I.J.Modern Education and Computer Science, **2013, 1, 42-55**
Published Online January 2013 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijmecs.2013.01.06

An Ontology-Based Approach for Multi-Agent Systems Engineering

Souleymane Koussoube
Institut Africain d'Informatique (IAI), Libreville, Gabon
Email: skoussoube@yahoo.fr

Armel Ayimdji
Département de Genie Informatique, Institut Universitaire de Technologie, Université de Douala, Cameroun
Email: ayimdji@gmail.com

Laure Pauline Fotso
Département d'Informatique, University de Yaoundé I, Cameroun
Email: laurepfotso@yahoo.com

***Abstract*—** This paper presents OBAMAS (Ontology-Based Approach for Multi-Agent Systems engineering), an ontology-based contribution to Agent Oriented Software Engineering. We propose a formal process for agentification, starting with an analysis phase which consists of the construction of three formal ontologies (a domain ontology, an ontology of functionalities, and an ontology of multi-agents systems) and their alignment to merge in a single one. The second step, which is a design phase, consists of the operationalization of the single ontology in order to infer in a more formal way the agents of the system. A case study is introduced to illustrate OBAMAS and to show its use and effectiveness in a real application, a distance learning system.

***Index Terms*—** Multi-agent system, Ontology, Ontology alignment, Operationalization, Description logics

I. Introduction

These last years, progress in software engineering were realized by the development of more and more complex, dynamic and often distributed systems. These progresses sometimes consisted in granting more autonomy to software in order to gain efficiency and robustness. These efforts gave birth to multi-agents systems (MAS) [1] and then, to Agent Oriented Software Engineering (AOSE) whose ambition is to provide specific methods for MAS development. Indeed, although most of MAS development methods are based on already existing software engineering paradigms such as object oriented (OO) or knowledge-based systems (KBS), these paradigms are not really suitable for MAS. There are fundamental differences between the OO view and the Agent Oriented (AO) view. For example, to set up MAS, we have to take into account the autonomy of agents. Agents embody a stronger notion of autonomy than objects, and, in particular, they decide for themselves whether or not to perform an action on request of another agent. This distinction between objects and agents has been nicely summarized in the following slogan: "*Objects do it for free; agents do it because they want to*" [1]. AOSE methodologies were developed to deal with these peculiarities of agents. Our contribution is in this context. We propose OBAMAS (Ontology-Based Approach for Multi-Agents Systems engineering), an approach for MAS development based on the intensive usage of ontologies.

The state of the art on AOSE methodologies [1, 2, 3, 4, 5, 6, 8] shows that many of them does not provide an explicit step of agents identification. In practice, the decomposition of the system into agents is essentially "intuitive". Our objective in this work is to define a more formal approach for the identification of agents (agentification) that is to provide a complete guide with a set of formalized rules allowing inferring in an objective way the agents of MAS. The present work extends a previous one introduced in [9] by bringing a better structure of the approach, the deepening of the various steps, as well as a comparison with some existing methodologies.

The rest of this article is organized as follows. Section II recalls the notions of agents and MAS, and then ontologies and their contributions in MAS. We present two key concepts in section III: the alignment and the operationalization of ontologies. In Section IV, we give a quick presentation of OBAMAS through the description of its different steps followed by an illustration of the methodology with a representative case study (a distance learning system) to show its use and effectiveness in a real application. In section V, we compare our methodology to related works in literature. A summary of the special features of our methodology is presented in section VI while section VII concludes this work.

II. Agents, Multi-Agents Systems and Ontologies

Agents have been studied and defined in several ways. We adopt a definition which is very often used in the MAS community:

Copyright © 2013 MECS

"*An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.*" [1].

Here we focus on intelligent agents that are capable of reactive (ability to perceive environmental changes and respond to these changes), proactive (capacity to take initiatives to achieve their design objectives) and social (ability to interact with other agents) behaviours [10]. Therefore, it makes sense to think that an agent is only meaningful if it is in a system where there are other agents with which it interacts so that the overall system performs the functions for which it was designed. The ability of agents to interact is essential, that's why there is a popular slogan in the MAS community:

"*There's no such thing as a single agent system.*" [1]

A MAS differs from a collection of independent agents by the fact that agents interact together to achieve a task or to reach a common goal. However, interoperability between agents is possible if knowledge is defined unambiguously for each agent. Ontologies [11] can be used to provide meaning to symbols in order to enable software agents to share and reuse knowledge. In the context of MAS, ontologies have been widely recognized for their outstanding contributions to interoperability, reusability, activities for MAS development and for MAS operations [7]. This is why FIPA[1] has defined a standard for the use of ontologies in MAS. Ontologies can therefore be used at all levels in the MAS development process. We believe that an appropriate description of components of the system and its functionalities by the use of formal ontologies can improve the identification and the structure of the different agents in the MAS.

III. Alignment and Operationalization of Ontologies

A. Alignment

In practice, several researchers working in different contexts develop and use different ontologies. The alignment is a way to integrate various ontologies and enable cooperation between them in order to use them together. The alignment operation consists, in the case of two ontologies as input, to output correspondences between elements (concepts and relations) of these ontologies. These correspondences may be equivalence, subsumption, disjunction of classes, temporal/spatial relations, fuzzy relations, etc.

[13] argues that merging two ontologies and then test each pair of concepts and roles via the *subsumption* relationship is good enough to match the terms having the same interpretation (or a subset of the interpretation of the other). The illustration is done on some concepts from two different ontologies formalized in a DL language.

In the first ontology, a Micro-company is defined as a company with at most five employees as follows:

$$\texttt{Micro-company} = \texttt{Company} \sqcap \leq 5\texttt{employee}$$

In a second ontology an SME (Small and Medium sized Enterprise) is defined as a firm with at most ten associates as follows:

$$\texttt{SME} = \texttt{Firm} \sqcap \leq 10\,\texttt{associate}$$

The following correspondences are then established:

$$\texttt{Company} \equiv \texttt{Firm}$$

$$\texttt{associate} \sqsubseteq \texttt{employee}$$

These two correspondences are used to infer a third one:

$$\texttt{Micro-company} \sqsubseteq \texttt{SME}$$

All the correspondences found here constitute an alignment of the two previous ontologies and they could be used in tasks using the concepts of both ontologies.

B. Operationalization

An ontology is a conceptual representation of knowledge, apart from the use that can be made. To integrate an ontology in a Knowledge base (KB), it is convenient to transcribe it into a suitable form for the intended use of the KB, that is to specify the semantics of the manipulation of axioms; this semantics being related to the application to be developed but not to the considered field. Therefore, to describe knowledge in terms of concepts, relations and their properties is not usually sufficient to achieve the operational goal of a Knowledge Based System (KBS). After the ontologization that leads to a formal ontology, an operationalization phase will follow to produce an operational ontology that is a formal representation suited to a specific application[2] [14].

The operationalization of an ontology consists:

- To specify the operational semantics that is added to the ontology to describe the reasoning mechanisms that will be implemented in the intended system. However this operational semantics does not change the formal semantics, but simply complete it by specifying reasoning mechanisms in the intended system.
- To translate the conceptual representation of the ontology in an operational language, this translation being constrained and directed by the operational semantics already specified.

The form to give to the ontology in a KBS must be operational in that the operationalization formalism must

[1] The Foundation of Intelligent Physical Agents. http://fipa.org

[2] The most content of this section on the operationalization of ontologies is from [14]

provide reasoning mechanisms to handle information in the KBS. For example, to perform automated reasoning, the operational formalism must allow the representation of derivation rules and effective mechanisms for implementation. To achieve this, we need to choose among the operations authorized by the formal semantics of the ontology, the ones that will be implemented in an operational system. Then, the issue is to define operationalization mechanisms for this transcription: *usage scenarios* [14, 15] can be used to address the problem.

