
1. Introduction

La plupart des méthodes de construction des systèmes multi-agents (SMA) prennent pour toile de fond les techniques orientées objets ou s'appuient sur les techniques de construction des systèmes à base de connaissances (SBDC) [8]. Cependant, bon nombre d'entre elles ne disposent pas d'une phase explicite d'identification des agents, pourtant l'une des tâches fondamentales liée à la construction d'un SMA est le processus de décomposition du système en agents (agentification). En pratique, cette décomposition est essentiellement « intuitive ». Notre objectif, dans ce travail, est de définir une démarche plus formelle pour l'agentification c'est-à-dire de fournir un guide complet constitué d'un ensemble de règles édictées et formalisées permettant de dériver de façon objective les agents d'un SMA.

L'interopérabilité dans un SMA n'est assuré que si les connaissances manipulées ont une sémantique claire, précise et sans ambiguïté pour tous les agents. Les ontologies [9] peuvent être utilisées pour fournir un sens aux différents symboles utilisés pour décrire un modèle du monde afin de permettre aux agents logiciels de partager la connaissance. Notre démarche se fonde sur l'utilisation d'ontologies (une ontologie du domaine d'application, une ontologie des SMA et une ontologie des fonctionnalités de l'application) décrites de façon formelle avec un langage de logique de description (LD) [1]. Une phase d'opérationnalisation [5] et un ensemble de règles permettent à la fin de dériver le squelette des agents composant le système. Nous avons appliqué la démarche proposée à un cas pratique qui est celui de l'enseignement à distance.

2. Le processus d'agentification proposé

Dans ses grandes lignes, le processus d'agentification que nous proposons comporte quatre étapes (voir *figure 1*):

- Produire une **ontologie de base** comprenant : une ontologie du domaine d'application, une ontologie « fonctionnelle » qui décrit à un certain niveau d'abstraction les fonctionnalités du système et une ontologie des SMA.
- Dériver une **ontologie complète** en enrichissant l'ontologie de base par rajout d'axiomes faisant intervenir les concepts des trois ontologies créées.
- Opérationnaliser l'ontologie complète en lui donnant une représentation permettant son utilisation conformément à l'objectif opérationnel à savoir la définition des agents. Ceci se fait en définissant les scénarii d'usage, étant entendu qu'un axiome donné peut admettre plusieurs scénarii d'usage : on obtient alors une **Ontologie Opérationnelle**.
- Mettre en œuvre l'ontologie opérationnelle pour obtenir la définition des Agents.

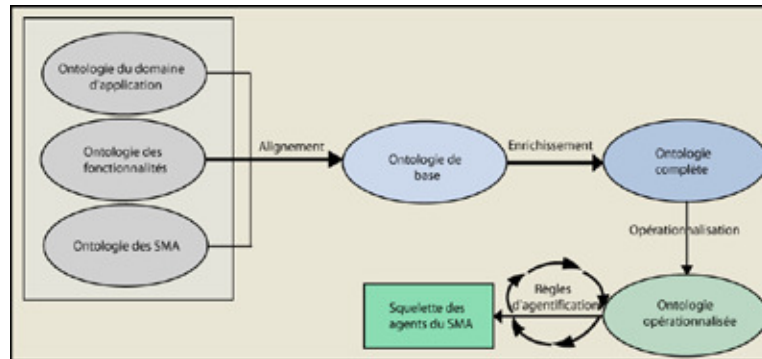


Figure 1. Etapes de notre méthode d'agentification

Ces ontologies sont formalisées à l'aide du langage de LD *ALCQI* sous forme de TBOX. Notons que les LD structurent les bases de connaissances (BDC) en deux niveaux : le niveau terminologique (TBOX) et le niveau factuel (ABOX). La TBOX est constituée d'un ensemble de concepts et de définitions de concepts formulées à travers les constructeurs d'équivalence noté \equiv , de subsumption (relation d'inclusion entre deux concepts) noté \sqsubseteq , et autres constructeurs ($\forall, \exists, \leq, \geq \dots$ etc.) fournis par le langage de LD utilisé (ici *ALCQI*). Le chapitre 1 de [1] introduit les notions de LD et de TBOX.

Nous allons illustrer notre démarche sur un cas concret qui est celui de la mise sur pied d'un système d'enseignement à distance (SED).

2.1. Construction de l'ontologie de base

La modélisation des SMA passe par la modélisation des concepts, des buts, des rôles et des interactions entre objets [7], les différentes ontologies constituant l'ontologie de base ont pour but de décrire ces entités.

