REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université TAHRI Mohammed Béchar
Faculté des Sciences Exactes
Département des Mathématiques et Informatique

N° d'ordre : UTMB/FSE/**PI**

Filière : Informatique

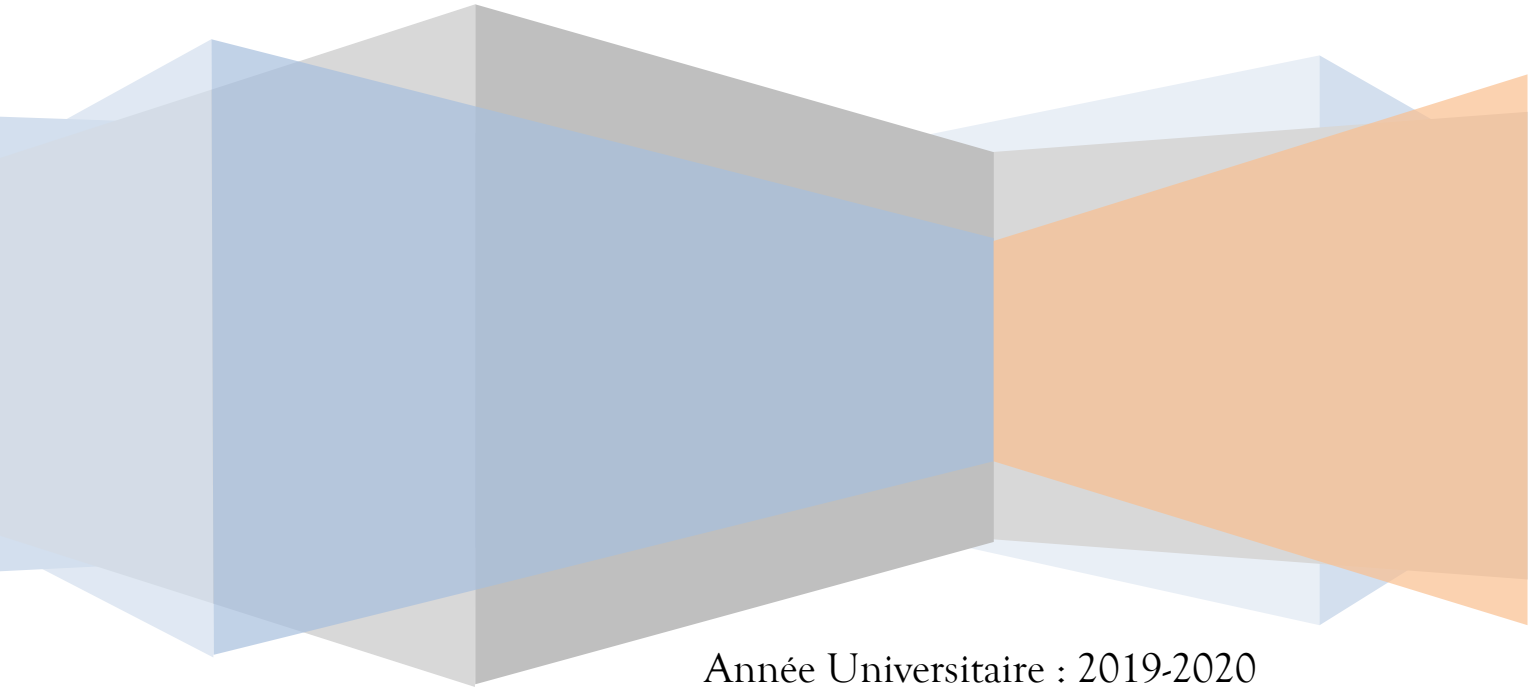
Spécialités : MSIA - MSIR - MIAD

Matières : SI - SMA

INGENIERIE DES SYSTEMES INTELLIGENTS

Théorie & Pratique

GHAZLI Abdelkader



Année Universitaire : 2019-2020

PREFACE

Ce polycopié a pour objectif de présenter les notions de bases et les différents concepts nécessaires pour construire des systèmes intelligents. Il s'adresse aux étudiants qui sont intéressé pour comprendre les notions fondamentales de l'intelligence artificielle, l'intelligence artificielle distribuée et les systèmes multi agents.

Ce polycopié présente une partie théorique et une autre partie très importante qui est la partie pratique inclut un ensemble important des exemples et des codes de programmation développés généralement en java.

Ce polycopié présente un support de cours de tous les spécialités master en informatique de notre université, la spécialité Systèmes d'Informations Avancées, la spécialité Systèmes Informatiques et Réseaux ainsi que la spécialité Intelligence Artificielle et Décision et spécialement dans les matières Systèmes intelligents 1, Systèmes intelligents 2 et Systèmes Multi Agents.

LISTE DES FIGURES

LISTE DES TABLES

LISTE DES ABREVIATIONS

DARPA	Defense Advanced Research Projects Agency
KIF	Knowledge Interchange Format
SUO KIF	Standard Upper Ontology Knowledge Interchange Format
XML	Extensible Markup Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
OWL	Ontology Web Language
OWL Lite	Lite Ontology Web Language
OWL DL	Ontology Web Language Description Logics
OWL Full	Full Ontology Web Language
URI	Uniform Resource Identifier

SOMMAIRE

Chapitre I

Intelligence Artificielle

Chapitre I : Intelligence Artificielle

I.1 Définition de l'Intelligence

L'intelligence est la capacité d'apprendre ou de comprendre grâce à l'expérience. C'est aussi la capacité d'acquise et de retenir des connaissances.

Elle désigne aussi la capacité mentale ou la capacité de répondre rapidement et de manière appropriée à une nouvelle situation. Autrement dit, c'est l'utilisation du raisonnement pour résoudre des problèmes. **[1]**

L'intelligence peut permettra à un individu de résoudre un problème ou de s'adapter à son environnement. Selon Allen Newell, c'est la capacité de produire et de manipuler des structures symboliques.

I.2 Définition de l'Intelligence Artificielle

L'intelligence artificielle ou encore IA, est un ensemble d'algorithmes conférant à une machine des capacités d'analyse et de décision lui permettant de s'adapter intelligemment aux situations en faisant des prédictions à partir de données déjà acquises.

Le but de l'intelligence artificielle est l'étude de la structure de l'information et de la structure de processus de résolution de problèmes indépendamment des applications et indépendamment d'une réalisation (John McCarty)

L'intelligence artificielle peut être vue comme un sous ensemble de programmes ou logiciels traitant de problèmes :

- De nature symbolique, ne nécessitant pas ou peu de calculs numériques
- N'ayant pas de solution algorithmique possible ou satisfaisante

I.2.1 Buts de l'IA

Le principal but de l'intelligence artificielle est de simuler le comportement intelligent de l'être humain. C'est la recherche de moyens (algorithmes, techniques, approches…) pour doter les machines et les systèmes informatiques de capacités intellectuelle similaires à celles des être humains.

Autrement dit, de rendre la machine intelligente, capable de prendre des décisions sont l'intervention directe d'un être humain.

I.2.2 Domaines d'applications de l'Intelligence Artificielle

Les domaines d'applications de l'intelligence artificielle sont vastes et divers, on peut citer : **[1]**

Compréhension

- De scènes : vision par ordinateur
- De parole : traitement de la parole
- De phrase : Analyse sémantique

Raisonnement

- Résolution des problèmes
- Démonstration des théorèmes
- Jeu et simulation

Réactions à l'environnement

- Apprentissage
- Prise de décisions à partir d'informations partielles

I.3 Connaissances

Pour que les entités (machines, agents…) peuvent communiquer et interchanger des messages entre eux c'est à dire de se faire comprendre, ils doivent donner une certaine représentation à leurs connaissances et se mettent d'accord sur le sens des différentes messages échangées.

Cognition : faculté de connaitre, activité mentales : perception, raisonnement, mémoire, représentation, apprentissage, langage, conscience.

En intelligence artificielle : ***Connaissance = Information +Sémantique***

I.3.1 Représentation des Connaissances

Une représentation des connaissances est un système qui définit une série de symboles et une série d'opérations sur ces symboles. Dans un système de représentation de connaissances on a besoin à la fois :

- d'un formalisme pour représenter les connaissances
- d'un mécanisme de raisonnement appelé souvent moteur d'inférence qui est capable d'enchainer des interfaces sur les représentations de ces connaissances

Newell a pu définir trois niveaux de représentation :

- **Niveau connaissance** : le niveau le plus abstrait qui comporte toutes les connaissances.
- **Niveau symbolique** : qui permet le codage des connaissances dans un langage formel.

- **Niveau implémentation :** qui présente la représentation interne des expressions (Structure des données : liste, tableau, matrice…)

Exemple :

Niveau connaissance : l'agent sait qu'il ya un avion de Béchar à Oran le 31-10-2018 à 8h30 qui coute 7000DA.

Niveau symbolique : PrixAvion (Béchar, Oran, 31-10-2018, 8h30, 7000DA)

Niveau implémentation : Table avec 5 colonnes : ville départ, ville arrivé, date, heure et prix.

I.4 Psychologie Cognitive

La psychologie cognitive n'est pas la seule discipline qui s'intéresse à l'esprit. Les disciplines qui partagent cet intérêt ont été rassemblé dans ce qu'il est maintenant courant appelé les sciences cognitives. Ces dernières cherchent à déterminer :

- Comment un système naturel (humain ou animal) ou artificiel (Robot) acquiert des informations sur le monde dans lequel il se trouve.
- Comment ces informations sont représentées et transformer en connaissances.
- Comment ces connaissances sont utilisées pour guider son attention et son comportement.

Les sciences cognitives rassemblent des contributions de plusieurs disciplines en citant la psychologie cognitive, la linguistique, les neurosciences et même la philosophie. D'autres disciplines peuvent êtres ajouté telles que la sociologie et l'anthropologie : science qui étudie l'être humain sous tous ses aspects à la fois physiques (anatomie, morphologiques, psychologiques, évolutifs…) et culturels (socioreligieux, psychologique, géographique …)

I.4.1 Définition

La psychologie cognitive est la sous discipline de la psychologie qui se focalise sur la cognition. Les sciences cognitives étudient donc l'intelligence ou comment on fait pour penser ?

La psychologie cognitive étudie les grandes fonctions psychologiques de l'être humain y compris la mémoire, le langage, l'intelligence, le raisonnement, la résolution de problèmes et la perception ou l'attention. **[2]**

I.4.2 Lien entre Connaissances et Raisonnement

La représentation formelle des connaissances ou des croyances permet d'automatiser divers traitements sur ces informations. C'est l'un des domaines de

recherche de l'intelligence artificielle. Le raisonnement est une Argumentation visant à établir une conclusion. On conduit des raisonnements pour accomplir quelques objectifs comme :

- Prise de décision
- Test d'une argumentation
- Vérification d'une proposition

Raisonnement : génération de nouvelles connaissances par inférence

I.4.3 Les Modes de Raisonnements

Un raisonnement est une suite de propositions vraies ou fausses liées les unes aux autres aboutissants à une conclusion. Il existe plusieurs types de raisonnements en citant :

Le Raisonnement Déductif : le raisonnement déductif va du général au particulier. Il tire des conséquences d'une loi, d'un principe, d'une règle générale et les appliquent à un cas particulier. C'est un raisonnement produisant des particularisations. **[3]**

Exemple : les hommes sont mortels, AEK est un homme, donc AEK est mortel.

Le Raisonnement Inductif : Le raisonnement inductif va du particulier au général, il envisage un cas précis pour en tirer les implications à un niveau général. C'est un raisonnement produisant des généralisations. **[2]**

Exemple 1: on établit d'abord un théorème pour n=1, on montre ensuite que s'il est vrai de n-1 il est vrai de n, on conclut qu'il est vrai pour tous les nombres entiers.

Exemple 2 : Analyse du sang ou de l'eau : on prend une petite quantité, on l'analyse, on généralise les résultats sur toute la quantité initiale.

Le Raisonnement par Elimination : ce type de raisonnement consiste à envisager toutes les solutions possibles pour répondre a une certaine situation ou problématique afin de montrer ensuite que sauf une qui répond vraiment et c'est la meilleure solution à garder.

Exemple : s'il existe plusieurs chemins pour atteindre une cible dont quelques routes sont très rapides, d'autres plus dangereux …etc. Le décideur commence par éliminer les routes qui ne satisfaisant pas ses demandes (rapidité, sécurité…) et enfin il va choisir un chemin après avoir éliminé tous les cas indésirables.

Le Raisonnement Causal : le raisonnement causal s'appuie sur les causes d'un fait, d'une situation ou d'un phénomène dans le but de tirer des conséquences. C'est un raisonnement qui sert de comprendre les faits toutes en cherchons ses causes.

Exemple : pourquoi pleut il : à cause de la vapeur d'eau condensé qui tombe des nuages.

Le Raisonnement Analogique : le raisonnement analogique établit un rapport entre deux domaines et en montre les ressemblances. Il procède à une comparaison avant d'aboutir une telle décision ou conclusion. **[3]**

Exemple : comparaison entre deux images : l'utilisateur commence à faire des ressemblances et s'il trouve des divergences il va conclure que les deux images à comparer ne sont pas similaires.

Le Raisonnement par l'absurde : ce type de raisonnement imagine des conséquences absurdes c'est à dire erronées d'une idée ou proposition afin de la contredire ou opposer pour montrer que cette proposition est fausse parce qu'elle conduit à des résultats erronés.

