Un protocole entre agents pour l'alignement d'ontologies

Alexandre Viollet

Mémoire de Master 2 Recherche

Mathématiques et Informatique

Spécialité Intelligence, Interaction, Information

Un protocole entre agents pour l'alignement d'ontologies

Alexandre VIOLLET

Responsables du projet : Jérôme Euzenat et Yves Demazeau

Examinateur : Danielle Ziébelin

Résumé

Les agents sont des programmes autonomes et communicants, conçus pour interagir avec d'autres agents. Dans notre cas les agents sont cognitifs : ils manipulent des connaissances ; ces connaissances sont représentées sous forme d'ontologies. Pour communiquer entre eux les agents s'échangent des messages en respectant un formalisme basé sur la théorie des actes de langage. Le langage de communication FIPA-ACL (le plus en vogue actuellement), permet également de spécifier l'ontologie utilisée pour exprimer le contenu du message.

Il n'existe cependant pas de formalisme universel propre au contenu sémantique d'un message. Lors d'un échange de messages il convient donc de mettre en rapport les deux ontologies différentes ; c'est pour cela qu'a été développée la notion d'alignement, à savoir un ensemble de couples pondérés d'éléments issus de deux ontologies.

Sur cette base, notre travail consiste à définir un protocole de communication entre agents qui leur permette de mettre en correspondance leurs ontologies pour pouvoir partager du sens. A cette fin nous intégrerons la notion d'alignement, ainsi que des ressources extérieures dont les agents auront besoin pour manipuler les alignements et optimiser leurs interactions.
Enfin, et afin de pouvoir s'intégrer de manière transparente dans un dialogue entre deux agents, le protocole se doit d'être modulaire.

Table des matières

Introduction

Ce mémoire a été réalisé principalement au sein de l'équipe EXMO de l'INRIA Rhône-Alpes (Institut National de Recherche en Informatique et en Automatique), et en collaboration avec l'équipe MAGMA du laboratoire Leibniz de l'institut IMAG. Le projet EXMO développe des outils théoriques pour aider à l'organisation, la manipulation, la composition et la présentation d'éléments de connaissance structurés. L'équipe MAGMA développe des études théoriques, des outils informatiques et des réalisations pratiques pour la simulation décentralisée de systèmes et la résolution distribuée de problèmes.

Ce projet s'inscrit dans le cadre des axes de recherche des deux équipes, de part ses dimensions liées à la manipulation de connaissances structurés et son application dans un système multi-agents.

Motivations et Objectifs

Un nombre grandissant de technologies font appel à des représentations de connaissances structurées de haut niveau pour manipuler les données dont elles disposent. Ces technologies, comme le web sémantique ou les systèmes multi-agents émergents, sont appelées à mettre en relation une multitude de ces sources de connaissances. C'est la communication entre les diverses entités de ces systèmes qui peut poser problème.

On s'intéresse plus précisément, dans notre étude, au cas de la communication entre agents logiciels. Ceux-ci manipulent des connaissances sous forme d'ontologies, notion que nous définirons ultérieurement. Lors de la communication entre deux de ces agents, des concepts issus des ontologies sont transmis au cœur même des messages ; certains de ces concepts, qui servent à exprimer une requête ou transmettre une information, sont inconnus au récepteur du message. En effet l'agent qui reçoit un message ne connaît pas forcément un certain concept, même s'il traite des connaissances du même domaine ; il peut en revanche connaître un concept qui est une spécialisation, ou une généralisation, du concept initial.

Il existe de nombreuses relations possibles entre des concepts représentés dans deux ontologies représentant des parties du même domaine de connaissances. Ces relations seront présentées au cours de notre étude.

Nous nous attaquerons à ce problème d'hétérogénéité sur la base de la notion d'alignement d'ontologies. Un alignement, que nous présenterons précisément plus loin, permet de mettre en relation des concepts à partir des relations qui existent entre eux.

Notre approche consiste à définir un protocole de communication entre agents qui permette de guider des alignements d'ontologies pour qu'un agent puisse finalement interpréter un message, équivalent à un message reçu, dont il connaît les concepts auxquels il se réfère.
Nous présentons d'abord dans cette étude les approches existantes pour faire face au problème de communication dans un système multi-agents, puis notre protocole et les services qui lui sont associés.

Plan

Ce mémoire est organisé en trois grandes parties :

- Les deux premiers chapitres abordent le contexte de notre travail, en présentant les moyens techniques existant ainsi que les principales approches relatives au problème que nous traitons. C'est ainsi que nous présenterons les notions de communication entre agents , de représentation et de manipulation des connaissances.

- Les trois chapitres suivants fournissent les éléments nécessaires à la conception du protocole de communication que nous considérons, ainsi que sa définition proprement dite.

- Le dernier chapitre permet d'envisager l'utilisation d'un tel protocole dans une situation réelle.

Chapitre 1 : Agents logiciels, connaissances et communication.
Nous présenterons tout d'abord le concept d'agent logiciel et les variations de ce type d'agent que l'on peut trouver dans la littérature. Nous ferons ensuite état de la notion d'ontologie pour représenter les connaissances. Nous présenterons enfin la communication entre agents, son lien avec les ontologies et le problème d'interopérabilité qui en découle.

Chapitre 2 : Quelques propositions pour le partage du sens entre agents.
Nous introduirons et analyserons les approches existantes pour traiter le problème dans les systèmes multi-agents.

Chapitre 3 : Comment rendre les messages intelligibles.
Nous présenterons dans cette partie les besoins identifiés pour le fonctionnement de notre protocole. Il s'agit de définir les structures et fonctions dont les agents doivent pouvoir disposer.

Chapitre 4 : Services d'alignement d'ontologies.
Nous présenterons en détail l'entité « Service d'alignement », nécessaire au fonctionnement du protocole, ainsi que les fonctions qui lui sont associées.

Chapitre 5 : Un protocole pour l'alignement d'ontologies.
Nous présenterons en détail le protocole de communication.

Chapitre 6 : Scénario d'utilisation.
Nous présenterons dans ce chapitre un scénario d'utilisation du protocole, dans un contexte concret.

Chapitre 1

Agents logiciels, connaissances et communication

Nous présentons dans ce chapitre les concepts à la base de notre étude, qu'ils soient d'ordres conceptuels ou techniques. Nous aborderons les notions d'agent logiciel, d'ontologie et, d'un point de vue plus technique, le modèle de communication entre agents promu par la FIPA [FIPA], ainsi que le langage OWL [W3C, 2004a] pour l'expression d'ontologies. Puis nous en arriverons au fond du problème traité par ce travail : l'interopérabilité.

1.1 Les Agents

Donnons tout d'abord une définition d'un agent. Un agent est, selon [FER95], une entité physique ou virtuelle, autonome, située dans un environnement, capable de le percevoir, capable d'y agir, de communiquer avec d'autres agents, et éventuellement de se reproduire. De plus, un agent possède un objectif individuel (fonction de satisfaction), des ressources, une représentation partielle de l'environnement, des compétences et il offre des services. Son comportement est fonction de ses observations, de ses connaissances, de ses croyances, de ses compétences, de ses interactions. L'autonomie d'un agent peut se définir par trois capacités :

> Les agents ont une existence propre, indépendante de l'existence des autres ;
>
> Les agents sont capables de maintenir leur viabilité dans des environnements dynamiques sans contrôle extérieur ;
>
> La prise de décision interne des agents quant au comportement à avoir est uniquement fonction des perceptions, connaissances et représentations du monde propre aux agents.

Nous considérerons donc un agent comme un programme ayant des propriétés particulières ; parmi ces propriétés signalons celles qui sont les plus significatives, à savoir l'autonomie, la communication avec d'autres programmes par l'intermédiaire d'envois de messages et le raisonnement à partir d'une base connaissances et d'événements extérieurs.
Nous présentons uniquement dans cette partie des concepts généraux permettant de distinguer différents types d'agents qui existent dans le monde des systèmes multi-agents.
Les agents peuvent être conçus pour différents types de systèmes caractérisés par les actions qui seront attendues des agents, leur niveau de raisonnement, leur sociabilité etc…

On distingue donc des agents cognitifs, intentionnels, rationnels, réactifs et bien d'autres encore.

De plus un agent peut faire parti de l'une des trois grandes familles suivantes ; il peut être *autonome*, *interagissant* ou *social*.

Définition (agent autonome) Un agent qui raisonne et agit/réagit avec l'environnement.

Définition (agent interagissant) Un agent qui a des capacités d'interaction avec les autres agents.

Définition (agent social) Un agent dont l'action s'inscrit dans un contexte social explicite.

Notons également qu'un grand nombre de types d'agents, du point de vue de l'aspect coordination, peuvent être obtenus à partir du modèle AEIO [Demazeau, 1995] qui définit 4 facettes intégrées à l'architecture de l'agent :

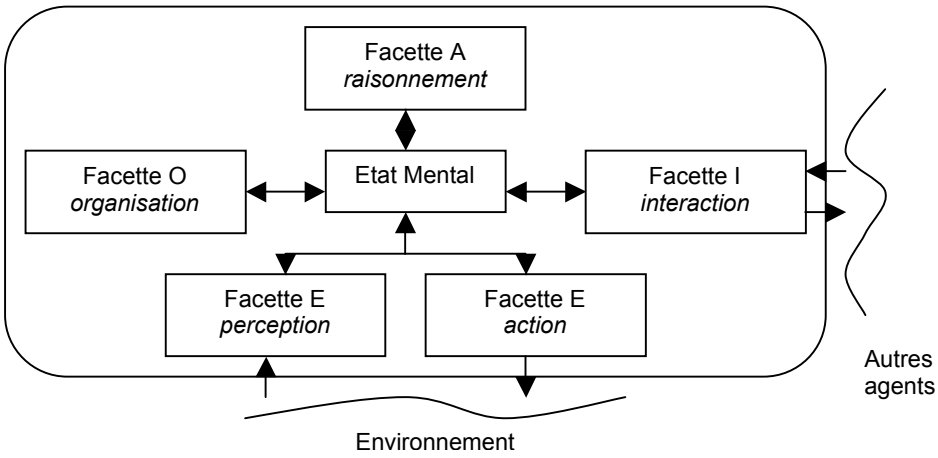


FIG 1. Facettes d'un agent, du point de vue de la coordination.

Sur le schéma précédent, la facette A désigne l'ensemble des fonctionnalités liées au raisonnement de l'agent. La facette E représente les capacités de perception et d'action de l'agent sur l'environnement. La facette I symbolise les capacités d'interaction de l'agent avec les autres agents, et la facette O concerne la gestion des agents entre eux, d'un point de vue organisationnel.

Nous définissons les agents de notre système comme des agents cognitifs (au sens de l'intelligence artificielle) par opposition aux agents réactifs qui ne suivent que le principe d'action / réaction (ou plus précisément celui d'évènement / réaction) induit de leur ancrage fort dans un environnement. Les agents que nous considérons sont aussi des agents qui interagissent entre eux par envois de messages (voir le paragraphe suivant).
Le fait que les agents soient cognitifs implique l'utilisation d'un langage élaboré qui puisse supporter la puissance de raisonnement mise en jeu. Leur nature communicative doit aussi être supportée par un modèle de communication efficace. Ce sont ces aspects que nous allons aborder dans les paragraphes suivants.

1.2 La communication entre agents

Comme nous l'avons présenté, la communication entre agents est donc réalisée par des échanges de messages, éventuellement englobés dans un protocole de communication qui définit une session de dialogue entre deux agents.

Afin de profiter d'une base solide pour la présentation du processus de communication, nous prenons comme exemple, tout au long de ce chapitre, le modèle promu par la FIPA [FIPA] : FIPA-ACL. Ce modèle, très largement répandu succède historiquement à KQML [KQML], l'unique alternative existante. Plus qu'étendre ce dernier en puissance d'expression, FIPA-ACL le simplifie également. Et de part la nature de la FIPA, dont le but est de promouvoir des standards dans le monde des systèmes à agents hétérogènes et coopérants, il est largement reconnu et supporté par la communauté d'individus et d'organismes manipulant des agents. Il définit, entre autre, le format des messages échangés entre agents, la notion d'annuaire et de pages jaunes.

Voici des exemples de messages, exprimés en FIPA-ACL. L'utilisation d'actes de langage comme *inform* ou *query-ref*, ainsi que les autres champs possibles, est détaillé dans les paragraphes suivants.

```
(INFORM
: SENDER (agent-identifier :name i)
: RECEIVER (agent-identifier :name j)
: CONTENT "appointment (today, INRIA_RA)"
: ONTOLOGY Scheduling
: LANGUAGE Prolog)
```

Exemple 1. l'agent i informe l'agent j d'un rendez-vous.
Le contenu, exprimé en Prolog, ce réfère à l'ontologie Scheduling.

Le message…

```
(QUERY-REF
 : SENDER
   (agent-identifier
     :name sender@paris.agentcities.org
     :addresses (sequence http://leap.crm-paris.com:5194/leap))
 : RECEIVER
   (agent-identifier
     :name aclping@paris.agentcities.org
     :addresses (sequence http://leap.crm-paris.com:5194/leap))
 : CONTENT ping)
```

```
doit entraîner une réponse du type…
(INFORM
 :SENDER
   (agent-identifier
     :name simple-ping@paris.agentcities.org
     :addresses (sequence http://leap.crm-paris.com:5194/leap))
 :RECEIVER
   (agent-identifier
     :name sender@paris.agentcities.org
     :addresses (sequence http://leap.crm-paris.com:5194/leap))
 :CONTENT  alive
 :LANGUAGE  PlainText)
```

Exemple 2. Exemple de communication tirée du projet AgentCities. L'objet de la communication est de faire un *ping* (Propriétaire du service : Motorola)

1.2.1 Les actes de langage

Nous présentons ici la théorie des actes de langage à la base des modèles de communication entre agents. Cette théorie a été fondée par Searle à partir des travaux d'Austin [Austin, 1962; Austin, 1970]. Austin participa notamment au développement de ce que l'on appelle communément la philosophie du langage ordinaire, et qui n'est autre que la pragmatique : la pragmatique s'intéresse à l'étude des comportements des individus en interaction.
Searle a donc rejeté et étendu certains aspects de cette théorie. Une distinction importante entre les deux théories est liée à la relation entre locution et illocution. En effet Searle laisse la valeur locutoire en dehors de son analyse de l'acte de langage ; et c'est ce qui permet principalement de considérer sa théorie comme une amélioration des travaux de Austin.

