République Algérienne Démocratique et Populaire Ministère de
l'Enseignement Supérieur et de la Recherche

Université A/Mira de Bejaia Faculté des Sciences Exactes
Département d'Informatique

جامعة بجاية
Tasdawit n Bgayet
Université de Béjaïa

Technologie des Agents

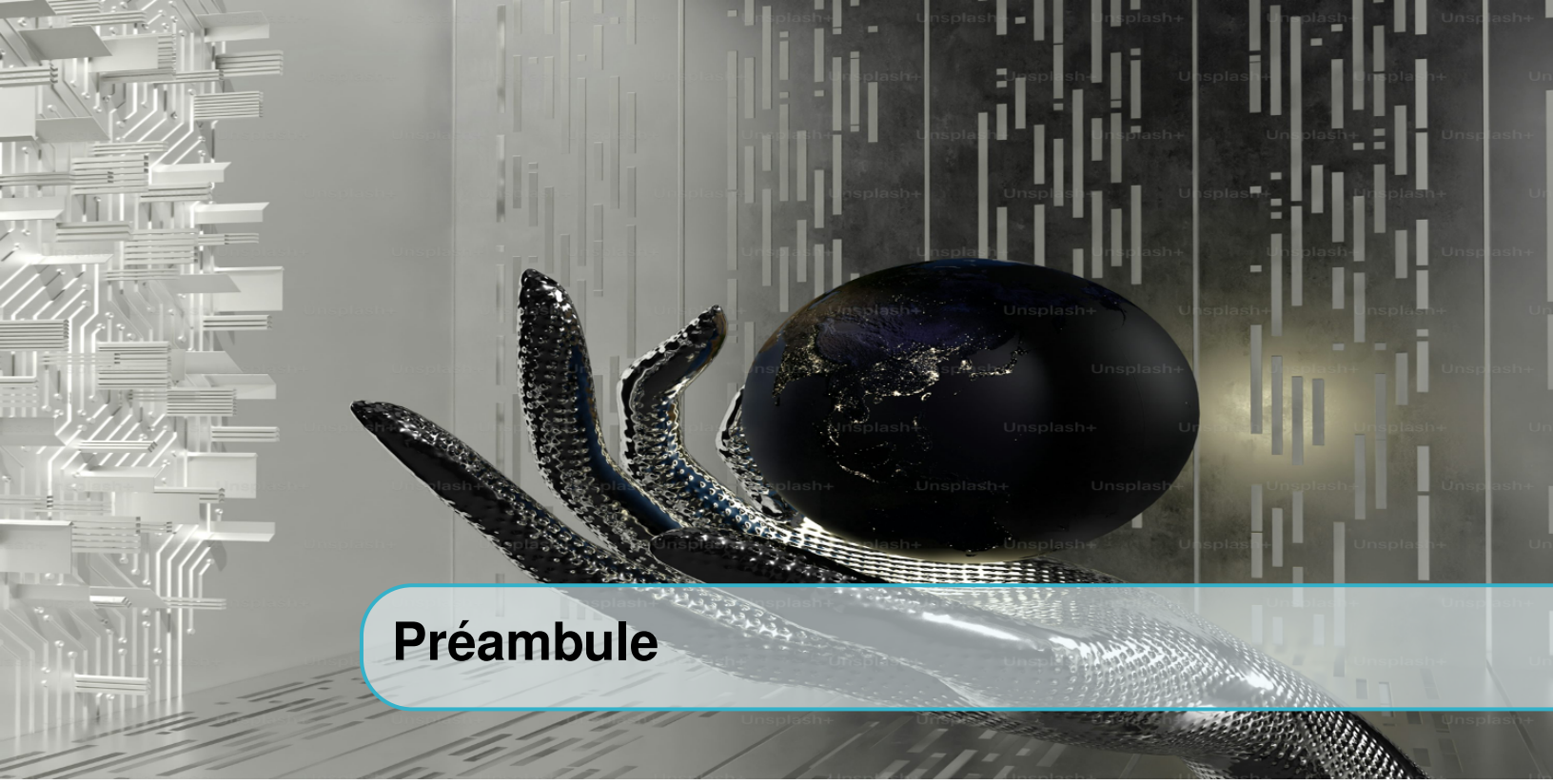
Presenté par :

Djamila BOUKREDERA-BOULAHROUZ

Cours destinéaux étudiants en Master en informatique option: Intelligence Artificielle

2024-2025

Table des Matières



Préambule

Avec le développement de l'intelligence artificielle (IA) et des technologies numériques, les systèmes informatiques d'aujourd'hui sont de plus en plus autonomes, distribués et interactifs. Leur design s'appuie désormais sur des architectures capables de percevoir, de raisonner et d'agir dans des environnements complexes et dynamiques. Ces changements posent de nouveaux défis pour la conception logicielle : comment concevoir des agents intelligents qui peuvent communiquer, coopérer, négocier et s'adapter ? Comment simuler le comportement collectif résultant d'entités autonomes ?

Pour répondre à ces questions, il faut faire appel à des approches de conception basées sur la modélisation et la simulation du comportement intelligent. Les méthodes traditionnelles de programmation séquentielle sont dépassées par la complexité des systèmes distribués et la demande d'incorporation de processus de décision autonomes. C'est dans ce cadre que s'inscrit la technologie des agents, qui est aujourd'hui l'un des paradigmes les plus prometteurs pour la construction de systèmes intelligents et adaptatifs.

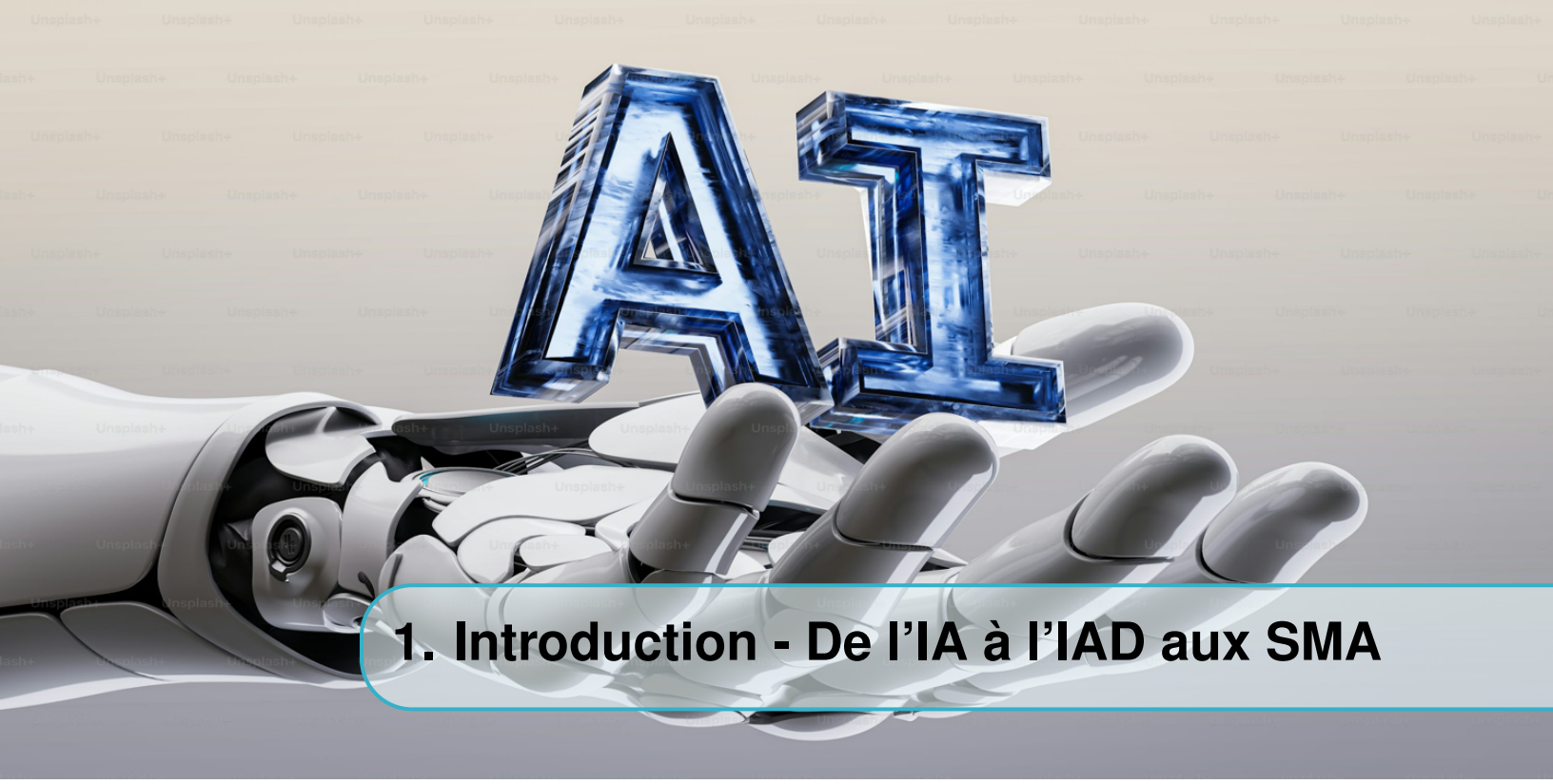
Ce cours vise à introduire les concepts de base des agents et des systèmes multi-agents, leurs mécanismes de communication, de coordination et de négociation. Il insiste sur la modélisation conceptuelle, les architectures cognitives et réactives, les protocoles d'interaction et les mécanismes de négociation automatique entre agents. Ce support s'adresse en premier lieu aux étudiants de Master 2 Informatique – Spécialité Intelligence Artificielle, mais il peut également servir de référence pour d'autres formations de niveau avancé en systèmes distribués, robotique, simulation ou ingénierie logicielle. Il s'adresse également à tout lecteur souhaitant appréhender les bases conceptuelles et méthodologiques de la technologie des agents et des systèmes multi-agents.

Ce document est structuré autour de cinq chapitres complémentaires :

- **Chapitre 1 — Introduction : de l'IA à l'IAD**
  Ce chapitre introduit l'évolution de l'intelligence artificielle vers l'intelligence artificielle distribuée (IAD). Il présente les motivations, les limites des approches centralisées et les principes de la décentralisation des décisions.

- **Chapitre 2 — Concepts fondamentaux des agents** Il définit la notion d'agent, ses caractéristiques, ses types (réactif, cognitif, hybride) et ses principales propriétés (autonomie, réactivité, proactivité, sociabilité).
- **Chapitre 3 — Modèles et architectures des agents**
  Ce chapitre détaille les différentes architectures d'agents (réactives, délibératives, hybrides, BDI, mobiles) ainsi qu'une comparaison entre les differentes architectures.
- **Chapitre 4 — Les systèmes multi-agents**
  Il aborde les fondements de la coopération et de la coordination, les protocoles d'interaction, la communication entre agents, ainsi que les comportements émergents dans les SMA.
- **Chapitre 5 — Négociation automatique**
  Ce dernier chapitre traite des stratégies de négociation, de la théorie des jeux, des enchères et des protocoles argumentatifs. Il illustre ces notions à travers une étude de cas concrète de négociation dans un SMA.

En conclusion, ce support de cours vise à fournir à l'étudiant une compréhension approfondie des principes théoriques et pratiques de la technologie des agents, ainsi qu'une méthodologie de modélisation et de conception de systèmes multi-agents capables d'autonomie, de coopération et de négociation dans des environnements dynamiques et complexes.



1. Introduction - De l'IA à l'IAD aux SMA

1.1 Introduction

L'intelligence artificielle (IA) est une discipline de l'informatique qui vise à reproduire, simuler ou étendre certaines capacités cognitives humaines à travers des machines.
L'intelligence artificielle a évolué au fil des décennies pour passer de systèmes isolés, prenant des décisions de manière autonome, à des systèmes plus collaboratifs et distribués [1, 2]. En effet, l'intelligence artificielle a d'abord émergé dans les années 50 avec une approche centralisée et symbolique, focalisée sur la résolution de problèmes logiques par des systèmes monolithiques tels que les systèmes experts. Cependant, cette vision a rapidement rencontré des limites face à la complexité du monde réel et l'émergence de problématiques distribuées qui nécessitent plus de flexibilité et d'adaptation. L'Intelligence Artificielle Distribuée (IAD) et les Systèmes Multi-Agents (SMA) sont alors apparus comme une solution pertinente pour la résolution de problèmes complexes dans des environnements dynamiques.

Ce chapitre propose une introduction approfondie à l'IAD et aux SMA qui représentent des domaines cruciaux de l'informatique moderne. Nous explorerons leur évolution, leurs concepts fondamentaux, leurs applications et les défis qu'ils visent à résoudre.

1.2 Influence Historique : Du Langage Machine à la Programmation Orientée Agents

L'évolution des paradigmes de programmation a jeté les bases des approches actuelles en IAD et SMA. Cette section retrace brièvement ce parcours historique, illustré par la figure 1.1.

1.2.1 Historique des Méthodes de Programmation

L'évolution des méthodes de programmation a été marqué par une recherche constante d'abstraction et de modularité pour gérer la complexité croissante des systèmes.

