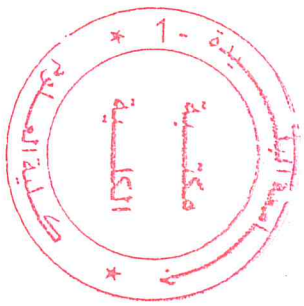
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad DAHLEB de Blida

Faculté des sciences

Département d'informatique



Mémoire

MASTER ACADEMIQUE

Domaine : Mathématiques et d'Informatique

Filière : Informatique

Spécialité: Ingénierie Logiciel

THEME

Un méta modèle permettant la modélisation des capacités des sous système à travers la notion de rôle

Rapport présenter par : Braik Abdellah

Kerbiche Oussama

Directrice de mémoire : Mme Cherfa Imène

Devant le jury :

Présidente : - Mme ABED

- Mme MADANI

Année Universitaire : 2016 /2017

Remerciements

Nous souhaitons manifester nos sincères remerciements à:

- الله tout puissant، de nous avoir permis d'arriver à ce niveau d'étude، et aussi pour nous avoir donné beaucoup de patience et de courage pour réaliser ce mémoire.

- Nos deux familles pour leur compréhension et leur présence tout au long de cette dure année.

- Mme I.Cherfa, notre promotrice, qui nous a suivis et prodigués de précieux conseils pour réaliser ce travail.

- Nos remerciements s'adressent aux membres du jury qui ont bien accepté de juger et d'´évaluer ce travail.

- Nos remerciements s'adressent à tous les enseignants qui ont contribué à notre formation durant les cinq ans d'études à cette université.

- A tous ceux qui de près ou de loin, ont contribué par leurs conseils, leurs encouragements et leurs amitiés, à l'édification de ce projet.

A toutes ces personnes, merci du fond du cœur

Résumé

Un système de systèmes (SoS) est un ensemble de systèmes autonomes interconnectés et coordonnés dont chacun possède ses propres capacités pour satisfaire des fonctions spécifiques que les systèmes indépendamment ne pourraient réaliser.

L'objectif de ce travail est de proposer une notation graphique pour la modélisation des capacités des sous système à travers la notion de rôle. Pour ce faire nous avons suivi un processus de deux phases dont la première phase consiste à faire une analyse ontologique pour spécifier les concepts nécessaires afin de proposer un méta modèle. La seconde phase consiste à introduire des éléments graphiques correspondant aux concepts du métamodèle proposé.

Nous avons développé un outil permettant la manipulation directe des concepts pour réaliser un diagramme de rôles. Une étude de cas a été utilisée pour valider l'outil.

Table des Matières

INTRODUCTION GENERALE 1

CONTEXTE 1
PROBLEMATIQUE 1
OBJECTIF 1
ORGANISATION DU MEMOIRE 1

PARTIE I: ETAT DE L'ART

CHAPITRE I: LES SYSTEMES DE SYSTEMES

I. INTRODUCTION 2
II. LES SYSTEMES DE SYSTEMES (SOS) 2
II.1 Définition de SOS 2
II.2 Caractéristiques de SOS 2
II.3 Les types de SoS 4
II.3.1 Virtuel (virtual) 4
II.3.2 Collaboratif (collaborative) 5
II.3.3 Reconnu (SoS Acknowledged) 5
II.3.4 dirigé (SOS directed) 6
II.4 Exemple de SoS 7
III. SPECIFICATIONS DES CAPACITES 7
III.1 La Capacite Maturite 8
III.1.1 Capacité actuelle (Currant capability) 8
III.1.2 Capacité héritée (Legacy capability) 8
III.1.3 Capacité de développement (Development capability) 9
III.2 Types De Capacités 9
III.2.1 Les capacités Technique 9
III.2.2 Les Ressources socio-techniques 9
III.2.3 Les capacités Manuelles 9
III.2.4 Les Ressources d'information 9
III.2.5 Les Ressources du personnel 10
III.3 Exemple de notation graphique pour la spécification des capacités 10
IV CONCLUSION 11

CHAPITRE II: METHODOLOGIES DE SPECIFICATION DES ROLES ET DES CAPACITES

I. INTRODUCTION 12
II. LES DIMENSIONS DE COMPARAISONS 12
II.1 La dimension de coopération et l'interaction 12
II.2 La dimension de rôle 12

II.3 La dimension de capacité 13
II.4 La dimension de l'émergence 13
III ANALYSE DES METHODES 13
III.1 AALAADIN 13
III.2 CASSIOPEE 16
III.3 MaSE 17
III.4 GAIA 20
III.5 TROPOS 22
III.6 PROMETHEUS 25
IV. ANALYSE DE LA COMPARAISON 27
IV.1 Synthèse de la comparaison des méthodes 27
IV.2 Conclusion de la comparaison 28
V. CONCLUSION 28

CHAPITRE III : LA META-MODELISATION

I. INTRODUCTION 30
II. LA META MODELISATION 30
III. LES METAS MODELES 32
III.1 La définition des métas modèles 32
III.2 Exemple De Meta modelé 33
III.3 Création de métamodèle selon MOF1.4 34
III.3.1 Classes (Meta classe) 35
III.3.2 Associations (Meta- Associations) 37
III.3.3 Agrégation 38
III.3.4 Références 40
III.3.5 Data Types 40
III.3.6 Package 40
IV. CONCLUSION 41

PARTIE II: CONTRIBUTION

CHAPITRE IV: METAMODELE POUR LA SPECIFICATION DES ROLES ET DES CAPACITES

I. INTRODUCTION 42
II. DESCRIPTION DE LA DEMARCHE UTILISEE 42
III. PHASE 1 : DEFINITION DES CONSTRUCTIONS SEMANTIQUES 43
III.1 Analyse ontologique 43
III.2 Définition des constructions ontologiques 44
III.3 Définition des constructions sémantique 44
IV. PHASE 2 : DEVELOPPEMENT DES CONSTRUCTIONS GRAPHIQUES 46
IV.1 Principe de la clarté sémiotique 47
IV.2 Principe de la discrimination perceptive 48
IV.3 Principe de l'expressivité visuelle 49

IV.4 Principe de la transparence sémantique 50
IV.5 Principe du double codage 50
IV.6 Principe de l'économie graphique 51
V. L'UTILISATEUR FINAL DE LA NOTATION GRAPHIQUE 51
VI. CONCLUSION 52

CHAPITRE V: IMPLEMENTATION

I. INTRODUCTION 53
II. ENVIRONNEMENT DE TRAVAIL ET LES OUTILS UTILISES 53
II.1 Java 8 53
II.2 Le Package Eclipse Modeling Tools dans Eclipse Neon.3 53
II.3 Eclipse Modeling Framework (EMF) 54
II.4 Graphical Modeling Framework (GMF) 54
II.5 le langage de contrainte d'objet (OCL) 54
III. PRESENTATION DE L'APPLICATION 55
III.1 Crée un diagramme 55
III.2 Dessiner un diagramme 55
III.3 remplir les propriétés et valider le diagramme 56
IV ETUDE DE CAS «URGENCE SOS » 56
IV.1 Présentation du cas d'étude 56
IV.2 Modélisation de l'étude de cas 56
V. CONCLUSION 59

CONCLUSION GENERALE 61

Liste des figures

CHAPITRE I: LES SYSTEMES DE SYSTEMES

Figure 1: SOS Virtuel 4
Figure 2: SOS Collaborative 5
Figure 3: SOS Reconnu 6
Figure 4: SOS dirigé 6
Figure 5: Approvisionnement réseau 11

CHAPITRE II: METHODOLOGIES DE SPECIFICATION DES ROLES ET DES CAPACITES

Figure 6: Le métamodèle UML d'AGR 14
Figure 7: Les diagrammes de structure organisationnelle de Aalaadin. 15
Figure 8: Les cinq calques de Cassiopée 17
Figure 9: Les différentes étapes et les modèles de MaSE. 18
Figure 10: une hiérarchie générale des objectifs dans MaSE 19
Figure 11: Exemple d'un modèle de rôles dans MaSE 20
Figure 12: Exemple de modèle de rôle 21
Figure 13: Exemple de modèle d'interaction 21
Figure 14: Notations utiliser dans Les diagrammes de vue du système 27
Figure 15: Notations utiliser dans Les diagrammes de vue des agents 27

CHAPITRE III: LA META-MODELISATION

Figure 16: Activité de modélisation et de méta-modélisation dans les termes de Minsky 30
Figure 17: Relation entre modèles et métamodèles 32
Figure 18: Métamodèle des diagrammes de cas d'utilisation 34

CHAPITRE IV : METAMODELE POUR LA SPECIFICATION DES ROLES ET DES CAPACITES

Figure 19 : le processus global de la démarche suivi 42
Figure 20: les anomalies d'analyse ontologique 43
Figure 21: le méta modèle proposé 45
Figure 22: Les principes de la conception des notations visuelles 46
Figure 23: les anomalies de la clarté sémiotique 48
Figure 24: les variables visuelles. 49
Figure 25: Cas d'utilisation de la chaine d'outils 51

CHAPITRE V: IMPLEMENTATION

Figure 26: Créé un diagramme rôle-capacité 55
Figure 27: Dessiner un diagramme rôle-capacité 55

Figure 28: Remplir les propriétés et valider le diagramme 56
Figure 29: La structure des systèmes de l'étude de cas..... 57
Figure 30: La structure du rôle collaboratif « la réanimation » 58
Figure 31: La structure du rôle collaboratif « le bloc opératoire » 59

Liste des tableaux

Tableau 1: Les modèles associés aux étapes de la méthodologie MaSE 18
Tableau 2: Synthèse de la comparaison des différentes méthodes 28
Tableau 3: Hiérarchie selon une méta-modélisation par objets 31
Tableau 4: les propriétés de l'attribut 36
Tableau 5: les propriétés de l'opération 36
Tableau 6: les propriétés de l'association 38
Tableau 7: la notation graphique proposée 47

Introduction générale

Contexte

Les systèmes de systèmes offrent des capacités globales sur des systèmes indépendants (appelés sous-système du SoS) les uns des autres, sujets à des domaines d'utilisation divers et variés, et pouvant évoluer pour répondre à de nouvelles formes de besoins d'utilisation. L'environnement d'un système de systèmes est constitué de tous ces sous-systèmes dont chacun possède ses propres capacités.

L'environnement d'un système de système est hétérogène et dynamique. Un système de système peut évoluer au cours du temps pour répondre à certaines situations, ou à sa propre mission.

Problématique

Pour spécifier la mission d'un système de système, une bonne connaissance des capacités des sous-systèmes semble indispensable. Une capacité peut être indépendante, ou peut dépendre aux capacités de d'autres systèmes.

Le problème qui se pose est que la connaissance des systèmes constituants dans la phase d'analyse et de conception n'est pas toujours évidente. Alors ce n'est qu'au moment de l'exécution qu'on découvre réellement les sous-systèmes qui composent le SoS.

Objectif

L'objectif de ce travail est d'offrir un moyen qui permet de décrire au préalable les rôles pouvant intervenir dans le SoS. Pour le faire nous allons proposer un méta modèle permettant la modélisation des capacités des sous système à travers la notion de rôle. Cette dernière permet de réaliser une modélisation indépendante de tout système, ce qui offre un niveau d'abstraction et un potentiel d'évolutivité.

Organisation du mémoire

Notre mémoire se compose de cinq chapitres organisés en deux parties:

- La partie état de l'art se compose de trois chapitres :

- Le premier chapitre introduit la notion du système de systèmes (SoS) avec ces différentes caractéristiques, et traite la spécification des capacités qu'un système donné met dans un SoS.

- Dans le deuxième chapitre, nous présentons les méthodes de développement de systèmes complexes qui nous offrent un moyen de décrire les rôles et les capacités. Le but de ce chapitre est de comparer les méthodes existantes et identifier leurs limites.

- Dans le troisième chapitre nous présentons la notion de méta modélisation et les métas modèles ainsi que les différents concepts liés à la méta-modélisation.

- La partie contribution se compose de deux chapitres :

- Le quatrième chapitre, qui consiste à proposer un méta modèle pour spécifier les rôles et les capacités et introduire une notation graphique pour ce méta modèle (syntaxe abstraite et syntaxe concrète).

- Dans le dernier chapitre, une implémentation de notre méta-modèle avec la plate-forme Java 8 est présentée. Ce chapitre comporte aussi une présentation de notre application ainsi qu'un déroulement avec une étude de cas.

- Finalement, on termine par une conclusion générale et quelques perspectives.

Partie I: Etat de l'art

CHAPITRE I: LES SYSTEMES DE SYSTEMES.....1

CHAPITRE II: METHODOLOGIES DE SPECIFICATION DES ROLES ET DES CAPACITES.....12

CHAPITRE III : LA META-MODELISATION..... 30

Chapitre I: Les systèmes de systèmes

I. Introduction

Bien que l'importance des «systèmes de systèmes» soit de plus en plus reconnu, il y'a eu peu d'accord sur ce qu'ils sont, ou sur quels principes ils devraient être construits.
Le but de ce chapitre est de décrire les systèmes de systèmes, une nouvelle classe de systèmes, nous présentons les différentes définitions qui existent dans la littérature, les caractéristiques, ainsi que les différents types qui existent.
Dans la seconde partie, nous décrivons la spécification des capacités qu'un système donné met dans un SoS, ainsi qu'une classification de capacités. Nous terminons ce chapitre par une comparaison des types de capacités.

II. Les systèmes de systèmes (SOS)

II.1 Définition de SOS

Des études détaillées de la littérature et des discussions sur les différentes définitions de SoS sont présentées dans [1] selon leurs propres mérites et leur application.
La définition unificatrice est que les systèmes de systèmes sont des systèmes intégrés à grande échelle qui sont hétérogènes et indépendamment exploitables par eux-mêmes, ils sont mis en réseau ensemble pour réaliser un objectif commun (l'objectif peut être le coût, la performance, la robustesse, etc.).[1]

II.2 Caractéristiques de SOS

En réponse à l'absence d'une définition claire et précise des SoS, Maier [5] distingue un système de systèmes de tout autre classe de systèmes en terme de cinq principales caractéristiques, parfois désignées par l'acronyme "OMGEE" (**O**perational, **M**anagerial, **G**eographically, **E**mergent, **E**volutionary) [2] ces caractéristiques sont [3]:

- **L'Indépendance Opérationnelle (*Operational Independence*):** L'indépendance opérationnelle des systèmes constituants signifie qu'ils doivent être capables de remplir leur propre mission indépendamment du SoS, lorsque les sous-systèmes sont séparés du SoS.

Cela signifie que chacun de ces sous-systèmes est indépendant et a une utilisation qui lui est propre. Comme exemple, le SoS de la force militaire navale, et ses sous-systèmes tels que les avions.

- **L'Indépendance Managériale** (*Managerial Independence*) **:** L'indépendance managériale des systèmes constituants signifie que les systèmes constituants doivent être autonomes, et doivent avoir une existence et une organisation indépendante du SoS. En d'autres termes, les systèmes constituants doivent être gérés indépendamment du SoS, ils doivent fonctionner de manière indépendante.

- **La Distribution géographique** (*Geographically distributed*) **:** La répartition géographique des systèmes constitutifs signifie qu'ils sont situés dans des endroits différents; Cette distribution géographique est relative et dépend largement des technologies disponibles et des moyens de communication. Les systèmes peuvent échanger des informations, mais ne peuvent échanger des quantités considérables d'énergie ou de matière. Cette dernière restriction permet de confiner le périmètre étudié.

- **Le Comportement Emergent** (*Emergent Behaviour*): L'apparition de comportements émergents dans une SoS signifie que le système global (SoS) présente des propriétés et des fonctionnalités qui ne sont pas présentes dans l'un de ses systèmes constitutifs. Les comportements émergents ne peuvent pas être localisés ou attribués à l'un de ces systèmes constitutifs. De plus, ces comportements ne peuvent pas toujours être anticipés, ce qui entraîne des difficultés à valider le système global.

Un exemple de comportement émergent est le routage IP (protocole Internet), qui est la base du réseau Internet. Aucun routeur IP ne connaît la topologie complète des interconnexions d'internet, ni même la configuration des interconnexions locales dans son voisinage. Comme la configuration des liaisons entre les routeurs change constamment, de même que la bande passante sur un lien donné, les tables de routage font constamment référence à une situation passée. Et pourtant, le routage IP est un processus efficace qui peut être digne de confiance, et qui transfère de manière efficace les paquets de la source à la destination prévue. Chaque routeur IP détermine quel routeur à proximité constituera le prochain point de progression, même s'il n'a aucune connaissance des routeurs ou des chemins potentiels accessibles dans son voisinage immédiat. Le routage IP est donc un protocole à comportement émergent, qui fonctionne également avec des informations incomplètes, imprécises et obsolètes, tout en fournissant un résultat efficace. [3]

- **Développement Évolutif** (*Evolutionary Development*) **:** Puisque le développement et la configuration des systèmes sont évolutifs, le système global (SoS) n'est jamais complètement formé. Son développement peut être progressif (les systèmes constitutifs ne sont pas nécessairement disponibles en même temps) et/ou évolutifs dans le temps (des fonctionnalités peuvent être ajoutées, effacées ou modifiées en fonction de l'expérience acquise ou l'évolution des besoins). De plus, chaque système constitutif peut avoir son propre cycle de vie, distinct du système global.

II.3 Les types de SoS

Les systèmes de systèmes sont classés en quatre catégories fondamentales selon la complexité, l'étendue, et le contrôle de gestion. Cette dernière est une dimension qui a été ajouté parce que les systèmes de systèmes d'une complexité et d'une étendue similaires ne devraient pas être considérés comme équivalents. Le contrôle de gestion est essentielle pour distinguer comment les SoS sont géré. Les quatre catégories sont [5] :

II.3.1 Virtuel (virtual)

Ce type de SoS manque d'une autorité centrale de gestion et d'un objectif clair. Il est souvent ad hoc[1] et les systèmes constitutifs ne sont pas nécessairement connus (Figure 1). Un exemple de ce type de SoS est l'Internet et tous les services que l'on peut trouver et intégrer de manière ad hoc [4].

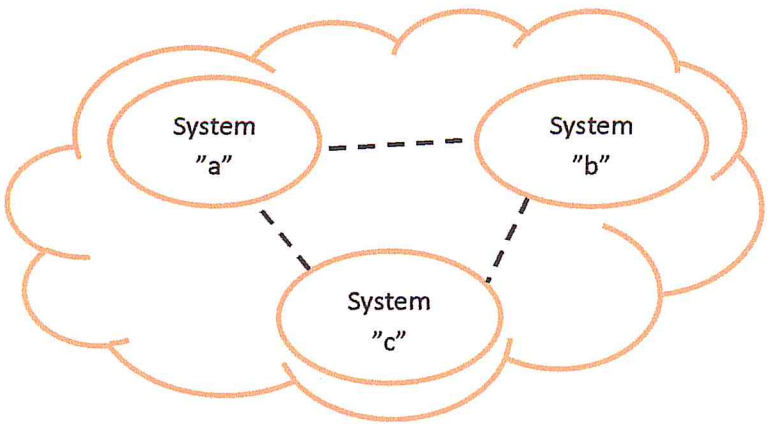


Figure 1: *SOS Virtuel [4]*

[1] **Ad hoc** est une locution latine qui signifie « pour cela ». Elle s'emploie de nos jours pour « qui a été institué spécialement pour répondre à un besoin ».