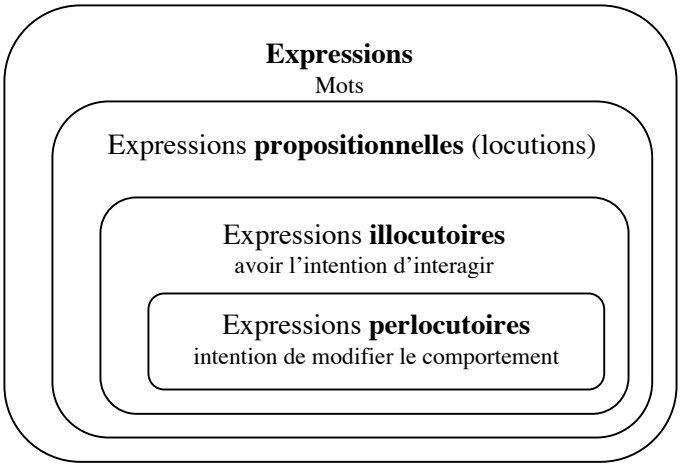


FIG 2. Les différents niveaux de langage dans la théorie de Searle.

Donnons les définitions essentielles de ces niveaux de langage :

Définition (locution) La valeur de locution est la signification, au sens classique du terme : acte de dire quelque chose.

Définition (illocution) La valeur d'illocution est l'acte effectué en disant quelque chose, l'intention au-delà des mots (conseiller, ordonner…).

Définition (perlocution) La valeur de perlocution représente les conséquences, les effets induits qui sont obtenus des autres (interlocuteurs), ou de soi.

Selon Austin la valeur d'illocution est fondamentalement la plus importante. Il précise également que cette valeur est spécifiquement conventionnelle, contrairement aux deux autres. C'est la clé de l'utilisation et de la compréhension du langage naturel, si bien que les langages d'interaction entre agents font l'amalgame entre acte illocutoire et acte de langage. De plus Searle considère l'acte de langage comme étant la plus fondamentale unité linguistique. Et son ouvrage intitulé Speech Acts [Searle, 1969] est à l'origine de nombreuses recherches dont « les résultats nous permettent de mieux appréhender les travaux sur les langages d'interaction entre agents » (Jean-luc Koning et Sylvie Pesty).

FIPA-ACL défini 22 actes de langage (aussi appelés performatifs), dont 4 actes primitifs qui sont *inform*, *request*, *confirm* et *disconfirm*. Avec les 18 actes composés à partir des actes primitifs, on obtient un ensemble d'actes de langage qui peut être découpé en 5 catégories, qui traduisent l'intention d'un message. On retrouve ces catégories comme étiquettes des colonnes du tableau suivant.

	Transmission d'information	**Demande d'information**	**Négociation**	**Action**	**Gestion d'erreurs**
Accept-proposal			x		
Agree				x	
Cancel				x	
Cfp (call for proposal)			x		
Confirm	x				
Disconfirm	x				
Failure					x
Inform	x				
Inform-if	x				
Inform-ref	x				
Not-understood					x
Propagate				x	
Propose			x		
Proxy				x	
Query-if		x			
Query-ref		x			
Refuse				x	
Reject-proposal			x		
Request				x	
Request-when				x	
Request-whenever				x	
Subscribe		x			

Tableau 1. Actes de communication du modèle FIPA-ACL, classés par catégories.

Dans ce modèle, chacun des actes de communication est défini par une pré-condition de faisabilité FP (« *Feasibility Precondition* ») et un résultat attendu RE (« *Rational Effect* »). Le modèle formel sur lequel se base la construction des actes de langage assure donc la stabilité de système. De plus, le modèle mental d'un agent est basé sur les trois attributs primitifs suivants : Croyance (*Belief*), Incertitude (*Uncertainty*) et But (Intention). Ils sont respectivement formalisés par les opérateurs modaux B, U et C (pour *Choice*) :

Bi(p) signifie que i (implicitement) croit (que) p
Ui(p) signifie que i est incertain à propos de p mais pense que p est plus probable que *not(p)*
Ci(p) signifie que i souhaite que p soie vérifiée

On peut ainsi donner la définition de l'acte *confirm* :

```
<i, confirm (j, P) >
FP : Bi(p) and Bi( Uj(p) )
RE : Bj(p)
```

FP : l'agent i croit que (p) est Vraie et i croit que Agent-j n'est pas sûr que (p) soit Vraie.
RE : l'agent j croit que (p) est Vraie.

1.2.2 Structure des messages

Un message FIPA-ACL définit un certain nombre de champs dont certains sont obligatoires et d'autres facultatifs. Les champs les plus courants sont donnés dans le tableau ci-dessous.

Champ	**Catégorie de l'élément**
performative	Type d'acte de communication : représente la valeur illocutoire d'un acte de langage.
sender	Participant : Agent émetteur
receiver	Participant : Agent récepteur
reply-to	Particiapnt
content	Contenu du message
language	Description du contenu : langage utilisé pour exprimer le contenu du message (voir *content*)
encoding	Description du contenu : codage du contenu.
ontology	Description du contenu : ontologie utilisée pour l'expression du contenu du message.
protocol	Contrôle de la conversation
conversation-id	Contrôle de la conversation
reply-with	Contrôle de la conversation : Demande au destinataire de répondre au présent message avec cet identifiant.
in-reply-to	Contrôle de la conversation
reply-by	Contrôle de la conversation

Tableau 2. Eléments d'un message FIPA-ACL.

1.2.3 Le protocole

Comme nous l'avons vu précédemment, les agents que nous considérons sont cognitifs et communicants ; ils ont besoin d'un langage élaboré pour pouvoir échanger des messages. Toutefois, un modèle structurant uniquement le format d'un message ne suffit pas en lui-même pour pouvoir structurer des conversations entre agents, ce qui est nécessaire pour conserver la cohérence des échanges successifs. Nous avons donc besoin de la notion de protocole pour supporter des conversations valides, ce qui permet aux actes de langage de prendre tout leur sens.

Un tel protocole permet d'inclure le comportement d'un agent dans un contexte particulier, auquel il doit se conformer en envoyant des messages cohérents grâce à l'utilisation d'actes de langage adaptés au discours (c'est-à-dire valides à n'importe quel moment de l'exécution du protocole) : le protocole définit les types de messages possibles à chaque instant.

FIPA-ACL supporte par défaut un certain nombre de protocoles comme, par exemple, des protocoles d'enchères, de requêtes ou de propositions. Ces protocoles sont décrits en utilisant la notation AUML qui est une extension de UML (*Unified Modeling Language*) aux agents [FIPA DIAG].

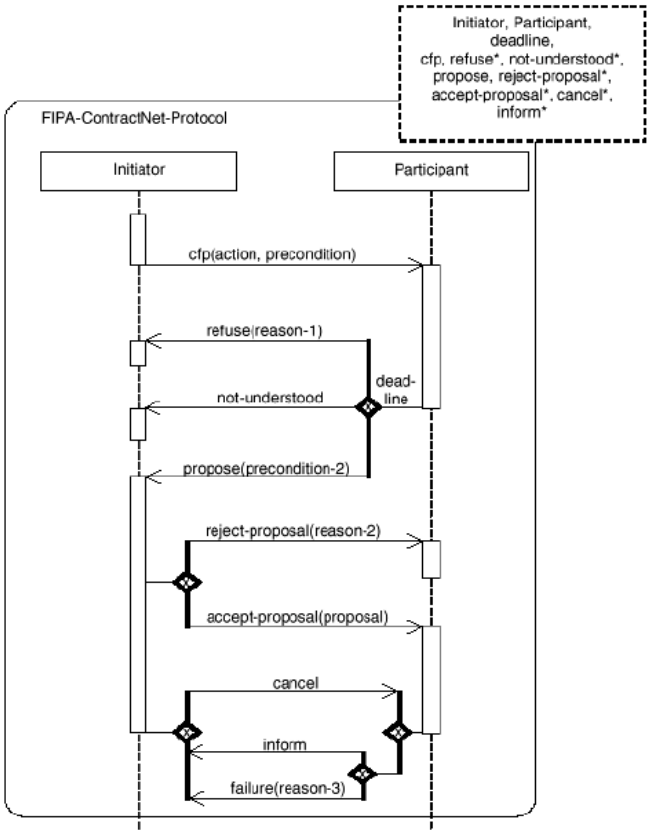


FIG 3. Le protocole *Contract-Net* (réseau contractuel), représenté en AUML.

1.2.4 Le langage de contenu

Le langage de contenu se réfère à la valeur locutoire d'un message envoyé d'un agent à un autre, c'est-à-dire au contenu « exprimé » de ce message, et à la façon de présenter ce contenu. Un bon langage de contenu doit être capable d'exprimer des expressions suffisamment riches, doit pouvoir être traité efficacement et doit bien s'adapter à l'ensemble du système au niveau du choix des technologies utilisées.

Un message FIPA-ACL, que nous présentons dans le chapitre suivant, supporte par défaut les langages FIPA-SL, KIF (Knowledge Interchange Format), XML et RDF ; FIPA-SL ayant été spécifiquement défini par la FIPA.
De plus la FIPA a défini une bibliothèque de langages pouvant être utilisé par les agents. L'ajout d'un langage à cette bibliothèque est contraint par quelques règles de base, comme le respect d'une nomenclature, d'un développement syntaxique satisfaisant et la livraison d'exemples d'utilisation ainsi que d'une documentation claire.
Signalons que l'emploi d'un langage de contenu n'est en aucun cas recommandé par rapport à un autre. La seule contrainte pour l'emploi d'un langage de contenu, dans le modèle FIPA-ACL, concerne la gestion des agents qui impose l'utilisation du langage FIPA-SL0.

Dans notre contexte précis, nous utiliserons le langage OWL [W3C, 2004a] comme langage de contenu, pour les raisons suivantes :

- Fort pouvoir expressif comme langage de représentation de connaissances (notamment : relations plus complexes et contraintes plus complexes qu'avec RDFS [W3C, 2004b], que OWL étend)

- Supporte les termes dérivés de différentes ontologies (ce n'est pas le cas pour FIPA-SL et KIF, ce qui représente une énorme limitation)

- S'adapte parfaitement au Web, pour lequel il a été conçu (informations et Services Web)

- Potentiellement largement accepté par la communauté, puisque défini et supporté par le w3 consortium, ce qui est un atout majeur pour l'interopérabilité.

- Uniformité du traitement global dans le système puisque nous l'avons choisi comme langage d'expression des connaissances / ontologies (voir le paragraphe suivant).

OWL semble être le langage le plus adapté à la modélisation, à la maintenance et au partage d'ontologies [Zou et al., 2003].

1.2.5 Découverte de services : Les DF

La notion de *Directory Facilitator* [FIPA DF] s'apparente à la technologie « Universal Description, Discovery and Integration » [UDDI] utilisée pour l'enregistrement et la publication de services Web.
A l'instar de celui-ci, un DF permet à un agent d'enregistrer (de retirer, et de rechercher) un service en fournissant le type et le propriétaire de ce service. La description peut, et de préférence doit, contenir des détails sur le service. Il s'agit donc d'un service de type « pages jaunes ».

1.3 Contenu

Nous aborderons dans cette partie la communication spécifique entre agents cognitifs, et nous présenterons la notion d'ontologie : le modèle de représentation des connaissances manipulées par les agents de notre système.
Une ontologie étant à la base du raisonnement d'un agent, nous verrons ensuite pourquoi un problème de compréhension est inévitable. Et afin de résoudre les éventuels conflits par l'utilisation d'un protocole de communication adapté, qui est l'objet de notre étude, nous présenterons la notion d'alignement d'ontologies ainsi que son rôle dans le contexte de notre projet.

1.3.1 Les ontologies

Définition (ontologie) Une ontologie est, selon Tom Gruber [Gruber, 1995], la spécification formelle et explicite d'une conceptualisation partagée d'un domaine, et où une conceptualisation est une vue abstraite du monde représenté comme un ensemble d'objets.

Autrement dit, une ontologie est une description, à la manière de la description formelle d'un programme, des concepts d'un domaine donné (classes) et des propriétés de ces concepts qui en décrivent les attributs (roles ou properties) et les restrictions (role restrictions).
Dans la pratique, une ontologie revêt souvent la forme d'une taxonomie de concepts ou d'une hiérarchie de classes. C'est un moyen qui s'apparente aux bases de connaissances mais qui est plus facilement compréhensible et réutilisable dans un contexte de transmission de connaissances, par exemple entre agents logiciels.

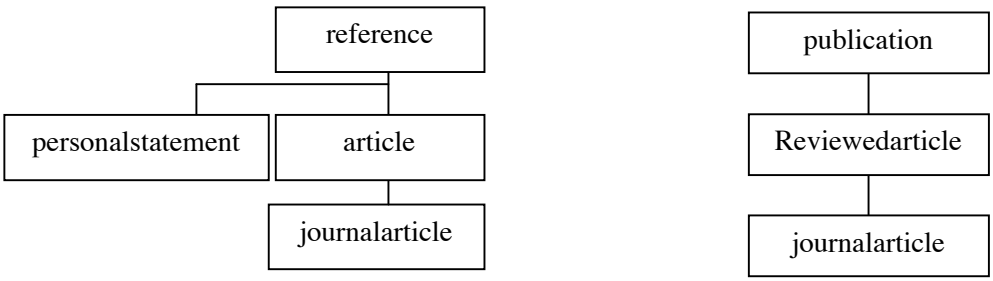


FIG 4. Deux ontologies se référant à un même domaine.

Les deux exemples de la figure font également entrevoir les différences de granularité qui peuvent exister entre deux ontologies relevant d'un même domaine: ceci étant à la base du problème de communication que nous traitons, nous aurons l'occasion d'y revenir ultérieurement.

Les motivations majeures pour l'utilisation d'ontologies sont donc principalement :

- Partager la compréhension de la structure des informations entre agents (humains ou logiciels).
- Permettre la réutilisation de connaissances spécifiques à un domaine.
- Mettre en évidence des connaissances « implicites ».
- Séparer les connaissances d'un domaine du contenu opérationnel du domaine.
- Analyser les connaissances du domaine.

1.3.1.1 Types d'ontologies

Les ontologies peuvent être classées selon trois axes, à savoir :
- leur degré de dépendance par rapport à une tâche, ou un domaine.
- le niveau de détail de leur axiomatisation.
- la nature de leur domaine.

La première approche permet de distinguer les ontologies de plus haut niveau (*top-level ontology*) qui décrivent des concepts généraux indépendants d'un problème ou d'un domaine particulier (espace, temps, matière, objet…), des ontologies de domaines ou encore d'applications.

La deuxième approche permet de faire la différence entre une ontologie de référence et une ontologie « grossière » partageable. Une ontologie de référence (off-line), ayant par définition une finesse de définition élevée grâce à de nombreux axiomes, permet d'être plus précis. En revanche, une ontologie avec un minimum d'axiomes (on-line) pourra être échangé et utilisé plus facilement.