- **Langage machine et assembleur** : Les premiers ordinateurs étaient programmés directement en langage machine, un ensemble d'instructions binaires directement interprétées par l'Unité Centrale de Traitement (CPU). Ce niveau de programmation est extrêmement détaillé et étroitement lié aux archi-

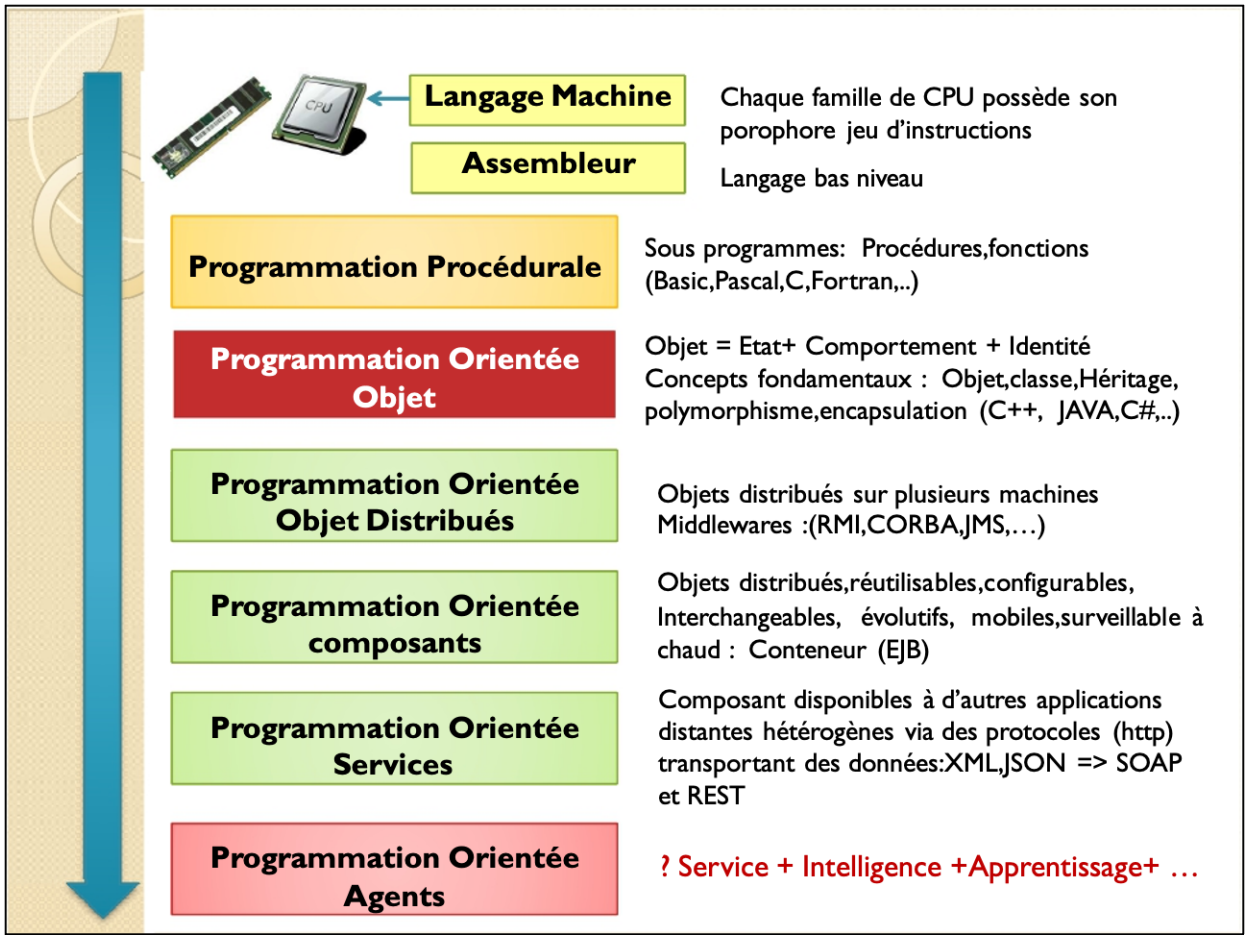


Figure 1.1: Evolution des méthodes de programmation.

tectures matérielles. Cette approche bas niveau limitait considérablement la capacité de développement de systèmes complexes.

- **Programmation procédurale** : L'émergence de la programmation procédurale avec des langages comme BASIC, Pascal, C et FORTRAN a marqué un premier tournant. Ce paradigme met l'accent sur l'organisation du code en sous-programmes (procédures et fonctions), favorisant la réutilisabilité et la structuration des tâches. Cependant, cette méthode montrait ses limites face à la complexité croissante des applications.
- **Programmation orientée objet (POO)** : La programmation orientée objet (POO) a révolutionné le développement logiciel en introduisant des concepts fondamentaux : l'objet comme entité combinant état, comportement et identité, ainsi que les principes d'héritage, de polymorphisme et d'encapsulation. Des langages comme C++, Java et C# ont permis de modéliser plus fidèlement le monde réel et de gérer des systèmes de grande envergure.
- **Programmation orientée objet distribuée (POOD)** : Ce paradigme a répondu aux besoins croissants de systèmes répartis sur plusieurs machines. Des middlewares comme RMI (Remote Method Invocation), CORBA (Common Object Request Broker Architecture) et JMS (Java Message Service) ont facilité la communication entre objets distants, ouvrant la voie aux architectures distribuées modernes.
- **Programmation orientée composants et services** : Ce paradigme a introduit des objets distribués, réutilisables, configurables et interchangeables. Les conteneurs comme EJB (Enterprise JavaBeans)

ont permis la surveillance et la gestion à chaud de ces composants, améliorant significativement la maintenabilité des systèmes complexes.

- **Programmation orientée agents** : représente l'évolution naturelle de ces paradigmes, intégrant intelligence artificielle, apprentissage automatique et capacités d'adaptation. Cette approche répond aux défis posés par les environnements dynamiques et imprévisibles à l'ère du numérique.

L'évolution vers des niveaux croissants d'abstraction et de distribution a permis la conception de systèmes de plus en plus complexes, capables de collaborer et de coopérer dans des environnements dynamiques. Cette évolution a conduit à l'émergence des concepts d'Intelligence Artificielle Distribuée et de Systèmes Multi-Agents.

1.2.2 Les Fondements de l'Intelligence Artificielle Classique

L'Intelligence Artificielle (IA) constitue la science et l'ingénierie de la construction de machines intelligentes capables d'adopter des comportements calculables et intelligents, similaires à ceux des humains [1]. Selon le célèbre test de Turing, proposé par Alan Turing en 1950, une machine peut être considérée comme intelligente si elle parvient à se faire passer pour un humain auprès d'un interlocuteur humain lors d'une conversation (voir figure1.2). L'objectif fondamental de l'IA consiste à doter les machines de capacités habituellement

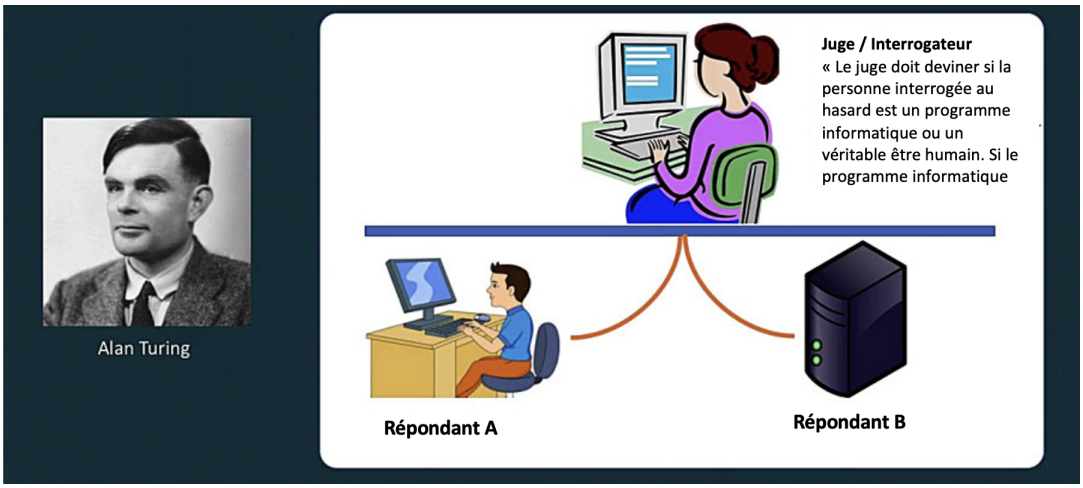


Figure 1.2: Test de Turing.

attribuées à l'intelligence humaine. Ces capacités incluent des tâches aussi diverses que jouer aux échecs à un niveau expert, communiquer dans un langage naturel, traduire des textes entre différentes langues, coordonner des mouvements complexes comme la conduite d'un véhicule, effectuer des tâches domestiques sophistiquées, ou encore reconnaître et identifier des personnes dans des images.

1.2.3 Une Brève Histoire de l'Intelligence Artificielle

L'IA a évolué à travers plusieurs étapes dont les plus importantes sont [3]:

- **Les débuts de l'IA (1943–1955) :** A cette époque, le terme **"IA"** n'existait pas encore. Des chercheurs comme McCulloch, Pitts ou Turing posent les premières bases [4] : modèles de neurones, apprentissage synaptique, et le test de Turing.
- **Naissance d'IA (1956)** : L'IA devient une discipline à part entière lors de la célèbre conférence de *Dartmouth*, où le terme *"IA"* est formellement proposé et adopté.
- **Age d'or et espoirs grandissants (1952-1969)** : Ce fut une période très active pour le jeune domaine de l'IA. Plusieurs programmes furent développés pour résoudre des problèmes d'une grande diversité

(résolution de théorèmes, jeux, blocs...). Ce fut aussi l'époque du Shakey, le premier robot à être capable de raisonner sur ses propres actions. les chercheurs croyaient alors que l'intelligence humaine serait rapidement simulée.

- **Premières désillusions (1966-1973)**: Les limites des approches centralisées apparaissent. Les traductions automatiques échouent, les algorithmes peinent à s'adapter aux situations complexes, et les ressources matérielles sont insuffisantes. Plusieurs projets sont alors abandonnés.
- **L'ère des systèmes experts (1969–1979)**: Les premiers systèmes experts axés sur des tâches spécifiques, comme DENDRAL et MYCIN, furent créés. ils ont montré qu'un raisonnement basé sur des règles peut égaler, voire dépasser, les performances humaines dans des domaines ciblés.
- **Le retour des réseaux de neurones (1986-présent)** : Grâce à l'algorithme de rétropropagation, les réseaux peuvent apprendre des fonctions complexes. L'apprentissage automatique devient un pilier majeur de l'IA.
- **Vers une IA moderne (Depuis les années 1990)** : L'IA devient une science rigoureuse, fondée sur des modèles mathématiques et des résultats expérimentaux, et appliquée à des problèmes concrets (santé, finance, industrie, etc.).

1.2.4 Limites de l'Approche Centralisée Traditionnelle

Les limites de l'IA traditionnelle sont principalement dues à sa conception centralisée classique avec une entité unique traitant toutes les données, reposant sur des machines séquentielles mono-processeur. L'avènement des systèmes experts dans les années 80 a marqué une étape importante de l'évolution de l'IA. Ils se sont révélés utiles mais uniquement dans des contextes très spécifiques. Leur approche rigide et leur dépendance à des bases de connaissances statiques les rendent peu adaptables aux environnements dynamiques et imprévisibles du monde réel.
Les problèmes du monde réel, comme la gestion du trafic ou la coordination de robots, ne peuvent pas être efficacement résolus par un seul système centralisé en raison de :

- La complexité computationnelle : trop d'informations à traiter
- L'imprévisibilité : environnements dynamiques
- Le besoin de collaboration entre plusieurs entités

De là découlent plusieurs autres limites critiques de l'IA classique notamment:

- Robustesse : Vulnérabilité aux informations incomplètes ou bruitées.
- Scalabilité : Explosion combinatoire dans les environnements complexes (frame problem).
- Grounding : Difficulté à lier les symboles à la réalité perceptuelle (problème de l'ancrage).
- Centralisation : Inadaptation aux problèmes distribués (e.g., robotique en essaim, marchés électroniques).

■ Exemple 1.1 Gestion du trafic routier

- **IA centralisée**: Une seule entité contrôle tous les feux de circulation, mais elle ne peut pas réagir efficacement aux embouteillages en temps réel, car l'entité centrale ne peut pas traiter instantanément toutes les informations locales et adapter sa réponse en conséquence.
- **IA distribuée**: Chaque feu de circulation est un agent qui ajuste son comportement en fonction des voitures détectées et des feux voisins. Cette approche permet une adaptation locale rapide tout en maintenant une coordination globale, résultant en une gestion du trafic plus fluide et plus réactive.

Afin de mieux illustrer les limites de l'IA classique, nous présentons dans ce qui suit comment le projet ***Shakey le robot*** a échoué dans les années 1960.

◆ ***Illustration*** : ***Shakey le robot*** est le premier robot générique capable de raisonner sur ses actions (voir figure 1.3). Il a été créé à la fin des années 1960 en Californie par SRI International. Ce robot autonome, l'un des premiers à combiner perception, raisonnement symbolique et action, devait naviguer dans un environnement structuré (pièces, murs, objets) en utilisant un système de planification logique. Son objectif est d'exécuter des tâches simples du type *"pousser la boîte A vers la rampe B"*. Il était basé sur une approche centralisée et un modèle symbolique global de l'environnement. Il génère un plan séquentiel optimal qu'il exécute avec rigidité.

- ***Échecs et limites de Shakey:*** Shakey souffrait de plusieurs faiblesses caractéristiques de l'IA classique :
  - **Rigidité du planificateur**: au moindre imprévu dans l'environnement (un objet mal placé, un obstacle imprévu), le robot devait replanifier entièrement sa séquence d'actions, souvent avec un échec à la clé ;
  - **Centralisation excessive** : toutes les décisions étaient prises par un contrôleur central unique, ce qui entraînait une réactivité très lente;
  - **Pas de modularité ni de spécialisation** : toutes les fonctions étaient intégrées dans une seule entité monolithique, ce qui engendre un manque de robustesse;
  - **Modélisation Symbolique Fragile**: C'est le problème de Grounding où les Symboles sont différents de la réalité physique. Par exemple, une ombre projetée sur une boîte est Interprétée comme "obstacle" ce qui provoque le blocage du plan.

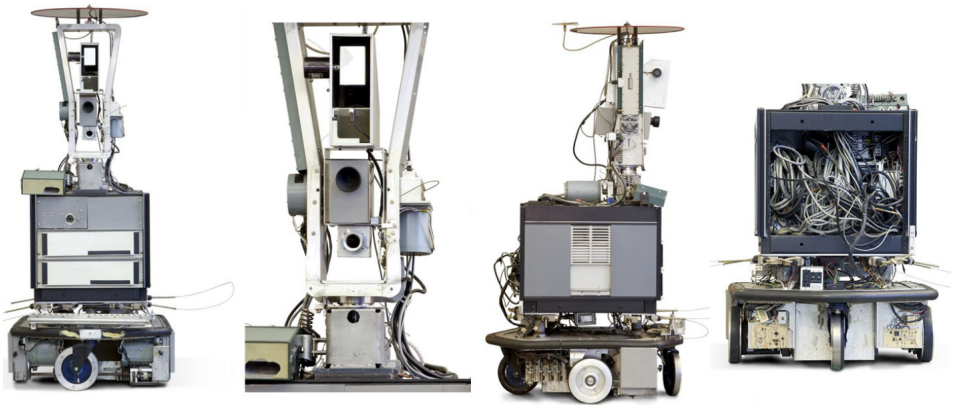


Figure 1.3: Robot Shakey (1966-1972).

Ces limitations ont montré qu'une IA strictement centralisée dans des environnements plus dynamiques ou incertains, devenait rapidement inefficace. Ces constats ont ouvert la voie à une nouvelle vision : une intelligence décentralisée, réactive, adaptative, incarnée par les systèmes multi-agents, où plusieurs entités coopèrent pour résoudre collectivement des tâches complexes.