- L'ontologie du domaine d'application

Pour la construction de l'ontologie du domaine d'application, nous avons procédé par l'encodage en *ALCQI* d'un diagramme de classes UML représentant le modèle conceptuel du domaine, en utilisant pour cela les règles de transcription proposées dans [2]. Nous donnons ci-dessous des descriptions *ALCQI* de quelques concepts :

$Etudiant \sqsubseteq \forall nom \bullet String \sqcap \forall email \bullet Email \sqcap \forall DernierDiplome \bullet Diplome \sqcap (\geq 1(inscrit)^- \bullet Inscription)$

$Enseignant \sqsubseteq \forall nom \bullet String \sqcap \forall email \bullet Email \sqcap (\geq 1(envoi) \bullet \top)$

$Reponse \sqsubseteq (\geq 1(correspond) \bullet \top) \sqcap (\leq 1(correspond) \bullet \top) \sqcap \forall (\geq 1(envoi)^- \bullet \top) \sqcap \forall (\leq 1(envoi)^- \bullet \top)$

- L'ontologie des fonctionnalités

Pour la mise sur pied de l'ontologie des fonctionnalités d'un système, nous proposons ici une méthode de passage d'un diagramme des cas d'utilisation à une

ontologie des fonctionnalités formalisée en *ALCQI*. En effet, les cas d'utilisation proposé par le langage UML offre une bonne modélisation fonctionnelle d'un système.

Notre approche propose de prendre en compte les acteurs, les cas d'utilisation et les relations. Un acteur sera représenté par un concept atomique, un cas d'utilisation également. Les relations (*participe*, *include*, *extends*) seront représentées par des rôles atomiques. Le **tableau 1** donne un récapitulatif de la formalisation *ALCQI* des descriptions de concepts et rôles de l'ontologie des fonctionnalités. Dans ce tableau, A , A_1 , et A_2 sont des acteurs, U , U_1 et U_2 des cas d'utilisation.

Expressions	Traduction en ALCQI
A_1 généralise A_2	$A_2 \sqsubseteq A_1$
A participe à U	$A \sqsubseteq \exists \text{participe} \bullet U$
U_1 inclut U_2	$U_1 \sqsubseteq \exists \text{include} \bullet U_2$
U_1 généralise U_2	$U_2 \sqsubseteq U_1$
U_1 étend U_2	$U_1 \sqsubseteq \exists \text{extends} \bullet U_2$

Tableau 1. Récapitulatif des traductions *ALCQI* de l'ontologie des fonctionnalités

Pour le cas de l'enseignement à distance, on peut entre autres identifier les cas d'utilisation "Faire une inscription", "Faire une recherche documentaire", "Faire une évaluation", les acteurs "Etudiant", "responsable du site". Les noms de concepts représentant les acteurs commencent par la lettre A tandis que ceux des cas d'utilisation commençant par la lettre U . On a par exemple les 2 descriptions de concepts suivantes :

$$A_Etudiant \sqsubseteq (\exists \text{participe} \bullet U_F_Inscription) \sqcap (\exists \text{participe} \bullet U_S_Cours) \sqcap (\exists \text{participe} \bullet U_F_Recherche) \sqcap (\exists \text{participe} \bullet U_F_Evaluation)$$

$$U_F_Recherche \sqsubseteq \exists \text{include} \bullet U_Authentification$$

- L'ontologie des SMA

L'ontologie du domaine des SMA décrit les entités qui doivent faire partie d'un SMA et leurs relations, ceci indépendamment du domaine opérationnel. Pour construire cette ontologie, nous utilisons des spécifications bien connues des SMA [4]. On a par exemple les deux descriptions suivantes :

$$Agent \sqsubseteq Entité \sqcap (\geq 1 \text{possède} \bullet Compétence) \sqcap (\geq 1 \text{possède} \bullet Ressource) \sqcap (\exists \text{suit} \bullet But) \sqcap (\geq 1 \text{manipule} \bullet Objet) \sqcap (\geq 1 \text{accomplit} \bullet Action) \sqcap (\exists \text{présente} \bullet Comportement) \sqcap (\geq 1 \text{perçoit} \bullet Information) \sqcap (\geq 1 \text{interagit} \bullet Environnement)$$

$$But \sqsubseteq Objectif \sqcap \exists \text{correspond} \bullet Rôle$$

- Alignement des trois ontologies

L'utilisation au sein d'un même système à base de connaissances (SBDC) de plusieurs ontologies portant sur des domaines complémentaires et bâties sur le même formalisme de représentation (ici *ALCQI*) nécessite un alignement [6]. L'alignement

permet d'établir les liens sémantiques entre concepts et relations de plusieurs ontologies afin de permettre la manipulation conjointe de ces dernières dans une même TBOX. Ainsi, aligner les ontologies du domaine, des fonctionnalités et des SMA consiste à déterminer les appariements (similarités) de concepts et de relations de ces ontologies. Ces appariements peuvent représenter des identités entre primitives conceptuelles (c'est-à-dire les concepts et les relations), mais également des subsomptions, ou d'autres liens conceptuels (exclusion, incompatibilité, ...etc). Une fois les appariements déterminés, on rajoute dans la TBOX les différentes propriétés reliant les primitives conceptuelles des ontologies. On obtient ainsi l'**ontologie de base**.