Le dernier point de vue permet de mettre en évidence des ontologies décrivant une classification de primitives utilisées par un langage de représentation des connaissances : concepts, attributs, relations…

1.3.1.2 Le langage OWL

Le web ontology language OWL [W3C, 2004a] est une recommandation du w3c pour la modélisation d'ontologies. C'est une couche plus formelle que le modèle RDF (*Resource Description Framework*) [W3C, 1999] pour la représentation de connaissances. OWL se décline en trois versions : OWL Full, OWL DL et OWL Lite.

Nous utiliserons donc OWL à la fois pour représenter les ontologies et comme langage de contenu des messages entre agents.

```
<rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xml:base="http://www.example.org/ontology1"
    xmlns="http://www.example.org/ontology1#">

<owl:Class rdf:ID="publication">
</owl:Class>

<owl:Class rdf:ID="reviewedarticle">
  <rdfs:subClassOf rdf:about="#publication"/>
</owl:Class>

<owl:Class rdf:ID="journalarticle">
  <rdfs:subClassOf rdf:about="#reviewedarticle"/>
</owl:Class>

</rdf:RDF>
```

```
<rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xml:base="http://www.example.org/ontology2"
    xmlns="http://www.example.org/ontology2#">

<owl:Class rdf:ID="reference">
</owl:Class>

<owl:Class rdf:ID="article">
  <rdfs:subClassOf rdf:about="#reference"/>
</owl:Class>

<owl:Class rdf:ID="journalarticle">
  <rdfs:subClassOf rdf:about="#article"/>
</owl:Class>

</rdf:RDF>
```

FIG 5. Exemples : deux ontologies exprimées en OWL.

1.3.2 Interopérabilité

Après avoir présenté les notions à la base de notre travail, revenons maintenant sur le problème auquel nous nous attaquons. Les agents, que nous avons présenté au début de ce chapitre, communiquent selon un modèle générique dont le principe a également été introduit dans les paragraphes précédents. Ils communiquent en se basant sur leurs ontologies. Ces ontologies, qui servent à représenter les connaissances et aussi à faciliter leur transfert, induisent toutefois un problème d'interopérabilité dans le contexte d'un système multi-agents. Différentes approches pour traiter ce problème sont présentées dans le chapitre suivant.
Nous présentons dans ce paragraphe le problème ainsi que les outils que nous utiliserons pour nous aider à le résoudre dans notre approche.

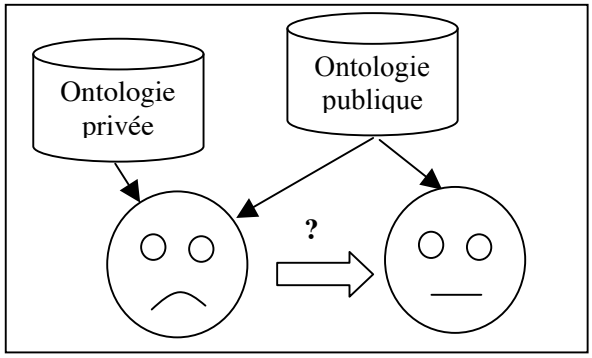


FIG 6. Le problème de communication entre deux agents.

La figure FIG 6, issue de [Stuckenschmidt and Timm, 2002], permet de bien appréhender le problème : il est lié à l'hétérogénéité des ontologies utilisées par les agents et à leur mise en correspondance. Les différentes approches que l'on peut envisager pour le résoudre sont les suivantes :

- l'Emergence : Cette approche générique promeut l'évolution libre d'ontologies publiques, ce qui n'est pas envisageable ; des ontologies privées existent et sont à prendre en considération.

- La Fusion : Cette approche promeut la fusion d'ontologies en une ontologie universelle; ce n'est pas non plus envisageable pour l'autonomie des agents.

- La mise en correspondance : c'est l'approche la plus prometteuse. Seuls circulent des alignements qui sont des informations de mise en correspondances entre ontologies (des ensembles de paires d'éléments issus de chaque ontologie). Ils peuvent être stockés publiquement, échangés, et cela ne nuit pas à l'autonomie des agents.

C'est cette dernière approche que nous retenons, mais elle nécessite une automatisation de la production d'alignements pour être valable. En effet la proposition de base est de définir manuellement des alignements, ce qui rend bien sûr impossible le passage à l'échelle. La production automatique d'alignements satisfaisants, ou alignement d'ontologies, est une activité de recherche actuellement en plein essor. Nous utiliserons donc cette approche dans la conception de notre protocole.

1.3.3 Alignement d'ontologies

Nous présentons ici le concept d'alignement d'ontologies de façon générale. Nous reviendrons sur son utilisation de manière plus détaillée lors de la définition des services dont nous avons besoin pour l'établissement de notre protocole.

Comme nous l'avons présenté, se pose donc la question de la comparaison entre deux ontologies. Quand deux agents dialoguent, ils doivent d'abord définir si leurs domaines de connaissances sont *compatibles* pour des échanges de requêtes ; en langage courant la question peut être formulée ainsi : « Parlent-ils de la même chose ? ». Il convient donc de déterminer quels types de relations existent entre les éléments de deux ontologies auxquelles ils se réfèrent dans leurs messages, si de telles relations existent.

En fait, l'objectif de l'alignement est d'identifier des concepts qui peuvent être considérés comme similaires, quelque soit l'utilisation qui est faite de l'alignement par la suite : il peut s'agir de tâches comme l'interprétation d'une requête, la traduction d'un message ou l'obtention d'axiomes de passage entre deux ontologies.

Définition (alignement d'ontologies) Un ensemble de paires d'éléments issus des deux ontologies qui ont été alignées : Selon la méthode utilisée, un alignement peut contenir des informations supplémentaires comme une mesure de similarité ou un facteur de confiance pour chaque couple d'éléments.

Aligner deux ontologies revient à définir, selon [Euzenat and Valtchev, 2003], les distances entre les entités et retenir les meilleurs couples au sens de la mesure de distance. Dans la pratique cette mesure se rapporte à une mesure de similarité entre types de données, valeurs et URI.

Nous donnons ci-dessous les catégories des méthodes de calcul de distance résumées dans [Euzenat and Valtchev, 2003] (issues uniquement de systèmes réels) :

- Terminologique : comparaison de labels d'entités.
- Selon structure interne : comparaison des intervalles de valeurs, la cardinalité des attributs.
- Selon structure externe : comparaison inter-entités ; positions dans une taxonomie.
- Extensionnelle (en général : traite d'instances de classes) : comparaison des entités rattachées.
- Sémantique : comparaison des modèles d'entités.

La méthode terminologique est, bien entendu, la plus facile à implémenter. On trouve notamment dans cette catégorie le calcul de la *distance d'édition* (comparaison des chaînes de caractères) ainsi que la comparaison terminologique assistée par un dictionnaire (de synonymes ou bilingue). On peut aussi imaginer faire appel à d'autres techniques comme, par exemple, la lemmatisation [Porter, 1980] qui fait appel à des procédés linguistiques pour identifier la racine d'un mot (et dont une forme dérivée pourrait ne pas être reconnue au moment de sa comparaison avec un autre mot).

```
<?xml version='1.0' encoding='utf-8' standalone='no'?>
<!DOCTYPE rdf:RDF SYSTEM "align.dtd">

<rdf:RDF xmlns='http://knowledgeweb.semanticweb.org/heterogeneity/alignment'>
<Alignment>
  <xml>yes</xml>
  <type>11</type>
  <onto1>file://localhost/Volumes/Phata/JAVA/align/rdf/onto1.owl</onto1>
  <onto2>file://localhost/Volumes/Phata/JAVA/align/rdf/onto2.owl</onto2>
  <map>
    <Cell>
      <entity1 rdf:resource='http://www.example.org/ontology1#reviewedarticle'/>
      <entity2 rdf:resource='http://www.example.org/ontology2#article'/>
      <measure rdf:datatype='xsd:float'>0.5333333333333333</measure>
      <relation>=</relation>
    </Cell>
    <Cell>
      <entity1 rdf:resource='http://www.example.org/ontology1#journalarticle'/>
      <entity2 rdf:resource='http://www.example.org/ontology2#journalarticle'/>
      <measure rdf:datatype='xsd:float'>0.0</measure>
      <relation>=</relation>
    </Cell>
  </map>
</Alignment>
</rdf:RDF>
```

FIG 7. Un alignement produit par l'API d'alignement développée dans le contexte de ce projet [Euzenat, 2004].

Chapitre 2

Quelques propositions pour le partage du sens entre agents

Nous présentons dans ce chapitre différentes approches qui abordent le même problème que celui que nous traitons : la compréhension partagée des connaissances au sein d'un système multi-agents (SMA), et dans lequel les acteurs manipulent des ontologies pour représenter leurs connaissances.

2.1 L'approche "NASA / Knowledge Evolution, Inc" : ONP

2.1.1 Introduction

Nous présentons dans cette partie le processus de négociation développé par S. Bailin et W. Truszkowski [Bailin and Truszkowski, 2002] : ONP (Ontology Negotiation Process). Nous mettrons en évidence les intérêts de cette initiative et ses inconvénients.

2.1.2 Présentation de cette approche

Le protocole présenté dans ce projet s'attaque à un problème très similaire à celui que nous traitons, à savoir permettre l'exploitation de bases de connaissances distribuées par des agents logiciels. Il traite également de mise en correspondance d'ontologies, mais utilise d'autres moyens que ceux que nous envisageons dans notre approche.
ONP intervient au niveau de la réception de chaque message échangé entre un agent A et un agent B : l'agent B récepteur va mettre en jeu différents mécanismes afin de comprendre le message qu'il reçoit, et ajouter les informations nouvellement acquises à son ontologie. Ce processus diffère du notre car les évolutions d'une ontologie suite à l'acquisition de données d'interprétation ne sont pas utilisables par d'autres agents ; ils devront eux mêmes effectuer des étapes d'interprétation, de clarification et d'ajournement, déjà effectuées précédemment par l'agent B avec l'agent A. Ces étapes sont toutefois dignes d'intérêt ; nous les présentons donc au cours du paragraphe suivant.
Description des 4 étapes centrales du processus ONP :

Interprétation. Au cours de cette phase, un agent récepteur B détermine si un message est correctement interprété : il vérifie d'abord l'existence des termes dans son ontologies. Le cas échéant il doit interroger une base de synonymes et exécuter, pour chacun des termes inconnus, une requête de confirmation auprès de l'agent émetteur A en lui proposant les synonymes obtenus.
Si l'ensemble W des termes encore inconnus a un cardinal inférieur à une valeur fixée comme acceptable pour la compréhension du message, alors l'agent récepteur transmet cet ensemble W à l'agent émetteur A : commence alors la phase de clarification.

Clarification. Pour chacun des termes de l'ensemble W, l'agent A cherche des synonymes dans sa propre ontologie. Il cherche également des liens possibles grâce à des relations de spécialisation, de généralisation et éventuellement d'autres.

Evaluation de la pertinence. C'est l'évaluation des résultats d'une requête par rapport à la requête elle même. Dans cette approche ONP, les résultats sont des URLs ou des documents ; il est donc facile de comparer les mots clés.

Mise à jour. Cette dernière étape a pour objectif la mise à jour de l'ontologie de l'un ou des deux agents : un nouveau concept, une nouvelle distinction ou simplement un nouveau terme peuvent ainsi être introduits dans une ontologie.

Ces étapes sont au cœur du processus de négociation, mais signalons aussi que le protocole global est géré, à un niveau supérieur, par une machine à états dont l'objectif est de déterminer le comportement successif d'un agent.

Notons que lors de la phase d'interprétation nous avons évoqué l'utilisation d'un service de mots externe : il s'agit de la base WordNet qui permet d'obtenir des synonymes et d'autres mots pour guider le processus.

2.1.3 Conclusion

Comme nous l'avons vu, l'objectif de cette approche est bien la compréhension de messages. ONP suit une démarche de négociation "terme à terme" (et ne concerne que les termes) qui est dépendante des ressources utilisées pour l'interprétation : Il est en effet difficile de conserver la sémantique d'un terme en lui choisissant un synonyme au hasard, même si ce synonyme est ensuite confronté à l'ontologie de laquelle est issu le terme initial. Signalons que seul le concepteur d'une ontologie connaît suffisamment le sens d'un terme de son ontologie et est donc à priori le seul capable de trouver un terme, manuellement, qui soit équivalent au terme original avec une précision complètement fiable.
De plus cette approche ne tire pas profit de façon optimale de l'interaction entre deux agents: même si les ontologies sont mises à jour, seuls des nouveaux éléments sont assimilés, sans aucun lien avec l'agent ayant permis cet apprentissage.

2.2 L'approche "MERIT / Infonomics"

2.2.1 Introduction

Nous présentons dans cette partie le processus d'apprentissage de correspondances (ou *mappings*) développé par F. Wiesman, N. Roos et P. Vogt [Wiesman et al., 2001] pour des instituts de recherche hollandais. Ce projet traite de la résolution du problème d'interopérabilité dans la communication entre agents par l'apprentissage de correspondances entre les ontologies qu'utilisent les agents pour raisonner et s'exprimer. Le principe développé ici est celui de la recherche de correspondances à partir d'instances de concepts.

2.2.2 Présentation de cette approche

Cette approche, tout comme les autres, s'attaque au problème d'hétérogénéité dans la représentation des connaissances. Le postulat de départ est ici que les alternatives existantes nécessitent des connaissances *à priori* trop contraignantes, d'où l'hypothèse de l'apprentissage automatique de *mappings* : cela permet de s'affranchir de la nécessité d'avoir des concepts partagés, des ontologies dérivées les unes des autres, et de l'intervention humaine pour spécifier manuellement les relations entre différentes ontologies.

La méthode se base sur la mise en correspondance d'instances de concepts afin de pouvoir communiquer à propos d'un concept subsumant. Elle se propose ainsi de résoudre une partie du problème de l'hétérogénéité des sources de connaissances : les conflits de noms, structurels et représentationnels.

Le processus d'apprentissage de cette approche se base sur les jeux de langage développés par Steels [Steels, 1996], et dont l'objectif est d'obtenir un système de communication consensuel en ne partant de rien : il s'agit globalement d'échanger des termes et de les évaluer.
L'utilisation du jeu de langage pour l'apprentissage de *mappings* vise tout d'abord à obtenir une attention conjointe des deux agents prenant part au dialogue : un agent transmet une instance à un autre agent, puis l'agent récepteur recherche le concept dont une instance a un maximum de mots en commun avec l'instance initialement transmise.
C'est pour cela qu'il est nécessaire que des instances soient communes aux deux agents, exprimées en fonction de leurs ontologies respectives.