1. Usage scenarios and contexts of use

According to [15], the operationalization of an ontology is done for a well-defined operational use, characterized by a specific usage scenario. The *usage scenario* describes the goal of the knowledge specified in the ontology. Such a scenario should describe how the elements of the ontology will contribute to the reasoning in the system. However, only relations and concepts descriptions (the axiomatic level) of the ontology will be concerned by the description of the usage scenario. Indeed, in the terminological level, the representation of a concept or a relation consists of a term and this representation doesn't change as a function of the operational objective of the system. The axiomatic level of the ontology distinguishes two types of knowledge:

- *Axiom schemas*. They are used to structure sets of conceptual primitives at the terminological level. These axiom schemas involve one or more concepts, or one or more relations. Among others, relations include *symmetry*, *transitivity*, *minimum* and *maximum cardinalities* of relations. For concepts, we have *subsumption* (to organise concepts into a hierarchy) or *disjunction* between concepts.
- *Axioms*. They are present only in heavy ontologies and represent the intentions of concepts and relations of the domain and, in general, knowledge not having a strictly terminological character [16]. They may not express a property on one or two primitives in particular, but rather represent properties involving several primitives. Thus, in MAS for example, the axiom "*Each agent provides at least one service*" can be seen as a property of cardinality for the ability of an agent to provide a service. However, the axiom "*If two agents A and B want an exclusive access to the same resource, there must be another agent C to manage the shared resource*" cannot be expressed as an algebraic property or a cardinality of a particular conceptual primitive and will be represented in the ontology of MAS by an axiom. Axioms are expressions that are always true. Their inclusion in ontologies can have several objectives: to define the meaning of components, to set restrictions on the value of some attributes, to verify the validity of the specified information or to deduce new ones.

The description of the usage scenario will therefore consist to specify how the axioms (and axiom schemas) will serve for reasoning in the intended system. The *context of use* of an axiom is the description of the role this axiom will play in the reasoning mechanisms implemented in the KBS. So, the operationalization assumes that a context of use is defined for each axiom. Contexts of use of axioms and axiom schemas of the ontology constitute the usage scenario of the ontology for the intended application.

2. The choice of a usage scenario

The operational form of an ontology combines inferential mechanisms and validation mechanisms to automatically manipulate the knowledge. For example, a computer-based teaching assistant should allow the user to apply the knowledge of a given field to derive new information or to validate his work. Such a system should also provide inferences and automatic evaluations of students.

We can distinguish two kinds of usage scenario:

- *Validation scenarios*, where ontological knowledge is used to validate knowledge with respect to the semantics of a domain;
- *Inferential scenarios*, where ontology knowledge is used to produce new information for a particular case.

On the other hand, an axiom can be triggered by the user of the KBS (an *explicit* use), or can be automatically applied by the system (*implicit* use). Then, to describe the role of an axiom in an ontology-based KBS, [14] propose four contexts of use:

- *Explicit inferential context of use* where the user triggers the axiom to produce new assertions: it is called an *explicit rule*.
- *Implicit inferential context of use* where the axiom is automatically applied by the system to produce new assertions: it is called an *implicit rule.*
- *Explicit validation context of use* is triggered by the user to monitor compliance of certain semantic knowledge with respect to the domain: it is called an *explicit constraint.*
- *Implicit validation context of use* is automatically applied by the system to test the compliance of a semantic knowledge base with respect to the domain: it is called an *implicit constraint.*

The operationalization of an ontology requires to choose a language for representing terminological knowledge and axioms, and to manipulate these representations for reasoning. The Operationalization language must be adapted to the domain to allow manipulating a representation of the axioms of the domain in agreement with the usage scenario. In this paper we use the First Order Logic (FOL) to represent axioms which cannot be represented in DL.

Copyright © 2013 MECS

IV. OBAMAS METHODOLOGY

A. The phases of the OBAMAS methodology

Generally, four main phases are concerned in software engineering: requirements, analysis, design and implementation. However, the aim of OBAMAS being to help the MAS developer to formally identify the agents of the system, its scope is limited to analysis and design phases (Fig. 1). The advantage being that the developer can then use his favourite development tools (platform, programming language, etc.) for the implementation.

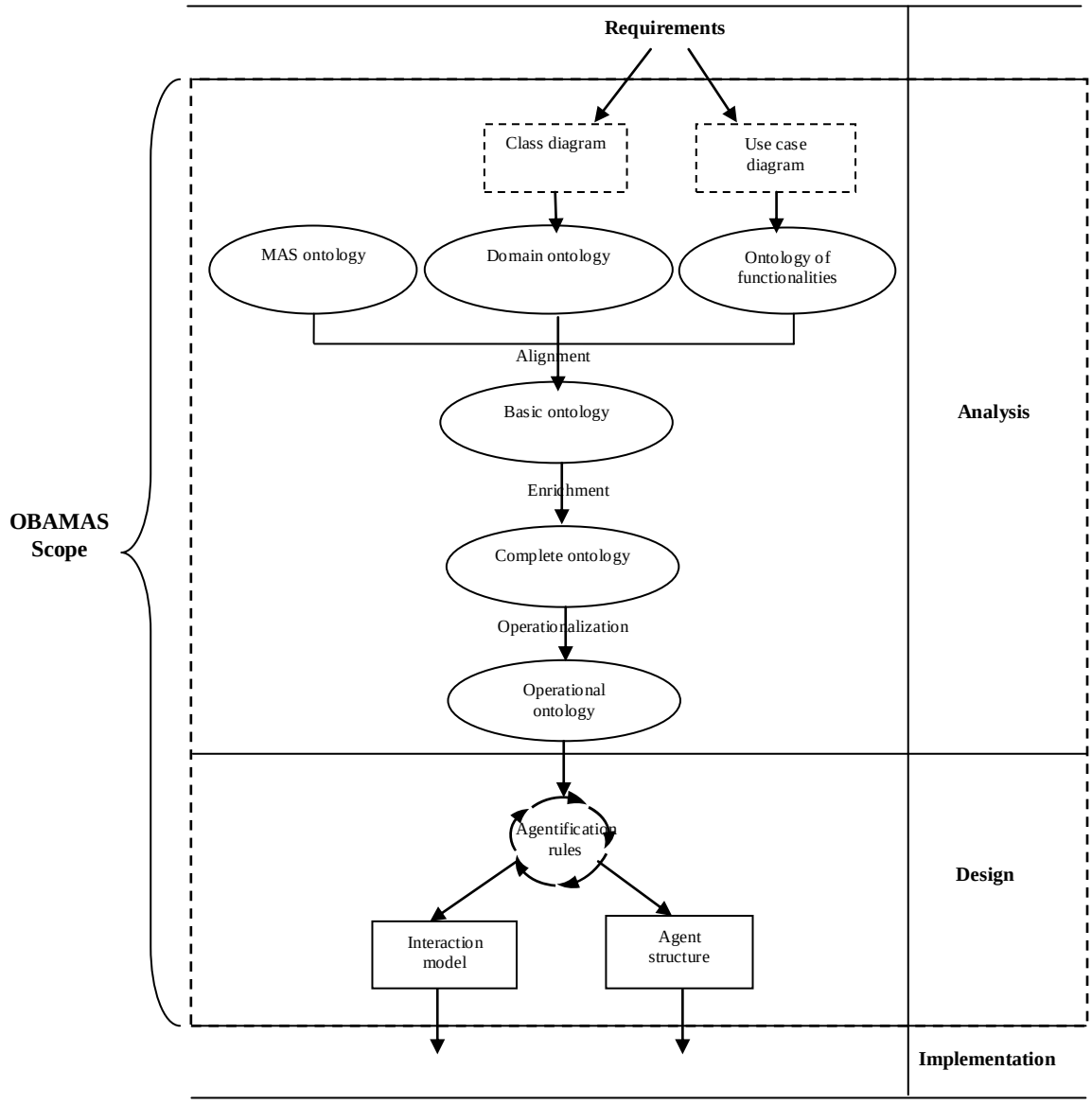

Figure 1. OBAMAS phases and models

The outline of the OBAMAS process sketches four (04) main steps covering the analysis and the design of the MAS:

The *Analysis phase* covers the three first steps which consist to :

i. Produce a *basic ontology* (noted O_B) obtained from the alignment and the merging of three ontologies: a *domain ontology* (noted O_A), an *ontology of functionalities* (noted O_f) which formally describes the functionalities of the system, and a *MAS ontology* (noted O_{MAS}). We represent this first step by the following formula: $O_B = \sum_{Align}(O_A, O_f, O_{MAS})$, where $\sum_{Align}$ is the alignment operation used to merge the three ontologies.

ii. Derive a *complete ontology* (noted O_C) by the enrichment of the basic ontology. The enrichment consists of adding axioms involving concepts appearing in the three first ontologies. We represent

this step by the formula: $O_C = Enrich(O_B)$, where *Enrich* is the operation of enrichment.

iii. Operationalize the basic ontology by giving a representation allowing its use according to the operational objective aimed at defining agents. This is done by defining a usage scenario as in [14]: we obtain an Operational Ontology (noted O_{op}). This step is characterized by the formula: $O_{op} = Oper(O_C)$, where *Oper* is the operation of operationalization.

The *Design phase* covers the last step which consists to:

iv. Operate the operational ontology to obtain the structure of the Agents and the interaction model.

To build the first three ontologies, we use some UML diagrams (class and use case diagrams). UML class diagrams already represent ontologies but, to be processed by computers, an ontology must be represented in an adequate formalism. Here we use the *ALCQI* description logics (DL) language in the form of a TBOX. Let us note that DL structure KB in two levels: the terminological level (TBOX) and the assertion level (ABOX). The TBOX contains a set of concepts and concepts descriptions formulated through the equivalence (noted ≡), the subsumption (relation of inclusion between two concepts, noted ⊑) and other constructors (∀, ∃, ≤, ≥, etc.) provided by DL languages. The interested reader can refer to the chapter 1 of [12] for a good description of DL, TBOXes, and ABOXes.

1. *The Analysis phase*

a. *Construction of the basic ontology*

The modelling of MAS goes through the modelling of concepts, goals, roles and interactions between objects. The constituents of the basic ontology are intended to describe these entities.

1) *The domain ontology*

To build the domain ontology, we suggest the encoding in *ALCQI* of a UML class diagram representing the conceptual model of the field using the transcription rules proposed in [17].

2) *The ontology of functionalities*

To build the ontology of functionalities of a system, we propose a method to translate a UML use case diagram in a TBOX formalized in *ALCQI*. UML use cases model is a good mean of functional modelling of a system. We propose to take into account *actors*, *use cases* and *relations*. An actor or a use case is represented by an atomic concept. The relations *participate* (between an actor and a use case), *include* (between use cases) and *extend* (between use cases or actors) are all represented by atomic roles (a role here is taken in the sense of DL). Table 1 summarizes our proposal for the *ALCQI* formalization of concepts and roles descriptions in the ontology of functionalities. In this table, *A*, A_1, and A_2 are actors, U, U_1 and U_2 are use cases.

TABLE I. SUMMARY OF THE ALCQI FORMALIZATION OF THE ONTOLOGY OF FUNCTIONALITIES

Expression	Translation in *ALCQI*
A_1 generalizes A_2	$A_2 \sqsubseteq A_1$
A participate to U	$A \sqsubseteq \exists participate \bullet U$
U_1 includes U_2	$U_1 \sqsubseteq \exists include \bullet U_2$
U_1 generalizes U_2	$U_1 \sqsubseteq U_2$
U_1 extends U_2	$U_1 \sqsubseteq \exists extend \bullet U_2$

3) *The multi-agents systems ontology*

The ontology of MAS describes the entities that should be part of MAS and their relations. This is done independently of the operational goal of any system, that's why it is built once and should be used for several MAS development. We were inspired by the diagram representing the concepts of the AOSE methodology called MESSAGE [5] (Fig. 2) and some well known requirements of MAS [1, 10].

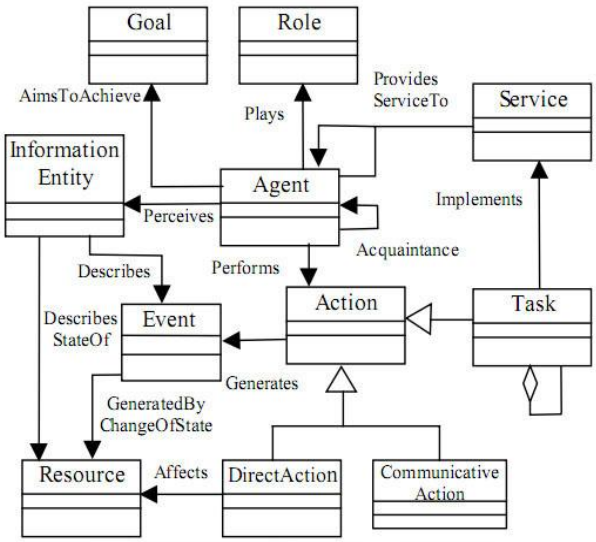

Figure 2. MESSAGE concepts [5]

The following are some partial concepts descriptions of the MAS ontology.

Agent⊑≥1provide•Service⊓≥1use•Resource⊓≥1play •Role⊓≥1execute•Action⊓∃present•Behaviour⊓≥1have•Believe⊓≥1have•Competence⊓≥1 acquaintance

Service ⊑ ≥1implementedBy •Task

Task ⊑Action

Action ≡ DirectAction ⊔ CommunicationAction

DirectAction ⊑ ≥ 1affect •Resource

⊤⊑ ∀Acquaintance •Agent ⊓∀acquaintance$^-$ •Agent

4) *The alignment of the three ontologies*

To use our three ontologies in the KBS intended to help in the definition of agents, we have to align them to

establish semantic links between entities in these ontologies. Once the correspondences are found, we merge the ontologies by adding these relations. Given that the obtained ontology is going to serve within the framework of a particular application, namely the identification of the agents, a phase of reengineering can follow to eliminate unwanted parts if any [13]. This step produces the *basic ontology.*

b. Construction of the complete ontology

The complete ontology is obtained by the enrichment of the basic ontology by adding axioms and axiom schemas.

Axiom schemas are expressed in *ALCQI*. Adding axiom schemas consists in decorating concepts and roles by specifying the types of the involved relations. It generally concerns reflexivity, anti-reflexivity, symmetry, anti-symmetry, transitivity, exclusivity, and incompatibility between relations or concepts, cardinality of some relations and signature of some relations. For example, the role "execute", whose signature is execute(Agent,Action), is anti-reflexive and anti-symmetric. This is expressed it by adding in the basic ontology the axiom schema:

$\top \sqsubseteq \forall execute \bullet Action \sqcap \forall execute^{-} \bullet Agent$

We formalize axioms and agentification rules in the First Order Logic (FOL) in order to use an inference engine (JESS for example) to automatically make logical deductions about knowledge. We add axioms and some formal rules which will be operationalized for agentification. In the following, for both axiom and rule, the term axiom will be used.