1.3 L'Intelligence Artificielle Distribuée : Paradigme et Principes

L'Intelligence Artificielle Distribuée (IAD) émerge comme une réponse naturelle aux limitations de l'IA centralisée [5]. L'IAD s'intéresse spécifiquement aux domaines pour lesquels sont inappropriés un seul résolveur de problème, une seule machine, ou un seul lieu de traitement. L'IAD se concentre sur la résolution de problèmes par la collaboration de plusieurs entités intelligentes autonomes.
Cette transformation conceptuelle peut être illustrée par un changement de métaphore fondamental : alors que l'IA traditionnelle repose sur la métaphore du "penseur isolé" - un système unique et omniscient capable de résoudre tous les problèmes - l'IAD adopte la métaphore de la "communauté de penseurs" - un ensemble

d'entités spécialisées qui collaborent et interagissent pour résoudre des problèmes complexes, comme le montre la figure 1.4.

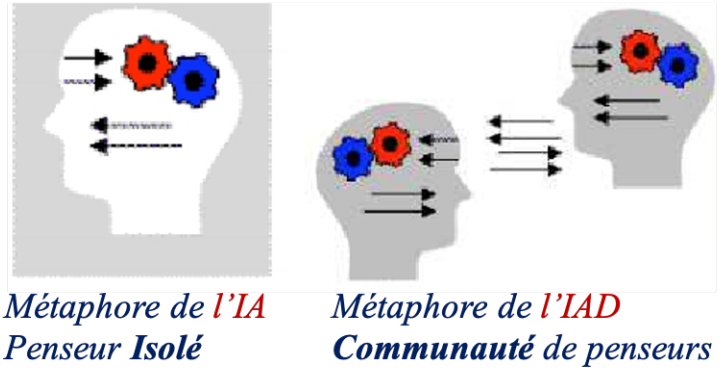


Figure 1.4: IAD versus IA.

1.3.1 Principes Fondamentaux de l'IAD

L'IAD est née au début des années 80 [2]. Elle propose une distribution de l'expertise sur un ensemble de systèmes capables d'interagir en coopération dans un environnement commun. Ces systèmes doivent être en mesure de résoudre les éventuels conflits et de coordonner leurs efforts pour mener à bien des tâches complexes, qu'il s'agisse de résolution de problèmes, d'aide à la décision, de reconnaissance de formes, ou de conduite de processus industriels.
L'essence de l'IAD réside dans l'étude de systèmes où des agents artificiels opèrent collectivement et de façon décentralisée pour accomplir une tâche commune. Cette approche combine deux dimensions complémentaires : la modélisation du savoir des agents (compétence individuelle) et la modélisation de leurs interactions (organisation sociale).
Il convient de noter que plusieurs facteurs motivent l'adoption d'une approche distribuée en intelligence artificielle. La plupart des applications et problèmes réels impliquent des systèmes qui sont naturellement physiquement et fonctionnellement distribués. Plutôt que de forcer une centralisation artificielle, l'IAD respecte et exploite cette distribution naturelle. Cette approche offre plusieurs avantages significatifs par rapport aux méthodes centralisées, notamment :

- **Décentralisation** : Il n'y a pas d'autorité centrale unique. Les agents sont autonomes et collaborent entre eux pour atteindre un objectif commun, ce qui réduit considérablement les points de défaillance uniques.
- **Robustesse et Tolérance aux Pannes** : grâce à la redondance et à la répartition des risques, le système est plus robuste face aux défaillances individuelles.
- **adaptabilité et flexibilité** : Les agents peuvent s'adapter à des environnements changeants.
- **Évolutivité** : Le système est généralement facile à étendre en ajoutant de nouveaux agents ou entités sans nécessiter une refonte complète.
- **Parallélisme et Efficacité** : La distribution des tâches permet un traitement parallèle améliorant considérablement les performances globales du système.
- **Modularité** : Réutilisation d'agents spécialisés.

Cependant, l'IAD présente également des défis :

- **Complexité de la coordination**: Si le système compte un grand nombre d'entités autonomes alors la gestion des interactions et de la coordination entre eux peut être très complexe.

- **Communication**: L'IAD nécessitent des mécanismes de communication efficaces et robustes pour éviter les congestions ou la perte d'informations.
- **Évaluation des performances** : La performance globale d'un système IAD peut être difficile à mesurer et à optimiser en raison de sa nature décentralisée.

1.4 Applications Concrètes de l'Intelligence Artificielle Distribuée

L'IAD trouve des applications dans de nombreux domaines :

1.4.1 Réseaux Sociaux et Systèmes Web à Grande Échelle

: Internet et les réseaux sociaux sont un exemple parfait qui illustre la nécessité de l'IAD. Ces systèmes doivent assurer la tâche de gérer des milliards d'utilisateurs en temps réel qui est impossible à réaliser par un serveur unique. C'est grâce à des agents intelligents répartis dans différents centres de données qui analysent continuellement les interactions utilisateurs que du contenu personnalisé en temps réel est recommandé.

■ **Exemple 1.2** Facebook, YouTube, Netflix utilisent des agents intelligents sophistiqués pour suggérer des vidéos, recommander des amis ou personnaliser les flux d'actualités. Ces systèmes apprennent continuellement des comportements des utilisateurs et s'adaptent aux préférences changeantes.

1.4.2 Transports Autonomes et Mobilité Intelligente

Le domaine des transports autonomes représente l'un des défis les plus complexes de l'IAD contemporaine. La coordination de milliers de véhicules autonomes sans créer d'embouteillages nécessite une approche distribuée sophistiquée. Une gestion centralisée s'avérerait inefficace car elle ne pourrait pas réagir instantanément aux changements locaux des conditions de circulation. Par contre, une approche distribuée permet aux voitures de communiquer entre elles (V2V – Vehicle-to-Vehicle) et avec les infrastructures (feux intelligents). Des systèmes comme ***Tesla Autopilot*** et ***Waymo*** illustrent cette approche, où chaque véhicule constitue un agent intelligent capable de prendre des décisions locales tout en participant à une coordination globale.

■ **Exemple 1.3 — Cas extrême : évacuation d'urgence en milieu urbain.** : Dans de tels scénarios, chaque véhicule autonome doit adapter son itinéraire en fonction des informations de trafic transmises par d'autres véhicules, tandis qu'un agent de contrôle central peut fournir une orientation globale sans contraindre les décisions locales.

1.4.3 Robotique Collective et Essaims Intelligents

Face à la question de savoir comment plusieurs robots peuvent coopérer pour accomplir une tâche complexe, l'approche centralisée présenterait un point de défaillance unique inacceptable. Cependant, en distribuant l'intelligence, les robots peuvent s'adapter aux conditions imprévues et maintenir leur fonctionnalité même en cas de défaillance de certains éléments du système.

■ **Exemple 1.4** Les missions d'exploration de Mars par la NASA utilisent cette approche : chaque drone cartographie une zone spécifique et transmet ses données aux autres membres de l'essaim, créant une carte collaborative de l'environnement.
Dans le domaine agricole, des essaims de drones surveillent les cultures et détectent les maladies ou les zones nécessitant une attention particulière. Cette approche distribuée permet une couverture efficace de grandes surfaces tout en maintenant une précision locale élevée.

1.4.4 Finance et Commerce Électronique

Le trading haute fréquence et le commerce électronique moderne exemplifient l'application de l'IAD dans des environnements où les décisions doivent être prises en millisecondes. Un seul agent central ne pourrait analyser simultanément toutes les bourses mondiales et identifier les opportunités d'arbitrage en temps réel. Des agents intelligents distribués dans différents marchés financiers détectent automatiquement les opportunités d'achat et de vente, exécutant des transactions à une vitesse impossible pour des traders humains. Amazon utilise cette approche pour ajuster dynamiquement les prix de millions de produits en fonction de la demande, de la concurrence et de multiples autres facteurs économiques.

■ **Exemple 1.5 — Marché boursier et crash éclair:.** En 2010, un "Flash Crash" a été causé par des milliers d'agents autonomes qui ont pris des décisions en chaîne, amplifiant les fluctuations du marché en quelques secondes.

1.4.5 Systèmes de Santé Intelligents

Dans le domaine de la santé et des systèmes hospitaliers intelligents, le problème est de gérer les ressources hospitalières et les soins aux patients dans plusieurs hôpitaux. Un seul hôpital ne peut pas gérer toutes les urgences d'un région, nécessitant une coordination distribuée entre plusieurs établissements.

Une approche distribuée permet à chaque hôpital, considéré comme un agent intelligent, d'échanger des ressources (ambulances, équipements spécialisés, personnel médical) et de coordonner l'allocation des patients selon la disponibilité des soins et l'urgence des cas.

■ **Exemple 1.6 — Pandémie de COVID-19:.** La pandémie de COVID-19 a particulièrement mis en évidence l'importance de cette approche, avec la répartition automatisée des respirateurs entre différents hôpitaux selon les taux d'occupation, et la coordination des laboratoires pour partager des données et accélérer la recherche médicale.

Outre ces exemples concrets de l'utilisation de l'IAD, il existe de nombreux autres domaines d'application notamment:

- **Systèmes de Contrôle et de Surveillance** : tels que la gestion du trafic aérien et le contrôle de processus industriels.
- **Réseaux Informatiques** : telles que l'optimisation du routage et la gestion des pannes.
- **Logistique et Chaînes d'Approvisionnement** : telles que l'optimisation des livraisons et la gestion des stocks.
- **Robotique Collaborative** : telles que l'exploration d'environnements inconnus et la manipulation d'objets lourds.
- **Internet des Objets (IoT)**.

1.5 Architecture de l'Intelligence Artificielle Distribuée

L'IAD s'articule autour de trois axes principaux qui représentent différentes approches de la distribution de l'intelligence : l'Intelligence Artificielle Parallèle, la Résolution Distribuée de Problèmes, et les Systèmes Multi-Agents.

1.5.1 Intelligence Artificielle Parallèle

L'Intelligence Artificielle Parallèle se concentre sur l'accélération des calculs en divisant une tâche complexe en sous-tâches pouvant être exécutées simultanément. Cette approche s'avère idéale pour les problèmes hautement parallélisables, notamment l'entraînement de modèles de deep learning et les simulations numériques intensives.

Cependant, cette approche ne s'intéresse pas prioritairement à la nature du raisonnement ni à l'intelligence des comportements individuels. Elle vise principalement l'optimisation des performances computationnelles par la parallélisation des traitements.

■ **Exemple 1.7 — Simulation climatique au CERN.** - **Défi**: Simuler 100 ans de changement climatique avec résolution 1km²
- **Resources**: 1000 processeurs sur 50 nœuds de calcul.
- **Décomposition**:
- Chaque processeur simule une région géographique de 100km x 100km
- Échange des conditions limites toutes les heures simulées
- Synchronisation globale toutes les 24h simulées
- **Résultats**:
- Temps de calcul : 48h réelles pour 100 ans simulés.
- Précision : Erreur <2% par rapport à simulation séquentielle.
- Efficacité : 85% (excellent pour ce type de problème).

1.5.2 Résolution Distribuée de Problèmes

La Résolution Distribuée de Problèmes adopte une approche axée sur la décomposition d'un problème complexe en sous-problèmes résolus par des entités collaboratives. Cette méthode s'adapte particulièrement bien aux problèmes comme l'optimisation du trafic, la gestion de réseaux de télécommunications, ou la planification de ressources dans des systèmes complexes.
Dans cette approche, les entités sont généralement interdépendantes, et le processus de résolution suit une logique coordonnée conduisant à un résultat global optimal. La coopération entre entités est généralement prédéfinie et structurée selon les caractéristiques du problème à résoudre.

1.5.3 Systèmes Multi-Agents (SMA)

Les Systèmes Multi-Agents représentent l'approche la plus sophistiquée de l'IAD. Elle est axée sur la coordination d'agents autonomes pour atteindre des objectifs individuels ou collectifs dans un environnement partagé. Cette approche s'avère idéale pour les environnements dynamiques et complexes comme les entrepôts automatisés, les jeux vidéo sophistiqués, ou les systèmes de recommandation adaptatifs.
Les SMA, formalisés dans les années 1990 [6, 7], ont pour objectif de faire coopérer un ensemble d'entités proactives et relativement indépendantes, appelées **"agents"**, dotées d'un comportement intelligent. Cette coopération vise à coordonner leurs objectifs et leurs plans d'actions pour résoudre des problèmes complexes nécessitant une expertise distribuée.
Les SMA trouvent des applications dans de nombreuses disciplines, notamment la simulation de phénomènes complexes, la modélisation de comportements sociaux, l'optimisation de processus industriels, et le développement de systèmes adaptatifs intelligents.
Le schéma de la figure 1.5 résume les trois axes principaux de l'IAD.

■ **Exemple 1.8 — Exemple récapitulatif : Contrôle d'un Carrefour Routier en milieu urbain . Problème**: Optimiser la circulation à un carrefour à 4 voies avec des flux variables de voitures.

- ***Approche IA Classique (Années 1970)*** :
  - **Système**: Feu tricolore à timer fixe
  - **Fonctionnement** : Cycle prédéfini : 60s vert voie A → 60s vert voie B;
    Programmé par un ingénieur après étude du trafic moyen;
  - **Limites** : Bloqué si afflux soudain (ex: match de foot);

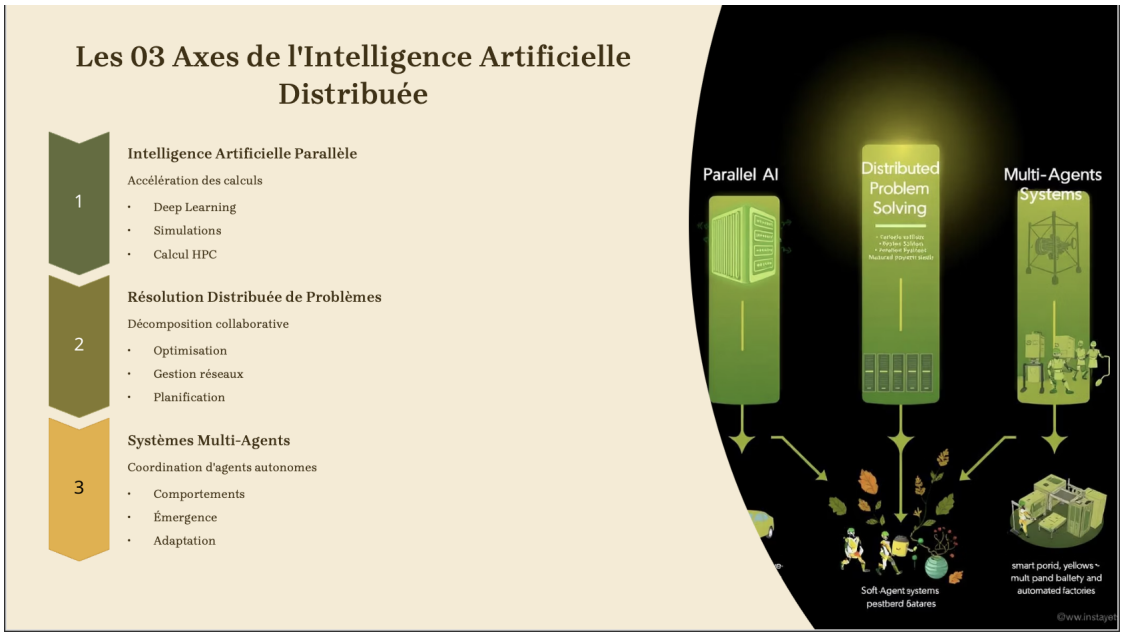


Figure 1.5: Architecture de l'Intelligence Artificielle Distribuée .