L'un des deux agents, nommons le A, peut ensuite commencer à établir un *mapping* entre les concepts terminaux du concept choisi : pour cela il a besoin que son interlocuteur, l'agent B, lui envoie des concepts terminaux de son concept, avec des instances correspondantes. Notons que les concepts terminaux doivent être identifiés de manière unique pour éviter des conflits de nommage, car ils sont transmis simultanément : la hiérarchie des concepts n'existe alors plus. L'intérêt est ici de s'affranchir des différences structurelles des ontologies ; les concepts sont mis à plat.
L'agent A cherche alors à établir des associations entre ses concepts terminaux et ceux de l'agent B. Il se base pour cela sur la proportion de mots communs dans les instances.

Des envois successifs de ce type permettent à l'agent A de maximiser des forces d'association liant les concepts terminaux. De plus, les associations sont exprimées grâce à des opérateurs spécifiques aux chaînes de caractères (*merge* et *split*) pour indiquer la relation entre deux concepts, et donc le moyen de transformer une instance d'un concept en l'instance d'un autre concept.
Le processus s'arrête quand l'agent A considère certaines associations comme correctes. Le *mapping* est alors final : c'est un *mapping* des concepts de l'agent A vers des concepts de l'agent B. Pour cette raison il est dit asymétrique, et ne permet donc pas à l'agent B de disposer de correspondances entre ses concepts et ceux de l'agent A.

2.2.3 Conclusion

Cette approche est une solution générique au problème d'hétérogénéité des ontologies. Elle souffre toutefois de quelques limitations. Tout d'abord les *mappings* obtenus ne sont pas bidirectionnels. De plus ils nécessitent un temps de calcul élevé à cause du parcours des concepts, des envois successifs d'ensembles { concepts + instances }, et du traitement des chaînes de caractères. La nécessité que les ontologies aient des instances de concepts en commun pour guider les associations limite également son utilisation dans le cas général.

2.3 L'approche "Alignement réciproque d'ontologies"

2.3.1 Introduction

Nous présentons dans cette partie l'approche développée par Les Gasser et Jun Wang de l'université de l'Illinois [Wang and Gasser, 2002]. Nous mettrons en évidence les intérêts de ce projet et ses inconvénients.

2.3.2 Présentation de l'approche

Cette approche présente une méthode d'alignement également basée sur l'échange d'instances d'ontologies, mais elle utilise, contrairement aux autres, une représentation ensembliste et définit des catégorisations du domaine qui servent à raisonner sur l'alignement à un niveau logique.
Lors d'un "jeu d'alignement", des concepts sont échangés entre deux agents. Selon l'appartenance ou non d'un concept à une catégorie les liens entre les concepts sont mis à jour.
Cette méthode repose donc sur l'échange d'objets représentationnels pour mener une action collective coordonnée au sein d'un système multi-agents. L'objectif est la convergence des agents vers une ontologie partagée, à partir des catégorisations.

Description du cycle de base du jeu d'alignement :

Sélection aléatoire de deux agents A et B.
A choisi aléatoirement une catégorie c de son ontologie, et choisi une instance o de c.
B reçoit o et repère une catégorie c' telle que o appartienne à c'. B choisi alors o' appartenant à c' (mais o' différente de o) l'envoie à l'agent A.
A informe B sur l'appartenance de o' à la catégorie c.
A et B adaptent éventuellement leurs ontologies, ainsi que les matrices d'association.

2.3.3 Conclusion

Le processus décrit dans cette approche, comme dans l'approche MERIT, est basé sur la manipulation d'instances ; il faut que des objets soient connus de part et d'autre. Il s'agit donc d'une approche fondée sur une vision extensionnelle (voir chapitre 1). Il n'est pas non plus possible d'automatiser la communication entre deux agents quelconques, mais le processus permet toutefois l'adaptation d'ontologies et la convergence vers une ontologie commune à un groupe d'agents ; la convergence absolue n'a cependant pas été prouvée de manière formelle.

2.4 Synthèse

Dans ce chapitre nous avons présenté différentes approches qui s'attaquent au problème de l'hétérogénéité des connaissances et au problème d'interopérabilité entre agents qui en découle. Nous remarquons cependant que ces approches regroupent un certain nombre de limitations et qu'aucune d'entre elles ne se propose de résoudre le problème dans le cas général.

Résumons ici les principales remarques d'ordre général que l'on peut faire en se basant sur ces approches, et afin de concevoir un système qui puisse traiter le problème d'interopérabilité entre agents d'une manière générale.

Il est souhaitable d'éviter que :

- tous les agents du système aient besoin d'implémenter un mécanisme d'apprentissage ou de raisonnement spécifique (du type *jeu de langage* entre deux agents), ce qui est une hypothèse trop lourde. Cela nuit également à leur indépendance.

- les sources de connaissances soient manipulées au niveau global du système.

- des objets représentationnels, des instances de concepts, soient nécessairement connus de différents agents (exprimées en fonction de l'ontologie de chacun).

- des recherches de correspondances entre concepts ne se basent que sur des données terminales, ou ne prennent même simplement pas en compte les relations existantes entre les concepts.

- trop de connaissances *a priori* soient nécessaires.

- des ressources hétérogènes ne soient amenées à interagir par l'intermédiaire de différents paradigmes : agents, services web, bases de données, objets….

C'est sur cette analyse que nous nous basons pour présenter notre approche.

Chapitre 3

Comment rendre les messages intelligibles

3.1 Introduction

Nous présentons ici, au regard des limitations évoquées dans le chapitre précédent, les mécanismes et les structures qui peuvent, ou qui doivent, entrer en ligne de compte lors de la conception de l'architecture permettant de réaliser nos objectifs d'interopérabilité dans un système Multi-agents.

Ces besoins sont les suivants :

Services d'alignements. Des services d'alignement, accessibles à chacun des agents. Un tel service est nécessaire à l'utilisation d'opérations liant alignements et ontologies, dont la production d'alignements. Ces Services ayant un rôle majeur dans notre approche, nous présentons par la suite les services (au sens de fonctions) qu'ils doivent offrir. Notons qu'un service d'alignement sera idéalement lui aussi un agent.

Bibliothèques. Des bibliothèques d'alignements, accessibles à tous les agents par l'intermédiaire de services d'alignement. Ces bibliothèques servant juste de moyen de stockage, nous ne les étudierons pas de manière détaillée.

Protocole. Un protocole d'interaction définissant les échanges de messages entre les entités du système, et permettant de piloter la production d'alignements.

Annuaires. Des annuaires, afin qu'un agent puisse trouver un interlocuteur si besoin est. Cela permet notamment de simplifier et d'accélérer les futures interactions d'un agent. Des exemples d'implémentations de cette notion d'annuaire sont les « Directory Facilitator » [FIPA DF] et la technologie UDDI [UDDI]. Toutefois, cette notion d'annuaire n'étant pas centrale dans notre problématique, nous ne l'évoquerons pas en détails. Nous considérons cependant que les agents sont capables de trouver des services d'alignement par ce type de moyens.

3.2 Service d'alignement

Ce service, que nous noterons SA dans les paragraphes suivants, doit permettre de produire et de manipuler des alignements. Il doit donc offrir les services suivants :

- Alignement d'ontologies.
- Filtrage d'alignements (selon un seuil acceptable de similarité entre les éléments des couples).
- Traduction de messages : il s'agit de reformuler un message en fonction d'une autre ontologie.
- Production de programmes de traduction (pour automatiser les transformations sans rechercher à obtenir un nouvel alignement).
- Production d'axiomes correspondant à un alignement.

Nous détaillerons ces services et la façon de les utiliser ultérieurement : Le chapitre 4 est d'ailleurs dédié au service d'alignement, qui est une ressource principale de notre système.

3.3 Protocole d'interaction

Ce protocole permet de structurer les échanges de messages entre deux entités du système : Agent-SA ou Agent-Agent. Il définit des échanges atomiques de messages auxquels les agents peuvent se conformer pour atteindre leurs objectifs. De plus le protocole est conçu pour être transparent vis-à-vis de la communication. Un agent peut donc tirer profit du protocole pour augmenter les chances de succès de l'interaction, tout en ne risquant pas de se retrouver dans un "état puit", ou disons de rester bloqué dans une situation imprévue. En effet tout échec survenant lors du déroulement du protocole doit avoir pour conséquence de laisser un agent dans l'état dans lequel il serait s'il n'avait pas utilisé le protocole.

Un tel protocole doit donc pouvoir assurer, de manière permanente, que le système se trouve dans un état stable. Nous décrirons le protocole dans le chapitre 5.

3.4 Annuaires

Sans entrer dans le détail des annuaires, précisons qu'un tel service doit permettre d'accéder à de nouveaux interlocuteurs, agents et services d'alignement. On s'attend donc qu'un service d'annuaire puisse fournir des informations sur ces entités et notamment, concernant les services d'alignement, la liste des algorithmes d'alignement qui peuvent être sélectionnés lors d'une requête d'alignement (voir le détail du protocole).

3.5 Synthèse

Nous avons présenté dans ce chapitre un ensemble de notions qui sont liées à ce projet. Certaines seront exploitées dans les chapitres suivants alors que d'autres n'entrent pas directement en ligne de compte pour la définition de notre protocole ; ces dernières ont néanmoins été présentées pour pouvoir être prise en compte lors du design d'un système global incluant notre protocole.

Signalons également que nous n'utilisons que des concepts du paradigme « Agent » :

- Un SA est un agent.

- Un SA peut être identifié et contacté par l'intermédiaire d'un annuaire (voir par exemple les DF du modèle FIPA).

- L'utilisation d'un SA est spécifié par un protocole. Et c'est un tel protocole que nous présenterons dans le chapitre 5.

- Les agents du système sont autonomes quant à leur manière d'interagir les uns avec les autres. Ce couplage faible, asynchrone, permet aux agents de conserver leur autonomie : nous le verrons lors de la définition des règles du protocole.

Nous allons maintenant présenter de manière détaillée, dans le chapitre suivant, les entités directement concernées par notre système.

Chapitre 4

Services d'alignement d'ontologies

Dans ce chapitre nous spécifions le service d'alignement (SA) et les structures de données que doivent manipuler les entités (agents) du système. Nous exprimerons donc nos besoins en termes d'opérations nécessaires à l'utilisation d'un service d'alignement, et nous définirons ce qui qualifie l'état des agents et des SA par rapport au protocole que nous définirons dans le chapitre suivant.

4.1 Accès à une ontologie

Nous devons prendre en compte l'accès aux ontologies. Nous entendons par *accès* le fait qu'un agent, ou un service d'alignement, puisse ou non manipuler une ontologie à partir de sa référence, son URI. Un agent peut éventuellement manipuler une ontologie O qu'il ne veut pas mettre en libre accès, pour des raisons de confidentialité par exemple ; un agent peut également se référer à une ontologie qu'un autre agent ne pourra pas manipuler pour des raisons techniques provenant de l'état du réseau lui-même. Il est donc nécessaire de tester cette accessibilité, ainsi que de permettre la re-formulation d'un message en se référant à une autre ontologie : cette reformulation doit pouvoir se faire soit à partir d'une autre ontologie, qu'il convient donc d'identifier, soit à partir d'un alignement existant entre l'ontologie O et une autre ontologie.

L'accès à une ontologie tierce pouvant être initié par l'un ou l'autre des agents dialoguant, les processus de manipulation d'ontologies et d'alignement doivent être conçus avec une grande flexibilité au niveau des paramètres de configuration.

4.2 Description des services de base

Comme on l'a vu jusqu'à présent, services et agents sont amenés à manipuler des ontologies et des alignements. Nous décrirons donc, dans les paragraphes suivants, les fonctionnalités qu'ils doivent mettre à la disposition des agents du système.

4.2.1 Aligner deux ontologies

L'alignement est la fonction centrale qui permet à un agent de commander l'alignement de deux ontologies à un service d'alignement. On donne ci-dessous le schéma général de ce processus d'alignement. Il convient de préciser que cette représentation permet de prendre en compte la production d'alignements partiels et complets ; on peut également, par cette même fonction, compléter des alignements.

Définition (Alignement partiel). Un alignement partiel est un alignement entre deux ontologies qui ne fait pas état de tous les couples d'éléments possibles entre les deux ontologies.
Cette limitation est une conséquence du choix des paramètres d'alignements ; ainsi un ensemble de termes à aligner ou un seuil de similarité à respecter peut entraîner la production d'un alignement qui ne sera que partiel.

Définition (Complément d'alignement). Compléter un alignement revient à produire un nouvel alignement à partir d'un alignement existant, en fournissant à la fonction d'alignement des paramètres différents de ceux fournis précédemment.

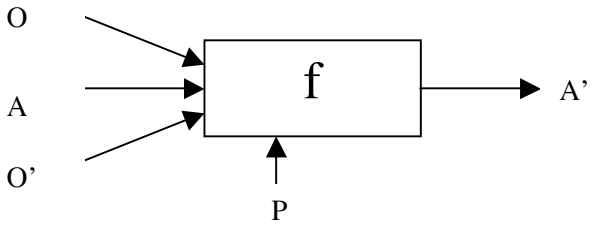


FIG 8. Fonction d'alignement, avec o et o' deux ontologies,
(A) un alignement préalable, et (p) un ensemble de paramètres.

L'alignement proprement dit doit donc prendre en compte différents paramètres complémentaires (P). Nous décrivons dans les paragraphes suivants, à titre d'exemples, des paramètres qu'il est intéressant de pouvoir spécifier lors de l'alignement : le choix de l'algorithme à utiliser pour aligner, le seuil de similarité acceptable pour un couple d'éléments et l'ensemble T des termes à considérer. [Euzenat, 2004] présente également, dans la description de l'API d'alignement, d'autres paramètres comme le niveau du format d'alignement et l'arité de l'alignement.

Algorithme. Le choix de l'algorithme d'alignement permet de spécifier comment la distance entre deux éléments sera calculée. La distance peut être une distance d'édition, c'est-à-dire se basant sur la différence de caractères entre deux chaînes, ou tout autre distance permettant d'évaluer la proximité de deux termes, que ce soit de manière syntaxique, sémantique, ou d'un tout autre ordre. Précisons que cette problématique est transparente de notre point de vue.

Seuil. Nous adoptons la définition de similarité présentée dans [Euzenat and Valtchev, 2003] à savoir la valeur, comprise entre 0 et 1, permettant d'évaluer la proximité de deux termes. Par seuil de similarité nous faisons référence à la valeur minimum acceptable de la distance entre deux termes ; une distance inférieure au seuil ne permet pas à un couple d'éléments d'apparaître dans l'alignement produit à partir des deux ontologies desquels ils sont issus.

