction

Pour supporter les aspects organisationnels dans les systèmes multi-agents, plusieurs modèles organisationnels pour les SMA ont été proposés dans la littérature. Bien que ces modèles aient apporté des éléments de réponse importants dans le développement des SMA organisationnels, un seul modèle développé qui est capable de représenter, à la fois, l'organisation structurellement, fonctionnellement et de fournir les informations nécessaires pour que cette organisation puisse évaluer son comportement et mette en œuvre de façon autonome des mécanismes d'adaptation et de réorganisation si nécessaire. Ce modèle est appelé OMACS (Oraganizationel Model for Adaptative Computational Systems) (Deloach et al., 2008). Il constitue une base solide pour le développement des SMA organisationnels et adaptatifs. Dans ce chapitre on va présenter ce modèle ainsi que l'ensemble des techniques, outils et méthodologies le supportant.

2. Le Framework OMACS

Scote DeLeatch (DeLoach, 2009) a proposé un Framework pour le développement des systèmes complexes, capables de s'adapter avec leurs environnements d'exécution. Ce Framework permet au système de construire sa propre organisation (se réorganiser) au cours de son exécution. Le composant clé de ce Framework est un modèle organisationnel, appelé OMACS (Oraganizationel Model for Adaptative Computational Systems) (Deloach et al., 2008). Il permet de capter et de fournir au système les informations requises sur sa structure et ses capacités, afin qu'il puisse s'adapter (en exécution) avec les changements de l'environnement et/ou des capacités de ses agents.

2.1. Modèle OMACS

La figure 2.1 représente le modèle organisationnel OMACS. Il a été inspiré à partir du méta-modèle de la méthodologie MaSE (Mutiagent System Engeniring)(Scott and Wood, 2001). Cette dernière est fondue sur le modèle AGR (Ferber et al., 2004).Une organisation selon le modèle OMACS est définie par le tuple $O=\langle G, R, A, C, \phi, P, \Sigma$, achieves, requires, possesses, capable, potential , oaf $\rangle$ (DeLoach, 2009; Deloach et al., 2008) où :

- G, R, A : représentent respectivement les ensembles des Buts (goals), Rôles, Agents.
- C : l'ensemble des capacités des agents.
- ϕ : Une relation sur $A \times R \times G$, qui décrit l'ensemble des affectations <Agent, Rôle, But> courantes.

- P (Policies set) : un ensemble des contraintes ou politiques organisationnelles (voir Section II.3).
- Σ : Modèle du domaine, spécifie l'environnement de l'organisation en termes d'objets, relations entre objets, et les opérations sur ces objets.
- Fonction *achieves*: $G \times R \rightarrow [0..1]$, elle estime l'efficacité d'un rôle R dans la réalisation d'un but G.
- Fonction *capable*: $A \times R \rightarrow [0..1]$, elle estime le rendement d'agent A lorsqu'il joue un rôle R.
- Fonction *requires* : $R \rightarrow P(C)$, elle définit l'ensemble des capacités requises pour jouer un rôle.
- Fonction *possesses* : $A \times C \rightarrow [0..1]$, permet d'évaluer la qualité d'une capacité C d'un agent.
- Fonction *potential* : $A \times R \times G \rightarrow [0..1]$, elle permet d'estimer l'efficacité d'un agent A jouant un rôle R pour réaliser un but G.
- Fonction *oaf* (Organization Assignement Function) : $\phi \rightarrow [0..\infty]$, elle permet d'évaluer la qualité de l'ensemble des affectations suggérées $\Sigma < Agent, Rôle, But >$.

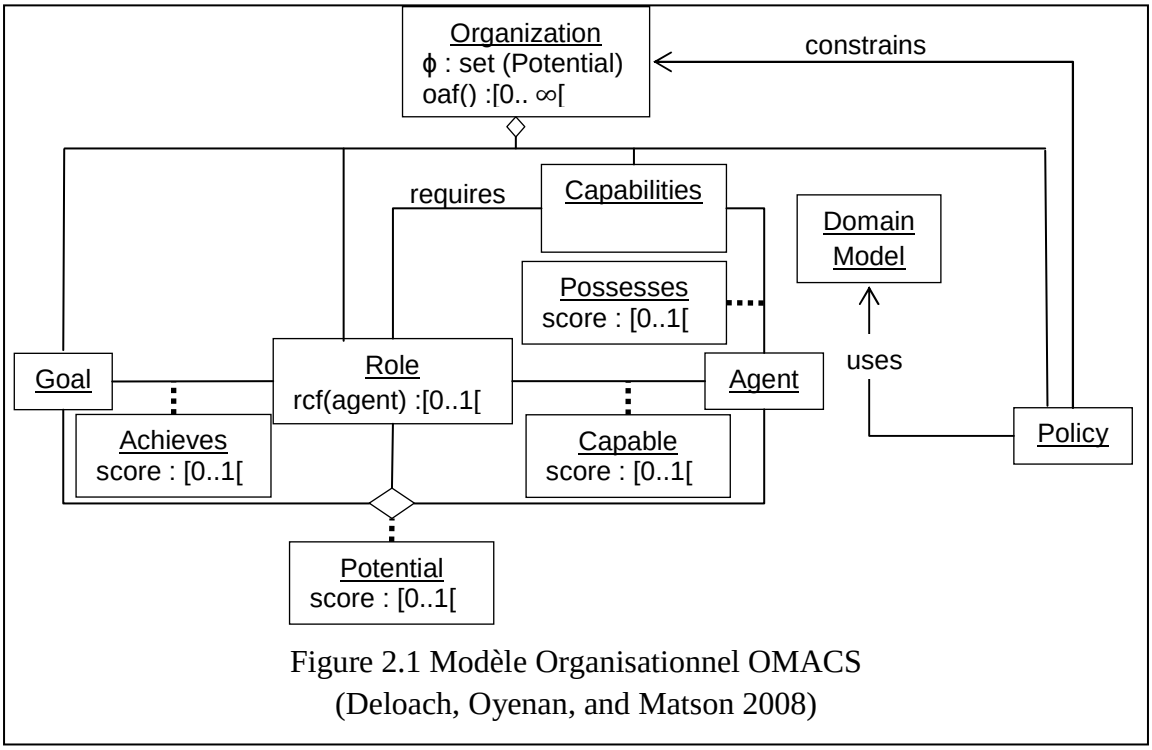


Figure 2.1 Modèle Organisationnel OMACS
(Deloach, Oyenan, and Matson 2008)

Chacun de ces éléments est décrit avec plus de détails ci-après:

Buts (Goals G) : Chaque organisation a un but global qui tente de le réaliser. Selon OMACS, ce but global peut être décomposé (par une séquence d'opérations de raffinement) en sous buts plus simples et plus fins (tâche atomique) dont chacun peut être réalisé par un et

un seul agent à la fois (tout seul). Ainsi, la réalisation de but global revient à la réalisation de tous ses sous buts atomiques (G) (l'ensemble des objectifs de l'organisation à achever). OMACS utilise le modèle GMODS (voir Section II.2.2) pour capter ses objectifs ainsi que leurs interrelations (père-fils, séquencement de réalisation, causalité de déclenchement, …etc.).

Rôles (R) : l'organisation définit un ensemble des rôles (R), qui seront affectés aux agents pour qu'ils réalisent les buts de l'organisation; un rôle définit une position dans l'organisation dont le comportement est censé réaliser un ou plusieurs buts. Chaque rôle est défini par un nom et une fonction *rcf* (*Role Capability function*) ; elle est utilisée pour mesurer la capacité d'un agent à jouer ce rôle.

rcf : c'est une fonction sur A dont la valeur de retour appartient à l'intervalle [0..1] (0 indique la capacité d'agent à jouer ce rôle) (Équation 2.1), *rcf* est définie lors de la spécification du système et calculer en terme des capacités requises pour jouer ce rôle. L'équation (2.2) représente l'équation *rcf* par défaut d'OMCAS (il suppose que toutes les capacités exigées ont la même importance) ; c'est la moyenne des valeurs des capacités possédées par l'agent et requises par le rôle (à condition que l'agent possède toutes les capacités requises (*possesses(A,C_i)>0*)).

$$rcf : A \rightarrow [0..1] \tag{2.1}$$

$$rcf(a) : \begin{cases} if \ \prod_{c \in requires(r)} possesses(a,c) == 0 & 0 \\ else & \frac{\sum_{c \in requires(r)} possesses(a,c)}{|requires(r)|} \end{cases} \tag{2.2}$$

Agents (A): Un agent est une entité intelligente (artificielle ou humaine), possède un ensemble de capacités, capable d'opérer dans des environnements dynamiques et de réaliser les buts qu'ils lui sont affectés. Chaque organisation OMACS contient un ensemble d'agents interactifs (A) (pas nécessairement homogènes).

Capacité (Capability C) : La capacité est une entité atomique, elle décrit une compétence ou une aptitude possédée par un agent; elle peut être une capacité software (algorithme, accès aux ressources et données, …etc.) ou hardware (capteur, effectuer). L'ensemble des capacités (C) dans une organisation est l'union des capacités possédées par les agents et celles requises par les rôles. OMACS utilise les relations *possesses()* et *require()* pour capter respectivement les capacités possédées et exigées par un agent.

Possesses: c'est une fonction sur A et C dont la valeur de retour appartient à l'intervalle [0..1]. Elle indique la qualité d'une capacité possédée par l'agent (la valeur 0 indique que l'agent ne possède pas cette capacité).

$$Possesses : A \times C \rightarrow [0..1] \quad (2.3)$$

Require: c'est une fonction sur R, elle permet de définir pour chaque rôle l'ensemble minimal des capacités qui doivent être possédées par un agent pour qu'il puisse jouer le rôle.

$$Require : R \rightarrow P(C) \quad [2] \quad (2.4)$$

Ensemble des affectations actuelles (ϕ) : c'est un ensemble des tuples agent-rôle-but *<a,r,g>*,où chaque tuple indique que l'agent « a » a été sélectionné par l'organisation pour jouer le rôle « r » afin qu'il réalise le but « g » .

Policies (P) : un ensemble de politiques et contraintes formelles, spécifié pour contraindre le comportement global de l'organisation, ces contraintes se portent sur : les affectations <Agent, Rôle But>, le comportement d'agent ou le processus de réorganisation (Harmon and DeLoach, 2008). Ce concept sera décrit avec plus de détails dans la Section II.3.

Modèle du Domine Σ : les objets existants dans l'environnement de l'organisation ainsi que les relations existent entre eux sont captés à l'aide d'un diagramme de classe (classe d'objet).

Acheives : chaque rôle dans l'organisation est responsable sur la réalisation d'un ou plusieurs buts. La fonction *acheives* associe à chaque paire (rôle, but) une valeur dans l'intervalle [0..1] indiquant l'estimation de l'efficacité de rôle dans la réalisation de but (si la valeur 0 le rôle ne peut pas réaliser le but). La fonction *acheives* peut être prédéfinie (lors de spécification d'organisation) ou bien apprise lors de fonctionnement d'organisation.

$$Acheives : R \times G \rightarrow [0..1] \quad (2.5)$$

Capable : cette fonction permet de calculer l'habilité d'un agent à jouer un rôle, à partir des capacités possédées par l'agent et la fonction *rcf* associée au rôle (*r.rcf()*). *Capable* retourne une valeur comprise entre 0 et 1 indiquant la capacité d'un agent à jouer le rôle (Équation 2.6).

$$Capable: A \times R \rightarrow [0..1] \quad (2.6)$$

[2] P(C) : l'ensemble des parties de C

$$\forall a: A, r: R \;\; Capable(a, r) = r.rcf(a) \qquad (2.7)$$

Potential : l'organisation n'est pas censée de trouver seulement une affectation <agent, rôle, but> réalisable, mais elle est obligée de trouver parmi les combinions (Agent x Rôle x But) possibles, la meilleure d'entre elles. Pour cela OMACS utilise la fonction *Potential* (équation 2.9) afin d'estimer l'efficacité d'un agent jouant un rôle pour réaliser un but.

$$Potential: A \times R \times G \rightarrow [0..1] \qquad (2.8)$$

$$\forall a: A, r: R, g: G \; Potential(a, r, g) = Achieves(r, g) * capable(a, r) \qquad (2.9)$$

Fonction d'assignation d'organisation (oaf) : l'organisation est censée sélectionner le meilleur ensemble d'affectation (ϕ) qui maximisera sa capacité à réaliser ses buts. La fonction *oaf* (*organizational assignment function*) permet d'évaluer la qualité de l'ensemble d'affectionnes (ϕ) sélectionné par l'organisation. Comme dans le cas d *rcf : oaf* est spécifique à l'organisation et au domaine d'application. *Oaf* spécifie une stratégie d'affectation permettant à l'organisation d'achever ses buts de manière efficace et optimale. L'équation 2.11 représente la fonction *oaf* par défaut d'OMACS (c'est la somme des potentiels des affectations).

$$oaf: \phi \rightarrow [0..\infty] \qquad (2.10)$$

$$oaf = \sum_{<a,g,r> \in \phi} Potential(a, r, g) \qquad (2.11)$$

2.2. Le Modèle du but GMODS (Goal Model For Dynamic System)

GMODS (DeLoach and Miller, 2010; Miller, 2007) est constitué de deux modèles : le modèle de spécification (statique) et le modèle d'exécution (dynamique)(Zhong and DeLoach, 2011). Le modèle de spécification permet de capter les objectifs de système ainsi leurs interrelations, tandis que le modèle d'exécution permet de suivre la progression de système lors de son exécution (enchaînement, création et réalisation des buts). Avant de décrire ces deux modèles, nous allons présenter les trois entités principales dans la définition de GMODS : buts, évènements et paramètres (DeLoach and Miller, 2010) :

- Le but représente un état observable et souhaitable du monde (Système et/ou Environnement). GMODS distingue entre : les Classes buts (une modélisation des types des objectifs de système), et l'instance but (qui représente le résultat d'une instanciation d'une classe but lors d'exécution).

- Un évènement est un phénomène observable lors d'exécution. GMODS supporte seulement la spécification des évènements qui peuvent changer l'état des buts du système lors d'exécution, tel que : la création (instanciation), déclenchement, suppression… etc.

- Les buts et les évènements sont paramétrés. Ce paramétrage offre aux agents des informations supplémentaires sur les instances des buts à achever, et les évènements qui surviennent.

2.2.1. Modèle de spécification

C'est un arbre (figure 2.2) de décomposition (conjonctive « et » et disjonctive « ou ») des classes des buts. La racine est le but global du système, les nœuds sont des buts intermédiaires; chaque but intermédiaire sera réalisé automatiquement si seulement un seul (tous ses) fil(s) disjonctif (conjonctifs) est (sont) réalisé(s). Tandis que les feuilles sont les buts atomiques à affecter aux agents. Le but global est jugé atteint si tous ses sous buts sont réalisés. Cette structure arborescente est enrichie par trois types de relations inter-buts (déclencheur, déclencheur négatif, précédence) ;

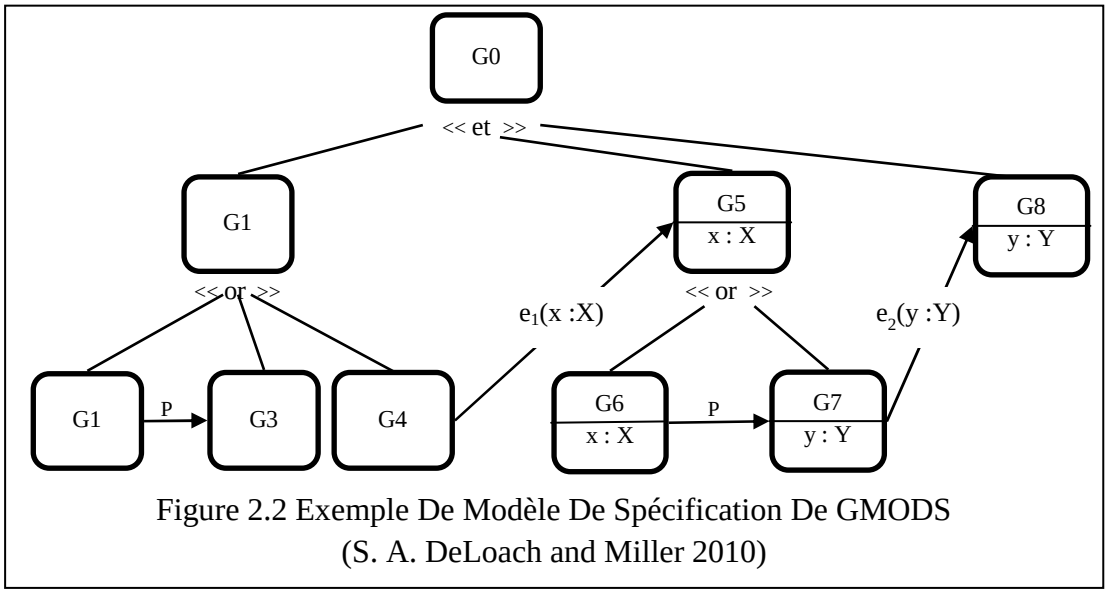


Figure 2.2 Exemple De Modèle De Spécification De GMODS
(S. A. DeLoach and Miller 2010)

- Le déclencheur: un déclencheur entre deux classes de but « *G'* » et « *G''* », basé sur un évènement « *e* » ($G' — e(p_i) — > G''$) définit que : l'occurrence d'un évènement « *e* » (dit évènement déclencheur) lors de la réalisation d'une instance de but « *G'* » provoque la création d'une instance de « *G'* » paramétrées par le paramètre de

l'évènement « $e(p_i)$ » (la figure2.2 inclus deux relations de déclencheur entre (g_4,g_5)et (g_7,g_8)).

- Le déclencheur négatif: un déclencheur négatif entre-deux classes de but « *G'* » et « *G''* », basé sur un évènement « *e'* »($G' \cdots e'(p_i) \cdots > G''$) indique que : l'occurrence d'un évènement « $e'(p_i)$ » (dit évènement déclencheur négatif) lors de la réalisation d'une instance de but « *G'* » provoque la suppression de toutes les instances de but « *G''* » paramétrées par le paramètre de l'évènement « *e* ».

- Précédence: permet d'imposer un ordre total ou partiel dans la réalisation des buts; une relation de précédence entre deux buts « *G'* » et « *G''* » ($G' \ —P— > G''$) assure qu'aucun agent ne puisse travailler sur « G'' » jusqu'à ce que toutes les instances de « G' » soient réalisées (la figure2.2 inclut deux relations de précédence entre (g_2,g_3)et (g_6,g_7)).

2.2.2. Modèle d'exécution:

Le modèle d'exécution de GMoDS supporte l'implémentation de Modèle de spécification et gère les cycles de vie des instances des buts (déclenché, activé, réalisé, échoué, supprimé, abandonné) lors d'exécution. La figure2.3 représente le modèle d'exécution de GMODS: les rectangles sont les états de cycle de vie des instances des buts et les arcs sont les transitions potentielles.