The operation of adding axioms is based on the functionalities of the system, as well as on the existence of agents and cooperation between agents. As in [3], use cases are compared to services (which are carried out by a set of tasks), links between actors and use cases represent interactions between agents which perform tasks. We identify a certain number of axioms using the concepts already defined in the basic ontology. It's important to note that these axioms are not specific to a particular system; they are independent of any system and then, can be reused for all MAS developed by OBAMAS. We give here a non exhaustive list of some of these axioms to add in the basic ontology to obtain the complete ontology:

For each service of the system, there is an agent which provides it.

$$\forall s\ Service(s) \rightarrow \exists a\ Agent(a) \wedge provide(a,s) \quad \text{(A1)}$$

If an actor a_1 generalizes another actor a_2, then there is acquaintance between the agents representing these actors (because a_1 will query a_2 to carry out the services provided by a_2).

$$\forall a_1, a_2, Actor(a1) \wedge Actor(a2) \wedge generalize(a1,a2)) \rightarrow acquaintance(a1,a2) \quad \text{(A2)}$$

If an actor corresponds to a concept described in the domain ontology (e.g. the actor `A_Student` and the concept `Student`) then this actor (seen as an agent), uses the concept as a resource to achieve its tasks.

$$\forall a, c, Actor(a) \wedge Concept(c) \wedge correspond(a,c)) \rightarrow use(a,c) \quad \text{(A3)}$$

If a use case u1 "includes" a use case u2, then there is acquaintance between the agents which provide them (because the agent performing u1 will call the one providing u2 to realize a part of the service it wants to achieve).

$$\forall u1, u2, a1, a2, U(u1) \wedge U(u2) \wedge Agent(a1) \wedge Agent(a2) \wedge include(u1,u2) \wedge provide(a1,u1) \wedge provide(a2,u2) \rightarrow acquaintance(a1,a2) \quad \text{(A4)}$$

If an agent cannot perform a given task related to some service, it can delegate the task to another agent with whom it has acquaintance.

$$\forall a1, t, Agent(a1) \wedge Task(t) \wedge (\neg competence(a1,t)) \rightarrow \exists a2, Agent(a2) \wedge callable(a2,t) \wedge acquaintance(a1,a2) \quad \text{(A5)}$$

If two actors take part in the same use case, then the two agents representing them can collaborate (this will depends on the semantics of this use case.)

$$\forall u, a1, a2, U(u) \wedge Agent(a1) \wedge Agent(a2) \wedge participate(a1,u) \wedge participate(a2,u) \rightarrow acquaintance(a1,a2) \quad \text{(A6)}$$

If some agents want to have an exclusive access to the same resource, the presence of another agent to manage the share is needed.

$$\forall a_1 \ldots a_n \bigwedge_{i=1}^{n} Agent(a_i) \wedge Resource(r) \bigwedge_{i=1}^{n} use(a_i,r) \Rightarrow \exists ag\ Agent(ag) \wedge \text{shareResource}\ (ag, r, a_1 \ldots a_n) \quad \text{(A7)}$$

If a use case u1 is not directly connected to any actor but extends another use case u2 which is a service provided by an agent a2 (for example "to suggest a rereading of a course" which extends "to make an evaluation"), then an agent a1 is created to provide the service u1 and a relation between a1 and a2 is established to make them collaborate.

$$\forall u1,u2,a1,a\ U(u1) \wedge U(u2) \wedge Agent(a1) \wedge Agent(a) \wedge participate(a1, u1) \wedge extends(u1,u2) \wedge \neg participate(a, u2) \rightarrow \exists a2\ Agent(a2) \wedge participate(a2, u2) \wedge acquaintance(a1,a2) \quad \text{(A8)}$$

If a use case u1 "extends" another one u2, then there is acquaintance between the two agents providing them (because the agent a2 can request for a1 during its execution).

$$\forall u1,u2,a1,a2,\ U(u1) \wedge U(u2) \wedge Agent(a1) \wedge Agent(a2) \wedge participate(a1, u1) \wedge participate(a2, u2) \wedge extends(u1,u2) \rightarrow acquaintance(a2,a1) \quad \text{(A9)}$$

If a use case *u* is not connected to any actor, but is included (via the relation "include") in others use cases u1, …,u_n, then it is an agent which provides the service corresponding to *u*. The agent providing *u* will be requested by the agents providing u1,…,u_n when necessary (this kind of use case is used only to factorize a common behaviour to several use cases).

$$\forall u1,\ldots,u_n,a,a1,\ldots a_n,\ \exists u\ U(u) \bigwedge_{i=1}^{n} (U(u_i)) \wedge Agent(a) \bigwedge_{i=1}^{n} (Agent(a_i)) \bigwedge_{i=1}^{n} (include(u_i,u)) \bigwedge_{i=1}^{n} (participate(a_i,u_i)) \wedge \neg(participate(a,u) \rightarrow \exists ag\ Agent(ag) \wedge participate(ag,u) \bigwedge_{i=1}^{n} acquaintance(a_i,ag) \quad \text{(A10)}$$

These axioms add some links between concepts of the three first ontologies. The obtained enriched basic ontology represents what we called the *complete ontology*.

c. Construction of the operational ontology

In DL, most of the axiom schemas are already represented by operational forms, i.e. forms with operational semantics which define the way in which these properties are used for reasoning. Precisely, the operational semantics of certain properties is determined by principal reasoning mechanisms of DL: *subsumption* (to check whether a concept is more general than another) and *instance checking* (to check whether an individual is an instance of a concept). *Signature of relations* (to check the domain and the range of relations) can also be used.

- The *subsumption relation* is used in an inferential context since it makes it possible to deduce all the possible types of an instance starting from its specified type. For example, if there is a subsumption relation `C` ⊑ `D` in the ontology, then the system knows that each instance of `C` is an instance of `D` and this could thus be used in the various reasoning mechanisms.
- *Instance checking* is used in a validation context, since the presence of an assertion `C(a)` (stating that `a` is an instance of `C`) will raise an error if `a` is not conform with respect to the definition of `C`. This assertion won't be used to infer any information about `a`.
- The *signature of a relation* is also used to validate, since only a violation of this property will influence the reasoning: the presence of a relation does not imply the types of the concepts involved. Thus, linking an instance of a concept with a relation whose signature doesn't contain this concept will raise an error. For example, the presence of the relation `acquaintance(a,b)` will not be used to infer that `a` and `b` are agents (since the signature of this relation is `acquaintance(Agent,Agent)` but only to check whether they are agents, if not, an error must be raised.

The other axioms (such as the symmetry of the acquaintance relation: *a1, a2 Agent(a1) Agent(a2) acquaintance (a1,a2)→ acquaintance (a2,a1)*) must be operationalized using reasoning primitives which are:

- *positive constraints* (if the assumption is present, then the conclusion must also be present)
- *negative constraints* (if the assumption is present, then the conclusion must be absent)
- *inference rules* (if the assumption is present, then the conclusion must be added).

To operationalize the complete ontology, we identify three (03) types of axioms. Each type gives place to a different operationalization form according to the chosen context of use.

