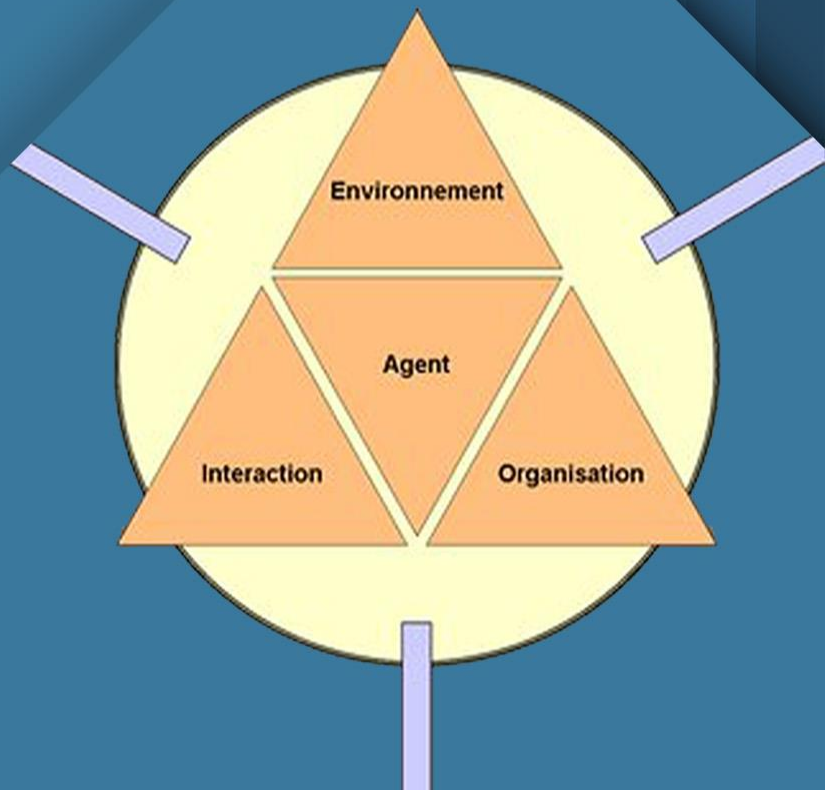
Intelligence artificielle distribuée et agents intelligents



Sommaire

1

Introduction aux Systèmes Multi-Agents (SMA)

Introduction aux Systèmes Multi-Agents (SMA)

Source et Inspiration

Les Systèmes Multi-Agents (SMA) tirent leur origine de la volonté d'imiter les systèmes complexes que nous observons dans la nature et dans les sociétés humaines. L'inspiration vient des colonies d'insectes, des écosystèmes, et des organisations sociales où, malgré l'absence d'un contrôle central, les individus interagissent de manière autonome pour atteindre des objectifs communs. Ces interactions mènent à des phénomènes d'auto-organisation et d'intelligence émergente, où le tout est plus que la somme de ses parties.

Rappels sur les Systèmes Experts et l'IA

Avant de plonger dans le monde des SMA, rappelons brièvement ce que sont les systèmes experts et l'intelligence artificielle (IA). Les systèmes experts sont des programmes informatiques conçus pour résoudre des problèmes complexes en imitant le raisonnement d'un expert humain dans un domaine spécifique. L'IA, quant à elle, cherche à créer des machines capables de réfléchir, d'apprendre, et d'agir de manière autonome. Les SMA représentent une branche de l'IA axée sur la création d'entités autonomes, ou agents, capables de coopérer, de négocier, et d'interagir pour résoudre des problèmes.

2 Fondements des Systèmes Multi-Agents

Un agent

On appelle agent un système mécanique, biologique ou logiciel qui interagit avec son environnement . Anne Nicole.

Un système multi-agents

Un Système Multi-Agents (SMA) comporte plusieurs agents qui interagissent entre eux dans un environnement commun. Certains de ces agents peuvent être des personnes ou leurs représentants (avatars), ou même des machines mécaniques. S’il y a moins de trois agents, on parle plutôt d ’interaction homme/machine, ou machine/machine que de systèmes multi-agents. Anne Nicole.

Les types d'Agents

Agents simples

Exécutent des tâches de base sans capacités d'apprentissage ou de raisonnement avancé.

Agents cognitifs

Capables de raisonnement et d'apprentissage, permettant une adaptation et une prise de décision complexes.

Agents mobiles

Peuvent se déplacer à travers différents environnements ou systèmes informatiques.

Agents coopératifs

Travaillent ensemble pour atteindre un objectif commun, partageant informations et ressources.