II.3.2 Collaboratif (collaborative)

Dans le cadre d'un SoS collaboratif, les équipes d'ingénierie des systèmes constituants travaillent ensemble plus ou moins volontairement pour atteindre des objectifs communs convenus. Dans ce type de SoS, il n'y a pas d'équipe d'ingénierie pour guider ou gérer les activités liées seulement aux systèmes constituants [4].

Par exemple l'Internet est un système collaboratif. Le groupe de travail sur l'ingénierie Internet établit des normes, mais n'a aucun pouvoir pour les faire respecter. Les accords entre les acteurs centraux sur la fourniture de services et le rejet fournissent le mécanisme d'exécution qui est nécessaire pour maintenir les normes (voir la Figure 2). L'Internet a commencé comme un système dirigé, contrôlé par l'Agence américaine des projets de recherche avancée, pour partager des ressources informatiques. Au fil du temps, elle est passée du contrôle central à des mécanismes de collaboration non planifiés [5].

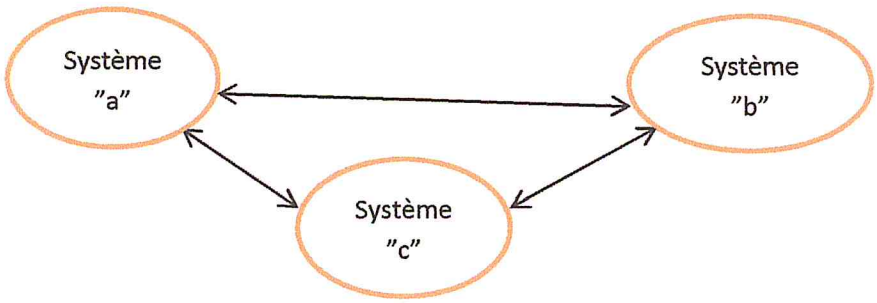


Figure 2: *SOS Collaborative [4]*

II.3.3 Reconnu (SoS Acknowledged)

Dans ce type, le SoS a des objectifs reconnus, un gestionnaire désigné et des ressources. Toutefois, les systèmes constitutifs conservent leur propriété indépendante, leurs objectifs, leur financement et leurs approches de développement et de maintien. La figure 3 illustre cela en utilisant des flèches unidirectionnelles entre l'équipe SoSE (SoS Engineering) et les systèmes constituants. La flèche unidirectionnelle signifie que l'équipe SoSE peut fournir des conseils aux systèmes constituants, mais que les systèmes constituants ne sont pas tenus de se conformer aux demandes du SoSE. [4]

Un exemple de ce type de SoS pourrait être le SoS militaire de commande et de contrôle. Ce dernier fait la transition d'un SoS collaboratif à un SoS reconnu en raison de l'importance des missions soutenues par le SoS ou des complexités des capacités transversales de SoS [4].

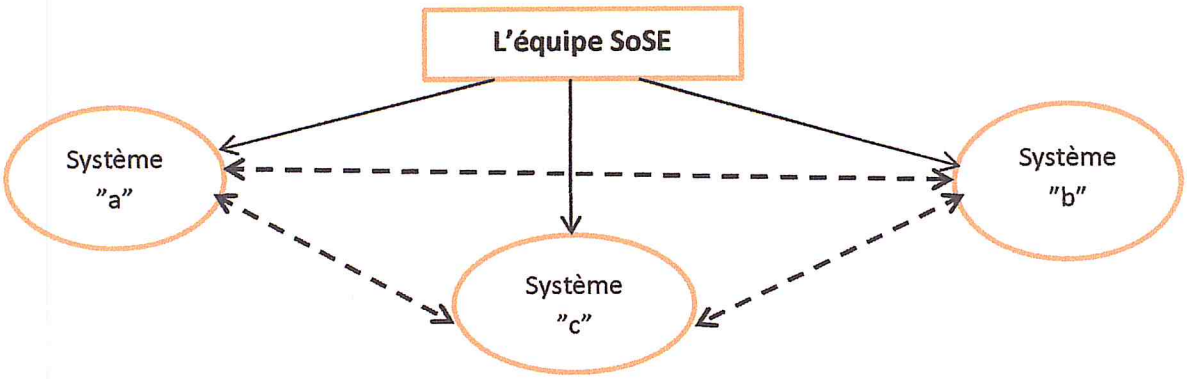


Figure 3: SOS Reconnu [4]

II.3.4 dirigé (SOS directed)

Un SoS dirigé est géré de façon centralisée par une équipe gouvernementale, d'entreprise ou d'intégrateur de systèmes, et construit pour remplir des objectifs spécifiques. Les systèmes constitutifs maintiennent leur capacité à fonctionner de façon indépendante, mais l'évolution est principalement contrôlée par l'organisation et la gestion du SoS. La figure 4 illustre cela en utilisant des flèches bidirectionnelles entre l'équipe SoSE et les principaux systèmes constitutifs (mais pas nécessairement tous) [4].

Les flèches bidirectionnelles indiquent que l'équipe SoSE a l'autorité (et souvent le financement) d'exiger que ces systèmes constituants (souvent par le biais d'une sorte de contrat) développent et soutiennent les capacités SoS.

Par exemple un réseau intégré de défense aérienne est généralement géré de manière centralisée pour défendre une région contre les systèmes ennemis, bien que ses systèmes composants conservent la capacité de fonctionner indépendamment, et le faire quand il est nécessaire sous le stress du combat. [5]

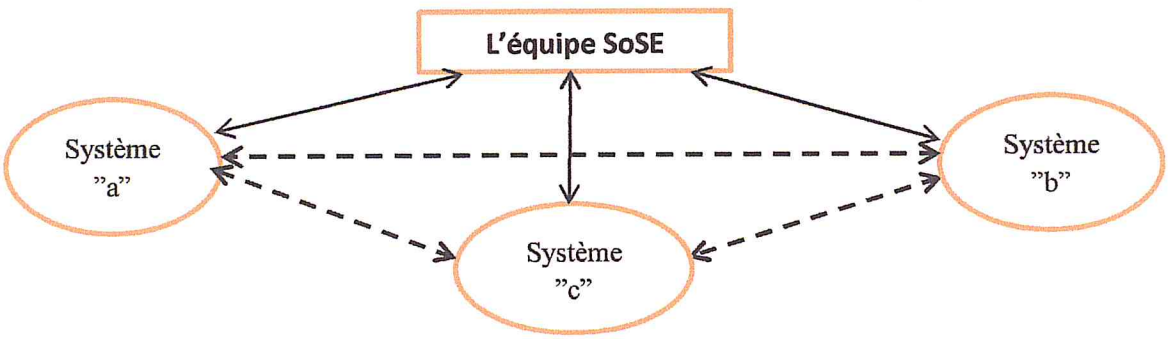


Figure 4: SOS dirigé [4]

II.4 Exemple de SoS

Les défenses aériennes des forces militaires modernes sont des exemples clairs de systèmes de systèmes. Un système intégré de défense aérienne se compose d'un réseau géographiquement dispersé d'éléments semi-autonomes. Il s'agit notamment des radars de surveillance, des systèmes de surveillance passive, des batteries de lancement de missiles, des sites de suivi et de contrôle des missiles, des radars aéroportés de surveillance et de suivi, des avions de chasse et de l'artillerie antiaérienne. Toutes les unités sont reliées entre elles par un réseau de communication avec le commandement et le contrôle appliqués aux centres locaux, régionaux et nationaux [6].

Lorsqu'il fonctionne en tant que système intégré, le réseau présente un comportement émergent à l'échelle du réseau. Par exemple, des stratégies optimales de tir de missiles et d'engagement et une utilisation sélective des radars pour rendre difficile le ciblage des éléments individuels. Cependant, les incertitudes de la guerre font qu'il est essentiel que le système soit capable de se replier efficacement dans des configurations moins intégrées et de faire de telles transitions soudainement et en pleine bataille.

III. Spécifications des capacités

Un SoS est un ensemble de systèmes mettant ensemble leurs capacités pour accomplir une mission, qu'un sous-système seul ne peut pas accomplir. Une capacité d'un sous-système désigne les services qu'un système donné met dans un SoS.

Comme l'environnement d'un SoS est dynamique et change constamment, il est nécessaire de faire une étude préliminaire des capacités fournies par les systèmes constituants. Au moment de la construction d'un SoS, l'expert du domaine d'application peut connaître au préalable quelques capacités offertes par les systèmes constituants.

Dans le domaine des SoS, il y'a peu de travaux qui se sont intéressés à la modélisation des capacités. Dans ce qui suit, nous présentons une classification des capacités prise des travaux de [7].

[7] propose que les capacités puissent être classées par type et par maturité, bien que l'indication de la maturité d'une capacité ne signifie pas nécessairement qu'elle est effectivement utilisée ou non utilisée dans un SoS donné à un moment donné.

III.1 La Capacite Maturite

On distingue trois niveaux de maturité pour les capacités :

III.1.1 Capacité actuelle (Currant capability)

Une capacité actuelle est considérée à la fois comme étant maintenue et disponible pour être utilisée dans un délai raisonnable (l'information étant stockée dans les métadonnées pour une capacité donnée). Cela n'implique pas qu'une capacité donnée soit statique en termes d'évolution, ou qu'elle sera toujours disponible, simplement qu'elle répond aux normes minimales établies pour atteindre les objectifs du SoS.

III.1.2 Capacité héritée (Legacy capability)

Une capacité héritée est celle qui a formé une partie intégrante du SoS dans le passé, mais n'est plus la méthode préférée pour effectuer une opération. Une capacité héritée peut ne pas garder le même niveau qu'une capacité actuelle. Les systèmes techniques en particulier peuvent conserver la capacité d'exécuter des tâches au-delà du temps dont ils sont tenus de le faire. Les procédures organisationnelles sont désaffectées, mais la formation et les connaissances détenues par les personnes concernées sont conservées dans le futur dans un contexte qui diminue de façon décroissante. Il existe de nombreux exemples de ce type de comportement dans le monde réel, nous présentons un exemple ci-dessous :

« *Au sein de l'administration de l'université, dans de nombreux départements, le poste de chef de département est transitoire, changeant à des périodes régulières, mais en l'absence d'un chef de département donné, il peut être plus logique de demander conseil à un chef précédent que de ne pas demander de conseil.* »

L'utilisation des capacités héritée n'est pas sans risques ou même sans coûts, mais elle pourrait accroître la fiabilité d'un SoS donné. Les capacités existantes ont des exigences de formation spécifiques pour le personnel si elles doivent être conservées de manière significative pendant un certain temps. Pour les systèmes techniques, il peut y avoir des risques importants liés au débit et à la compatibilité. Cependant, ces fonctionnalités représentent l'histoire des SoS, quelque chose qui doit être documenté et peut être appelé lorsque cela est nécessaire.

III.1.3 Capacité de développement (Development capability)

Les capacités de développement indiquent l'avenir des SoS dans la mesure où elles documentent les capacités qu'ils auront à l'avenir. Elles peuvent introduire de nouveaux dangers dans un SoS donné, ainsi que supprimer les anciennes capacités.

III.2 Types De Capacités

Les types de capacités sont classés en fonction de leurs caractéristiques dans les catégories suivantes [7] :

III.2.1 Les capacités Technique

Les capacités techniques (Notée T dans la notation, voir figure 5) permettent aux systèmes d'exposer les artefacts au système plus large. Ceux-ci pourraient inclure:

- Les applications, les services.
- Les systèmes d'exploitations / les plateformes.
- Les réseaux.
- Les ressources physiques telles que les PC.

III.2.2 Les Ressources socio-techniques

Les capacités sociotechniques (Notée ST dans la notation, voir figure 5) fournissent des ressources impliquant à la fois des éléments techniques et organisationnels. Par exemple, une capacité de traitement des demandes de remboursement pourrait impliquer à la fois une personne chargée de la comptabilité pour l'autorisation et la vérification, et une composante technique pour assurer le stockage à long terme des données.

III.2.3 Les capacités Manuelles

Les processus manuels (noté M dans la notation, voir figure 5) ne comportent aucun système technique. La prise de décision, par exemple, peut dépendre entièrement sur un agent humain donné.

III.2.4 Les Ressources d'information

Une capacité ressource d'information (notées I dans la notation, voir figure 5) implique que l'information est fournie par un système donné et peut être utilisée par d'autres systèmes ou des capacités exposées individuellement.

III.2.5 Les Ressources du personnel

Certaines unités organisationnelles permettent aux utilisateurs d'utiliser d'autres systèmes ou des capacités individuelles. Par exemple, un parc de véhicules peut fournir des conducteurs de véhicules. Cela sépare la personne du système. Une personne a une formation, et peut faire beaucoup de choses, mais ils ne sont pas un système. La formation du personnel pour des systèmes donnés n'est souvent pas effectuée par ceux qui utilisent le système, mais par du personnel de formation spécifique. Une ressource du personnel est noté P dans la notation (voir figure 5).

Revenant à l'exemple précèdent, on constate que le système est le parc de véhicules, et le conducteur est l'utilisateur de notre système, mais ce dernier ne peut pas former des techniciens pour la maintenance de ce système par contre cette mission est effectué par un personnel de formation spécifique.

III.3 Exemple de notation graphique pour la spécification des capacités [7]

La figure ci-dessous illustre une notation graphique pour la spécification des capacités en utilisant un exemple du programme NPfiT (National Program for Information Technology). Un seul système possédant une capacité héritée a été utilisé dans l'exemple, bien que beaucoup d'autres existent dans le SoS plus large. Il montre les capacités (illustrées à l'aide d'eclipses) de trois systèmes de réseau de communication différents (illustrés à l'aide de rectangles) en usage dans le NHS (National Health Service). Le réseau N3 et le TMS (Transaction and Messaging Spine) livré grâce au nouveau projet NHS Spine peuvent fournir des capacités de transfert de données. La relation entre Spine et le TMS est illustrée à l'aide d'une flèche de décomposition. Cela permet de décomposer les systèmes en sous-systèmes si nécessaire. En réalité, le type de transfert de données qu'ils supportent diffère dans la mesure où le nouveau système TMS peut délivrer des données d'une manière plus sûre que N3, mais l'essence du diagramme est qu'il existe deux moyens actuellement viables de transmission de données. Le NHS Lincolnshire IT Network est un système parallèle et séparé qui est remplacé par TMS. Lorsque TMS et N3 fournissent une formation pour l'utilisation de leurs systèmes en plus de l'accès lui-même, le système Lincolnshire n'est plus utilisé activement et attend le déclassement ou, l'évolution vers un nouveau réseau (COIN).

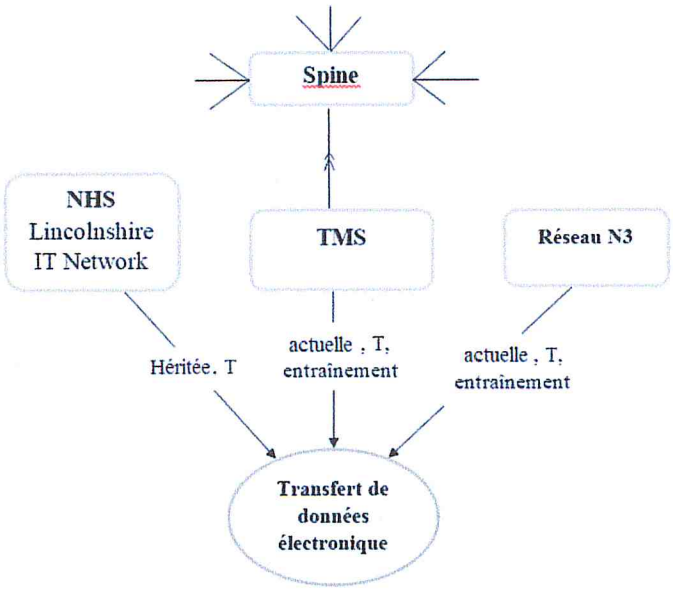


Figure 5: Approvisionnement réseau [7]

IV Conclusion

Nous avons définis dans la première partie de ce chapitre les systèmes de systèmes (SoS), ces caractéristiques ainsi que les types de SoS qui existent. Comme le concept de SoS n'est pas très courant, nous avons présenté un exemple de SoS.

Dans la seconde phase nous avons décrit la spécification des capacités qu'un système donné met dans un SoS. Il n'y a pas de travaux qui existent dans cette piste à part le travail qu'on a présenté. Ce dernier offre une taxonomie riche sur les capacités, ainsi qu'une notation graphique permettant de décrire les capacités des sous-systèmes.

La taxonomie présentée dans ce chapitre est intéressante pour distinguer les capacités. Le seul inconvénient de cette notation est que les sous-systèmes sont instanciés. En effet, lors de la spécification d'un SoS, l'expert du domaine d'application ne connaît pas en détail les sous-systèmes qui vont appartenir au SoS, et ne connaît pas non plus les comportements émergents. De ce fait, nous avons décidé d'élever le niveau d'abstraction et d'utiliser la notion de rôle au lieu du concept de sous-système.

Les seuls travaux qui comportent ces notions sont les méthodologies de développements des systèmes complexes et des systèmes à comportements émergents qui sont surtout basés sur les systèmes multi-agents. Dans le chapitre suivant nous allons comparer comment ces méthodologies permettent de décrire les rôles et les capacités.

Chapitre II: Méthodologies de spécification des rôles et des capacités

I. Introduction

Dans ce chapitre nous allons présenter les méthodes de développement de systèmes complexes et de systèmes à comportement émergent, qui sont tous des méthodes multi-agents. Ces méthodes offrent un moyen pour spécifier les notions de rôle et de capacité. Comme ces méthodes sont nombreuses et évoluent rapidement, il est nécessaire de faire une comparaison de l'existant afin de dégager les manques qu'il y'a dans ces méthodes.

Ce chapitre est composé de trois parties. Nous commençons par déterminer les dimensions de comparaison. Par la suite nous présentons les différentes méthodes. Et enfin, nous allons analyser et comparer les différentes méthodes.

II. Les dimensions de comparaisons

La problématique de la comparaison des méthodes a été souvent soulevée dans la littérature pour aider le concepteur à choisir la méthode la plus adaptée au problème donné ou pour présenter les différences essentielles entre les différentes méthodes.

L'objectif de notre travail est de proposer un moyen permettant la spécification des rôles et des capacités, sachant que les rôles peuvent coopérer entre eux. De là, nous définissons les dimensions de comparaison que nous allons détailler dans ce qui suit.

II.1 La dimension de coopération et l'interaction

Dans un SoS, les systèmes constituants coopèrent en mettant ensemble leurs capacités afin d'avoir des capacités plus élevées. Donc cette dimension s'avère importante pour notre travail.

II.2 La dimension de rôle

Les rôles sont un concept fondamental utilisé, en tant que blocs de base, pour la modélisation de la structure organisationnelle des systèmes multi-agents, pour la spécification des protocoles d'interaction, et enfin pour la spécification fonctionnelle du comportement des agents [11]. Nous voulons utiliser ce concept pour la construction d'un SoS. Donc, cette dimension est essentielle pour notre travail.

II.3 La dimension de capacité

La construction d'un SoS repose sur les capacités fournies de chaque système constituant. Comme nous voulons utiliser un concept plus abstrait que le système constituant qui est le concept de rôle, on doit pouvoir spécifier pour chaque rôle les capacités qui lui sont attribuées.

Pour cette dimension, nous allons utiliser les critères de maturité et de type présentés déjà dans le chapitre précédent, vu qu'elles sont utiles dans le domaine des SoS.