Ensemble de termes. Préciser l'ensemble T des termes à considérer lors de l'alignement de deux ontologies, O et O', permet de limiter le parcours de ces ontologies. Les termes fournis sont les termes de O à aligner avec O'. On parle donc dans ce cas d'alignement partiel, dans la mesure ou seul un sous-ensemble des éléments de O va être confronté aux éléments de O'.

Fonction d'alignement. Une fonction d'alignement de la forme suivante doit donc être disponible et utilisable pendant l'exécution de notre protocole (cf. chapitre suivant) :

align (O, O', a, P) → a'
où - O et O' sont les deux ontologies à aligner,
- a est l'alignement partiel (éventuellement vide),
- P est l'ensemble de paramètres.
- a' est l'alignement résultant.

4.2.2 Seuiller un alignement

Un service d'alignement doit permettre à un agent de filtrer un alignement. Il s'agit d'obtenir un nouvel alignement à partir d'un alignement duquel il veut supprimer les paires d'éléments dont la mesure de similarité est inférieure à une valeur seuil : si la valeur spécifiée pour cette fonction est supérieure à la mesure d'un couple, alors ce couple est supprimé et n'apparaîtra pas dans l'alignement résultant. Les paramètres de cette fonctionnalité sont donc les suivants : un alignement (a) et une valeur de seuil (s).

On peut définir ainsi une telle fonction :

TrimAlignment (a, V) → a'
où - a est un alignement,
- V est la valeur de seuil,
- a' est l'alignement résultant

Signalons que cette fonction n'apparaît pas dans la description du protocole car les agents et les services d'alignement sont libres de seuiller un alignement à tout moment.

4.2.3 Génération d'axiomes

La génération d'axiomes donne à un agent la possibilité d'obtenir des relations entre concepts. Il est ensuite libre d'utiliser les axiomes dans la mesure de ses capacités. Il pourra par exemple se servir d'un ensemble d'axiomes pour traduire un message lui-même, ou mettre à jour une/son ontologie.

Notons que les relations évoquées précédemment peuvent être, par exemple, des relations d'équivalence, de généralisation ou de spécialisation.

Fonction d'axiomatisation. Une fonction de production d'axiomes est définie de la façon suivante :

axioms (a) → X
où - a est l'alignement,
- X est l'ensemble d'axiomes correspondant.

axioms (O, O') → X
où - O et O' sont deux ontologies,
- X est l'ensemble d'axiomes correspondant.

Signalons que axioms(a) est équivalent à axioms(align(O,O')).

Nous donnons ci-dessous un exemple d'ensemble d'axiomes, exprimé en OWL et généré par l'API d'alignement de [Euzenat, 2004] :

```
<rdf:RDF
   xmlns:owl=http://www.w3.org/2002/07/owl#
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

      <owl:Ontology rdf:about="">
          <rdfs:comment>Aligned ontologies</rdfs:comment>
           <owl:imports rdf:resource="http://www.example.org/ontology1"/>
           <owl:imports rdf:resource="http://www.example.org/ontology2"/>
      </owl:Ontology>

      <owl:Class rdf:about="http://www.example.org/ontology1#reviewedarticle">
      <owl:equivalentClass rdf:resource="http://www.example.org/ontology2#article"/>
      </owl:Class>

      <owl:Class rdf:about="http://www.example.org/ontology1#journalarticle">
      <owl:equivalentClass rdf:resource="www.example.org/ontology2#journalarticle"/>
      </owl:Class>
</rdf:RDF>
```

4.2.4 Traduire un message

Un service d'alignement SA doit permettre de reformuler le contenu propositionnel d'un message, c'est-à-dire le traduire en des termes d'une autre ontologie. Il peut donc le faire à partir de l'ontologie O, celle ayant servie à la composition initiale du message, et d'un alignement entre cette même ontologie O et l'ontologie cible O'.
La traduction peut également se baser directement sur les deux ontologies O et O'. SA peut, dans ce cas, produire un alignement de manière invisible et ainsi retomber sur le cas précédent de traduction.
Notons qu'une ontologie tierce O' peut être obtenue grâce à la fonction *match* définie dans le paragraphe suivant.

Fonction de traduction. La fonction de traduction est définie de la façon suivante :

translate (M, O, O') → M'
où - M est le message initial, exprimé en fonction de l'ontologie O,
- O et O' sont deux ontologies,
- M' est le message traduit, exprimé en fonction de O'.

translate (M, O, a) → M'
où - M est le message initial,
- O est l'ontologie a partir de laquelle M est exprimé,
- a est un alignement entre O et une autre ontologie O',
- M' est le message traduit, exprimé en fonction de O'.

Signalons que translate(M, O, a) est équivalent à translate(M, O, align(O,O')).

4.2.5 Trouver une ontologie

Afin de pouvoir utiliser une ontologie à la place d'une autre, il faut d'abord la trouver. Nous avons donc besoin d'une fonction qui, en fonction d'une ontologie O et d'un ensemble de termes, soit capable de trouver une ontologie O' suffisamment proche de O en termes de relations inter-concepts.

Fonction de recherche d'une ontologie. La forme de cette fonction est la suivante :

match (O, T) → O'
où - O est une ontologie quelconque,
- T est un ensemble de termes (éventuellement vide) de l'ontologie O à considérer comme prioritaires dans la recherche.
- O' est l'ontologie est la plus proche de O, conceptuellement.

Précisons que la recherche de l'ontologie O' dépend fortement du service d'alignement SA qui effectue la recherche et de l'ensemble T.

4.2.6 Obtenir un programme de traduction

Un programme de traduction doit permettre à un agent de pouvoir transformer localement un message exprimé en fonction d'une ontologie O en un nouveau message exprimé selon une ontologie O'. Le programme lui-même peut être écrit dans un langage quelconque, par exemple java ou xslt, mais l'agent qui impose un langage de traduction devra bien sûr s'assurer qu'il connaît le langage et pourra donc exécuter, ou interpréter, le programme.

Fonction de requête d'un programme de traduction. Les formes de cette fonction sont les suivantes :

program (O, O', lg) → P
où - O et O' sont deux ontologies,
- lg est le langage dans lequel P doit être exprimé,
- P est le programme résultant.

program (a, lg) → P
où - a est un alignement,
- lg est le langage dans lequel P doit être exprimé,
- P est le programme résultant.

Signalons que program(a, lg) est équivalent à program(align(O,O'), lg).