Exemple : Démontrer que Zéro n'a pas d'inverse : au premier lieu on suppose que 0 à un inverse. Si 0 à un inverse, alors il existe un réel a tel que a x 0=1 , Zéro est un absorbant et on aboutir à l'égalité 0=1 qui est une absurdité. Donc notre première proposition (Zéro à un inverse) est fausse et on conclu que zéro n'a pas d'inverse.

I.4.4 Représentation des Connaissances

La représentation des connaissances désigne un ensemble d'outils et de procédés destinés d'une part à représenter et d'autre part à organiser le savoir pour l'utiliser et le partager. Plusieurs outils sont utilisés pour représenter les connaissances en citant :

Les réseaux sémantiques : l'utilisation des réseaux sémantiques consiste à utiliser des graphes pour la représentation des connaissances. Cette représentation vient de l'idée de représenter graphiquement des concepts et leurs liens et relations. **[4]**

La logique : la logique est un outil très essentiel dans l'IA. C'est un langage constitué d'un formalisme de représentation des connaissances définit par un alphabet, des règles de constitution des phrases ou formules et un calcul de valeurs de vérité pour ces phrases.

Les études logiques des problèmes posés par l'intelligence artificielle ont mis en évidence d'autres types de logiques pour répondre à des situations ou la logique classique ne peut pas répondre ou être utilisé pour moduler ou représenter un tél système ou problème. Parmi ces logiques on trouve la logique floue, la logique modale, la logique temporelle, la logique non monotone, la logique épistémique… etc

Logique non monotone : joue un rôle très important en intelligence artificielle car elle élimine une partie de limitations de la logique classique dans le domaine de la représentation et du raisonnement en utilisant les connaissances d'une façon très proche de l'usage quotidien. **[4]**

Une logique non monotone est une logique formelle dont la relation des conséquences n'est pas monotone.

L'utilisation de la logique non monotone est proche du raisonnement humain, par un manque d'information ou manque de temps, on raisonne souvent avec des connaissances partielles. On révisé ces conclusions lorsqu'on a plus d'informations.

Exemple :

Si B est déductible à partir de A

Si p est ajouté au A

B n'est pas forcément toujours déductible du nouveau germe.

Ce qu'on note : A $\rightarrow$ B ce n'est pas équivalent à A, p $\not\rightarrow$B

La logique des défauts : est une logique non monotonique proposée par Rayon Reiter pour formaliser le raisonnement avec des prétentions de défaut. Contrarient à la logique classique qui peut seulement exprimer que quelque chose est vraie ou fausse, cette logique peut exprimer des faits comme « par défaut, quelque chose est vraie ».

La logique épistémique : c'est une logique qui traite la logique des connaissances d'agents pris individuellement. En logique épistémique, il ya plusieurs agents qui ont la capacité de raisonner en prenant en compte les connaissances qu'ils ont de certaines propositions ou les connaissances des autres agents. C'est typiquement la logique d'un joueur qui raisonne sur son jeu.

La logique temporelle : la logique temporelle est une logique modale, c'est à dire un système de déduction basé sur la logique classique à laquelle on rajoute des opérations. Pour permettre de suivre l'évolution temporelle d'un système, cette logique ajoute de modalités permettant de donner une infirmation sur le temps.

I.5 Ontologies et Traitement Automatique des Langages Naturels

L'exploitation des connaissances en informatique a pour objectif de permettre un dialogue (une coopération) entre les utilisateurs et les systèmes informatiques qui doivent avoir accès non seulement aux termes utilisés par l'être humain, mais aussi à la sémantique qui leur est associée.

Les ontologies offrent une connaissance partagée qui peut être échangée entre des personnes et des systèmes hétérogènes.

Les ontologies jouent un rôle important dans différents domaines tels que l'intelligence artificielle, le génie logiciel, et spécialement dans le Web Sémantique. Elles fournissent en effet les structures conceptuelles utilisées pour la description des ressources du Web, et permettent ainsi un traitement automatique de l'information

I.5.1 Ontologie

Une ontologie est une spécification explicite d'une conceptualisation qui permet de spécifier dans un langage formel les concepts d'un domaine et leurs relations. Une ontologie permet de définir ou de fournir une sémantique formelle (descriptive) pour l'information et nous permettant son exploitation par un ordinateur. Une ontologie est l'ensemble structuré des termes et concepts représentant le sens d'un champ d'informations. **[5] [6]**

Exemple d'une ontologie : Conception d'une carte électronique

- Une carte électronique est un ensemble de composants.
- Un composant peut être soit un condensateur, soit une résistance, soit une puce.
- Une puce peut être soit une unité de mémoire, soit une unité de calcul.

I.5.2 Types des Ontologies : les ontologies peuvent être classifiées en trois classes : **[6]**

Ontologies applicatives : contiennent des connaissances nécessaires pour une application donnée, spécifiques et non réutilisables.

Ontologies de domaine : réutilisable par plusieurs applications sur le même domaine.

Ontologies génériques ou top ontologies : Expriment des conceptualisations valables dans différents domaines. C'est comme s'il s'agit d'un regroupement des ontologies.

I.5.3 Langages pour Ontologies

L'objectif premier d'une ontologie est de modéliser un ensemble de connaissances dans un domaine donné. Cependant, la formalité et l'excitabilité d'une

ontologie dépendant du langage utilisé pour représenter et exploiter celle-ci. Les ontologies décrivent plusieurs choses telles que : **[6]**

- *Classes* : ensembles ou collections.
- *Attributs* : propriétés, fonctionnalités, caractéristiques ou paramètres.
- *Relations* : liens existants entre les objets.

La plupart des langages des ontologies se basent ou sont proches de la logique du premier ordre où les connaissances sont représentées sous formes d'assertions.

En intelligence artificielle, les langages d'ontologies sont des langages formels utilisés pour construire des ontologies. Ils permettent l'encodage des connaissances sur des domaines spécifiques et incluent souvent des règles de raisonnement prenant en charge le traitement de ces connaissances. Les quatre langages ontologiques les plus populaires sont KIF, OWL, RDF + RDFS et DAML + OIL

KIF: Knowledge Interchange Format, est un langage informatique permettant l'échange de connaissances entre divers programmes. La sémantique de KIF repose sur une conceptualisation du monde en termes d'objets et de relations entre ces objets. **[6]**

Il a été créé à l'origine dans le cadre de l'effort de partage des connaissances de la DARPA. Il y a eu un certain nombre de versions de KIF et plusieurs langues basées sur KIF ont été créées, telles que SUO-KIF.

Exemple dans SUO-KIF: Un véhicule ferroviaire est un véhicule conçu pour se déplacer sur des voies ferrées.

```
(subclass RailVehicle LandVehicle)
(documentation RailVehicle
"A Vehicle designed to move on &%Railways.")
(=>  (instance ?X RailVehicle)
     (hasPurpose ?X
       (exists (?EV ?SURF)
         (and  (instance ?RAIL Railway)
           (instance ?EV Transportation)
           (holdsDuring (WhenFn ?EV)
               (meetsSpatially ?X ?RAIL))))))
```

OWL: est un langage d'ontologie destiné au Web sémantique. C'est une extension linguistique du schéma RDF. OWL est compatible avec les premiers langages ontologiques comme DAML + OIL. Il inclut de conjonctions, de disjonctions, existentielles et universelles, qui peuvent être utilisées pour effectuer des inférences logiques et dérivés des connaissances. Cependant, certaines constructions de se langage sont très complexes et par conséquent trois sous langages sont conçues : OWL Lite, OWL DL et OWL Full .Un exemple d'OWL est présenté si dessous. **[5]**

Exemple : Classe OWL : ontologie Pizza

```
<owl:Class>
    <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#MozzarellaTopping"/>
        <owl:Class
    rdf:about="#PeperoniSausageTopping"/>
        <owl:Class rdf:about="#JalapenoPepperTopping"/>
        <owl:Class rdf:about="#TomatoTopping"/>
        <owl:Class rdf:about="#HotGreenPepperTopping"/>
    </owl:unionOf>
</owl:Class>
```

RDF: Resource Description Framework , est une norme de description des ressources sur le Web développée par le World Wide Web Consortium (W3C). Il est basé sur le langage XML (EXtensible Markup Language). RDF est en train de devenir un langage largement reconnu et un formalisme de représentation pouvant servir pour l'échange d'informations. **[5]**

RDF est basé sur l'idée d'identifier des éléments sur le Web via des références d'URI. Chaque instruction RDF consiste en un triplet: un sujet, un prédicat (également appelé propriété) et un objet out une valeur. Les relations entre les ressources sont décrites en termes de propriétés et de valeurs. Si dessous un exemple de RDF.

Exemple : Description d'un magasin de disques en RDF

```
<rdf:Description
rdf:about="http://www.recshop.fake/cd/Hide your heart">
    <cd:artist>Bonnie Tyler</cd:artist>
    <cd:country>UK</cd:country>
    <cd:company>CBS Records</cd:company>
    <cd:price>9.90</cd:price>
    <cd:year>1988</cd:year>
</rdf:Description>
```

RDFS : est un système extensible du RDF orienté objet. Il étendre RDF à la description d'ontologies. RDF permet de représenter des déclarations de propriétés sur des ressources, mais ne permet pas d'exprimer des connaissances sur les propriétés ou sur les types de ressources :

- Quelles sont les propriétés autorisées sur un type de ressources ?
- Quelles sont les valeurs autorisées pour une propriété ?
- Quels sont les liens entre les types de ressources (généralisation / spécialisation) ?

RDFS permet aux développeurs de définir un vocabulaire particulier pour les données RDF et de spécifier le type d'objet auquel ces attributs peuvent être appliqués.

OIL: Ontology Interchange Language , est un langage d'ontologie basé sur RDFS. Il a été créé conformément à ces exigences par un groupe de chercheurs principalement européens pour avoir une sémantique formelle bien définie avec des propriétés de raisonnement établies pour assurer l'exhaustivité, l'exactitude et l'efficacité et ainsi pour avoir un lien approprié avec les langages Web existants tels que XML et RDF pour assurer l'interopérabilité. **[6]**

Exemple : herbivore dans OIL

```
<rdfs:Class rdf:ID="herbivore">
    <rdf:type
        rdf:resource="http://www.ontoknowledge.org/oil/ RDFSschema/#DefinedClass"/>
    <rdfs:subClassOf rdf:resource="#animal"/>
    <rdfs:subClassOf>
      <oil:NOT>
        <oil:hasOperand rdf:resource="#carnivore"/>
      </oil:NOT>
    </rdfs:subClassOf>
</rdfs:Class>
```

DAML: DARPA Agent Markup Language, est une initiative sponsorisée par le gouvernement américain visant à développer un langage et des outils facilitant le concept de Web sémantique. DAML comprend deux parties, le langage d'ontologie et un langage permettant d'exprimer des contraintes et d'ajouter des règles d'inférence. Il inclut également des correspondances vers d'autres langages Web sémantiques tels qu'OIL, KIF, XML et RDF.

Le langage DAML est une extension de XML et de RDF. La dernière version du langage (DAML + OIL) fournit un ensemble riche de constructions permettant de créer des ontologies lisibles et compréhensibles par des machines. Si dessous un exemple à l'aide de la clause Restriction du DAML. **[6]**

```
<daml:Class rdf:about="Process">
    <rdfs:comment>
      A Process can have at most one name, but names need not be unique.
    </rdfs:comment>
   <rdfs:subClassOf>
        <daml:Restriction daml:maxCardinality="1">
          <daml:onProperty rdf:resource="#name"/>
        </daml:Restriction>
   </rdfs:subClassOf>
</daml:Class>
```

I.6 Traitement Automatique des Langages Naturels

On regroupe sous le vocabulaire TALN, Traitement Automatique des Langages Naturels l'ensemble des recherches et développements visant à modéliser et reproduire à l'aide des machines la capacité humaine à produire et à comprendre des énoncés linguistiques dans des buts de communication. C'est une discipline à la frontière de la linguistique, de l'informatique et de l'intelligence artificielle qui concerne l'application des techniques et programmes informatiques à tous les aspects du langage humain. [7]

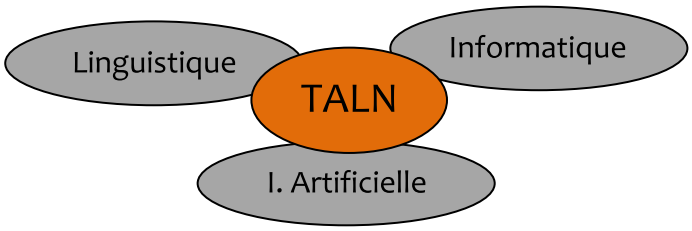


Figure I.1 : Disciplines du TALN

I.6.1 Les Applications du TALN

Traitement documentaire

- La traduction automatique
- La recherche de documents
- La lecture automatisée de documents
- les réponses automatiques aux questions
- L'analyse d'un corpus

La production de documents

- Les claviers auto correcteurs
- La reconnaissance optique de caractères
- Les correcteurs d'orthographes ou de syntaxe

Traitements du signal

- La reconnaissance automatique de la parole
- La synthèse de la parole
- Le traitement de la parole

I.6.2 Les Difficultés du TALN

Les difficultés en TALN sont principalement dûes à deux choses, l'ambiguïté du langage et la quantité d'implicite contenue dans les communications naturelles.