II.4 La dimension de l'émergence

L'émergence se traduit par l'apparition d'un nouveau comportement (désirable ou non désirable) qui résulte de la coopération entre les systèmes constituants d'un SoS. Quand on parle de comportement émergent au niveau du SoS, ça signifie que de nouvelles capacités sont apparues suite à la coopération. Cette dimension est très importante dans notre travail.

III Analyse des méthodes

Dans ce qui suit, nous allons présenter les principales méthodes permettant de spécifier les rôles et les capacités. Pour chacune de ces méthodes, nous allons faire une description suivie des concepts clés de celle-ci en vérifiant les différentes dimensions qu'on a vues dans la section II et on termine avec ces notations.

III.1 AALAADIN

AALAADIN est un méta-modèle générique de systèmes multi-agents basé sur les trois concepts principaux d'agents, de groupes et de rôles. C'est un méta-modèle pour décrire les organisations parce qu'on peut construire différents modèles d'organisations dans AALAADIN, comme les organisations de marché ou hiérarchiques. [17]

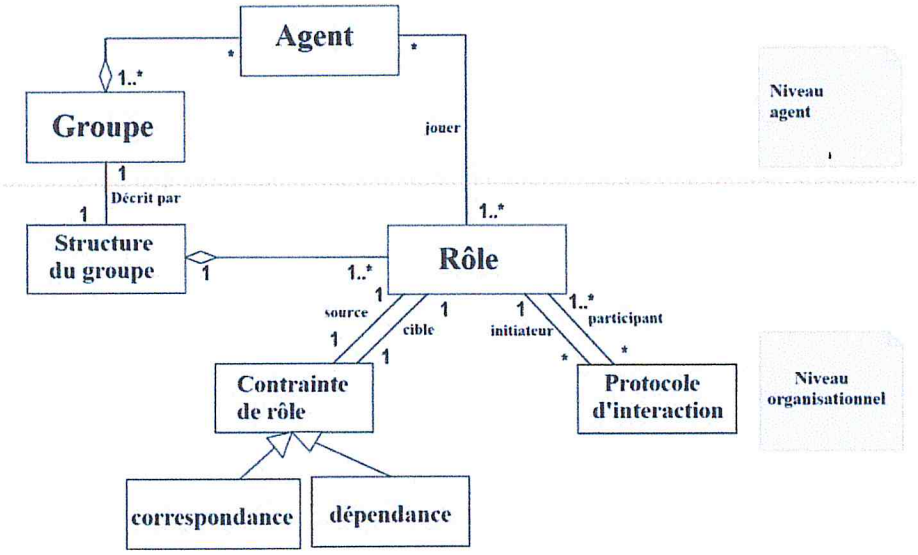


Figure 6: *Le métamodèle UML d'AGR* [17]

Les concepts clés.

Aalaadin va de pair avec le modèle AGR (**Agent**, **Groupe**, **Rôle**) (**Figure 6**) qui basé sur les concepts suivants [12]:

Agent : Quasiment aucune contrainte n'est posée sur l'architecture interne ou sur le modèle de l'agent. L'*agent* est simplement décrit comme une entité autonome communicante qui joue des *rôles* au sein de différents *groupes*. Cette très faible sémantique est volontaire. Le but est de laisser toute liberté au concepteur pour choisir l'architecture appropriée à ses besoins.

Groupe : Le groupe est la notion primitive de regroupement d'agents. Chaque agent peut être membre d'un ou plusieurs groupes. D'une façon un peu simpliste, un groupe peut être vu comme un moyen d'identifier par regroupement un ensemble d'agents. Plus classiquement, associé au rôle, il définira la structuration organisationnelle d'un SMA usuel. Les différents groupes peuvent par ailleurs se recouper librement.

Rôle: Le rôle est la représentation abstraite d'une fonction, d'un service ou d'une identification d'un agent au sein d'un groupe particulier. Un rôle est caractérisé par son unicité (vraie ou fausse), ses *compétences* (ou *services*), et ses *capacités* afin de remplir son rôle .Chaque agent peut avoir plusieurs rôles, un même rôle peut être tenu par plusieurs agents, et les rôles sont locaux aux groupes.

Les organisations obtenues peuvent être hiérarchisées par composition de groupes, grâce à des agents frontières appartenant à deux groupes par exemple. Aalaadin propose une vision centrée organisation plutôt que centrée agent pour des raisons de sécurité, de normes et de responsabilités [12]. Une organisation peut être vue à deux niveaux (voir figure 7) :

- *La structure organisationnelle* ce qui persiste lorsque des composants ou des individus entre ou quittent une organisation, c'est-à-dire les relations qui font un ensemble d'éléments un tout.
- *l'organisation concrète* est une instance de structure organisationnelle avec des agents, des rôles et des groupes particuliers.

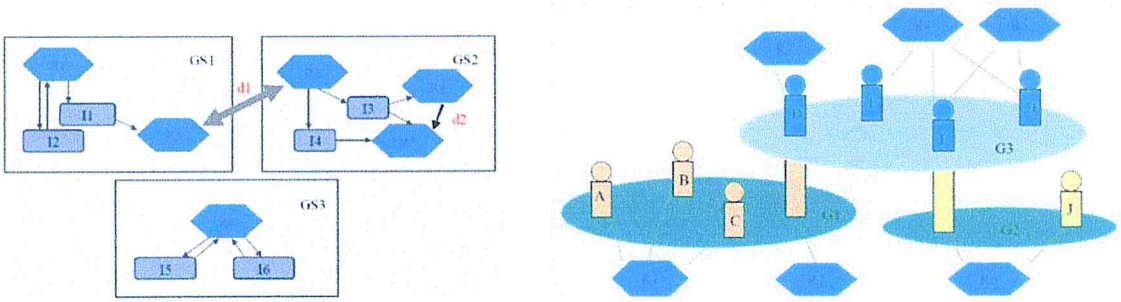


Figure 7: *Les diagrammes de structure organisationnelle (à gauche) et d'organisation concrète (à droite) dans Aalaadin.* [12]

Les notations.

La modélisation graphique d'une organisation suivant AGR peut passer par trois types de diagrammes différents : [12]

Les diagrammes de structures organisationnelles : Dans cette notation, les structures de groupe, c'est-à-dire la représentation abstraite des groupes, sont représentées comme des rectangles dans lesquels se trouvent les rôles, qui représentés en hexagones. Les contraintes sont représentées par des flèches entre les rôles. Nous utilisons deux types de flèches. Les grandes flèches sont utilisées pour la correspondance et des flèches fines sont utilisées pour modéliser les dépendances. (Voir figure 7) ;

Les diagrammes d'organisations concrètes (*cheeseboard*) dans le diagramme de cheeseboard, Un groupe est représenté comme un ovale qui ressemble à un tableau. Les agents sont représentés comme des quilles qui se trouvent sur le tableau et parfois par le biais du tableau lorsqu'ils appartiennent à plusieurs groupes. Un rôle est représenté comme un Hexagone et

une ligne relie cet hexagone aux agents. La figure 7 donne un exemple d'organisation concrète en utilisant le diagramme de fromage. Dans cette image, l'agent F est membre de G2 et G3, jouant les rôles R4 et R5 dans G2 et R6 dans G3.

Les diagrammes organisationnels d'interaction spécifient les interactions entre groupes et/ou rôles. Ce sont des diagrammes de séquences type UML, dans lesquels les lignes de vies sont attribuées à des rôles et non à des instances de classes, comme dans A-UML (Agent UML). Ces diagrammes permettent notamment de représenter les créations ou destructions de groupes.

III.2 CASSIOPEE

La méthode de Cassiopée est un moyen d'aborder un type de résolution de problèmes où les comportements collectifs sont mis en service à travers un ensemble d'agents. Il n'est pas ciblé sur un type spécifique d'application ni exige une architecture d'agents donnée. Cependant, on suppose que bien que les agents puissent avoir des objectifs différents, le but du concepteur est de les faire se développer de manière coopérative [13].

Les concepts clés.

Cassiopée s'appuie sur plusieurs concepts, à savoir ceux du rôle, de l'agent, de la dépendance et du groupe. Un agent est vu comme un ensemble de rôles organisés en trois niveaux [13] :

Les rôles individuels, qui sont les différents comportements que les agents sont individuellement capables d'effectuer, indépendamment de la politique qu'ils choisiront de les exécuter.

Les rôles relationnels, c.-à-d. comment ils choisissent d'interagir les uns avec les autres (en activant / désactivant les rôles individuels), en ce qui concerne les dépendances mutuelles de leurs rôles individuels.

Les rôles organisationnels, ou comment les agents peuvent gérer leurs interactions pour devenir ou rester organisés (en activant/désactivant leurs rôles relationnels).

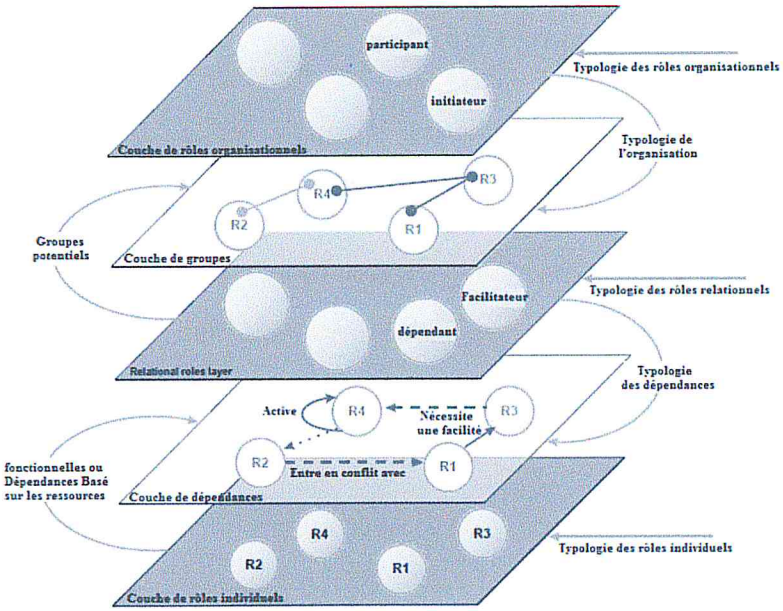


Figure 8: Les cinq calques de Cassiopée (en gris le micro-niveau, en blanc le macroniveau)[13]

Les notations.

Selon l'auteur de [15], la spécification des comportements des agents repose principalement sur deux types de graphes dans Cassiopée : les graphes de couplage des comportements élémentaires, et les graphes d'influences (voir figure 8). Ces graphes sont des graphes de type états/dépendances. Les nœuds représentent les états possibles d'un agent, et les flèches spécifient des dépendances en termes de coopération entre les états par exemple. Les graphes d'influences permettent de dériver des rôles relationnels et des types d'agents par groupement d'états.

III.3 MaSE

MaSE est une méthodologie pour développer un Système multi-agent hétérogène. L'objectif de MaSE est de guider un développeur de système d'une spécification de système initiale à une implémentation de système multi-agent. Cela se fait en dirigeant le concepteur à travers un ensemble de modèles de système interconnectés. Bien que la majorité des modèles MaSE soient graphiques et dérivés des modèles UML standard pour décrire les types d'agents dans un système et leurs interfaces aux autres agents [18]

Cette méthodologie est composée de deux phases : une phase d'analyse et une phase de spécification, chaque phase consiste en plusieurs étapes, voir figure 9.

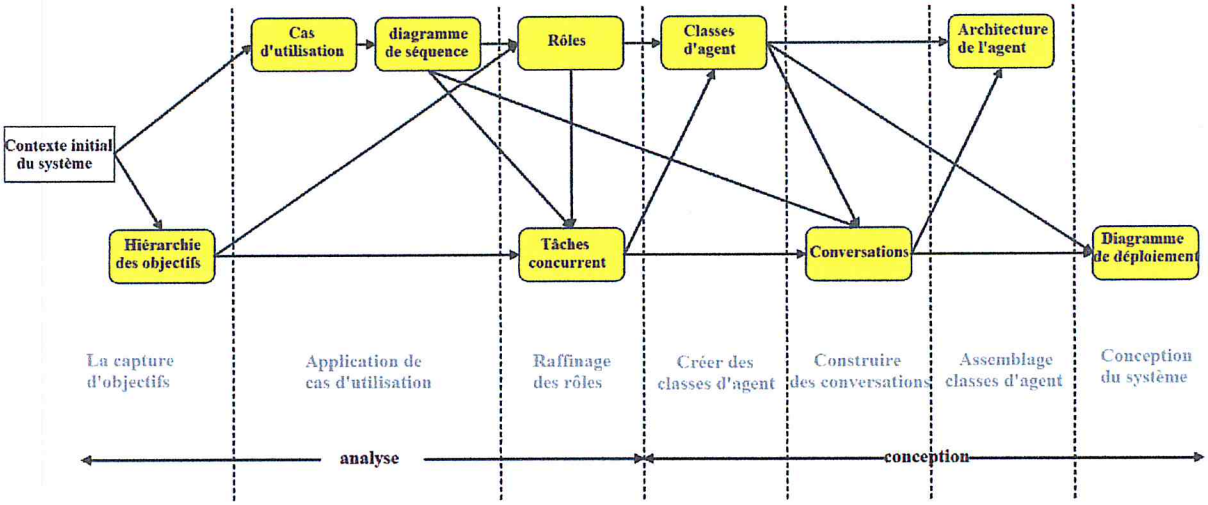


Figure 9: *Les différentes étapes et les modèles de MaSE.* [15]

Les concepts clés.

MaSE manipule neufs modèles associés à chaque étape comme le montre le tableau suivant :

Phases	Modèles
1) La phase d'analyse	
- capturer les objectifs	Hiérarchie des objectifs
- appliquer les cas d'utilisation	Cas d'utilisation, diagrammes de sequence
- raffiner les rôles	Tâches concurrentes, modèles de role
2) La phase de conception	
- construire les classes d'agent	Diagrammes de classes d'agent
- construire les communications	Diagrammes de communications
- rassembler les classes d'agent	Diagrammes d'architecture d'agents
- conceptualiser le système	Diagrammes de déploiement

Tableau 1: *Les modèles associés aux étapes de la méthodologie MaSE* [14]

Les notations.

Selon Gauthier Picard [15], les nombreux modèles de MaSE utilisent des notations directement inspirées de la conception objet. La hiérarchie de but est simplement définie grâce à un arbre dont les nœuds sont les buts et les liens représentent les décompositions. Les diagrammes de séquences sont quasi-similaires aux diagrammes UML du même nom, avec la différence que ce sont des rôles et non des classes qui communiquent. Les cas d'utilisations apparaissent au même niveau que les buts. Les tâches sont modélisées grâce à des automates à états finis, ainsi que les protocoles. Les diagrammes d'agents sont des diagrammes de classes UML. Enfin, les diagrammes de déploiement sont aussi très proches de la notation UML.

Dans MaSE Les propriétés sociales (groupe, organisation) ne sont pas non plus clairement définies, contrairement aux normes (règles organisationnelles) ou protocoles (conversations). MaSE fournit un outil, *agentTool*, qui permet une génération de code partielle [29]. La figure 10 montre un exemple d'une hiérarchie générale des objectifs :

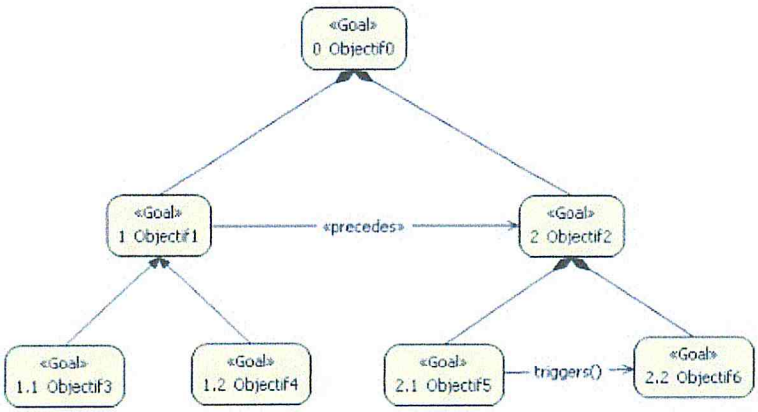


***Figure 10:** une hiérarchie générale des objectifs dans MaSE* [14]

Les relations hiérarchiques entre les objectifs sont des relations AND ou OR, la notation d'agrégation d'UML est utilisée pour présenter le raffinement AND, et la notation de généralisation d'UML est utilisée pour présenter le raffinement OR. Dans la figure précédant, l'objectif Objectif0 peut être réalisé si les deux objectifs *Objectif1* et *Objectif2* sont réalisés. L'objectif *Objectif1* peut être réalisé si l'objectif *Objectif1* ou l'objectif *Objectif2* est réalisé. Un objectif peut être réalisé avant ou après un autre objectif (*Objectif1* doit être réalisé avant *Objectif2*); la réalisation d'un objectif peut déclencher un autre objectif (*Objectif5* déclenche *Objectif6*). Si un objectif n'a pas un déclencheur, il sera réalisé suite à l'initialisation du système. Un déclencheur est représenté par une flèche entre deux objectifs avec le nom de l'événement déclenchant et l'ensemble des paramètres : *événement* $(p_1, \ldots p_n)$. [14]

A partir du diagramme hiérarchique des objectifs, les objectifs sont transformés en rôles et tâches associés. Chaque objectif doit être associé à un rôle, et chaque rôle doit être joué par un agent. MaSE ne modélise pas les interactions utilisateur-ordinateur, mais on peut créer un rôle spécifique représentant l'interface utilisateur, les autres ressources comme des fichiers, des bases de données,…etc.

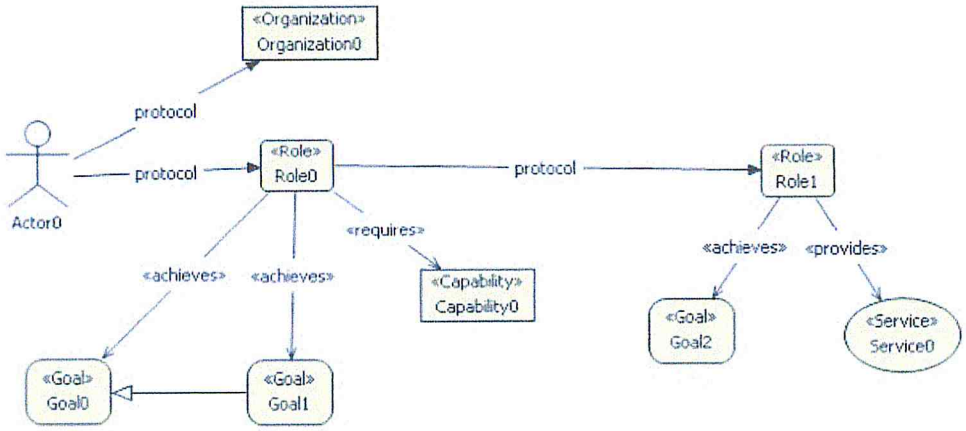


***Figure 11:** Exemple d'un modèle de rôles dans MaSE* [14]

Dans l'exemple de la figure 11, Le rôle *Role0* accomplit les objectifs *Goal0* et *Goal1*, et il exige la capacité *Capability0*. Le rôle *Role1* accompli l'objectif *Goal2* et fournit le service *Service0*. La relation entre *Goal0* et *Goal1* est une relation de précédence (*Goal0* précède *Goal1*). Des protocoles gèrent les relations entre les rôles et les acteurs. [14]

III.4 GAIA

Est une méthodologie pour l'analyse et la conception des systèmes. La méthodologie de Gaia est à la fois générale, car elle permet de développer tous les types de systèmes et complète, car elle traite à la fois du niveau macro (Sociétal) et les aspects micro-niveau (agent) des systèmes. Gaia est fondée sur la vue d'un système multi-agent au tant que organisation composée de différents rôles interactifs [19].