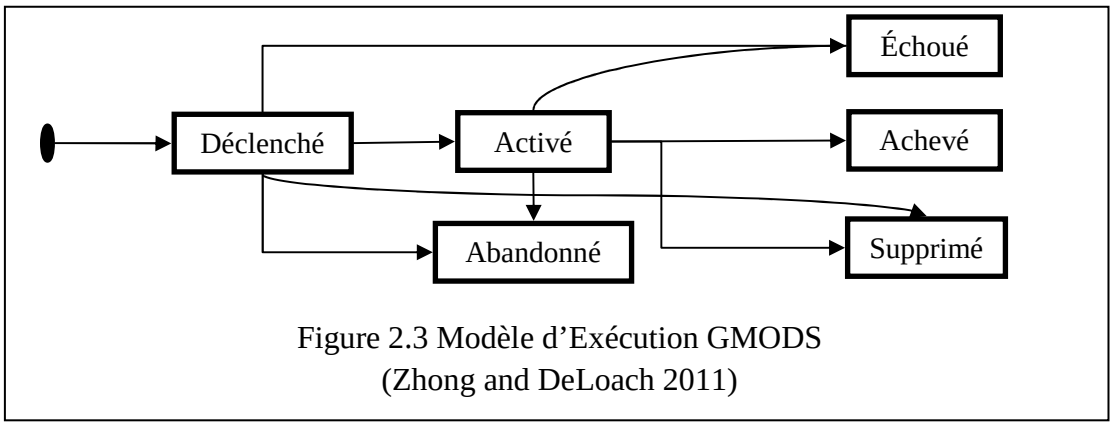


Figure 2.3 Modèle d'Exécution GMODS
(Zhong and DeLoach 2011)

Lors de l'exécution, les buts à réaliser sont instanciés selon l'apparition de certains évènements particuliers; soit les évènements définis dans le modèle de spécification (déclencheurs, et précédence), ou les évènements liés aux états de réalisation des buts (achevé, échoué… etc.).

- État Déclenché: Une fois un but est instancié, il est mis dans l'état déclenché (Trrigered), il reste dans cet état jusqu'il soit activé, échoué, abandonné ou supprimé.
- État activé : Il contient les instances des buts déclenchés et qui ne sont pas précédés par aucune autre instance (relation précédence du modèle de spécification). Les buts activés sont les buts actuels que le système doit réaliser. Ensuite ces buts peuvent se transiter vers l'état achevé, échoué, abandonné ou supprimé.
- État achevé : Il contient les instances des buts activés qui sont réalisés par des agents.
- État échoué : Il contient les instances des buts que le système n'est pas parvenu à les réaliser (impossible de les réaliser).
- État abandonné : Il contient les instances des buts qui ne sont plus revendiqués par le système (qui n'ont pas encore réalisé et qui ne doivent pas l'être).
- État supprimé: Il contient les instances des buts supprimés à cause d'un déclencheur négatif.

Le modèle d'exécution est proposé pour supporter l'implémentation de la spécification fonctionnelle élaborée à partir de GMODS. Chaque système qu'il l'implémente (tels que les Systèmes OMACS) peut l'interroger avec deux opérations: *déclencheur initial* et *survenu* (DeLoach and Miller, 2010).

- Le *déclencheur initial* : permet de marquer le lancement du système et de lancer la création de l'ensemble initial des instances des buts à réaliser (selon le modèle de spécification, les buts qui ne sont pas en attente d'être déclenchés par d'autres buts).
- L'opération *survenu* : permet de lancer la mise à jour de modèle d'exécution (évolué le contenu de ses états) en signalant la présence d'un évènement, que ce soit un évènement lié aux états de réalisation des buts (achevé, échoué) ou un évènement d'application défini dans la spécification (déclencheur, déclencheur négatif… etc.).

Ces deux opérations (*déclencheur initial*, *survenu*) causent particulièrement le changement de l'ensemble des buts activés. Dans les systèmes OMACS, ce changement est vu comme une modification (évolution) des objectifs de l'organisation, sollicitant celle-ci à se réorganiser pour s'adapter avec cette nouvelle situation (Zhong and DeLoach 2011).

2.3. Architecture Organisationnelle des Agents OMACS

Malgré qu'OMACS n'exige pas une architecture spécifique que ce soit pour l'organisation ou pour les agents, mais on présentera dans cette section un exemple d'architecture générale proposée par (DeLoach, 2009), construite autour de la philosophie OMACS. Cette architecture suggère que chaque agent est constitué de deux composants : le composant de contrôle (CC) et le composant d'exécution (EC) (figure 2.4). Le premier est dédié au raisonnement sur l'organisation (macro), tandis que le deuxième est dédié aux raisonnements comportementaux d'agent (spécifique à l'application).

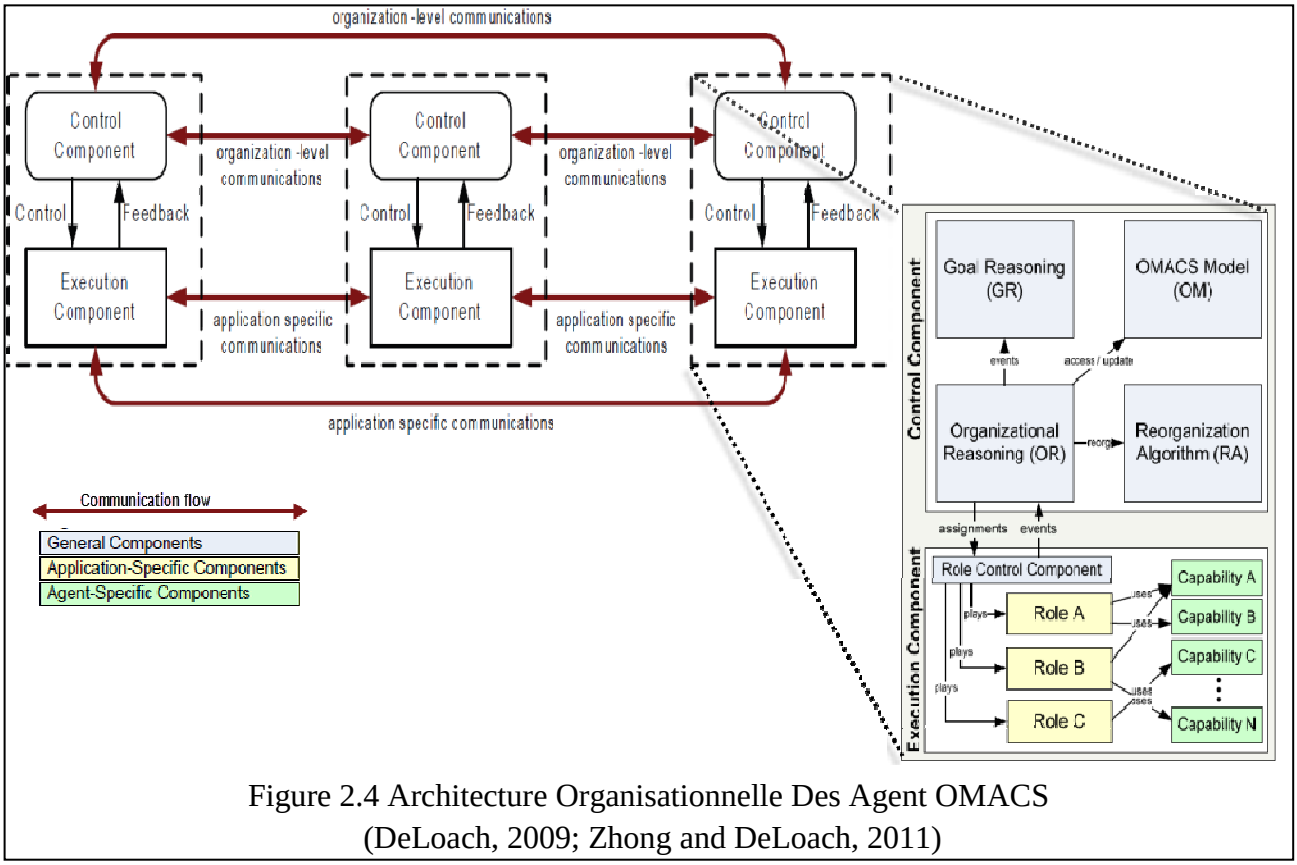


Figure 2.4 Architecture Organisationnelle Des Agent OMACS
(DeLoach, 2009; Zhong and DeLoach, 2011)

Le composant de contrôle (CC) : Pour raisonner sur l'organisation, chaque agent possède une instance de modèle organisationnel, un ensemble d'algorithmes de raisonnement et un ensemble de protocoles de coordination permettant aux différents agents de l'organisation de coopérer et de partager les différentes connaissances organisationnelles afin de maintenir l'organisation et se réorganiser si nécessaire. Ce composant contrôle le comportement d'agent en déterminant les rôles dont l'agent doit jouer, les buts à satisfaire ainsi que l'ensemble d'agents qui peut coordonner avec eux. CC est constitué de 4 éléments :

- Modules de raisonnement sur les buts (GM) : implémente le modèle GMODS (Modèle d'exécution)
- Le modèle organisationnel (OM) : il contient des connaissances organisationnelles sur l'organisation OMACS courante (les agents disponibles, les buts à réaliser et les affectations courantes…)
- L'algorithme de réorganisation (RA): une fois déclenchée il retourne les nouvelles affectations (Agent, Rôle, But) nécessaires pour l'adaptation de système.
- Module de raisonnement global d'organisation (OR) : il lie les trois éléments précédant entre eux (GR, OM, RA), il joue le rôle d'interface entre le CC d'agent et son CE ou entre, le CC d'agent et les autres CC d'agents. Une fois, un évènement est observé (provenant d'un autre agent ou de CE de l'agent lui-même) le GM est activé pour actualiser l'ensemble des buts organisationnels à réaliser (ajout et/ou suppression), ensuit l'OM est à son tour actualisé (si nécessaire). Si le module OR détecte l'inadéquation de l'organisation actuelle, il peut décider de déclencher la réorganisation. Le résultat de la réorganisation sera communiqué aux autres agents (d'autre CC), et/ou passé au composant d'exécution de l'agent lui-même pour qu'il accomplisse la nouvelle affectation (rôle, but).

Le composant d'exécution (CE) : il correspond à la partie de l'agent spécifique au domaine d'application. Le CE contient un composant de contrôle de rôle (RCC), un ensemble de codes de rôles et un ensemble de capacités. RCC est responsable, d'une part, sur la transmission des évènements d'exécution (réalisation/échec de but, panne, changement d'environnement… etc.) au composant de contrôle d'agent et, d'autre part, sur l'accomplissement de rôle affecté par le CC. Une fois notifié, il déclenchera l'exécution de rôle approprié (le rôle est réalisé soit selon un plan prédéfini fourni au moment de conception d'agent ou bien de manière autonome).

L'architecture présentée ci-dessous est une architecture décentralisée où les différents composants de contrôle travaillent ensemble pour garder la cohérence de l'organisation en constituants un contrôle organisationnel distribué. En revanche, il existe une approche centralisée dérivée de celle-ci dans laquelle, tous les composants de contrôle sont regroupés dans un seul agent maître (Organisateur) responsable sur la gestion de l'organisation et sur l'assurance de son bon fonctionnement. Tous les autres agents de

l'organisation (esclaves) sont constitués uniquement de composant d'exécution, recevant de l'organisateur leurs affectations (Agent, Rôle, But).

2.4. Réorganisation dans un Système OMACS

Une organisation OMACS est censée trouver les meilleures affectations possible <*Agent, rôle, but* > susceptibles de réaliser ses objectifs de manière efficace et optimale (DeLoach, 2009). Mais divers événements peuvent survenir lors de l'exécution d'un système multi-agents pouvant causer la réorganisation du système. En général, la réorganisation est déclenchée lorsqu'un événement survient changeant l'état actuel de l'organisation ou de son environnement. Parmi ces évènements on peut citer :

- La modification de l'ensemble de buts G: par exemple, la réalisation d'un but peut déclencher l'activation et/ou l'annulation d'autre (GMODS). Aussi dans certains cas l'organisation peut décider qu'il n'est plus nécessaire de réaliser certains buts.

- L'échec de réalisation d'un but : lorsqu'agent ne peut pas réaliser un objectif qui lui a été assigné (panne, inefficacité).

- Le changement dans l'ensemble des agents (ex. l'arrivée des agents qui étaient inactifs au début).

- Changement dans les capacités des agents : l'acquisition ou la perte des capacités, augmentation ou dégradation des qualités des capacités déjà possédées. Cela peut changer l'ensemble de rôles qu'un agent peut jouer.

Après la détection d'un évènement déclencheur, l'organisation peut décider de se réorganiser pour qu'elle puisse s'adapter avec ces nouvelles conditions d'opération (marquer par cet évènement). Le processus de réorganisation OMACS consiste à trouver un nouvel ensemble d'affectation (ϕ^{i+1}) plus adéquat à la situation courante que l'ensemble actuel (ϕ^{i}). De manière plus formelle, le problème de réorganisation dans une organisation OMACS revient à trouver une solution à un problème d'affectation sous contraintes; l'affectation des agents aux rôles pour qu'ils réalisent les objectifs actuels de l'organisation de tel sort que cette affectation soit applicable, optimale et respectant un ensemble des contraintes organisationnelles prédéfinies (Politiques ou Polices).

La figure 2.5 décrit l'algorithme général de réorganisation, proposé dans (DeLoach, 2009) pour les systèmes à base OMACS. Il permet d'avoir une solution

optimale (l'optimalité ici réfère à l'organisation avec le meilleur résultat retourné par la fonction *oaf*). À noter qu'il n'existe pas un algorithme de réorganisation standard pour tous les systèmes à base OMACS, car le problème de réorganisation est spécifique au domaine d'application (il diffère d'un système à un autre). Ainsi l'algorithme présenté ci-dessous, décrit simplement la philosophie OMACS, il est inapplicable dans un cas réel (dû à sa performance; on n'est pas obligé de faire une réorganisation totale chaque fois qu'un nouveau but arrive) (DeLoach, 2009).

function reorganize(oaf, Gw, Aw)
1. for each g ∈ Gw
2. for each role r ∈ R
3. if achieves(r,g) > 0 **then** m ← m ∪ ⟨r,g⟩
4. ps ← P_Φ(powerset(m))
5. for each agent a ∈ Aw
6. for each set s ∈ ps
7. if capable(a,s.r) **then** pa ← pa ∪ ⟨a,s⟩
8. pas ← powerset(P_Φ(pa))
9. for each assignment set i from c
10. for each assignment x from pa
11. Φ ← Φ ∪ ⟨x.a,x.si⟩
12. if P_Φ(Φ) is valid
13. if oaf(Φ) > best.score then best ← ⟨oaf(Φ),Φ⟩
14. return best.Φ

Figure 2.5 L'Algorithme Général De La Réorganisation (S. DeLoach 2009)

L'algorithme démarre avec les ensembles courants des agents (A_w) et les buts à achever (G_w). Il va créer toutes les affectations <rôle, but> possibles à partir des buts de G_w et les rôles qui peuvent les réaliser (lignes 1-3). Ensuite, il crée un powerset « ps» (Ensemble des parties d'un ensemble) (ligne 4) de tous les ensembles possibles des paires < rôle, but >, puis il supprime de ce *powerset* tous les sous-ensembles non valides (qui violent les contraintes d'affectation). Dans la troisième étape (Lignes 5-7) il crée les affectations possibles « *pa* », entre les agents de Aw et les paires < but, rôle> dans chaque sous-ensemble de « *ps* » . Puis il supprime les affectations non valides de « *pa* » (qui viole les contraintes affectation), et il crée l'ensemble de tous les ensembles d'affectation possibles « *pas* » (ligne 8). Ensuite dans les lignes 10 à 13, il va calculer pour chaque ensemble d'affectation construit (ϕ_i) la valeur de fonction *oaf*. Enfin, il renvoie l'affectation celle qui a la meilleure valeur *oaf* comme étant la solution optimale (ligne 14).

2.5. La Méthodologie O-MASE

O-MaSE (Organization Multi-Agent Systems Enginnering)(Garcia-Ojeda et al., 2008), une extension de méthodologie Multi-Agent Systems Enginnering (MaSE) (Scott and Wood, 2001), est une méthodologie orientée agent personnalisable c.-à-d.; elle offre aux ingénieurs la possibilité d'instancier des processus de développement orientés agents adaptés à leurs besoins (Garcia-Ojeda et al., 2008).

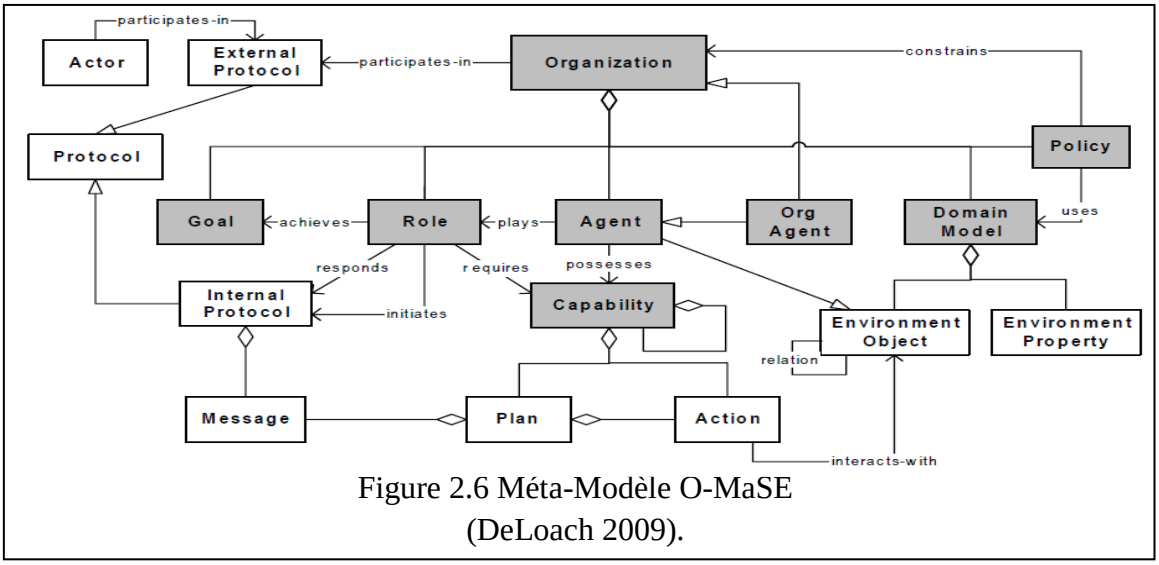


Figure 2.6 Méta-Modèle O-MaSE (DeLoach 2009).