Nous donnons ici un exemple de programme XSLT :

```
<xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transform version="1.0"
        xmlns:owl="http://www.w3.org/2002/07/owl#"
        xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

  <xsl:template match="http://www.example.org/ontology1#reviewedarticle">
        <xsl:element name="http://www.example.org/ontology2#article">
               <xsl:apply-templates select="*|@*|text()"/>
        </xsl:element>
  </xsl:template>

  <xsl:template match="http://www.example.org/ontology1#journalarticle">
        <xsl:element name="http://www.example.org/ontology2#journalarticle">
               <xsl:apply-templates select="*|@*|text()"/>
        </xsl:element>
  </xsl:template>

  <!-- Copying the root -->
  <xsl:template match="/">
        <xsl:apply-templates/>
  </xsl:template>

  <!-- Copying all elements and attributes -->
  <xsl:template match="*|@*|text()">
    <xsl:copy>
        <xsl:apply-templates select="*|@*|text()"/>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

4.2.7 Ajouter un alignement à la bibliothèque

Un service d'alignement doit permettre à un agent d'ajouter manuellement un alignement à sa bibliothèque. L'agent doit alors simplement fournir un alignement et les URIs des deux ontologies desquelles est issu l'alignement.

Fonction d'ajout d'un alignement. Une telle fonction d'ajout d'un alignement est nécessaire pour qu'un agent, ou un service d'alignement, puisse ajouter une nouvelle entrée à son ensemble A d'alignements. Cette fonction est de la forme suivante :

storeAlignment (a, O, O')
où - a est un alignement,
- O et O' sont les deux ontologies correspondantes.

Fonction de vérification de l'origine. Cette fonction sert à vérifier qu'un alignement a été produit à partir d'un couple donné d'ontologies. Elle est utile lors de l'ajout d'un triple à un ensemble A, pour ne pas dupliquer d'informations. Cette fonction est de la forme suivante :

verifSources (a, O, O') → B
où - a est un alignement,
- O et O' sont les deux ontologies présumées être à l'origine de l'alignement a,
- B est une valeur booléenne : VRAI si (a) résulte de l'alignement de O et O', FAUX sinon.

4.2.8 Existence d'alignements

Afin de simplifier les interactions entre agents et services d'alignement, nous souhaitons que ceux-ci puissent mémoriser des alignements correspondant à des couples d'ontologies. Il semble également intéressant de pouvoir indiquer à un agent qu'un autre service d'alignement à déjà traité un certain couple. Ces opérations, qui sont décrites dans le chapitre suivant, utilisent des structures de données que nous présentons dans le paragraphe suivant, ainsi que les fonctions de base suivantes :

Fonction de requête d'un alignement. Cette fonction, qui permet d'obtenir un alignement correspondant à un couple d'ontologies, est la suivante :

getAlign (O, O') → a
où - O et O' sont deux ontologies,
- (a) est un alignement de O et O'

Fonction de requête d'un service d'alignement. Cette fonction permet d'obtenir un service d'alignement correspondant à un couple d'ontologies :

getService (O, O') → sa
où - O et O' sont deux ontologies,
- (sa) est un service d'alignement qui a déjà traité le couple (O, O')

4.3 Structures de données

Nous présentons ici les structures de données dont les agents et les services d'alignements du système ont besoin pour pouvoir utiliser le protocole. En effet, les agents, ainsi que les services d'alignements, doivent pouvoir évaluer les conditions spécifiées lors de la définition du protocole : ils doivent maintenir la cohérence de certaines données et donc se conformer aux spécifications suivantes.

La première partie du tableau contient les structures présentes dans les agents et les services d'alignements, la deuxième partie contient la structure spécifique aux services d'alignement, et la troisième contient le niveau d'intégration du protocole qui est une variable spécifique aux agents du système.

Nom	Contenu	Description	Statut	
			Pour un agent	pour un SA
A	<o, o', a>	Ensemble des triples associant deux ontologies à un alignement.	O	M
SA	<o, o', sa>	Ensemble des associations : service d'alignement - paires d'ontologies.	O	M
L	Lg	Ensemble des langages supportés.		M
level	N : integer	Niveau d'intégration du protocole.	M	

Tableau 3. Structures de données utilisées par les agents et les services d'alignement.
M (*Mandatory*)= obligatoire, O = Optionnel.

4.3.1 Description des ensembles

L'ensemble (A) est un ensemble de triples : ces triples sont constitués de deux ontologies et d'un alignement associé. L'ensemble (A) d'un agent, ou d'un service d'alignement, permet donc à celui-ci de mémoriser des associations permettant de retrouver soit un alignement existant, soit une paire d'ontologies correspondant à un alignement.
L'ensemble (SA) est également un ensemble de triples qui peut être géré soit par un agent, soit par un service d'alignement. Cet ensemble associe un couple d'ontologies à un service d'alignement, ce qui permet d'interroger directement ce dernier. Si un service d'alignement sa-1 apparaît dans un tel triple, alors on considère que sa-1 a déjà traité des requêtes d'alignement à partir de ce même couple d'ontologies. L'ajout d'un triple <o,o',sa> à un ensemble SA a donc pour pré requis l'alignement de o et o' par le service sa. La procédure d'ajout d'un triple à un ensemble SA n'étant pas au cœur de notre travail, nous ne la décrirons pas.

L'ensemble (L) est l'ensemble des langages qu'un service d'alignement est capable de manipuler. (L) sert essentiellement à la production de programmes de traduction. Signalons aussi qu'il n'y a pas de limitations dans les langages supportés ; il peut s'agir de java, xslt ou autre. Il est cependant plus utile, dans le type de système ouvert que nous considérons, de pouvoir générer du code facilement interprétable par le plus grand nombre possible d'agents.
Signalons que (L) n'est utile que pour un service d'alignement car un agent n'a pas d'intérêt à maintenir une telle liste ; il n'est pas prévu qu'il puisse générer un programme de traduction.

4.3.2 Statut des structures et niveau d'implémentation

Le statut d'une structure, donnée dans le tableau précédent, permet de distinguer si elle doit être obligatoirement supportée par les entités susceptibles de la gérer. Le statut de (A) et (SA) signifie que les bases d'alignements et de services d'alignement peuvent ne pas être prises en compte par les agents du système. Dans un premier niveau d'implémentation du protocole, le concepteur d'un agent peut ainsi décider de ne pas supporter les ensembles (A) et (SA) ; dans ce cas la variable *level* doit être initialisée à 1. Signalons que si la valeur de *level* est 1 et que les ensembles sont implémentés, ces derniers ne seront cependant pas exploités. Une valeur de *level* égale à 2 impose que les ensembles (A) et (SA) soient implémentés et supportés : cela permet de tirer pleinement parti du protocole de coordination.

Remarque importante : Afin de faciliter la présentation du protocole, nous ne ferons pas état du niveau d'intégration lors de la définition des règles. Nous considérerons en effet que le niveau correspond par défaut à la prise en compte maximum des structures. Notons également que, dans l'état actuel de la description du protocole, le choix du niveau inférieur d'intégration implique qu'un agent qui doit évaluer une condition faisant état d'un ensemble A, ou SA, doit se comporter comme si le test d'appartenance d'un élément à cet ensemble avait échoué. Cela revient en fait à refuser toute opération sur cet ensemble, sans essayer de la réaliser.

4.4 Synthèse

Nous avons présenté dans ce chapitre les fonctions de base qui seront utilisées par les agents et les services d'alignement au cours du déroulement du protocole. Certaines de ces fonctions sont indépendantes du contexte dans lequel elles sont appelées, et d'autres sont dépendantes du contexte : ainsi les fonctions relatives à l'existence d'alignements n'ont de sens que dans le cadre d'un appel auprès d'un agent du système qui maintient la structure de données A correspondante. Signalons aussi, avant de présenter le protocole lui-même, que les détails d'implémentation des fonctions de base sont pour l'instant laissés aux programmeurs du système : une fonction ne pouvant pas produire un résultat dans des conditions normales sera par exemple amenée à renvoyer la valeur *null*, pour permettre à un agent de prendre les mesures nécessaires et éventuellement tenter un nouvel appel.

Chapitre 5

Protocole d'alignement d'ontologies

Dans cette partie nous étudions le protocole en détail, c'est-à-dire les échanges de messages possibles entre deux entités du système, agents ou SA, afin de déterminer si les termes utilisés dans un message permettent au récepteur de ce message de l'interpréter : ceci revient à déterminer, comme on l'a évoqué jusqu'à maintenant, si chacun des concepts du message peut être relié à d'autres connaissances déjà assimilées, grâce aux alignements.

Il est important de rappeler que tous les échanges de messages décrits dans cette partie ont lieu en amont de toute action liée à l'exécution / interprétation du contenu du message lui-même.

5.1 Architecture et représentation

Nous présentons le protocole, dans les paragraphes suivants, sous forme de règles qui définissent les échanges de messages possibles entre les acteurs du système sous certaines conditions. Cette représentation est tirée de [Euzenat, 1997] et semble être la plus adaptée à la description de notre protocole grâce à sa simplicité d'écriture et de lecture, ainsi que pour la rigueur de sa notation qui facilite le codage.
Signalons que la notation AUML [AUML] est une alternative très intéressante à la notre. On la préférera cependant dans une deuxième étape d'écriture, une fois que toutes les interactions sont bien définies, afin de ne pas se perdre dans la notation graphique qui, de prime abord, peut sembler déroutante pour des novices de la modélisation. Mais une fois l'outil maîtrisé, AUML se révèle tout aussi exploitable pour la transmission de protocoles, et peut être même plus efficace pour avoir un aperçu rapide du déroulement d'un protocole. Rappelons que les protocoles d'interaction du modèle FIPA-ACL sont spécifiés en AUML.

5.1.1 Les règles

La partie supérieure de la règle définit l'envoi d'un message de X vers Y, alors que la partie inférieure définit la réponse qui doit être envoyée par Y à Z (Z est éventuellement X) si la condition (C) exprimée à droite de la règle est vérifiée. Avant l'envoi de sa réponse, Y peut effectuer un certain nombre d'actions (a1,…an).

(nom de la règle) $\dfrac{X - \text{message} \rightarrow Y}{a1,..an,\ Y - \text{reponse} \rightarrow Z}\ C$

L'ensemble des règles est divisé en deux parties principales, selon les opérations à effectuer : certaines fonctions ne peuvent êtres effectués que par les services d'alignement et d'autres aussi bien par les services d'alignement que par les agents eux-mêmes.

5.1.2 Contrôle de la conversation

Nous considérons que les échanges de messages décrits dans les paragraphes suivants utilisent et respectent les informations d'identification de messages spécifiées par les performatifs FIPA-ACL de contrôle de la conversation. Nous partons ainsi de l'hypothèse qu'un agent B répond à un agent A en lui précisant l'identifiant de réponse correspondant au message initial, émit par l'agent A.
Nous posons cette condition pour éviter tout problème lié à des croisements de messages ; cela peut arriver fréquemment dû à la nature même du système, à savoir des agents communicants de manière asynchrone à travers un vaste réseau.

Les communications doivent donc respecter la règle illustrée par l'exemple de la figure 9.

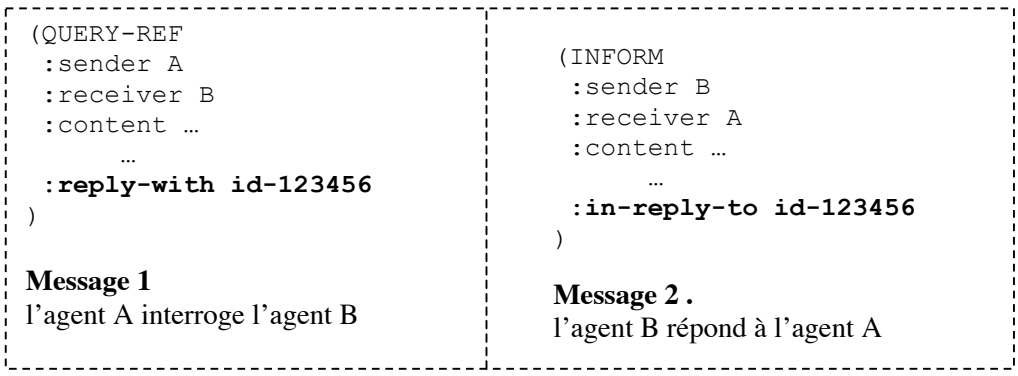


FIG 9. Principe d'identification des interlocuteurs
pour le suivi de la cohérence des échanges
dans un langage de communication entre agents.

5.1.3 Nomenclatures

Nous donnons dans les tableaux suivant la liste des performatifs et pseudo-performatifs utilisés dans les règles qui seront présentées par la suite. Nous distinguons les performatifs issus du modèle FIPA-ACL de ceux que nous introduisons spécifiquement pour ce projet et que nous identifierons par le label PCAC (pour Protocole de Coordination entre Agents Cognitifs). Notons que ces derniers sont plus des actions qui sont considérées par FIPA-ACL comme un langage de contenu de type *Speech Acts*.

Signalons également que les fonctions élémentaires utilisées dans les messages (fonctions qui ont été présentées dans le chapitre précédent) sont appelées par l'agent émetteur d'un tel message sur les structures de données propres à cet agent.

performatif	**Utilisation dans notre protocole**
request	L'émetteur demande au récepteur d'effectuer une action.
inform	Utilisé en réponse à une requête.
refuse	Refuse d'effectuer une action (un pseudo-performatif PCAC précise la raison).
failure	Echec d'une action tentée par le récepteur suite à la requête de l'émetteur.
query-if	L'émetteur demande au récepteur d'effectuer un test.
not-understood	Un paramètre est inconnu du récepteur.
confirm	Confirme qu'un tuple est bien présent dans une structure de données.

Tableau 4. Performatifs FIPA-ACL utilisés dans notre protocole (la liste complète est donnée dans le chapitre 1).

Pseudo-performatif	**Signification**
align	Demande d'effectuer un alignement d'ontologies
inaccessible	La ressource transmise est inaccessible. (problème réseau, droit d'accès…)
inconsistent-set	L'ensemble transmis n'est pas cohérent pour l'exécution de la tâche préalablement requise.
get-prog	Demande d'un programme de traduction.
get-axioms	Demande d'un ensemble d'axiomes.
find	Demande de rechercher une ontologie.
tsl	Demande de traduction d'un message (en fonction d'un alignement ou d'un couple d'ontologies)
declare	Déclaration d'une donnée à mémoriser (nouveau triple)
is-align	Transmet une interrogation sur l'existence d'un alignement.
forward	Transmet la référence d'un service d'alignement à contacter.
unknown-pair	Indique qu'un couple de données (ici d'ontologies) est inconnu.

Tableau 5. *Pseudo-performatifs* PCAC, spécifiques à notre protocole

5.2 Services fournis par les services d'alignement

5.2.1 Alignement d'ontologies

L'alignement d'ontologies peut être commandé à un service d'alignement soit par un agent, soit par un autre service d'alignement. Les paramètres de cette fonction d'alignements sont les suivants : deux ontologies (O et O'), un alignement (éventuellement vide) et un ensemble (P) de paramètres. (P) peut contenir des paramètres comme le niveau d'alignement (se reporter à [Euzenat, 2004]), une valeur de seuil pour filtrer les paires selon leur similarité, un sous-ensemble de termes à considérer ou un algorithme d'alignement. Des valeurs par défaut doivent bien sûr être gérés par les services d'alignement.

L'alignement renvoie en retour, si tout s'est bien passé, un autre alignement (a') produit en fonction des paramètres fournis.

Signalons qu'aucune condition particulière ne fait référence à l'alignement (a) fournit en paramètre ; celui-ci peut être un alignement partiel existant entre O et O', mais il peut aussi être vide.

(align-success)

$$\frac{X - \text{request}(\text{align}(O, O', a, P)) \rightarrow SA}{SA - \text{inform}(\textit{align}(O,O',a,P) \rightarrow X}$$

O et O' sont accessibles, SA interprète les paramètres de l'ensemble (P) sans problème et peut donc aligner.

Il peut toutefois se produire une erreur si l'un des paramètres n'est pas accessible, ou que l'un des paramètres n'est pas conforme à ce qu'attend le service d'alignement.

(align-access-denied)

$$\frac{X - \text{request}(\text{align}(O, O', a, P)) \rightarrow SA}{SA - \text{refuse}(\text{inaccessible}(R)) \rightarrow X}$$

R est la ressource inaccessible (R est soit O, soit O', soit a, soit P)

(align-bad-set)

$$\frac{X - \text{request}(\text{align}(O, O', a, P)) \rightarrow SA}{SA - \text{failure}(\text{inconsistent-set}(P)) \rightarrow X}$$

O et O' sont accessibles, P n'est pas conforme pour SA

5.2.2 Trouver une ontologie

Ce service permet à un agent de trouver une ontologie qui soit suffisamment proche de la sienne (similaire) pour qu'il puisse s'en servir lors de la reformulation d'un message. En effet, un agent peut décider de ne pas dévoiler l'ontologie qu'il utilise, que ce soit pour des raisons de confidentialité ou d'accessibilité ; il doit alors trouver une ontologie qui lui permette de formuler son message en utilisant les concepts les plus proches possible des siens.