In the following lines, H indicates a conjunction of concepts or of relations, r and $r_{i,\ i=1,\ \ldots,m}$ are relations, R a conjunction of relations, and $C_{i,\ i=1,\ \ldots,p}$ are concepts, x and $x_{i,\ i=1,\ \ldots,n}$ are instances of concepts. Let *iic* (resp. *eic*) the *implicit* (resp. *explicit*) *inferential context*, and *icv* (resp. *ecv*) the *implicit* (resp. *explicit*) *context of validation.*

axioms of type 1 :

$\forall x_1, \ldots, x_n H \rightarrow \exists y_1, \ldots, y_p C_1(y_1) \wedge \ldots \wedge C_p(y_p) \wedge R$

For example, the axiom A1:

$\forall s\ Service(s) \rightarrow \exists\ a\ Agent(a) \wedge provide(a,s)$

The operationalization is as follow:

- *iic* (resp. *eic*): First, we add an implicit (resp. explicit) rule $\forall x_1,\ldots,x_n H \rightarrow action1(\exists y_1,\ldots,y_p C_1(y_1) \wedge \ldots \wedge C_p(y_p) \wedge R)$**;**

  action1 checks, as soon as this hypothesis is present, the existence of instances y_i and relations in *R*; if they do not exist, it infers these instances. Furthermore we add *p* negative implicit constraints: $\forall x_1,\ldots,x_n\ H \wedge C_i(y) \rightarrow action2(C_j(y))$ where C_i and C_j are disjoint concepts and action2 prevents the presence of an instance *y* belonging to both C_i and C_j.

  icv or **ecv**: the axiom is operationalized in the form $\forall x_1,\ldots,x_n H \rightarrow action3(\exists y_1,\ldots,y_p C_1(y_1) \wedge \ldots \wedge C_p(y_p) \wedge R$ where action3 checks that instances y_i exists, belongs to concepts C_i, doesn't belong to any instance of incompatible concepts and finally that the signatures of relations in the conjunction *R* are met.

Axioms of type 2:

$\forall x_1, \ldots, x_n\ H \rightarrow r_1(..) \wedge \ldots \wedge r_m(..)$

For instance, the symmetry of the acquaintance relation between agents is formalized in FOL as $\forall a1,a2$ *Agent(a1) Agent(a2) acquaintance(a1,a2)*$\rightarrow$ *acquaintance(a2,a1)*

The operationalization is as follow:

- *iic*: Addition of an implicit rule $\forall x_1, \ldots, x_n$ $H \rightarrow action1(r_1(..) \wedge ... \wedge r_m(..))$. The action verifies in the KB the existence of the relations r_i as soon as the hypothesis is present. In case there is no relation, they are inferred and added.
- *eic*: Addition of an explicit rule $\forall x_1, \ldots, x_n$ $H \rightarrow action2(r_1(..) \wedge ... \wedge r_m(..))$; the action is defined as in **iic** and the addition of *m* implicit negative constraints: $\forall x_1,\ldots,x_n H(\wedge r_i(..))_{i=1..m} \rightarrow action3(r'_j(..))_{j=1,...m,i\neq j}$ where r'_j are exclusive relations of r_i. Action3 defines constraints as soon as the hypothesis and the relations r_i are present so that no exclusive relation of r_i is present. If no exclusive relation of r_i is present in the ontology, the constraint is replaced by *m'* implicit negative constraints: $\forall x_1,\ldots,x_n$ $H(\wedge r_i(..))_{i=1..m,i\neq j} \rightarrow action3(r'_{jk}(..))_{k=1,...m'}$ where r'_{jk} are all relations incompatible with r_i. These constraints exclude the presence of the incompatible relations of r_i as soon as the hypothesis and the relations r_i are present.
- *icv* (resp. **ecv**): The axiom is operationalized in the form of *m* implicit negative constraints: $\forall x_1,\ldots,x_n H(\wedge r_i(..))_{i=1..m} \rightarrow action3(r'_j(..))_{j=1,...m,i\neq j}$ where r'_j are relations exclusive to r_i. If no exclusive relation of r_i is present in the ontology, the constraint is replaced by *m'* negative implicit constraints:

  $\forall x_1,\ldots,x_n H(\wedge r_i(..))_{i=1..m,i\neq j} \rightarrow action3(r'_{jk}(..))_{k=1,...m'}$

  where r'_{jk} are all relations incompatible with r_i.

Axioms of type 3:

$\forall x_1, \ldots, x_n$ $H \rightarrow \neg(r_1(..) \wedge ... \wedge r_m(..))$

For example, the anti-symmetry of the inclusion between use cases which is formalized as: $\forall u1,u2$ *U(u1) U(u2) include(u1,u2)* $\rightarrow \neg$*include(u2,u1).*

The operationalization is as follow:

- *iic* : The axiom is operationalized in the form of *m* implicit rules: $\forall x_1,\ldots,x_n H(\wedge r_i(..))_{i=1..m} \rightarrow action3(r'_j(..))_{j=1,...m,i\neq j}$ where r'_j are exclusive relations of r_i. If no exclusive relation of r_i is present in the ontology, the corresponding rule is replaced by *m'* implicit rules: $\forall x_1,\ldots,x_n H(\wedge r_i(..))_{i=1..m,i\neq j} \rightarrow action3(r'_{jk}(..))_{k=1,...m'}$ where r'_{jk} are relations which are incompatibles with r_i. Action3 is defined as in **eic** of axioms of type 2.
- *cie* : As in the **iic**, except that the *m* and *m'* rules are explicit. In addition, we add a negative constraint $\forall x_1,\ldots,x_n H \rightarrow action4(\neg(r_1(..) \wedge ... \wedge r_m(..)))$
- *icv* or *ecv*: addition of a negative constraint $\forall x_1,\ldots,x_n$ $H \rightarrow action4(\neg(r_1(..) \wedge ... \wedge r_m(..)))$

  Action4 checks that there is no relation between some instances of concepts once the hypothesis is verified.

2. The design phase

The purpose of the design phase is to operate the operational ontology to derive the *structure of the agents* of the system (their nature, their associated services, their resources, etc.) and the *interaction model* (exchange protocols between the agents). The mechanism for identifying agents, resources and collaboration relations between agents consists of a sequence of rules based on the concepts of the operational ontology. Except for few of them, most of the axioms will be operationalized in an explicit inferential context because it's the designer of the MAS who triggers the rules to infer agents, assign resources, and find cooperative relations existing between them.

The axiom schemas are already in an operational form in the complete ontology (in particular, the subsumption relations from the three first ontologies) and allow identifying the system functionalities and all the actors involved. Due to the relations created by the alignment of the first ontologies, functionalities are assimilated to services provided by the system, actors becoming agents providing these services. Then the sequence of rules of existence such as A1, A5, A7, A8 and A10 identify the other agents of the system.

The phase of resource allocation follows. According to the ontology of MAS, each agent must have access to resources to perform tasks implementing services. These tasks are in fact actions performed by agents, either to communicate with other agents or to handle resources. The concepts defined in the domain ontology are resources and thus are assigned to each agent, depending on the role played in the system (i.e. based on the services it provides). Operating an axiom such as A3, operationalized in an inferential context, can also allocate resources to agents.

B. Case study : a distance learning system

To illustrate our approach, we design a "minimum" multi-agents platform for distance education. Our choice is motivated by the fact that this is an open and dynamic system because when deploying the system, we do not know how many students will enrol, or when they will do it, yet they interact strongly with other components of the system and also interact with each other. We first give some requirements of this platform.

a. Requirements of our distance learning system

The pedagogy of the distance learning system (DLS) is led by a director and a board that includes the educational council, site managers, and representatives of teachers and of students. The DLS allows students to enrol and attend courses at distance. Among the actors

of the DLS there are students seeking courses on the platform, requesting registration for these courses. Once registrations are validated, students seek documentation, participate in forum discussions, take exams …etc. Tutors monitor students, answer questions, give advices. Teachers may do any of tutors' tasks but specifically, they produce educational materials for courses and evaluate students. Site managers add new courses in the list of available courses; manage enrolment and student courses of study. The list of available courses is described in a catalogue.