Les concepts clés.

Gaia se veut être générale et applicable à n'importe quel domaine, et compréhensible par la distinction entre macro-niveau et micro-niveau. Les agents modélisés, pouvant être hétérogènes, sont des systèmes computationnels à gros grain, qui vont essayer de maximiser une mesure de qualité globale. Toutefois, Gaia ne prend pas en compte les systèmes admettant de réels conflits. L'organisation (d'une centaine d'agents environ) est clairement statique dans le temps, de même que les services offerts par les agents [15].

Gaia manipule six modèles d'analyse et de conception différents :

_ *Le modèle de rôle* qui identifie les différents rôles devant être joués par les agents du système. Le modèle de rôle est décrit par une table comprenant quatre parties. La première partie correspond à une description textuelle du rôle. La deuxième partie décrit les activités et les protocoles correspondant au rôle. La troisième partie décrit les permissions. Dans cette partie les ressources sont représentées par des variables labellisées et les permissions par des attributs décrivant les droits octroyés pour manipuler les ressources (lecture, écriture, etc.). Enfin la quatrième partie décrit les responsabilités du rôle à travers des propriétés de sûreté (*safety*) et de vivacité (*liveness*). Celles-ci sont exprimées en utilisant les opérateurs de logique temporelle. Un exemple du modèle de rôle représentant le rôle d'un reviewer est donné dans la figure 12 [10].

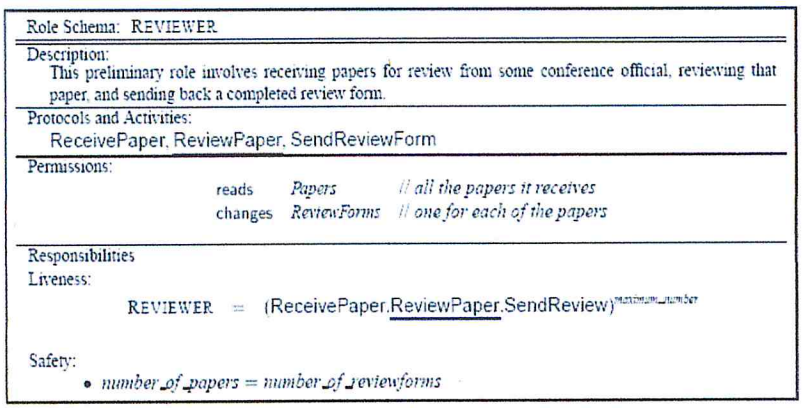


Figure 12: *Exemple de modèle de rôle* [10]

_ *Le modèle d'interaction* qui il définit les protocoles de communication entre agents. Dans Gaia, un protocole définit une action demandant une interaction entre deux agents. Le modèle d'interaction est défini par une table comprenant un label, un expéditeur, un récepteur, une description, des entrées et des sorties et une description textuelle. La figure 13 représente un exemple de table d'interaction illustrant la réception d'un papier par un reviewer [10].

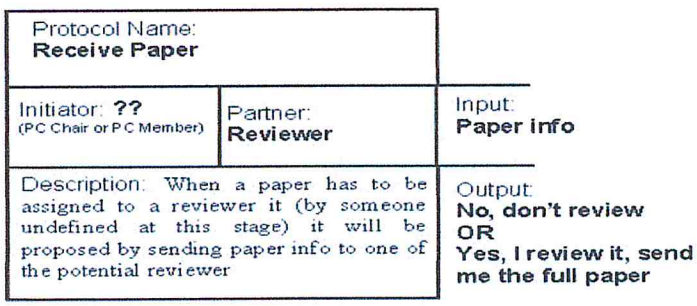


Figure 13: *Exemple de modèle d'interaction* [10]

- *Le modèle d'agent* permet de spécifier les types d'agents (un ensemble de rôles d'agent) et leurs instances. [16]
- *Le modèle de service* permet de spécifie les principaux services (blocs d'activités avec leurs conditions) liés aux rôles des agents. [16]
- *Le modèle d'organisation* ou d'accointances qui définit la structure de l'*organisation* grâce à des graphes orientés représentant les voies de communication entre agents. Différent types de relations existent : contrôle, pair (*peer*), dépendance, et d'autres pouvant être définis. [15]
- *Le modèle environnemental* qui décrit les différentes ressources accessibles caractérisées par les types d'actions que les agents peuvent entreprendre pour les modifier. [15]

Les notations.

Gaia ne fournit pas de notation graphique à proprement parler. En effet, le modèle de rôles et les protocoles ne sont décrits que par des tables (Fig. 12, Fig. 13). La notation introduite par Gaia est considérée comme semi-formelle. Elle a été appliquée avec une grande facilité et plusieurs exemples pédagogiques Nous remarquons dans cette notation que les aspects dynamiques ne sont pas réellement modélisés. Ces aspects sont uniquement introduits au niveau des propriétés de sûreté et de vivacité. Pour pallier ce problème, l'utilisation d'AUML a été suggérée. [10]

III.5 TROPOS

Tropos[2] est une méthodologie, pour la construction des systèmes orientés agent. Elle repose sur deux idées clés. Tout d'abord, la notion d'agent et toutes les notions de mentalité connexes (par exemple les objectifs et les plans) sont utilisées dans toutes les phases du développement de logiciels, de l'analyse précoce jusqu'à la mise en œuvre effective. Deuxièmement, Tropos couvre également les premières phases de l'analyse des besoins, permettant ainsi une compréhension plus approfondie de l'environnement où le logiciel doit fonctionner et du type d'interactions qui devraient se produire entre les logiciels et les agents humains, Tropos adopte une approche transformationnelle, dans le sens où, à chaque étape, les modèles vont

[2] Tropos est déviré du terme grec *tropé* signifiant "facilement modifiable ou adaptable".

être raffinés de manière itérative par ajout ou suppression d'éléments ou relations dans les modèles. [20]

Les concepts clés.

Les concepts manipulés par Tropos sont [20] :

- L'acteur qui représente un agent physique, social ou logiciel ainsi qu'un rôle ou un poste.
- le rôle qui une caractérisation abstraite du comportement d'un acteur dans un contexte particulier
- la position qui est un ensemble de rôles joués par un acteur ;
- le but qui est l'intérêt stratégique d'un acteur qui peut être soft (secondaire) ou hard (primordial ou de fonctionnement) ;
- le plan qui est une façon de satisfaire un but ;
- la ressource qui est une entité physique ou informationnelle dont a besoin un acteur pour fonctionner;
- la dépendance sociale entres acteurs où un acteur (depender) dépend d'un autre (dependee) pour atteindre un but, exécuter un plan ou obtenir une ressource.
- Capacité, qui représente la capacité d'un acteur à définir, choisir et exécuter un plan pour la réalisation d'un objectif, compte tenu de certaines conditions mondiales et en présence d'un événement spécifique.
- Croyance, qui représente la connaissance des acteurs du monde.

Tropos définit une modélisation selon cinq points de vue complémentaires [15] :

- *Social* : quels sont les différents acteurs et que font-ils ? Quelles sont leurs obligations et leurs capacités ?
- *Intentionnel* : quels sont les buts adéquats et comment sont-ils reliés ? Comment sont-ils remplis et par qui des dépendances sont-elles demandées ?
- *Communicationnel* : comment les acteurs dialoguent-ils et comment interagissent-ils ?
- *Orienté procédé (processus)* : quels sont les processus (business/computer) adéquats?
- *Orienté objet* : quels sont les classes et objets adéquats et quels sont leurs liens ?

Les notations.

Tropos utilise un langage de modélisation visuelle basé sur le cadre i*[3] qui concéder les acteurs (agents, rôles ou positions), les objectifs et les dépendances des acteurs comme concepts primitifs pour la modélisation d'une application lors de l'analyse des besoins anticipés [22].

Ainsi, un modèle Tropos peut être représenté comme un ensemble de diagrammes, comme par exemple des diagrammes d'acteurs, décrivant le réseau de relations de dépendance entre les acteurs, les diagrammes de but qui illustrent l'analyse des objectifs et des plans du point de vue d'un acteur spécifique. Au cours de la phase de conception détaillée, les diagrammes d'interaction UML AUML et les diagrammes d'activité sont utilisés pour décrire les processus que les agents peuvent utiliser pour remplir leurs objectifs et les messages qu'ils échangent lors de l'exécution de ces processus dans l'environnement distribué. Enfin, pour la mise en œuvre concrète du système, une plate-forme d'agent, telle que JADE ou Jack, peut être choisie. [23]

Ces différentes notations, sont utilisées pour produire les principaux diagrammes suivants:[20]

- *diagramme d'acteur* : Il représente les acteurs identifiés dans le domaine et les dépendances stratégiques parmi eux. Ce type de diagramme est employé pour définir les besoins des utilisateurs du système, pour montrer leurs intentions ainsi que les relations entre les acteurs. Ce diagramme représente un acteur par un nœud et les dépendances par les liens entre les nœuds,

- *diagramme de but* : il représente la perspective d'un acteur spécifique (ses buts, ses plans et ses ressources). Il est dessiné comme un ballon et contient des graphiques dont les nœuds sont des objectifs (ovales) et / ou des plans (forme hexagonale) et dont les arcs sont les différentes relations qui peuvent être identifiées parmi ses nœuds.

- *diagramme de compétence (capacité):* chaque compétence (ou ensemble corrélé de compétences) est modélisée par un diagramme d'états/transitions. Les événements externes définissent l'état initial d'un diagramme de capacité; les plans sont décrits par

[3] i * (prononcé "i stars") ou i * Framework est un langage de modélisation adapté à une phase précoce de la modélisation du système afin de comprendre le domaine du problème.

les nœuds du diagramme, les arcs des transitions modélisent les événements et les croyances,

- *diagramme de plan :* Chaque nœud de plan d'un diagramme de capacité peut être spécifié par des diagrammes d'activité UML.
- *diagramme d'interaction d'agent :* les diagrammes de séquence AUML peuvent être exploités. Dans les diagrammes de séquence AUML, les agents correspondent à des objets dont la ligne de vie est indépendante de l'interaction spécifique à modéliser, les actes de communication entre les agents correspondent à des arcs de messages asynchrones

III.6 PROMETHEUS

La méthodologie Prometheus, a été développée sur plusieurs années en collaboration avec le logiciel orienté vers l'agent. La méthodologie a été enseignée lors d'ateliers industriels et de cours universitaires. Il s'est avéré efficace pour aider les développeurs à concevoir, documenter et créer des systèmes d'agents. Prométhée diffère des méthodologies existantes en ce sens qu'il s'agit d'une méthodologie détaillée et complète (début-fin) pour le développement d'agents intelligents qui ont évolué à partir de l'expérience industrielle et pédagogique [24].

Les concepts clés.

Selon Gauthier Picard [15], On retrouve dans Prometheus les mêmes concepts que dans Gaia ou MaSE. Les agents sont des agents à gros grains, BDI et fortement *pro-actifs*. Ils peuvent faire partie de *groupes* (fixés), possèdent des accointances de travail, avec lesquels ils interagissent suivant des *protocoles* spécifiés, fournissent des *services* (ce sont leurs actions) et ont des capacités (des modules). Les agents peuvent envoyer des messages et planifier leurs actions en fonction de *buts*. Bien sûr, ils possèdent des données ou *croyances* sur les autres agents.

L'autonomie des agents est assurée par l'encapsulation des buts et des plans. Bien que la notion de groupe soit avancée, les notions de rôles sont inexistantes, et leur gestion non abordée. Par contre la sûreté est assurée lors de regroupements (en termes de droits d'accès). Une notion intéressante, et peu présente dans les autres méthodes, est la notion d'*incident*

pouvant survenir en cours de fonctionnement. Cette notion reste cependant floue, et semble correspondre à la notion d'exception. [15]

Les notations.

Prometheus définit des notations dédiées et ré-utilise UML et A-UML. Nous pouvons distinguer sept notations spécifiques [15] :

- Des *descripteurs textuels* permettent de spécifier des fonctionnalités du système, des scénarios, ou des agents à partir de champs textuels (comme les buts, les actions, les interactions possibles).

- *Les diagrammes de couplages de données* permettent de regrouper les fonctionnalités du système aux données produites en termes de production, de modification ou d'exploitation.

- Les *diagrammes d'accointances d'agent* décrivent les interconnexions entre agents de différents types. C'est un diagramme de classes UML simplifié où les agents sont représentés par des classes (sans compartiment) et les connexions sont représentées par des associations binaires avec multiplicité.

- Les *diagrammes de vue du système* relient les agents, les événements et les objets partagés dans un même modèle en utilisant une notation dédiée (Figure 14).

- Les *diagrammes d'interaction* montrent les interactions entre agents.

- *les diagrammes de séquence UML et A-UML* pour modéliser les protocoles.

- Les *diagrammes de vue des agents* sont similaires aux diagrammes de vue du système mais montrent les interactions entre capacités au sein d'un agent. Chaque message, perception ou action identifié au niveau du système doit être pris en compte à ce niveau (Figure 15).

- Les *diagrammes de vue des capacités* montrent les interactions et contraintes sur les actions et les événements entre les plans et les capacités.

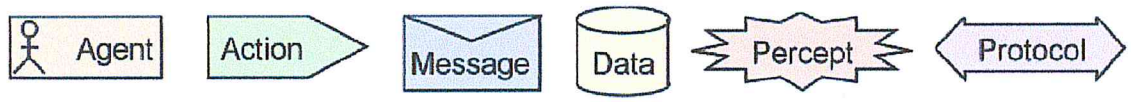


Figure 14: *Notations utiliser dans Les diagrammes de vue du système* [26]

Figure 15: *Notations utiliser dans Les iagrammes de vue des agents* [26]

IV. Analyse de la comparaison

Le tableau 2 montre une synthèse de la comparaison des méthodes présentées dans la section III.

IV.1 Synthèse de la comparaison des méthodes

Lire la table suivant l'axe des méthodes (de gauche à droite) nous indique, si une dimension méthodologique donnée est largement prise en compte par les méthodes orientées agents analysées. On peut constater que la coopération ou les rôles sont communément abordées. Par contre, une description détaillée des capacités est toujours absente.

	Méthodologies de développement de systèmes multi-agents					
Dimensions	Aalaadin	Cassiopée	MaSE	Gaia	Tropos	Prometheus
Coopération/Interaction	✓✓	×	✓✓	✓	✓	✓✓
Emergence	✓✓	✓✓	✓×	✓✓	✓	✓
Rôle	✓✓	✓✓	✓✓	✓✓	✓✓	×
Capacité	×	×	×	×	✓	✓

Capacité Maturité	Actuelle	×	×	×	×	×	×
	Héritage	×	×	×	×	×	×
	Développement	×	×	×	×	×	×
Type de capacité	Technique	×	×	×	×	×	×
	Ressources sociotechniques	×	×	×	×	×	×
	Manuel	×	×	×	×	×	×
	Ressources d'information	×	×	×	×	×	×
	Ressources du personnel	×	×	×	×	×	×

Tableau 2: *Synthèse de la comparaison des différentes méthodes*

Notation :

(✓✓) Pour les propriétés pleinement et explicitement prises en charge ;

(✓) pour les propriétés prises en charge de manière indirecte ;

(✓×) pour des propriétés potentiellement prises en charge ;

(×) pour des propriétés non prises en charge ;

IV.2 Conclusion de la comparaison

La comparaison des méthodologies fait ressortir toute la diversité des techniques d'ingénierie orientée agent. Ces méthodologies se limitent parfois à des aspects très spécifiques des systèmes multi-agents ou bien ne couvrent pas la totalité de nos exigences (comme le concept de capacité qui n'est pas bien détaillé, ou le comportement émergent qui est présent mais qui n'apparait pas bien dans les notations).

V. Conclusion

Dans ce chapitre, nous nous somme intéressé aux concepts couverts par les méthodes de développement de systèmes les plus répondus. Nous nous sommes intéressé sur les notions des rôles et des capacités afin d'exprimer nos besoins compte-tenu de nos objectifs (proposer un méta modèle pour la spécification des capacités des sous système à travers la notion de rôle dans un SOS).

Nous avons commencé par définir les dimensions de comparaison, nous avons par la suite présenté les différentes méthodes pour arriver à la fin vers deux tableaux synthétisant les différentes méthodes.

Avant de proposer notre propre méta modèle, nous allons dans le chapitre suivant donner une vue globale sur la méta-modélisation.

Chapitre III :
La méta-modélisation

I. Introduction

Comme l'objectif de notre étude est de proposer un méta-modèle pour la spécification des capacités des sous système à travers la notion de rôle dans un SOS, il semble logique de connaitre la notion de méta-modélisation ainsi que les différents concepts liés.

Ce chapitre commence par donner une introduction générale du méta modélisation, définit ensuite les méta-modèles et en donne un exemple, enfin il présente les différents concepts pour la création d'un méta-modèle selon la norme MOF1.4

II. La méta modélisation

Avant d'étudier ce qu'est un méta-modèle, il semble logique de s'intéresser d'abord à ce qu'est un modèle. Minsky propose la définition suivante : un objet (A*) est un modèle d'un objet A pour un observateur O dans la mesure où O peut utiliser (A*) pour répondre aux questions qu'il se pose sur A. dans ce cas, O utilisera le modèle (A*) comme support de raisonnement. L'action de modélisation permet donc de passer de l'objet du modèle (A dans la définition de Minsky) vers le modèle (A*), à travers un principe d'abstraction visant à simplifier la représentation de l'objet à modéliser. Il est possible de poursuivre ce travail d'abstraction pour créer des méta-modèles. La création d'un métamodèle revient à construire une représentation d'un modèle, toujours dans le but de fournir un support de raisonnement. L'action de méta-modélisation permet donc de passer d'un modèle (A* dans la définition de Minsky) vers le méta-modèle (A**) [32].

Comme le résume la Figure16, l'objet (A**) ne peut pas être obtenu directement par modélisation de l'objet (A), mais bien par la modélisation de l'objet (A*). Cette relation d'ordre dans l'activité de modélisation fait apparaitre des niveaux de modélisations distincts. [32]

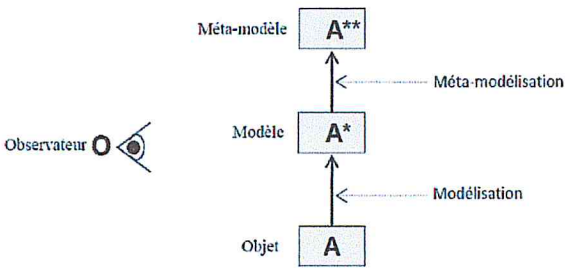


Figure 16: *Activité de modélisation et de méta-modélisation dans les termes de Minsky* [32]

La principale notation pour la méta-modélisation par objet est le MOF[4] (Meta Object Facility). L'OMG[5] (Object Management Group) a défini deux variantes de MOF : EMOF (MOF essentiel) et CMOF (MOF complet). Le but du MOF est de définir un langage unique et standard pour décrire des méta-modèles. Il est constitué d'un ensemble relativement restreint (bien que non minimal) de concepts « objets » permettant de modéliser ce type d'information. Par exemple, UML est l'un des méta-modèles décrit en utilisant le MOF [32]. Le MOF définit 4 niveaux de modélisation : [33]

- M0 : système réel, système modélisé
- M1 : modèle du système réel défini dans un certain langage
- M2 : méta-modèle définissant ce langage
- M3 : méta-méta-modèle définissant le méta-modèle
  - Le niveau M3 est le MOF
  - Dernier niveau, il est méta-circulaire : il peut se définir lui-même

Dans la Méta-modélisation UML, on retrouve également les 4 niveaux Mais avec le niveau M3 définissable en UML directement à la place du MOF.