Ambiguïté : Parmi les problèmes les plus importants du traitement des langages naturels, on trouve en premier lieu le problème de l'ambigüité. On dit qu'il ya ambigüité lorsqu'il ya plus d'une interprétation pour une structure linguistique donnée. [7]

Exemple : Prononciation de la lettre i : Lit, Voir, Maison

Implicite : les énoncés naturels peuvent être implicites et par conséquent la compréhension complète de la majorité des énoncés est difficile, voire impossible pour une machine qui ne dispose pas de base de connaissances additionnelle donnant accès à la fois à un savoir sur le monde (ou le domaine) en général (connaissance statique) et sur le contexte de l'énonciation (connaissance dynamique).

I.6.3 Les Niveaux de Traitement

Plusieurs niveaux de traitement sont nécessaires pour parvenir à une compréhension complète d'un énoncé en langage naturel. Ces niveaux correspondent à des modules qu'il faudrait développer et faire coopérer dans le cadre d'une application complète de traitement de la langue. [7]

Le niveau lexical

Le but de cette étape de traitement est de passer des formes atomiques (tokens) identifiées par le segmenteur aux mots, c'est-à-dire de reconnaître dans chaque chaîne de caractères une (ou plusieurs) unité(s) linguistique(s), dotée(s) de caractéristiques propres (son sens, sa prononciation, ses propriétés syntaxiques, etc).

Le niveau syntaxique

La syntaxe est l'étude des contraintes portant sur les successions licites de formes qui doivent être prises en compte lorsqu'on cherche à décrire les séquences constituant des phrases grammaticalement correctes. Le niveau syntaxique est le niveau qui définit la façon avec laquelle les mots sont arrangés dans une phrase.

Le niveau sémantique

La sémantique se préoccupe du sens des énoncés. Une phrase comme le jardin de la porte mange le ciel bien que grammaticalement parfaitement correcte, n'a pas de sens dans la plupart des contextes. Donc La sémantique correspond à la signification des mots ou un groupe de mots.

Le niveau pragmatique

Le niveau pragmatique et totalement dissociable au niveau sémantique et contrairement à la sémantique qui se préoccupe du sens des énoncés, la pragmatique porte sur les attitudes (vérité, désirabilité, probabilité…) que les locuteurs adoptent vis à vis des énoncés et sur les opérations logiques que ces attitudes déclenchent.

Le niveau phonologique

La phonologie se préoccupe des sonorités de la langue orale, ce qui est important pour la reconnaissance vocale.

I.7 Les Actes des Langages

La pragmatique linguistique est largement développée sur la base de la théorie des langages. Un acte du langage est un moyen mis en œuvre par un locuteur pour agir sur son environnement par ses mots, il cherche à informer, demander, convaincre, promettre… etc ses interlocuteurs.

L'initiateur de cette théorie est le philosophe britannique Austin dans son ouvrage : How to do things with words (1962), ensuite elle était développée par J.R Searle dans ces deux ouvrages Les Actes de Langage (1972), et Sens et expression, 1982.

I.7.1 Classification des Actes

Selon Austin, l'énonciation est le fruit de trois activités complémentaires : actes locutoires, actes illocutoires et actes perlocutoires. **[8]**

Les actes locutoires (Que dit-il ?) : se rapportent à la formulation d'un énoncé. Ces actes sont satisfaits lorsque l'énoncé est correctement formulé.

Les actes illocutoires (Que fait-il ?) : désignent l'action effectuée sur l'auditeur lors de la formulation de l'énoncé (donner un ordre, poser une question …) .Ils sont accomplis avec succès lorsque l'effet attendu sur l'auditeur est obtenu (ordre exécuté, réponse à une question … etc)

Les actes perlocutoires (Pourquoi faire ?) : que l'on accomplit par le fait d'avoir dit quelque chose et qui relèvent des conséquences de ce que l'on a dit comme une question qui peut servir à interrompre ,montrer qu'on est là… etc

I.7.2 Types des Actes

Searle a dégagé cinq classes majeures d'actes de langages : **[9]**

- **Les assertif**s : assertions, affirmations
- **Les directif**s : ordre, demande
- **Les promissifs** : promesse, invitation
- **Les expressifs** : félicitations, remerciements
- **Les déclaratifs** : déclaration de guerres

Chapitre II

Intelligence Artificielle Distribuée

Chapitre II : Intelligence Artificielle Distribuée

II.1 Définition

L'évolution des domaines d'application de l'IA pour couvrir des domaines plus complexes tels que l'aide à la décision ,la reconnaissance et la compréhension des formes a montré les limites de l'IA classique qui s'appuie sur une centralisation au sein d'un système unique.

Le but de l'intelligence artificielle est de simuler le comportement intelligent d'un être humain ou un seul individu, mais souvent l'intelligence chez les humains n'est pas centralisée et cela est peut être justifié qu'on ne peut pas trouver une seule personne qu'elle peut résoudre toutes les problématiques du monde de différentes natures. Par conséquent l'intelligence chez les humais est distribuée d'où vient la nécessité de distribuer l'intelligence artificielle sur plusieurs entités ou agents afin d'améliorer les performances d'un tel system dit intelligent pour pouvoir résoudre les problèmes complexes.

L'IAD est peut être vue comme un passage du comportement individuel aux comportements collectifs pour combler les limites de I.A classique à résoudre des problèmes complexes. La distribution de l'intelligence peut être vue comme une distribution de plusieurs choses y compris la connaissance, le contrôle ou même les ressources.

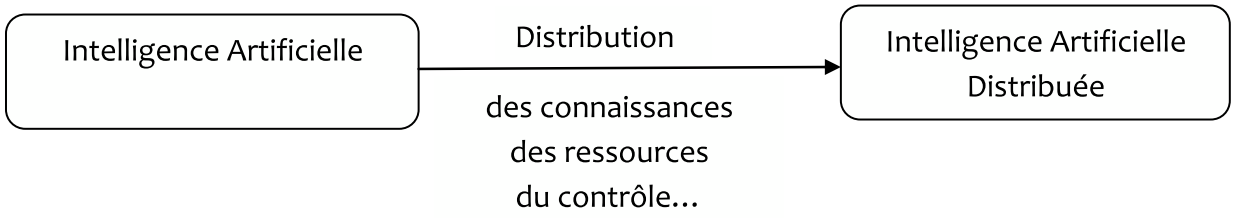


Figure II.1 : Intelligence Artificielle Distribuée

L'IAD, ou intelligence artificielle distribué simule le comportement intelligent de plusieurs agents. Elle s'intéresse à des comportements intelligents qui sont le produit de l'activité coopérative de plusieurs agents. Plusieurs problèmes proprement liés à l'IAD peuvent être apparu :

- La modélisation des connaissances et le problème de sa répartition entre les différents agents.
- La génération des plans d'actions et la gestion des conflits.
- Le problème de communication et interaction entre agents.

L'IAD couvre trois axes fondamentaux : la résolution distribuée des problèmes, l'intelligence artificielle parallèle et les Systèmes multi agents

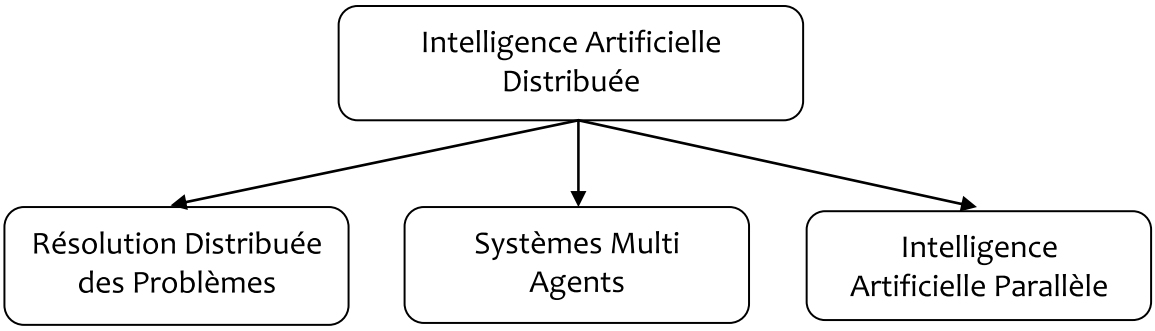


Figure II.2 : *Axes de l'IAD*

II.2 Résolution Distribuée des Problèmes : RDP

Comment diviser un problème particulier sur un ensemble d'entités distribuées et coopérants. Elle s'intéresse aussi à la manière de partager la connaissance du problème et d'en obtenir la solution.

II.3 Intelligence Artificielle Parallèle : IAP

Elle s'intéresse au développement des langages et des algorithmes parallèles pour l'IAD dans le but d'améliorer les performances des systèmes intelligents.

II.4 Système Multi Agents : SMA

Un SMA, ou Système multi Agents est de faire coopérer un ensemble d'entités, plus précisément agents dotés d'un comportement intelligent, coordonner leurs buts et leurs plans d'actions pour résoudre un ou plusieurs problèmes.

II.4.1 Apports des SMA

L'approche multi agent présente plusieurs avantages y compris :

- L'adaptation à la réalité : simple à comprendre
- La coopération : possibilité de faire coopérer plusieurs agents
- La résolution de problèmes complexes
- La modularité : qui simplifie le développement
- La réutilisation : puisque le système est composé d'un ensemble de modules
- L'efficacité : elle a été appliquée avec succès dans plusieurs domaines
- La fiabilité : donne de bons résultats.

II.4.2 Concept d'Agent

Un agent peut être une chose ou une personne qui agit de manière autonome et produit des changements dans le monde auquel il se situe.

Un agent est toute entité active dont le comportement est utilement décrit à l'aide de notions mentales telles que les connaissances, les buts, les intentions et peut être même les croyances.

Un agent intelligent capable de prendre en temps limité des décisions rationnelles dans un environnement.

Un agent est une entité physique (être humain, animal, robot…) ou abstraite (logiciel, programme…) situé dans un environnement, qui agit d'une façon autonome et flexible pour attendre les objectifs pour lesquels il a été conçu.

II.4.3 Propriétés d'un Agent

Situé : un agent peut recevoir des entrées sensorielles provenant de son environnement comme il peut aussi effectuer des actions qui sont susceptibles de changer cet environnement. Le monde réel et l'internet sont des exemples d'environnements ou les agents peuvent êtres situés.

Autonome : un agent doit être capable d'agir dans l'intervention directe d'un être humain ou d'un autre agent. Il doit avoir complètement le contrôle de ses actions et de son état interne.

Flexible : veut dire que l'agent doit être capable de répondre à temps, c'est-à-dire qu'il soit capable de percevoir son environnement et répondre rapidement aux changements qui se produisent sur ceci.

Proactif : un agent doit être capable d'avoir un comportement opportuniste, il n'agit pas simplement en réponse à son environnement mais il tente de satisfaire ces buts ou sa fonction d'utilité.

Social : un agent doit être capable d'interagir avec d'autres agents que se soient artificiels ou humains afin de résoudre et complète ces tâches ou bien pour aider les autres agents à compléter les leurs.

Donc d'après Ferber [10], un agent :

- Percevoir son environnement
- Possède ses propres ressources
- Est mû par un ensemble de capacité
- Peut agir sur son environnement
- Tendre à satisfaire ses objectifs
- Communique avec d'autres agents
- Offre des services
- éventuellement se reproduire

Un agent peut aussi avoir la possibilité de négocier avec d'autres agents, d'apprendre de nouveaux connaissances à partir de l'expérience, comme il peut se déplacer dans des niveaux différents qui peuvent êtres physiques (mémoire, disque dur…) ou virtuels (niveaux d'exécutions dans un système d'exploitation…)

II.4.4 Types des Agents

Un système multi agents est un système composé d'un ensemble d'agents interagissant entre eux selon certaines relations pour accomplir un ensemble de tâches. Les agents peuvent être classifiés selon les fonctionnalités attribuées et le degré de complexité de chaque agent.

II.4.4.1 Agent Réactif

Comme son nom l'indique, un agent réactif ne fait que réagir aux changements qui surviennent dans l'environnement. Autrement dit, un tel agent ne fait que d'acquérir des perceptions et de réagir à celles-ci en appliquant certaines règles prédéfinies. Etant données qu'il n' ya pas pratiquement de raisonnement, ces agents peuvent agir et réagir très rapidement. **[11]**

Les humains aussi utilisent cette manière pour agir dans plusieurs situations là ou il est préférable de ne pas (trop) penser (Reflexe) et de réagir immédiatement. Par exemple, lorsqu'une personne met la main sur une plaque très chaude, elle ne commence pas à se demander si c'est chaud, si ça fait mal et s'il faut ou non qu'elle retire sa main. Dans ce cas, elle retire sa main immédiatement sans réfléchir.

Exemple : Agent à reflexe simple

Ce type d'agents agit uniquement en se basant sur ses perceptions courantes. Il utilise un ensemble de règles prédéfinis du type : **Si** condition **alors** actions. Par exemple pour un agent en charge du contrôle de la défense, on pourrait avoir la règle suivante : Si missile en direction alors lancer missile d'interception.