Gaspillage si aucune voiture sur une voie;
  - **Constat**: L'intelligence est centralisée dans le programmeur, incapable d'adaptation.
- ***Approche IAD (Années 1990)***
  - **Système**: Feux "intelligents" avec capteurs
  - **Fonctionnement** : - Agent Capteur qui Détecte le nombre de voitures par voie;
    - Agent Contrôleur qui Ajuste la durée du vert selon des règles simples ;
    - Coordination minimale : priorité aux voies saturées;
    - Avantage : Adaptation basique aux variations
  - **Limites** : Ne voit que son carrefour → bouchons en aval;
  - **Constat**: L'intelligence est distribuée mais reste locale et coopérative.
- ***Approche SMA (Années 2020)***
  - **Système**: Réseau de feux communicants
  - **Fonctionnement** : Agent Feu par carrefour, avec objectifs de maximiser son flux tout en minimisant les bouchons voisins (solidarité);
    - Interaction :
    * Négociation entre feux adjacents : "Je te cède 20s de vert si tu absorbes mon surplus"*
    *Apprentissage des flux quotidiens;
    - Émergence : Onde verte auto-organisée sur un axe
  - **Constat**: L'intelligence émerge des interactions, créant une optimisation globale non programmée.

1.6 Conclusion

L'IA classique a jeté les bases symboliques de l'intelligence [4], mais son incapacité à gérer la complexité distribuée a conduit à l'émergence de l'IAD [5]. Il s'agit d'une avancée considérable, car elle permet à des entités intelligentes de travailler ensemble, en répartissant la « pensée » entre différents agents. Cependant,

elle restait limitée par ses interactions protocolaires et l'homogénéité de ces agents. Les SMA ont désormais franchi une étape importante car ils ont transformé l'intelligence d'une entité individuelle en un phénomène collectif, où l'hétérogénéité, le conflit et la dynamique sociale deviennent les moteurs de la complexité créative.

1.7 Exercices

Exercice 1.1 ***Scénario*** : À 8h00, un bus scolaire (prioritaire) approche du carrefour central.

1. Décrire ce qui se passe avec les différentes approches: IA classique, IAD et SMA?
2. Quel paradigme semble le plus adapté aux villes modernes ? Justifiez .

Solution Proposée:

1. voir le tableau 1.1.
2. Les SMA permettent une réponse dynamique aux événements imprévus (ex: bus prioritaire) grâce à la communication décentralisée, contrairement aux autres approches.

Approche	Question	Réponse
IA classique	Que se passe-t-il ?	Le feu reste vert pour la voie programmée, le bus doit attendre son tour.
IAD	Comment l'agent capteur pourrait-il détecter le bus ?	Par une balise RFID ou une caméra avec reconnaissance des véhicules prioritaires.
SMA	Interaction entre agents pour donner la priorité au bus ?	L'agent bus envoie un message : « Priorité demandée au carrefour X ». L'agent feu négocie avec les feux adjacents pour créer un corridor vert.

Table 1.1: Solution proposée de la question 1 de l'exercice 1.1

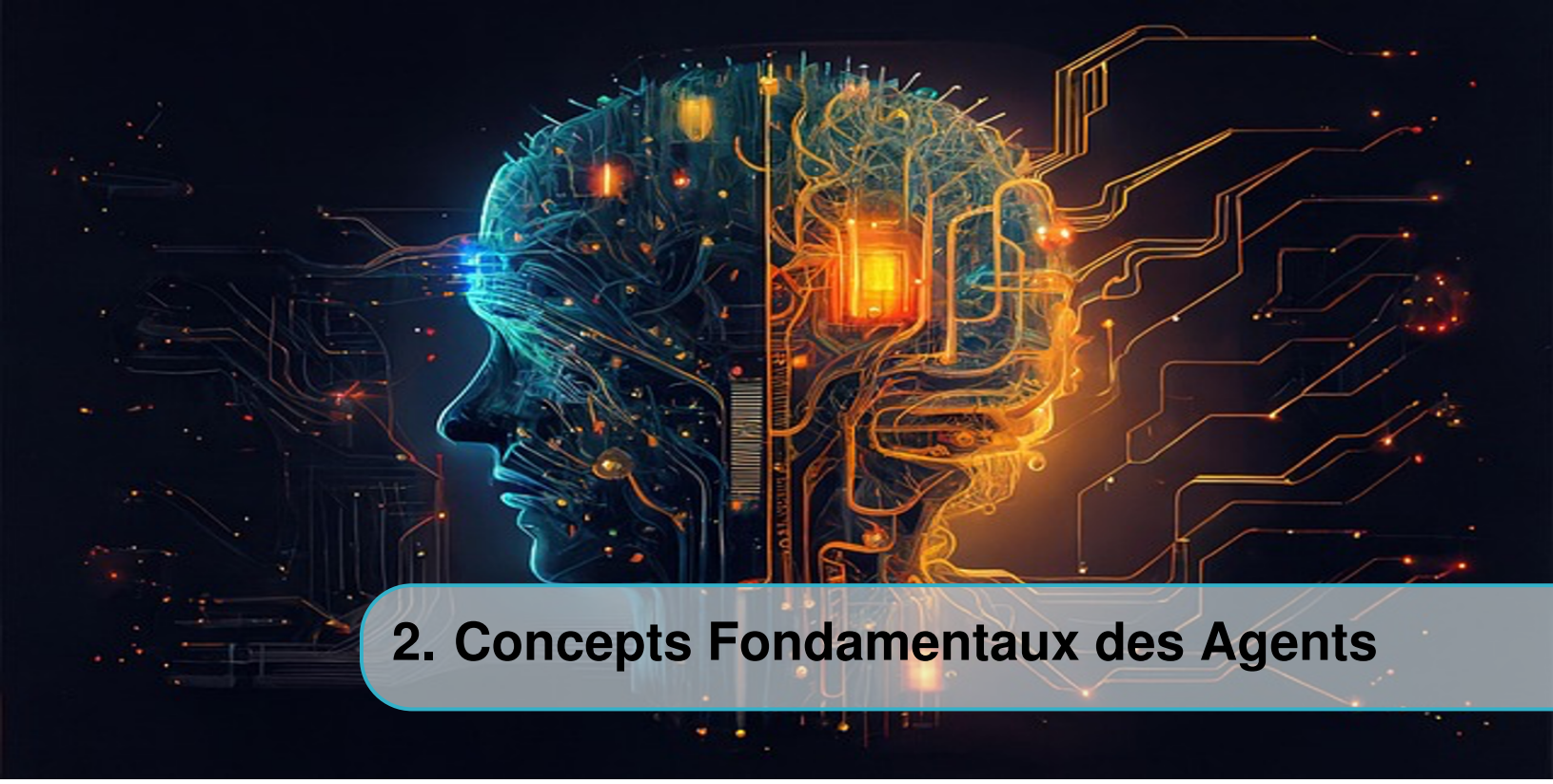
Exercice 1.2 — Distinguer entre l'IA, l'IAD et les SMA.

- **Partie 1 : Questions Conceptuelles**
  1. Quelle est la principale limitation des systèmes d'IA traditionnels (centralisés) qui a motivé l'émergence de l'Intelligence Artificielle Distribuée (IAD) ?
  2. Expliquez la différence fondamentale entre l'IAD et les Systèmes Multi-Agents (SMA). En quoi les SMA sont-ils une concrétisation ou une approche spécifique de l'IAD ?
  3. Donnez deux caractéristiques essentielles d'un "agent" au sein d'un Système Multi-Agents qui le distinguent d'un simple "objet" dans la programmation orientée objet distribuée.
  4. Dans quel type d'environnement un SMA serait-il particulièrement pertinent, et pourquoi ?
- **Partie 2 : Scénarios d'Application**
  Pour chacun des scénarios ci-dessous, identifiez l'approche qui vous semble la plus appropriée (IA traditionnelle/centralisée, IAD, ou SMA) et justifiez votre choix en expliquant pourquoi cette approche est la mieux adaptée aux exigences du problème.
  1. **Scénario A : Système de Recommandation de Films**
     Une plateforme de streaming souhaite améliorer son système de recommandation de films. Le système actuel collecte l'historique de visionnage et les notes d'un utilisateur, puis utilise un algorithme complexe pour lui suggérer des films basés sur ses préférences individuelles.

Toutes les données sont traitées sur un serveur centralisé et les recommandations sont générées pour chaque utilisateur indépendamment.
- Quelle approche est la plus appropriée ? Justifiez.

2. **Scénario B : Gestion Intelligente du Trafic Urbain**
   Une ville souhaite optimiser la fluidité du trafic routier. Le système doit gérer dynamiquement les feux de signalisation, les panneaux à messages variables et les informations des capteurs de trafic disséminés dans toute la ville. L'objectif est de minimiser les embouteillages, réagir rapidement aux accidents ou événements inattendus, et permettre une certaine autonomie de décision aux carrefours individuels tout en assurant une coordination globale.
   - Quelle approche est la plus appropriée ? Justifiez.
3. **Scénario C : Contrôle de Flotte de Drones de Livraison**
   Une entreprise de livraison utilise une flotte de centaines de drones autonomes pour livrer des colis dans une grande zone urbaine. Chaque drone doit planifier son itinéraire, éviter les obstacles, gérer sa batterie et sa charge utile. De plus, les drones doivent pouvoir se coordonner entre eux pour optimiser la répartition des colis, éviter les zones de congestion aérienne, et partager les informations sur les conditions météorologiques locales ou les zones de livraison difficiles, même si un drone tombe en panne.
   - Quelle approche est la plus appropriée ? Justifiez.

- **Partie 3 : Réflexion Critique**
  Imaginez un problème où une solution purement "IA traditionnelle" serait suffisante, mais où l'implémentation d'un SMA serait une "sur-ingénierie" (trop complexe pour le besoin). Décrivez le problème et expliquez pourquoi le SMA ne serait pas nécessaire.



2. Concepts Fondamentaux des Agents

2.1 Introduction

Le concept d'agent intelligent constitue une notion fondamentale en intelligence artificielle, en informatique distribuée et dans de nombreux domaines connexes. Ces entités autonomes, capables de percevoir leur environnement et d'agir sur celui-ci, représentent une approche novatrice pour résoudre des problèmes complexes dans des contextes dynamiques et incertains.
Ce chapitre propose une exploration approfondie des fondements des agents intelligents, leurs propriétés distinctives, les différents types et modèles d'agents, en mettant l'accent sur leurs définitions, propriétés, types et environnements d'évolution. Une attention particulière sera portée aux caractéristiques de l'environnement, un facteur crucial dans la conception et le comportement des agents.

2.2 Qu'est-ce qu'un Agent ?

La notion d'agent est utilisée dans de nombreux domaines notamment en sociologie, en biologie, en psychologie cognitive, en psychologie sociale et en informatique [6]. Un agent peut être un système mécanique (imprimante), biologique (plantes, animaux, humains) ou logiciel (programme) qui interagit avec son environnement. Cependant, il n'existe pas de définition universellement acceptée de ce qu'est un agent. Plusieurs définitions coexistent dans la littérature scientifique [8, 7]. Dans ce qui suit nous présentons les définitions les plus populaires émanant de grands chercheurs dans ce domaine:

◆ *Définitions*

- *Russell (1997)* [5] : "Un agent est une entité qui perçoit son environnement et agit sur celui-ci."
- *Wooldridge et Jennings (1995)* [9] : "Un agent est un système informatique, situé dans un environnement, et qui agit d'une façon autonome pour atteindre les objectifs (buts) pour lesquels il a été conçu."
- *IBM* : "Les agents intelligents sont des entités logicielles qui réalisent des opérations à la place d'un utilisateur ou d'un autre programme, avec une sorte d'autonomie, et pour faire cela ils utilisent une

sorte de connaissance ou de représentation des buts ou des désirs de l'utilisateur" [10].
- ***Shoham (1993)*** [8] : "Un agent est une entité qui fonctionne continuellement et de manière autonome dans un environnement où d'autres processus se déroulent et d'autres agents existent."
- ***Ferber (1995)*** [15] : "Un agent est une entité autonome, réelle ou virtuelle, évoluant dans un environnement, capable de percevoir et d'agir dessus, qui dans un univers multi-agents, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de ses connaissances et des interactions avec les autres agents."

Pour les besoins de ce cours, nous adopterons une définition synthétique : un ***agent*** est une entité, qu'elle soit logicielle, matérielle ou biologique, qui ***perçoit*** son environnement à travers des ***capteurs*** (senseurs) et agit sur cet environnement par l'intermédiaire d'***actionneurs*** (effecteurs), dans le but d'atteindre ses objectifs ou d'influencer son état interne et/ou externe, voir figure 2.1.

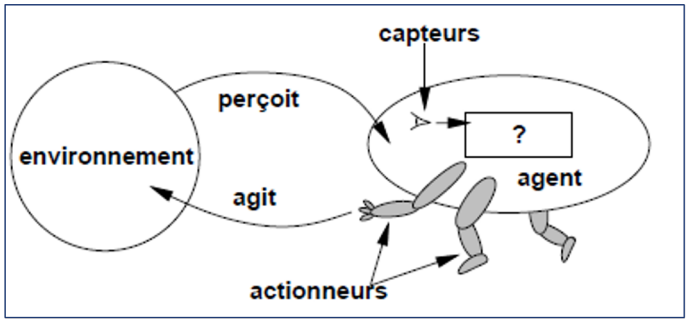


Figure 2.1: Agent & Environnement.