La Méthodologie O-MASE est fondue sur un méta-model OMACS étendu avec des concepts additionnels: (1) Acteur externe utilisé pour désigner tout système ou agent externe à l'organisation (2) Protocole qui est une séquence des messages pouvant existés entre un acteur externe et l'organisation (protocole externe), ou entre les rôles de l'organisation (protocole interne) (3) Plan est l'une des capacités de l'agent, c'est une séquence de messages ou actions qui peuvent être utilisés pour réaliser un but (4) le modèle du domaine O-MASE, qui est décrit par l'ensemble des objets environnementaux (les objets qui appartiennent aux environnements du système) et l'ensemble des propriétés spécifiant les processus gouvernant l'environnement (DeLoach, 2009). La figure 2.6 représente le méta-modèle de la méthodologie O-MASE (les éléments communs avec OMACS sont en gris), cet entendement d'OMACS permet à O-MaSE d'être plus ouvert, capable de prendre en considération non seulement les systèmes OMACS, mais plutôt une large variété des SMA.

Généralement, une instance personnalisée d'O-MASE pour des systèmes basés OMACS est un ensemble de modèles de spécification définissant ce système (Garcia-Ojeda et al., 2008; Oyenan and DeLoach, 2010). La figure 2.7 représente un exemple d'une telle

instanciation (Oyenan and DeLoach, 2010), les modèles O-MASE qui peuvent être utilisés sont (pas nécessairement tous) :

- **Le modèle de but** : permet aux concepteurs de capturer les objectifs du système ainsi que les relations entre buts. O-MaSE utilise GMoDS (Goal Model for Dynamic System) comme modèle de spécification des buts.

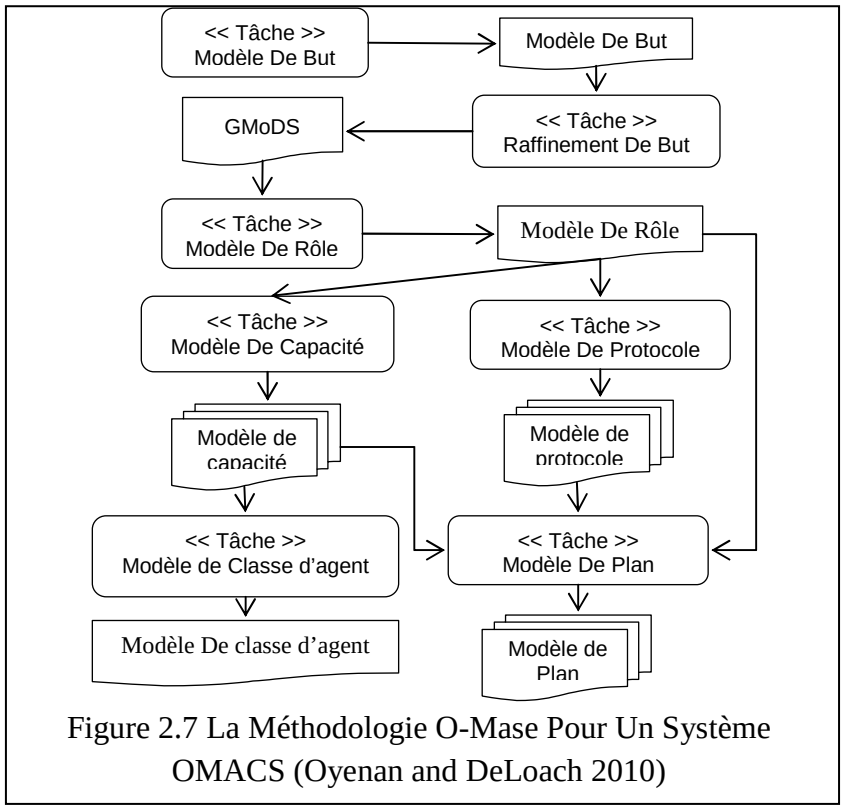


Figure 2.7 La Méthodologie O-Mase Pour Un Système OMACS (Oyenan and DeLoach 2010)

- **Le Modèle de l'organisation** : décrit les relations qui peuvent existées entre l'organisation (système à développer) et les acteurs externes.

- **Le modèle de rôle** : définit les types des rôles qui existent dans l'organisation et les objectifs qu'ils y sont conçus à atteindre.

- **Le modèle du domaine** : spécifie l'environnement du système Multi-agents (en termes d'objets et relations inter-objets).

- **Le modèle de protocole** : il est similaire au diagramme d'interaction AUML(Bauer et al., 2001) , il est utilisé pour décrire les séquences des messages envoyés entre : les rôles, les organisations et les acteurs externes.

- **Le modèle de classe d'agent** : définis l'ensemble des classes d'agents et sous-organisation qui peuplent l'organisation.

- **Le modèle de capacité** : capte les capacités d'un agent, il les définit en termes d'actions à utiliser dans le modèle de plan.

- **Le modèle de plan d'agent** : c'est un automate à états finis, qui décrit comment un agent jouant un rôle peut atteindre un but.

- **Le modèle de politique** : contient une spécification formelle des politiques et contraintes organisationnelles.

O-MASE est supportée par l'environnement de développement Agenttool III (Garcia-Ojeda and DeLoach, 2009a). Il permet à travers son composant AgentTool Process Editor (APE) (Garcia-Ojeda and DeLoach, 2009b) d'élaborer des processus de développement orientés agents fondus sur la méthodologie O-MASE. La Section suivante mettra plus de lumière sur l'environnement AgentTool III.

2.6. AgentTool III (aT^3) :

AgentTool III [3] (Garcia-Ojeda and DeLoach, 2009a) est un successeur d'AgentTool (Scott and Wood, 2001) développé pour supporter la Méthodologie MaSE 2000-2001). C'est un environnement de développement des systèmes multi-agent basé sur la Méthodologie O-MASE. Il a été développé en Java sur la plate-forme Éclipse[4], sous forme d'un ensemble de plug-ins. Il fournir les outils nécessaires pour l'analyse, la conception et l'implémentation des systèmes multi-agent adaptatifs. AT^3 comprend quatre composantes principales: l'éditeur graphique, l'éditeur de processus, Framework de vérification et l'outil de génération de code :

l'éditeur graphique : il permet d'éditer les différents modèles O-MaSE décrits dans la section précédente (modèle de but, organisation, politique… etc.). La figure 2.8 représente l'ensemble des modèles supportés par les éditeurs graphiques d'aT^3.

L'éditeur de processus : AgentTool Process editor (APE) est basé sur EPF (Eclipse Process Framework)[5]. Il permet aux ingénieurs de méthode d'instancier des processus O-MaSE, de les créer et de les maintenir à l'aide de cinq composants de base: une bibliothèque de

[3] http://agenttool.cis.ksu.edu/
[4] http://www.eclipse.org/
[5] http://www.eclipse.org/epf/

fragments des méthodes, un ensemble de directives de construction de processus, l'outil d'édition, un vérificateur de cohérence de processus et un outil de gestion des processus(Garcia-Ojeda and DeLoach, 2009b).

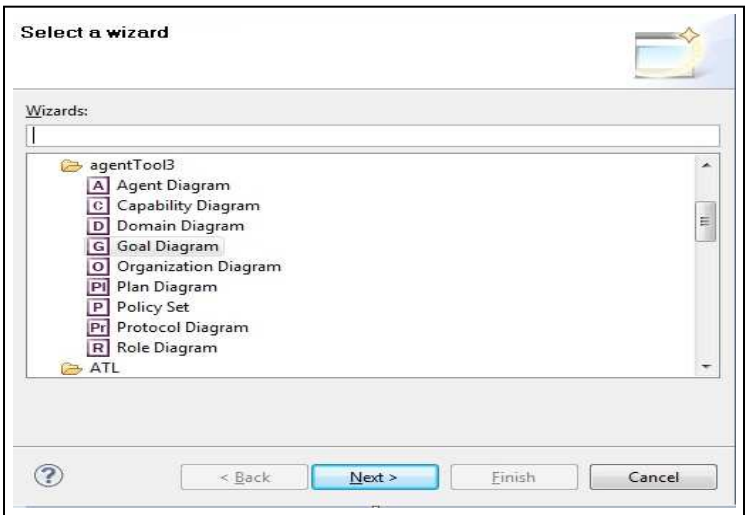


Figure 2.8 Les Model O-MaSE Dans aT^3

Le Framework de vérification : aT^3 offre au concepteur un moyen de maintenir la cohérence entre ses modèles O-MaSE élaborés, à l'aide d'un ensemble de règles de vérification prédéfinies. Puisque les méthodes sont personnalisables, cet ensemble de règles peut également être modifié.

Générateur de code : aT^3 permet la génération automatique de code Jade à partir des modèles O-MASE établis à l'aide des éditeurs graphiques. Dû au niveau de détail des modèles de O-MaSE, le générateur est capable de produire 100 % du code nécessaire pour créer des systèmes JADE fonctionnels (Garcia-Ojeda and DeLoach, 2009a).

À noter que le processus de développent du système à base d'OMACS est intégré de nature dans aT^3 (OMACS et le noyau de O-MASE); à partir d'une édition des modèles O-MASE spécifiant le système à développer (avec les éditeurs graphiques), et la vérification de leur inter-cohérence, aT^3 permettra de générer automatiquement le code Jade nécessaire à l'implémentation du SMA à base d'OMACS.

3. Le Contrôle des Organisations OMACS

Le Modèle OMACS se base sur la notion de Politiques (Policies) pour contraindre le comportement de l'organisation et contrôler son adaptation (réorganisation), de telle sorte que le système multi-agent organisationnel développé soit fiable et prédictible (Harmon and

DeLoach, 2008; Harmon et al., 2008). Ces politiques sont des règles et contraintes d'ordre social, portaient sur : l'affectation des agents aux rôles, le comportement d'un agent jouant un rôle, l'enchaînement de réalisation des buts de système. De manière générale, OMACS définit trois types de contraintes (DeLoach, 2006; Deloach et al., 2008) :

- **Les contraintes d'affectation ou d'assignation (Assignement Policies P_{ϕ})** : la philosophie d'OMACS considère que l'affectation des agents aux rôles afin de réaliser les buts, n'est contrainte que par la possession des agents les capacités requises par les rôles pour réalisent les buts (c.-à-d. Potential (a, g, r) >0). Main dans certains cas (spécifique au domaine d'application), des contraintes additionnelles sur l'ensemble affectations (ϕ) sont requise, par exemple : un agent ne peut jouer qu'un seul rôle à la fois, un agent « A » est empêché de jouer un rôle « R » bien qu'il possède toutes les capacités nécessaires...etc.

- **Les contraintes comportementales (Behavioral Polocies P_{beh})** : décrits comment, les agents de l'organisation doivent se comporter les uns par rapport aux autres, comment un agent doit-il se comporter tout au long de son existence dans l'organisation (par exemple, l'adoption d'un rôle par l'agent peut exiger ou interdire l'adoption d'un autre rôle par le même agent).

- **Les contraintes de réorganisation (Reorganization Polocies P_{reorg})** : ce sont des règles organisationnelles définies pour contrôler l'adaptation de système. Elles sont considérées comme des heuristiques spécifiées pour guider le processus de réorganisation afin qu'il produise de manière optimale et correcte des organisations efficaces. À noter que ce type de contraintes sont spécifiques au domaine d'application; ils peuvent être obtenus à partir d'une connaissance préalable de l'évolution du système et son environnement, ou bien après une phase d'évolution expérimentale du système (algorithme génétique… etc.).

Généralement dans les systèmes multi-agent, ces contraintes organisationnelles réfèrent à des lois et règles rigides. Il est inacceptable de les violer ou de les suspendre une fois activées. Cela peut causer une limitation de la flexible du système, la diminution de sa capacité d'adaptation et l'altération de son efficacité (Harmon and DeLoach, 2008). Pour préserver la flexibilité des systèmes tout en gardant la possibilité de guider et de contrôler leurs comportements, OMACS offre la possibilité de définir ces politiques et contraintes sous deux angles différents:

- **Contraintes Hard (Lois « Law Polices »)** : ce sont les contraintes jugées nécessaires à les maintenir et inacceptables de les violer, car elles permettent de préserver la consistance dans l'organisation, telles que les contraintes de sûreté et de vivacité.

- **Contraintes Soft (Guidance Policies)** : cette catégorie des contraintes offre plus de flexibilité à l'organisation que les celles hard. Elle comporte les contraintes qui ne sont pas essentielles à maintenir. Donc, l'organisation peut les suspendre (le violer) dans certains cas (lorsqu'il n'y a aucun moyen d'atteindre son objectif sans violer la politique) dans le but de garder son efficacité. À noter qu'il est nécessaire de définir une stratégie de suspension de ces règles, de telle sorte que le coût de violation soit minimal (le sous-ensemble minimal de règles à violier pour garder l'efficacité).

Comme nous avons cité précédemment, ces contraintes organisationnelles représentent, généralement, des propriétés à vérifier (nombre courant des rôles joués par un agent) ou des évènements à surveiller (affectation <A, R, G>) tout au long du fonctionnement du système. Pour cela, une approche formelle a été proposée dans (DeLoach, 2002; Harmon and DeLoach, 2008) qui utilise une logique temporelle avec quantification pour spécifier (hors ligne) les contraintes organisationnelles OMACS (Hard et Soft) sous forme des propriétés sur la trace d'exécution du système qui doivent être prises en considération lors de l'implémentation de l'algorithme de réorganisation. L'environnement aT^3 de sa part fournit un éditeur spécifique qui permet, d'une part, de spécifier en logique temporelle quantifiée ces contraintes (hard et soft) et, d'autre part, de définir une stratégie de suspension des contraintes soft (lorsqu'il est nécessaire) basée sur la priorité entre contraintes; les contraintes seront violées avec un ordre ascendant de priorité.

4. Discussion

Le Framework OMACS propose une solution complète pour le développement des SMA organisationnels et adaptatifs. La force de ce Framework réside dans le modèle organisationnel qu'il propose. Il permet de définir des contraintes organisationnelles et réorganisationnelles pour contrôler le comportement global du système (inclus la réorganisation). Cependant, il souffre de deux lacunes importantes:

- **La faiblesse de la logique du formalisme** : OMACS supporte une seule logique pour la spécification des politiques; une logique temporelle avec quantification. Ce

qui rend la tâche de définition des contraintes fortement liée au domaine de cette logique et incapable de définir certaines contraintes qui nécessitent des logiques de formalisation plus fortes, telles que les contraintes de sûreté, de vivacité, où il est plus approprié d'utiliser la logique temporelle passée,future, etc.

- **Absence d'un mécanisme d'imposition des politiques** : malgré que Framework OMACS supporte la spécification des contraintes, il n'existe pas d'instanciation de mécanismes d'imposition ou de vérification de conformité de comportement aux contraintes spécifiées (la version actuelle d'outils aT[3] (1.1.4)[6] ne supporte pas la génération du code à partir de la spécification des contraintes). Ce problème est laissé aux développeurs qui sont censés d'implémenter l'organisation avec un comportement conforme aux contraintes spécifiées, ce qui n'est pas toujours facile, car on parle de systèmes distribués et complexes avec des comportements dynamiques et parfois imprévisibles.

5. Conclusion

Nous avons présenté dans ce chapitre, OMACS, l'un des meilleurs modèles organisationnels proposés pour la modélisation des SMA adaptatifs. Ses points forts ne résident pas seulement dans ses spécificités et sa complétude, mais aussi dans l'ensemble d'outils et techniques et méthodologies développés autour de lui. Malgré l'ampleur de travail fait et le niveau de maturité atteint, nous avons constaté que l'aspect contrôle du comportement de l'organisation n'a pas été bien affiné, surtout au niveau du contrôle du comportement réorganisationnel et cela est dû, d'une part, à la faiblesse de la logique du formalisme qu'il propose et, d'autre part, à l'absence d'un mécanisme d'imposition des politiques. Le chapitre suivant présentera notre approche proposée pour améliorer le Framework OMACS au niveau des lacunes enregistrées.

[6] http://projects.cis.ksu.edu/gf/project/agenttool/frs/

Chapiter III : Approche Proposée

1. Introduction

L'objectif de notre travail est d'instaurer certains niveaux de contrôle global sur les comportements dynamiques des SMA organisationnels, surtout lors d'adaptation (réorganisation), afin de s'assurer qu'ils ne divergent pas vers des situations indésirables ou même chaotiques. Pour ces raisons, nous avons opté pour le Framework OMACS, qui consiste à définir des contraintes et politiques organisationnelles et réorganisationelles pour contraindre le comportement de l'organisation. Cependant, le Framework OMACS ne supporte que la spécification des contraintes, il ne fournit aucun mécanisme d'imposition et d'assurance d'application de ces contraintes. À ce stade nous proposons d'utiliser une approche de Génie logiciel bien connue par sa capacité d'augmenter la fiabilité des systèmes développés; il s'agit de la vérification lors d'exécution (Runtime Verification). Nous présentons dans ce chapitre la solution envisagée pour surmonter les lacunes enregistrées dans le Framework OMACS.

2. Vérification lors d'exécution: Technique et outils

La vérification lors d'exécution (Runtime Verification(RV)) (Havelund and Rosu, 2001; Sokolsky and Viswanathan, 2003; Barringer et al., 2005) est une approche de vérification formelle évolutive (lors d'exécution), largement utilisée pour augmenter la fiabilité des logiciels développés, elle consiste à vérifier dynamiquement (lors d'exécution, parfois après l'exécution) la conformité du comportement du système, à la spécification des besoins préétablie (Meredith, 2012). Généralement, cette spécification comportementale est formalisée sous forme des propriétés et contraintes de sûreté à vérifier. L'une des techniques de RV utilisée est la surveillance dynamique d'exécution (Runtime Monitoring) où un moniteur est généré, et exécuté en parallèle avec le système concerné pour capter, analyser sa trace d'exécution et vérifier si elle respecte les contraintes définies. Dans certains cas, le moniteur permet même de corriger le comportement de système (Jin, 2012; Hussein et al., 2012).

Il y a eu plusieurs propositions des systèmes de surveillance et vérification des traces d'exécution (Runtime Monitoring), ils se diffèrent par le paradigme utilisé pour la mise en œuvre du moniteur (programmation par contrat, Programmation orientée Aspect (AOP) et Runtime Verification) (Meredith, 2012) :

- **Programmation orientée aspect (AOP) :** (Kiczales et al., 1997) le paradigme aspect est largement utilise dans la vérification d'exécution des systèmes, il permet

d'instrumenter le corps de programme en y introduisant: (1) des sondes logiciels pour capter les évènements à surveiller. (2) des portions de code additives pour analyser le séquencement d'apparition des évènements et gérer les anomalies détectées dans les traces d'exécution (enregistrement de situation d'anomalie (log), recouvrement…). Parmi les outils proposés, il y a : Tracematche ,et J-LO (Allan et al., 2005; Bodden, 2005).