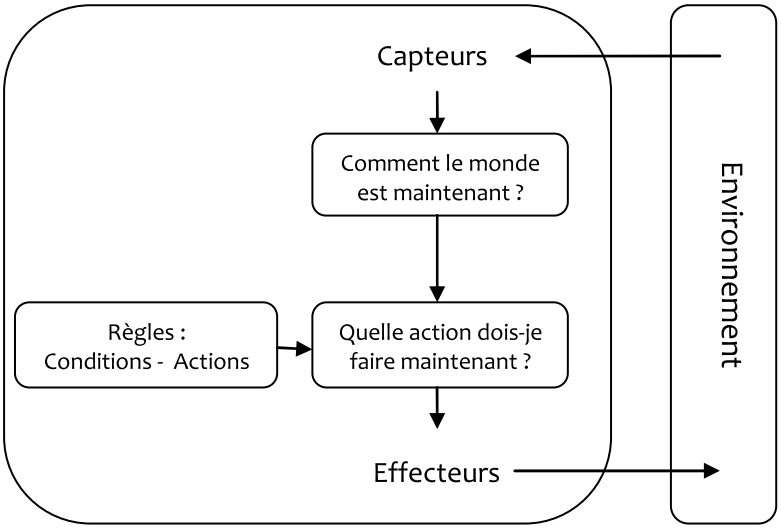


Figure *II.3* : Agent à reflexe simple

Le comportement de ce type d'agents est très rapide mais peut réfélchi puisque les régles son presque en liaison directe avec les perceptions de l'agent et ses actions. Le comportement des agents réactifs peut etre défini par une fonction :

Agir S-------A telles que

S={S1, S2, S3,…} : L'ensemble des états de l'environnement

A={A1 ,A2,A3,…} :L'ensemble des actions qu'il doit réaliser

II.4.4.2 Agent Cognitif

Les agents cognitifs sont des agents qui effectuent un certain raisonnement pour choisir leurs actions. Un tel choix peut se faire en se basant sur les buts de l'agent ou sur une certaine fonction d'utilité. **[10]**

Exemple : Agent ayant des buts

Contrairement aux agents réactifs qui utilisent leurs connaissances sur l'état actuel de l'environerment pour choisir leurs actions, les agents cognitifs qui raisonnent sur les buts tiennent compte d'une certaine projection dans le futur . Ils se posent des questions comme « qu'est ce que va arriver si je fais telle ou telle action ? » et « est ce que je serais satisfait si cela se produit ? »

Bien entendu , l'agent raisonnant sur ses buts prends en général beaucoup plus de temps agir qu'un agent réactif . le comporetemnt d'un agent cognitif peut être défini par la fonction suivante :

Agir S*-----A telles que :

S={S1, S2, S3,…} : L'ensemble des états de l'environnement

A={A1 ,A2,A3,…} : L'ensemble des actions qu'il peut réaliser

L'agent peut choisir l'action en fonction de son historique, il integre son passé à son comportement comme le montre l'exemple suivant :

Historique 1 : S0----A0----S1----A1----S2----A2----S3

Agir (S0,S1,S2)=A2

Le comportement d'un agnet dans un environnement est l'ensemble de tous le historiques stisfaisants ses conditions.

Exemple : Agent utilisant une fonction d'utilité

Dans plusieurs situations, les buts ne sont pas suffisants pour générer un comportement de haute qualité . par exemple s'il ya plusieurs chemins possibles pour

atteindre le but, certains seront plus rapides et d'autres plus dangereux. Dans cette situation, l'agent raisonnant uniquement sur ses buts n'a pas de moyens pour choisir le meilleur chemin . Donc il faut mieux de définir une fonction d'utilité qu'attribue une valeur numérique à chacun des états afin que l'agent peut prendre des meilleurs décisions.

Comparaison

la table suivante présente une comparaison entre un système à base d'agents cognitifs et un systéme à base d'agent réactifs.

Table II.1 : Comparaison entre un système à base d'agents Cognitifs et un systéme à base d'agent Réactifs **[11]**

Systéme à base d'agents Cognitifs	Systéme à base d'agents Réactifs
Représentation explicite de l'environnement	Pas de représentation explicite de l'environnement
Peut tenir compte de son passé	Pas de mémoire de son historique
Agents complexes	Fonctionnements : Stimulus - Actions
Petit nombre d'agents	Grand nombre d'agents

Exemple d'un SMA : Fouragement de Fourmis

- Un agent : Une fourmis
- Suit les phéromones
- Dépose des phéromonessur leurs retours si elles sont chargées de nourriture
- Les phéromones s'évaporent avec le temps

Quel type d'agents s'agit' il ? il s'agit d'un agent réactif

Remarque : la tendance réactive prétend qu'il n'est pas nécessaire que chaque agent soit intelligent pour que le système global soit intelligent. Donc un agent réactif n'a pas d'intelligence individuelle mais plutôt collective.

II.4.4.3 Agents Hybrides

Chacune des architectures cognitives ou réactives est appropriée pour un certain type de problémes . pour la majorité des problémes, ni une architecture complétement réactive, ni une architecture complétement cognitive est appropiée . Comme pour les humains, les agents doivent pouvoir réagir très rapidement dans certaines situations(Comportement reflexe) tandis que dans d'autres, ils doivent avoir un comportemnt plus réfléchi.

Dans ce cas, on parle d'une architecture hybride dans laquelle on trouve généralement plusieurs couches arrangées verticalement ou horizontalement . **[11]**

Représentation Horizontale : comportement incohérent , compétition entre les niveaux pour cela il faut choisir un niveau médiateurqui permet le contrôle centralisé.

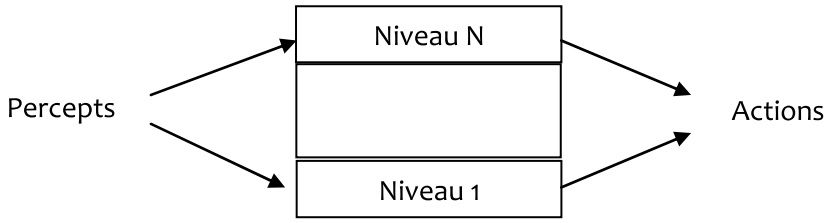


Figure *II.4* : Représentation Horizontale

Représentation Verticale :

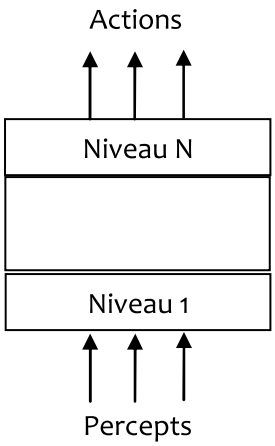


Figure *II.5* : Représentation Verticale

II.4.5 Architecture BDI

Plusieurs modèles ont été proposés pour décrire la façon dans un être humain raisonne. le modèle le plus adopté est le modèle de croyance-désir-intention (BDI), proposé à l'origine par Bratman en 1987 en tant que théorie philosophique du raisonnement pratique. **[12]**

L'architecture BDI (Belief Desires Intention) est construite à partir du théorème concernant le raisonnement pratique ou les agents sont en général représenté par ces états mentales qui sont les croyances , les désirs et les intentions .**[13]**

Croyances (Beliefs) : se sont des données que l'agent posséde en provenance de l'environnement .

Exemple : nuage ---- possible de pluie

La croyance représente ce que l'agent comprend de son environnement . les croyances peuvent être incorrectes, incomplétes ou incertaines. Les croyances peuvent

changer au fur et à mesure que l''agent par sa capacité de perception ou par l'interaction avec d'autres agents .

Désirs : les différents états que l'agent souhaite réaliser, c'est-à-dire les buts que l'agent sohaite atteindre.

Intention : sous ensemble des désirs, représentent les actions quel'agent à décidé de faire pour accomplir ses désirs. **[10]**

Exemple

L'exemple suivant va apporter plus de clarification à ce modèle.

L'agent Mohamed a la croyance que si quelqu'un passe son temps à étudier, cette personne peut faire une thése de doctorat . En plus ,Mohamed a le désir de voyager beaucoup, de faire une thése doctorat et d'obtenir un poste d'assistant à l'université.

Le désir de voyager beaucoup n'est pas consistant avec les deux autres et Mohamed après réflexion, décide de choisir parmi ces désirs inconsistants les deux derniers.

Comme il se rend compte qu'il ne peut pas réaliser ses deux désirs à la fois, il décide de faire d'abord une thése de doctorat . En ce moment Mohamed à l'intention de faire une thése .

II.4.5.1 Composants d'une architecture BDI

Le shéma suivant rassemble les différents composant d'une architecture BDI

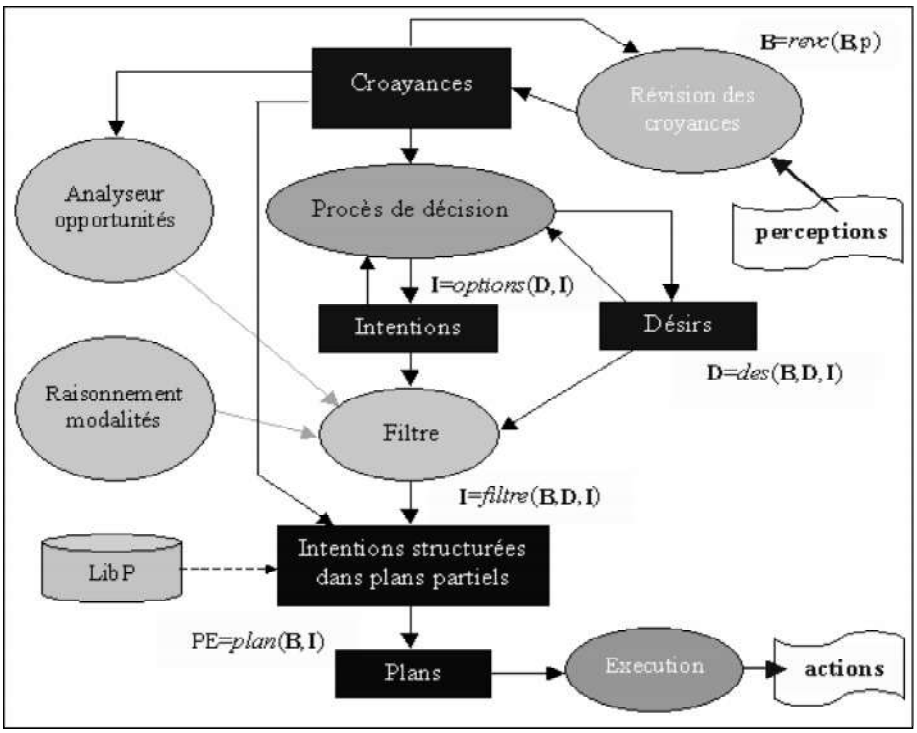


Figure *II.6* : Architecture BDI **[13]**

Révision des croyances : responsable de la mise à jour des croyances de l'environnement et les nouvelles perceptions de l'environnement.

Proces de décision : il permet de générer les intentions de l'agent en évaluant ces désirs.

Analyseur opportunités : En conséquence de ce qu'il percoit de son environnement et de sa révision des croyances , l'agent peut detecter des nouvelles opportunités qui favorisent la réalisation de ses intentions ou qui peuvent même empêcher cette réalisation . Cette analyse est effectuée par la composante Analyseur opportunités.

Filtre : le filtre sert essentiéllement à faire un choix parmies différentes possibilités d'intentions en se basant sur les désirs et sur les croyances.

Raisonnement modalités : Ce type de raisonnement permet de décider comment on peut arriver a un etat du monde désiré , ce module avec le filtre permet de planifier les intentions .

Librairie de plan (Lib P) : un plan est une séquence d'actions à exécuter dans le temps. Une bibliothéque de plans contiennent la liste de tous les plans que l'agent peut utiliser pour répondre à une certaine situation en parcourant cette bibliothéque.

Exécution : responsable de réaliser les actions choisies sur l'environnement.

Les Fonctions : [13]

***Revc* :** B x P ->B cette fonction est réalisée par la composante Révision des croyances, C'est une fonction qui prend en charge la révision des croyances de l'agent lorsqu'il reçoit de nouvelles perceptions sur l'environnement, où P représente l'ensemble des perceptions de l'agent.

***Options* :** D x I->I cette fonction réalisée par la composante Processus de décision représente le processus de décision de l'agent prenant en compte ses désirs et ses intentions courantes.

***Des* :** B x D x I->D est la fonction qui peut changer les désirs d'un agent si ses croyances ou intentions changent. Cette fonction est également réalisée par la composante Processus de décision.

***Filtre* :** B x D x I->I est la fonction la plus importante réalisée par la composante Filtre car elle décide des intentions à poursuivre.

II.4.5.2 Avantages de l'architecture BDI

- Elle s'appuie sur une théorie connue et appréciée de l'action rationnelle des humains.

- La théorie a été formalisée dans une logique symbolique formelle, rigoureuse.
- Elle a été implémentée et utilisée avec succès dans beaucoup d'applications.

II.4.6 Les Agents Mobiles

Un agent mobile est un agent capable de se déplacer dans son environnement, qui peut être physique (réel ou simulé) ou structurel (niveaux d'exécution par exemple) grâce à des dispositifs assurant sa mobilité.

La mobilité permet à un agent de se déplacer d'un site à un autre pour accéder à des données ou à des ressources ou de se rapprocher d'un objectif à accomplir à titre d'exemple. L'agent peut se déplacer avec son code, ses données propres et peut être même avec son état d'exécution.

II.4.6.1 Quelques Exemples d'applications

- Commerce électronique : suivi de salle de marché
- Filtrage, recherche d'information
- Collecte d'informations distribuée
- Système de workflow et système collaboratif
- Monitoring et notification
- Détection d'intrusion
- Calcul parallèle
- Jeux sur internet

II.4.6.2 Mobilité d'un Agent

Au niveau de la mise en œuvre, les migrations d'agents mobiles peuvent s'effectuer selon deux modes :

Migration Forte

La totalité de l'agent (code, données et unité d'exécution) migre vers le nouveau site. Dans cette migration, l'agent suspendu son exécution avant d'être transféré. Une fois arrivé sur le nouveau site, il redémarre son exécution au point de contrôle précédent.

