



Rapport de TP

Module : Technologie des agents
Master 1 SII

TP

Implémentation d'un système multi-agents de ventes et achats de produits vestimentaires

- Réalisé par :

BENHADDAD Wissam
BOURAHLA Yasser

13-05-2018

Table des matières

I	Système expert et modèle d'inférence	2
1	Introduction	3
1.1	Problématique et besoins	3
1.2	Définitions	4
1.2.1	Base de connaissances	4
1.2.2	Règles de production	4
1.2.3	Moteur d'inférence	4
1.2.4	Système expert	4
2	Implémentation d'un système expert en Java	5
2.1	Outils utilisés	5
2.1.1	Langage de programmation :	5
2.1.2	IDE :	5
2.2	Analyseur de règles de production	5
2.2.1	Analyseur de variables (variableParser)	5
2.2.2	Analyseur de règles (ruleParser)	6
2.3	Modifications apportées	6
2.3.1	Typage des variables	6
2.3.2	Introduction de nouvelles conditions	7
2.4	Interface modifié	7
2.5	Conclusion	7
II	Système multi-agents dans le domaine commercial	8
1	Introduction	9
1.1	Problématique	9
1.2	Définitions	9
1.2.1	Agent intelligent	9
1.2.2	Système multi-agents	10
2	Implémentation d'un système multi-agents avec la plateforme JADE	11
2.1	Outils utilisés	11
2.2	Schéma globale	11
2.2.1	Agent annexe	12
2.2.2	Agent central	12
2.3	Communication entre les agents	12
2.3.1	Communication d'ajout de service	13
2.3.2	Communication de requêtes	15

3	Application dédiée	16
3.1	Interface homme-machine	16
3.2	Conclusion	16
III	L'interpréteur JASON	17
1	Introduction	18
1.1	Problématique et besoins	18
1.2	Définitions	18
1.2.1	Architecture CDI (Croyance-Désir-Intention)	18
1.2.2	AgentSpeakL	18
1.2.3	La plateforme JASON	19
2	Exemples de communication entre agents avec JASON	20
2.1	Environnement de travail	20
2.2	Inférence locale	21
2.3	Scénario simple de communications	21
3	Comparaison avec la plateforme JADE	23
3.1	Principales différence	23

Première partie

Système expert et modèle d'inférence

Chapitre 1

Introduction

1.1 Problématique et besoins

Dans le domaine de l'intelligence artificielle, nous sommes souvent confrontés à la modélisation des connaissances d'un format brute(e.g langage naturel) en un format interprétable par les machines à travers un formalisme mathématique(logique), une des premières propositions fut l'introduction d'un système de règles logiques (ensembles de conditions et leurs conséquences) accompagné d'un mécanisme de déduction(à l'instar de la façon dont l'homme traite un problème) nommé **Système expert**.



FIGURE 1.1

1.2 Définitions

1.2.1 Base de connaissances

Une base de connaissance est un ensemble de connaissance modélisée de telle sorte à être compréhensibles par un ordinateur, elle peut être sous la forme d'une base de règle de productions (voir point suivant), d'une base de faits¹ ou des deux en même temps.

1.2.2 Règles de production

Les règles de production sont des formules logiques de types :

$A_1 \wedge A_2 \wedge \dots \wedge A_n \rightarrow B$ où :

- $A_1 \wedge A_2 \wedge \dots \wedge A_n$ Sont des formules logiques bien formée (de la logique propositionnelle ou celle des prédicats en générale), elle représentent les prémisses (conditions) d'application de la règle.
- B Représente la connaissance déductible des conditions précédentes.
- \rightarrow le symbole de l'implication logique classique

Si toutes les conditions A_i sont vérifiées alors on peut ajouter la nouvelle connaissance B à la base de connaissances.

1.2.3 Moteur d'inférence

Dans un système expert, un moteur d'inférence est un module qui prend en entrée une base de connaissances BC , une base de faits BF et essaye, en appliquant soit l'algorithme de chaînage avant ou bien celui du chaînage arrière (tout dépendant du besoin ou la nature de la question posé par un utilisateur) de trouver un cheminement logique (i.e une succession d'application de règle de production) de telle sorte à arriver une connaissance B en particulier, ou bien à un ensemble de connaissances dérivable des faits introduit en entrée.

1.2.4 Système expert

Un système expert est un outil dont le but est d'imiter le comportement d'un véritable expert humain dans un domaine particulier, et donc de parvenir à travers l'observation de différents faits à répondre à une question ou de proposer un ensemble de solutions à un problème, c'est un outil d'aide à la décision très utile dans le domaine de l'intelligence artificielle.