■ **Exemple 2.1 — Exemples d'agents : .**
- **Agent humain** : yeux, oreilles et autres organes comme **capteurs** ; mains, jambes, voies vocales comme **actionneurs**;
- **Agent Robotique** : caméras et détecteurs de portée infrarouge comme **capteurs** ; divers moteurs comme **actionneurs**, voir figure 2.2;
- **Agent Logiciel** : clics souris, frappes clavier, contenu des fichiers et paquets réseau comme **entrées sensorielles** ; affichage à l'écran, écriture de fichiers et envoi de paquets sur réseau comme **actionneurs**

L'intelligence d'un agent peut être vue comme une fonction qui associe un historique de données sensorielles (*percept history*) à une action :

$$f : P \rightarrow A^*$$

En pratique le processus est implémenté par un programme sur une architecture matérielle et logicielle particulière.

2.3 Propriétés des Agents Intelligents

Un agent intelligent possède plusieurs propriétés essentielles qui définissent son comportement dans un environnement complexe [9, 6] : la **réactivité**, la **proactivité**, la **sociabilité**, l'**autonomie** et la **rationalité**.

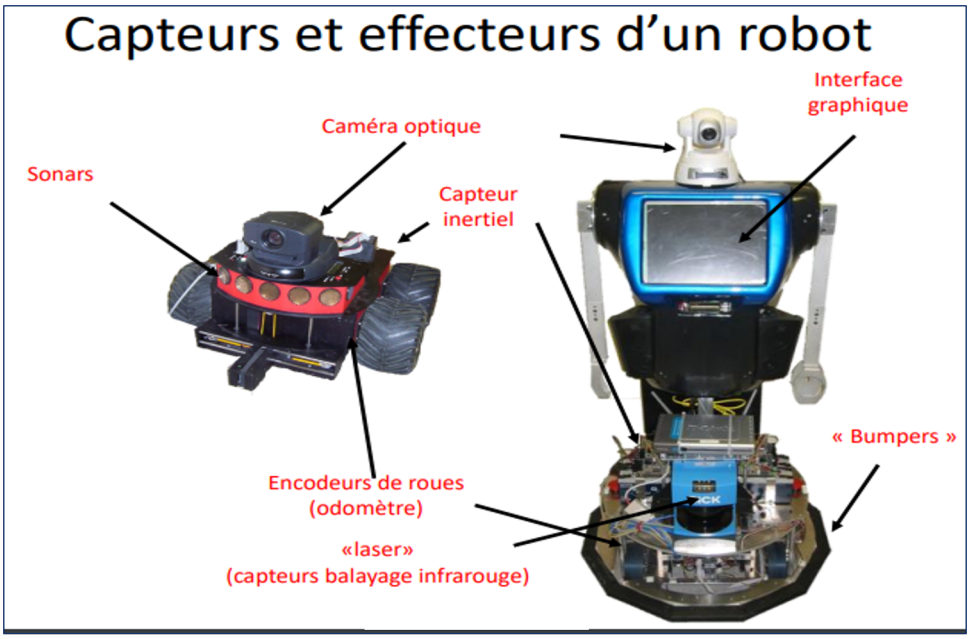


Figure 2.2: Capteurs et Effecteurs d'un Agent Robot.

2.3.1 Réactivité

Un agent **réactif** est capable de percevoir les changements dans son environnement (stimuli) et de réagir immédiatement sans **conserver d'historique** ou de **mémoire** sur les événements passés. Il suit une logique d'**action-réaction**, souvent basée sur des règles conditionnelles de type :

Si condition vérifiée alors exécuter action

Les agents réactifs sont utilisés dans des systèmes où les décisions doivent être prises rapidement, sans analyse approfondie du contexte. Ils sont rapides mais peuvent être inefficaces car ils peuvent répéter des actions inutiles.

▪ Exemple 2.2 — robot aspirateur intelligent.
Soit un robot aspirateur intelligent qui doit nettoyer une pièce. Il fonctionne de manière réactive en suivant ces règles :

1. **Si** le sol est sale, **alors** il aspire.
2. **S**'il rencontre un mur, **alors** il change de direction.
3. **Si** la batterie est faible, **alors** il retourne à sa station de recharge.

◆ *Comportement attendu* :

- Si on place de la poussière devant lui, il réagit immédiatement en aspirant.
- Si l'on met un obstacle (chaise, mur), il ne réfléchit pas, il change simplement de direction.
- Il ne garde aucune mémoire de l'endroit qu'il a déjà nettoyé, il se contente de réagir en temps réel.

2.3.2 Proactivité

Contrairement à un agent réactif, un agent **proactif** ne se contente pas de réagir aux stimuli de l'environnement, mais **anticipe** des événements futurs et prend des **initiatives** pour atteindre ses objectifs à long terme. Il ne se limite pas à l'action-réaction, mais **planifie** des actions à l'avance pour optimiser son comportement.

Un agent proactif est **persistant** car il est muni d'au moins un but qu'il cherche à satisfaire de manière persistante tant que :

- Il pense que c'est encore possible (précondition logique);
- Il possède les ressources pour le faire (précondition physique).

R **Différence clé avec la réactivité:**
- Réactif → Réagit aux événements immédiats sans anticipation
- Proactif → Prend des décisions en fonction d'objectifs à long terme.

■ **Exemple 2.3 — Robot aspirateur intelligent.**
Reprenons l'exemple du robot aspirateur, mais cette fois-ci avec une **approche proactive**:

1. **Planification du nettoyage** : Il crée une carte de la pièce et planifie un chemin optimal pour couvrir toute la surface.
2. **Anticipation des obstacles**: Plutôt que de changer de direction au hasard, il détecte les objets à l'avance et ajuste son parcours.
3. **Gestion de la batterie** : Il surveille son niveau de charge et retourne à sa base avant que la batterie ne soit trop faible.
4. **Détection des zones sales fréquentes** : Il apprend que certaines zones sont plus sales que d'autres (exple : entrée de la maison) et y retourne plus souvent.

R **Différence clé avec l'agent réactif:**
- Réactif → nettoie au hasard et peut repasser plusieurs fois au même endroit: **rapide mais inefficace**
- Proactif → planifie une stratégie pour optimiser le nettoyage et éviter de repasser inutilement: **plus intelligent**.

2.3.3 Sociabilité

Un agent sociable est capable de **communiquer et interagir** avec d'autres agents ou avec des humains pour atteindre un objectif commun [9]. Il peut **échanger** des informations, **négocier**, **collaborer** ou se **coordonner** avec d'autres entités. Par exemple, des drones de livraison qui communiquent entre eux pour éviter les collisions et optimiser les livraisons.

■ **Exemple 2.4 — Système de réservation de vols.**
Soit un système multi-agents où plusieurs agents interagissent pour réserver un voyage:

- **Agent utilisateur** : exprime les besoins (trouver un vol pas cher vers la destination demandée);
- **Agent comparateur de prix** : communique avec différents sites pour collecter les offres disponibles;
- **Agent compagnie aérienne** : fournit les vols disponibles avec leurs prix;
- **Agent hôtelier** : propose des hôtels correspondants aux dates du vol.

2.3.4 Autonomie

Un agent autonome est capable de prendre des décisions et d'agir **sans intervention humaine**. Il peut gérer ses ressources et choisir la meilleure action en fonction de ses objectifs. L'agent a un **auto-contrôle** de son état et de ses actions.
- **Caractéristiques de l'autonomie** :

- Prise de **décisions indépendante**: un agent autonome doit prendre ses décisions **« soi-même »** et ne doit dépendre de personne (agent ou humain) pour prendre ses décisions;
- Pouvoir de dire **"non"** à certaines requêtes. Par exemple, un agent peut décider de ne pas traiter un message en raison de sa charge de travail ou Parce que des buts plus importants doivent être satisfaits;
- Anticipation du non-respect des engagements: Les engagements ne sont que des promesses, Les agents doivent anticiper le non-respect des engagements.

Ce concept d'agent autonome est très "humain" mais le défi réside dans : Comment programmer l'autonomie ?

■ Exemple 2.5 — Voiture autonome.
Une voiture sans conducteur doit décider seule de :
- Accélérer ou freiner en fonction de la circulation.
- Changer de voie si un obstacle est détecté.
- S'arrêter aux feux rouges et respecter le code de la route.

Elle ne demande aucune intervention humaine, elle fonctionne en analysant son environnement.

2.3.5 Adaptabilité

Un agent adaptatif peut s'**adapter** à des environnements dynamiques et changeants. Il modifie son comportement en fonction des nouvelles informations ou des changements dans son environnement.
Il ne suit pas un **script fixe**, mais s'ajuste aux nouvelles situations pour atteindre son objectif. Pour cela, il doit:
- Percevoir et évaluer continuellement la situation;
- Construire des représentations en cours de fonctionnement, c'est-à-dire être capable d'apprentissage;
- Élaborer des plans dynamiques qui lancent des processus internes ou au contraire les stoppent.

Par exemple, un véhicule autonome ajuste sa trajectoire en fonction des obstacles ou des conditions routières.

2.3.6 Capacité d'apprentissage

Un agent peut être doté de capacités d'apprentissage lui permettant d'améliorer ses performances au fil du temps en tirant des leçons de ses expériences passées. Il utilise des techniques comme le **machine learning** ou le **reinforcement learning**. Par exemple, un système de recommandation comme Netflix améliore ses suggestions en fonction des interactions passées de l'utilisateur.

2.4 Typologie des Agents

La classification des agents intelligents repose sur leurs caractéristiques comportementales, leurs capacités cognitives et leur niveau d'autonomie [6, 7]. Cette typologie vise à mieux comprendre le rôle que peut jouer chaque type d'agent dans un système intelligent et à adapter leur conception en fonction des besoins spécifiques de la tâche.
Les critères principaux utilisés dans cette classification sont :
- La capacité de réagir à l'environnement (réactivité);
- La capacité de planification et de raisonnement (cognition);
- Le niveau de décision autonome;
- L'interaction avec d'autres agents ou avec des humains;
- La capacité à combiner plusieurs formes de comportement (hybridation);
- L'optimisation des actions selon une fonction d'utilité (rationalité).

Plus de détails sur les architectures des agents seront discutés dans le chapitre suivant.

2.4.1 Agents Réactifs

Les agents **réactifs** sont les plus simples. Leur comportement est dicté par une logique stimulus-réponse : à chaque perception correspond une action immédiate. Ils ne possèdent ni mémoire ni modélisation interne de l'environnement.
Les agents réactifs ne sont **pas autonomes ni intelligents**, car ils ne prennent pas en compte un objectif global, ils réagissent uniquement à l'instant présent. **- Caractéristiques :**

- Pas de représentation du monde;
- Pas d'apprentissage;
- Réponse immédiate à des événements;
- Pas de prise en compte d'objectifs à long terme.

■ Exemple 2.6 — Un robot aspirateur basique.
- Si un obstacle est détecté → changer de direction.
- Si le sol est sale → aspirer.
- Sinon → avancer aléatoirement.

2.4.2 Agents Cognitifs

Les agents **cognitifs** sont plus avancés que les agents réactifs. intègrent une modélisation interne du monde, stockent des informations, planifient des actions et prennent des décisions fondées sur l'analyse de l'environnement [6]. Ils sont plus intelligents, mais nécessitent plus de ressources pour stocker et traiter les informations.
- Caractéristiques:

- Construisent une représentation interne du monde;
- Mémorisent et utilisent des expériences passées;
- Planifient des actions en anticipant les conséquences;
- S'adaptent aux obstacles en ajustant leur stratégie.

Dans ce type d'agents, on retrouve les propriétés: **Proactivité, Autonomie et Adaptabilité**.

■ Exemple 2.7 — Robot aspirateur cognitif.
Un robot aspirateur cognitif utilise un modèle interne du monde pour :
1. Mémoriser les zones déjà nettoyées.
2. Planifier le trajet le plus efficace pour minimiser le temps et l'énergie.
3. S'adapter aux obstacles en ajustant son itinéraire.
4. Apprendre des habitudes de l'utilisateur (ex. : il sait qu'il y a souvent des miettes sous la table après le repas).

2.4.3 Agents Hybrides

Les agents **hybrides** combinent les avantages des agents réactifs (rapidité de réaction) et cognitifs (prise de décision complexe) [9]. Ils utilisent une architecture à plusieurs couches, où les couches réactives gèrent les urgences tandis que les couches cognitives gèrent les plans à long terme.
- Caractéristiques:

- Couche réactive pour les réponses immédiates;
- Couche cognitive pour la planification, la mémorisation et l'apprentissage;
- Capacité d'adaptation à différents contextes.

▪ Exemple 2.8 — Une voiture autonome.
Une voiture autonome fonctionne avec deux niveaux de décision:
- Réactif: Elle freine immédiatement si un obstacle est détecté.
- Cognitif: Elle planifie l'itinéraire en tenant compte du trafic et des préférences du conducteur.

2.4.4 Agents Rationnels

Un agent **rationnel** prend toujours la meilleure décision selon :
- Ses objectifs,
- Les informations disponibles,
- Une f**onction d'utilité** qui évalue la qualité d'une action.
Un agent rationnel est celui qui fait la bonne action dans une situation donnée.

◆ ***Fonction d'Utilité*** : La fonction d'utilité est une mesure qui quantifie le degré de réussite d'une action ou le bénéfice que l'agent retire d'un état ou d'une action. Elle guide les décisions de l'agent.

- Caractéristiques:
- **Objectif clair:** L'agent sait ce qu'il veut maximiser ou minimiser.
- **Décision optimale**: Il choisit l'action qui maximise son utilité ou minimise son coût.
- **Rationalité limitée:** Il prend des décisions optimales en fonction des informations disponibles, même si elles sont incomplètes.

▪ Exemple 2.9 — IA joueur d'échecs.
Un agent qui joue aux échecs évalue chaque coup possible et choisit celui qui maximise ses chances de gagner. Il utilise une **fonction d'utilité** pour noter les coups :
- +10 points pour un coup gagnant.
- -5 points s'il perd une pièce.
- 0 point pour un coup neutre.

Les agents rationnels sont essentiels en intelligence artificielle car ils permettent de prendre des décisions optimales grâce à une fonction d'utilité. La bonne action est celle qui assurera le plus de succès à l'agent. une synthèse comparative des différents types d'agents est présentée dans le tableau 2.1.

Table 2.1: Synthèse comparative des types d'agents

Type d'Agent	Réactivité	Cognition	Autonomie	Planification	Exemple
Agent Réactif	Oui	Non	Faible	Non	Robot aspirateur basique
Agent Cognitif	Oui	Oui	Moyenne	Oui	Assistant vocal intelligent
Agent Hybride	Oui	Oui	Forte	Oui	Voiture autonome
Agent Rationnel	Oui	Oui	Forte	Optimale	IA jouant aux échecs

2.5 Rationalité et Mesure de Performance

Un agent rationnel doit décider quand et comment évaluer le succès de l'agent [5, 6]. Pour cela, il faut définir une **mesure de performance** (comment) et le **moment** de calculer cette mesure (quand).