Migration Faible

Une fois arrivé sur un site distant, l'agent est réexécuté.

II.4.6.3 Station d'accueil

Les stations d'accueils des agents mobiles doivent fournir quelques supports :

Support d'exécution : pour que l'agent puisse accomplir ses tâches

Support de communication : l'agent mobile peut avoir besoin de communiquer avec la station mère ou avec d'autres agents.

Sécurité : la station d'accueil doit assurer la sécurité de l'agent mobile

II.4.7 Coopération entre Agents

Une interaction est une mise en relation dynamique de deux ou plusieurs agents. La coopération est la forme générale de l'interaction. La mise en œuvre d'un travail collectif propose quelques buts comme l'augmentation de la vitesse de résolution des tâches et l'augmentation de nombre de tâches réalisables. [11]

II.4.7.1 Modèles de Coopérations

La coopération entre les agents consiste en général de décomposer les tâches en sous-tâches puis à les répartir entre les différents agents, il existe plusieurs styles de décompositions possibles chacun décrit un modèle de coopération. [11]

Partage de tâches et de résultats : nécessite le pouvoir de décomposer un problème à des sous problèmes peuvent être traité de façon indépendante et avec un minimum communication entre agents. Ce modèle semble particulièrement adapté pour les domaines ou il existe une hiérarchie des tâches.

Commande : un agent supérieur décompose le problème en sous-problèmes qu'il répartit entre les autres agents, ceux-ci le résolvent et renvoient les solutions partielles à l'agent supérieur.

Appel d'offre : un agent supérieur décompose le problème en sous-problèmes dont il diffuse la liste. Chaque agent qui le souhaite envoie une offre ; l'agent supérieur choisit parmi celles-ci et distribue les sous-problèmes. Le système travaille ensuite en mode commande.

Compétition : dans ce mode coopération, l'agent supérieur décompose et diffuse la liste des sous problèmes comme dans le mode appel d'offre, chaque agent résout un ou plusieurs sous-problèmes et envoie les résultats correspondants à l'agent supérieur qui à son tour fait le tri.

II.4.8 Communication entre Agents

La communication est la mise en relation d'un émetteur avec un ou plusieurs récepteurs dans l'intention d'atteindre certains objectifs.

Les agents communiquent et interagissent pour synchroniser leurs actions et résoudre des conflits. Parmi les modèles de coopération on trouve : [11]

II.4.8.1 Communication par partage d'informations : un espace mémoire est partagé vue comme un tableau sur lequel les agents écrivent et trouvent des informations et des réponses partielles.

II.4.8.2 Communication par envoi de messages : un langage d'acteur est la technique la plus utilisée pour la mise en œuvre de ce modèle ou les agents doivent utiliser un protocole de communication qui leur permet de structurer et d'assurer la continuité entre un début et une fin. L'échange de messages nécessite un langage avec :

- une syntaxe : langage commun pour représenter les informations et les requêtes
- une sémantique : un vocabulaire structuré et un cadre partagé de connaissance : une ontologie partagée

II.4.8.3 Actes du Langage

Présentent un grand intérêt pour la communication dans les SMA. Ils désignent l'ensemble des actions intentionnelles effectué au cours d'une communication.

Toute communication est faite avec l'objectif de satisfaire un but, une intention.

Communication= actions de différentes types.

Types des actes du langage

Les assertifs : *affirmation* : il fait beau aujourd'hui.

Les directifs : *les commandes* : demander, regarde le livre.

Les promissifs : *engagent les locuteurs à accomplir certains actes dans le futur* : je voudrais à l'université demain.

Les expressifs : *ils donnent aux destinataires des informations sur l'état mental du locuteur* : je suis joyeuse.

Les déclaratifs : *ils accomplissent un acte par le fait de prononcer l'énoncé* : je déclare la séance ouverte.

II.4.8.4 Langages d'agents

De nombreux langages de communications entre agents se sont développés. Dont les plus connus sont :

- KQML
- FIPA-ACL

II.4.9 KQML

En 1997 et aux Etats-Unis, un standard de fait de langage de communication de haut niveau appelé KQML (Knowledge Query and Manipulation Language) a été développé fondé sur la théorie des actes de langage. Il permet l'échange des informations, indépendant de la syntaxe du contenu et de l'ontologie utilisé et indépendant du mécanisme de transport (TCP/IP, SMTP…). **[12]**

Un message KQML comporte trois 3 couches : **[14]**

La couche Contenu : Il s'agit du contenu réel du message qui peut être écrit dans un langage de représentation. KQML peut transporter des messages écrits dans n'importe quel langage de représentation (PROLOG, KIF, LISP, C, KQML, XML ...)

La couche Communication : cette couche contient des informations du niveau un peu plus bas permettant la description de quelques paramètres de base de la communication telles que l'identité de l'émetteur ou celle du récepteur du message, ainsi qu'un identificateur unique pour le message.

La couche Message : Cette couche constitue le cœur du langage. Elle incorpore La performative (action : affirmation, demande…) que l'expéditeur attache au contenu. Puisque le contenu est opaque à KQML, des arguments décrivant le contenu du message tel que le langage utilisé et l'ontologie sont aussi inclut.

Ontologie est une spécification ou une vue simplifié et abstraite du domaine qui sera représenté. En d'autres termes, une ontologie définit le vocabulaire dans un domaine donné pour que les agents puissent se comprendre.

Enonciation performative : c'est faire quelque chose à l'aide du langage, c'est la réalisation d'un acte. Elle est peut être réussite ou non.

II.4.9.1 Performatives du KQML

KQML comporte 36 performatives répartis en 3 catégories :

- **Les 18 performatives de discours** : ask-if, ask-one, tell, describe, stream-all …
- **Les 11 performatives d'interconnexion** : register, unregister, broadcast …
- **Les 7 performatives d'exception** : error, sorry, standby …

Quelques performatives

La table suivante présentes quelques performatives du langage KQML ou :

- E : l'agent émetteur
- R : l'agent récepteur
- C : le contenu du message
- BVC : la base virtuelle de connaissances

Table II.2 : Quelques performatives KQML **[14]**

PERFORMATIVE	DESCRIPTION
ask-one	E veut que seulement R répond à sa question C
ask-if	E veut savoir si la réponse à la question précisée en C se trouve dans la BVC de R
ask-abou	E veut connaître toutes les propositions pertinentes dans les BVC de R
Tell	E affirme au R que C est dans la BVC de E
broadcast	E veut que R transmette à son tour la performative à toutes ses connexions
Error	E considère le message précédent de R comme mal formé
Sorry	R ne peut pas fournir plus d'information
stream-about	une version réponse multiple de ask-about
Next	E veut la réponse suivante de R à un flux d'une performative
Eos	termine un flux de réponses (end of stream)

II.4.9.2 Exemples de messages KQML

Exemple 1 : L'agent A1 demande à l'agent A2 le prix d'un portable OPPO F9 et A2 lui répond.

```
(ask-one
:sender A1
:receiver A2
:content (val (prix OPPO F9 ²))
:language KIF
:ontology Mobiles )
```

```
(tell
:sender A2
:receiver A1
:content (=(prix OPPO F9) (scalar 58 000 DA))
:language KIF
 :ontology Mobiles )
```

Exemple 2 : L'agent A1 demande à l'agent A2 toutes les informations concernant l'imprimante HPJet, et A2 lui répond par plusieurs messages qui se terminent avec un 'eos':

```
 (stream-about
:sender A1
:receiver A2
:reply-with hpj
:language KIF
: ontology imprimantes
:content HP-Jet )
```

```
(tell
 :sender A2
:receiver A1
 :in-reply-to hpj
:content (=(resolution HP-Jet) (scalar 300 dpi))
:language KIF
 : ontology imprimantes )

 (tell
:sender A2
:receiver A1
 :in-reply-to hpj
 :content (=(prix HP-Jet) (scalar 190 USD))
 :language KIF
 : ontology imprimantes )

 (eos
 :sender A2
 :receiver A1
 :in-reply-to hpj )
```

II.4.9.3 Avantages et Inconvénients du KQML

+ Premier standard de communication
+ Beaucoup d'applications et plateformes supportent KQML

- Ambigüité et imprécision
- Classes de performatives absents car il n'ya pas d'expressifs

Chapitre III

Plateformes des Systèmes Multi Agents

Chapitre III : Plateformes des Systèmes Multi Agents

III.1 Introduction

Une plate-forme de développement des systèmes multi-agents est une infrastructure de logiciels utilisée comme environnement pour le déploiement et l'exécution d'un ensemble d'agents. Une plate-forme à agents permet :

- La création des agents
- L'exécution des agents
- La migration des agents
- La terminaison des agents

Plusieurs plateformes pour développer des agents existent parmi eux : **[12]**

- Janus
- Jade
- Jadex
- Aglet
- Madkit
- AgentBuilder

III.2 JESS

Jess est une bibliothèque écrite en Java pour la manipulation des systèmes experts et par conséquent, pour l'utiliser, vous aurez besoin d'une machine virtuelle Java (JVM). Jess 7.1 est compatible avec toutes les versions publiées de Java à partir de JDK 1.4, y compris le JDK 1.6 et la dernière version.

Les anciennes versions de Jess numérotées 4.x étaient compatibles avec les versions JDK 1.0, 5.x fonctionnaient avec JDK 1.1, et Jess 6 travaillait avec le JDK 1.2 et plus.

Pour utiliser l'environnement de développement intégré JessDE, vous devez disposer de la version 3.1 ou ultérieure d'Eclipse. La bibliothèque Jess sert d'interprète pour une autre langue. Le langage Jess est une forme hautement spécialisée de Lisp.

Après avoir télécharger Jess . Il est fourni en tant que fichier .zip unique qui peut être utilisé sur toutes les plates-formes supportées Windows, UNIX ou Macintosh. Ce fichier contient tout ce dont vous avez besoin pour utiliser Jess . Lorsque le fichier zip est décompressé, vous devriez avoir un répertoire nommé Jess71p1 /. Dans ce répertoire, il devrait y avoir les fichiers et sous-répertoires suivants: **[15]**

Table III.1 : Composition du fichier ZIP de Jess

README	Un guide de démarrage rapide.
LICENSE	Informations sur vos droits concernant l'utilisation de Jess.
bin	Répertoire contenant un fichier de commandes Windows (jess.bat) et un script de shell UNIX (jess) que vous pouvez utiliser pour démarrer l'invite de commande Jess
lib	Un répertoire contenant Jess lui-même, en tant que fichier d'archive Java. Notez qu'il ne s'agit pas d'un fichier d'archive "cliquable"; vous ne pouvez pas double-cliquer dessus pour lancer Jess. C'est délibéré. Ce répertoire contient également l'API JSR-94 (javax.rules) dans le fichier jsr94.jar.
docs/	Cette documentation. "index.html" est le point d'entrée pour le manuel Jess.
examples/jess	Un répertoire de petits exemples de programmes en langue Jess.
examples/xml	Un répertoire de petits exemples de programmes dans JessML, le langage de règles XML de Jess.
eclipse	JessDE, l'environnement de développement intégré de Jess, fourni sous la forme d'un ensemble de plugins pour Eclipse 3.0.
Src (facultatif)	Si ce répertoire est présent, il contient la source complète du moteur de règles Jess et de l'environnement de développement, y compris un script Ant pour sa création.

III.2.1 Interface de Ligne de Commande

Jess a une interface de ligne de commande interactive. La distribution les scripts que vous pouvez exécuter pour obtenir un invité de commande. Ils sont tous deux dans le répertoire bin /. Le jess.bat est pour lancer l'interface de commande Jess sous Windows.

Pour exécuter un fichier de code Jess à partir de l'invité Jess, utilisez la commande batch: Jess> (batch "examples/jess/sticks.clp")

Pour exécuter le même programme Jess directement à partir de l'invité du système d'exploitation, vous pouvez passer le nom du programme en tant qu'argument au script qui démarre Jess: C:\Jess71p1> bin\jess.bat examples\jess\sticks.clp **[15]**

Remarque

Lorsque vous travaillez sur la ligne de commande Jess, il est pratique de pouvoir modifier et réinviter les commandes précédentes. La bibliothèque Jess ne supporte pas cela directement, mais l'excellent produit open source JLine peut ajouter cette capacité à n'importe quel programme Java en ligne de commande, y compris Jess.

III.2.2 Console graphique

La classe jess.Console est une version graphique simple de l'interface de ligne de commande Jess. Vous tapez dans un champ de texte au bas de la fenêtre, et la sortie apparaît dans une fenêtre de défilement.

Exemple : C:\Jess71p1> java -classpath lib\jess.jar jess.Console

III.2.3 Programmation Java avec Jess

Pour utiliser Jess en tant que bibliothèque à partir de vos programmes Java, le fichier jess.jar qui se trouve dans le répertoire lib doit être installé et définit sur votre chemin de classe.

La définition du chemin de classe implique généralement la modification d'une variable d'environnement et l'installation d'une extension standard signifie simplement copier jess.jar dans votre répertoire $ (JAVA_HOME) / jre / lib / ext.