1. Ensemble de connaissances que le système considère comme vraies.

Chapitre 2

Implémentation d'un système expert en Java

2.1 Outils utilisés

2.1.1 Langage de programmation :

Nous avons opté pour le langage Java, car il offre une grande flexibilité et facilite l'implémentation qui est due au fait qu'il soit totalement orienté-objet.

2.1.2 IDE :

IntelliJ Idea L'environnement de développement choisi est IntelliJ IDEA, spécialement dédié au développement en utilisant le langage Java. Il est proposé par l'entreprise JetBrains et est caractérisé par sa forte simplicité d'utilisation et les nombreux plugins et extensions qui lui sont dédiées.

2.2 Analyseur de règles de production

Afin de minimiser la modification du code, et pour des soucis de gain de temps, nous avons implémenter un mini-analyseur dont le but est de charger le contenu de deux fichiers(variables,rules) dans la base de faits d'un expert, le format adopté est inspiré des langages balisés, ce module se compose de sous modules :

2.2.1 Analyseur de variables (variableParser)

Il s'occupe de prendre en entrée un fichier rempli avec des variables et leurs valeurs possible(dans langage que nous avons mis au point), toute variable est de la forme suivante :

$\langle \text{VarName} : \text{Type} \rangle \text{val}_1, \dots, \text{val}_n, \langle / \text{VarName} \rangle$

- La liste de valeurs possibles $\text{val}_1, \dots, \text{val}_n$, peut être vide.
- **VarName** est le nom de la variable qui servira à l'identifier plus tard dans le programme.
- **Type** est une structure que nous avons conçu pour supporter entre autre les type primitifs : String, Int, Double.. mais aussi les intervalles($[a,b]$, $]a,b]$, $[-\infty, b]$...)

Quelques exemples de variables :

- $\langle \text{Season} : \text{String} \rangle \text{Summer, Winter, Autumn, Spring}, \langle / \text{Season} \rangle$
- $\langle \text{Temperature} : \text{Double} \rangle 28.5, 15.5, \langle / \text{Temperature} \rangle$
- $\langle \text{Price} : \text{Double} \rangle \langle / \text{Temperature} \rangle$

2.2.2 Analyseur de règles (ruleParser)

Ce module prend en entrée un fichier de règles écrite elles aussi dans une langage dédié, toute règle s'écrit comme suit :

$\langle \text{VarResult} \rangle = \langle \text{cond}_1 \rangle, \dots \langle \text{cond}_n \rangle,$

Où VarResult et cond_i ont la même structure suivante :

$\langle \text{VarName} / \text{Type} / \text{cond} / \text{value} \rangle$

- **VarName** est le nom de la variable qui servira à l'identifier plus tard dans le programme.
- **Type** est le même type structuré vu dans 2.2.1
- **cond** est la condition sur la valeur de la variable courante(= , < , > != ..) **value** La valeur suivant le type de la variable

2.3 Modifications apportées

Le système expert de base qui nous a été demandé d'améliorer était très limité(gère les valeurs comme des chaines de caractères uniquement, conditions d'égalité seulement entre les valeurs, ..), nous avons donc décidé d'ajouter certaines fonctionnalités a ce systèmes :

2.3.1 Typage des variables

L'inconvénient en travaillant seulement avec des String est le manque de flexibilité des valeurs, nous avons donc subdivisé le typage en 4 groupes :

- **StringVariableValue** : Type basique, généralement la plus part des variables ont ce type, il peut désigner un catégorie, un nom, une propriété nominale ... etc.
- **IntegerVariableValue** : Type basique pour désigner les valeurs discrètes telle que le nombre l'âge.
- **DoubleVariableValue** : Type basique pour désigner les valeurs continues (beaucoup plus fréquentes) telles que le prix, la température ... etc
- **IntervalUnion** : Type complexe pour désigner tout séquence de valeur non dénombrable comme l'union de plusieurs intervalles ($[0,2] \cup [5,22] \cup [55,69[$), une valeur minimum (value > min) ou maximum ... etc

2.3.2 Introduction de nouvelles conditions

Les conditions utilisables n'étant que l'égalité et l'inégalité (qui ne fonctionnait pas à la base), nous avons opté pour une restructuration du système d'évaluation, avec l'introduction du typage vu précédemment nous avons pu introduire deux nouveaux opérateur de conditions qui sont le $> (\geq)$ et le $< (\leq)$, ces deux opérateur nous ont permis ainsi de délimiter sous forme d'intervalle les valeurs de certaines variables.