A course is characterized by a unique identification number, a title, contents, and a fee. A given exercise is associated to a course. It is characterized by a difficulty level, a list of concepts pre-requisite, and a solution. Each course must be structured and follow the decomposition: *Chapter - Paragraph - Notion*. For each course in which a student is enrolled, he may send questions by email to the teacher responsible for the course. Each exam takes place in a limited time on the platform with the presence of the course instructor who can answer questions for better understanding.

b. Deploying OBAMAS for the case study

Following the OBAMAS methodology, we begin with the analysis phase. First of all, we build the three first ontologies to be aligned to constitute the basic ontology.

The domain ontology

To build the domain ontology, we proceeded by the encoding in *ALCQI* of an UML class diagram representing the conceptual model of the field, using the transcription rules proposed in [17]. The following are some partial concepts descriptions:

Student $\sqsubseteq \exists$id $\sqcap \leq$1id $\sqcap \exists$email $\sqcap \geq$1registered^{-} $\bullet$Registration

Module $\sqsubseteq \exists$code $\sqcap \leq$1code $\sqcap \exists$date $\sqcap \leq$1date $\sqcap \geq$1concern$^{-}\bullet$Registration

Registration $\sqsubseteq \exists$registered $\sqcap \leq$1registered $\sqcap \exists$concern $\sqcap \leq$1concern

$\top \sqsubseteq \forall$contain $\bullet$Subject $\sqcap \forall$contain$^{-}\bullet$Module

Subject $\sqsubseteq \geq$1code $\sqcap \geq$1contain$^{-}\sqcap \geq$1support$\bullet$PedagogicDoc

CourseMaterial $\sqsubseteq$ PedagogicDoc $\sqcap \geq$1contain2

$\top \sqsubseteq \forall$contain2$\bullet$Chapter $\sqcap \forall$contain2$^{-}\bullet$CourseMaterial

Chapter $\sqsubseteq \exists$title $\sqcap \leq$1title $\sqcap \geq$1contain2$^{-}\sqcap \geq$1contain3

Paragraph $\sqsubseteq \exists$title $\sqcap \leq$1title $\sqcap \geq$1contain3$^{-}\sqcap \geq$1describe$\bullet$Notion

$\top \sqsubseteq \forall$contain3$\bullet$Paragraph $\sqcap \forall$contain3$^{-}\bullet$Chapter

TutorialsSheet $\sqsubseteq$ PedagogicDoc $\sqcap \geq$1contain4

Exercise $\sqsubseteq \exists$number $\sqcap \leq$1number $\geq$1contain4$^{-}\sqcap \geq$1dealWith$\bullet$Notion

$\top \sqsubseteq \forall$contain4$\bullet$Exercise $\sqcap \forall$contain4$^{-}\bullet$TutorialsSheet

Notion $\sqsubseteq \exists$keyword $\sqcap \leq$1keyword $\sqcap \geq$1prerequisite

$\top \sqsubseteq \forall$prerequisite$\bullet$Notion

The ontology of functionalities

For the case of the distance education, among others, we can identify the use cases "Request registration", "Attend courses", "Research of course materials", "Take part in Evaluation", "Monitor students", "Manage course". Some actors are "Student", "Teacher", and "Site manager". Fig.3 shows a part of the use case diagram of the system.

Copyright © 2013 MECS

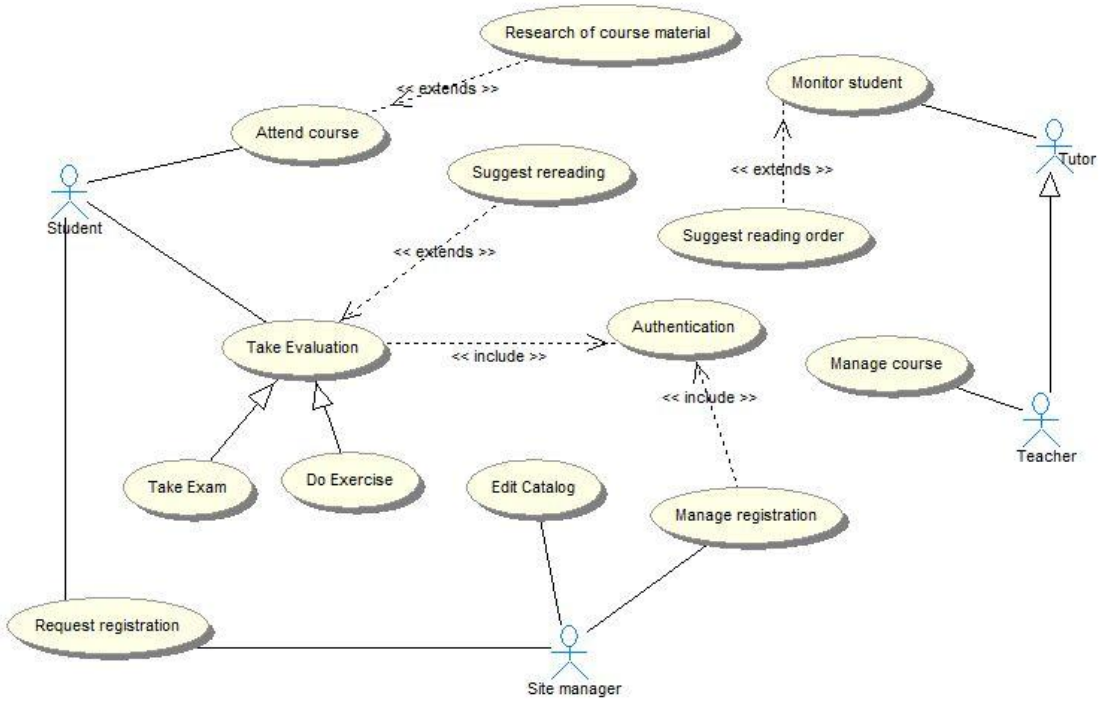

Figure 3. The partial use case diagram of the DLS

For readability, let the names of concepts representing the actors start with the letter A, while those of use cases begin with the letter U. Some of the obtained concept descriptions are:

A_Student ⊔ A_Teacher ⊔ A_Webmaster ⊑ Actor

A_Student ⊑
(∃participate•U_RequestRegistration)⊓(∃ participate •U_AttendCourse)⊓ (∃participate •U_Research)⊓(∃participate•U_TakeEvaluation)

A_Teacher ⊑ (∃participate•U_ManageCourse) ⊓ (∃participate•U_ManageStudent) ⊓ (∃participate•U_Research)

U_Research ⊑ ∃include•U_Authentication

U_AttendCourse ⊑ ∃extends•U_TakeEvaluation

U_TakeExam ⊔ U_MakeExercise ⊑ U_TakeEvaluation

The ontology of MAS

Since this ontology doesn't change as a function of the MAS to develop, it was already built when exposing the analysis phase of OBAMAS.

The three ontologies built are then aligned. We tested pairs of concepts and roles of the three ontologies via equivalence and subsumption relations. Some of the correspondences found are the following:

U_RequestRegistration ⊑ Service (1)

Student ⊑ Resource (2)

provide ⊑ participate (3)

Actor ⊑ Agent (4)

(1) Correspondence between a concept of the ontology of functionalities and a concept of the ontology of MAS. It was obtained by observing that every use case is a service provided by the system.

(2) Correspondence between a concept of the domain ontology and a concept of the ontology of MAS. It was obtained by observing that any concept can be defined as a resource, since each concept contains information needed for future agents to achieve their tasks.

(3) Correspondence between a role of the ontology of MAS and a role of the ontology of functionalities. It was obtained by observing that an agent who provides a service participates in a use case.