◆ *Mesure de performance:* le ou les critères qui déterminent le **succès** d'un agent; elle peut être externe, fixée par le concepteur ou propre à la tâche.
Le **moment** d'évaluer la performance est également important: nous voulons mesurer la performance sur le long terme, qu'il s'agisse d'un travail de huit heures ou d'une durée de vie.

Ainsi, être rationnel à un instant donné dépend de quatre paramètres :

1. La mesure de performance qui définit le degré du succès,
2. Tout ce que l'agent a perçu jusqu'à présent (séquence de perceptions),
3. Ce que l'agent sait de l'environnement,
4. Les actions que l'agent peut effectuer

Un agent **rationnel idéal** est un agent qui, pour chaque séquence de perceptions possible, exécute l'action qui permet de maximiser **sa mesure de performance**, en se basant sur la séquence de perceptions et sur toutes les connaissances intégrées dont il dispose.

■ **Exemple 2.10**
Soit un agent supposé aspirer un sol sale Les mesures de performance possibles:
- Quantité de saleté ramassée en 8 heures,
- Quantité d'électricité consommée et la quantité de bruit généré,
- **Une meilleure option**:
  - Récompenser l'agent pour un plancher propre:
  - Exemple: un point pour chaque carré propre à chaque intervalle de temps
  - Peut-être avec une pénalité pour l'électricité consommée.

R Il faut distinguer entre :

- **Rationalité** $\neq$ **Omniscience**: Un agent omniscient connaît le résultat réel de ses actions. L'omniscience est impossible en réalité.
- **Rationalité** $\neq$ **Clairvoyance** : Les perceptions peuvent ne pas fournir toute l'information pertinente. Les résultats de l'action peuvent ne pas être conformes aux attentes.
- **Rationalité** $\neq$ **Réussite**: La rationalité se réfère au succès escompté compte tenu de ce qui a été perçu. Si perceptions insuffisantes, alors résultats non satisfaisants.

2.6 Description de l'Environnement de la Tâche (PEAS)

Avant de concevoir un agent intelligent, il est essentiel de modéliser formellement l'environnement dans lequel il doit opérer, ainsi que la tâche qu'il doit accomplir. Pour cela, on utilise le modèle PEAS pour décrire l'environnement de la tâche, voir le tableau 2.2.

■ **Exemple 2.11 — Exemple d'Interaction d'un agent commercial avec un Client.** **Scénario**: Un client visite un site e-commerce pour acheter un smartphone.

1. **Capteurs (S)**:
   - L'agent détecte que l'utilisateur a consulté plusieurs modèles de smartphones.
   - Il analyse les avis et les caractéristiques qui intéressent le client.
2. **Traitement**

Table 2.2: Framework PEAS pour la conception d'agents

Composant	Description	Exemple (Robot Aspirateur)
P - Performance	Critères d'évaluation de la réussite de l'agent	Surface nettoyée, énergie économisée
E - Environnement	Monde dans lequel l'agent opère	Maison avec meubles, escaliers
A - Actionneurs	Actions que l'agent peut effectuer	Roues motrices, système d'aspiration
S - Capteurs (Sensors)	Moyens de perception de l'environnement	Capteurs de poussière, infrarouge, gyroscope, détection de collision, lidar

- L'agent utilise un algorithme de recommandation pour proposer un modèle qui correspond aux préférences de l'utilisateur (marque, budget, fonctionnalités).

3. **Actionneurs (A)**
   - Affichage d'un message personnalisé : "Vous semblez intéressé par un smartphone ! Découvrez nos offres spéciales sur les modèles Samsung."
   - Ajout d'un code promo temporaire pour inciter à l'achat.
4. **Évaluation de la Performance (P)**
   - Si le client achète le produit, la performance de l'agent est positive.
   - Sinon, l'agent ajuste sa stratégie en envoyant un rappel par e-mail ou en proposant un produit alternatif.

2.6.1 Exemples d'application du modèle PEAS

Plusieurs exemples d'application du modèle PEAS sont présentés dans le tableau 2.4

2.6.2 Importance du PEAS dans la conception d'Agents

Le modèle PEAS est un outil fondamental pour concevoir des agents rationnels. Il offre une grille claire pour comprendre ce que l'agent doit accomplir, dans quel contexte, avec quels moyens d'action et de perception. Cette approche permet une conception modulaire et une implémentation plus fiable des agents dans des environnements complexes. Nous notons que :

- Une mauvaise définition du PEAS conduit souvent à des comportements non désirés.
- Le PEAS permet de clarifier les attentes et de guider le choix de l'architecture de l'agent (réactif, cognitif, hybride...).
- Le comportement optimal dépend fortement du PEAS : un changement dans la mesure de performance peut totalement modifier les stratégies de l'agent.

2.7 Propriétés de l'Environnement

Les propriétés de l'environnement influencent la conception, l'architecture et les stratégies des agents intelligents. On distingue plusieurs propriétés fondamentales d'un environnement. La combinaison de ces propriétés détermine sa complexité et influence le niveau d'intelligence requis par l'agent.

2.7.1 Accessibilité: Entièrement Observable vs. Partiellement Observable

Est-ce que les capteurs de l'agent lui donnent accès à l'état complet de l'environnement à tout moment ?

• **Observable (Accessible)** : Un environnement est **accessible** si les capteurs détectent tous les aspects pertinents pour le choix de l'action. L'agent dispose de toutes les informations nécessaires pour prendre une

Table 2.3: Exemples de PEAS pour différents agents intelligents

Agent	Robot Aspirateur	Voiture Autonome	Agent Commercial Internet
Performance (P)	• Maximiser la surface nettoyée • Minimiser la consommation d'énergie • Éviter les obstacles	• Minimiser le temps de trajet • Éviter les accidents • Respecter le code de la route	• Maximiser les ventes • Améliorer la satisfaction client • Proposer des recommandations pertinentes
Environnement (E)	• Maison avec différentes pièces • Obstacles (meubles, animaux, escaliers) • Surfaces variées	• Routes, autoroutes • Feux de signalisation • Piétons, autres véhicules • Conditions météo	• Site e-commerce • Base de données produits • Profils clients • Historique d'achats
Actionneurs (A)	• Roues pour se déplacer • Brosse pour nettoyer • Système d'aspiration	• Moteur • Frein • Volant • Clignotants	• Affichage de recommandations • Envoi de notifications • Adaptation des prix • Chat interactif
Capteurs (S)	• Capteur de poussière • Capteur de distance • Gyroscope • Capteur de batterie	• Caméras • LIDAR • GPS • Capteurs de vitesse • Radar	• Analyse des clics • Suivi des paniers d'achat • Traitement du langage naturel • Analyse comportementale

décision optimale. Un environnement accessible est pratique parce que l'agent n'a pas besoin de maintenir un état interne pour garder la trace du monde.

• **Partiellement Observable (Inaccessible)** : Certaines informations sont manquantes ou cachées.

■ **Exemple 2.12**
Exemple Accessible :
- Un jeu d'échecs : L'agent (joueur IA) voit entièrement l'échiquier et connaît la position de toutes les pièces.
Exemple Inaccessible :
- Une voiture autonome : Elle ne connaît pas toujours l'état exact des autres conducteurs (fatigue, intentions, etc.), elle doit se baser sur des capteurs imparfaits. Ceci nécessite des agents basés sur un modèle ou des

agents apprenants pour inférer l'état caché du monde.

2.7.2 Déterminisme : Déterministe vs. Stochastique

Est-ce que le prochain état de l'environnement est complètement déterminé par son état courant et l'action de l'agent ?

Un environnement est **déterministe** si les résultats des actions de l'agent sont entièrement prévisibles. Il est **stochastique** si les résultats sont incertains, même avec les mêmes conditions initiales.
• **Déterministe** : L'état suivant est entièrement déterminé par l'état actuel et les actions choisies. Un environnement déterministe donne toujours le même résultat pour une même action.

• **Stochastique (non-déterministe)** : Introduit un facteur d'incertitude dans le résultat des actions. Si l'environnement est inaccessible, il peut sembler non déterministe. Il est alors difficile de garder une trace de toutes les activités des aspects inaccessibles.

■ **Exemple 2.13**
Exemple Déterministe :
- Un robot industriel qui place des objets sur une chaîne de production : si son bras est bien calibré, chaque action donne le même résultat.
- Un jeu de puzzle où chaque mouvement a une conséquence unique et prévisible.
Exemple Stochastique :
- Une prévision météorologique : Même avec des données complètes, il y a une part d'incertitude dans le climat (vents, nuages imprévus).
- Nécessite des agents capables de raisonner sous incertitude (probabilités, fonctions d'utilité).

2.7.3 Temporalité : Épisodique vs. Séquentiel

Est-ce que l'action prochaine de l'agent dépend de l'épisode précédent?
Un épisode est une séquence perception-action.

• **Épisodique** : chaque action est indépendante des précédentes. l'expérience de l'agent est divisée en **"épisodes"**. Chaque épisode consiste en la perception puis l'action de l'agent. La qualité de son action ne dépend que de l'épisode lui-même, car les épisodes ne dépendent pas des actions effectuées dans les épisodes précédents.
Les environnements épisodiques sont beaucoup plus simples parce que l'agent n'a pas besoin de penser à l'avenir.

•**Séquentiel (Non Épisodique**) : L'action dépend des décisions passées.

■ **Exemple 2.14**
Exemple Épisodique :
- Un agent de tri de courrier : Il analyse chaque enveloppe indépendamment et ne tient pas compte des précédentes.
- Un système de détection de spam qui traite chaque email indépendamment.
Exemple Non Épisodique :
- Un assistant virtuel : Il doit se souvenir des interactions passées pour améliorer ses recommandations.

- Un jeu d'échecs, où les mouvements passés déterminent la configuration actuelle du plateau.

2.7.4 Dynamisme : Statique vs. Dynamique

Est-ce que l'environnement change pendant que l'agent délibère?

Un environnement est statique s'il ne change pas pendant que l'agent délibère. Il est dynamique si l'environnement peut évoluer indépendamment de l'agent.
• **statique** : L'agent peut planifier tranquillement.

• **Dynamique** : L'environnement change constamment, exigeant réactivité.

• **Semi-dynamique** : L'environnement ne change pas, mais le score de performance de l'agent change avec le temps.

■ **Exemple 2.15**
Exemple Statique :
- Un puzzle : Rien ne bouge tant que l'agent ne modifie pas les pièces.
- Un système de recommandation de films où la base de données ne change pas significativement en temps réel.
Dynamique :
- Une voiture autonome : Les autres voitures, les piétons et les feux de signalisation changent en permanence.
- Un agent de contrôle de trafic aérien, où les avions se déplacent constamment.

2.7.5 Structure : Discret vs. Continu

La distinction entre discret et continu peut être appliquée :
- à l'état de l'environnement
- à la façon dont le temps est géré
- et aux perceptions et aux actions de l'agent.

Un environnement est discret si les perceptions, les états et les actions sont en nombre fini ou clairement définis. Il est continu si ces éléments varient sur des plages continues.
• **Discret** : État du monde représenté par une grille, des symboles.

•**Continu** :États exprimés par des grandeurs physiques, temps, espace (ex: valeurs réelles).

■ **Exemple 2.16**
Exemple Discret :
- Les jeux d'échecs sont discrets - il y a un nombre fixe de coups possibles à chaque tour.
- Les cases d'un plateau de jeu, les commandes ON/OFF.
Continu :
- La conduite en taxi est continue - la vitesse et l'emplacement du taxi et des autres véhicules balayent une gamme de valeurs continues.
- Un agent de contrôle de trafic aérien, où les avions se déplacent constamment.

2.7.6 Agent unique vs. Multi-agents

Est ce qu'il y a un seul agent ou plusieurs qui interagissent ensemble?

Un environnement est dit mono-agent s'il n'y a qu'un seul agent qui agit. Il est multi-agents si plusieurs agents interagissent (coopération, compétition ou coexistence).
• **Mono-Agent** : L'agent est seul responsable du résultat.

• **Multi-Agents** : L'agent doit composer avec les autres d'où la nécessité de mécanismes de communication, de coordination, de négociation, et de modélisation des autres agents.
Le tableau 2.4 résume les différents types d'environnement où peuvent se situer des agents intelligents.

Table 2.4: Récapitulatif des propriétés de l'environnement.

Propriété	Deux cas opposés	Exemple simple	Exemple complexe
Accessibilité	Accessible / Inaccessible	Jeu d'échecs	Conduite automobile
Déterminisme	Déterministe / Stochastique	Robot industriel	Météo
Temporalité	Épisodique / Séquentiel	Tri de courrier	Dialogue conversationnel
Dynamisme	Statique / Dynamique	Puzzle	Réalité urbaine
Structure	Discret / Continu	Morpion	Robot mobile
Agents impliqués	Mono-agent / Multi-agents	Jeu solo	Système multi-drones

2.7.7 Exemples d'environnements

• **Complexité dans un environnement de conduite autonome** : Il s'agit d'un environnement extrêmement complexe qui nécessite un agent très évolué (architecture hybride, capacités de raisonnement, apprentissage, adaptation en temps réel, etc.). Les caractéristiques de l'environnement d'une voiture autonome sont résumés comme suit:

Propriété	Situation dans la conduite autonome
Accessibilité	✓ Partiellement observable
Déterminisme	✓ Stochastique
Temporalité	✓ Séquentiel
Dynamisme	✓ Dynamique
Structure	✓ Continu
Nombre d'agents	✓ Multi-agents

D'autres exemples sont résumés dans le tableau 2.5

2.8 Agent versus Objet: Comparaison des Paradigmes

Est-ce que les agents nous apportent quelque chose de nouveau par rapport à l'objet?

Environnement	Observable	Déterministe	Épisodique	Statique	Discret	Agents
Mots-croisés	Complètement	Déterministe	Séquentiel	Statique	Discret	Un
Échec avec une horloge	Complètement	Stratégique	Séquentiel	Semi	Discret	Multi
Poker	Partiellement	Stratégique	Séquentiel	Statique	Discret	Multi
Backgammon	Complètement	Stochastique	Séquentiel	Statique	Discret	Multi
Conduire un taxi	Partiellement	Stochastique	Séquentiel	Dynamique	Continu	Multi
Diagnostique médical	Partiellement	Stochastique	Séquentiel	Dynamique	Continu	Un
Analyse d'image	Complètement	Déterministe	Épisodique	Semi	Continu	Un
Robot ramasseur de pièces	Partiellement	Stochastique	Épisodique	Dynamique	Continu	Un
Contrôleur de raffinerie	Partiellement	Stochastique	Séquentiel	Dynamique	Continu	Un
Enseignant interactif	Partiellement	Stochastique	Séquentiel	Dynamique	Discret	Multi

Table 2.5: Exemples d'Environnements.