- **La programmation par contrat (DBC) :** la programmation par contrat (Meyer, 1988) est une technique orientée objet (OO), qui permet d'ajouter des spécifications sémantiques au programme sous forme des contrats (pré/post conditions, invariantes), ces contrats sont compilés dans le corps de programme sous forme des assertions, afin de vérifier (en cours d'exécution) si les contrats prédéfinis sont respectés. Parmi les outils de surveillance de trace d'exécution à base de DBC on a: jass, JContactor, JML (Java Modeling Langage)(Bartetzko et al., 2001; Abercrombie and Karaorman, 2002; Leavens et al., 2000) ….

- **La vérification lors d'exécution (RunTime verification) :** dans ce paradigme les moniteurs sont automatiquement synthétisés à partir des spécifications formelles des propriétés à vérifier, ensuite ces moniteurs sont déployés hors ligne (débogage) ou en ligne (dynamiquement) dans le but de vérifier si les propriétés sont respectées lors d'exécution. Parmi les outils proposés : MaC, PathExplorer (JPaX), Eagle, RuleR (Barringer et al., 2004; Chen and Roşu, 2003; Havelund and Roşu, 2001)

Approche	Langage	Logique	Champ	Mode	Gestionnaire
Hawk	Java	Eagle	Globale	Inline	Violation
J-Lo	Java	ParamLTL	Globale	Inline	Violation
Jass	Java	Assertions	Globale	Inline	Violation
JavaMac	Java	PTLTL	Class	Outline	Violation
Jcontractor	Java	Contrats	Globale	Inline	Violation
JML	Java	Contrats	Globale	Inline	Violation
JPaX	Java	LTL	Class	Offline	Violation
RuleR	Java	RuleR	Globale	Inline	Violation
Rover	Java, Verilog, VHDL				
Spec#	C#	Contrats	Globale	Inline/Offline	Violation
Temporal	C,C++,	MiTL	Class	Inline	Violation
Tracematches	Java	Expr Reg	Global	Inline	Validation

Table 3.1 Les systèmes de surveillance dynamique (RM) (Meredith, 2012).

Bien qu'il y ait plusieurs propositions d'outils et d'approches, chacun d'eux diffère des autres par : le langage de programmation ciblé (dans lequel le programme à surveiller est écrit), la logique de formalisme utilisée pour spécifier les propriétés à vérifier (ERE ,LTL…) ,le mode d'exécution : inline (tissé avec le programme a surveillé) ,online (exécuté en parallèle), outline (indépendant de programme à surveiller, il reçoit les événements à distance), offline (débogage, vérifier le log de trace d'exécution); le type de gestionnaire d'exception (validation, violation de propriété à vérifier), le champ de surveillance (classe invariante, globale…). Le tableau 3.1 résume les différences entre ces outils et ces systèmes de surveillance (Runtime Monitoring) proposés (Meredith, 2012).

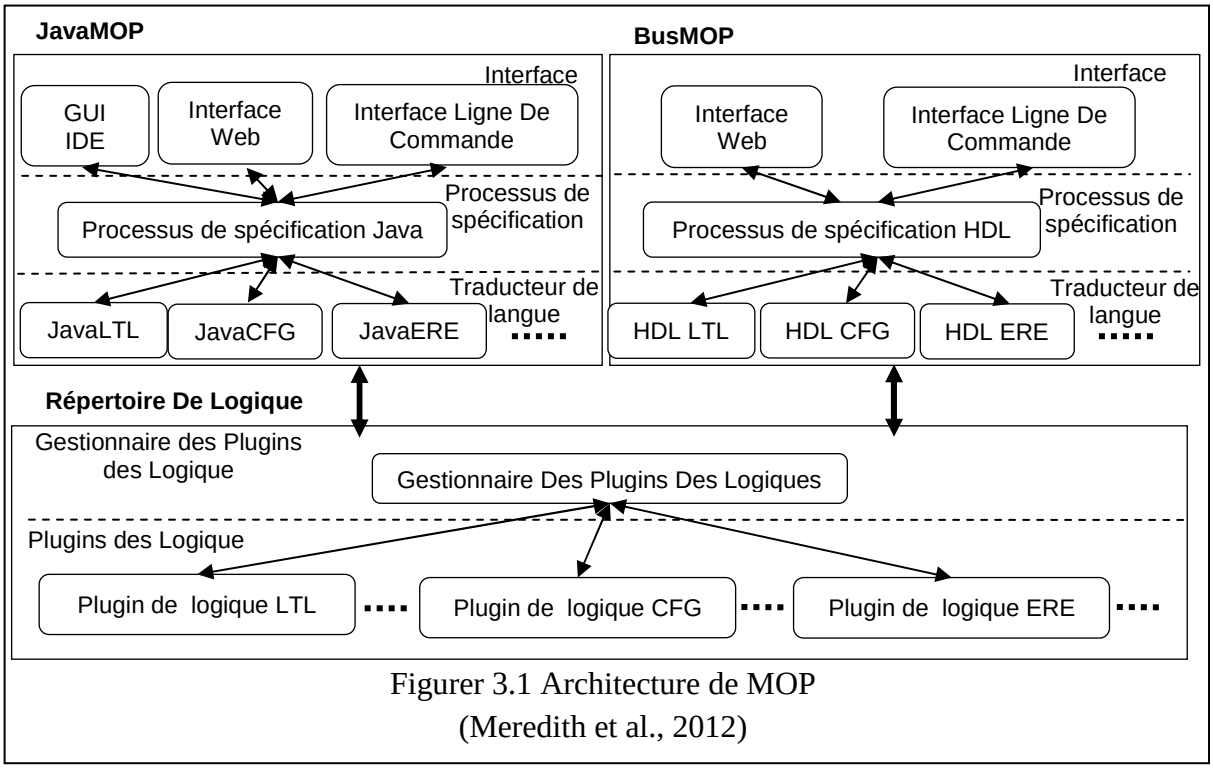
L'inconvénient majeur des outils proposés, c'est qu'ils ne supportent qu'une seule logique de formalisme à la fois, ce qui les rend fortement liées à cette logique et inappropriées pour certaines applications (domaine spécifique)(Meredith et al., 2012). Aussi les techniques de surveillance qu'ils déploient causent des surcharges dans l'exécution du système (augmentation du temps d'exécution, lourdeur… etc.)(Meredith et al., 2012). À cet effet, Un seul Framework est parvenu à traiter ces problèmes, le Framework MOP (Monitoring oriented programming) (Chen and Roşu, 2003, 2007; Chen et al., 2004, 2009; Meredith, 2012; Meredith et al., 2012). C'est un Framework de surveillance (Moniteuring) générique, indépendant des domaines de spécification et personnalisable. La section qui suit mettra plus de lumière sur ce Framework.

3. Le Framework MOP (Monitoring Oriented Programming)

MOP est un Framework générique et extensible permettant à partir d'une propriété spécifiée dans une logique, de synthétiser automatiquement le code moniteur associé à cette propriété et d'intégrer le code moniteur généré dans le code du programme à surveiller. Le rôle du code de moniteur est de vérifier dynamiquement le comportement du système (Meredith et al., 2012) par rapport aux propriétés. Les approches traditionnelles de surveillance se concentrent principalement sur la détection de violations des propriétés. MOP va au-delà, en permettant à l'utilisateur de fournir du code à exécuter lorsque les propriétés sont violées ou validées, non seulement pour la signalisation de l'état, mais pour le recouvrement et la correction de comportement dans le cas où celui-là est inapproprié (ne respecte pas les propriétés). À noter que MOP est un système de surveillance paramétré permettant, non seulement la spécification des simples propriétés portants sur un comportement global (abstrait), mais aussi la surveillance des traces par la définition des spécifications paramétrées sur des propriétés portant sur le comportement des objets

(instances paramétrées)(Chen et al., 2009). La spécification paramétrée permet d'avoir un niveau de surveillance plus précis. Elle permet de vérifier, non seulement le comportement global ou le comportement d'un seul objet, mais aussi le comportement issu de l'interaction entre objets.

3.1. Architecture



Figurer 3.1 Architecture de MOP
(Meredith et al., 2012)

Comme nous avons cité précédemment, MOP est un Framework de surveillance générique et indépendant des domaines de spécification. Il sépare entre la génération et l'intégration de moniteur, il offre à l'utilisateur la possibilité d'instancier une MOP-instance et de la personnaliser avec des logiques de formalisme et un langage de programmation (dans lequel le système à surveiller est écrit) spécifique (Meredith et al., 2012). Cette spécificité de MOP vient de son architecture ouverte et extensible, la figure 3.1 décrit cette architecture ; elle est construite autour de 2 composants principaux : le répertoire de logiques et le langage Client(Meredith, 2012).

Le répertoire de logiques : c'est le composant commun entre les différentes instances MOP. Il contient les différentes logiques supportées par MOP sous forme des plug-ins de logique et un gestionnaire des plug-ins.

- **Plug-in de logique :** Chaque plug-in implémente et encapsule un algorithme de génération de moniteur spécifique à la logique qu'il y engendre. Il permet de générer

à partir d'une spécification de propriété un pseudo-code de moniteur (sous forme d'automate d'états finis). Parmi les plug-ins des logiques on a : automate à état fini (FSM) , expression régulière étendue (ERE), logique temporelle linéaire (LTL), logique temporelle passée/future PTLTL/FTLTL, grammaire à contexte libre (CFG), …etc. (Meredith et al., 2012).

- **le gestionnaire :** il joue le rôle d'intermédiaire entre le langage client et les plug-ins des logiques. Il permet de diriger les requêtes de génération de moniteur provenant de langage client vers le plug-in de logique approprié, et de retourner les résultats de requête (un pseudo-code du moniteur) issue du plug-in au langage client.

Langage Client : la partie spécifique pour chaque instance MOP, il représente le langage de spécification du domaine d'application. Généralement, le nom du langage client correspond au nom d'instance MOP qu'elle l'utilise. Il est responsable sur tous les aspects de surveillance liés au langage qu'il l'engendre tel que : la traduction de pseudo-code en code réel de moniteur, instrumentation du code moniteur dans le programme à surveiller. Le langage Client est composé de trois couches :

- **La couche inférieure** : contient les traducteurs des langages qui sont responsables sur la traduction des codes moniteurs abstraits (pseudo-codes générés par les plug-ins de logique) aux codes moniteurs concrets (dans le langage de programmation spécifique).

- **La couche intermédiaire** : contient le processus de spécification qui, d'une part, extrait les clauses ou les formules des propriétés à partir de spécification et les envoie au répertoire de logique pour qu'il produise le code moniteur associe et, d'autre part, il insère le code concret de moniteur dans le corps du programme à surveiller.

- **La couche supérieure**: elle représente l'interface utilisateur qui permet aux utilisateurs de manipuler l'instance MOP : de spécifier les propriétés à surveiller, de lancer la génération de moniteur et l'instrumentation de son code dans le programme.

Cette indépendance entre les deux composants répertoire des logiques et langage client ainsi la transparence entre la génération et l'instrumentation de moniteur permettent de faciliter, d'une part, la tâche d'élargissement de l'ensemble de logiques du formalisme MOP par d'autres logiques et, d'autre part, l'instanciation des instances MOP personnalisable

spécifique au domaine d'application. Actuellement, il existe deux instances MOP qui ont atteint à un niveau de maturité élevé : Bus-MOP (dédié au bus PCI) (Pellizzoni et al., 2008) et JavaMOP (dédie au programme java)(Chen et al., 2006). Dans le cadre de notre approche, nous avons opté pour JavaMOP, pour surveiller et contrôler le système OMACS. Nous détaillons dans ce qui suit cette instance de MOP.

3.2. JavaMOP

JavaMOP (Chen and Roşu, 2003; Chen et al., 2006; Meredith et al., 2012) est une instance de Framework générique MOP dédiée à la surveillance des programmes java. À partir d'une spécification paramétrée des propriétés à vérifier, JavaMOP génère un code moniteur en AspectJ (Kiczales et al., 2001) prêt à être tissé par n'importe quel compilateur AspectJ dans le programme java à surveiller. JavaMOP offre aussi, à l'utilisateur, la possibilité de fournir de code java à exécuter lors de validation ou violation de spécification.

De manière générale, la spécification dans JavaMOP est une combinaison de la spécification des évènements et des propriétés sur ces évènements (Jin, 2012). Pour définir des propriétés paramétrées sur le comportement d'un programme java, JavaMOP exige:

- Les évènements que le moniteur va utiliser pour construire (capter) la trace d'exécution du programme, ces évènements représentent généralement des appels de méthodes, renvoi d'un résultat par une méthode, accès ou modification d'un champ, création ou destruction d'un objet… etc. JavaMOP utilise une syntaxe étendue d'Aspect, pour définir ces évènements.

- La formalisation des propriétés sur l'occurrence des évènements (identifiés précédemment) dans la trace d'exécution, en utilisant les logiques de formalisme supportées par JavaMOP.

3.3. Syntaxes de spécification

3.3.1. La syntaxe générale de MOP

Le Framework MOP propose une syntaxe de spécification générique, partagée par ses différentes instances. Chaque instance MOP utilise une version personnalisée (selon le langage de programmation ciblé et l'ensemble de logique du formalisme utilisé) de cette syntaxe. La figure 3.2 représente la syntaxe générique de MOP (décrite avec la grammaire EBNF); les non terminant sont entourés par '< 'et'>', '['et']' indiquent la portion de

grammaire qui peut apparaitre une ou plusieurs fois , tandis que '{'et'}' indique la portion de grammaire optionnelle (zéro ou une seule fois) (Meredith, 2012). Une spécification MOP (< Specification >) est constituée de quatre composants principaux :

```
<Specification> ::= <Header>
                        {<Event>}
                        {<Propriety>
                           {<Propriety handler >}
                        } "}"
<Header> ::= {< Instance Modifier>} < Id> <Instance Parameters> "{"
             {< Instance Declaration>}
<Event> ::= ["creation"] "event"<Id>< Instance Event Def >"{" <Instance Action>"}"

< Propriety >::= <logic Name> ":"< logic syntax>

<propriety handler >::="@" < logic state ><Instance Handler >

---

< Instance Modifier> ::=< Id>
<Instance Parameters>::=<JavaMOP parameters>|<BusMOP parameters>|...
<Instance Declaration>::=<JavaMOP Declaration>|<BusMOP Declaration>|...
<Instance Event Def >::=<JavaMOP Event Def >|<BusMOP Event Def >|...
<Instance Action>::=<JavaMOP Action>|<BusMOP Action>|...
<Instance Handler>::=<JavaMOP Handler >|<BusMOP Handler>|...

<logic Name>::=< Id>
```

Figurer 3.2 Syntaxe Générale De MOP

- < **Header** > : l'entête est constitué d'un identifiant de la spécification <Id>, un ensemble de paramètres <Instance paramètres>, et un ensemble de modificateurs <Instance Modifiers>. Les modificateurs sont des identificateurs (spécifiques pour chaque instance MOP) permettant de spécifier les options d'exécution du moniteur (ou des moniteurs ; car il peut y avoir plusieurs moniteurs, instanciés à partir de même code moniteur généré). L'entête contient aussi la déclaration d'un ensemble de variables locales au moniteur (la déclaration de variables dépend de l'instance MOP utilisé, BusMOP ne supporte pas la déclaration des variables).

- < **Event**> : représente la définition des évènements à surveiller (point d'observation) dans l'exécution de système, chaque évènement est défini par le mot 'event' suivi d'un identifiant unique <Id> et une définition (spécifique pour chaque instance) de condition de déclenchement de cet évènement <Instance Event Def>. Chaque déclaration d'évènement peut contenir un ensemble d'actions <Instance Action>

(écrit dans le langage ciblé par l'instance MOP) à exécuter lors de l'observation de cet évènement dans la trace d'exécution (afin de modifier l'état de moniteur ou du programme à surveiller, manipuler les variables locales… etc.). Pour l'indication manuelle de l'évènement déclencheur de la surveillance, MOP propose de marquer cet évènement par le mot 'creation' au début de sa spécification.

- **<propriety>** : chaque spécification MOP peut contenir zéro ou plusieurs propriétés. Chaque propriété est définie sous la forme « <logic Name> : < logic syntax> » où : <logic Name> désigne l'identifiant de la logique de formalisme utilisé, et < logic syntax> la clause de propriété (écrit dans la syntaxe de logique utilisée). Les identifiants des évènements sont utilisés comme des prédicats dans les clauses des propriétés

- **<propriety handler >** : contient la définition de gestionnaire (Handler) de propriété, qui est un ensemble d'actions à exécuter < Instance Handler> lorsque le moniteur atteint certains états de logique de formalisme < Logic State> (match, fail, validate, violationn... etc.).

3.3.2. La syntaxe JavaMOP

JavaMOP personnalise la syntaxe MOP décrite ci-dessus par : (1) l'utilisation du langage java comme langage de description des actions à exécuter (< Instance Handler>, <Instance Action>) et pour la déclaration des variables et paramètres de spécification (2) l'utilisation de syntaxe AspectJ pour la spécification des évènements <Instance Event Def> et (3) la proposition d'un ensemble de modificateurs <Instance Modifiers>. la figure 3.3 représente la syntaxe JavaMOP (Meredith, 2012), les entités personnalisées dans cette spécification sont :

- **<JavaMOP Modifier>** : les modificateurs proposés par JavaMOP sont :
  - les trois modificateurs ***« full-building », « maximal-building », 'any-binding'*** (portant sur l'instanciation des paramètres du moniteur); vu qu'il peut y avoir plusieurs instances de moniteurs qui s'exécutent simultanément (chacun est associé à un ou plusieurs paramètres de propriété) JavaMOP propose trois modificateurs pour spécifier l'instance (ou les instances) de moniteur qui peut exécuter son gestionnaire <propriety handler > (ex., dans le cas de violation d'une propriété). Pour *« full-building » :* seulement les instances avec une

assignation complète de ses paramètres (chaque paramètre est lié à un objet c.-à-d., non nul) qui peuvent exécuter leurs codes de gestionnaire. Par contre pour *« maximal-building »* seulement les instances avec une assignation courante maximale qui peuvent exécuter leurs codes de gestionnaire. En revanche pour *« any-binding »* n'importe quelle instance concernée par l'évènement peut exécuter son code de gestionnaire.

```
<JavaMOP Modifier>::= ”full-binding ”| ”maximal-binding”| ”any-binding”| ”connected”|
                      ”unsynchronized ”| ”decentralized”| ”parthread ”| ”suffix”
<JavaMOP parameters>::= “(“[{<Java Type> <Id>”,”}<Java Type> <Id>}]”)”
<JavaMOP Declaration>::= syntaxe of declaration in Java
<JavaMOP Event Def> ::= <AspectJ advice spec>”:”
                        <AspectJ Pointcut>[“&&”<JavaMOP Pointcut>]
<JavaMOP Pointcut>::=”thread (”<Id>”)” | ”condition (”<Boolean Expression>”)”
                     |<AspectJ Pointcut>|<JavaMOP Pointcut>”&&”<JavaMOP Pointcut>

<AspectJ Pointcut> ::= syntax of Pointcut in AspectJ
<AspectJ AdviceSpec> ::= syntax of AdviceSpec in AspectJ
<Boolean Expression> ::= <Id> | “!”<Boolean Expression>| <Boolean Expression>
                         <Boolean Operator> <Boolean Expression>|
                         “(”<Boolean Expression> “)”

<Boolean Operator> ::= “ || ” | “&&” | “ | ” | “&” | “ == ” | “! = ”
<JavaMOP Action>::=Java statements, which my refer to monitor local variables
<JavaMOP handler>::= Java statements with additional keywords
<Java Type>::= Any valid Java type
```

Figurer 3.3 Syntaxe De JavaMOP (Meredith 2012)

- Le modificateur ***« connected »*** exige que le paramètre assigné (non nul) dans une instance moniteur doive être connecté par évènement actuel (qui a causé le déclenchement de gestionnaire) pour que cette instance puisse exécuter son Handler.