2.2. Construction de l'ontologie complète

L'ontologie complète est obtenue par enrichissement de l'ontologie de base en ajoutant des schémas d'axiomes et axiomes sous la forme de formules de la logique des prédicats du premier ordre (LPO). L'ajout de schémas d'axiomes consiste à décorer les concepts et les rôles en précisant les types des relations auxquelles ils appartiennent. Il s'agit en général des relations de réflexivité, d'anti-réflexivité, de symétrie, d'antisymétrie, de transitivité, d'exclusivité, d'incompatibilité entre deux relations ou deux concepts, de cardinalité d'une relation et de signature d'une relation. Par exemple le rôle « *possede* », de signature (*Agent, Compétence*), est une relation *Antiréflexive et Antisymétrique*. L'ajout d'axiomes se base sur les axiomes de couverture fonctionnelle, d'existence d'agents et de coopération inter-agents. Les cas d'utilisation sont assimilés aux tâches et les relations sont les interactions entre les agents qui exécutent les tâches et l'environnement [3]. Nous avons dégagé et ajouté dans notre ontologie un certain nombre d'axiomes dont :

- Tout cas d'utilisation doit être lié à une tâche du système.

$$\forall u U(u) \Rightarrow \exists t T\grave{a}che(t) \wedge l i \acute{e}(t, u) \quad (\mathbf{A1})$$

- Pour toute tâche du système, il doit exister au moins un agent pour l'exécuter.

$$\forall t T\grave{a}che(t) \Rightarrow \exists a Agent(a) \wedge ex\acute{e}cute(a, t). \quad (\mathbf{A2})$$

- Si un agent ne sait pas exécuter une tâche, il peut déléguer la tâche à un autre agent appartenant à son réseau d'accointance.

$$\forall a, t Agent(a) \wedge T\grave{a}che(t) \wedge (\neg comp\acute{e}tence(a, t)) \Rightarrow \exists a_1 Querable(a_1, t) \wedge accointance(a, a_1) \quad (\mathbf{A3})$$

- Si des agents veulent accéder à la même ressource, il faut la présence d'un autre agent pour gérer le partage de la ressource entre eux.

$$\forall a_1 \dots a_n \bigwedge_{i=1}^n Agent(a_i) \wedge Ressource(r) \bigwedge_{i=1}^n utilise(a_i, r) \Rightarrow \exists ag Agent(ag) \wedge partageRessource(ag, r, a_1 \dots a_n) \quad (\mathbf{A4})$$

Ces axiomes permettent de faire les liens entre les concepts des trois ontologies constituant l'ontologie de base. L'ontologie ainsi enrichie va représenter l'**ontologie complète**.

2.3. Construction de l'ontologie opérationnelle

Le processus d'opérationnalisation consiste à plonger l'ontologie dans un langage opérationnel conformément à l'objectif opérationnel visé qui ici est la définition des agents devant constituer le futur SMA. Pour opérationnaliser notre ontologie, nous utilisons la méthode proposée dans [5] en choisissant des scénarii d'usage, qui peuvent être inférentiels ou de validation. Les schémas d'axiome et axiomes de domaine doivent être opérationnalisés à l'aide des primitives du raisonnement que sont : les **contraintes positives** (si l'hypothèse est présente alors la conclusion doit également l'être), les **contraintes négatives** (si l'hypothèse est présente alors la conclusion doit être absente) et les **règles d'inférence** (si l'hypothèse est présente alors la conclusion doit être ajoutée). Les axiomes à ajouter lors de la phase d'enrichissement de l'ontologie sont de plusieurs types, et pour chaque type nous donnons des règles d'opérationnalisation.

Dans ce qui suit, H désigne une conjonction de concepts ou de relations, r et $r_i, i=1, \dots, n$ des relations, R une conjonction de relations, C et $C_i, i=1, \dots, n$ des concepts, x et $x_i, i=1, \dots, n$ des instances de concepts. Nous noterons **cii** (resp. **cie**) le *contexte inférentiel implicite* (resp. *explicite*), et **cvi** (resp. **cve**) le *contexte de validation implicite* (resp. *explicite*).