2.4 Interface modifié

L'interface de base n'étant pas appropriée à notre besoin, nous avons décidé de la modifier et la rendre adaptée au domaine choisi, plus de détails seront donnée dans la partie suivante

2.5 Conclusion

Les systèmes experts malgré leur simplicité ont prouvé leur efficacité durant les débuts de l'IA, cependant leur manque d'autonomie et de réaction avec le monde extérieur restait encore un obstacle majeur pour atteindre un niveau d'intelligence semblable à celui de l'homme, une extensions des systèmes expert à donc était développée, appelé système à agent-intelligents.

Deuxième partie

Système multi-agents dans le domaine commercial

Chapitre 1

Introduction

1.1 Problématique

1.2 Définitions

1.2.1 Agent intelligent

Un agent est un logiciel qui agit de façon autonome. C'est un programme qui accomplit des tâches à la manière d'un automate et en fonction de ce que lui a demandé son auteur, en revanche un **agent intelligent** est une entité autonome capable de percevoir son environnement grâce à des capteurs et aussi d'agir sur celui-ci via des effecteurs afin de réaliser des buts¹. Un agent intelligent peut également apprendre ou utiliser des connaissances pour pouvoir réaliser ses objectifs.

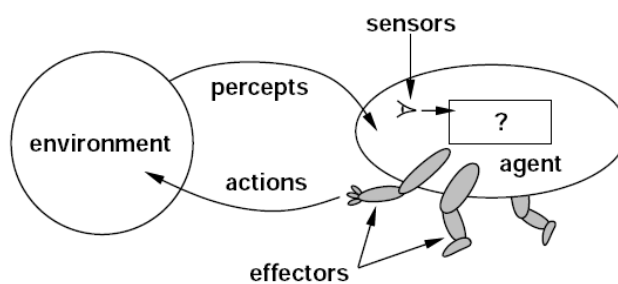


FIGURE 1.1 – Agent intelligent interagissant avec le monde extérieur

1.2.2 Système multi-agents

un système multi-agents (SMA) est un système composé d'un ensemble d'agents (un processus, un robot, un être humain, etc.), situés dans un certain environnement et interagissant selon certaines relations. Un agent est une entité caractérisée par le fait qu'elle est, au moins partiellement, autonome.

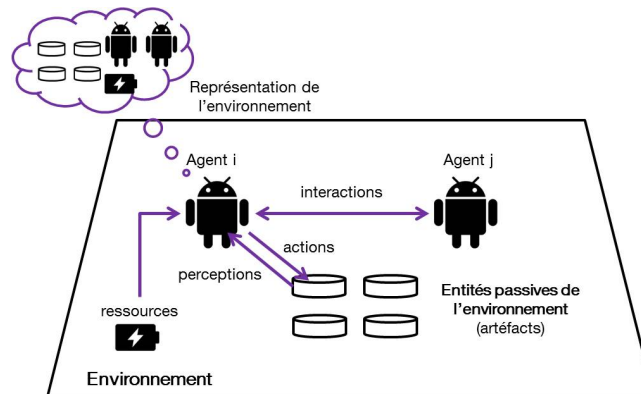


FIGURE 1.2 – Système multi-agents en coopération

Chapitre 2

Implémentation d'un système multi-agents avec la plateforme JADE

2.1 Outils utilisés

2.2 Schéma globale

Notre conception du système multi-agents se base sur trois types d'agents :

Agent central : c'est l'agent qui s'occupe de la gestion des centres de ventes, il reçoit les requêtes des utilisateurs, après traitement il leur retourne les produits qu'ils cherchent.

Agent annexe : représente les vendeurs. Il reçoit une requête de l'agent central et il lui remet les produits présents dans sa base de données qui correspondent à la requête.

Agent enregistreur : c'est l'agent qui garde les informations sur les différents agents centraux et annexes.

2.2.1 Agent annexe

L'agent annexe se charge non seulement de répondre aux requêtes émises par les centres de ventes mais aussi d'inférer des attributs à partir des informations de la requête pour mieux répondre à cette dernière.

Chaque agent annexe a une liste de règles qui permettent à un système expert de savoir si cet agent peut posséder un produit donnée.