La programmation orientée objet (POO) est un paradigme bien établi dans la conception de logiciels [6]. Chaque entité est un objet passif qui regroupe des données (attributs) et des comportements (méthodes). Les objets n'agissent que lorsqu'ils reçoivent des appels de methodes. Le paradigme des agents intelligents va au delà de POO en offrant une modélisation plus riche grâce à des entités intelligentes autonomes et proactives qui s'adaptent dans des environnements dynamiques complexes [9, 7]. La figure 2.3 présente une illustration des deux paragigmes.
Bien que les agents puissent être implémentés comme des objets, le paradigme agent introduit des caractéristiques comportementales et décisionnelles absentes dans la POO traditionnelle.
Le tableau 2.6 récapitule les differences clés entre les deux paradigmes.

- Quand utiliser quel paradigme ?

- ✗ **Paradigme Objet** :
  - ✓ Idéal pour les **systèmes centralisés** et bien définis où les interactions sont **déterministes**.
  - ✓ Utilisé en **développement logiciel classique** (applications, jeux vidéo, gestion de bases de données, etc.).
- ✗ **Paradigme Agent** :
  - ✓ Adapté aux **systèmes dynamiques et distribués** nécessitant de l'**adaptabilité** et de l'**autonomie**.
  - ✓ Utilisé pour les **systèmes multi-agents**, l'**intelligence artificielle**, l'**IoT**, la **robotique**, les marchés financiers, etc..

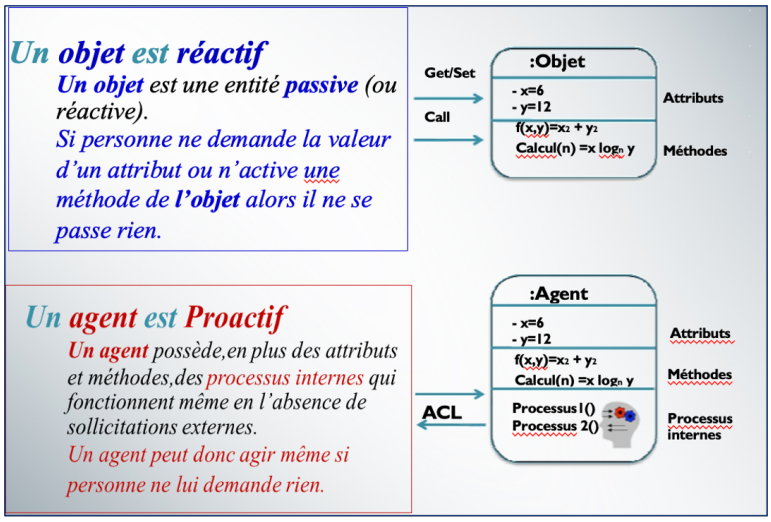


Figure 2.3: Les deux paradigmes Agent vs Objet.

Table 2.6: Comparaison des caractéristiques comportementales : Objet vs Agent

Critère	Objet	Agent
Autonomie	N'agit que lorsqu'une méthode est appelée	Elevé, Prend des décisions sans intervention externe
Réactivité	Réagit uniquement aux appels de méthodes, Ne réagit pas aux événements	Perçoit son environnement et réagit aux événements de manière autonome
Proactivité	Non, Ne possède aucun objectif propre. attend qu'on lui donne une tâche	Oui, Peut planifier des actions et poursuivre activement des buts à long terme
Sociabilité	Interaction limitée aux relations entre objets définies dans le programme.	Communication via messages (ex. ACL) lui permettant de collaborer avec d'autres agents pour résoudre un problème.
Encapsulation	Attributs et méthodes	Croyances, désirs et intentions (modèle BDI)
Contrôle	Contrôlé par le programme principal	Se contrôle lui-même (thread ou boucle interne)
Exécution	Exécute une méthode quand elle est appelée	Fonctionne en continu selon l'état et les perceptions

2.9 Domaines d'Application des Agents Intelligents

Les agents intelligents trouvent des applications dans une grande diversité de domaines, notamment ceux caractérisés par la complexité, la dynamique, la distribution ou l'incertitude [10, 6] tels que présentés par la figure 2.4 et le tableau 2.7.

Table 2.7: Domaines d'application des agents intelligents.

Domaine	Rôle des agents
Robotique autonome	Perception, adaptation, navigation autonome, interaction avec l'environnement physique
Transport autonome	Décision autonome, anticipation, coordination avec autres véhicules et infrastructures
E-commerce et recommandation	Recommandation personnalisée, négociation, adaptation aux préférences utilisateur
Systèmes multi-agents	Coordination, négociation, organisation distribuée dans des environnements complexes
Domotique / Intelligence ambiante	Réaction contextuelle, apprentissage des habitudes, contrôle intelligent du cadre de vie
Internet des objets (IoT)	Agents embarqués dans des capteurs/objets pour le suivi, la gestion et l'action locale intelligente
Finance et marchés	Trading algorithmique, agents enchérisseurs, évaluation des risques, détection de fraude
Jeux et simulation	Contrôle de PNJ, simulation sociale, comportement adaptatif dans des environnements immersifs
Éducation et formation	Agents pédagogiques, tutorat intelligent, personnalisation du contenu éducatif
Santé et assistance	Agents compagnons, assistance aux soins, suivi intelligent des patients et diagnostic

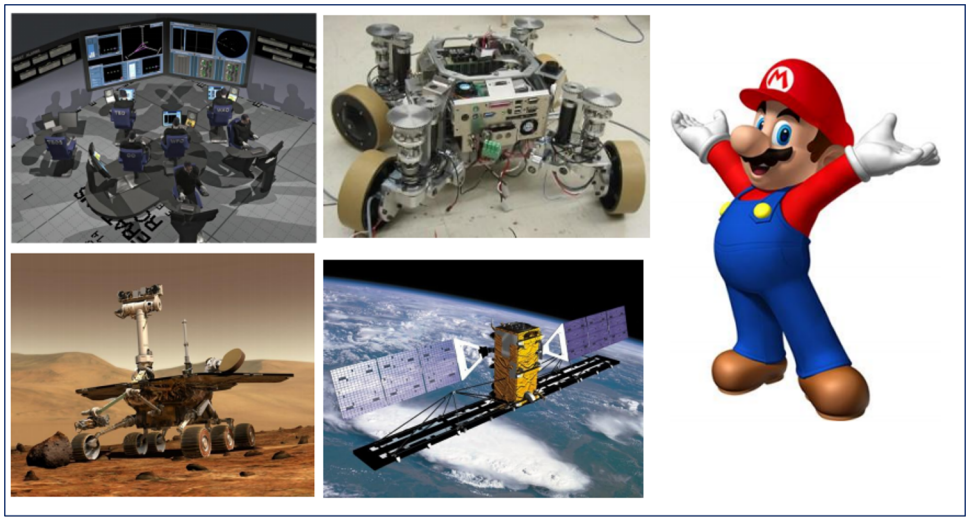


Figure 2.4: (1) Système d'aide à la décision; (2) Azimut-3; (3) Rover de la NASA; (4) Radarsat-II de l'ASC; (5) Mario de Nintendo.

2.10 Conclusion

Les agents intelligents représentent un paradigme puissant pour concevoir des systèmes autonomes capables d'évoluer dans des environnements complexes et dynamiques [6, 7]. Grâce à leurs propriétés de réactivité, proactivité, sociabilité, autonomie et adaptabilité [9], les agents représentent des

outils précieux pour résoudre des problèmes nécessitant une approche décentralisée et intelligente. afin de concevoir des systèmes multi-agents efficaces, il est impératif de bien comprendre les différents types d'agents, leurs propriétés et les caractéristiques de leur environnement [8]. Le choix entre les différentes architectures d'agents dépend largement du contexte d'application et des contraintes spécifiques du domaine considéré.

2.11 Exercices

Exercice 2.1 — Comprendre les notions de base et les propriétés d'un agent..

1. Comparez les définitions d'un agent selon les auteurs Russel et Ferber ? Quels éléments clés ajoute Ferber ?
2. Pourquoi dit-on qu'un agent rationnel n'est pas nécessairement omniscient ? Illustrez par un exemple.
3. Expliquez la différence entre autonomie et proactivité avec un exemple concret.
4. Même chose avec Adaptabilité vs Apprentissage.

Exercice 2.2 — Identifier et classifier différents types d'agents..

Classez les agents suivants dans la typologie : réactif, cognitif, hybride ou rationnel. Justifiez votre choix :

1. Un robot aspirateur sans mémoire.
2. Un système de navigation GPS.
3. Une IA jouant aux échecs.
4. Un assistant vocal apprenant les habitudes de l'utilisateur.

Exercice 2.3 — Être capable de spécifier un environnement de tâche..

1. Pour un agent de sécurité domotique (maison intelligente), remplissez les composantes PEAS :
   - Mesure de Performance
   - Environnement
   - Actuateurs
   - Capteurs
2. Faites de même pour un agent de chatbot médical.

Exercice 2.4 — Rationalité et utilité .

Un agent doit récolter des pommes dans un verger. Chaque pomme récoltée rapporte +5. Se déplacer coûte -1, et tomber d'une échelle coûte -20.

1. Calculez l'utilité de deux stratégies :
   - Stratégie A : Récolter 10 pommes avec 15 déplacements et 0 chute.
   - Stratégie B : Récolter 12 pommes avec 10 déplacements et 1 chute.
2. Quelle stratégie un agent rationnel choisirait-il ?

Exercice 2.5 — Conception d'Agent.

Concevez un agent pour optimiser l'irrigation dans une ferme intelligente :

1. Définissez PEAS.
2. Choisissez entre réactif, cognitif, ou hybride — justifiez.
3. Quelles propriétés environnementales rendent la tâche complexe ?

Exercice 2.6 — Agent vs Objet.

1. Pourquoi un agent de réservation de vols ne pourrait-il pas être implémenté comme un simple objet ?
2. Donnez un exemple où le paradigme objet est plus adapté que le paradigme agent.

Exercice 2.7 — Défis des Environnements Complexes.

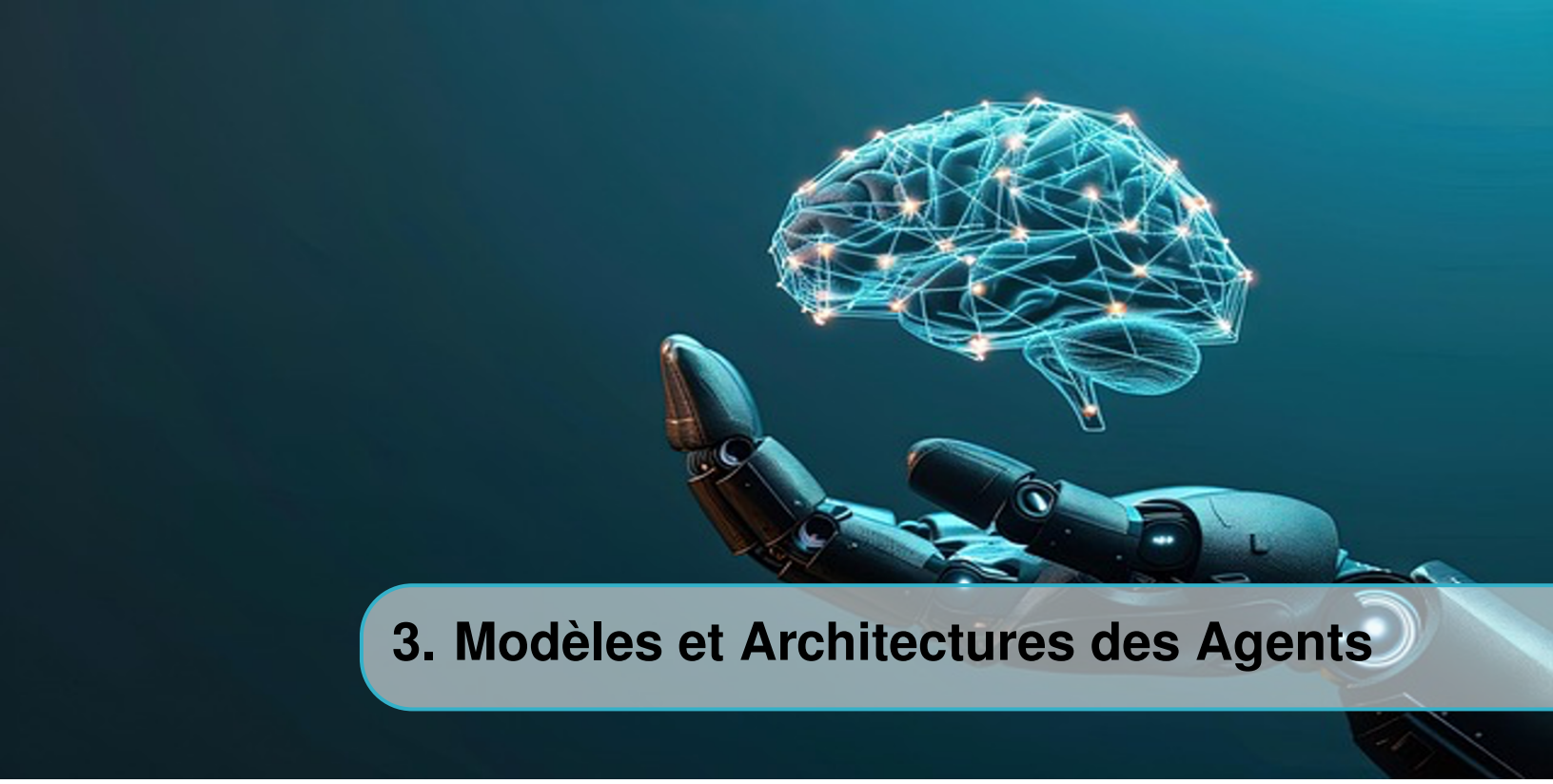
Un agent doit négocier des contrats dans un marché financier :

1. Pourquoi cet environnement est-il partiellement observable, stochastique, et multi-agent ?
2. Proposez une architecture d'agent hybride pour gérer ces défis.

Exercice 2.8 — Types d'agents.

Un agent qui suit les règles suivantes est-il réactif, proactif, ou hybride ? Justifiez.

- S'il détecte un obstacle, il change de direction.
- Il cartographie les zones nettoyées et optimise son parcours.
- Il retourne à la base quand la batterie est faible.