Les niveaux de modélisation	Normes OMG de modélisation
M3 – Niveau méta-méta-modèle	MOF
M2 – Niveau méta-modèle	UML, CWM, SPEM, etc
M1 – Niveau modèle	UML
M0 – Niveau instance	Informations réelles (Objets)

***Tableau 3:** Hiérarchie selon une méta-modélisation par objets [32]*

[4] MOF est un formalisme de modélisation

[5] L'OMG (Object Management Group) est un consortium à but non lucratif d'industriels et de chercheurs, Dont l'objectif est d'établir des standards permettant de résoudre les problèmes d'interopérabilité des Systèmes d'information

III. Les métas modèles

III.1 La définition des métas modèles

Selon MOF [35] (Meta Object Facility) : La couche de métamodèle comprend les descriptions (c'est-à-dire les méta-métadonnées) qui définissent la structure et la sémantique des métadonnées. Les méta-métadonnées sont regroupées de manière informelle sous forme de métamodèle. Un métamodèle est un «langage abstrait» pour décrire différents types de données; C'est-à-dire une langue sans syntaxe ou notation concrète.

Autrement dit un métamodèle définit la structure que doit avoir tout modèle conforme à ce métamodèle Par exemple, le métamodèle UML définit que les modèles UML contiennent des packages, leurs packages des classes, leurs classes des attributs et des opérations, etc. [34]

La figure 17 illustre la relation entre un métamodèle et l'ensemble des modèles qu'il structure. Les métamodèles fournissent la définition des entités d'un modèle, ainsi que les propriétés de leurs connexions et de leurs règles de cohérence. MOF les représente sous forme de *diagrammes de classes.* [34]

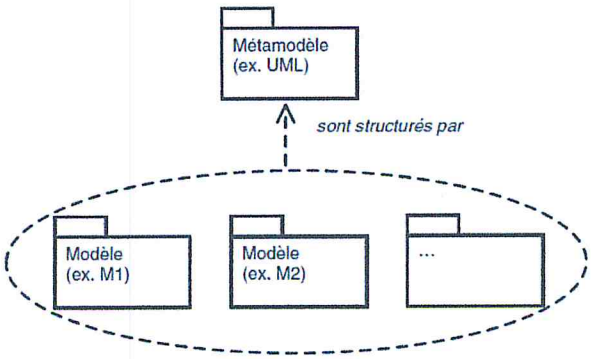


***Figure 17:** Relation entre modèles et métamodèles* [34]

Rappelons que les diagrammes de classes permettent de représenter les notions d'un domaine et leurs propriétés, que ces notions ou entités soient organisées ou non sous forme d'objets. Cette utilisation des diagrammes de classes a le double avantage de permettre de définir très précisément les métamodèles et de les rendre eux aussi pérennes et productifs. [34]

III.2 Exemple De Meta modelé

Avant de définir précisément et théoriquement ce que sont les métamodèles, il nous paraît important d'en présenter un exemple. Nous montrerons ainsi à quoi servent les métamodèles et comment ils sont élaborés conceptuellement.

Métamodèle de diagramme de cas d'utilisation

L'exemple de métamodèle que nous présentons est celui de Mr **Xavier Blanc** [34].
Selon l'auteur son propos n'est pas d'expliquer comment faire des diagrammes de cas d'utilisation mais de présenter le métamodèle des diagrammes de cas d'utilisation. Il s'intéresse uniquement à la structure de ces diagrammes.

La définition suivante des diagrammes de cas d'utilisation n'est pas standard, mais **l'auteur Xavier Blanc** la proposé afin de faciliter la présentation :

> « Un diagramme de cas d'utilisation contient des **acteurs,** un **système** et des **cas d'utilisation.** Un acteur a un nom et est *relié* aux cas d'utilisation. Un acteur peut *hériter* d'un autre acteur. Un cas d'utilisation a un intitulé et peut *étendre* ou *inclure* un autre cas d'utilisation. Le système a lui aussi un nom, et il inclut tous les cas d'utilisation. »

Les informations représentant les concepts fondamentaux des diagrammes de cas d'utilisation sont en gras et les relations existantes entre ces concepts en italique.

La figure 18 illustre ce métamodèle. Il contient trois classes, acteur, système et cas d'utilisation. La classe acteur possède un attribut nommé nom, dont le type est une chaîne de caractères. La classe cas d'utilisation possède un attribut nommé intitulé, dont le type est une chaîne de caractères. La classe système possède un attribut nommé nom, dont le type est une chaîne de caractères. Ce métamodèle contient une association nommée hérite, qui à la classe acteur comme source et comme cible. Il contient aussi deux associations, étend et inclut, qui ont pour source et pour cible la classe cas d'utilisation. Il contient enfin une relation d'agrégation entre la classe système et la classe cas d'utilisation.

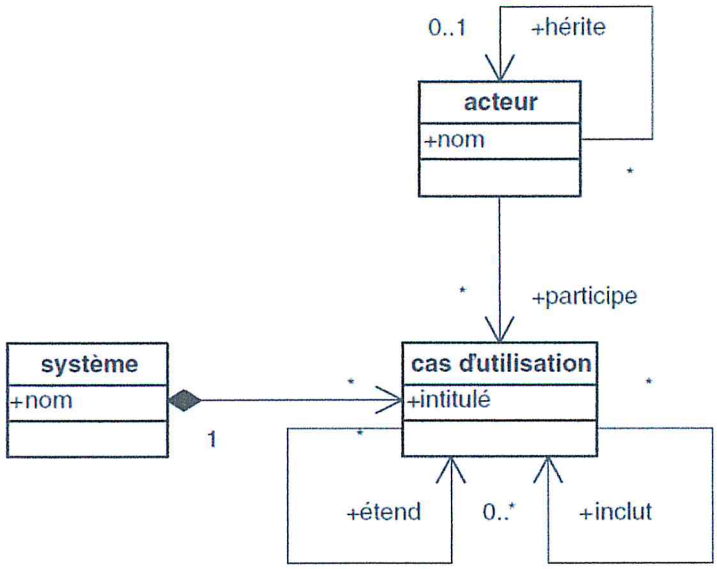


Figure 18: *Métamodèle des diagrammes de cas d'utilisation* [34]

Ce métamodèle définit que les modèles en conformité avec lui ne peuvent contenir que des acteurs, des systèmes et des cas d'utilisation. Un acteur a un nom, un cas d'utilisation a un intitulé et un système a un nom. Ce métamodèle définit aussi que les modèles conformes peuvent faire apparaître des relations d'héritage entre les acteurs et qu'ils peuvent en outre faire apparaître des relations d'extension et d'inclusion entre cas d'utilisation. Les cas d'utilisation de ces modèles doivent en outre être contenus dans un système.

III.3 Création de métamodèle selon MOF1.4

Après avoir vu qu'un métamodèle était une sorte de diagramme de classes et en avoir présenté un exemple de métamodèle d'un diagramme de cas d'utilisation, nous allons expliquer précisément ce qu'est un métamodèle.

Étant donné qu'il existe plusieurs façons de faire des métamodèles (MOF1.3, MOF1.4, MOF2.0, EMF, etc.) et que nous ne pouvons toutes les présenter, nous avons retenu la façon MOF version 1.4. Tous les métamodèles publics de l'OMG sont réalisés avec cette version, assez simple d'utilisation [34].

Afin de discerner les classes constituantes d'un métamodèle des autres classes, telles que les classes Java, MOF1.4 propose d'utiliser le terme *métaclasse*. Un métamodèle est ainsi constitué d'un ensemble de métaclasses. De même, afin de discerner les objets instances des métaclasses des autres objets, MOF1.4 propose d'utiliser le terme *méta-objet*. Ainsi, un modèle est constitué d'un ensemble de méta-objets instances de métaclasses [34].

Les concepts de base de MOF1.4 sont les suivants :

III.3.1 Classes (Meta classe)

Les classes sont des descriptions de type de métaobjets MOF "instance de première instance". Les classes définies au niveau M2 ont logiquement des instances au niveau M1. Ces instances ont l'identité, l'état et le comportement de l'objet. L'état et le comportement des instances de niveau M1 sont définis par la classe de niveau M2 dans le contexte de l'information commune et des modèles de calcul définis par la spécification MOF.

Les instances de classe appartiennent à des classes qui ont un impact sur certains aspects de leur comportement. Il est possible d'énumérer toutes les instances d'une classe dans une étendue de classe.

Les classes peuvent comporter trois types de fonctionnalités. Attributs et opérations décrits ci-dessous et Références décrites dans "Références"(voir la section III.3.4). Les classes peuvent également contenir des exceptions, des constantes, des types de données, des contraintes et d'autres éléments.

- **Les attributs (Meta-attribut)**

Un attribut définit un emplacement notionnel ou un support de valeur, généralement dans chaque instance de sa classe.

Propriété	**Description**
Nom	Unique dans la portée de la catégorie Attributs.
Type	Peut-être une Classe ou un DataType.
"isChangeable" flag	Peut être ou non modifiable. S'il est non modifiable, cela signifie en quelque sorte que le méta-attribut a une valeur constante.

"isDerived" flag	Peut être considéré comme dérivé. Si le méta-attribut est dérivé, cela signifie que sa valeur peut être calculée, par exemple, grâce aux valeurs d'autres méta-attributs de la métaclasse.

Tableau 4: les propriétés de l'attribut [35]

Les propriétés d'agrégation d'un attribut dépendent du type de l'attribut; Voir III.3.3, «Agrégation»

- Opérations (Meta opération)

Les opérations sont des «crochets» pour accéder aux comportements associés à une classe. Les opérations ne spécifient pas le comportement ou les méthodes qui implémentent ce comportement. Au lieu de cela, ils spécifient simplement les noms et les signatures de type par lesquels le comportement est invoqué. Les opérations ont les propriétés suivantes.

Propriété	**Description**
Nom	Unique dans le champ de la classe.
Liste des paramètres de position ayant les propriétés suivantes:	
Nom du paramètre:	
Le type de paramètre	peut être désigné par une classe ou un DataType
Paramètre direction "in", "out", ou "In out"	Détermine si les arguments réels sont transmis du client au serveur, du serveur au client, ou les deux.
Un type de retour facultatif.	
Une liste des exceptions qui peuvent être soulevées par une invocation.	

Tableau 5: les propriétés de l'opération [35]

- Généralisation de classe

Le MOF permet aux classes d'hériter d'une ou plusieurs autres classes. En suivant le terme UML, le modèle MOF utilise le verbe «généraliser» pour décrire la relation d'héritage (c.-à-d., Une super-classe généralise une sous-classe).

La généralisation de la classe MOF est similaire à la généralisation dans UML. La sous-classe hérite de tous les contenus de ses super-classes (c.-à-d., Tous les méta-attributs et toutes les méta-opérations et les références de super-classes, et tous les types de données, exceptions et

constantes existants). Toute contrainte explicite qui s'applique à une super-classe et tout comportement implicite pour la super-classe s'appliquent également à la sous-classe.

Si une sous-classe hérite de plusieurs super-classes, il est impératif que ces sous-classes n'aient pas de attributs ni de opérations de mêmes noms.

- Classes abstraites

Une classe (metaclasse) peut être définie comme «abstraite». Une classe (metaclasse) abstraite est utilisée uniquement pour l'héritage. Et aucun méta-objet ne peut être une instance directe de cette classe (metaclasse).

- Classes ' Leaf' et 'Root'

Une classe (metaclasse) peut être définie comme feuille (leaf) ou racine (root) d'un arbre d'héritage. Si la classe (metaclasse) est feuille, cela signifie qu'aucune autre classe (metaclasse) ne peut en hériter. Si la métaclasse est racine, elle ne peut hériter d'une autre classe (metaclasse).

III.3.2 Associations (Meta- Associations)

Les associations (méta-associations) sont la construction principale du modèle MOF pour exprimer les relations dans un méta-modèle. Une association (méta-association) permet de définir une relation entre deux classes (métaclasses). En fait, une association (méta-association) permet de définir la structure des liens entre les méta-objets instances des classes (métaclasses) reliées par association (méta-association). [35]

- L'association Ends

Chaque association MOF contient exactement deux extrémités de l'association décrivant les deux extrémités des liens. L'Association Ends définit les propriétés suivantes.

Propriété	**Description**
Un nom de cette **'END'**	Ceci est unique au sein de l'Association.
Un type de cette **'END'**	Cela doit être une classe.
Spécification de la multiplicité	Voir « Multiplicités de l'association **'END'** »
Une spécification d'agrégation	Voir «Agrégation de l'association»
Un paramètre de "navigability"	Contrôle si les références peuvent être définies pour cette **END** (voir III.3.4, "Références", "Références")

Un paramètre de "changeability"	Détermine si cette **END** d'un lien peut être mise à jour "en place".

Tableau 6: les propriétés de l'association [35]

- **Multiplicités de l'association END**

Chaque extrémité de l'association a une spécification de multiplicité. Bien qu'ils soient conceptuellement similaires aux multiplicités d'attributs et d'opérations, il existe une différence importante:

- Cette multiplicité permet de spécifier le nombre d'instances de la classe (métaclasse) identifiée par le type de l'extrémité pouvant être liées à exactement une instance de la classe (métaclasse) de l'autre extrémité de l'association.

III.3.3 Agrégation

Dans un métamodèle MOF, Classes et DataTypes peuvent être liés à d'autres classes en utilisant des associations ou des attributs. Dans les deux cas, les aspects du comportement des relations peuvent être décrits comme une sémantique d'agrégation.

- **Sémantique d'agrégation**

Le MOF prend en charge deux types d'agrégation pour les relations entre les instances (c'est-à-dire «composite» et «non agrégé»). Une troisième séparation d'agrégation - "partagée" - n'est pas prise en charge dans cette version de la spécification MOF.

Une relation non agrégée est une liaison (conceptuellement) relié entre les instances ayant les propriétés suivantes:

• Il n'y a aucune restriction spéciale sur la multiplicité des relations.

• Il n'y a aucune restriction spéciale sur l'origine des instances dans les relations.

• Les relations n'ont pas d'incidence sur la sémantique du cycle de vie des instances liées. En particulier, la suppression d'une instance ne provoque pas la suppression d'instances connexes.

_ En revanche, une relation composite est une liaison (conceptuellement) plus forte entre les instances ayant les propriétés suivantes:

• Une relation composite est asymétrique, avec une extrémité désignant le "composite" ou "entier" dans la relation et l'autre désignant les "composants" ou "pièces".

• Une instance ne peut pas être un composant de plus d'un composite à la fois, sous une relation composite.

• Une instance ne peut pas être un composant de lui-même, ses composants, ses composants et ainsi de suite sous une relation composite.

• Lorsqu'une instance «composite» est supprimée, tous ses composants sous une relation composite sont également supprimés et tous les composants des composants sont supprimés et ainsi de suite.

• La règle de fermeture de composition: une instance ne peut pas être un composant d'une instance d'une étendue de package différente.

- Agrégation de l'association

La sémantique d'agrégation d'une association est spécifiée explicitement à l'aide de l'attribut "agrégation" de l'associationEnds. Dans le cas d'une association "composite", l'attribut "agrégation" de l'associationEnd "composite" est défini sur true et l'attribut "aggregation" du "composant" AssociationEnd est défini sur false. En outre, la multiplicité de l'associationEnd "composite" doit être "[0..1]" ou "[1..1]" conformément à la règle selon laquelle une instance ne peut pas être un composant de plusieurs composites.

- Agrégation d'attributs

La sémantique d'agrégation efficace pour un attribut dépend du type de l'attribut. Par exemple:

• Un attribut dont le type est exprimé en Data Type a une sémantique "non agrégée".

• Un attribut dont le type est exprimé en classe a une sémantique "composite".

Il est possible d'utiliser un Data Type pour coder le type d'une Classe. Cela permet au métamodèle de définir un attribut dont la valeur ou les valeurs sont des instances d'une classe sans encourir les frais généraux de la sémantique "composite".

III.3.4 Références

Afin de faciliter les navigations entre méta-objets via les méta-associations dont les extrémités sont navigables, MOF1.4 propose le concept de référence (reference). Une référence est une sorte de méta-attribut permettant essentiellement de naviguer via les méta-associations.

Comme un méta-attribut, une référence à un nom, un type, une multiplicité, etc. La différence avec les méta-attributs réside dans le fait que le type de la référence ne peut être qu'une métaclasse reliée par une méta-association avec la métaclasse qui contient la référence. De plus, l'extrémité qui pointe vers l'autre métaclasse (celle qui ne contient pas la référence) doit être navigable. La multiplicité de la référence doit être la même que la multiplicité de cette extrémité. La référence est en fait une sorte d'alias pour atteindre les méta-objets liés. [34]

III.3.5 Data Types

Les définitions de métamodèle doivent souvent utiliser des valeurs de paramètres d'attribut et de fonctionnement qui ont des types dont les valeurs n'ont pas d'identité d'objet. Le MOF fournit le concept Data Type pour combler ce besoin. Data Types peut représenter en deux types [35] :

1. Les types de données primitives comme Boolean, Integer et String sont les éléments de base pour l'expression de l'état.

2. Les constructeurs de type de données permettent au métamodèle de définir des types de données plus complexes. Les constructeurs de type de données standard du MOF sont des types d'énumération, des types de structure, des types de collecte et des types d'alias.

III.3.6 Package

MOF1.4 propose le concept de *package,* qui permet de grouper différents éléments d'un même métamodèle. L'objectif est, d'une part, de regrouper les éléments d'un métamodèle portant sur un même domaine, par exemple, les éléments relatifs aux diagrammes de classes, et, d'autre part, de gérer plus facilement les méta-objets instances d'un ensemble de métaclasses. Il est ainsi possible de stocker dans un même espace mémoire des méta-objets dépendant les uns des autres. [34]

IV. Conclusion

Nous avons commencé ce chapitre par une introduction de la méta-modélisation avec ces différents niveaux. Nous avons détaillé les différents concepts qui nous ont permis de créer notre Métamodèle selon la norme MOF1.4.

Dans le chapitre qui suit, nous allons nous baser sur une démarche de deux phases permettant de proposer d'abord un Métamodèle servant à modéliser les capacités, et permettant ensuite de proposer une notation graphique pour le représenter.

Partie II: Contribution

CHAPITRE IV : METAMODELE POUR LA SPECIFICATION DES ROLES ET DES CAPACITES 42

CHAPITRE V: IMPLEMENTATION 53

Chapitre IV : Métamodèle pour la spécification des rôles et des capacités

I. Introduction

Pour spécifier la mission d'un système de système, une bonne connaissance des capacités des sous-systèmes est indispensable. Une capacité peut être indépendante, ou peut dépendre des capacités d'autres systèmes.