III.2.4 L'environnement du développeur JessDE

Jess 7 inclut un environnement de développement basé sur Eclipse. Il comporte un éditeur, un débogueur et un visualiseur. D'autres composants comme un navigateur de règles et d'autres outils seront peut être inclus dans les prochaines versions. **[15]**

III.2.4.1 Installation du JessDE

L'environnement du développeur Jess (JessDE) est fourni sous la forme d'un ensemble de plugins pour Eclipse version 3.1 ou ultérieure.

Pour installer JessDE, décompressez tous les fichiers de Jess71p1 / eclipse dans le répertoire d'installation Eclipse. Confirmez qu'un répertoire nommé "plugins / gov.sandia.jess_7.1.0" existe dans votre répertoire d'installation Eclipse, puis redémarrez Eclipse.

Vous pouvez ensuite créez votre premier projet à l'aide de l'assistant "Nouveau projet" pour créez un nouveau fichier dans ce projet nommez-le a titre d'exemple "hello.clp". Il devrait ouvrir dans l'éditeur Jess, qui a une petite boule d'argent avec un "J" rouge comme icône.

Remarque : Si vous effectuez une mise à jour à partir d'une version précédente de JessDE, vous devez lancer Eclipse avec le commutateur de ligne de commande "-clean" pour le forcer à mettre à jour les informations qu'il met en cache à propos des plugins JessDE. Si vous ne le faites pas, de nombreuses options de JessDE peuvent être désactivées. Vous avez seulement besoin de le faire une fois après l'installation. **[15]**

III.2.5 Bases de la langue Jess

La plupart du temps, vous écrivez des règles Jess dans le langage des règles Jess. Jess peut sembler un peu étrange au début, mais il ne faut pas longtemps pour apprendre. il est très expressif et peut exprimer des relations logiques complexes avec très peu de code.

III.2.5.1 Symboles en Jess

Les symboles ressemblent beaucoup à des identifiants dans d'autres langues. Un symbole Jess peut contenir des lettres, des chiffres et la ponctuation suivante: $ * = + / <> _ ? #.

Un symbole ne peut pas commencer par un nombre; il peut commencer par quelques signes de ponctuation (certains ont une signification particulière en tant qu'opérateurs lorsqu'ils apparaissent au début d'un symbole).

Les symboles Jess sont sensibles au majuscule et minuscule : Doo, DOO et doo sont tous des symboles différents. Les meilleurs symboles se composent de lettres, de chiffres, de traits de soulignement et de tirets. les tirets sont des séparateurs de mots traditionnels. **[15]**

Exemple : Foo first-value contestant#1 _abc

Il y a trois symboles spéciale: nil, qui s'apparente quelque peu à la valeur nulle de Java; et TRUE et FALSE, qui sont des valeurs booléennes dans Jess.

III.2.5..2 Nombres en Jess

Jess utilise les fonctions Java parseInt (java.lang.String), parseLong (java.lang.String) et parseDouble (java.lang.String) pour analyser les nombres entiers, longs et à virgule flottante, respectivement.

Exemple : 3 4. 5.643 5654L 6.0E4 1D

III.2.5.3 Chaines de caractères en Jess

Les chaînes de caractères dans Jess sont notées en utilisant des guillemets doubles (") .Les antislashs (\) peuvent être utilisées pour échapper les symboles de citation incorporés. Notez que les chaînes de Jess sont différentes des chaînes Java de plusieurs manières importantes car on ne peut pas intégrer un saut de ligne dans une chaîne en utilisant "\ n", par exemple. Les retours à la ligne réels sont autorisés à l'intérieur de chaînes entre guillemets doubles, ils deviennent une partie de la chaîne.

Exemple : "doo" "Hello, World" "\"Nonsense,\" he said firmly." "Hello,There"

La dernière chaîne est équivalente à la chaîne Java "Hello,\nThere".

III.2.5.4 Listes en Jess

Dans Jess, une liste se compose toujours d'un ensemble de parenthèses inclut zéro ou plusieurs symboles, nombres, chaînes ou autres listes.

Exemple : (+ 3 2) (a b c) ("Hello, World") () (deftemplate foo (slot bar))

III.2.5.5 Commentaires en Jess

Jess prend en charge deux types de commentaires de programmeur: les commentaires de lignes de style Lisp et les commentaires de bloc de style C. Les commentaires de ligne commencent par un point-virgule (;) et s'étendent jusqu'à la fin de la ligne de texte. **[15]**

Exemples :
Commentaire de ligne:

```
; This is a list
(a b c)
```

Commentaires de bloc fonctionnent

```
/*
  Here is an example of a list:
  (a b c)
*/
```

III.2.5.6 Appel d'une fonction en Jess

Tout le code de Jess (structures de contrôle, affectations, appels de procédure) prend la forme d'un appel de fonction. Il n'y a pas d '"opérateurs"; tout est un appel de fonction. Cependant, certaines fonctions ont des noms qui ressemblent à des opérateurs Java, et dans ces cas, ils fonctionnent comme leurs homologues Java.

Les appels de fonction dans Jess sont simplement des listes. Les appels de fonction utilisent une notation de préfixe; une liste dont la tête est un symbole qui est le nom d'une fonction existante peut être un appel de fonction. Par exemple, une expression qui utilise la fonction + pour ajouter les nombres 2 et 3 serait écrite (+ 2 3). Lorsqu'elle est évaluée, la valeur de cette expression est le nombre 5 (pas une liste contenant l'élément unique 5!)

Exemples

```
Jess> (+ 2 3)
5
Jess> (+ (+ 2 3) (* 3 3))
14
```

Vous pouvez imbriquer des appels de fonction; la fonction externe est responsable de l'évaluation des appels de fonctions internes.

Plusieurs fonctions sont intégrées dans Jess, l'une d'elles est la fonction d'impression, qui est utilisée pour envoyer du texte à la sortie standard de Jess, ou à un fichieret l'autre fonction utile est batch, qui évalue un fichier de code Jess. **[15]**

Exemples

```
Jess> (printout t "The answer is " 42 "!" crlf)
The answer is 42!

Jess> (batch "examples/jess/hello.clp")
Hello, world!
```

III.2.5.7 Variables en Jess

Les variables de programmation dans Jess sont des identifiants qui commencent par le point d'interrogation (?). Le point d'interrogation fait partie du nom de la variable. Le nom peut contenir n'importe quelle combinaison de lettres, de chiffres, de tirets (-), de traits de soulignement (_), de deux points (:) ou d'astérisques (*).

Une variable peut faire référence à un seul symbole, numéro ou chaîne, ou à une liste. Vous pouvez assigner une valeur à une variable en utilisant la fonction de liaison: bind. Les variables ne doivent pas (et ne peuvent pas) être déclarées avant leur première utilisation (sauf pour les variables spéciales appelées defglobals).

Pour voir la valeur d'une variable à l'invite Jess>, vous pouvez simplement taper le nom de la variable.

Exemples

```
Jess> (bind ?x "The value")
"The value"

Jess> (bind ?a 123)
123
Jess> ?a
123
```

III.2.5.8 Variables globales (ou defglobals)

Toutes les variables que vous créez à l'invite Jess> ou au «niveau supérieur» de n'importe quel programme en langage Jess sont effacées chaque fois que la commande de réinitialisation "reset" est émise.

Pour créer des variables globales qui ne sont pas détruites par "reset", vous pouvez utiliser la construction defglobal. (defglobal [? <nom-global> = <valeur>] +)

Les noms de variables globaux doivent commencer et se terminer par un astérisque. Lorsqu'une variable globale est créée, elle est initialisée à la valeur donnée. Lorsque la commande de réinitialisation "reset" est ensuite appelée, la variable peut être réinitialisée à cette même valeur, en fonction du paramètre actuel de la propriété reset-globals. **[15]**

Exemples

```
Jess> (defglobal ?*x* = 3)
TRUE
Jess> ?*x*
3
Jess> (bind ?*x* 4)
4
Jess> ?*x*
4
Jess> (reset)
TRUE
Jess> ?*x*
3
```

```
Jess> (bind ?*x* 4)
4
Jess> (set-reset-globals nil)
FALSE
Jess> (reset)
TRUE
Jess> ?*x*
4
```

III.2.5.9 Structures de contrôle

En Java, les flux de contrôle (branchement et bouclage, gestion des exceptions, etc.) sont gérés par une syntaxe spéciale et des mots clés tels que if, while, for et try.

Dans Jess, comme nous l'avons déjà dit, tout est un appel de fonction, et le flux de contrôle ne fait pas exception. Par conséquent, Jess inclut des fonctions nommées if, while, for, et try, ainsi que d'autres comme foreach. Chacune de ces fonctions fonctionne de manière similaire à la construction Java du même nom. **[15]**

Exemples

Une simple boucle

Par exemple, une boucle "while" de Jess ressemble à ceci:

```
Jess> (bind ?i 3)
3
Jess> (while (> ?i 0)
        (printout t ?i crlf)
        (-- ?i))
3
2
1
FALSE
```

Décisions et branchements

```
Jess> (bind ?x 1)
1
Jess> (if (> ?x 100) then
        (printout t "X is big" crlf)
      else
        (printout t "X is small" crlf))
X is small
```

Structure if-else if

```
Jess> (bind ?x 75)
75
```

```
Jess> (if (> ?x 100) then
        (printout t "X is big" crlf)
      else (if (> ?x 50) then
             (printout t "X is medium" crlf)
           else
             (printout t "X is small" crlf)))
X is medium
```

III.2.5.10 Définir des fonctions dans Jess

Vous pouvez définir vos propres fonctions dans jess en utilisant la construction deffunction :

```
(deffunction <function-name> [<doc-comment>] (<parameter>*)
  <expr>* [<return-specifier>])
```

Le champ < function-name > doit être un symbole, il désigne le nom de la fonction. L'option <doc-comment> est une chaîne qui peut décrire le but de la fonction. Chaque paramètre doit être un nom de variable. Il peut y avoir un nombre arbitraire d'expressions <expr>. L'option <return-specifier> donne la valeur de retour de la fonction. **[15]**

Exemple

```
Jess> (deffunction max (?a ?b)
  (if (> ?a ?b) then
     (return ?a)
  else
     (return ?b)))
TRUE
```

III.2.5.11 Les Facts en Jess

Dans Jess, il y a trois sortes de faits: les faits non ordonnés (unordered facts), les faits cachés (shadow facts) et les faits ordonnés (ordered facts).

Vous pouvez voir une liste de tous les faits dans la mémoire de travail en utilisant la commande facts . Les faits sont ajoutés en utilisant les fonctions assert, add et definstance.

Vous pouvez supprimer des faits avec les fonctions retract et undefinstance. La fonction de modification vous permet de modifier les valeurs d'emplacement des faits déjà dans la mémoire de travail.

Et enfin, vous pouvez complètement effacer tous les faits et d'autres données en utilisant la commande clear. **[15]**

Modèles

Dans Jess, chaque fait a un modèle qui est quelque chose comme une classe Java. Pour créer un modèle vous utilisez la construction deftemplate ou la fonction defclass. D'autres fois, les modèles sont créés automatiquement pour vous, soit lorsque vous définissez une defrule, soit lorsque vous utilisez la fonction add ou la méthode jess.Rete.add (java.lang.Object).

Unordered facts

Avant de pouvoir créer des faits non ordonnés, vous devez définir les emplacements qu'ils utilisent en utilisant la structure deftemplate. À titre d'exemple, définissant le modèle suivant:

```
Jess> (deftemplate automobile
  "A specific car."
  (slot make)
  (slot model)
  (slot year (type INTEGER))
  (slot color (default white)))
```

La clause extends facultative de la construction deftemplate vous permet de définir un modèle en termes d'un autre. Par exemple, vous pouvez définir une auto-utilisée comme une sorte d'automobile avec plus de données:

```
Jess> (deftemplate used-auto extends automobile
  (slot mileage)
  (slot blue-book-value)
  (multislot owners))
TRUE
```

Maintenant vous pouvez définir un fact de la manière suivante :

```
Jess> (reset)
Jess> (assert (automobile (model LeBaron) (make Chrysler)
      (year 1997)))
<Fact-1>
Jess> (facts)
```

```
f-0   (MAIN::initial-fact)
f-1   (MAIN::automobile (make Chrysler) (model LeBaron)
                (year 1997) (color white))
For a total of 2 facts in module MAIN.
```

Shadow facts: Raisonnement sur des objets java

Ici, les faits sont cachés et doivent avoir un modèle. Jess crée le modèle automatiquement en regardant une classe Java. **[15]**

Exemple

```
import java.io.Serializable;

public class Account implements Serializable {
   private float balance;
   public float getBalance() { return balance; }
   public void setBalance(float balance) {
      this.balance = balance;
   }
   // Other, more interesting methods
}
```

Ensuite il faut créer un modèle

```
Jess> (deftemplate Account
   (declare (from-class Account)))
```

Ensuite on peut automatiquement ajouter des slots correspondant aux propriétés JavaBeans de la classe Account. Dans notre exemple il y aura un slot nommé balance correspondant à la méthode getBalance ().

La fonction defclass peut être utilisée à la place pour créer un modèle

```
Jess> (defclass Account Account)
```

III.2.5.12 Règles en Jesss

Les réles peuvent agir en fonction du contenu d'un ou de plusieurs faits. Les règles sont définies dans Jess en utilisant la construction defrule. Une règle très simple ressemble à ceci:

```
Jess> (defrule welcome-toddlers
   "Give a special greeting to young children"
   (person {age < 3})
```

```
=>
(printout t "Hello, little one!" crlf))
```

Les règles Jess ne se déclenchent que lorsque le moteur de règles est en cours d'exécution (bien qu'elles puissent être activées lorsque le moteur ne fonctionne pas). Pour démarrer le moteur, nous lançons la commande **run**.