- Le modificateur ***« decentralized »*** pour indiquer le mode d'indexation des instances de moniteur existant (si ce modificateur n'est pas spécifié, le mode d'indexation par défaut est centralisé).

- Un modificateur ***« unsynchronized »*** pour spécifier si les états des moniteurs doivent être protégés contre les accès concurrents (si ce modificateur n'est pas spécifié, le mode est synchronisé).

- Le modificateur ***« parthread »*** si JavaMOP est utilisé pour surveiller des threads, ce modificateur exige au moniteur de surveiller la trace de chaque thread indépendamment des autres.

- Le modificateur ***« suffix »*** pour indiquer le mode de correspondance (matching) entre traces d'exécution et l'occurrence des évènements; correspondance totale ou partielle, le mode par défaut est le mode total.

- **<JavaMOP Parameters>** , **<JavaMOP Declaration> :** la déclaration Java ordinaire des paramètres (comme dans les méthodes) et des variables; (*Type id [,Type id])***.**

- **<JavaMOP Action> , <JavaMOP handler>** : des instructions Java ordinaires , pour définir les actions à exécuter, elles peuvent aussi être utilisées pour manipuler les variables locales des moniteurs. À noter que <JavaMOP handler> étend la syntaxe java par deux variables, et une expression spécifiques **:**

  - **'__RESET' :** une expression spéciale (retourne void) permettant de réinitialiser le moniteur (remise à l'état initial), mais sans modifier les valeurs des variables locales de moniteur.

  - ***'__LOC'*** : une variable de type String contient le numéro de la ligne du code qui a généré l'évènement actuel (causant le déclenchement du gestionnaire « *handler* »).

  - ***'__MONITEUR'*** **:** une variable de type moniteur contient l'instance de moniteur (objet) actuelle. Cette variable permet d'accéder (en lecture/écriture) aux variables locales de cette instance.

- **<JavaMOP Event Def> :** JavaMOP utilise une syntaxe AspectJ étendu (l'extension d'ensemble de Pointcut) pour capter les évènements, ainsi leur contexte d'exécution. Un évènement est défini sous forme d'un Advice (Kiczales et al., 2001) sur la portion de code Java qu'il l'implémente. Les Pointcut additionnels de JavaMOP <JavaMOP Pointcut> sont **:**

  - ***' thread'*** *:* pour capter les threads java; l'identifiant <Id> peut être le nom de classe ou variable de type thread.

- ***'condition'*** *:* elle prend une expression booléenne comme paramètre ; un évènement contenant un Pointcut « condition » ne se déclenche que si l'expression booléenne est vraie. La différence entre ce pointcut et le pointcut traditionnel « if » d'AspectJ, c'est que les variables locales de moniteur peuvent être utilisées dans l'expression booléenne.

JavaMOP représente l'un des meilleurs systèmes de surveillance dynamique proposés jusqu'à présent, et ça dû à ses spécificités :

- Un système de surveillance indépendant de logique de formalisme.
- Sa force d'expression et la richesse de son répertoire de logiques de formalisme.
- La possibilité d'étendre l'ensemble des logiques qu'il supporte par d'autres.
- Le seul système qui supporte les logiques à état non fini, telles que la grammaire à contexte libre CFG (Jin et al., 2012).
- La génération automatique des moniteurs efficaces à partir de spécification.
- Il supporte efficacement la surveillance des propriétés paramétrées.
- Le seul système qui supporte efficacement la surveillance des propriétés paramétrées à contexte libre.
- Le seul qui supporte simultanément les logiques temporelles LTL passées et futures.

4. Approche proposée

Vu les spécificités de JavaMOP présentés dans la section précédente ; surtout en ce qui concerne sa force d'expression et la possibilité de générer automatiquement du code moniteur associé aux propriétés spécifiées, nous proposons d'intégrer JavaMOP dans le Framework OMACS pour contrôler le comportement dynamique instauré dans l'organisation OMACS et surmonter les insuffisances enregistrées dans ce Framework au niveau de spécification et implémentation des politiques organisationnelles et réorganisationnelles. La figure 3.4 représente notre proposition d'intégration de JavaMOP dans l'environnement aT^3. Cette approche de développement des SMA OMACS avec adaptation contrôlée est constituée de 5 phases :

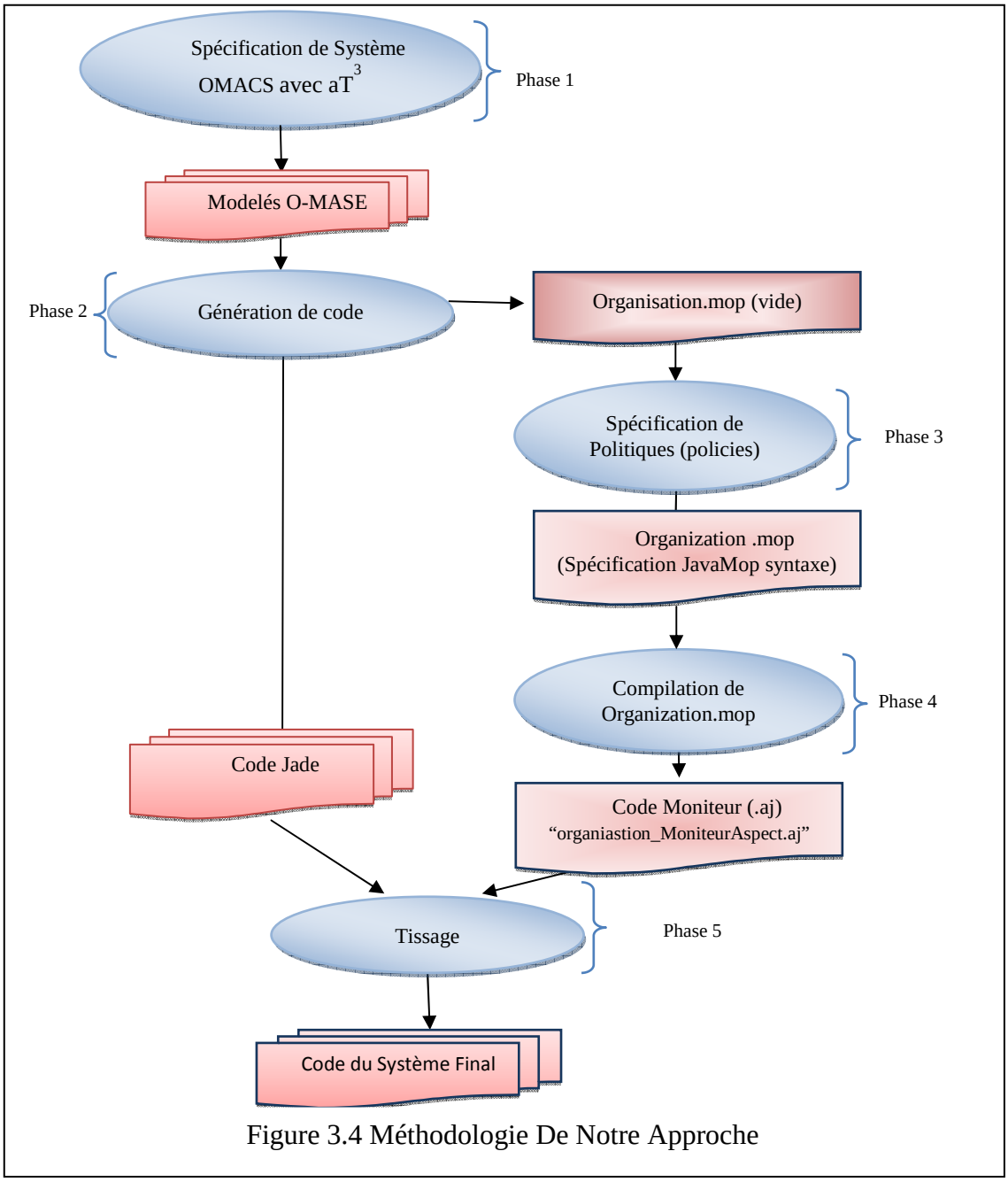


Figure 3.4 Méthodologie De Notre Approche

Phase1 : c'est la phase de spécification et de définition du système OMACS à développer. Après la définition de processus O-MASE spécifique au système concerné, nous élaborons les modèles O-MASE issus de cette instanciation à l'aide des éditeurs graphiques d'aT^3, tout en assurant leur inter-consistance (à l'aide de Framework de vérification). À noter que les politiques ou contraintes organisationnelles et réorganisationnelles ne seront pas définies (avec l'éditeur du modèle politique O-MASE de aT^3) dans cette phase, mais elles seront définies dans la phase 3 (dû aux lacunes de Framework OMACS).

Phase2 : Une fois la phase de spécification terminée, nous procéderons à la génération du code qui l'implémente. Le processus de génération va produire automatiquement le code

Jade associé à la spécification ainsi qu'un fichier *mop* (vide) dans lequel nous allons définir les contraintes et les politiques OMACS.

Phase3 : c'est la phase de spécification des contraintes et politiques comportementales (contraintes organisationnelles et réorganisationnelles) de l'organisation OMACS, en utilisant le Framework JavaMOP. Les contraintes seront définies dans le fichier *mop*, généré dans la phase 2, sous forme des propriétés sur la trace d'exécution du système Jade développé.

Selon la syntaxe JavaMOP, nous commençons dans un premier temps à spécifier les évènements que le moniteur va utiliser pour construire la trace d'exécution du système Jade (tels que l'affectation des agents aux rôles, activation / réalisation ou échec d'un but, …etc.). Comme nous avons cité dans la section III.3.3 ,ces évènements seront définis sous forme de pointcut en AspectJ sur le code Jade qu'il implémente. Ensuite, nous formalisons la (les) propriété (s) à surveiller dans l'une des logiques de formalisme supportées par JavaMOP.

À la fin, nous ajoutons pour chaque propriété (si nécessaire) le code du gestionnaire de la propriété en Java (Handler), qui sera exécuté lorsque la propriété est vérifiée ou violée (par exemple on peut décrire comment l'organisation doit se comporter lors de violation d'une contrainte de réorganisation).

Phase 4 : Après avoir finalisé la spécification des politiques et ajouté les codes à exécuter lors de leurs validations/violations, nous allons compiler la spécification MOP (décrite dans le fichier *mop*) avec le Framework JavaMOP, pour générer le code de moniteurs associes aux propriétés spécifiées. Le code généré est un code AspectJ prêt à être tissé dans le programme Jade (généré dans phase 2) afin de contrôler et corriger si nécessaire son comportement.

Phase 5 :C'est la phase de génération du système final. Après avoir généré le mécanisme d'imposition et du contrôle d'application des politiques (code Moniteur), le code du moniteur sera tissé dans le code original du système jade à l'aide du compilateur AspectJ. Le code moniteur sera exécuté en parallèle avec le code système afin de surveiller, contrôler et corriger si nécessaire le comportement global de l'organisation.

5. Conclusion

Nous avons présenté dans ce chapitre une nouvelle approche de contrôle de réorganisation des SMA. Il s'agit d'une proposition d'amélioration du Framework OMACS.

L'idée de notre proposition étant d'intégrer le Framework JavaMOP dans le processus de développement OMACS. Le choix de l'environnement MOP est, à notre avis, un bon choix, car il permet de combler les lacunes du Framework OMACS par sa richesse de répertoire de logiques (de JavaMOP), et par la possibilité de l'étendre avec d'autres formalismes, ce qui va enrichir OMACS pour exprimer et spécifier les contraintes et politiques. Dans le chapitre qui suit, nous présenterons la validation de notre approche à travers un outil que nous avons développé et qui fusionne les deux Frameworks aT^3 et JavaMOP.

Chapiter IV : Présentation De L’environnement

1. Introduction

Ce chapitre présente l'environnement que nous avons développé dans le cadre de notre projet et dont l'objectif est de supporter et valider notre approche de contrôle de la réorganisation dans les SMA. Cet environnement utilise à la fois le Framework aT^3 (version 1.1.4) et l'outil JavaMOP (version 2.3.1)[7] pour supporter le développement des SMA organisationnels adaptatifs; de la phase de spécification jusqu'à l'implémentation.

2. Plateforme de développement

Vu que notre travail consiste à améliorer l'outil aT^3, nous étions obligés de travailler avec Eclipse, la plateforme sur laquelle aT^3 a été développé. Eclipse est un environnement de développement intégré (IDE) libre, extensible, universel et polyvalent. Il permet de créer des projets de développement de logiciel dans un (ou plusieurs) langage(s) de programmation. Eclipse IDE est principalement écrit en Java (à l'aide de la bibliothèque graphique SWT d'IBM). La spécificité d'Eclipse vient du fait que son architecture est totalement développée autour de la notion de plugin, ce qui facilite l'extension et la personnalisation de cet environnement. Cette spécificité d'Eclipse nous a facilité la tâche d'intégration de JavaMOP dans l'environnement aT^3 ; nous avons gardé l'architecture de conception d'origine d'AgentTool III (un ensemble de plug-ins), où nous avons enrichi l'ensemble de plug-ins aT^3 par d'autres plug-ins tels que le plug-in JavaMOP (JavaMOP qui n'est qu'un simple projet java).

3. Description de l'environnement aT^{3+}

Dans cette section, nous allons présenter notre amélioration d'AgentTool III (baptisé aT^{3+}). Les figures 4.1 et 4.2 représentent les huit éditeurs graphiques des modèles O-MASE (figure 4.1 : Capacité, But, Rôle et Organisation ; figure 4.2 : Domaine, Protocole, Plan et Classe d'Agent) supportés par aT^3.

Après avoir élaboré tous les modèles O-MASE nécessaires, nous pouvons générer le code Jade correspondant au système décrit. La figure 4.3 présente le menu de génération du code ainsi que le projet jade généré (éléments 2 et 3 respectivement). L'élément 4 de la figure 4.3 représente le fichier *Mop* généré, où nous allons spécifier les contraintes et les politiques organisationnelles et réorganisationnelles d'OMACS. À la fin de cette phase, nous

[7] https://code.google.com/p/javamop/

serons en mesure de générer le code Aspect du moniteur. La figure 4.4 décrit la réalisation de cette étape (la partie 3 représente le code moniteur en AspectJ).

Après avoir terminé le développement, la phase de tissage du code moniteur (AspectJ) dans le code jade et la production du système final est accomplie automatiquement lors du lancement de l'exécution du système (tissage lors de compilation : le compilateur AspectJ « *ajc* » est sélectionné automatiquement par défaut par aT^{3+} pour accomplir cette tâche).

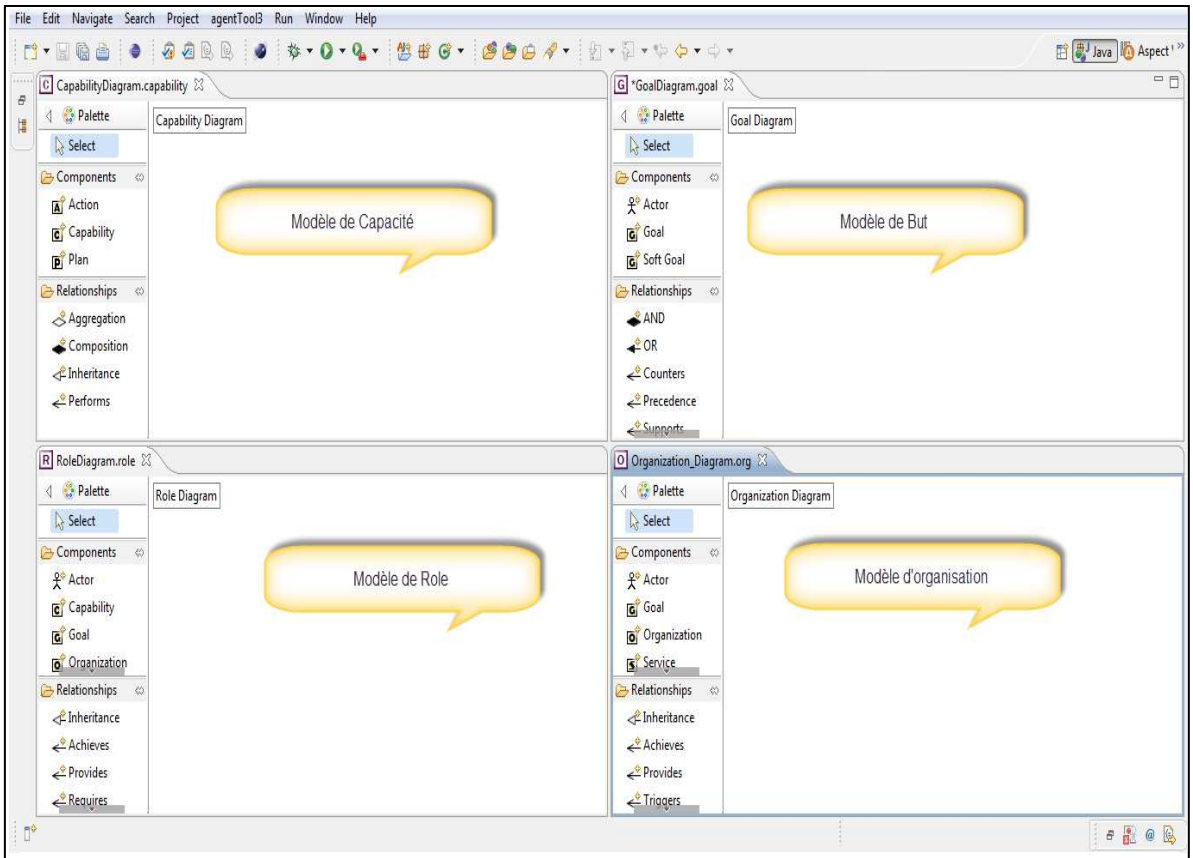


Figure 4.1 Les Editeurs Graphiques d'AgentTool III (1)