Lors de la spécification des capacités avec les méthodes existantes, nous avons soulevé deux inconvénients. D'une part, si on utilise la notation proposé dans le domaine des SoS, il n'est pas intéressant de spécifier avec les sous-systèmes vu que la connaissance des systèmes constituants dans la phase d'analyse et de conception n'est pas toujours complète. D'autre part, les méthodes de spécification des systèmes complexes qui utilisent les concepts de rôle et de capacité ne permettent pas d'exprimer clairement le comportement émergent, ni les détails sur les capacités. C'est pourquoi nous allons proposer un méta modèle permettant la modélisation des capacités des sous système à travers la notion de rôle, nous allons développer une notation graphique pour le représenter.

II. Description de la démarche utilisée

Pour la proposition d'une notation graphique permettant la modélisation des rôles, nous avons suivi la démarche des deux phases, proposée par Moody [36]. Cette démarche comporte deux phases. Dans la première phase, une analyse ontologique est nécessaire pour spécifier les concepts nécessaires pour la notation que l'on veut proposer, afin de définir les structures sémantiques correspondantes (voir la section III). La seconde phase consiste à introduire des éléments graphiques correspondant aux constructions sémantiques développées dans la phase précédente. La figure 19 montre le processus global requis pour la proposition d'une notation graphique (voir la section IV).

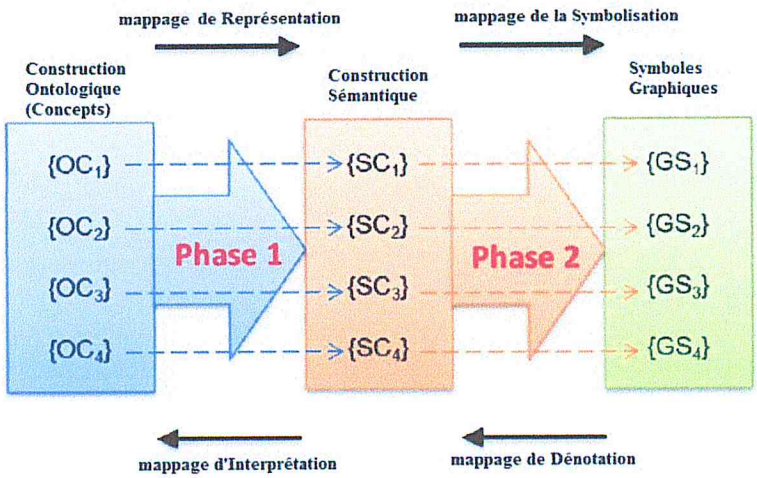


Figure 19 : le processus global de la démarche suivi [37]

III. Phase 1 : Définition des constructions sémantiques

III.1 Analyse ontologique

L'analyse ontologique permet de définir les constructions sémantiques (semantic constructs) nécessaires pour le langage. Elle implique un mappage bidirectionnel entre la notation graphique et le concept ontologique. Le mappage d'interprétation décrit celui de la notation à l'ontologie, tandis que le mappage de représentation décrit le mappage inverse [36]. Selon la théorie, il devrait y avoir une correspondance individuelle entre les concepts de l'ontologie et les constructions dans la notation. Sinon, une ou plusieurs des anomalies suivantes se produiront (figure 20) [37] :

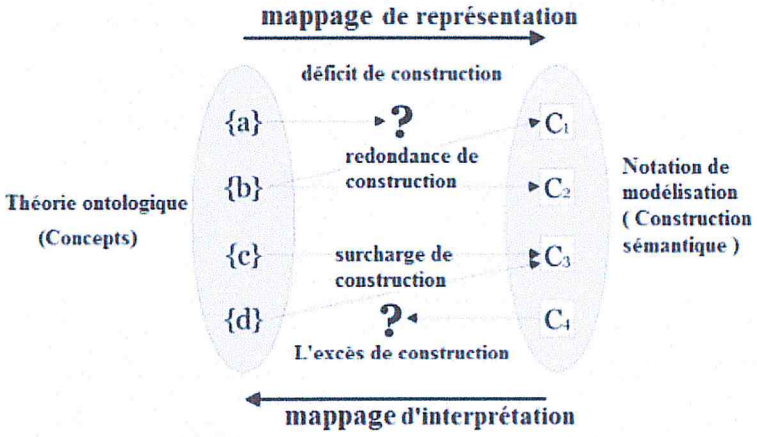


***Figure 20:** les anomalies d'analyse ontologique [36]*

- Le déficit de construction existe lorsqu'il n'y a pas de construction dans la notation correspondant à un concept ontologique particulier.
- La surcharge de construction existe lorsqu'une construction de notation unique peut représenter plusieurs concepts ontologiques.
- La redondance de construction existe lorsque des constructions de notation multiples peuvent être utilisées pour représenter un seul concept ontologique.
- L'excès de construction existe lorsqu'une construction de notation ne correspond à aucun concept ontologique.

Si le déficit de construction existe, la notation est dite ontologiquement incomplète ; Si l'une des trois autres anomalies existe, il est ontologiquement peu clair.

III.2 Définition des constructions ontologiques

Dans cette phase, nous avons proposé une ontologie basée sur deux concepts principaux afin de définir les constructions sémantiques nécessaires pour développer notre notation. Ces concepts ont été déduits au chapitre II, lorsque nous avons défini les dimensions de comparaison entre les méthodes.

C'est l'analyse du domaine des SoS, ainsi que les objectifs du sujet qui nous ont permis de déduire les concepts ontologiques suivants :

- **Le rôle, la coopération et le comportement émergent:** le rôle est une représentation abstraite nécessaire pour rendre la description de la mission indépendante du contexte de l'exécution du système. Comme les SoS se basent sur la notion de coopération, nous distinguons le rôle individuel du rôle collaboratif. Nous utilisons le rôle individuel lorsque nous n'avons besoin que d'un seul rôle pour accomplir une sous mission. Le rôle collaboratif quant à lui, est utilisé lorsqu'une collaboration entre les rôles est nécessaire pour accomplir une sous mission.

- **Les capacités et les dépendances :** les capacités sont les ensemble de fonctionnalités qu'un rôle est capable de fournir au SoS. Elles peuvent être classées par type et par maturité (voir section III chapitre1), bien que l'indication de la maturité dans une capacité ne signifie pas nécessairement qu'elle est effectivement utilisée ou non dans un SOS donné à un moment donné. Les dépendances entre les capacités sont des liens qui expriment qu'une capacité A dépend de la capacité B. Comme exemple, la capacité d'une ambulance à transporter un malade dépend de la capacité du conducteur à conduire l'ambulance.

En se basant sur ces concepts ontologiques, nous proposant le méta-modèle décrit dans la figure 21, qui constitue la base sémantique de notre notation graphique.

III.3 Définition des constructions sémantique

Le méta-modèle proposé constitue la base sémantique de notre notation. Il est basé sur les concepts ontologiques décrits précédemment.

Le concept de rôle est représenté par la méta-classe rôle, elle généralise également les deux méta-classes rôle individuel et rôle collaboratif. Cette dernière se compose de deux ou plusieurs rôles (c'est à dire peux contenir les rôles individuels ou collaboratifs ou les deux à la

fois). Cette relation de composition entre la méta-classe rôle collaboratif et la méta-classe rôle permet de représenter le concept ontologique coopération.

Le concept de capacité est représenté par la méta-classe capacité. Elle est liée à la méta-classe rôle pour représenter le fait qu'un rôle comporte au moins une capacité.
Les rôles qui sont composés d'autres rôles vont avoir de nouvelles capacités, ceci représente le comportement émergent, ce sont des capacités n'appartenant pas aux rôles composants le rôle collaboratif.

La dépendance entre les capacité est exprimée par la relation réflexive « dépend » qui a comme source et destination en même temps la méta-classe Capacité.
La classification de la capacité selon la maturité et selon le type est représentée par les deux attributs maturité et type dans la méta-classe Capacité. Comme les valeurs des attributs sont prédéfinies, nous avons créé deux énumérations afin de définir les valeurs possibles. La figure 21 illustre notre méta-modèle qui constitue la syntaxe abstraite de notre notation.

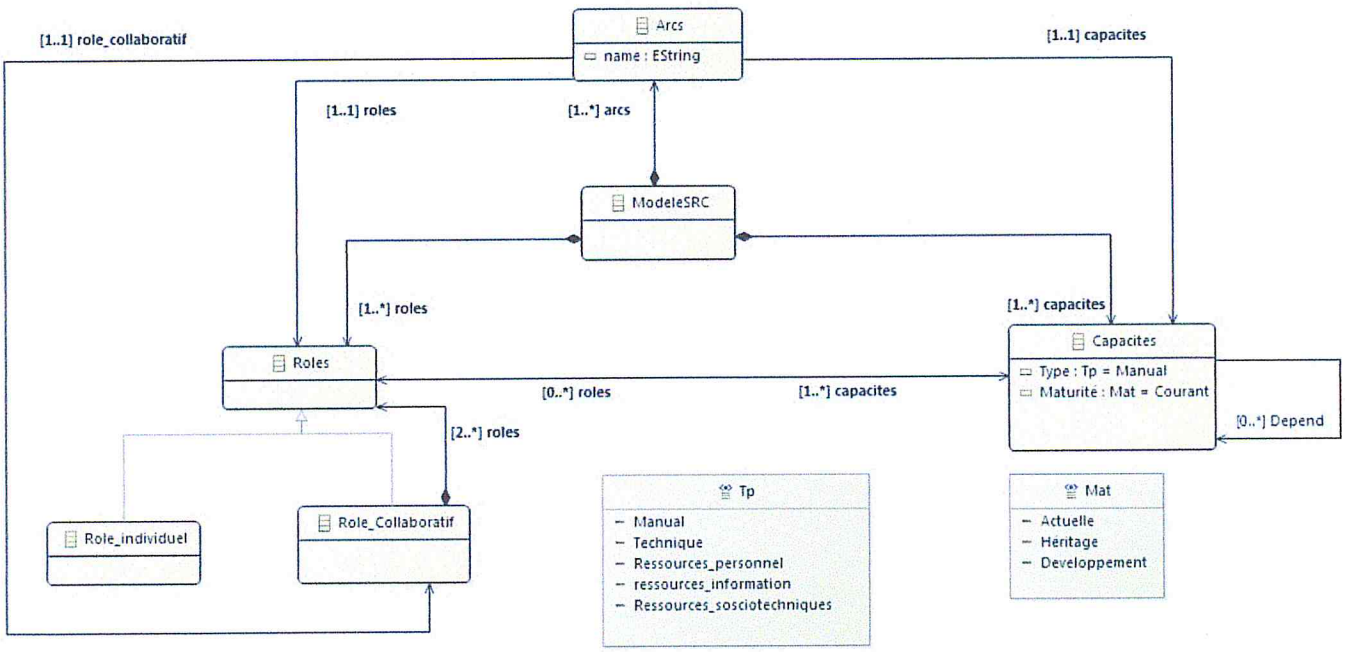


***Figure 21:** le méta modèle proposé*

IV. Phase 2 : Développement des constructions graphiques

Dans cette phase, nous introduisons les constructions graphiques nécessaires pour la notation graphique. Chaque concept dans le méta modèle doit correspondre à un élément graphique. Beaucoup de chercheurs pensent que la proposition d'une notation graphique est une question de goût et d'esthétique. Cependant, des études empiriques ont montré que la forme visuelle des notations affecte de manière significative la compréhension surtout par les utilisateurs novices [37].

Comme on envisage que l'utilisateur finale de notre notation est l'expert du domaine d'application, car lui seul connaît bien le domaine. Nous voulons proposer une notation simple et intuitive. Dans [36], il existe un ensemble de principes pour concevoir des notations visuelles cognitives (voir la figure 22). Ces principes ont été synthétisés à partir de preuves empiriques.

Nous avons suivi six principes pour notre notation graphique. Nous n'avons pas pu suivre les neuf principes parce qu'on a été un peu limité par les outils EMF et GMF d'éclipse. Ce n'était pas possible pour nous de satisfaire certains principes.

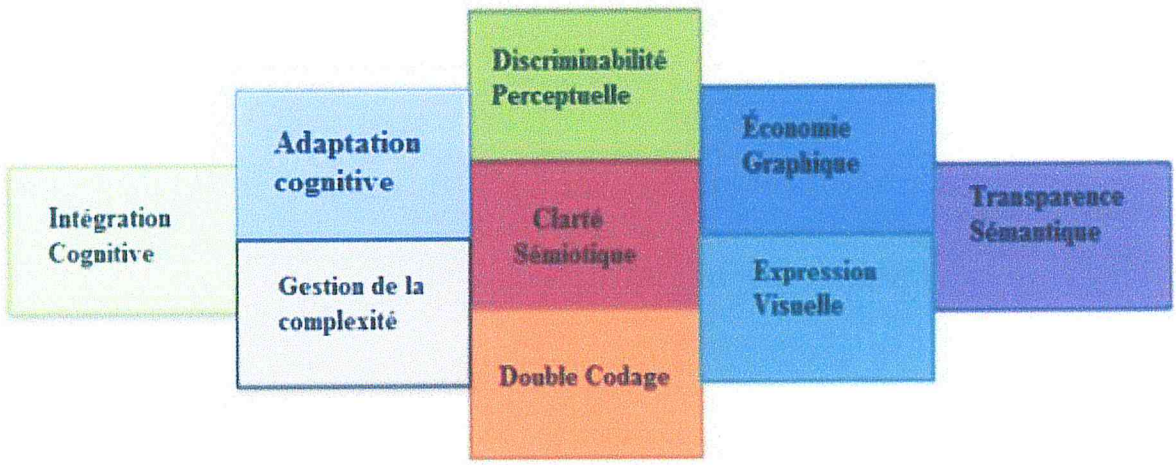


Figure 22: *Les principes de la conception des notations visuelles [36]*

_ Le Tableau 7 présente les symboles graphiques proposés pour chacune des constructions sémantiques développées

Constructions sémantiques	symboles graphiques	
Rôles	Rôle individuel	
	Rôle collaboratif	
Capacités		
Arcs	Rôles → Capacités	
	Rôles → Rôles	
	capacité → capacité (dépendance)	

***Tableau 7:** la notation graphique proposée*

IV.1 Principe de la clarté sémiotique

Selon la théorie des symboles de Goodman « Pour une notation, afin de satisfaire les exigences, il doit y avoir une correspondance individuelle entre les symboles et leurs concepts référents » [36]. Lorsqu'il n'y a pas de correspondance unilatérale entre les constructions et les symboles, une ou plusieurs des anomalies suivantes peuvent se produire (en utilisant des termes similaires à ceux utilisés dans l'analyse ontologique) (figure 23) [36] :

- Le déficit de symboles : qui se produit lorsqu'il existe des constructions sémantiques qui ne sont pas représentées par un symbole graphique.
- La redondance de symboles : qui se produit lorsque plusieurs symboles graphiques peuvent être utilisés pour représenter la même construction sémantique.

- La surcharge de symboles : qui se produit lorsque deux constructions différentes peuvent être représentées par le même symbole graphique.
- L'excès de symboles : qui se produit lorsque les symboles graphiques ne correspondent à aucune construction sémantique.

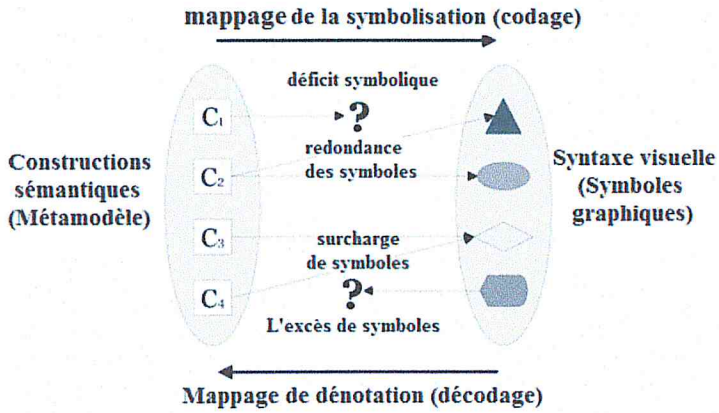


***Figure 23:** les anomalies de la clarté sémiotique [36]*

La clarté sémiotique représente une extension de l'analyse ontologique au niveau de la syntaxe visuelle. Dans notre travail, il existe deux symboles pour modéliser les deux types de rôles, un symbole pour les capacités, et pour chaque type d'arc il existe un symbole le représentant. De ce fait, le principe de la clarté sémiotique est respecté.

IV.2 Principe de la discrimination perceptive

Le principe de la discrimination perceptive concerne la facilité et la précision avec lesquelles différents symboles d'une notation peuvent être différenciés les uns des autres. [37] Un diagramme ne peut être interprété avec précision que si ses symboles peuvent être discernés avec précision. Une conception de notation devrait viser à augmenter la distance visuelle entre ses symboles afin de maximiser la discrimination [37]. Cette dernière est mesurée par le nombre et la portée des variables visuelles utilisées. [36] La figure 24 présente les huit variables visuelles élémentaires qui peuvent être utilisées pour décoder l'information.

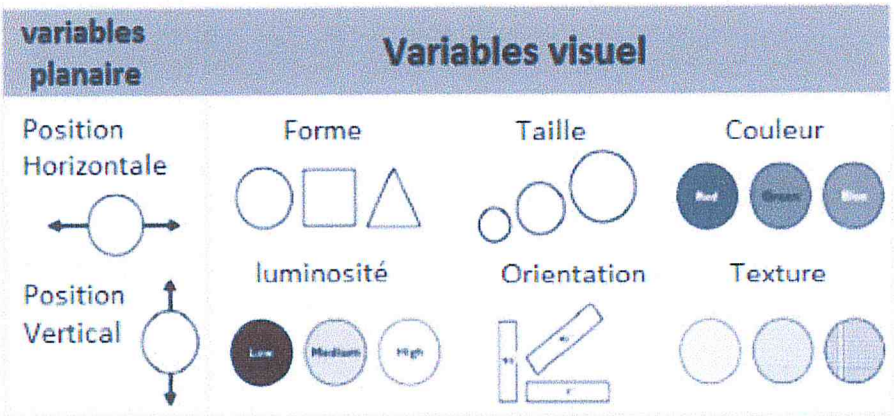


***Figure 24:** les variables visuelles. [36]*

La variable de forme est la plus influente de toutes les variables visuelles. La forme est le facteur prédominant utilisé par les humains pour classer les objets dans le monde réel. [36] Cela signifie que différentes formes sont perçues comme présentant des constructions sémantiques distinctes, mais les différences dans d'autres variables visuelles sont perçues comme présentant des constructions sémantiques différentes mais semblables [37]. Par conséquent, des formes similaires devraient être utilisées pour représenter les mêmes constructions ou similaires.

Dans notre travail, la phase précédente a révélé un ensemble de six constructions sémantiques qui peuvent être classées selon les concepts principaux en trois catégories: les rôles, les capacités et les arcs les reliant. Nous avons choisi les rectangles pour les rôles, les cercles pour les capacités et les flèches pour les arcs (voir le Tableau 7).

IV.3 Principe de l'expressivité visuelle

Le principe de l'expressivité visuelle concerne l'utilisation absolue de l'espace de conception graphique. Une notation est considérée comme expressive visuellement si elle utilise un grand nombre de variables visuelles et utilise une large gamme de valeurs de chaque variable [37]. Dans notre travail, nous avons utilisé comme variables la couleur et de luminosité. Une discussion sur l'utilisation de chaque variable est fournie ci-dessous:

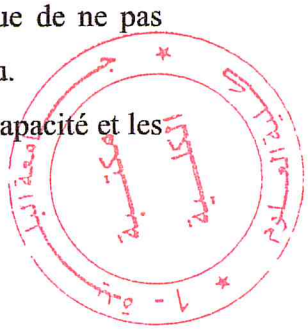
La taille: La taille est une variable visuelle ordinaire qui peut être utilisée pour coder des informations différentes. Nous l'avons utilisé pour différencier les trois types d'arcs.