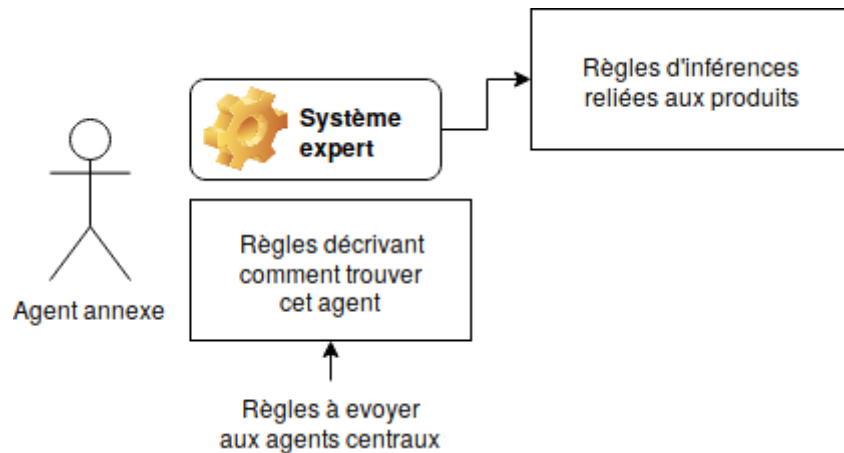


FIGURE 2.1 – Illustration de l'agent annexe

2.2.2 Agent central

L'agent central reçoit d'abord les règles des agents annexes afin qu'il puisse par la suite les contacter après avoir reçu une requête de l'utilisateur.

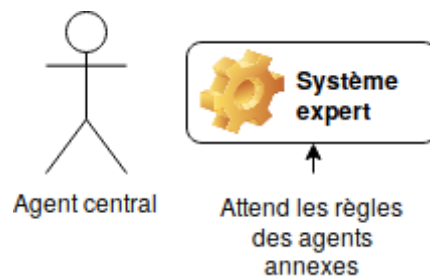


FIGURE 2.2 – Illustration de l'agent central

2.3 Communication entre les agents

La communication entre les agents se divise en deux parties principales :

- Communication d'ajout de service.
- Communication de requête.

2.3.1 Communication d'ajout de service

Ce type de communication est principalement géré par l'agent enregistreur. Tout agent qui arrive dans le système devra informer un agent enregistreur pour qu'il puissent garder ses informations.

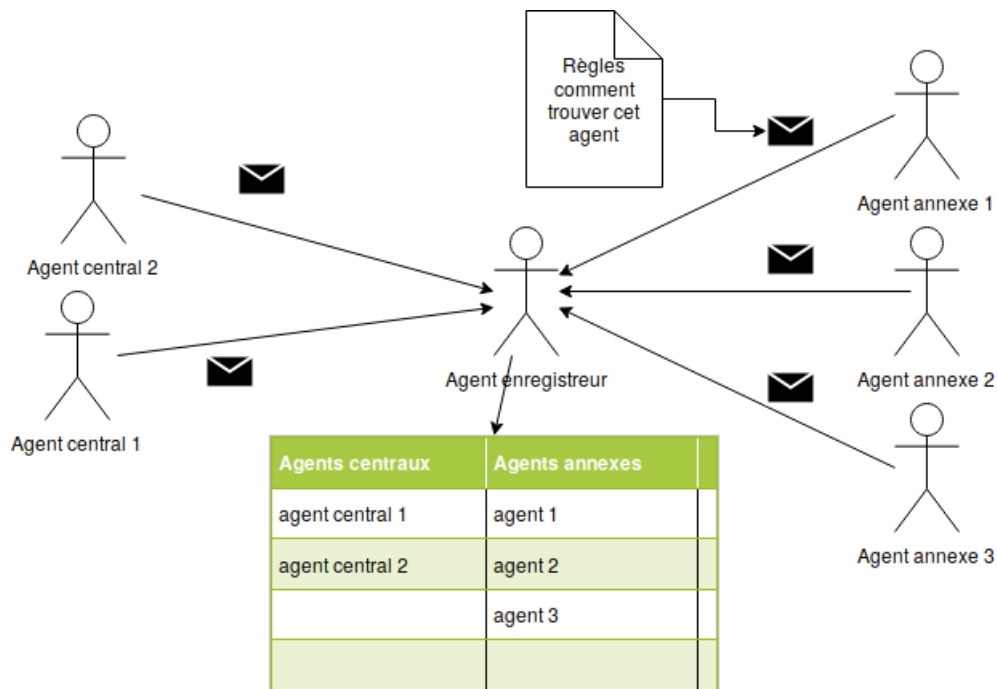


FIGURE 2.3 – Enregistrement des agents

Par la suite, l'agent enregistreur répond aux agents centraux et il leur communique les règles concernant les agents annexes afin qu'ils puissent les trouver lors de l'arrivée d'une requête utilisateur.