De plus, même si un service d'alignement est libre dans sa recherche d'une telle ontologie, il est conseillé d'inclure un sous-protocole du type contract-net qui permette l'interrogation d'autres acteurs du système, par l'intermédiaire d'un appel d'offre (CFP "Call For Proposal"). Cela permet de maximiser les chances de trouver une ontologie qui convienne à l'agent qui a émis la requête.

Les paramètres de cette fonction sont les suivants : une ontologie O et un ensemble T de termes de O que l'agent souhaite utiliser dans son prochain message ; cet ensemble permet de guider la recherche. Notons que T peut être un ensemble vide, mais cela diminue les chances de trouver une ontologie pour la réécriture d'un message, tout en augmentant le temps de recherche.

(search-success)
$$\frac{X - \text{request}(\text{find}(O, T)) \rightarrow SA}{SA - \text{inform}(\textit{match}(O, T)) \rightarrow X}$$
O est accessible, $\forall t \in T, t \in O$

(search-access-denied)
$$\frac{X - \text{request}(\text{find}(O, T)) \rightarrow SA}{SA - \text{refuse}(\text{inaccessible}(O)) \rightarrow X}$$
O est inaccessible

(search-bad-set)
$$\frac{X - \text{request}(\text{find}(O, T)) \rightarrow SA}{SA - \text{failure}(\text{inconsistent-set}(T)) \rightarrow X}$$
O est accessible, $\exists t \in T / t \notin O$

5.2.3 Traduction d'un message

La traduction d'un message M peut se faire de deux manières : soit en précisant l'ontologie à partir de laquelle le message M a été exprimé ainsi qu'un alignement, soit en précisant deux ontologies (la première étant celle qui a été utilisée pour exprimer le message M).

Traduction à partir d'un alignement :

Dans ce cas de figure, la requête est faite à partir du message M, de l'ontologie O et d'un alignement a. Si tous les concepts du message M apparaissent dans l'ontologie O alors le message peut être traduit. Précisons que la référence à l'ontologie O permet de rechercher les concepts de M dans l'alignement (a), et ainsi d'identifier les concepts équivalents dans l'autre ontologie (celle qui n'est pas O dans l'alignement).
Si un concept de O présent dans M n'apparaît pas dans l'alignement (a), alors le message de réponse fera état de cette incohérence dans les données.

(trans-align-success) $\dfrac{X - request\ (\ tsl\ (\ M, O, a))\ \rightarrow SA}{SA - inform\ (\ \textit{translate}\ (\ M, O, a\ )\ ) \rightarrow X}$ $\forall\ e \in M\ /\ e \in a$

(trans-align-bad-set) $\dfrac{X - request\ (\ tsl\ (\ M, O, a))\ \rightarrow SA}{SA - refuse\ (\ inconsistent\text{-}set\ (a)\ ) \rightarrow X}$ $\exists\ e \in M\ /\ e \notin a$

Traduction à partir des ontologies :

Dans ce cas de figure, un problème peut survenir quant à l'accessibilité d'une ontologie. Signalons que si les deux ontologies sont inaccessibles, il n'est pas nécessaire d'envoyer deux fois un message comme celui de la règle (trans-onto-access-denied). Si au moins une ontologie n'est pas accessible, alors la traduction ne peut en aucun cas se faire ; il faut alors trouver une autre ontologie et recommencer la traduction.

(trans-onto-success) $\dfrac{X - request\ (\ tsl\ (\ M, O, O'))\ \rightarrow SA}{SA - inform\ (\textit{translate}\ (\ M, O, O'\ ))\ \rightarrow X}$ O et O' sont accessibles

(trans-onto-access-denied) $\dfrac{X - request\ (\ tsl\ (\ M, O, O'))\ \rightarrow SA}{SA - refuse\ (\ inaccessible\ (\ O^*\ )\ ) \rightarrow X}$ O* est inaccessible (O* est soit O soit O')

5.2.4 Obtenir un programme de traduction

Un programme de traduction donne à un agent la possibilité de pouvoir traduire un message à n'importe quel moment, sans repasser par un service d'alignement : l'exécution du programme permet de convertir directement, et localement, un message M exprimé en fonction d'une ontologie O en un message M' exprimé en fonction d'une ontologie O'.
Du point de vue d'un agent, la requête d'un tel programme peut se faire soit directement à partir d'un alignement (a), soit à partir de deux ontologies O et O'.
Rappelons que le langage lg dans lequel est exprimé le programme de traduction peut être n'importe quel langage de programmation, par exemple java, xslt ou visual basic.

Programme à partir d'un alignement :

Dans les règles suivantes, lg représente le langage dans lequel doit-être exprimé le programme de traduction que X demande ; SA(L) représente l'ensemble L associé à SA, c'est-à-dire l'ensemble des langages que SA supporte, et l'appel de *program*(a, lg) retourne le programme de traduction lui-même.
De plus, si le langage (lg) n'est pas connu de SA, alors ce dernier le fait savoir à X par un message approprié.

(get-prog-align-success)
$$\frac{X - \text{request}(\text{get-prog}(a, lg)) \rightarrow SA}{SA - \text{inform}(\textit{program}(a, lg)) \rightarrow X} \quad lg \in SA(L)$$

(get-prog-align-bad-lg)
$$\frac{X - \text{request}(\text{get-prog}(a, lg)) \rightarrow SA}{SA - \text{refuse}(\text{not-understood}(lg)) \rightarrow X} \quad lg \notin SA(L)$$

Programme à partir des ontologies :

Ce cas de figure est très proche du cas précédent. La requête d'un programme se fait directement à partir d'un couple d'ontologies et non plus d'un alignement ; un cas supplémentaire peut donc se produire : celui d'une ontologie inaccessible.

(get-prog-onto-success)
$$\frac{X - \text{request}(\text{get-prog}(O, O', lg)) \rightarrow SA}{SA - \text{inform}(\textit{program}(O, O', lg)) \rightarrow X} \quad lg \in SA(L),\ O \text{ et } O' \text{ accessibles}$$

(get-prog-onto-bad-lg)
$$\frac{X - \text{request}(\text{get-prog}(O, O', lg)) \rightarrow SA}{SA - \text{refuse}(\text{not-understood}(lg)) \rightarrow X} \quad lg \notin SA(L)$$

(get-prog-onto-bad-access)
$$\frac{X - \text{request}(\text{get-prog}(O, O', lg)) \rightarrow SA}{SA - \text{refuse}(\text{inaccessible}(O^*)) \rightarrow X} \quad lg \in SA(L),\ O^* \text{ est inaccessible } (O^* \text{ est soit O soit O'})$$

5.2.5 Obtenir un ensemble d'axiomes

Ce service permet à un agent de demander l'ensemble des axiomes correspondant à un alignement ou à une paire d'ontologies (voir le chapitre 4 / génération d'axiomes pour plus d'informations sur les axiomes).

Axiomes à partir d'un alignement :

Les deux cas qui peuvent se produire dépendent de l'alignement lui même : si celui-ci est vide alors aucun axiome ne peut être produit.

(get-axioms-align-success) $\dfrac{\text{X – request (get-axioms(a))}\rightarrow\text{SA}}{\text{SA – inform (}\textit{axioms (a)}\text{)} \rightarrow \text{X}}$ $a \neq \emptyset$

(get-axioms-align-bad-set) $\dfrac{\text{X – request(get-axioms(a))}\rightarrow\text{SA}}{\text{SA – failure (inconsistent-set (a))} \rightarrow \text{X}}$ $a = \emptyset$

Axiomes à partir d'une paire d'ontologies:

Ce cas de figure où sont utilisées directement des ontologies, un problème d'accessibilité est toujours possible.

(get-axioms-onto-success) $\dfrac{\text{X – request (get-axioms(O, O'))}\rightarrow\text{SA}}{\text{SA – inform (}\textit{axioms (O, O')}\text{)} \rightarrow \text{X}}$ O et O' accessibles

(get-axioms-onto-bad-access) $\dfrac{\text{X – request(get-axioms(O,O'))}\rightarrow\text{SA}}{\text{SA – refuse (inaccessible (O*))} \rightarrow \text{X}}$ O* est inaccessible, O* est soit O soit O'

5.2.6 Ajouter un alignement à la bibliothèque

Rappelons tout d'abord que SA(A) est l'ensemble des alignements dont SA dispose dans sa bibliothèque (ensemble A de SA).

Dans ce cas de figure, puisque l'alignement a n'appartient pas aux alignements que SA considère comme étant associés aux ontologies O et O', SA l'ajoute à son ensemble A de triples et répond à X que l'alignement a bien été mémorisé :

(declare-align-success) $\dfrac{X - \text{request (declare (O, O', a))} \rightarrow SA}{\begin{array}{c} SA(A) := SA(A) \cup \langle O,O',a\rangle, \\ SA - \text{confirm (O, O', a)} \rightarrow X \end{array}}$ $\langle O,O',a\rangle \notin SA(A)$, *verifSources*(a, O, O')

Si le triplet est déjà connu de SA, alors SA envoie le même message que dans le cas précédent ; cela revient à confirmer à X que SA(A) contient bien le triple initialement transmis.

(declare-align-existing) $\dfrac{X - \text{request (declare (O, O', a))} \rightarrow SA}{SA - \text{confirm (O, O', a)} \rightarrow X}$ $\langle O,O',a\rangle \in SA(A)$

Si l'alignement transmis ne correspond pas aux ontologies également transmises, alors la règle suivante sera déclanchée :

(declare-align-bad-set) $\dfrac{X - \text{request (declare (O, O', a))} \rightarrow SA}{SA - \text{refuse (inconsistent-set (a))} \rightarrow X}$ $\langle O,O',a\rangle \notin SA(A)$, not(verifSources(a,O,O'))

5.3 Services communs aux services d'alignement et aux agents

5.3.1 Tester l'existence d'un alignement

Ce service permet à un agent ou à un service d'alignement de savoir si un autre agent ou service d'alignement possède déjà un alignement entre deux ontologies données. Cela permet de ne pas re-générer un alignement alors qu'un « bon » alignement existe peut-être.

Dans les règles suivantes, X représente soit un agent soit un service d'alignement, et S peut être soit un service d'alignement soit un agent. S(A) est l'ensemble A des alignements de S. S(SA) est l'ensemble des services d'alignement dont S stocke les références. Et < O, O', _ > est l'ensemble des alignements associés aux ontologies O et O'.

Dans le premier cas de figure, si un alignement existe, alors une référence de l'alignement en question est directement transmise avec la réponse.

(test-align-success)
$$\frac{X - \text{query-if (is-align (O, O')) } \rightarrow S \qquad \langle O, O', \_ \rangle \in A(S)}{S - \text{inform (} \textit{getAlign}(O,O') \text{) } \rightarrow X}$$

Si le couple d'ontologies (O, O') est complètement inconnu, alors le test d'existence échoue et la valeur FAUX doit être transmise en réponse :

(test-align-unknown)
$$\frac{X - \text{query-if (is-align (O, O')) } \rightarrow S \qquad \langle O, O', \_ \rangle \notin A(S), \ \langle O, O', \_ \rangle \notin SA(S)}{S - \text{inform (false)} \rightarrow X}$$

Le dernier cas de figure permet, en cas d'absence d'alignement, d'obtenir la référence d'un service d'alignement qui possède au moins un alignement entre O et O', ou qui est capable d'en produire un. Comme nous l'avons évoqué plus haut, les conditions de mémorisation de ces références ne sont pas traitées par ce travail.

(test-align-forward)
$$\frac{X - \text{query-if (is-align (O, O')) } \rightarrow S \qquad \langle O, O', \_ \rangle \notin A(S), \ \langle O, O', \_ \rangle \in SA(S)}{S - \text{inform(forward(}\textit{getService}(O,O')\text{))} \rightarrow X}$$

5.3.2 Obtenir un alignement

L'obtention directe d'un alignement, survenant préférentiellement juste après un test d'existence (voir le cas précédent), permet de récupérer un alignement existant auprès d'un autre agent ou service d'alignement.

Dans les règles suivantes, X représente soit un agent soit un service d'alignement, et S peut également être soit un service d'alignement soit un agent.

Dans ce premier cas de figure, un alignement existe :

(get-align-success) $$\frac{X - \text{request}(\text{get-align}(O, O')) \rightarrow S \quad \langle O, O', \_ \rangle \in S(A)}{S - \text{inform}(\textit{getAlign}(O,O')) \rightarrow X}$$

Dans le deuxième cas de figure, aucun alignement ne correspond au couple d'ontologies (O,O') ; un message adapté doit donc être transmis en réponse :

(get-align-unknown) $$\frac{X - \text{request}(\text{get-align}(O, O')) \rightarrow S \quad \langle O, O', \_ \rangle \notin S(A)}{S - \text{refuse}(\text{unkown-pair}(O,O')) \rightarrow X}$$

5.4 Synthèse

Comme nous l'avons expliqué jusqu'à présent, le protocole que nous avons présenté est à la fois modulaire et flexible : n'importe quel agent, à son tour, peut demander à un service d'alignement d'effectuer une opération qu'un autre agent aurait pu tout aussi bien commander, à son propre tour. Les opérations sont atomiques et le protocole laisse donc toute liberté aux agents de choisir leur façon de gérer les alignements. De plus il n'y a aucune contrainte pour un agent ; tout arrêt de suivi du protocole est transparent et celui-ci peut être repris ultérieurement. Enfin, l'intégration dans notre approche de la production de programmes de traduction permet d'accélérer et d'automatiser les traductions.

Chapitre 6

Scénario d'utilisation

Dans ce chapitre nous considérons un cas d'utilisation relativement simple mais qui se veut réaliste, sans toutefois être implantable directement. Nous partons en effet d'hypothèses relatives à l'environnement technique dans lequel se déroule le scénario, ainsi que d'hypothèses par rapport au niveau applicatif se trouvant en amont de l'exécution de notre protocole. Nous ne détaillerons donc ni les protocoles de communication sans fil évoqués, ni l'utilisation des applications servant à saisir des données.
Signalons que nous ne considérons ici que des agents de niveau 1 par rapport aux contraintes d'utilisation du protocole : dans cet exemple simple aucun agent ni service d'alignement n'a besoin d'être mémorisé.
Précisons enfin que les exemples de code ou pseudo-code présentés au cours de ce chapitre ont pour but d'illustrer notre propos ; la syntaxe, par exemple du contenu des messages, pourra ne pas être rigoureusement valide pour le langage considéré.

6.1 Les acteurs du systèmes

Nous présentons ici les différentes entités intervenant au sein du scénario, ainsi que leurs objectifs.

- Un agent A, matérialisé par l'utilisation d'un PDA (i.e. assistant personnel) supportant le développement d'applications communicantes via un protocole de communication, éventuellement sans fil du type IrDA (infra-rouge) ou Bluetooth.

- Un agent B, matérialisé par un ordinateur portable capable de communiquer avec l'agent A. L'agent B doit également pouvoir communiquer avec un service d'alignement SA distant (serveur S), soit par l'intermédiaire d'une connexion avec un téléphone mobile, soit par l'intermédiaire d'un tout autre type de connexion permettant cette communication. Notons également que cette machine est équipée d'une plateforme PHP (qui sera utilisée pour exécuter un script de traduction dans notre exemple).

- Un service d'alignement SA, matérialisé par un serveur S relié à une plateforme de réception de messages aériens (type GSM...). Ce serveur représente un service d'alignement connu de B.

Dans le cas du scénario que nous présentons ici, l'agent A souhaite obtenir des compléments d'information à propos de ressources relatives au monde de la recherche, qu'il s'agisse des détails d'une conférence à venir, ou de publications.
De plus nous considérons que l'agent B évolue dans le même domaine de connaissances, mais se base sur une autre ontologie pour échanger des informations liées au domaine de la recherche. Nous présentons les ontologies manipulées par ces deux agents dans le paragraphe suivant.

6.2 Les ontologies utilisées

Nous présentons ici les deux ontologies utilisées par les agents A et B, respectivement, pour exprimer le contenu de leurs messages quand ils communiquent au sujet de documents, et plus particulièrement de documents ayant un sens dans le domaine de la recherche : il s'agit d'un domaine permettant une grande utilisation de la variété des types de documents relatifs au monde de l'édition.

Les ontologies présentées dans les figures 10 et 11 sont des versions simplifiées d'ontologies développées par Nick Knouf (MIT), Antoine Zimmermann (INRIA RA) et Jérôme Euzenat (INRIA RA) pour décrire des entrées bibTeX.

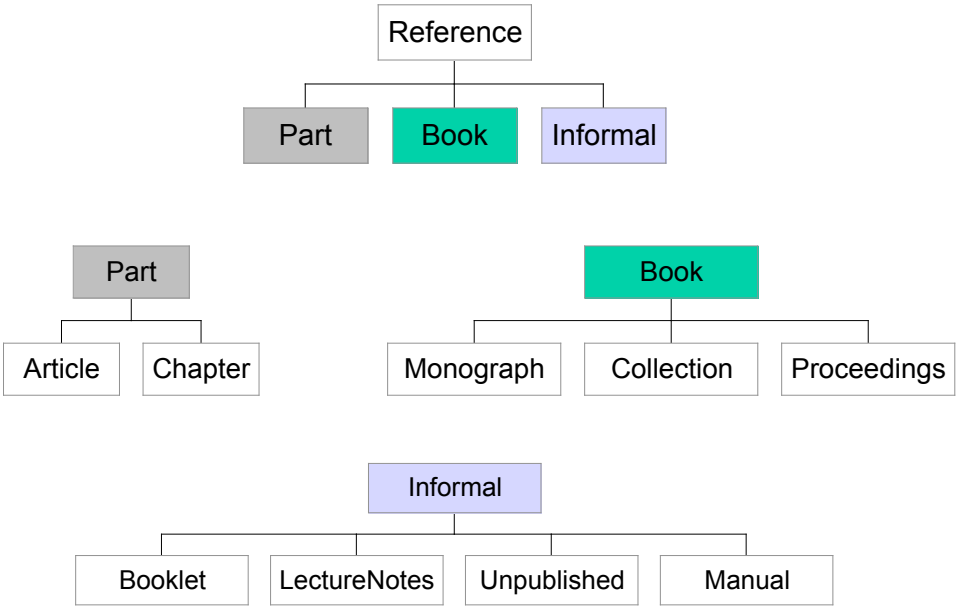


FIG 10. Ontologie de l'agent A, représentée de manière décomposée.
Chaque concept représenté possède des attributs
qui ne sont eux mêmes pas représentés.

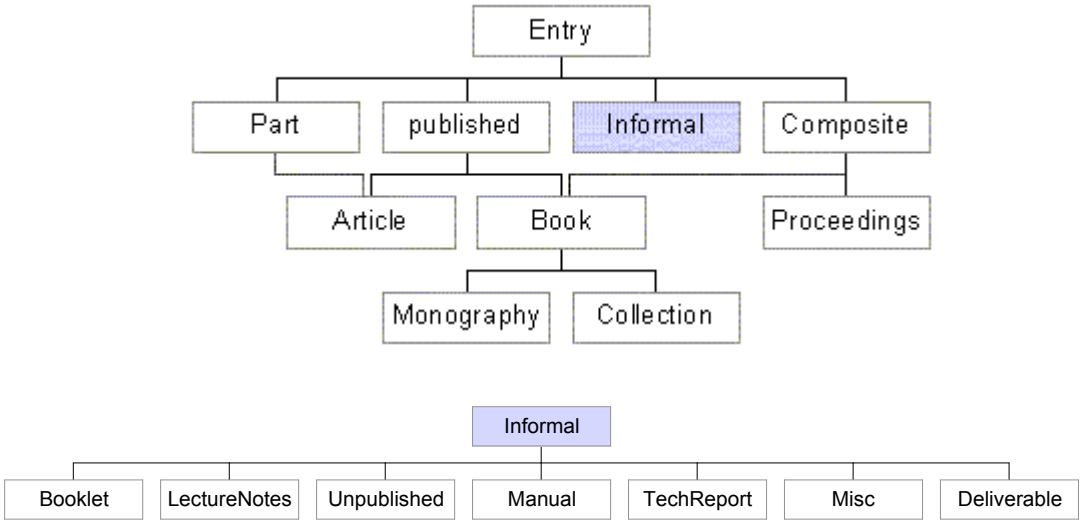


FIG 11. Ontologie de l'agent B, représentée de manière décomposée.
Chaque concept représenté possède des attributs
qui ne sont eux mêmes pas représentés.

6.3 Déroulement du scénario

Nous étudions dans ce paragraphe une communication entre l'agent A et l'agent B, au cours de laquelle notre protocole PCAC sera amené à être utilisé. Comme nous l'avons précisé dans l'introduction, les aspects techniques de la communication, comme la synchronisation des appareils ou le protocole d'échange de données, ne sont pas abordés.

- Etape 1

L'agent A crée une nouvelle entrée dans sa base de connaissances, suite à l'ajout manuel de données concernant les actes d'une conférence. La nouvelle instance est toutefois incomplète car toutes les données n'étaient pas connues au moment de la saisie.

- Etape 2

L'agent A est mis en contact avec l'agent B afin d'obtenir plus d'informations : les deux interlocuteurs s'identifient.

- Etape 3

L'agent A initie le dialogue dans le but d'obtenir les informations manquantes. Il envoie le message suivant à l'agent B (exprimé en FIPA ACL, que l'on suppose connu des 2 agents) :