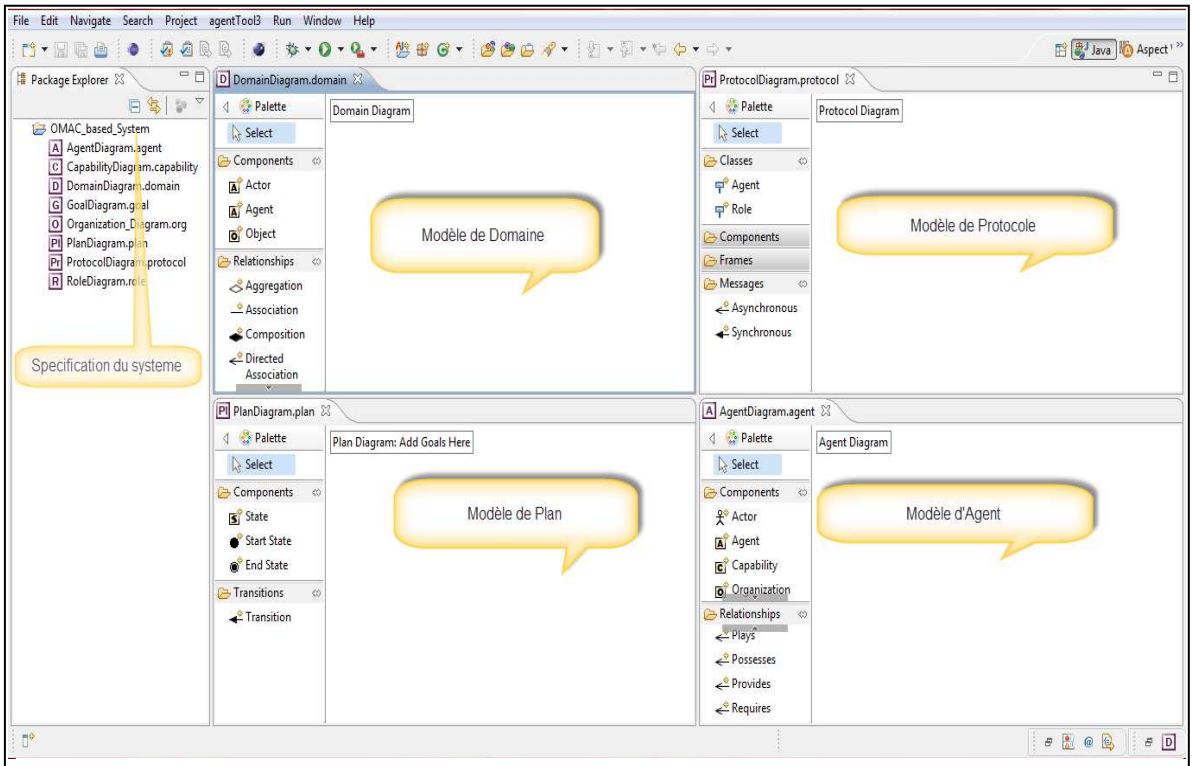


Figure 4.2 Les Editeurs Graphiques d'AgentTool III (2)

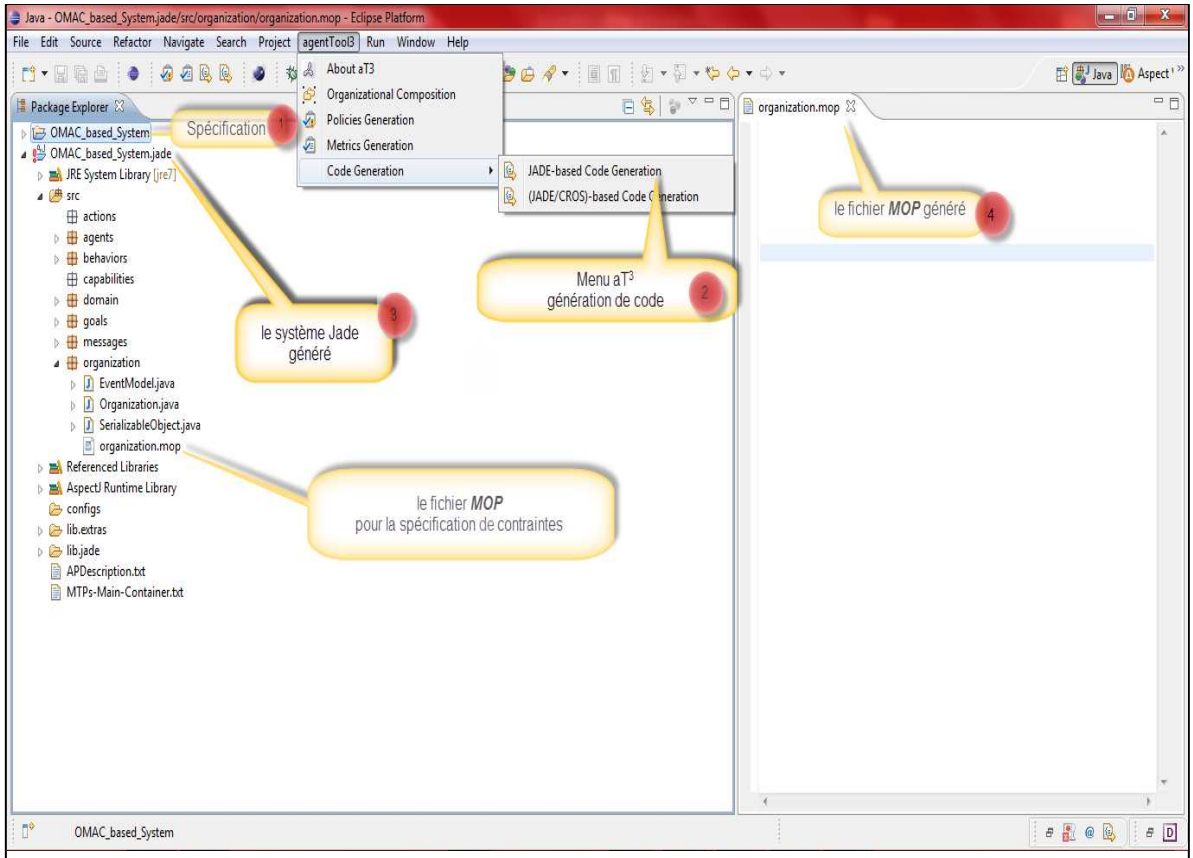


Figure 4.3 La Génération Du Code Jade

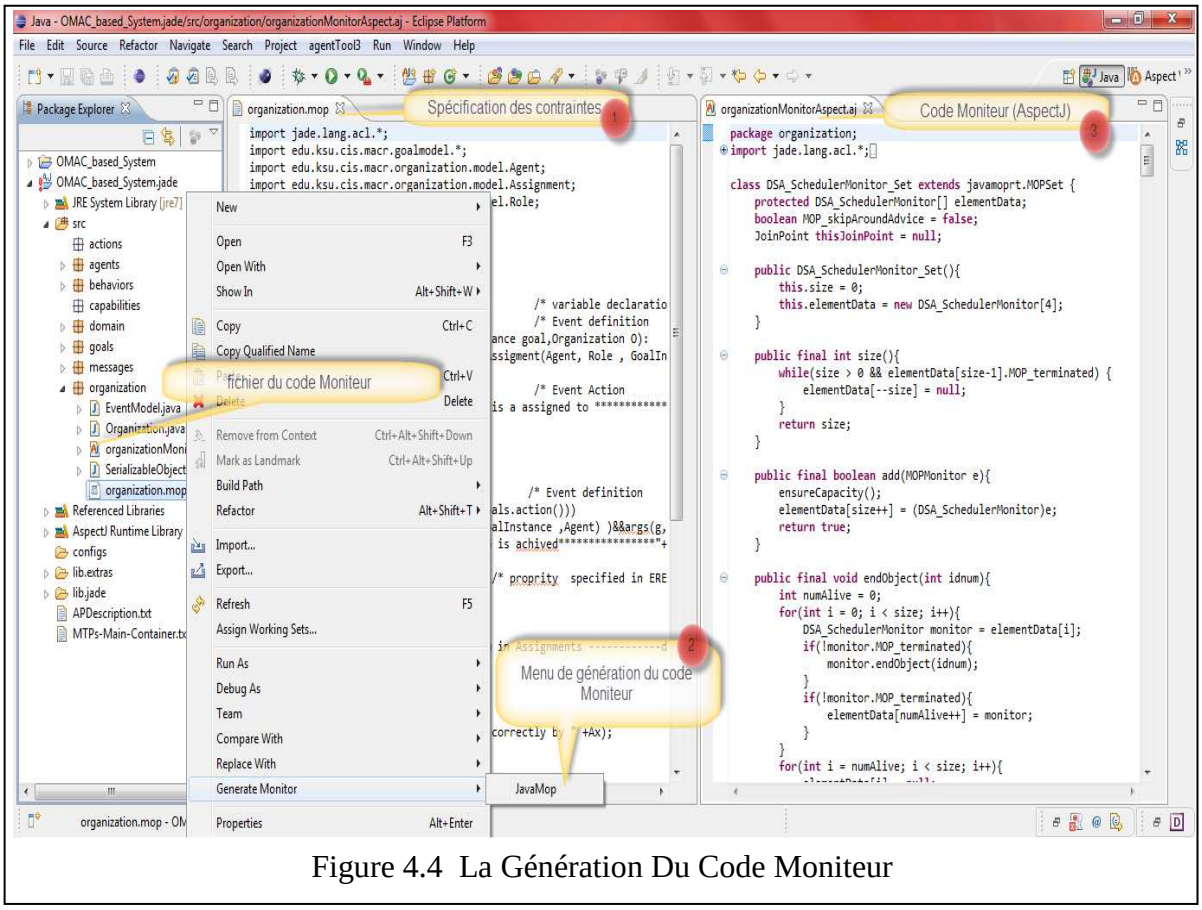


Figure 4.4 La Génération Du Code Moniteur

4. Étude de cas

Après avoir décrit l'environnement aT^{3+}, on va illustrer l'application de notre approche sur un exemple concret de SMA organisationnel. L'exemple représente un système d'information à base d'un réseau de capteurs (Sensor Network). Le système est conduit par un SMA organisationnel. Le but de cette organisation est de fournir à l'observateur (humain) l'ensemble d'informations et mesures décrivant l'état actuel d'une zone à surveiller. Ces informations sont collectées à l'aide des capteurs répartis sur la zone de surveillance. Le système est composé d'un ensemble de capteurs homogènes (S) et des centres de traitement des données (DC) (où les données sont collectées, traitées et vérifiées). Le système dispose d'une interface graphique (GUI) permettant à l'observateur d'activer le système et d'observer son feedback (figure 4.5).

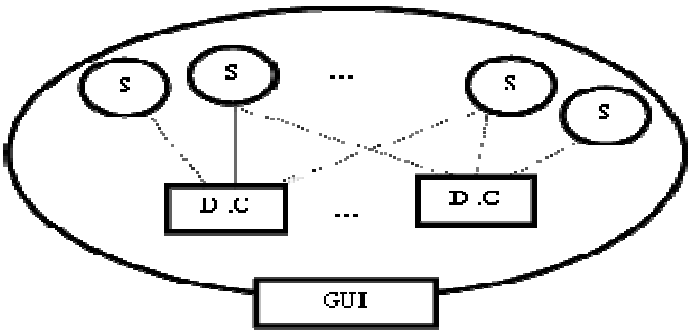


Figure 4.5 Réseau Des Capteurs

4.1. Spécification O-MASE

Pour spécifier ce système, nous avons adopté un processus O-MASE simplifié comportant cinq modèles : le modèle de l'agent, le modèle de rôle, le modèle d'organisation, le modèle de but, et le modèle de domaine. La figure 4.6 présente le modèle de but (GMoDS), tandis que la figure 4.7 décrit les trois premiers modèles

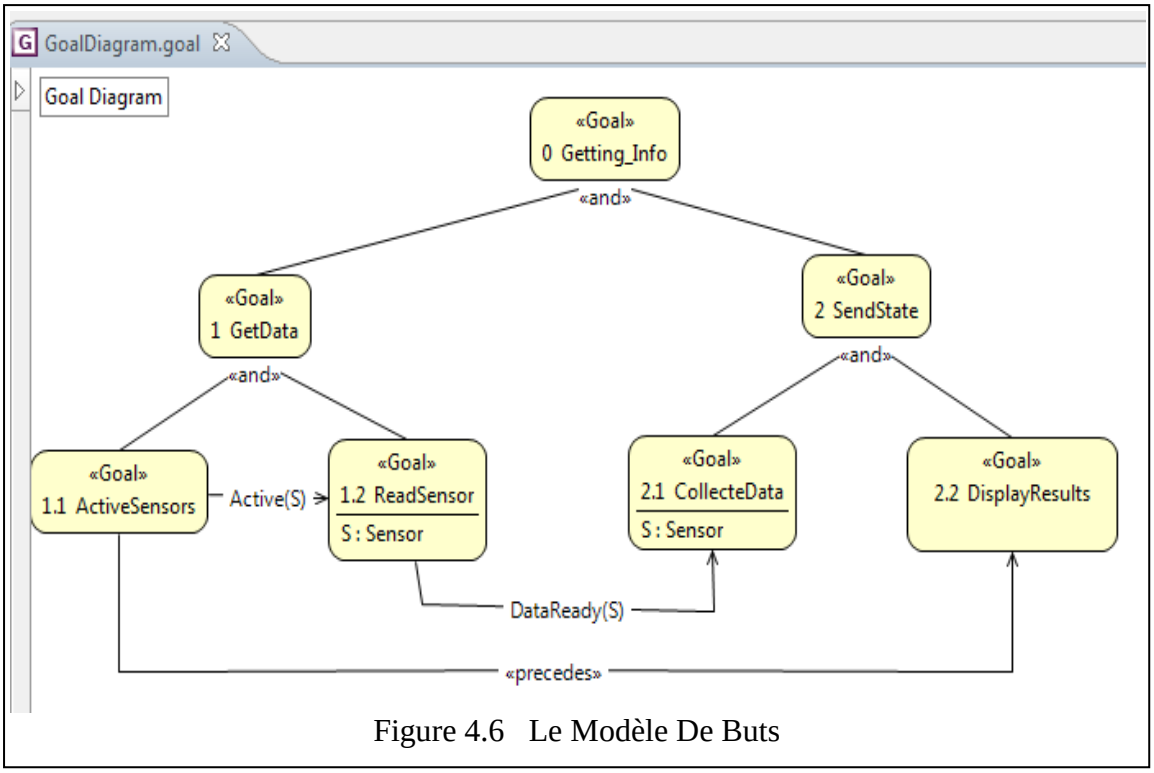


Figure 4.6 Le Modèle De Buts

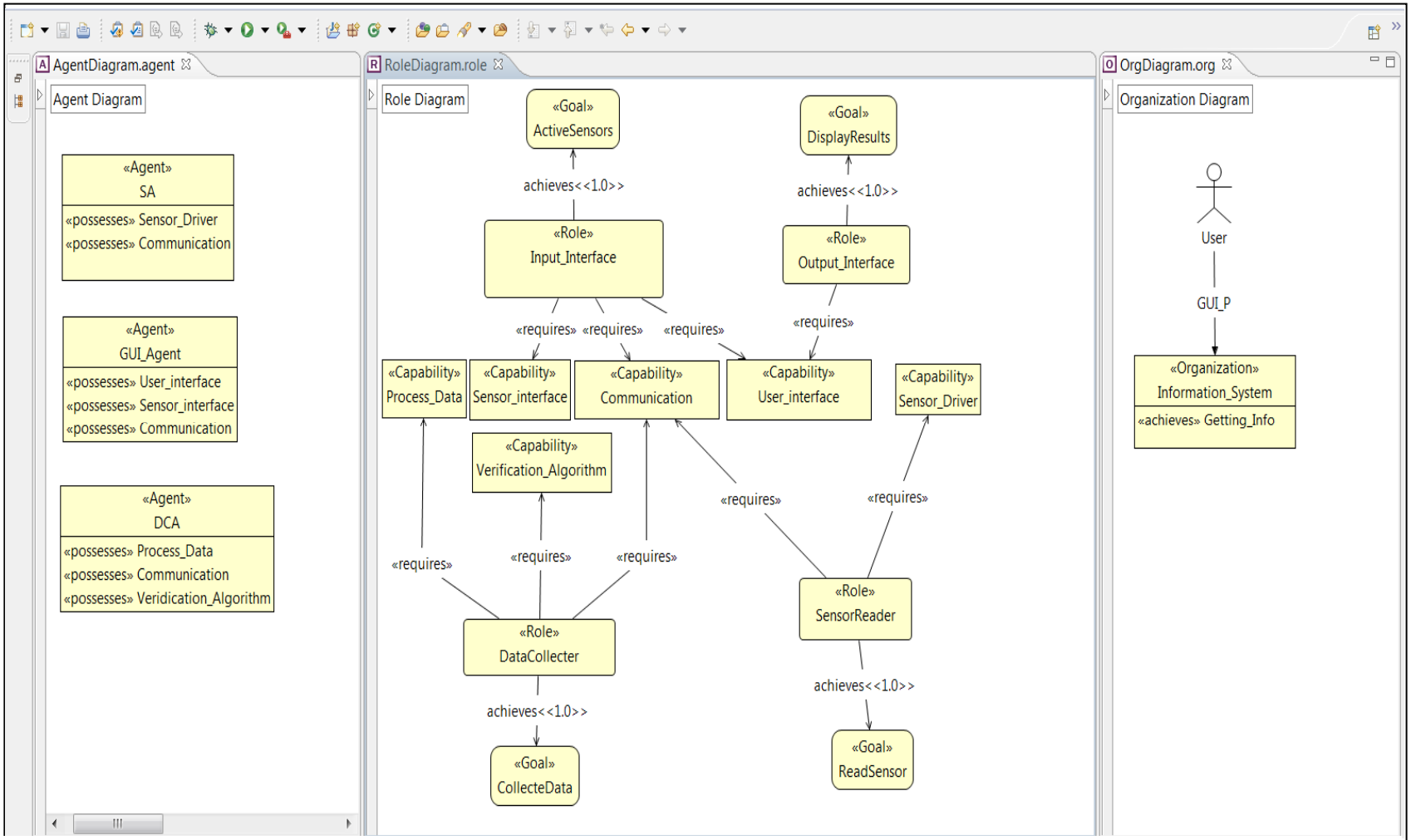


Figure 4.7 Les Modèles Agent, Rôle et Organisation

4.2. Propriétés à vérifier:

Le système développé est construit autour de : 25 capteurs homogènes (S), contrôlés par un ensemble d'Agents *SA* ; 3 centres de traitement des données (DC) gérés par 3 agents *DCA* (Data Centre Agent) (chaque agent est affecté de façon permanente à un centre), et un seul agent GUI (GUI-Agent). Pour montrer l'apport de l'approche proposée dans les situations conflictuelles, nous supposons que le système ne puisse pas supporter plus de 9 Agents *SA* pour contrôler les 25 capteurs disponibles. Ainsi un agent *SA* ne peut pas travailler sur deux capteurs simultanément. Par ailleurs, l'organisation doit partager les tâches de traitement d'informations entre les trois centres de données disponibles de manière équitable. Donc l'organisation est censée respecter deux contraintes de réorganisation :

Contrainte 1 : un *SA* ne peut pas travailler sur deux capteurs en même temps. Pour formaliser en JavaMop cette contrainte, nous avons défini les deux évènements suivants :

- **Play (SA, R, Gs)** : capte la tentation de l'organisation d'affecter un agent *SA* au rôle *Sensor Reader* pour qu'il réalise le but G_S (*Read Sensor*).
- **Achieve (G):** évènement de réalisation d'un but G.