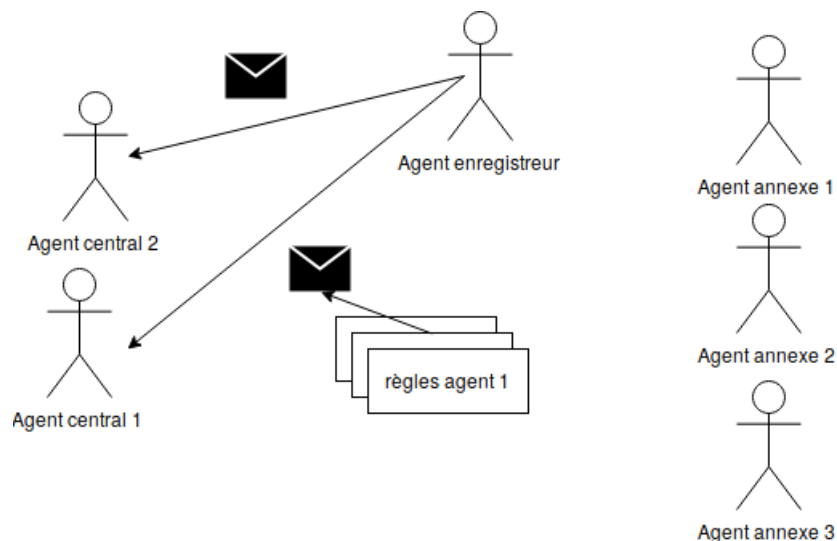


FIGURE 2.4 – Enregistrement des agents

Jusqu'à présent le vrai rôle de l'agent enregistreur n'apparaît pas. C'est quand un nouveau agent qui se connecte dans le système qu'on aperçoit son rôle. Le nouveau agent n'a pas à communiquer avec tous les autres agents pour qu'il soit connu dans son environnement, il suffit d'informer l'agent enregistreur pour réaliser cela. Quand un agent annexe arrive, il envoie ses informations à l'agent enregistreur, ce dernier informe les agents centraux de son arrivée pour qu'ils puissent le contacter.

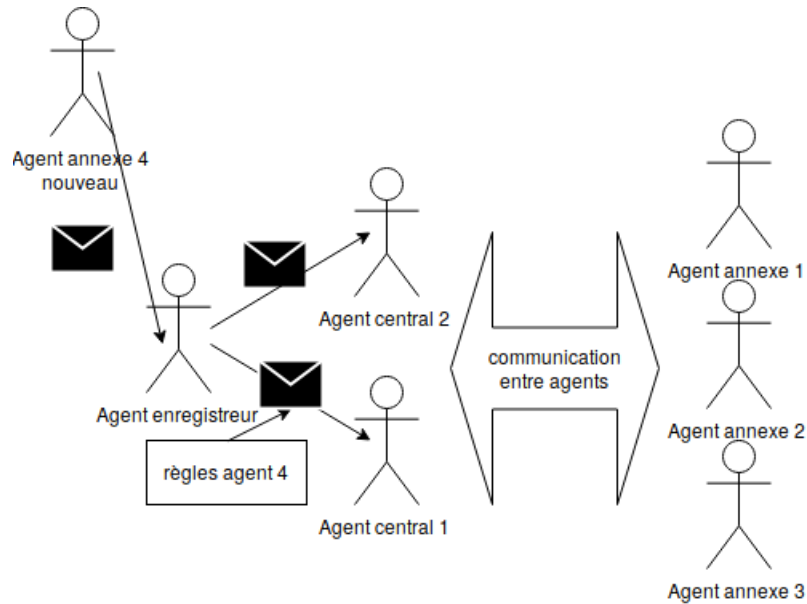


FIGURE 2.5 – L'arrivée d'un nouveau agent annexe

Quand un agent central arrive, l'agent enregistreur l'informe des agents annexes existant, et l'ajoute à la liste des agents centraux.