La couleur : La couleur est considérée comme l'une des variables visuelles les plus cognitives. Des études ont montré que le système visuel humain est très sensible aux variations de couleur [36]. Les humains peuvent distinguer avec précision et rapidité les différentes couleurs. Cependant, si la couleur est mal utilisée, elle peut avoir un effet trompeur. En ce qui concerne notre notation nous avons utilisé les deux couleurs, noir pour l'arrière-plan de l'icône de rôle et le jaune pour celui de la capacité afin de garantir une grande discriminabilité.

La luminosité : Le terme «luminosité» est généralement interprété comme la force d'une couleur particulière [36]. Nous avons utilisé cette variable visuelle pour représenter les différents types de rôle, on distingue les deux types de rôles par une dégradation de couleur.

IV.4 Principe de la transparence sémantique

Le principe de la transparence sémantique se réfère à l'utilisation de représentations graphiques dont les apparences suggèrent leur signification [36]. Par exemple, l'utilisation d'une icône d'un médecin dans un modèle de cas d'utilisation est une indication claire que l'acteur représenté par l'icône est un médecin du monde réel. L'utilisation de représentations graphiques sémantiquement transparentes accélère la reconnaissance et améliore la clarté, en particulier pour les utilisateurs débutants. Cependant, il faut faire preuve de prudence lors de l'utilisation de représentations graphiques sémantiquement transparentes car une mauvaise représentation graphique peut être trompeuse plutôt qu'utile, ce qui est pire que de ne pas avoir de représentations graphiques sémantiquement transparentes en premier lieu.
Dans la notation que nous avons proposée, on a utilisé trois icones, une pour la capacité et les deux autres pour les types des rôles (individuel, collaboratif).



IV.5 Principe du double codage

Bien que l'utilisation du texte dans les diagrammes soit visiblement interdite, elle est réellement encouragée à utiliser pour compléter mais ne pas remplacer les graphiques. Selon la double théorie du codage, l'utilisation d'une combinaison de graphiques et de textes est plus efficace que l'utilisation de l'un ou de l'autre seul [37]. Nous distinguons entre les trois types de flèches par la taille, et par le texte aussi. Par exemple la flèche liant une capacité à une autre comporte le texte « dépend », pour exprimer qu'une capacité dépend d'une autre.

IV.6 Principe de l'économie graphique

Le principe de l'économie graphique se réfère au nombre de différentes catégories de symboles utilisées dans une notation. Des études ont montré que la capacité humaine de discriminer des symboles perceptuellement distincts est d'environ six catégories [36], qui définissent une limite supérieure efficace pour la complexité graphique.

L'objectif ici devrait être d'évaluer si la notation dépasse la limite supérieure. Cependant, compter le nombre de catégories n'est pas nécessaire car tous les symboles présentés dans le Tableau 7 appartiennent à une des trois catégories dans notre notation. Par conséquent, le nombre de symboles perceptuellement distincts reste à 3. Si les icônes ont été incluses dans l'évaluation, le nombre atteindrait 6, qui sont encore dans la limite supérieure.

V. L'utilisateur final de la notation graphique

Pour décrire les rôles, et les capacités dans un SoS, nous avons proposé une notation permettant à l'utilisateur du système de le faire. L'utilisateur final de notre notation est l'expert du domaine d'application parce que personne ne peut connaître mieux les rôles pouvant participer à un SoS.

Pour décrire ce que peut faire l'expert du domaine d'application utilisant notre notation, nous présentons dans la figure 25 le diagramme de cas d'utilisation de notre application.

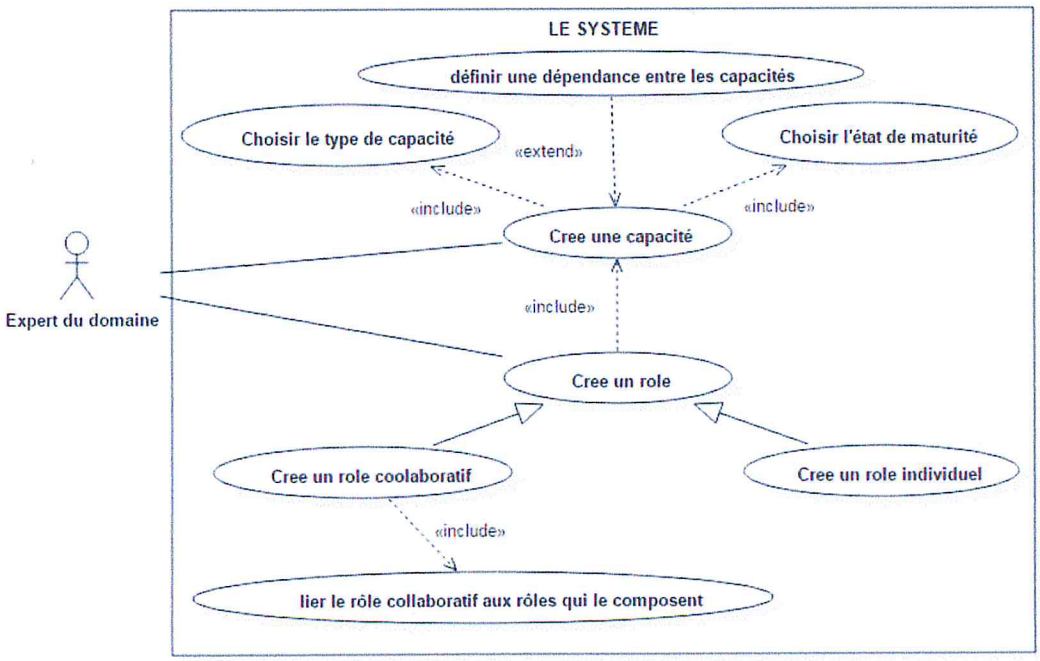


Figure 25: *Cas d'utilisation de la chaine d'outils*

1. *Le 1er cas d'utilisation : Cree une capacité :*

Ce cas permet à l'expert du domaine de créer une capacité et lui attribuer ces informations complémentaires tel que : le nom, le type et son état de maturité. De plus, il peut définir les dépendances entre les capacités.

2. *Le 2ème cas d'utilisation : Cree un rôle*

Ce cas permet à l'expert du domaine de créer un rôle et d'avoir la possibilité de choisir deux types d'entre eux, individuel et collaboratif, ce dernier se compose de plusieurs rôles individuels ou même collaboratif selon le besoin d'utilisation, donc il doit lier les rôles entre eux dans le cas d'un rôle collaboratif.

Ce cas d'utilisation dépend du cas précédent «Cree une capacité», car chaque rôle doit avoir au moins une capacité.

VI. Conclusion

Nous avons commencé ce chapitre par décrire la démarche suivie pour proposer notre notation qui permet à l'expert du domaine d'application la modélisation des capacités des sous système à travers la notion de rôle.

Nous avons décrit la première phase qui consiste à faire une analyse ontologique pour spécifier les concepts nécessaires, afin de définir les structures sémantiques correspondante (à travers le méta modèle). Dans la seconde phase nous avons proposé une notation graphique correspondant aux constructions sémantiques développées dans la première phase. Pour que la notation graphique soit simple et intuitive, nous avons utilisé six principes de construction d'une notation graphique. Nous avons terminé le chapitre par présenter les différents cas d'utilisation de notre système.

Dans le chapitre suivant nous allons présenter l'implémentation de notre méta-modèle. Nous décrivons l'environnement de notre travail, ainsi que les outils du développement utilisé avec une étude de cas.

Chapitre V: Implémentation

I. Introduction

Après avoir proposé notre méta-modèle, ainsi que la notation graphique correspondante, nous allons présenter dans ce chapitre l'implémentation de la notation graphique. Nous commençons par définir l'environnement de travail, ainsi que les outils de développement utilisés. Nous clôturons ce chapitre par la présentation de notre application à travers une étude de cas.

II. Environnement de travail et les outils utilisés

Dans notre application nous avons utilisé la plate-forme Java 8 comme environnement de développement et d'exécution ainsi que le package de modélisation Eclipse Modeling Tools avec ces différents éléments nécessaires tels qu'Eclipse Modeling Framework (EMF) et le Graphical Modeling Framework (GMF), que nous allons présenter ci-dessous

II.1 Java 8

Java 8 est la dernière version de Java. Elle offre de nouvelles fonctionnalités, des performances accrues et des corrections de bug pour améliorer l'efficacité de développement et d'exécution des programmes Java. La nouvelle version de Java est d'abord mise à disposition des développeurs afin qu'ils disposent du temps adéquat pour effectuer les opérations de test et de certification [38]. Java 8 comprend des fonctionnalités pour la productivité, la facilité d'utilisation, la programmation polyglotte améliorée, la sécurité et les performances améliorées. [39].

Nous avons utilisé le langage JAVA parce qu'en plus des performances citées ci-dessus, il comporte le package Eclipse Modeling Tools qui permet la manipulation des méta-modèles.

II.2 Le Package Eclipse Modeling Tools dans Eclipse Neon.3

Pour la réalisation de notre implémentation nous avons utilisé Eclipse IDE avec la version Neon.3, cette dernière comporte le package de modélisation "Eclipse Modeling Tools" qui fournit des outils et des runtimes pour la construction d'applications basées sur des modèles. on peut l'utiliser pour concevoir graphiquement des méta-modèles de domaine, pour exploiter ces modèles au moment du design en créant et en éditant des instances dynamiques, pour

collaborer via le support d'équipe d'Eclipse avec des installations pour comparer et fusionner des modèles et des instances de modèles afin de générer un code Java de celles-ci [40].

II.3 Eclipse Modeling Framework (EMF)

EMF est un Framework de modélisation, et de génération de code pour la construction d'outils et d'autres applications basées sur un méta-modèle. À partir d'une spécification de modèle décrite dans XMI (XML Metadata Interchange), EMF fournit des outils et un support d'exécution pour produire un ensemble de classes Java pour le modèle, ainsi qu'un ensemble de classes permettant l'affichage et l'édition basée sur les commandes du modèle et un éditeur de base [41].

Un méta-modèle est créé et défini dans le format Ecore, qui est essentiellement un sous-ensemble de diagrammes de classe UML. À partir d'un modèle Ecore, on peut générer un code Java [41].

II.4 Graphical Modeling Framework (GMF)

GMF est un Framework de l'environnement de travail Eclipse. Il fournit une infrastructure permettant l'exécution d'éditeurs graphiques basés sur les Framework EMF. Le projet GMF fournit une approche axée sur le modèle pour générer des éditeurs graphiques dans Eclipse. En définissant une boite d'outils graphique, on peut générer un éditeur graphique entièrement fonctionnel basé sur le temps d'exécution de GMF [43].

II.5 le langage de contrainte d'objet (OCL)

Le langage de contraintes d'objet est un langage formel utilisé pour décrire les expressions sur les modèles UML. Ces expressions spécifient généralement des conditions invariantes qui doivent être conservées pour le système en cours de modélisation ou des requêtes sur des objets décrits dans un modèle. Notez que lorsque les expressions OCL sont évaluées, elles n'ont pas d'effets secondaires (c'est-à-dire que leur évaluation ne peut pas modifier l'état du système d'exécution correspondant) [44].

Dans notre projet nous avons utilisé le plugin OCL de Package Eclipse Modeling Tools pour vérifier la sémantique des modèles crées.

III. Présentation de l'application

III.1 Crée un diagramme

Pour crée un diagramme on clique sur File → New → RoleCapacité diagram, comme nous montre la figure 26.

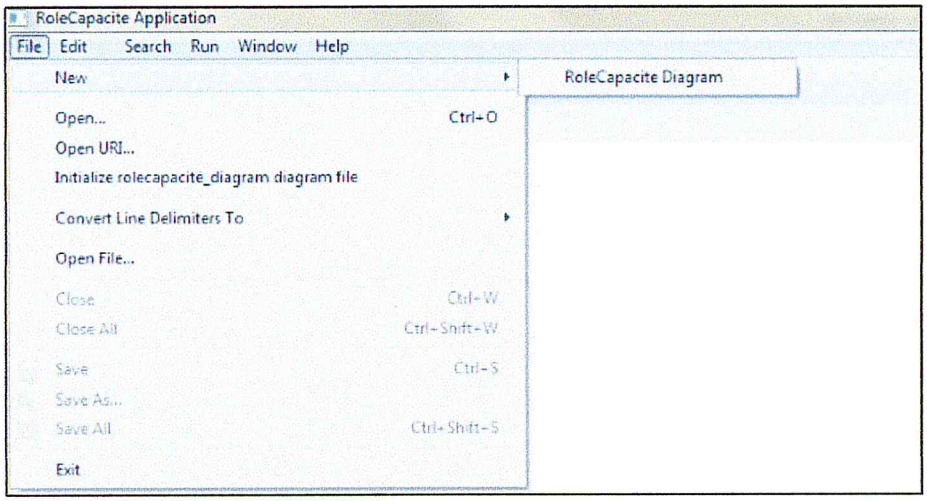


Figure 26: *Créé un diagramme rôle-capacité*

III.2 Dessiner un diagramme

Pour dessiner un diagramme nous avons une palette (partie1) qui contient une boite à outils des concepts nécessaires. Il suffit juste glisser chacun d'entre eux dans la zone de modélisation (partie2) comme nous montre la figure 27.

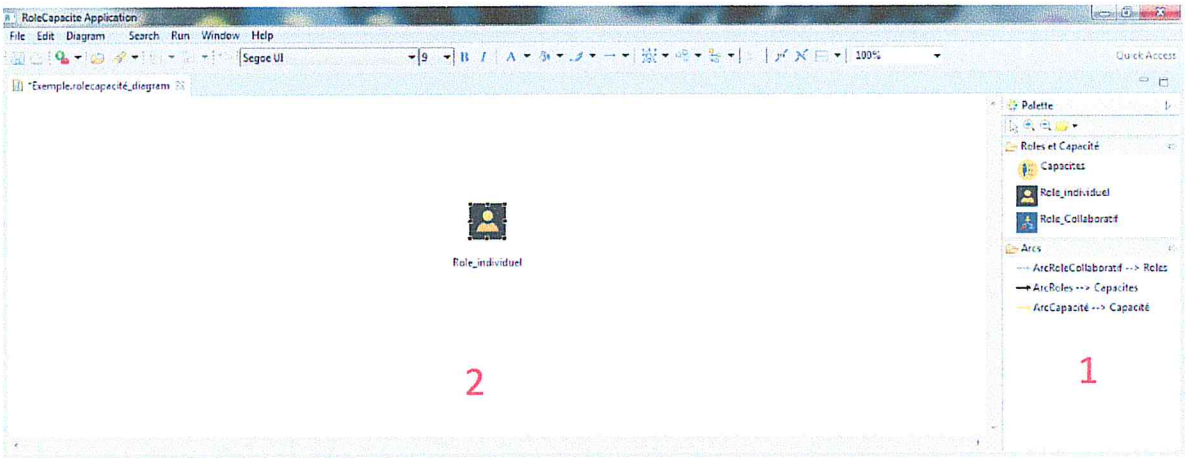


Figure 27: *Dessiner un diagramme rôle-capacité*

III.3 remplir les propriétés et valider le diagramme

Dans la zone des propriétés (partie3) on trouve les différents attributs des concepts qui doivent être rempli, ensuite nous validons la sémantique de notre diagramme par l'outil OCL (partie4) comme nous montre la figure 28.

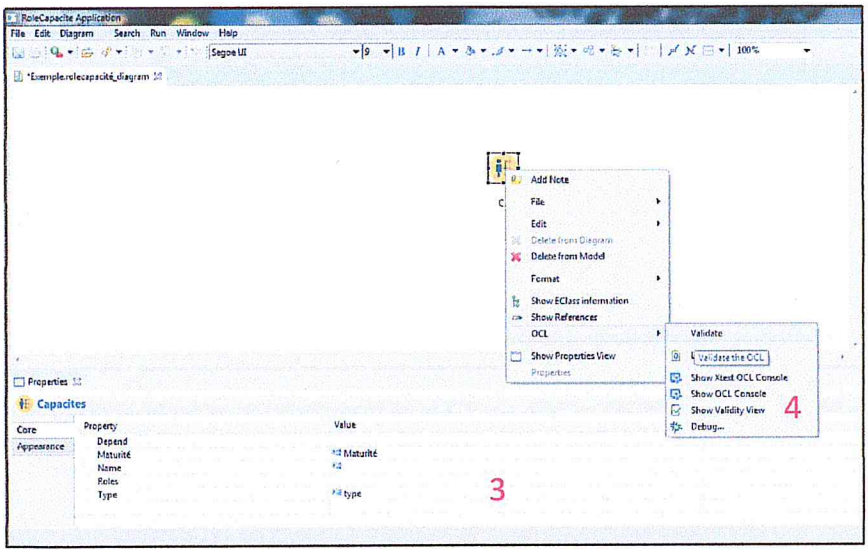


Figure 28: *Remplir les propriétés et valider le diagramme*

IV Etude de cas «Urgence SoS »

IV.1 Présentation du cas d'étude

SoS-RU (SoS de Réponse d'Urgence) est un SoS qui agit en tant que superviseur et coordinateur dans le cas d'urgences à grande échelle qui nécessitent la coordination des systèmes d'intervention d'urgence indépendants. Bien que ces systèmes soient tous développés, gérés et maintenus de manière indépendante. Le SoS-RU comporte les systèmes d'intervention d'urgence comme les services d'urgence des hôpitaux, les services d'aide médicale urgente (SAMU), les pompiers, le service de la protection civile et la police….

IV.2 Modélisation de l'étude de cas

Nous considérons dans cette étude de cas ces différents systèmes comme des rôles collaboratifs (voir la figure 29).

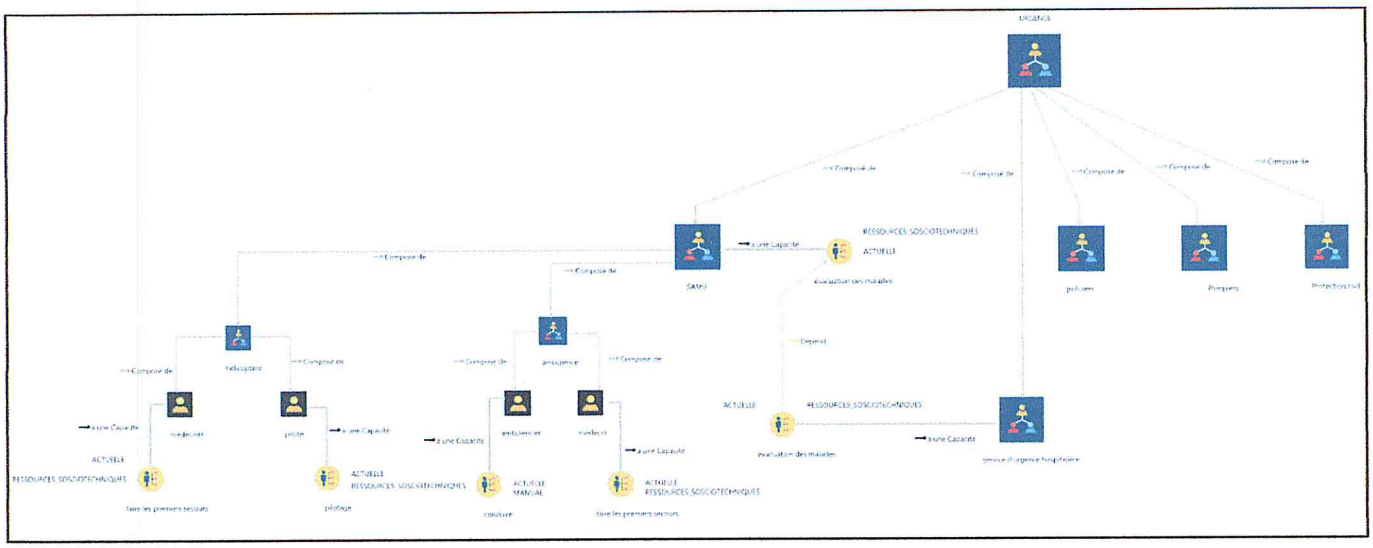


Figure 29: La structure des systèmes de l'étude de cas

Le rôle de réponse aux situations d'urgence est composé de cinq rôles, les services d'urgence des hôpitaux, les pompiers, le service de la protection civile, la police et les services d'aide médicale urgente (SAMU), ce dernier se compose de deux rôles collaboratifs qui sont l'ambulance et hélicoptère et une capacité d'évacuation des malades.