Les règles sont identifiées de manière unique par leur nom. Si une règle déclarée porte le même nom d'une règle ancienne, la première version est supprimée et ne se déclenche plus, même si elle a été activée au moment de la définition de la nouvelle version. **[15]**

III.2.5.13 Interrogation de la mémoire de travail

La plupart du temps, vous accéderez à la mémoire de travail en faisant correspondre les motifs d'une règle. Mais de temps en temps, vous allez écrire un code de procédure qui doit extraire directement les données de la mémoire de travail.

Recherche linéaire
Une façon traditionnelle de rechercher une collection d'éléments est d'utiliser la recherche linéaire et un filtre jess.Filter, une fonction booléenne qui indique si un élément doit faire partie du résultat de la recherche ou non.

La construction de defquery
Bien que la recherche linéaire soit pratique mais elle est inefficace. La construction defquery vous permet de créer un type particulier de règle sans partie droite. Une requête vous donne un objet jess.QueryResult qui vous donne accès à une liste de toutes les correspondances. L'utilisation d'une defquery implique trois étapes:

- Ecrire la requête
- Invocation de la requête
- Utiliser les résultats

Exemple :
Ecrire la requête
On définie d'abord la Template Person
Jess> (deftemplate person (slot firstName) (slot lastName) (slot age))

Si on veut trouver des personnes avec un nom donné. Une fois ce nom est trouvé on peut aussi récupérer son prénom et son âge.

Nous écrivons un modèle qui correspond aux faits qui nous intéressent. Nous déclarons la variable? Ln, ce qui en fait un paramètre de la requête, et nous incluons également les variables? fn et? age liées aux slots firstName et age, respectivement. **[15]**

```
Jess> (defquery search-by-name
  "Finds people with a given last name"
  (declare (variables ?ln))
  (person (lastName ?ln) (firstName ?fn) (age ?age)))
```

Ensuite on peut remplir la base des faits par:

```
Jess> (deffacts data
  (person (firstName Fred)   (lastName Smith)    (age 12))
  (person (firstName Fred)   (lastName Jones)    (age 9))
  (person (firstName Bob)    (lastName Thomas)   (age 32))
  (person (firstName Bob)    (lastName Smith)    (age 22))
  (person (firstName Pete)   (lastName Best)     (age 21))
  (person (firstName Pete)   (lastName Smith)    (age 44))
  (person (firstName George) (lastName Smithson) (age 1))
  )
Jess> (reset)
```

Invocation de la requête

Maintenant, nous pouvons appeler la requette en utilisant la fonction run-query * dans Jess, ou la méthode jess.Rete.runQueryStar (java.lang.String, jess.ValueVector) en Java. Dans les deux cas, nous devons transmettre le nom de famille qui nous intéresse en tant que paramètre de requête. Dans Jess, cela ressemble :

```
Jess> (reset)
Jess> (bind ?result (run-query* search-by-name Smith))
```

Les arguments de run-query * sont le nom de la requête et les valeurs de chaque paramètre. La fonction run-query * renvoie un objet jess.QueryResult, que nous stockons dans la variable? Result. En Java, le code équivalent ressemblerait à ceci:

```
Rete engine = ...
 QueryResult result = engine.runQueryStar("search-by-name", new
ValueVector().add("Smith"));
```

Utiliser les résultats

L'interface de la classe jess.QueryResult est très similaire à celle de java.sql.ResultSet. Nous utilisons jess.QueryResult.next () pour passer au résultat suivant, et nous utilisons l'ensemble des méthodes getXXX () pour renvoyer les valeurs liées aux variables définies dans la requête.

```
Jess> (while (?result next)
```

```
   (printout t (?result getString fn) " " (?result getString ln)
            ", age " (?result getInt age) crlf))
Fred Smith, age 12
Bob Smith, age 22
Pete Smith, age 44
FALSE
```

La même boucle en Java ressemblerait

```
while (result.next()) {
   System.out.println(result.getString("fn") + " " + result.getString("ln")
                  + ", age" + result.getInt("age"));
}
```

III.3 Jade

JADE (Java Agent DEvelopment Framework) est un Framework logiciel entièrement implémenté en langage Java. Il simplifie la mise en œuvre des systèmes multi-agents. Un système basé sur JADE peut être distribué sur des machines (qui n'ont même pas besoin de partager le même système d'exploitation).

Jade est entièrement implémentée en langage Java et la configuration minimale requise est la version 5 de JAVA (environnement d'exécution ou JDK). JADE est un logiciel libre distribué par Telecom Italia, le détenteur des droits d'auteur, en source ouverte, conformément aux conditions générales de la licence LGPL. JADE contient : **[16]**

- **Un runtime Environment** : l'environnement ou les agents peuvent vivre.
- **Une librairie de classes** : que les développeurs utilisent pour écrire leurs agents
- **Des outils graphiques** : qui facilitent la gestion de la plateforme des agents

Une plateforme sous jade est constituée d'un ensemble de conteneurs (instance du JADE). Chaque plateforme doit contenir un conteneur spécial appelé main-container et tous les autres conteneurs s'enregistrent auprès de celui-là dés leur lancement. Un main-container contient toujours trois agents spéciaux appelés RMA, AMS et DF qui sont lancés automatiquement.

- **RMA (**Remote Management Agent**)** qui se préoccupe de gestion de l'interface graphique de la plateforme.
- **AMS** (Agent Management System) qui fournit le service de nommage (pour assurer par exemple que chaque agent possède un identifiant unique dans la plateforme) et qui représente l'autorité de la plateforme (par exemple il est possible de créer/arrêter des agents en envoyant des requêtes à l'AMS). **[17]**
- **DF** (Directory Facilitator) qui fournit un système de pages jaunes qui permet aux agents de retrouver les agents fournisseurs de services.

III.3.1 Configuration de la plateforme jade

Voici les étapes à suivre pour installer JADE :

Téléchargez le fichier JADE-all-3.6.zip de l'adresse suivante : http://jade.tilab.com/download.php

Décompressez le fichier (Exemple dans C:\ JADE-all-3.6) qui se compose de quatre autres fichiers ZIP (JADE-bin-3.6.zip , JADE-doc-3.6.zip, JADE-examples-3.6.zip, JADE-src-3.6.zip). Décompressez ces 4 fichiers

Mettre à jour la variable classpath : Dans la zone variable système, essayez de trouver la variable d'environnement qui porte le nom CLASSPATH ou il faut la créer si elle n'existe pas. Ensuite vous l'attribuer une valeur qui est la concaténation des chemins des quatre fichiers jar http.jar, iiop.jar, jade.jar, jadeTools.jar situés dans le chemin C:\JADE-all-3.6\JADE-bin-3.6\jade\lib séparé par des virgules. **[16]**

Pour vérifier que l'opération est bien réalisée, tapez dans la fenêtre exécuter (Démarrer->Exécuter) ou dans l'invite de commande, la commande suivante : Java jade.Boot –gui. La fenêtre qui présente la plateforme jade doit être lancée :

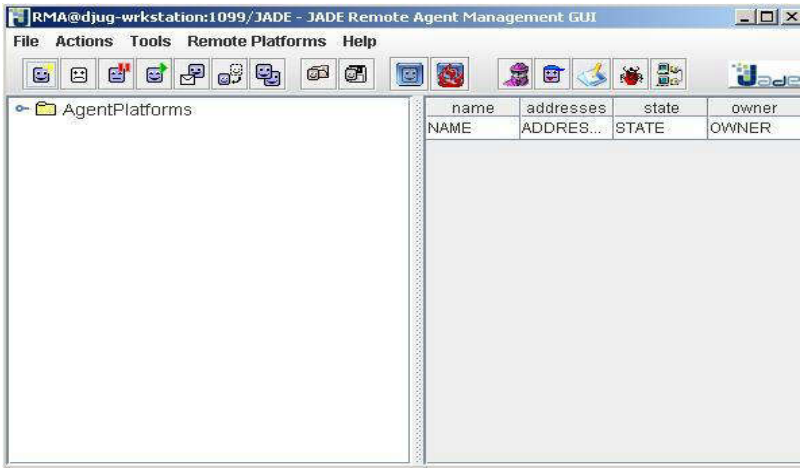


Figure III.1 : Plateforme Jade

III.3.2 Premier Agent en Jade

Un agent Jade est défini en tant que sous-classe de la classe prédéfinie Agent et son code principal doit être placé ou appelé dans une méthode appelée setup. Il suffit juste d'importer la classe Agent. Voici un exemple d'un agent nommé HelloWorldAgent qui imprime un simple message de hello suivi de son nom.

```
package NomDuPackage;
import jade.core.Agent;

public class HelloWorldAgent extends Agent {

 protected void setup() {
        System.out.println("Hello World! My name is "+getLocalName());
        // Make this agent terminate
        doDelete();
 }
}
```

La fonction doDelete() est ajouté pour terminer l'exécution de l'agent dans la plateforme.

Si des erreurs apparaîtraient vous ajoutez les quatre jars "http.jar" "iiop.jar" "jade.jar" "jadeTools.jar" situés dans C:\JADE-all-3.6\JADE-bin-3.6\jade\lib à votre projet on cliquant sur java build path >> Libraries>> add external JARs sous Eclipse.

Pour compiler et lancer l'agent sous Eclipse, vous allez dans run>>Run configuration puis double cliquez sur java application et dans l'onglet " main " et dans la zone de saisie Main class, tapez le code suivant : jade.Boot puis cochez la case : " Include librairies when searching for a main class " et dans l'onglet arguments : tapez le code suivant : -gui jade.Boot NomDuL'agent:LeNomDuPackage.LeNomDeLaClasse puis cliquez sur apply pour ne pas refaire cette configuration plusieurs fois dans le même projet . Ensuite vous cliquez sur run pour lancer l'exécution et voir le résultat.
L'agent est identifié auprès de l'AMS par un identifiant unique appelé GUID (Global Unique IDentifier).Cet identifiant est un objet de type jade.core.AID

Voici une liste de quelques fonctions de la plateforme jade : **[16]**

getAID() : retourne l'identifiant unique de l'agent
getName() : retourne le nom de l'agent, < nomlocal > @ < plateforme >
getLocalName() : retourne uniquement le nom locale
getHap() : retourne l'adresse de la plateforme
getAMS() : retourne l'AID de l'AMS de la plateforme sur laquelle l'agent est situé.

Plugin EJADE

EJADE est un plugin Eclipse qui permet aux développeurs d'agents sous JADE et JADEX de lancer la plateforme et déployer des agents facilement. Sa façon de configuration est plus simple et très utile, il vous suffit de sélectionner les fichiers d'agent situés n'importe où dans votre projet, d'effectuer un clic droit et de les déployer et vous n'avez pas besoin de vous soucier de classpath, utilisateur lib ou d'autres configurations. Il est peut être configuré juste en trois simples étapes : **[18]**

- Télécharger le fichier compressé ejade
- Décompresser le
- Copier le fichier it.fbk.sra.ejade_x.x.x dans le fichier Eclipse/Plugins

La version: 0.8.0 est livré avec JADE v3.6, JADEX v0.96.

III.3.3 Arguments d'un Agent sous Jade

Une liste d'arguments peut être passée à un agent lors de son lancement. En Jade, les arguments peuvent être récupérés par la méthode getArguments() de la classe Agent qui renvoie un tableau d'objets. **[17]** Le passage d'arguments peut se faire par la commande suivante :

java jade.Boot NomDuL'agent:LeNomDuPackage.LeNomDeLaClasse (arg1 arg2)

Les arguments sont séparés par des espaces

```
protected void setup() {
Object[] args = getArguments();
if (args != null) { for (int i=0; i.args.length; i++) { … }
}}
```

III.3.4 Comportement d'un Agent sous Jade

Un agent peut posséder un ou plusieurs comportements. Jade exécute les agents en tant que processus.

Les comportements "behaviours" définissent le comportement de l'agent. Ols sont implémenté comme un objet de la classe Jade.core.Behaviours. L'ajout d'un comportement est réalisé par la méthode addBehaviour() dans la méthode setup(), comme il peut être appelé à partir d' un autre comportement (Behaviour) de l'agent.

Chaque Behaviour doit implémenter au moins les deux méthodes :

- action() : qui désigne les opérations à exécuter par le Behaviour
- done() : qui retourne vrai lorsque la tâche implémentée par le comportement est terminée.

Il existe deux autres méthodes dont l'implémentation n'est pas obligatoire mais qui peuvent être très utiles :

- onStart() : appelée juste avant l'exécution de ma méthode action();
- onEnd() : appelée juste après le retournement de true par la méthode done().

Il existe plusieurs types de comportements, parmi eux , on trouve : **[16]**

OneshotBehaviour : qui est une instance de la classe jade.core.behaviours.OneShotBehaviour. Les tâches seront exécutées une seule fois puis il se termine. La classe OneShotBehaviour implémente par default la méthode done() et elle retourne toujours true.

Cyclic Behaviour : les tâches seront exécutées d'une manière répétitive. Le cyclic behaviour est une instance de la classe jade.core.behaviours.CyclicBehaviour. La classe CyclicBehaviour implémente la méthode done() qui retourne toujours false.