```
(QUERY-REF
 :sender A
 :receiver B
 :content
   Proceedings(
    title('Proc of the 1st European Semantic Web Symposium'),
    Date(year('2004')),
    publisher(X)
   )
 :language Prolog
 :ontology http://domaine-de-A/onto-de-A.owl
 :reply-with id-123456
)
```

- Etape 4a

Si l'agent B ne supporte pas notre protocole et ne peut pas interpréter le message, deux options s'offrent à lui : soit il ne répond pas et l'agent A aura le choix entre réessayer d'envoyer un message ou abandonner, soit il a par défaut la faculté de répondre par un message du type NOT-UNDERSTOOD ou FAILURE.

- Etape 4b

Dans ce cas de figure l'agent B va tenter de "comprendre" le message en exploitant le protocole. Il se connecte alors au serveur S pour utiliser le service d'alignement SA : il demande un alignement entre son ontologie (onto-de-B.owl) et celle communiquée par l'agent A (règle **align-*xxx*** du protocole).
Cette requête peut être représentée, de manière informelle, par le code suivant :

```
Alignment AL ;
AlignmentParameters P ;

AL = null ;
P.algorithm = SA.LEXICAL_DISTANCE_WITH_ATTRIBUTES ;
P.threshold = 0.8 ;
P.terms = null ;

Id = SendMessage(
            B, SA, request,
            align ( http://domaine-de-B/onto-de-B.owl,
                    http://domaine-de-A/onto-de-A.owl,
                    P,
                    AL )
            )
```

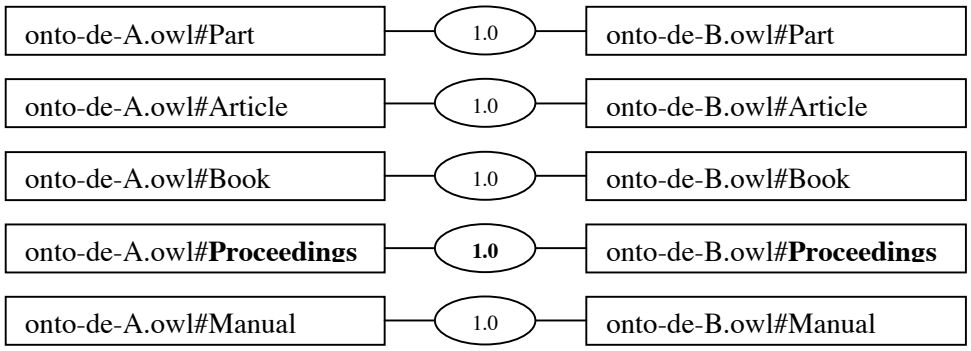
Dans le pseudo-code précédent :

A et P sont des objets représentant respectivement un alignement et un ensemble de paramètres d'alignement.

SendMessage est une primitive représentant l'envoi d'un message FIPA-ACL de B vers SA, du type *request* ; le contenu est représenté par le paramètre suivant le type. *SendMessage* retourne un identifiant de message, celui qui a été transmis à SA.

- Etape 5

SA calcul un alignement en fonction des paramètres qui lui ont été transmis. Nous considérons que l'alignement est produit et renvoyé à B sans problème (cas de la règle **align-success** du protocole) ; l'alignement est mémorisé par l'objet AL présenté précédemment.
L'alignement obtenu peut être schématiser ainsi (liste des couples non exhaustive) :



Notons que les noms de concepts des deux ontologies sont très similaires. Un algorithme simple d'alignement, et ne prenant éventuellement pas en compte les attributs des concepts, suffirait pour obtenir un alignement exploitable immédiatement. Signalons également que les couples reliant 'Book' et 'Booklet' pourraient apparaître dans l'alignement si le seuil de similarité requis était inférieur à celui de ces couples (autour de 0.57 pour 4 lettres en commun sur les 7 que comporte 'Booklet').

- Etape 6

L'alignement obtenu par l'agent B est satisfaisant, selon lui, pour interpréter le message envoyé par A. Il estime cependant (pour des raisons non précisées ici) qu'il sera amené à interagir de nouveau avec l'agent A. Il demande donc au service SA un programme de traduction de messages basé sur l'alignement qu'il vient d'obtenir.
B étant capable d'interpréter des scripts PHP, et sachant que SA peut générer de tels scripts, il émet donc la requête suivante (règle **get-prog-align-*xxx***) :

```
Id = SendMessage(
      B, SA, request, get-prog ( AL, 'PHP' )
     )
```

- Etape 7

SA renvoie le script PHP correspondant sans poser de problème (cas de la règle **get-prog-align-success** du protocole) : il s'agit d'un fichier que l'on nommera **scr-B-A.php**.

- Etape 8

L'agent B peut désormais convertir les messages exprimés par l'agent A, en fonction de l'ontologie onto-de-A, en des messages qu'il peut directement interpréter, c'est-à-dire exprimés en fonction de son ontologie onto-de-B. Signalons que la traduction inverse est possible si le programme de traduction a été conçu pour la permettre.

La traduction proprement dite peut alors se faire par un appel à la page PHP, interprétée par le moteur PHP de l'agent B, ressemblant au code suivant :

```
scr-B-A.php?message="Proceedings( title('Proc of the 1st
European Semantic Web Symposium'), Date(year('2004')),
publisher(X) )"
```

La page retournée suite à cet appel contient le nouveau message ; dans cet exemple le nouveau message est similaire à l'ancien puisque les concepts étaient identiques. Cela permet néanmoins à l'agent B d'interpréter le message en fonction de son ontologie.

- Etape 9

Nous considérons ici que B reconvertit la réponse qu'il obtient suite à l'interprétation du message. Il pourra donc répondre à A en lui envoyant un message exprimé en fonction de onto-de-A.
Signalons également que nous considérons *publisher* comme un attribut qui ne pose pas de problème d'interprétation.

- Etape 10

L'agent B répond donc à l'agent A par l'envoi du message suivant :

```
(INFORM
 :sender B
 :receiver A
 :content
   X = (    Name('Springer-Verlag' ),
            Address(    city('Heidelberg' ),
                        country('DE') )
       )
 :language Prolog
 :ontology http://domaine-de-A/onto-de-A.owl
 :in-reply-to id-123456
)
```

L'agent A peut alors mettre à jour sa base de connaissances. L'interaction s'est terminée avec succès.

6.4 Synthèse

Bien que l'exemple d'alignement présenté ici paraisse simple, rappelons qu'il revient à l'algorithme d'alignement d'identifier des couples cohérents de concepts entre ontologies, en fonction de leurs noms, de leurs attributs ou encore des relations qui lient ces concepts. Le protocole que nous avons introduit et présenté tout au long de ce mémoire, et mis en situation dans ce chapitre, permet de guider le partage de connaissances entre agents et donc la communication entre entités cognitives conçues indépendamment les unes des autres. Notons enfin que la communication présentée ici étant asynchrone, et les messages identifiés, tout échec d'une opération de la part d'un acteur du système ne met en aucun cas le système global lui-même en péril.

Conclusion

Bilan

Nous avons présenté un protocole de communication entre agents qui peut s'insérer de manière transparente dans un échange entre agents. Ce protocole, qui se base sur les travaux d'alignements d'ontologies et l'API [Euzenat, 2004], permet d'aider les agents à se retrouver dans une situation où un message venant d'un autre agent peut être interprété.

Nous avons pour cela défini et utilisé la notion de service d'alignement, interface à des bibliothèques d'alignement. Ces services permettent aux agents d'invoquer des services élémentaires comme l'alignement de deux ontologies, la production d'axiomes ou la traduction de message.

Perspectives

L'implémentation et la validation de ce protocole restent encore à accomplir. Son implémentation est toutefois directe grâce au système de règles que nous avons présenté en guise de spécification du protocole. De plus, afin de faire évoluer le protocole, il serait intéressant de considérer que les règles puissent véhiculer des références aux ontologies et aux alignements au lieu de les transmettre directement, afin de pouvoir optimiser le trafic des informations échangées. On pourra de plus considérer les aspects conversationnels également au niveau des règles : transmettre un identifiant de message permet de traiter une réponse correctement quel que soit le moment où cette réponse arrive ; et cela sans poser d'hypothèses sur le respect du modèle de communication utilisé. Enfin, l'impact du niveau d'intégration du protocole pourra être plus amplement explicité en apparaissant dans le système de règles.

D'un point de vue plus général, il est souhaitable de concevoir de nouvelles modalités d'interactions entre les agents et les services d'alignement pour obtenir de meilleurs alignements, et surtout pouvoir les utiliser de manière efficace au sein de la communication entre agents cognitifs.

Remerciements

Je remercie Jérôme Euzenat et Yves Demazeau pour leur soutien et qui, par leurs conseils, m'ont permis de faire aboutir ce projet. Je remercie également les personnes dont j'ai croisé la route à l'INRIA Rhône-Alpes : le personnel et les stagiaires !

Références

[AUML] Agent Unified Modelling Language. http://www.auml.org

[Austin, 1962] J.L. Austin. How to do things with words. Clarendon Press, Oxford, 1962.

[Bailin and Truszkowski, 2002] Sidney C. Bailin and Walt Truszkowski. Ontology Negotiation: How agents can really get to know each other.

[Bechhofer et al., 2003] Sean Bechhofer, Rapahel Voltz, and Phillip Lord. Cooking the semantic web with the OWL API. In Proc. 2nd International Semantic Web Conference (ISWC), Sanibel Island (FL US), 2003.
http://potato.cs.man.ac.uk/papers/cooking03.pdf
API : http://sourceforge.net/projects/owlapi

[Demazeau, 1995] Yves Demazeau. From interactions to collective behaviour in agent-based systems. European Conference on Cognitive Science, Saint-Malo – France, Avril 1995.

[Euzenat, 1997] Jérôme Euzenat. A protocol for building consensual and consistent repositories. Rapport de recherche 3260, INRIA Rhône-Alpes, Grenoble (FR), septembre 1997, 46p.

[Euzenat, 2004] Jérôme Euzenat. An API for ontology alignment. In Proc. 3rd International Semantic Web Conference (ISWC), Hiroshima (JP). A paraître 2004.

[Euzenat and Valtchev, 2003] Jérôme Euzenat, Petko Valtchev. An integrative proximity measure for ontology aligment. In Proc. 2nd International Semantic Web Conference (ISWC). Workshop on semantic information integration, Sanibel Island (FL US), pp33-38, 2003.

[Euzenat and Valtchev, 2004] Jérôme Euzenat and Petko Valtchev. Similarity-based ontology alignment in OWL-lite. In Proc 15th european conference on artificial intelligence (ECAI), Valencia (ES), 2004.

[Ferber, 1995] J. Ferber. Les systèmes multi-agents. Vers une intelligence collective, Editions InterEditions, 1995.

[FIPA] Foundation for Intelligent Physical Agents. http://www.fipa.org

[FIPA DIAG] FIPA Modeling : Interaction Diagrams. http://www.fipa.org
voir aussi : http://www.auml.org/auml/documents/ID-03-07-02.pdf

[FIPA DF] FIPA Directory Facilitator. http://www.fipa.org, Reference : fipa00023.

[Gruber, 1995] Tom R. Gruber. Towards principles for the Design of Ontologies Used for Knowledge Sharing. International Journal of Human and Compute Studies. Vol43 907-928

[KQML] Tim Finin, Yannick Labrou, and J. Mayfield. KQML (Knowledge Query and Manipulation Language) as an agent communication language. In J. M. Bradshaw, editor, Software Agents. MIT Press, 1997.
http://www.cs.umbc.edu/kqml/papers/kqmlacl.pdf

[Porter, 1980] Porter. An algorithm for suffix stripping, Program, Vol. 14, no. 3, pp 130-137 – 1980. http://www.tartarus.org/~martin/PorterStemmer

[Searle, 1969] J.R. Searle. Speech Acts. Cambridge University Press, 1969

[Smith et al., 2003] M.K. Smith, C. Welty, and D. McGuinness. Web Ontology Language guide. http://www.w3.org/TR/owl-guide/, 2003

[Steels, 1996] L. Steels. Emergent adaptive lexicons. In P. Maes, editor, From Animals to Animats 4: Proceedings of the Fourth International Conference On Simulating Behavior, Cambridge Ma., 1996. The MIT Press.

[Stuckenschmidt and Timm, 2002] Heiner Stuckenschmidt & Ingo J. Timm. Adapting Communication Vocabularies using Shared Ontologies. AAMAS workshop OAS, 2002

[UDDI] Universal Description, Discovery and Integration. http://www.uddi.org

[W3C, 1999] Resource Description Framework. http://www.w3.org/RDF/

[W3C, 2004a] Web Ontology Language.
http://www.w3.org/TR/2004/REC-owl-features-20040210

[W3C, 2004b] RDF Schema. http://www.w3.org/TR/2004/REC-rdf-schema-20040210/

[Wang and Gasser, 2002] Jun Wang and Les Gasser. Mutual Online Ontology Alignment. AAMAS workshop OAS, 2002. http://cis.otago.ac.nz/OAS2002/Papers/oas02-22.pdf

[Wiesman et al., 2001] F. Wiesman, N. Roos et P. Vogt. Automatic ontology mapping for agent communication. MERIT-Infonomics Research Memorandum series, 2001

[Zou et al., 2003] Youyong Zou, Tim Finin, Li Ding, Harry Chen, Rong Pan. Using semantic web technology in multi-agent systems: a case study in the TAGA trading agent environment. Proceedings of the 5th international conference on Electronic commerce. Pages: 95 – 101, 2003.