• **Axiomes de type 1** : $H \rightarrow r_1(..) \wedge \dots \wedge r_n(..)$

Par exemple, une relation réflexive r est traduite par : $\forall x C(x) \rightarrow r(x, x)$.

- **cii** : Ajout d'une règle implicite $H \rightarrow \text{action}(r_1(..) \wedge \dots \wedge r_n(..))$. L'action va se charger de vérifier dans la BDC l'existence des relations r_i dès que l'hypothèse H est présente. Dans le cas où elles n'existent pas elles seront tout simplement inférées et ajoutées.

- **cie** : Ajout d'une règle explicite $H \rightarrow \text{action}(r_1(..) \wedge \dots \wedge r_n(..))$, l'action étant définie comme dans le **cii**, et de n contraintes négatives implicites:

$H (\wedge r_i(..))_{i=1..n, i \neq j} \rightarrow \text{action1}(r'_j(..))_{j=1..n}$ où les r'_j sont les relations exclusives de r_i . **Action1** va définir des contraintes dès que l'hypothèse H et les relations r_i seront présentes pour qu'aucune relation exclusive de r_i ne soit présente. Si aucune relation exclusive n'est présente, la contrainte est remplacée par m contraintes négatives implicites: $H (\wedge r_i(..))_{i=1..n, i \neq j} \rightarrow \text{action1}(r'_{jk}(..))_{k=1..m}$ où les r'_{jk} sont toutes des relations incompatibles de r_i . Ces contraintes excluent la présence des relations incompatibles aux relations r_i dès que l'hypothèse H et les relations r_i sont présentes.

- **cvi** (resp. **cve**) : Ajout de n contraintes négatives implicites (resp. explicites) et si aucune relation exclusive n'est présente, ajout de m contraintes négatives.

• **Axiomes de type 2** : $H \rightarrow \neg(r_1(..) \wedge \dots \wedge r_n(..))$

Par exemple, l'anti-réflexivité d'une relation r est traduite par : $\forall x C(x) \rightarrow \neg(r(x, x))$.

- **cii** : Ajout de n règles implicites ou éventuellement m règles implicites comme dans le cas du **cie** des axiomes de type 1.

- *cie* : Comme dans le *cii* mais de plus une contrainte négative

$H \rightarrow \neg(r_1(..) \wedge \dots \wedge r_n(..))$ doit être ajoutée.

- *cvi* ou *cve* : Ajout d'une contrainte négative $H \rightarrow action2(\neg(r_1(..) \wedge \dots \wedge r_n(..)))$. *Action2* va vérifier qu'il n'y a pas de relation entre certaines instances de concepts une fois que H est vérifié.

• **Axiomes de type 3** : $H \rightarrow \exists x_1, \dots, x_n C_1(x_1) \wedge \dots \wedge C_n(x_n) \wedge R$

Par exemple, l'axiome d'existence des tâches $\forall u U(u) \Rightarrow \exists t Tache(t) \wedge I \acute{e}(t, u)$.

- *cii* (resp. *cie*) : l'axiome est opérationnalisé sous la forme d'une règle implicite (resp. explicite) $H \rightarrow action(\exists x_1, \dots, x_n C_1(x_1) \wedge \dots \wedge C_n(x_n) \wedge R)$, l'action devant vérifier dès que l'hypothèse est présente l'existence des instances x_i et des relations contenues dans R , si elles n'existent pas elle devra inférer ces instances, et de n contraintes implicites négatives : $H (\wedge C_i(x)) \rightarrow action3(C_j(x))$ où C_i et C_j sont des concepts disjoints et *action3* permet d'éviter la présence d'une instance x appartenant aux concepts C_i et C_j .

- *cvi* ou *cve* : l'axiome est opérationnalisé sous la forme $H \rightarrow action4(\exists x_1, \dots, x_n C_1(x_1) \wedge \dots \wedge C_n(x_n) \wedge R)$ où *action4* va vérifier que les instances x_i existent et appartiennent aux concepts C_i , qu'aucune instance n'appartient à des concepts incompatibles et enfin que les signatures des relations contenues dans la conjonction R sont respectées.

Illustration : Pour illustrer ces règles d'opérationnalisation, considérons l'axiome **A2**. Cet axiome (de type 3) sera opérationnalisé, selon les contextes, de la façon suivante :

cii (resp. *cie*): Ajout d'une règle implicite (resp. explicite) $\forall t Tache(t) \Rightarrow action(\exists a Agent(a) \wedge execute(a, t))$. L'action va vérifier pour toute tâche l'existence d'au moins un agent qui l'exécute, si aucun agent n'existe, elle devra en inférer.

cvi ou *cve* : Ajout d'une contrainte $\forall t Tache(t) \Rightarrow action4(\exists a Agent(a) \wedge execute(a, t))$. *Action4* va vérifier que l'agent a existe et que ce dernier exécute la tâche t .