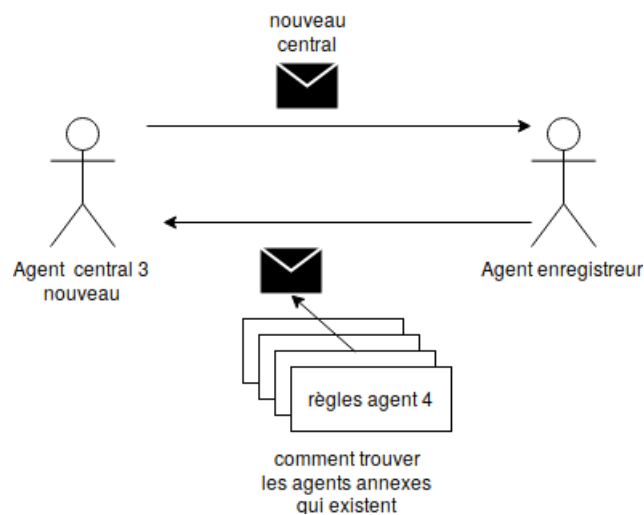


FIGURE 2.6 – L'arrivée d'un nouveau agent central

2.3.2 Communication de requêtes

Ce type de communication concerne la partie des communications qui résulte de l'arrivée d'une requête utilisateur. L'agent central qui reçoit la requête lance son moteur d'inférence pour déduire les agents susceptibles d'avoir les produits spécifiés par la requête. La requête est alors envoyée à ces agents pour qu'ils puissent y répondre.

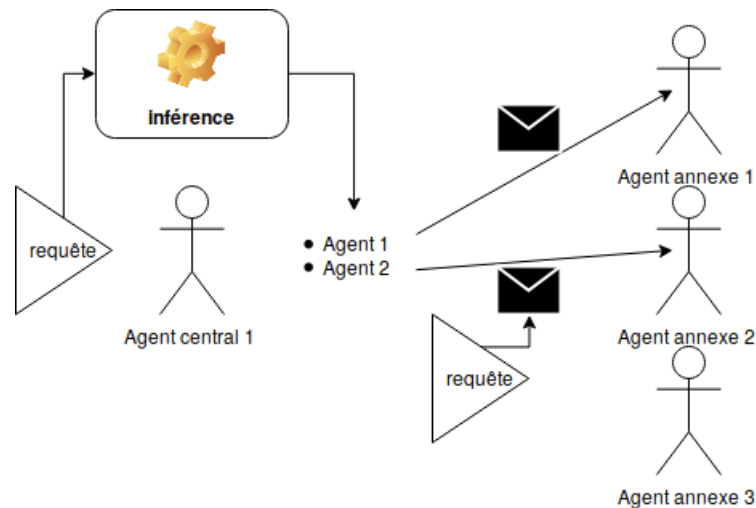


FIGURE 2.7 – La sélection des agents annexes

L'agent annexe qui reçoit la requête commence d'abord par essayer d'inférer de nouvelles connaissances sur le produit que l'utilisateur cherche. Ensuite il cherche dans sa base de données les produits qui correspondent à la requête. Le résultat obtenu est retourné à l'agent central pour qu'il les propose à l'acheteur.

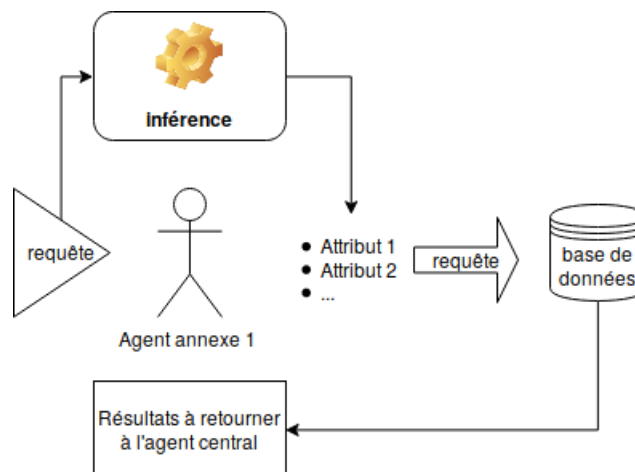


FIGURE 2.8 – Le travail d'un agent annexe

Chapitre 3

Application dédiée

3.1 Interface homme-machine

3.2 Conclusion

Troisième partie

L'interpréteur JASON

Chapitre 1

Introduction

1.1 Problématique et besoins

Durant la partie I, nous avons du concevoir nous même un moteur d'inférence utilisant le langage JAVA, cependant il serait plus intéressant de remplacer ce moteur basique par un moteur plus performant et plus adéquat aux système multi-agent, ce qui nous a amené à l'utilisation de la plateforme JASON.