Nous essayons d'éviter la présence dans la trace d'exécution du système de deux évènements ***Play*** successifs pour le même agent sans qu'il y ait un évènement ***Achieve*** qui les sépare. Nous avons formalisé ce comportement inacceptable en JavaMOP sous forme des expressions régulières (ERE) suivantes :

ERE: **(Play Play^{+})**…… (P1)

Si le moniteur détecte la présence de cette séquence dans la trace d'exécution de l'organisation, il nie le deuxième événement *Play* causant cette violation. Ceci oblige l'organisateur à chercher une autre affectation.

Contrainte 2 : la distribution des tâches de traitement de données entre les trois DC doit être équitable, c.-à-d. les nombres des buts *Collect Data* (T_i) réalisés par chaque agent *DCA* doivent être égaux (T_t est le nombre total des buts *Collect Data* déclenchés) :

Si T_t multiple de 3 : $$T_i = {}^{T_t}/_{3} \quad (4.1)$$

Sinon, il peut avoir des: $$T_j = \left({}^{T_t}/_{3}\right) + 1 \quad (4.2)$$

Pour formaliser cette contrainte, nous avons défini les évènements suivants :

- **New_Job ()** : utilisé par le moniteur pour compter le nombre total des buts *Collect Data* déclenchés (T_t). Il est appelé à chaque occurrence d'un évènement *DataReady(S)* (voir le modèle de buts).
- **Play2 (DCA, Gs)** : capte l'affectation d'un agent *DCA* au rôle *DataCollecter* pour qu'il réalise un objectif Gs (*Collect Data*).
- **Prob (As)**: survient avant qu'un évènement Play2 cause une violation de l'équation (4.1) (Ti + 1> Moy (Ts)). Ici Moy signifie moyenne arithmétique.

Nous avons formalisé cette contrainte en logique temporelle linière (LTL):

*LTL***: [] (Play2) S not (Prob)** ..... (P2.a)

*(**Play2** est acceptable tant qu'il n'a pas de problème **Prob**)*

Mais, dans certaines situations (*Ts* n'est pas un multiple de 3 ; équation (4.2)) cette politique (P2. a) doit être suspendue. Pour cela, nous avons défini un quatrième évènement, ***Ignorer(As)*** pour détecter cette situation.

- **Ignorer (As):** se déclenche lorsqu'une occurrence de **Play2** viole l'équation (4.1) (Ti + 1> Moy (Ts)) mais le nombre total des buts n'est pas multiple de 3; la seconde équation (4.2).

Pour gérer cette situation, nous avons reformulé la deuxième contrainte comme suit

*LTL***: (Play2 S (Prob => (*) ignore))** ..... (P2.b)

*(**Play2** est acceptable tant qu'il n'y a pas de problème **Prob** ou que **Prob** est ignoré)*

Comme pour la première contrainte, si le moniteur détecte la violation de cette contrainte (P2.b) dans la trace d'exécution de l'organisation, il va nier le deuxième événement *Play2* causant cette violation. Ce qui oblige l'organisateur à chercher une autre affectation.

4.3. Architecture du Système et algorithme de réorganisation

Pour la mise en œuvre du système, une architecture organisationnelle centralisée a été utilisée (DeLoach, 2009), où toutes les connaissances, mécanisme de raisonnement et algorithme de réorganisation sont implémentés au niveau d'un seul agent gestionnaire

(Agent Organisateur) (d'ailleurs la fonction de génération du code du aT^3 par défaut génère automatiquement un code jade basé sur architecture centralisée).

Lors d'exécution, l'organisation est définie par l'ensemble des buts actifs et l'ensemble des affectations courantes (ϕ). Une fois un évènement (déclencheur de réorganisation) est perçu par l'organisateur (évènement organisationnel tel que la réalisation ou l'échec d'un but ; ou bien un évènement spécifique à la réalisation d'un nouvel objectif), il va mettre à jour sa liste des buts actifs et lance le processus de réorganisation. Ce processus est censé produire un ensemble d'affectations optimal (ϕ), réalisable (qui a la meilleure valeur non nulle de la fonction oaf) et acceptable (ne viole pas les contraintes).

L'algorithme de réorganisation que nous avons implémenté pour ce système d'information est un algorithme simple fondu sur l'hypothèse suivante :

- L'algorithme est basé sur la philosophie OMACS (l'affectation des agents aux rôles pour qu'ils réalisent les buts ne dépend que de la possession d'agents les capacités exigées par le rôle).
- Les fonctions *oaf* et *rcf* sont les fonctions par défaut d'OMACS (Section II.2.1)
- Tous les *SA* (respectivement *DCA*) sont homogènes (ont les mêmes qualités de capacité *Possesses(A,C_i)=1*).
- Il n'y a pas de dégradation de qualité de capacité lors d'exécution (toujours *Possesses(A,C_i)=1*).
- Dû aux hypothèses précédentes, toutes les affectations potentielles pour réaliser un but *ReadSensor* (respectivement *CollecteData*) sont égales ; ∀ À ϵ *SA*, ∀ G ϵ *Ens(ReadSensor*) Potential(A,G,R)=1 (respectivement ∀ A ϵ *DCA*, ∀ G ϵ *Ens(CollecteData)* Potential(A,G,R)=1) . Pour cela, l'algorithme va prendre seulement la première affectation possible qu'il trouve.
- L'algorithme cherchera uniquement les solutions réalisables (non nécessairement acceptables).
- L'algorithme de réorganisation s'arrête lorsque le but global est achevé ou bien il n'y pas de buts à réaliser.

4.4. Exécution et résultats

Après avoir terminé la spécification du système, la génération de son code jade et le code AspectJ du moniteur à l'aide de l'outil aT^{3+}, nous passons à la validation du système final. Pour des raisons de clarté, nous préférons présenter quelques cas des tentatives de violation des contraintes 1 et 2 et comment le système a traité ces cas (figures 4.8 et 4.9 respectivement). Afin d'optimiser la capture de la trace d'exécution, nous avons préféré de n'afficher que les évènements jugés pertinents :

- « *# |New Event < E > |* » : Indique l'occurrence d'un évènement déclencheur de la réorganisation.
- « *$$ < G >* » : Indique la réalisation d'un but.
- « *--- >New Assignment* » : Indique un processus de réorganisation et l'affectation résultante.

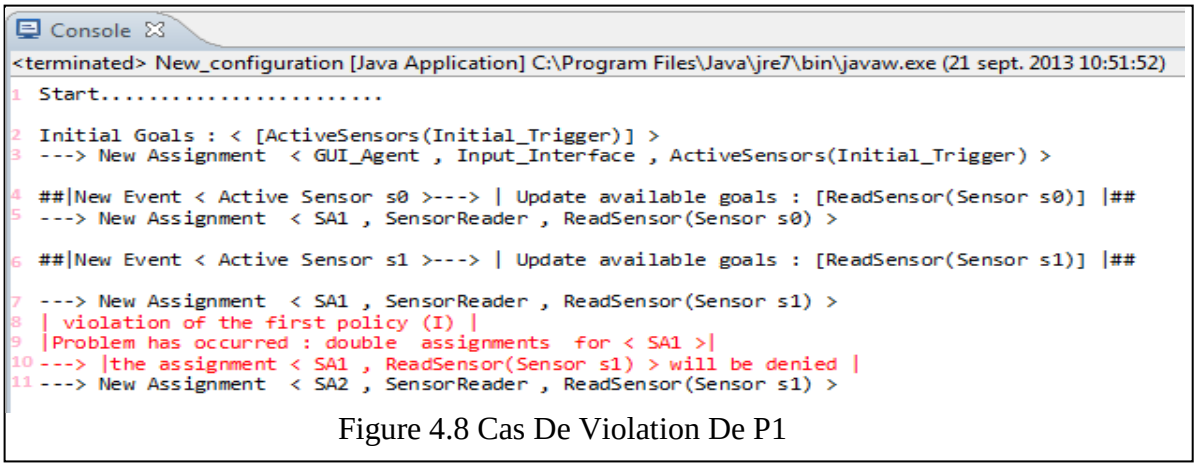


Figure 4.8 Cas De Violation De P1

L'assurance de la respectassions de la contrainte P1 : Dans la figure 4.8, après le lancement du système, un but initial *ActiveSensors (Initial_Trigger)* est déclenché automatiquement, entraînant une réorganisation du système afin de trouver un agent capable de réaliser ce but. Le résultat de cette réorganisation est la sélection du *GUI-Agent* pour qu'il joue le rôle *Input Interface* et réalise le but *ActiveSensors* (Initial_Trigger) (ligne 3). Le *GUI-Agen*t commence à activer les 25 capteurs du système. Au début, il active le capteur *S0* (événement *Active (S0)*), ce qui déclenche un nouveau but à satisfaire *ReadSensor (S0)* (ligne 4), une réorganisation est lancée de nouveau. Comme résultat, on trouve que la réalisation du but *ReadSensor (S0)* est attribuée à *SA1* (ligne 5). À ce moment-là le *GUI-Agent* active le second capteur *S1*, provoquant une mise à jour des buts réalisés (l'ajout de

ReadSensor (S1)) (ligne 6). L'organisation lance une autre fois le processus de réorganisation. Le résultat de cette réorganisation est la sélection de *SA1* pour qu'il réalise *ReadSensor (S1)* (Ligne 7), ce que représente une violation de la 1ere contrainte, car *SA1* est toujours en train de réaliser *ReadSensor (S0)*. Le moniteur a détecté cette violation (lignes 8, 10) et il agit par l'annulation de l'affectation provoquant cette violation *(<Read Sensor (S1), SA1>*) (ligne 11). Par conséquent, l'agent organisateur va détecter que sa dernière affectation n'a pas été appliquée (*ReadSensor (S1)*), il va lancer un nouveau processus de réorganisation pour trouver un autre agent capable de réaliser *ReadSensor (S1)*. Cette fois-ci, il choisit *SA2* pour achever ce but au lieu de *SA1* (ligne 11).

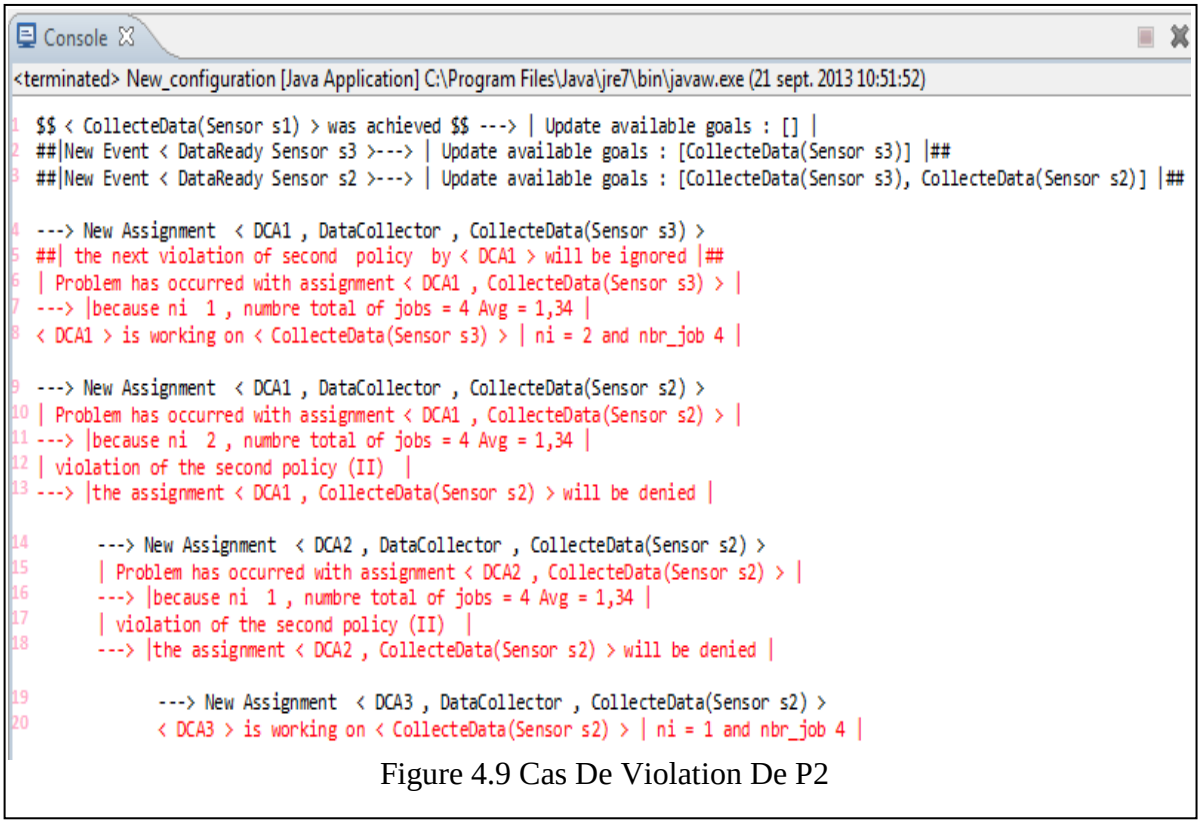


Figure 4.9 Cas De Violation De P2

L'assurance de respectassions de la contrainte P2 : La figure 4.9 décrit une étape avancée dans l'exécution du système, où quatre buts *CollectData(S0,S1,S2,S3)* sont déclenchés ,deux parmi eux (*CollectData(S0) CollectData(* S1)) sont réalisés précédemment par *DCA1* et *DCA2* respectivement (ils ne sont pas représentés dans la figure 4.9). Le système lance un processus de réorganisation pour tenter de trouver des agents (Data Center) capables de réaliser le reste des buts (*S2* et *S3*) (les lignes 2 et 3 représentent respectivement le déclenchement des buts *CollectData S2, S3*). Après la réorganisation, le système choisit *DCA1* pour atteindre *CollectData (S2)* (ligne 4). Malgré que nous avons exigé que la répartition des tâches entre les DC doive être équitable, ce qui n'est pas le cas dans cette

affectation (voir P2.a) (car le nombre de buts affectés au *DCA1* serait supérieur au moyen), le nombre des buts est de quatre (n'est pas multiple de 3) et ce cas peut être ignoré (lignes 5-8). Ensuite, le système lance une autre réorganisation pour satisfaire le dernier but *CollectData* (S2). Il va sélectionner une autre fois DCA1; ce qu'est une violation du la contrainte P2, car, le nombre de buts affectés à DCA1 est supérieur à la moyenne (ligne 10). Cette affectation sera suspendue, et le système lance une autre réorganisation (ligne 13). Le résultat de cette réorganisation est une autre violation de P2, car, il a choisi DCA2 (si elle est acceptée, il sera impossible de faire équilibrer la charge entre les centres (DC1= 2, DC2=2, DC3=0)) (lignes 14-17). Cette dernière affectation est annulée, ce qui oblige l'organisateur de lancer la réorganisation une troisième fois (lignes 18-19). Cette fois-ci il a choisi la bonne affectation, il a sélectionné DCA3 pour réaliser *CollectData* (S2) (ligne 20).

À la fin de l'exécution du système, l'organisation a achevé son but principal (*Getting-Info*) correctement sans qu'il n'y ait pas une violation des contraintes définies (P1 et P2). Les neuf agents *SA* ont été utilisés correctement pour faire fonctionner les 25 capteurs (aucun agent n'a travaillé simultanément sur deux capteurs). Les 25 flux d'informations mesurés par les capteurs (but CollectData) ont été distribués sur les 3 centres de traitement de manière équitable (tableau 4.1)

Agent	DCA1	DCA2	DCA3
Nombre des buts traités	9	8	8

Table 4.1: Nombre des buts *CollectData* traités par chaque agent

Tout au long de l'exécution du système, 52 buts ont été déclenchés (1 *ActiveSensor* ,25 *ReadSensor*, 25 *CollectData* ,1 *DisplayResult*), causant 111 processus de réorganisation, dont 36 sont dus aux affectations qui violent de P1 et 24 sont dus aux violations de P2.

5. Conclusion

Nous avons présenté dans ce chapitre une amélioration de l'environnement aT^3 par l'intégration de l'outil JavaMOP. Le résultat de cette amélioration est un environnement de développement des SMA (baptisé aT^{3+}) supportant le processus de développement des SMA organisationnelles adaptatives et contrôlables (de la spécification à l'implémentation). L'intérêt de l'intégration de JavaMOP étant de fournir une plateforme formelle dédiée à la fois à la spécification des contraintes organisationnelles et à l'instanciation de mécanisme de contrôle et d'assurance de l'application de ces contraintes.

Conclusion Générale

La réorganisation des systèmes multi-agents est une tâche indispensable pour leur adaptation aux changements souvent imprévus de l'environnement. Dans la littérature peu travaux ont été réalisés, dans ce contexte, en concevant des modèles organisationnels tenant compte des aspects réorganisationnels durant leur phase d'implémentation. Parmi ces modèles, nous citons OMACS, un modèle supporté par plusieurs méthodologies et techniques de développement de systèmes multi-agents. Cependant, la spécification des contraintes organisationnelles, leur implémentation ainsi que leur contrôle ne sont pas bien maitrisés. Ces aspects sont laissés à la charge de programmeur.

Afin de surmonter ce problème, nous avons proposé dans ce mémoire une approche de développement de systèmes multi-agent adaptatifs basée sur le modèle organisationnel OMACS. Nous avons utilisé conjointement les deux outils JavaMOP comme une solution pour contourner les problèmes du Framework OMACS. L'intégration de JavaMOP avec aT^3 nous a permis de définir un processus solide et relativement complet dédié à la conception des SMA adaptatifs contrôlés.