Generic Behaviour : Un Generic Behaviour est une instance de la classe jade.core.behaviours.Behaviour. Le Generic Behaviour n'implémente pas par default la méthode done() qui sera ajouté par le programmeur lui même afin de planifier la condition qui termine l'exécution de son Behaviour.

L'exemple suivant illustre l'utilisation de ces trois types des comportements

```
import jade.core.Agent;
import jade.core.behaviours.Behaviour;
import jade.core.behaviours.CyclicBehaviour;
import jade.core.behaviours.OneShotBehaviour;

public class SimpleAgent extends Agent {
 protected void setup() {
 //OneShotBehaviour
            addBehaviour(new OneShotBehaviour(this){
                        public void action(){
  System.out.println("OneShotBehaviour"+getLocalName()); }});

  // CyclicBehaviour
                    addBehaviour(new CyclicBehaviour(this) {
                    public void action() {  System.out.println("cyclique... "); } });

  // generic behaviour
  addBehaviour(new GBehaviour ());  }


 private class GBehaviour extends Behaviour {
  int a=0 ;

  public void action() { System.out.println("la valeur de a est :"+ a); }

  public boolean done() {  return a == 5;  }

  public int onEnd() {
   myAgent.doDelete();
   return super.onEnd(); } } }
```

III.3.5 Utilisation des services

Pour qu'un agent puisse offrir des services, il faut qu'il se référencer auprès du DF pour décrire les services qu'il fournit. **[16]**

```
DFAgentDescription dfd = new DFAgentDescription();
 dfd.setName(this.getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("TypeService");
dfd.addServices(sd);
DFService.register(this, dfd);
```

l'agent peut se désinscrire du DF à sa sortie du système par l'utilisation de la méthode deregister() qui deenregistre sa présence au sein du DF.

```
DFService.deregister(MonAgent);
```

Un tel agent peut rechercher la description d'agents fournissant un service particulier (*TypeService* dans notre exemlple)

```
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd= new ServiceDescription();
sd.setType("TypeService");
dfd.addServices(sd);
DFAgentDescription[] results = DFService.search(this, dfd);
 for (int i = 0; i < results.length; i++) System.out.println(results[i].getName());
```

III.3.6 Communication en Jade

Jade utilise le FIPA ACL comme un langage de communication qui a une syntaxe proche du KQML. Un message est une instance de la classe ACLMessage appartenant au package jade.lang.acl . La définition d'un ACLMessage vérifie les spécifications FIPA.

Il est composé au minimum d'une performative, d'un ensemble de destinataires, et contient de préférence un contenu, un expéditeur, un protocole d'interaction et un id de conversation. Comme Il peut spécifier un langage de contenu et/ou une ontologie. **[17]**

III.3.6.1 Création et envoi de messages

Les performatifs sont des variables statiques de la classe ACLMessage comme INFORM , FAILURE, REQUEST et SUBSCRIBE. Un message est peut être créé comme instance de la classe ACLMessage.

```
ACLMessage inform = new ACLMessage(ACLMessage.INFORM);
inform.setContent("contenu");
inform.setProtocol("information");
```

Ensuite il faut Ajouter les destinataires par la méthode addReceiver() avant d'envoyer du message par appel de la méthode send() de la classe Agent.

```
inform.addReceiver(new AID("destinataire", AID.ISLOCALNAME));
inform.addReceiver(new AID("destinataire2", AID.ISLOCALNAME));
send(inform);
```

III.3.6.2 Réception d'un message

La réception d'un message sous jade peut se faire de ceux manières :

Réception bloquante: La méthode ACLMessage m = blockingReceive()se bloque jusqu'à ce qu'un message arrive. L'agent passe dans l'état 'Waiting' jusqua la réception d'un nouveau message et par conséquent tous ses comportement seront arrêtés jusqu'à la réception d'un nouveau message. **[16]**

```
ACLMessage receive = blockingReceive();
```

Réception non-bloquante: La méthode ACLMessage m = receive() retourne null si la file de message est vide. L'agent reste dans l'état 'Active' et son comportement reste activé, et exécuté de façon cyclique, même si aucun message n'a été reçu. L'ajout de la méthode block () suspend le comportement jusqu'à la réception d'un nouveau message.

```
ACLMessage receive = receive();
 if(receive != null){ … }
else{ block(); }
```

III.4 Jadex

Jadex est un moteur de raisonnement orienté agent pour l'écriture d'agents rationnels avec XML et le langage de programmation Java. Jadex est une plateforme pour développer et manipuler des agents BDI.

L'objectif est de soutenir la construction des systèmes multi-agents en utilisant des notions mentales.

Pour développer des applications avec Jadex, le programmeur doit créer deux types de fichiers: les fichiers de définition d'agent XML (ADF) et les classes Java pour les implémentations du plan. **[34]**

III.4.1 Démarrage d'un agent

La configuration de l'environnement Jadex est assez simple et peut être effectuée en quelques étapes simples. **[19]**

Dans cette Démonstration, nous utiliserons la version Standalone de Jadex. La distribution Jadex doit être extraite dans un répertoire local, appelé JADEX_HOME à titre d'exemple. Ensuite, vous avez essentiellement deux options pour démarrer Jadex.

- La première consiste à configurer manuellement l'environnement java dans la CLASSPATH sur tous les fichiers jars contenus dans le répertoire JADEX_HOME / lib.
- Le second est en utilisant l'option -jar lors du démarrage via la commande java : java jadex.standalone.Platform ou java -jar jadex-launch-3.0.0-RC1.jar

III.4.2 Création d'un Projet Jadex

Après le démarrage de la plate-forme Jadex avec la commande expliquée ci-dessus, le Jadex Control Center (JCC) devrait apparaître avec son interface utilisateur.

Un projet contient des paramètres sur les dossiers de projet utilisés ainsi que des paramètres d'outils divers et d'interface utilisateur. Tous ces paramètres (paramètres de la fenêtre, les chemins ajoutés ... etc.) seront stockés dans un fichier nommé project.xml.

D'autres fichiers de propriétés supplémentaires seront créés automatiquement pour stocker les paramètres individuels pour les plugins que vous utilisez.

Pour ajouter des fichiers à votre projet, cliquez sur le bouton "Ajouter un chemin". Pour enregistrer les paramètres du projet sur le disque, sélectionnez "Enregistrer les paramètres" dans le menu "Fichier".

De plus, les réglages sont également sauvegardés automatiquement lors de la fermeture du JCC via le bouton de fermeture de la fenêtre ou "Quitter" dans le menu

"Fichier". Vous pouvez également utiliser plusieurs projets avec des paramètres différents. Pour changer de projet, cliquez simplement sur "Load Settings from File" dans le menu "File" et choisissez le fichier de paramètres du projet avec lequel vous voulez travailler.

Quelques exemples d'applications BDI disponibles dans le JCC

- **Alarmclock**: Un agent qui affiche une petite interface utilisateur d'horloge et capable de gérer les heures d'alarme et de lire des chansons MP3 lorsqu'une alarme est due.
- **Blocksworld**: empilement de blocs sur une table pour atteindre une configuration cible spécifique de blocs.
- **Booktrading**: les agents acheteurs et vendeurs qui négocient les prix des livres.
- **Hunterprey**: Les chasseurs recherchent des proies (nourriture vivante) dans un monde virtuel.
- **Marsworld**: Exploitation coopérative du minerai sur Mars par différents types de robots.
- **Ping**: agents simples qui envoient des messages ping et des messages de réponse.
- **Puzzle** : Un agent qui essaie de résoudre un casse-tête en essayant des mouvements et en les ramenant éventuellement. **[19]**

III.4.3 Création du premier Agent Jadex

Ouvrez un éditeur de code source ou un IDE de votre choix

Créez un nouveau fichier de définition d'agent (ADF) appelé TranslationA1.agent.xml on utilison eclipse avec un plug-in web-tools pour éditer des ADF ou un autre éditeur XML avancé.

Dans ce fichier, toutes les propriétés importantes de démarrage d'agent sont définies d'une manière conforme à la spécification de schéma Jadex. La première propriété de l'agent est son nom qui doit être le même que le nom du fichier (similaire aux fichiers de classe Java), dans ce cas, il est défini sur TranslationA1

```
<!-- A simple translation agent. -->
<agent xmlns="http://jadex.sourceforge.net/jadex"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://jadex.sourceforge.net/jadex
              http://jadex.sourceforge.net/jadex-bdi-2.3.xsd"
 name="TranslationA1"
 package="jadex.bdi.tutorial">
</agent>
```

Après la création du fichier xml, Démarrez le JCC et utilisez le bouton "Ajouter un chemin" pour ajouter le répertoire racine de votre exemple de package. Puis ouvrez le dossier jusqu'à ce que vous puissiez voir votre fichier "TranslationA1.agent.xml".

III.5 Configuration des variables d'environnements java : JAVA_HOME

Voici les étapes à suivre pour configurer les variables d'environnement java

- Faites un clic avec le bouton droit de la souris sur l'icône de Mon ordinateur. Puis, sélectionnez l'option Propriétés.
- Cliquez ensuite sur l'onglet Avancé.
- Sélectionnez maintenant le bouton Variables d'environnement.
- Cliquez sur Nouveau en dessous des variables du système
- Écrivez le nom de la nouvelle variable : JAVA_HOME.
- Copiez maintenant la valeur du chemin d'installation java du genre : C:\Program Files\Java\jdk1.7.0_13\jre
- Cliquez ensuite sur le bouton OK.
- Sélectionnez ensuite Appliquer.
- Redémarrez votre ordinateur afin de vous assurer que vos changements ont été pris en compte.

Conclusion Générale

Actuellement, Les systèmes intelligents sont très répandus et présents dans presque tous les domaines de notre vie quotidienne. Aujourd'hui, de nombreuses applications basé sur ces systèmes dit intelligent se présentent notamment dans les téléviseurs numériques, les feux de circulation, les compteurs intelligents, les voitures, ou même dans les nouveaux systèmes de gouvernances.

Ce polycopié présente un aperçu sur les concepts de base de l'intelligence artificielle et les systèmes multi agents afin de mieux maitriser la conception et le fonctionnement d'un tel système dit intelligent ?

Les systèmes intelligents ont la capacité de calculer, de raisonner et d'apprendre grâce à l'expérience. Ils peuvent même stocker et récupérer des informations, résoudre des problèmes et aussi ils sont capable d'utiliser le langage naturel pour communiquer.

Ce polycopié est constitué de deux chapitres théoriques qui discutent l'intelligence artificielle et l'intelligence artificielle distribuée et un autre chapitre qui présente des plateformes logicielles qui sont utilisé pour s'entrainer aux concepts de l'intelligence artificielle via des exemples d'illustration.

Bibliographie & Webographie

[1] François Yves Villemin, *Intelligence Artificielle*, Support de Cours, Département informatique, conservatoire national des arts et métiers, France, 2012.

[2] Patrick Lemaire, André Didierjean, *Introduction à la psychologie cog*nitive, De Boeck Supérieur ISBN: 918-2-8073-0784-1, 2018.

[3] Laure Léger, *Manuel de psychologie cognitive*, ISBN 978-2-10-074363-6, Dunod, Paris, 2016.

[4] Daniel Kayser, *La Représentation des Connaissances*, Hermès - Lavoisier, ISBN13 978-2-86601-647-0 ,1997 .

[5] Hanene Ghorbel Agrebi, Afef Bahri, Rafik Bouaziz, *Les langages de description des ontologies : RDF & OWL* , huitièmes journées scientifiques des jeunes chercheurs en Génie Electrique et Informatique GEI'08 , 2008.

[6] Abdelmoutia Telli , *Raisonnement sur les ontologies légères*, Thèse de Doctorat , Université Mohamed Khider de BISKRA ,2018.

[7] Laurent Audibert, *Traitement Automatique du Langage Naturel (TALN) Outils d'analyse de données textuelles*, Support de cours, Université Paris 13,2010.

[8] John Langshaw Austin, *Quand dire c'est Faire*, tr. fr. 1979 - SEUIL, Coll. Points, 1962.

[9] John R Searle, Les actes de langage, Hermann, coll. Savoir : lettres, 1972.

[10] Jacques Ferber, *Les Systèmes Multi-Agents : Vers une Intelligence Collective,* InterEditions, Paris, 1995.

[11] Nadia Kabachi, *Intelligence Artificielle Distribuée et Systèmes Multi-Agents*, Support de Cours.

[12] Alexandre Grouls, *Agents et Systèmes Multi-Agents: Vers Une Synthèse de Ces Concepts*, Mémoire Présenté Comme Exigence Partielle de La Maîtrise en Informatique, 2013.

[13] Amal El Fallah Segrounichi, Cours online, *Agents intelligents*, Politechnica University of Bucharest , 2002.

[14] Bernard EspinassE, *Cours en Communication et langages de communication dans les SMA*, Aix-Marseille Université (AMU), 2012.

[15] Site officielle Jess : *https://www.jessrules.com/*

[16] Site officielle Jade : *https://jade.tilab.com/*

Bibliographie & Webographie

[17] Fabio Bellifemine, Giovanni Caire, Dominic Greenwood, *Developing Multi-Agent Systems with JADE*, John Wiley & sons, Ltd, 2007.

[18] Site officielle Ejade : http://selab.fbk.eu/dnguyen/ejade/index.html

[19] Lars Braubach, Alexander Pokahr (Distubuted Systems Groub), *Jadex User Guide*, Université de Hambourg , Germany, 2007