(4) This correspondence is deduced from the previous three. Indeed, each actor plays a role whose functionalities are realized by the use cases in which it participates.

Once the matching is done, we merge ontologies by adding these correspondences as new relations to obtain the **basic** ontology.

To enrich the basic ontology, we introduce in the first ontologies some axiom schema précising the signature of some relations such as

⊤ ⊑ ∀contain•Subject ⊓ ∀contain-•Module

⊤ ⊑ ∀acquaintance•Agent ⊓ ∀acquaintance-•Agent

which formally states that the signatures of the roles contain and acquaintance are respectively

Copyright © 2013 MECS

contain(Module,Subject) and acquaintance (Agent,Agent).

For the enrichment, we also add characteristics of some relations:

∀a1, a2 Agent(a1) Agent(a2) acquaintance(a1,a2) → acquaintance(a2,a1) (Symmetry of the acquaintance relation)

DirectAction ⊓ CommunicationAction ⊑ ⊥ (Disjointness of Direct actions and communication actions performed by agents)

We noticed that the enrichment of the basic ontology by axioms doesn't depend on the system to build. We just add those proposed in the section dedicated to the analysis that is A1, A2, A3 …etc. to have the complete ontology.

Finally, we just have to design our MAS by operating the operational ontology. The design phase consists to define the agent structure and the interaction model.

Agent structure.

For our distance education platform, we start by identifying the first agents, actors of the system.

StudentAgent plays the role of a *Student* within the system. It requests inscriptions, accesses course materials, carries out information retrieval, sends mails to teachers, to other students, to webmasters, and then receives answers. It also takes part in discussion forums and evaluations online. To request an inscription, it needs the use of the resource Student (a concept defined in the domain ontology by attributes such as *Name, LastDegree, email*). The axiom A3, operationalized in an inferential context affects this concept as a resource for this agent.

TutorAgent plays the role of *Tutor* for the students. It ensures the monitoring of students and the answers to their questions. This agent has specific properties in the sense that it does not intervene only when a human tutor is connected to the platform but, it carries out some automatic tasks such as the suggestion of an order of reading course materials (the use case "U_ReadingOrder"), or the suggestion of a second reading of some specific parts of a course material according to the marks obtained during evaluations (the use case "U_RereadingSuggestion"). More than the other agents, it must be cognitive.

TeacherAgent plays the role of *Teacher* within the system. It designs and deposits teaching material (related to the courses for which it is responsible) on the platform and evaluates the students.

SiteManagerAgent plays the role of *Webmaster*. It publishes the course catalogue, validates the registrations requested by students, and exchanges mails with them.

In addition, the implementation of the operational forms of the axioms introduced at the time of the step of enrichment will make it possible to deduce the existence of other agents and the associated resources.

Thus, the operationalization of axiom A7 in *iic* or *eic* permits to infer the agent *ResourceCoordinatorAgent* which plays the role of *agent coordinator* with respect to the access to the resources in order to avoid conflicts.

The axiom A8 operationalized in *iic* or *eic* permits to infer the agents:

MailerAgent plays the role of *mail manager* on the platform. It is charged to deal with mail sending and reception, to signal the presence of new mails, to manage the messages on discussion forums …etc.

SearcherAgent provides the search service on the platform. This agent will be called for information retrieval.

It is necessary to note that if the agent *TutorAgent* did not exist yet (i.e. if there were not any human actor being able to play the role of Tutor); he would have been created by A8. In which case it would have been a purely virtual agent using only ontology-based reasoning mechanisms to render his services.

The axiom A10 operationalized in *iic* or *eic* permits to infer the agent:

AuthenticationAgent which ensures the role of *service authentication* on the platform. It is used every time another agent requests authentication in order to take actions.

The axiom A1 which is of *type 1* is operationalized, according to the contexts, in the following way:

iic (resp. *eic*): addition of an implicit (resp. explicit) rule
∀sService(s)→action1(∃aAgent(a) provide(a,s)).
The action checks for any service the existence of at least an agent which provides it, if no agent exists, it infers it.

cvi or *cve*: Addition of a constraint ∀s Service(s) → action3(∃a Agent(a) provide(a,s)). Action3 checks that agent a exists and that it renders the services.

The interaction model

The interaction model describes the relations between agents i.e. the allowed sequences of messages between agents. In OBAMAS, the interaction model is generated by operating operational axioms such as A2, A4, A5, A6, A8, A9 and A10. These axioms operationalized in an inferential context allow inferring collaboration relations between the agents.

For our DLS, the axiom A5 for example, allows to create between SiteManagerAgent and Authentication Agent, a relation of collaboration, the first having to call the second when it needs to be authenticated for the execution of certain Tasks. Fig. 4 shows the collaboration relation existing between the agents of the DLS.

Copyright © 2013 MECS

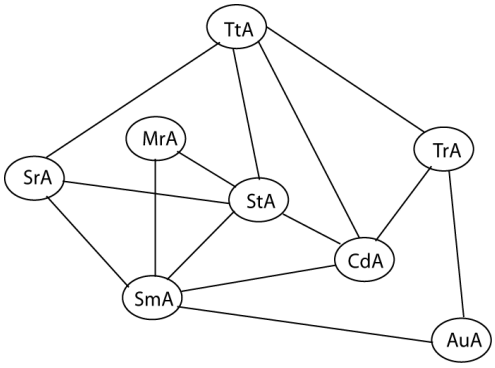

Figure 4. The interaction diagram

In Fig. 4, StA = StudentAgent, TtA = TutorAgent, TrA = TeacherAgent, SmA = SiteManagerAgent, CdA = CoordinatorAgent, MrA = MailerAgent, SrA = SearcherAgent, AuA = AuthenticationAgent.

The design step is made quasi automatically by triggering rules. However, the designer can refine the produced structures and make implementation choices.

V. Related works

Last decades witnessed a growing interest in MAS development techniques. In this section, we briefly present the main features of some works related to our approach to develop MAS from ontologies. We identified and investigated some AOSE methodologies which have been well cited in the literature and which integrate the use of ontologies: MAS-CommonKADS [6], MESSAGE [5], MASE [4,8] and PASSI [2]. Here, we restrict our presentation to their use of ontologies.

- *MAS-CommonKADS* (Multi-Agent System - Knowledge Analysis and Development System) [6] is based on the knowledge engineering methodology CommonKADS. This approach makes use of ontologies to model the application domain and the knowledge manipulated by each agent. However aspects of reusability and interoperability are neglected because MAS-CommonKADS does not refer to ontologies in terms of inter-agent communication. Finally, it is difficult to clearly justify the use of ontologies in this approach.
- *MESSAGE* (Methodology for Engineering Systems of Software Agent) [5] is a methodology that uses Rational Unified Process (RUP) and considers UML as a starting point at which it adds some concepts (such as role, goal, task …etc.) suitable for MAS modeling. It aims to gather the good features of existing AOSE. MESSAGE uses an ontology to model the application domain and to base the reasoning mechanisms for agents. However, as MAS-CommonKADS, it doesn't make any reference to the ontology in the communication between agents.
- *MASE* (Multi-Agent System Engineering) in the first version [8] makes no reference to the use of ontologies. In its extended version [4], ontologies are introduced to specify the application domain. The main goal here is to allow developers specify the information flow between agents, using the concepts defined in the ontology, to ensure that each agent has the information needed to accomplish its tasks. The main use of ontologies in MASE is made for communication. This means that tasks performed by other agents (such as reasoning processes) are not based on the ontology. This also shows a partial exploitation of the potential of ontologies.
- *PASSI* (a Process for Agent Societies Specification and Implementation) [2] is based on object oriented modeling techniques and artificial intelligence, and uses UML notations. PASSI models the application domain by an ontology which also serves in communication and reasoning processes for agents that are themselves identified by use cases of the system. PASSI In this sense goes beyond the first three in the use of ontologies.