Pour l'ambulance nous distinguons deux rôles individuels dont : l'ambulancier et le médecin. Chacun d' entre eux a des capacités.

- L'ambulancier a la capacité de conduire.
- le médecin a plusieurs capacités, comme « faire les premiers secours »...

Pour l'hélicoptère nous distinguons deux rôles individuels dont : le pilote et le médecin. Chacun d' entre eux a des capacités

- le pilote a la capacité de pilotage de l'hélicoptère.
- le médecin a plusieurs capacités, comme « faire les premiers secours »...

Pour bien illustrer les détails de notre application, nous modélisons le service d'urgence hospitalière qui est composé de trois rôles, deux collaboratifs, dont la réanimation et le Bloc opératoire, et un individuel qui est le laborantin, et a une capacité d'évacuation de malades.

Premièrement, la réanimation est composée de trois rôles individuels, le médecin réanimateur, l'infirmier et l'anesthésiste. Chacun d'entre eux a des capacités.

- Le médecin réanimateur détermine la technique anesthésique, décide l'aptitude du patient, gère l'urgence vitale, évalue l'état du patient.
- L'infirmier fait les soins paramédicaux.
- L'anesthésiste réalise l'anesthésie sous la dépendance du médecin réanimateur.

Nous pouvons voir cette structure dans la figure 30.

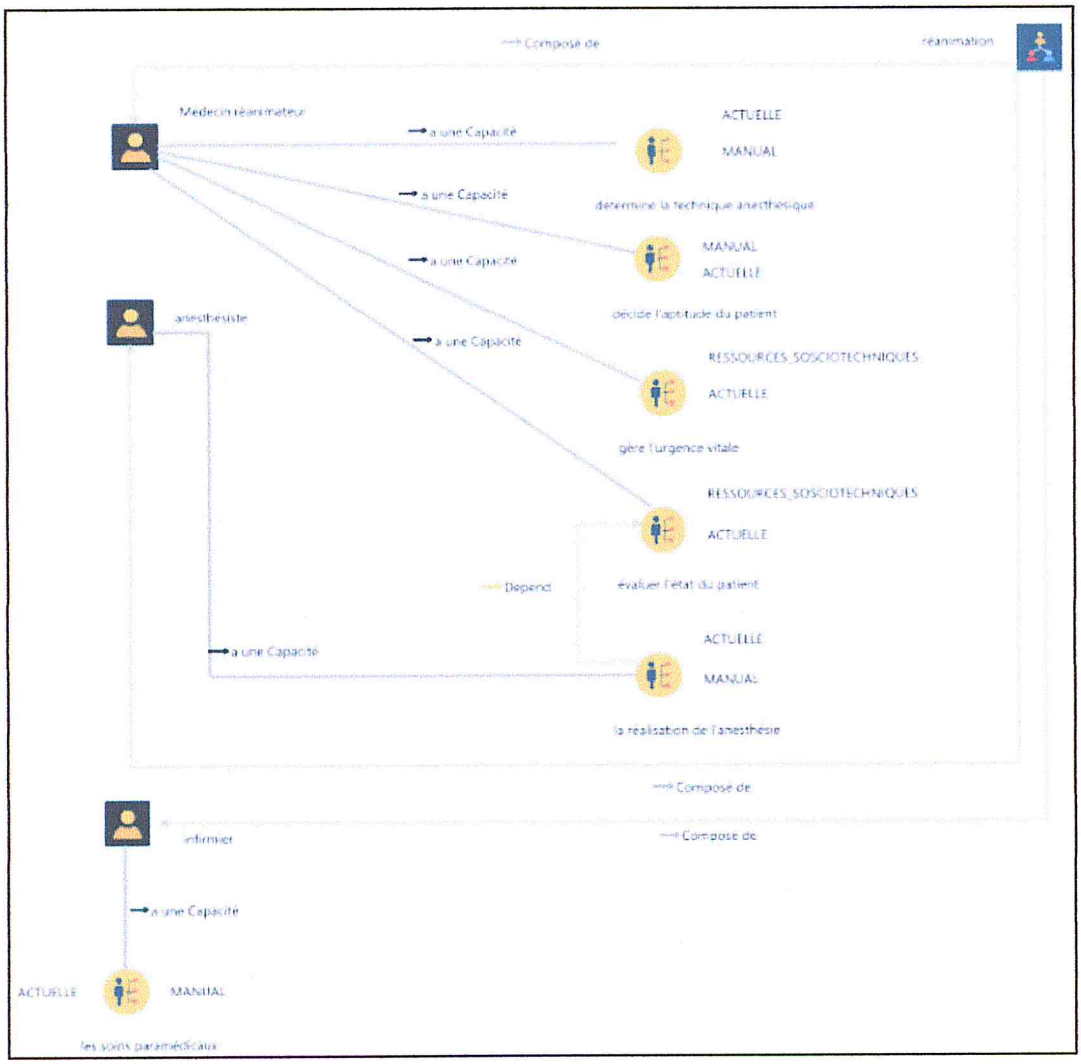


Figure 30: *La structure du rôle collaboratif « la réanimation »*

Pour le bloc opératoire, nous distinguons sept rôles individuels dont : le médecin résident, médecin assistant, médecin maître-assistant, infirmier instrumentiste, médecin réanimateur, infirmier et anesthésiste. Chacun d' entre eux a des capacités.

- L'infirmier instrumentiste dispense les produits et les matériaux pendant l'opération.
- Le médecin résident peut faire des interventions chirurgicales sous la surveillance d'un médecin assistant, donner des traitements médicaux et faire des consultations.
- Le médecin assistant a les mêmes capacités que le médecin résident.
- Le médecin maître-assistant a les mêmes capacités que celles du médecin assistant en plus il enseigne et fait des conférences.

_ Nous pouvons voir cette structure dans la figure 31.

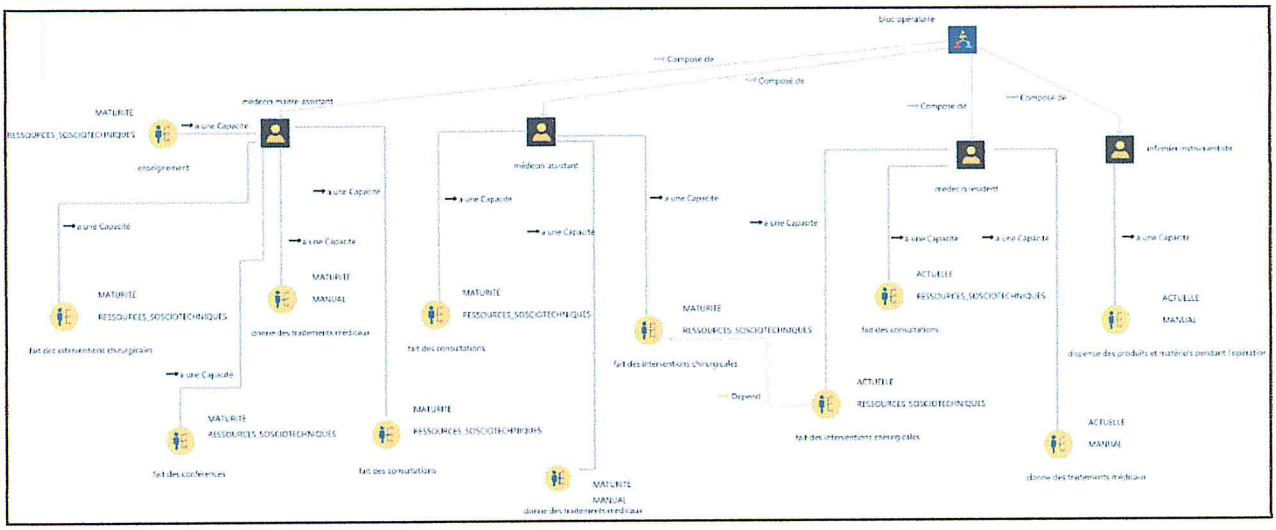


Figure 31: *La structure du rôle collaboratif « le bloc opératoire »*

Troisièmement, on trouve le rôle individuel laborantin qui a plusieurs capacités, comme « faire des analyses médicales et réaliser des bilans »

En fin, la capacité d'évacuation des malades du rôle de service d'urgence hospitalière dépend de celle du rôle SAMU (voir la figure 29).

V. Conclusion

Nous avons présenté dans ce chapitre l'implémentation de notre méta-modèle. Nous avons commencé par définir l'environnement de travail, ainsi que les outils du développement utilisés. Nous avons par la suite présenté l'application que nous avons développée suivie d'une étude de cas.

Conclusion générale

Conclusion générale

Le travail qui nous a été demandé consiste à proposer une notation graphique pour la modélisation des capacités des sous système à travers la notion de rôle. Cette dernière est une notion plus abstraite que la notion du système constituants, et nous offre le moyen donc de décrire au préalable les rôles pouvant intervenir dans le SOS. Pour qu'au moment de l'exécution du SoS, il suffit juste d'instancier les rôles.

Pour cela, nous avons suivi une démarche de deux phases. Dans la première phase, nous avons effectué une analyse ontologique pour déduire les concepts nécessaires afin de définir les structures sémantiques correspondante résumées dans un méta modèle. Dans la deuxième phase, ensuite nous avons proposé des éléments graphiques qui correspondent aux constructions sémantiques développées.

Le méta-modèle a été implémenté, on a donc développé un outil facile à utiliser, dédié principalement aux experts du domaine. Ces derniers peuvent créer des rôles individuels, des rôles collaboratifs, peuvent associer des rôles collaboratifs aux rôles qui les composent. De plus, ils peuvent associer des rôles aux capacités correspondantes, et peuvent exprimer les liens de dépendance entre les capacités. La sémantique du modèle crée est vérifiée par le plugin OCL d'éclipse.

Ce travail nous a permis d'acquérir des connaissances sur la construction des SoS, sur la modélisation des rôles et des capacités, ainsi que sur la méta-modélisation. Les perspectives envisagées pour améliorer ce travail sont multiples, nous identifions deux d'entre elles :

- La spécification des contraintes OCL sur les capacités.
- L'enrichissement de notre méta-modèle par l'attribution des rôles aux missions des SoS.

Cette perspective est très intéressante vu qu'elle permet de trouver les rôles qui vont accomplir la mission à travers les capacités requises par la mission et les capacités fournies par les rôles.

Bibliographie

[1]. Jamshidi, M. (Ed.). (2008). Systems of systems engineering: principles and applications. New York : CRC Press, p4.

[2]. Nielsen, C. B., Larsen, P. G., Fitzgerald, J., Woodcock, J., & Peleska, J. (2015). Systems of systems engineering: basic concepts, model-based techniques, and research directions. ACM Computing Surveys (CSUR), 48(2), 18.

[3]. D. Luzeaux & J-R. Ruault, (2010) *systems of systems, United* state *:* British Library Cataloguing, p14-15.

[4]. Lane, J. A., & Epstein, D. (2013). What is a system of systems and why should i care? . University of Southern California, available at http://csse.usc.edu/TECHRPTS/2013/reports/usc-csse-2013-500.pdf.

[5]. Maier, M. W. (1996, July). Architecting principles for systems-of-systems. In INCOSE International Symposium (Vol. 6, No. 1, pp. 565-573).

[6]. Mark W. Maier , *architecting principles for systems of systems* consulter 04 November 2016, www.cesames.net/fichier.php?id=252 , p5

[7]. Lock, R., & Sommerville, I. (2010, March). Modelling and analysis of socio-technical system of systems. In Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on (pp. 224-232). IEEE.

[10]. Azaiez, S. (2007). Approche Dirigée par les modèles pour le développement de systèmes multi-agents (Doctoral dissertation, Université de Savoie).

[11]. Hameurlain, N. (2005). Spécification formelle des composants d'interaction à base de rôles: compatibilité et substituabilité. In In 3èmes Journées Francophones Modèles Formels de l'Interaction (pp. 133-142).

[12]. Ferber, J., Gutknecht, O., & Michel, F. (2003, April). Agent/group/roles: Simulating with organizations. In Fourth International Workshop on Agent-Based Simulation (ABS03).

[13]. Drogoul, A., & Zucker, J. D. (1998). Methodological issues for designing multi-agent systems with machine learning techniques: Capitalizing experiences from the robocup challenge. Rapport technique LIP6, 41.

[14]. Ali, A. (2009). L'approche multi-agents pour le pilotage des systèmes complexes appliquée aux systèmes du trafic urbain (Doctoral dissertation, Université Blaise Pascal-Clermont-Ferrand II).

[15]. Picard, G. (2004). Méthodologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalité émergente (Doctoral dissertation, Université Paul Sabatier Toulouse III).

[16 Cernuzzi, L., & Zambonelli, F. (2004, April). Experiencing AUML in the GAIA Methodology. In ICEIS (3) (pp. 283-288).

[17]. Ferber, J., & Gutknecht, O. (1998, July). 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). IEEE.

[18]. DeLoach, S. A., & Wood, M. F. (2000). Multiagent systems engineering: the analysis phase (No. AFIT/EN-TR-00-02). AIR FORCE INST OF TECH WRIGHT-PATTERSONAFB OH SCHOOL OF ENGINEERING.

[19]. Wooldridge, M., Jennings, N. R., & Kinny, D. (2000). The Gaia methodology for agent-oriented analysis and design. Autonomous Agents and multi-agent systems, 3(3), 285-312.

[20]. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., & Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems, 8(3), 203-236.

[21]. Morandini, M., Dalpiaz, F., Nguyen, C. D., & Siena, A. (2014). The tropos software engineering methodology. In Handbook on Agent-Oriented Design Processes (pp. 463-490). Springer Berlin Heidelberg.

[22]. P Giorgini, M Kolp, J Mylopoulos, M Pistore , The Tropos Methodology, In Bergenti. F, Gleizes. M-P, Zambonelli. F (Eds), Méthodologies and Software Engineering for Agent Systems (pp 89-106). New York : Springer Science+Business Media

[23]. Bertolini. D, A Tropos Model-Driven Development Environment. Conference: The 18th Conference on Advanced Information Systems Engineering (CAiSE '06), Forum Proceedings, Theme: Trusted Information Systems, Luxembourg, June 5-9, 2006

[24]. Padgham L., Winikoff M. (2003) Prometheus: A Methodology for Developing Intelligent Agents. In: Giunchiglia F., Odell J., Weiß G. (eds) Agent-Oriented Software Engineering III. (Pp 174-185). Lecture Notes in Computer Science : Springer, Berlin, Heidelberg

[25]. Talib, A. M., Atan, R., Abdullah, R., & Murad, M. A. (2011). Multi agent system architecture oriented prometheus methodology design to facilitate security of cloud data storage. Journal of Software Engineering, 5(3), 78-90.

[26]. Padgham L., Winikoff M. (2005) Prometheus: A Practical Agent-Oriented Methodology. In: Henderson-Sellers. B & Giorgini. P, (eds) Agent-Oriented Methodologies. (Pp 107-135). Idea Group Inc (IGI) : Hershey, Pennsylvanie, États-Unis

[27]. Cossentino, M., Gaglio, S., Garro, A., & Seidita, V. (2007). Method fragments for agent design methodologies: from standardisation to research. International Journal of Agent-Oriented Software Engineering, 1(1), 91-121.

[28]. Sabas, A. R. S. È. N. E. (2001). Systèmes Multi-agents: une analyse comparative des méthodologies de développement. Mémoire de Maîtrise.

[29]. Scott A. DeLoach. Analysis and Design using MaSE and agentTool. *Conference : Artificial Intelligence and Cognitive Science.* presented at 12th Midwest Miami University (MAICS 2001), Oxford, Ohio, March 31 - April 1, 2001

[30]. Ferber J. (1995), Les Systemes Multi Agents: vers une intelligence collective. InterEditions : Paris, France

[31]. O. GUTKNECHT et J. FERBER: Vers une méthodologie organisationnelle de conception. Dans M.-P. GLEIZES et P. MARCENAC, éditeurs : Ingénierie des systèmes multi-agents –Actes des 7èmes JFIADSMA, pages 93–104. Hermès, 1999.

[32]. Amirat, A. (2010). Contribution à l'élaboration d'architectures logicielles à hiérarchies multiples (Doctoral dissertation, Nantes).

[33]. Cariou. E, (2016) Ingénierie des Modèles : Méta-modélisation. Consulter 06 Janvier 2017, http://ecariou.perso.univ-pau.fr/cours/idm/cours-meta.pdf

[34]. Blanc. X. (2005) MDA en action: Ingénierie logicielle guidée par les modèles. Eyrolles : Paris, France

[35] Meta Object Facility (MOF) Specification, Version 1.4.1. formal/05-05-05. Consulter 06 Janvier 2017, http://www.omg.org/spec/MOF/ISO/19502/PDF

[36]. D. Moody, "The 'physics' of notations: Toward a scientific basis for constructing visual notations in software engineering," IEEETrans. Softw. Eng., vol. 35, no. 6, pp. 756–779, Nov. 2009.

[37]. El-Attar, M., Luqman, H., Karpati, P., Sindre, G., & Opdahl, A. L. (2015). Extending the UML statecharts notation to model security aspects. IEEE Transactions on Software Engineering, 41(7), 661-690.

[38]. Oracle corporation, Informations sur Java 8, Consulter 06 juin 2017, https://www.java.com/fr/download/faq/java8.xml

[39]. Oracle corporation, Java 8, Consulter 06 juin 2017, http://www.oracle.com/technetwork/java/javase/overview/java8-2100321.html

[40]. Eclipse, Eclipse Modeling Tools, Consulter 06 juin 2017, https://www.eclipse.org/downloads/packages/eclipse-modeling-tools/neon3

[41]. Eclipse, Eclipse Modeling Framework (EMF), Consulter 06 juin 2017, https://www.eclipse.org/modeling/emf/

[42]. Eclipse, Eclipse Modeling Framework (EMF), Consulter 06 juin 2017, https://www.eclipse.org/modeling/emf/

[43]. Eclipse Consortium. Eclipse Graphical Modeling Framework (GMF) Consulter 06 juin 2017, https://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_1

[44] Information technology - Object Management Group, Object Constraint Language (OCL)
Consulter 06 juin 2017, http://www.omg.org/spec/OCL/ISO/19507/PDF