Architectures d'un Agent

Les agents peuvent être conçus selon différentes architectures, déterminant leur manière de fonctionner et d'interagir :

Architecture réactive

Les agents réagissent directement aux stimuli de l'environnement sans représentation interne ou état passé (exemple : l'architecture subsumption).

Architecture Deliberative/Cognitive

Les agents possèdent une représentation interne du monde et utilisent le raisonnement pour prendre des décisions (exemple : les systèmes basés sur les buts ou les plans).

Architecture Hybride

Combine les approches réactives et cognitives pour tirer parti des avantages de chacune.

Fonctionnement d'un Agent dans un SMA

Perception

L'agent perçoit son environnement à travers des capteurs, recevant des informations sur l'état actuel et les changements.

Décision

En fonction de son architecture et de ses objectifs, l'agent décide de l'action à entreprendre.

Action

L'agent agit sur l'environnement via des effecteurs, influençant ainsi l'état du système.

Communication

Les agents échangent des informations pour coordonner leurs actions, partager des connaissances, ou négocier des décisions.

Interactions entre Agents

Les interactions sont au cœur des SMA, permettant la collaboration, la négociation, et la compétition entre agents :

Protocoles de Communication

Définissent les règles d'échange d'informations, souvent basés sur des standards comme FIPA ACL.

Mécanismes de Coordination

Assurent que les actions des agents sont alignées vers les objectifs communs, réduisant les conflits et optimisant l'efficacité collective.

Stratégies de Négociation

Permettent aux agents de parvenir à des accords mutuellement bénéfiques, souvent nécessaires lors de la répartition des ressources ou de la résolution de conflits.

3 JADE (Java Agent Development Framework)

JADE (Java Agent Development Framework)

Introduction à JADE

JADE est un cadre logiciel open source entièrement écrit en Java, conçu pour faciliter le développement de systèmes multi-agents robustes et scalables. Il fournit un ensemble d'outils et une runtime qui respectent les standards FIPA (Foundation for Intelligent Physical Agents) pour l'inter-opérabilité des systèmes multi-agents.

Caractéristiques principales de JADE

Plateforme Indépendante

En étant basé sur Java, JADE peut être exécuté sur n'importe quelle plateforme supportant Java, offrant une grande flexibilité dans le déploiement des applications.

Conformité aux Standards FIPA

JADE implémente les spécifications FIPA pour les agents et les communications entre agents, assurant ainsi la compatibilité et l'inter-opérabilité avec d'autres systèmes conformes.

Architecture Distribuée

Permet de créer des systèmes répartis sur plusieurs machines, facilitant la scalabilité et la robustesse.

Facilité de Développement

Offre une API simple pour développer des agents et gérer leurs comportements, communications, et interactions.

Fonctionnement d'un Agent avec JADE

Création d'un Agent

En étant basé sur Java, JADE peut être exécuté sur n'importe quelle plateforme supportant Java, offrant une grande flexibilité dans le déploiement des applications.

Définition des Comportements

Utiliser différentes classes de comportement fournies par JADE (comme OneShotBehaviour, CyclicBehaviour, TickerBehaviour) pour spécifier les actions de l'agent.

Communication entre Agents

Utiliser les ACL Messages (Agent Communication Language) de JADE pour permettre aux agents d'échanger des informations et des requêtes.

Exécution et Gestion

Lancer la plateforme JADE et déployer les agents sur celle-ci. Les agents peuvent être surveillés et gérés via le GUI (Graphical User Interface) fourni par JADE.

Exemple Pratique

1. Le lancement de la plateforme jade

```
try {
      Runtime runtime = Runtime.instance();
      Properties properties = new ExtendedProperties();
      properties.setProperty(Profile.GUI, "true");
      ProfileImpl profileImpl = new ProfileImpl(properties);
      AgentContainer mainContainer =
      runtime.createMainContainer(profileImpl);
      mainContainer.start();
} catch (Exception e) {
      e.printStackTrace();
}
```

Exemple Pratique

2. La création d'une class d'Agents

```
import jade.core.Agent;


public class MyFerstAgent extends Agent {
	@Override
	protected void setup() {
		System.out.println("Hello world !!");
		System.out.println("I am ready.....");
	}
}
```

Exemple Pratique

3. Le lancement d'un agent

```
Runtime runtime = Runtime.instance();
ProfileImpl profile = new ProfileImpl(false);
profile.setParameter(Profile.CONTAINER_NAME,
"NewContainer");
AgentContainer agentContainer =
runtime.createAgentContainer(profile);
agentContainer.start();
AgentController agentController =
agentContainer.createNewAgent
("FerstAgent","Agents.MyFerstAgent", new Object[]{});
agentController.start();
```