Comme perspectives à moyen terme, nous envisageons de:

1. Étendre l'ensemble des logiques supportées par JavaMOP avec d'autres logiques plus appropriées aux SMA telles que : CTL, la logique déontique… etc.
2. Étendre JavaMOP pour qu'il supporte la définition des méta-contraintes pour contrôler l'activation et la désactivation des contraintes soft.
3. Améliorer l'environnement aT^{3+} par développer un éditeur textuel intelligent pour la spécification des contraintes capable de faciliter et de vérifier syntaxiquement et symétriquement les contraintes définies avant la génération de code moniteur.
4. Étendre le Framework de vérification d'OMACS pour qu'il supporte la vérification de la cohérence entre les modèles O-MaSE et les spécifications de contraintes.

Bibliographies

Abdallah, S., and Lesser, V. (2007). Multiagent reinforcement learning and self-organization in a network of agents. In Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, p. 39.

Abercrombie, P., and Karaorman, M. (2002). jContractor: Bytecode instrumentation techniques for implementing design by contract in Java. Electronic Notes in Theoretical Computer Science *70*, 55–79.

Alberola, J.M., Julian, V., and Garcia-Fornes, A. (2011). Open issues in multiagent system reorganization. In Highlights in Practical Applications of Agents and Multiagent Systems, (Springer), pp. 151–158.

Alberola Oltra, J.M. (2013). Reorganization in Dynamic Agent Societies.

Aldewereld, H., Penserini, L., Dignum, F., and Dignum, V. (2008). Regulating organizations: The ALIVE approach. In Workshop on Regulations Modelling and Deployment (ReMoD-08),@ CAISE, pp. 37–48.

Aldrich, H. (1999). Organizations evolving (Sage).

Aldrich, H. (2008). Organizations and environments (Stanford University Press).

Allan, C., Avgustinov, P., Christensen, A.S., Hendren, L., Kuzins, S., Lhoták, O., De Moor, O., Sereni, D., Sittampalam, G., and Tibble, J. (2005). Adding trace matching with free variables to AspectJ. ACM SIGPLAN Notices *40*, 345–364.

Argente, E., Botti, V., and Julian, V. (2011). GORMAS: An organizational-oriented methodological guideline for open MAS. In Agent-Oriented Software Engineering X, (Springer), pp. 32–47.

Argente, E., Billhardt, H., Cuesta, C.E., Esparcia, S., Görmer, J., Hermoso, R., Kirikal, K., Lujak, M., Pérez-Sotelo, J.-S., and Taveter, K. (2013). Adaptive Agent Organisations. In Agreement Technologies, (Springer), pp. 321–353.

Artikis, A., Kaponis, D., and Pitt, J. (2009). Dynamic Specifications for Norm-Governed Systems.

Balbo, F., Seghrouchni, A.E.F., and Pinson, S. (2002). Organisation et applications des SMA, chapitre La coordination d'actions par planification Multi-Agents. Hermès.

Barnett, W.P., and Carroll, G.R. (1995). Modeling internal organizational change. Annual Review of Sociology 217–236.

Barringer, H., Goldberg, A., Havelund, K., and Sen, K. (2004). Rule-based runtime verification. In Verification, Model Checking, and Abstract Interpretation, pp. 44–57.

Barringer, H., Finkbeiner, B., Gurevich, Y., and Sipma, H. (2005). Runtime Verification (RV'05), volume 144 of ENTCS (Elsevier).

Bartetzko, D., Fischer, C., Möller, M., and Wehrheim, H. (2001). Jass—Java with assertions. Electronic Notes in Theoretical Computer Science *55*, 103–117.

Bauer, B., Müller, J.P., and Odell, J. (2001). Agent UML: A formalism for specifying multiagent software systems. International Journal of Software Engineering and Knowledge Engineering *11*, 207–230.

Bernon, C., Cossentino, M., and Pavón, J. (2005). Agent-oriented software engineering. The Knowledge Engineering Review *20*, 99–116.

Bodden, E. (2005). J-LO-A tool for runtime-checking temporal assertions. Master's thesis, RWTH Aachen university.

Bou, E., López-Sánchez, M., and Rodríguez-Aguilar, J.A. (2007). Towards self-configuration in autonomic electronic institutions. In Coordination, Organizations, Institutions, and Norms in Agent Systems II, (Springer), pp. 229–244.

Campos, J., Esteva, M., López-Sánchez, M., Morales, J., and Salamó, M. (2011). Organisational adaptation of multi-agent systems in a peer-to-peer scenario. Computing *91*, 169–215.

Carvalho, G., Almeida, H., Gatti, M., Vinicius, G., Paes, R., Perkusich, A., and Lucena, C. (2006). Dynamic law evolution in governance mechanisms for open multi-agent systems. In Second Workshop on Software Engineering for Agent-oriented Systems,.

Chen, F., and Roşu, G. (2003). Towards monitoring-oriented programming: A paradigm combining specification and implementation. Electronic Notes in Theoretical Computer Science *89*, 108–127.

Chen, F., and Roşu, G. (2007). Mop: an efficient and generic runtime verification framework. In ACM SIGPLAN Notices, pp. 569–588.

Chen, F., d' Amorim, M., and Roşu, G. (2004). A formal monitoring-based framework for software development and analysis. In Formal Methods and Software Engineering, (Springer), pp. 357–372.

Chen, F., d' Amorim, M., and Roşu, G. (2006). Checking and correcting behaviors of java programs at runtime with java-mop. Electronic Notes in Theoretical Computer Science *144*, 3–20.

Chen, F., O'Neil Meredith, P., Jin, D., and Roşu, G. (2009). Efficient formalism-independent monitoring of parametric properties. In Automated Software Engineering, 2009. ASE'09. 24th IEEE/ACM International Conference On, pp. 383–394.

Chopinaud, C., Taillibert, P., and Seghrouchni, A.E.F. (2005). Contrôle de l'émergence dans les systèmes d'agents cognitifs autonomes.

Coutinho, L.R., Sichman, J.S., and Boissier, O. (2009). Modelling dimensions for agent organizations. Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models.

Decker, K. (1996). TAEMS: A framework for environment centered analysis & design of coordination mechanisms. Foundations of Distributed Artificial Intelligence 429–448.

Deloach, S.A., Oyenan, W.H., and Matson, E.T. (2008). A capabilities-based model for adaptive organizations. Autonomous Agents and Multi-Agent Systems *16*, 13–56.

DeLoach, S. (2009). OMACS: A framework for adaptive, complex systems. Handbook of Research on Multi-agent Systems: Semantics and Dynamics of Organizational Models 76–98.

DeLoach, S.A. (2002). Modeling organizational rules in the multi-agent systems engineering methodology. In Advances in Artificial Intelligence, (Springer), pp. 1–15.

DeLoach, S.A. (2006). Engineering organization-based multiagent systems. In Software Engineering for Multi-Agent Systems IV, (Springer), pp. 109–125.

DeLoach, S.A., and Kolesnikov, V.A. (2006). Using design metrics for predicting system flexibility. In Fundamental Approaches to Software Engineering, (Springer), pp. 184–198.

DeLoach, S.A., and Miller, M. (2010). A goal model for adaptive complex systems. International Journal of Computational Intelligence: Theory and Practice *5*, 83–92.

Desanctis, G., and Monge, P. (1998). Communication processes for virtual organizations. Journal of Computer-Mediated Communication *3*, 0–0.

Dignum, M.V. (2004). A model for organizational interaction: based on agents, founded in logic.

Dignum, V. (2009). The role of organization in agent systems. Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models 1–16.

Dignum, V., and Dignum, F. (2009). A logic of agent organizations. Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models 220–241.

Dignum, V., and Dignum, F. (2012). A logic of agent organizations. Logic Journal of IGPL *20*, 283–316.

Dignum, V., Dignum, F., and Sonenberg, L. (2004). Towards dynamic reorganization of agent societies. In In Proceedings of Workshop on Coordination in Emergent Agent Societies, pp. 22–27.

Dignum, V., Vázquez-Salceda, J., and Dignum, F. (2005). Omni: Introducing social structure, norms and ontologies into agent organizations. In Programming Multi-Agent Systems, (Springer), pp. 181–198.

Drogoul, A., Corbara, B., and Fresneau, D. (1995). MANTA: New experimental results on the emergence of (artificial) ant societies. Artificial Societies: The Computer Simulation of Social Life 190–211.

Esparcia, S., and Argente, E. (2012). Forces that drive organizational change in an adaptive virtual organization. In Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference On, pp. 46–53.

Esteva, M., Padget, J., and Sierra, C. (2002). Formalizing a language for institutions and norms. In Intelligent Agents VIII, (Springer), pp. 348–366.

Ferber, J. (1999). Multi-agent systems: an introduction to distributed artificial intelligence (Addison-Wesley Reading).

Ferber, J., and Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems. In Multi Agent Systems, 1998. Proceedings. International Conference On, pp. 128–135.

Ferber, J., Gutknecht, O., and Michel, F. (2004). From agents to organizations: an organizational view of multi-agent systems. In Agent-Oriented Software Engineering IV, (Springer), pp. 214–230.

Ferber, J., Michel, F., and Báez, J. (2005). AGRE: Integrating environments with organizations. In Environments for Multi-agent Systems, (Springer), pp. 48–56.

Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. International Journal of High Performance Computing Applications *15*, 200–222.

Garcia-Ojeda, J.C., and DeLoach, S.A. (2009a). agentTool III: from process definition to code generation. In Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2, pp. 1393–1394.

Garcia-Ojeda, J.C., and DeLoach, S.A. (2009b). agentTool process editor: supporting the design of tailored agent-based processes. In Proceedings of the 2009 ACM Symposium on Applied Computing, pp. 707–714.

Garcia-Ojeda, J.C., DeLoach, S.A., Oyenan, W.H., and Valenzuela, J. (2008). O-MaSE: a customizable approach to developing multiagent development processes (Springer).

Gâteau, B., Boissier, O., Khadraoui, D., and Dubois, E. (2005). MoiseInst: An Organizational Model for Specifying Rights and Duties of Autonomous Agents. EUMAS *2005*, 484–485.

Guessoum, Z., Ziane, M., and Faci, N. (2004). Monitoring and organizational-level adaptation of multi-agent systems. In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2, pp. 514–521.

Harmon, S.J., and DeLoach, S.A. (2008). Trace-Based Specification of Law and Guidance Policies for Multi-Agent Systems. In Engineering Societies in the Agents World VIII, (Springer), pp. 333–349.

Harmon, S.J., DeLoach, S.A., and Caragea, D. (2008). Leveraging Organizational Guidance Policies with Learning to Self-Tune Multiagent Systems. In Self-Adaptive and Self-Organizing Systems, 2008. SASO'08. Second IEEE International Conference On, pp. 223–232.

Havelund, K., and Rosu, G. (2001). Workshops on Runtime Verification (RV'01, RV'02, RV'04), volume 55, 70 (4), to appear of ENTCS. Elsevier *2002*, 2004.

Havelund, K., and Roşu, G. (2001). Monitoring java programs with java pathexplorer. Electronic Notes in Theoretical Computer Science *55*, 200–217.

Hexmoor, H. (2010). Reorganization in massive multiagent systems. In Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference On, pp. 189–195.

Hoogendoorn, M., and Treur, J. (2009). An adaptive multi-agent organization model based on dynamic role allocation. International Journal of Knowledge-based and Intelligent Engineering Systems *13*, 119–139.

Horling, B., and Lesser, V. (2005). Quantitative organizational models for large-scale agent systems. In Massively Multi-Agent Systems I, (Springer), pp. 121–135.

Horling, B., Benyo, B., and Lesser, V. (2001). Using self-diagnosis to adapt organizational structures. In Proceedings of the Fifth International Conference on Autonomous Agents, pp. 529–536.

Hübner, J.F., Sichman, J.S., and Boissier, O. (2002). A model for the structural, functional, and deontic specification of organizations in multiagent systems. In Advances in Artificial Intelligence, (Springer), pp. 118–128.

Hübner, J.F., Sichman, J.S., and Boissier, O. (2004). Using the Moise+ model for a Cooperative Framework of MAS Reorganisation. In Advances in Artificial intelligence–SBIA 2004, (Springer), pp. 506–515.

Hussein, S., Meredith, P., and Rosu, G. (2012). Security-policy monitoring and enforcement with javamop. In Proceedings of the 7th Workshop on Programming Languages and Analysis for Security, p. 3.

Jennings, N.R. (1999). Agent-oriented software engineering. In Multiple Approaches to Intelligent Systems, (Springer), pp. 4–10.

Jennings, N.R. (2000). On agent-based software engineering. Artificial Intelligence *117*, 277–296.

Jin, D. (2012). Making Runtime Monitoring of Parametric Properties Practical. University of Illinois.

Jin, D., Meredith, P.O., Lee, C., and Rosu, G. (2012). Javamop: Efficient parametric runtime monitoring framework. In Software Engineering (ICSE), 2012 34th International Conference On, pp. 1427–1430.

Kalache, A., Mokhati, F., and Badri, M. (2014). Towards A New Approach For Controlling The Reorganization Process Of Multi-Agent Systems. To Be Appear in IJATS-IGI-Gobal.

Kamboj, S., and Decker, K.S. (2007). Organizational self-design in semi-dynamic environments. In Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, p. 202.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J. (1997). Aspect-oriented programming (Springer).

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W.G. (2001). An overview of AspectJ. In ECOOP 2001—Object-Oriented Programming, (Springer), pp. 327–354.

Kota, R., Gibbins, N., and Jennings, N.R. (2012). Decentralized approaches for self-adaptation in agent organizations. ACM Transactions on Autonomous and Adaptive Systems (TAAS) *7*, 1.

Kubera, Y., Mathieu, P., and Picault, S. (2008). Interaction-Oriented Agent Simulations: From Theory to Implementation. In ECAI, pp. 383–387.

Leavens, G.T., Ruby, C., Leino, K.R.M., Poll, E., and Jacobs, B. (2000). JML (poster session): notations and tools supporting detailed design in Java. In Addendum to the 2000 Proceedings of the Conference on Object-oriented Programming, Systems, Languages, and Applications (Addendum), pp. 105–106.

Lesser, V., Decker, K., Wagner, T., Carver, N., Garvey, A., Horling, B., Neiman, D., Podorozhny, R., Prasad, M.N., and Raja, A. (2004). Evolution of the GPGP/TAEMS domain-independent coordination framework. Autonomous Agents and Multi-agent Systems *9*, 87–143.

Mahmoud, S., Griffiths, N., Keppens, J., and Luck, M. (2013). Norm Emergence through Dynamic Policy Adaptation in Scale Free Networks. In Coordination, Organizations, Institutions, and Norms in Agent Systems VIII, (Springer), pp. 123–140.

McShane, S.L., Travaglione, A., and Olekalns, M. (2003). Organisational behaviour on the Pacific Rim.

Meredith, P.O. (2012). Efficient, expressive, and effective runtime verification. University of Illinois.

Meredith, P.O., Jin, D., Griffith, D., Chen, F., and Roşu, G. (2012). An overview of the MOP runtime verification framework. International Journal on Software Tools for Technology Transfer *14*, 249–289.

Meyer, B. (1988). Object-oriented software construction (Prentice hall New York).

Miller, M. (2007). A goal model for dynamic systems. Kansas State University.

Morin, E. (1977). La méthode, Tome 1: la nature de la nature. Seuil, Paris.

Nair, R., Tambe, M., and Marsella, S. (2003). Role allocation and reallocation in multiagent teams: Towards a practical analysis. In Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 552–559.

Oyenan, W.H., and DeLoach, S.A. (2010). Towards a systematic approach for designing autonomic systems. Web Intelligence and Agent Systems *8*, 79–97.

Parunak, H.V.D., and Odell, J.J. (2002). Representing social structures in UML. In Agent-Oriented Software Engineering II, (Springer), pp. 1–16.

De Paz, J.F., Zato, C., Villarubia, G., Bajo, J., and Corchado, J.M. (2014). Distribution of Roles in Virtual Organization of Agents. In The 8th International Conference on Knowledge Management in Organizations, pp. 485–497.

Pellizzoni, R., Meredith, P., Caccamo, M., and Rosu, G. (2008). Hardware runtime monitoring for dependable cots-based real-time embedded systems. In Real-Time Systems Symposium, 2008, pp. 481–491.

Picard, G., Hübner, J.F., Boissier, O., and Gleizes, M.-P. (2009). Réorganisation et auto-organisation dans les systèmes multi-agents. Journées Francophones Sur Les Systèmes Multi-Agents (JFSMA 2009) 89–98.

Rao, A.S., and Georgeff, M.P. (1992). An abstract architecture for rational agents. KR *92*, 439–449.

Robbins, S. (1987). Organization Theory - Structure, Design and Applications (La théorie de l'organisation - structure, conception et applications),.

Scott, A., and Wood, M. (2001). Developing multiagent systems with agenttool. In Intelligent Agents VII Agent Theories Architectures and Languages, (Springer), pp. 46–60.

Da Silva, V.T., Choren, R., and de Lucena, C.J. (2004). A UML based approach for modeling and implementing multi-agent systems. In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2, pp. 914–921.

So, Y., and Durfee, E.H. (1993). An organizational self-design model for organizational change. Ann Arbor *1001*, 48109.

Sokolsky, O., and Viswanathan, M. (2003). Runtime Verification (RV'03), volume 89 of ENTCS (Elsevier).

Stone, P., and Veloso, M. (1999). Task decomposition and dynamic role assignment for real-time strategic teamwork. In Intelligent Agents V: Agents Theories, Architectures, and Languages, (Springer), pp. 293–308.

Le Strugeon, E., Agimont, G., Mandiau, R., and Millot, P. (1996). Organisation évolutive d'un systeme multi-agents: Illustration par des agents-robots miniers. Distribuée et Systeme Multi-Agents 167–176.

Tambe, M., Adibi, J., Al-Onaizan, Y., Erdem, A., Kaminka, G.A., Marsella, S.C., and Muslea, I. (1999). Building agent teams using an explicit teamwork model and learning. Artificial Intelligence *110*, 215–239.

Varela, F.J. (1989). Autonomie et connaissance: Essai sur le vivant (Éditions Du Seuil).

Varela, F.J., and Bourgine, P. (1992). Toward a practice of autonomous systems: Proceedings of the First European Conference on Artificial Life (The MIT Press).

Wang, Z., and Liang, X. (2006). A graph based simulation of reorganization in multi-agent systems. In Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp. 129–132.

Weyns, D., Haesevoets, R., Helleboogh, A., Holvoet, T., and Joosen, W. (2010). The MACODO middleware for context-driven dynamic agent organizations. ACM Transactions on Autonomous and Adaptive Systems (TAAS) *5*, 3.

Wooldridge, M., Jennings, N.R., and Kinny, D. (2000). The Gaia methodology for agent-oriented analysis and design. Autonomous Agents and Multi-Agent Systems *3*, 285–312.

Zhong, C., and DeLoach, S.A. (2011). Runtime models for automatic reorganization of multi-robot systems. In Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 20–29.