2.4. Dédution des agents du SMA

Il est question de mettre en œuvre¹ l'ontologie opérationnelle pour déduire les différents agents. L'enchaînement des règles d'existence comme A1, A2, A3 ou A4 en contexte inférentiel explicite ou implicite, permet de déduire les tâches (A1), les agents qui les exécutent (A2), y compris les agents négociateurs (A4) à la charge de gérer les accès aux ressources afin d'éviter les conflits. D'autres axiomes ont montré l'existence d'un agent planificateur qui détermine les actions à entreprendre pour exécuter une tâche ainsi qu'un agent coordonnateur chargé de coordonner les actions de plusieurs agents. Chacun de ces agents présentent des services et les services nécessitent des compétences pour être exécutés. Ainsi seront affectés à chaque agent en fonction du rôle joué dans le système, les services, compétences et ressources associés. Pour notre SED, l'axiome A1 permet de créer pour le cas d'utilisation *U_D_Inscription* (demander des inscriptions) une tâche *ta* qui lui est liée. A2 crée pour cette tâche, un agent *ag*

¹ La mise en œuvre se fait avec les outils libres Java et Eclipse avec son plugin EJIP pour JADE (Java Agent Development Framework) et le moteur Prolog gratuit SWI-Prolog.

chargé de l'exécuter, le service « *DemandeInscription* » lui est donc attribué. Selon A3, cet agent appelle un agent *agl* chargé de la tâche « *authentication* » qui est une sous-tâche de *ta*. Par ailleurs, chaque attribut du concept « *Etudiant* », tel que *Nom*, *DernierDiplôme*, *email*, est une compétence de l'agent *ag*, tandis que l'acteur *Etudiant* intervenant dans le processus d'inscription est considéré comme une ressource. L'axiome d'existence de l'agent *ag* doit être opérationnalisé dans un contexte inférentiel explicite étant donné qu'il revient à l'étudiant de solliciter une inscription.

3. Conclusion

Ce premier développement de SMA (enseignement à distance) s'appuyant sur les ontologies a montré que cette approche est intéressante et prometteuse car formalise le processus de développement des SMA, notamment la phase d'identification des agents. D'autres applications et l'achèvement du processus d'automatisation de cette démarche restent nécessaires pour valider l'approche proposée. D'autre part, les diagrammes d'activité et de collaboration d'UML pourront être utilisés afin de mieux décrire les tâches, et les interactions inter-agents. Afin de montrer que notre approche n'est pas isolée, des travaux sont en cours pour l'intégrer au sein d'une méthode de construction bien connue des SMA : la méthode GAIA.

4. Bibliographie

- [1] F. Baader et al. « *The description logic handbook : Theory, implementation and applications* », Cambridge University Press, Cambridge, UK, 2003. ISBN 0-521-78176-0.
- [2] D. Berardi, D. Calvanese, G. De Giacomo, « Reasoning on UML class diagrams using description Logics based systems », *In proceedings of KI'2001 Workshop on applications of description logics*, Vienna, Septembre 18, 2001.
- [3] A. Chella, « Applying UML Use Case Diagrams to Agents Representation », *AIIA 2000*.
- [4] J. Ferber, « *Multi-Agent Systems : An Introduction to Distributed Artificial Intelligence* », Addison Wesley, 1999. ISBN : 0-201-36048-9.
- [5] F. Fürst, « Contribution à l'ingénierie des ontologies : une méthode et un outil d'opérationnalisation », thèse de Doctorat d'Informatique, Ecole Polytechnique de l'Université de Nantes (EPUN), 2004.
- [6] F. Fürst et F. Trichet, « Aligner des ontologies lourdes : une méthode basée sur les axiomes », article, Ecole des Mines de Nantes, 2005.
- [7] R. Girardi, C. Gomes de Faria, L. Marinho, « Ontology-based domain modeling of multi-agent systems », *OOPSLA 2004*, pp. 51--62. Vancouver, Canada. October 24th to 28th, 2004.
- [8] G. Picard, « Méthodologie de développement des SMA adaptatifs et conception des logiciels a fonctionnalité émergente », thèse de doctorat, Université Paul Sabatier, 2004.
- [9] M. Uschold, M. Gruninger, « Ontologies : principes, methods and application », *Knowledge Engineering Review*, Vol. 11, Issue 2, pp. 93-155, 1996.