OBAMAS (Ontology-Based Approach for Multi-Agent System Engineering) differs from these methods by the fact that it is entirely based on the use of formal ontologies. Indeed, OBAMAS advocates the use of ontologies from the modelling of the application domain, the functionalities of the future system and the field of MAS, to their operationalization to identify agents, resources and interactions between agents. Then, as in some of the methods described above, the complete ontology also serves as a common basis for communications to ensure that there is no ambiguity in the terminology. The reasoning mechanisms are also based on this ontology.

VI. Features and contributions of OBAMAS

OBAMAS includes the following features:

- *Flexibility*: OBAMAS advocates the use of UML class and use cases diagrams to build ontologies for the domain and functionalities. However this approach is not required, the developer is free to use any other methodology for building ontologies. In addition the method only covers the analysis and the design, leaving the developer free for implementation choices.
- *Reusability*: Components of the MAS uses the same ontology (the operational one) for interactions and reasoning processes. Moreover, the ontology of the MAS does not vary regardless of the application to set up, it is totally reusable. The only effort in another area will be to build the other two constituting the basic ontology. In addition, the proposed axioms are not dependent on any domain or any particular application. Once these rules are validated, they remain valid and usable for the development of MAS in the frameworks of several other applications.

Copyright © 2013 MECS

- *iterative* and *incremental:* OBAMAS is defined so that we can start with the specifications of a minimum initial requirements, build a partial domain ontology describing only the relevant concepts, a corresponding ontology of functionalities and proceed to the identification of agents involved. We can then add new requirements and restart the cycle.
- The process can be *automated*: Due to the use of formal ontologies in the analysis phase, OBAMAS allows automatic identification of agents via a trigger mechanism for deduction rules. In this first application, the operationalization of our ontology in order to deduce the complete structure of agents and the interaction model is done by hand to check carefully and partially validate the proposed mechanism. In a more practical way, OBAMAS can be automated by integrating JADE (a java-based MAS development platform), JENA (a java API to manipulate ontologies) and JESS (a Java programming environment for systems based on logical rules). JESS and JENA will be integrated to work simultaneously as the rules used by JESS may sometimes involve concepts and relations defined in the operational ontology.

VII. Conclusions

The development of a MAS (distance learning system) entirely based on the use of ontologies shows that the approach is interesting and promising because it formalizes MAS development process, especially the identification of agents and their interactions. Ontologies are initially used for agents identification, and then they are used by agents to realize their various tasks, and for communication. OBAMAS has the merit of benefiting all the advantages of the use of ontologies in the context of MAS, namely interactions, reusability, activities of MAS development and MAS operations.

Other applications of OBAMAS remain necessary for its enrichment and its validation. In addition, the complete ontology could be enriched by the use of UML sequence and collaboration diagrams in order to better describe functionalities and the interaction protocols between agents i.e., the allowed sequences of messages between agents and the constraints on the contents of these messages. Moreover, it would be quite interesting to see whether OBAMAS can be coupled with another well-known method such as MASE in order to share the advantages of each of them.

References

[1] M. Wooldridge, *An Introduction to MultiAgent Systems*, Chichester: John Wiley & Sons Ltd, 2002. ISBN 0-471-49691-X

[2] P. Burrafato, M. Cossentino, "Designing a multi-agent solution for a bookstore with the PASSI methodology", *in Proceedings of the 4th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002)*, Toronto, Canada, 2002.

[3] A. Chella, M. Cossentino, U. Lo Faso, "Applying UML Use Case Diagrams to Agents Representation", *Convegno AI*IA 2000, Milano, 2000*.

[4] J. DiLeo, T. Jacobs, S. DeLoach. "Integrating Ontologies into Multiagent Systems Engineering", *In Proceedings of the 4th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002)*, Bologna, Italy, 2002.

[5] Eurescom, "MESSAGE: Methodology for Engineering Systems of Software Agents", Technical information, Broadcom Eireann Research Ltd., September 2001.

[6] C.A. Iglesias, M. Garijo, J.C. Gonzalès, J.R. Velasco, "Analysis and Design of Multi-Agent Systems using MAS-CommonKADS", *In Intelligent Agents IV (LNAI Volume 1365), ed. M.P. Singh, A. Rao, and M. Wooldridge*, 313-326. Berlin: Springer-Verlag, 1998.

[7] Falasconi, S., G. Lanzola, and M. Stefanelli. Using Ontologies in Multi-Agent Systems. In Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'96), Banff, Canada, 1996.

[8] M. Wood, S.A. DeLoach, "An Overview of the Multiagent Systems Engineering Methodology", *in Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering (AOSE-2000)*, Limerick, Ireland, 207-221, 2000.

[9] A. Ayimdji, S. Koussoubé, L.P. Fotso, Développement de systèmes multi agents à partir d'ontologies, *In Proceedings of Colloque Africain sur la Recherche en Informatique et Mathématiques appliquées (CARI'10)*, pp. 575-582, Yamoussoukro, Côte d'Ivoire, Octobre 2010.

[10] M. Wooldridge, N. R. Jennings, Intelligent agents: theory and practice, *The knowledge Engineering Review, vol. 10(2)*, pp. 115-152, 1995.

[11] M. Uschold, M. Grüninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review, Volume 11 Number 2*, pp. 93-155, 1996.

[12] F. Baader et al., *The description logic handbook: Theory, implementation and applications*, Cambridge University Press, Cambridge, UK, 2003. ISBN 0-521-78176-0

[13] J. Euzenat, P. Shvaiko, *Ontology matching*, Heidelberg, DE, Springer-Verlag, 2007. ISBN 978-3-540-49611-3

[14] 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.

[15] M. Gruninger, M. S. Fox, "Methodology for the Design and Evaluation of Ontologies", *Proc. of Workshop on Basic Ontological Issues in Knowledge Sharing in IJCAI 95*, Montreal, Canada, 1995.

[16] S. Staab, A. Maedche, "Axioms are objects too: Ontology engineering beyond the modelling of concepts and relations", Research report 399, Institute AIFB, Karlsruhe, 2000.

[17] 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.

Dr. Souleymane KOUSSOUBE, holds a Ph.D in Artificial Intelligence from the University Paul Sabatier of Toulouse (France) with a Master of science Degree in Database and System integration from the University of Nice/CERAM (France). He is a Lecturer and the Director General of Institut Africain d'Informatique of Libreville (Gabon) and his research directions include: knowledge representation, ontologies, Business intelligence, Dataming, Machine learning, computational intelligence and multi-agents systems.

Armel AYIMDJI is a Lecturer at the department of computer engineering of the Douala University Institute of Technology (Cameroon). He will defend his Ph.D. thesis in Artificial Intelligence in few months at the University of Yaoundé I (Cameroon). His research directions include: knowledge representation, ontologies, computational intelligence, multi-agents systems, and software engineering.

Prof. FOTSO Laure Pauline holds a PhD in Operation Research & Statistics with Minor in Computer Science from the Rensselaer Polytechnic Institute of Troy New York with a Master of Science Degree in Computer Science from SUNY at Albany. Vice-Rector at the University of Dschang in Cameroon, her main research interests include: Combinatorial Optimization Algorithms, Multi-Agents System modeling, Multi-criteria Optimization, Scheduling, Markov Process Applications, Database Development and Implementation and Knowledge base system.

Copyright © 2013 MECS