1.2 Définitions

1.2.1 Architecture CDI (Croyance-Désir-Intention)

Une architecture BDI est conçue en partant du modèle "Croyance-Désir-Intention", en anglais "**Belief-Desire-Intention**", de la rationalité d'un agent intelligent. tel que les **croyances** sont les informations que l'agent possède sur l'environnement et sur d'autres agents qui existent dans le même environnement, les **désirs** représentent les états de l'environnement, et parfois de lui-même, que l'agent aimerait voir réalisés et finalement les **intentions** d'un agent sont les désirs que l'agent a décidé d'accomplir ou les actions qu'il a décidé de faire pour accomplir ses désirs(Même si tous les désirs d'un agent sont consistants, l'agent peut ne pas être capable d'accomplir tous ses désirs à la fois)

1.2.2 AgentSpeakL

AgentSpeak est un langage orienté agent. il est basé sur la programmation logique(Prolog) et les architectures CDI pour les agents cognitives.

```
1 fact (0,1)
2
3 +fact (X,Y)
4   : X < 5
5   <- + fact (X+1, (X+1)*Y) .
6
7 +fact (X,Y)
8   : X == 5
9   <- .print ("fact 5 == ", Y) .
```

FIGURE 1.1 – Exemple d'un agent voulant calculer le factoriel d'un nombre

1.2.3 La plateforme JASON

JASON est une plateforme qui a pour but de faciliter le développement de système multi-agent en offrant un environnement de travail complet comportant un éditeur de texte, un débogger et un compilateur AgentSpeak.

Chapitre 2

Exemples de communication entre agents avec JASON

2.1 Environnement de travail

L'IDE de la plateforme JASON est jEdit, il se présente comme tout IDE classique avec les fonctionnalités adéquates pour la manipulation des agents et des architectures BDI.

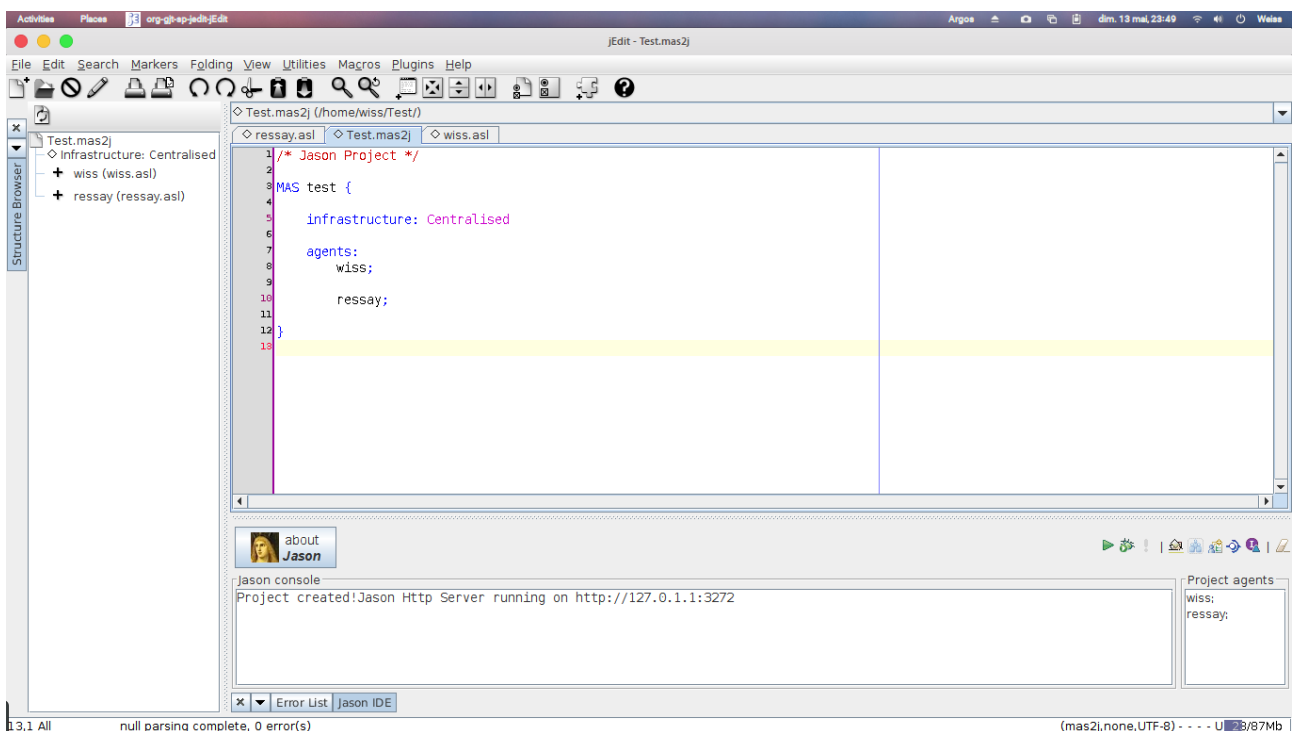


FIGURE 2.1 – Fenêtre d'édition de l'IDE jEdit

2.2 Inférence locale

Un agent dans JASON peut utiliser l'architecture BDI pour lancer son propre moteur d'inférence, prenons l'exemple suivant :

- l'agent a une croyance initiale que la saison est l'été.
- il ajoutera la croyance `article(tshirt)` si la saison est l'été en exécutant le plan : `sayTshirt`.

le code est le suivant :

```
// Agent wiss in project Test.mas2j

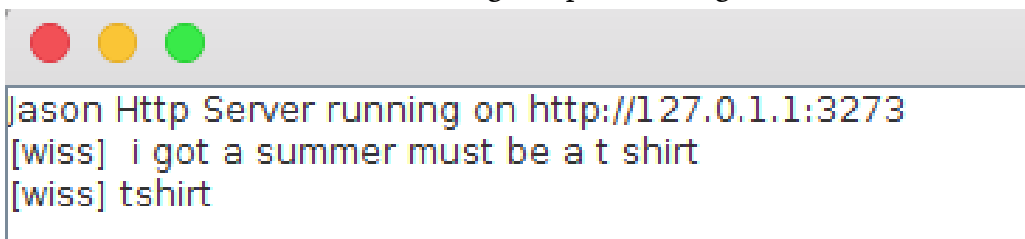
/* Initial beliefs and rules */
season(summer).
/* Initial goals */

!sayShirt.

/* Plans */

+!sayShirt : true <- .print("tshirt").
+article(tshirt) : season(summer) <- .print(" i got a summer must be a t shirt ").
```

FIGURE 2.2 – Code agentSpeak de l'agent wiss



```
Jason Http Server running on http://127.0.1.1:3273
[wiss] i got a summer must be a t shirt
[wiss] tshirt
```

FIGURE 2.3 – résultat

2.3 Scénario simple de communications

Le scenario est le suivant :

- l'agent Ressay envoie un message a l'agent Wiss lui disant que l'article est tshirt.
- l'agent Wiss attendra un message de la part de Ressay seulement car il l'a marqué comme étant fiable.
- Wiss ré-exécute l'inférence du scénario précédent.

```

1 // Agent ressay in project Test.mas2j
2
3 /* Initial beliefs and rules */
4
5 /* Initial goals */
6
7 !start.
8
9 /* Plans */
10
11 +!start : true <- .send(wiss,tell,article([tshirt])).

```

FIGURE 2.4 – Ressay envoie le message

```

1 // Agent wiss in project Test.mas2j
2
3 /* Initial beliefs and rules */
4 sincere(ressay).
5 +article(tshirt)[source(A)] : sincere(A)[source(self)] <- !sayShirt.
6 /* Initial goals */
7
8 /* Plans */
9
10 +!sayShirt : article(tshirt) <- .print("tshirt").

```

FIGURE 2.5 – Wiss reçoit

```

Jason Http Server running on http://127.0.1.1:3274
[wiss] tshirt

```

FIGURE 2.6 – Résultat

Chapitre 3

Comparaison avec la plateforme JADE

3.1 Principales différence

JADE et JASON sont très différents dans leur conceptualisation, l'un utilise le langage java comme interprète des actions entre agent et l'autre un langage d'abstraction plus adapté à ces dites communications. Leurs domaine d'application sont assez différents, il JADE sera préféré en cas de communication simple entre agents sans qu'il y ait besoin de réaliser un traitement d'inférence localement, JASON quant à lui se démarque par le fait qu'il peut lui aussi assurer une communication entre plusieurs agents et offre en plus un environnement adéquat pour l'inférence grâce à l'utilisation du langage AgentSpeak.

Table des figures

1.1	3
1.1	Agent intelligent interagissant avec le monde extérieur	9
1.2	Système multi-agents en coopération	10
2.1	Illustration de l'agent annexe	12
2.2	Illustration de l'agent central	12
2.3	Enregistrement des agents	13
2.4	Enregistrement des agents	13
2.5	L'arrivée d'un nouveau agent annexe	14
2.6	L'arrivée d'un nouveau agent central	14
2.7	La sélection des agents annexes	15
2.8	Le travail d'un agent annexe	15
1.1	Exemple d'un agent voulant calculer le factoriel d'un nombre	19
2.1	Fenêtre d'édition de l'IDE jEdit	20
2.2	Code agentSpeak de l'agent wiss	21
2.3	résultat	21
2.4	Ressay envoie le message	22
2.5	Wiss reçoit	22
2.6	Résultat	22