3. Modèles et Architectures des Agents

3.1 Introduction

Dans le domaine de l'intelligence artificielle, les agents constituent une approche fondamentale pour concevoir des systèmes autonomes capables d'interagir avec leur environnement [11, 12]. Ce chapitre explore les différents modèles et architectures d'agents, classés selon leur complexité croissante et leur capacité de prise de décision. De l'agent réactif simple aux architectures sophistiquées comme BDI (Belief-Desire-Intention), nous examinerons comment ces systèmes peuvent être conçus pour répondre aux exigences de différents domaines d'application.

3.2 Fondements des Architectures d'Agents

Rappelons qu'un agent intelligent peut être défini comme une entité autonome qui perçoit son environnement à travers des capteurs et agit sur cet environnement à travers des effecteurs, voir figure 3.1.

Un agent peut être modélisé comme une fonction qui transforme l'état perçu en action (t représente l'instant actuel):

$$Agent : Perception_t \rightarrow Action_t$$

Cette architecture repose sur trois éléments fondamentaux :

- ✓ **Perceptions (P)** : Informations captées de l'environnement par les capteurs
- ✓ **Actions (A) :** Réponses envoyées à l'environnement via les effecteurs
- ✓ **Programme de l'agent (f) :** Fonction de décision définissant l'action en fonction de la perception et, dans certains cas, de la mémoire ou du raisonnement

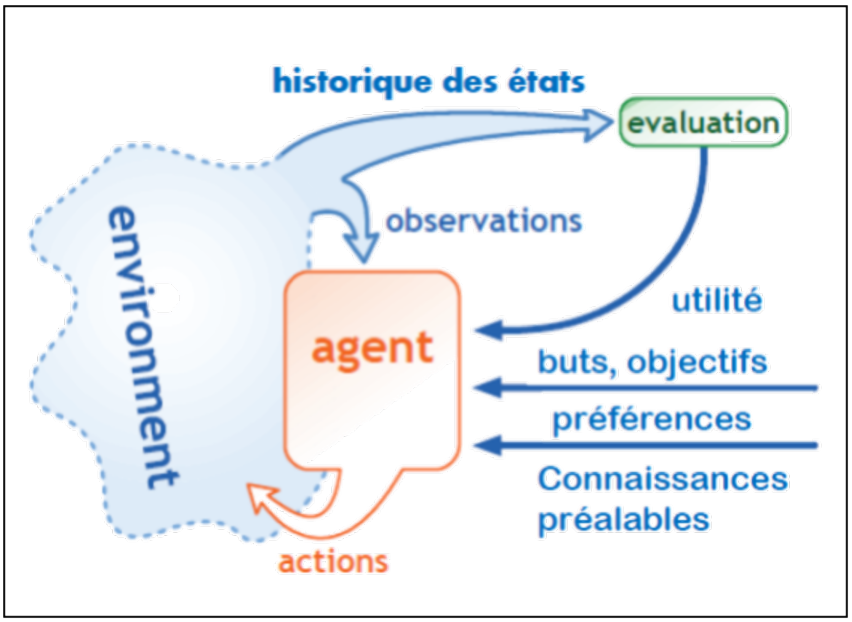


Figure 3.1: cycle de vie d'un agent.

3.3 Modèles d'Architectures d'Agents

En fonction de leur complexité et de leur capacité à gérer l'incertitude et la dynamique de l'environnement, les agents peuvent être classés selon quatre types de base, ordonnés par généralité croissante [12, 11]. Dans cette section, nous nous sommes basés sur le livre de [1]:

1. Agent simple réflexe (réactif)
2. Agent réflexe avec état interne
3. Agent basé sur les buts
4. Agent basé sur l'utilité

Tous ces types d'agents peuvent être augmentés pour devenir des **agents apprenants**, c'est-à-dire qu'ils peuvent améliorer leurs performances avec l'expérience. L'une des architectures les plus représentatives qui permet de bâtir un agent cognitif est l'architecture BDI (Belief, Desire, Intention) [13]. Dans ce qui suit, nous allons détailler chacun de ces modèles.

3.3.1 Agents Réflexes Simples (Agents Réactifs)

✗ **Principe de fonctionnement**: L'agent **simple réflexe** constitue l'architecture la **plus élémentaire**. Il fonctionne selon un **modèle stimulus-réponse direct**, en réagissant immédiatement à une perception en utilisant une table de règles conditionnelles $(Si perception = X, alors action = Y$.

✗ **Modèle** :

$$f(P) \rightarrow A$$

✗ **Caractéristiques** :

- ✓ Utilisation exclusive de règles conditionnelles (IF-THEN)
- ✓ Aucune mémoire ni représentation interne du monde
- ✓ Réaction immédiate basée uniquement sur le percept courant.

✗ **Avantages**:
- ✓ **Simplicité** : Faciles à concevoir et à implémenter.
- ✓ **Rapidité d'exécution** : Les décisions sont prises rapidement car elles ne nécessitent pas de calculs complexes ni de raisonnement approfondi.
- ✓ **Efficacité dans des environnements simples et statiques** : Fonctionnent bien lorsque la relation entre perception et action est directe et prévisible.

✗ **Limites**:
- ✓ **Manque de flexibilité** : Incapables de s'adapter à des environnements complexes ou dynamiques.
- ✓ **Problèmes dans des environnements partiellement observables** : Sans mémoire, ils ne peuvent pas distinguer des situations qui paraissent identiques mais qui nécessitent des actions différentes en fonction du passé.
- ✓ **Comportement non optimal** : Les règles peuvent ne pas toujours conduire au meilleur comportement à long terme.

✗ **Architecture** : Comme le montre la figure 3.2, l'architecture d'un agent réactif comprend :
- **Capteurs** : Pour percevoir l'environnement
- **Règles conditionnelles** : Base de règles ***IF-THEN***
- **Effecteurs** : Pour exécuter les actions

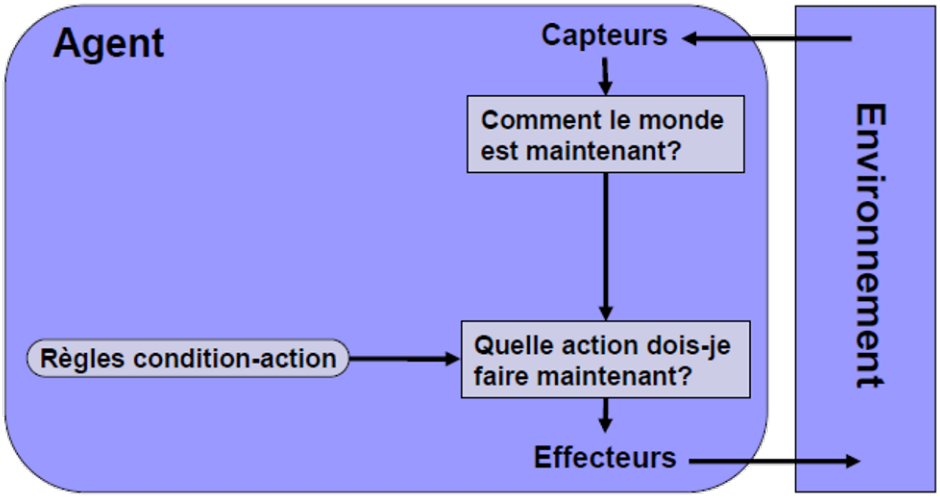


Figure 3.2: Fonction d'un Agent Réflexe Simple.

■ **Exemple 3.1 — Un thermostat qui ajuste la température :.**

- Si la température > 25°C → Allumer la climatisation
- Si la température < 18°C → Allumer le chauffage

La fonction d'un agent réflexe simple est décrite dans la figure 3.3.

Programme Agent

```
function SIMPLE-REFLEX-AGENT(percept) returns action
  static: rules, a set of condition-action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  return action
```

Figure 3.3: Architecture d'un Agent Réflexe Simple.

■ **Exemple 3.2 — Agent Aspirateur comme agent réactif (réflexe).**

```
function REFLEX-VACUUM-AGENT( [location,status]) returns an action
  if status = Dirty then return Suck
  else if location = A then return Right
  else if location = B then return Left
```

3.3.2 Agents Réflexes avec État Interne (Model-Based Reflex Agent)

Pour surmonter les limitations des agents purement réactifs, l'architecture avec état interne introduit une capacité de mémorisation et de maintien d'un modèle interne de l'environnement afin d'adapter leurs réactions.

✗ **Principe de fonctionnement**: Cette architecture fonctionne selon quatre principes clés :
- **Interprétation contextuelle** : Combinaison des nouvelles perceptions avec les connaissances passées
- **Maintien d'état** : Conservation d'une trace de l'état interne, incluant les parties non directement observables. Les actions sont choisies en fonction de l'état interne du monde.
- **Anticipation** : Capacité à prévoir l'évolution de l'environnement
- **Conscience de l'impact** : Prise en compte de l'effet des propres actions sur l'environnement.

✗ **Modèle** :

$$f(P,E) \rightarrow (A) \textit{ où E représente l'état interne}$$

✗ **Avantages**:
- ✓ **Meilleure performance dans des environnements partiellement observables** : Le modèle interne permet de compenser l'information manquante des perceptions directes.
- ✓ **Capacité à gérer des séquences d'actions** : Peut planifier des actions en fonction de l'évolution de l'état interne.
- ✓ **plus grande Adaptabilité** : Plus flexible que les agents réflexes simples.

✗ **Limites**:
- ✓ **Plus grande complexité** : La gestion et la mise à jour du modèle du monde ajoutent de la complexité.
- ✓ **Dépendance à la précision du modèle** : Si le modèle du monde est incorrect, les décisions peuvent être sous-optimales.

✓ **Coût computationnel** : La mise à jour du modèle peut être coûteuse en ressources.

✗ **Architecture** : Comme le montre la figure 3.4, l'architecture d'un agent réflexe avec état interne comprend :
- **Capteurs** : Pour percevoir l'environnement
- **Module de mise à jour de l'état interne** : Maintenance du modèle du monde
- **Règles conditionnelles** : Prise de décision basée sur l'état et les perceptions
- **Effecteurs** : Pour exécuter les actions

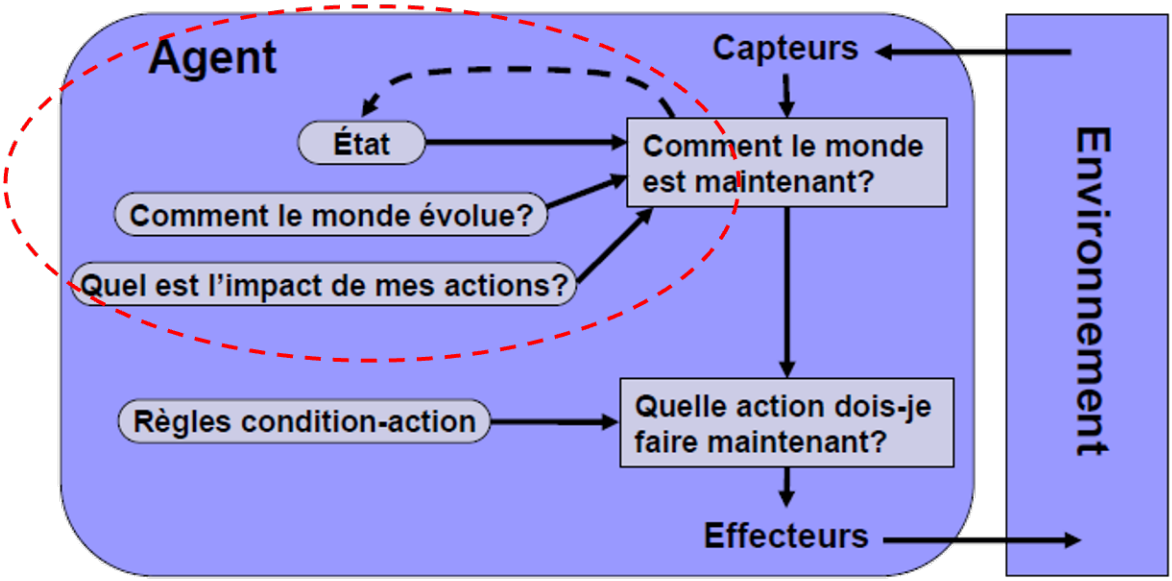


Figure 3.4: Architecture d'un Agent Réflexe avec État Interne.

Comme décrit dans la figure 3.5, un agent réflexe à état interne fonctionne en trouvant une règle dont la condition correspond à la situation courante (telle que définie par le percept et l'état interne mémorisé) et en faisant ensuite l'action associée à cette règle.

Programme Agent

```
function REFLEX-AGENT-WITH-STATE(percept) returns action
  static: state, a description of the current world state
          rules, a set of condition-action rules

  state ← UPDATE-STATE(state, percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  state ← UPDATE-STATE(state, action)
  return action
```

Figure 3.5: Fonction d'un Agent Réflexe avec État Interne.

■ **Exemple 3.3 — Robot aspirateur.**

✗ **Problème avec un agent réflexe simple :**
- Si un aspirateur fonctionne uniquement avec des règles conditionnelles (ex : Si sol sale → aspirer), il ne pourra pas retenir les endroits déjà nettoyés.
- Il risque de passer plusieurs fois au même endroit ou d'oublier de nettoyer certaines zones.

✗ **Solution avec un état interne :** L'aspirateur mémorise les endroits déjà nettoyés.

■ **Exemple 3.4 — Conducteur de voiture dans le brouillard.**

✗ **Problème avec un agent réflexe simple :**
- Un conducteur qui ne perçoit que la route devant lui et qui ne se souvient pas des conditions précédentes risque un accident en cas de virage serré ou de ralentissement soudain.

✗ **Solution avec un état interne :**
- Le conducteur se souvient de la route parcourue et ajuste sa vitesse en conséquence.
- Il peut aussi deviner la présence d'un virage en analysant la vitesse à laquelle la route semble disparaître dans le brouillard.

3.3.3 Agent basé sur les buts (Goal-based agent)

L'agent basé sur les buts introduit la notion d'objectif explicite et de planification. Il ne se contente plus de réagir, mais choisit ses actions en fonction de leur capacité à atteindre un but spécifique.

✗ **Principe de fonctionnement**: Les agents basés sur les buts utilisent un objectif pour guider leur prise de décision. Pour atteindre leur but, ils peuvent avoir besoin d'explorer différentes séquences d'actions et de raisonner sur les conséquences de ces actions:
- Buts (Goals) : Des états désirés que l'agent cherche à atteindre.
- **Planification** : L'agent doit planifier une séquence d'actions pour atteindre le but. Cela implique souvent la recherche dans l'espace des états.
- **Connaissance des actions** : L'agent doit connaître les effets de ses actions sur l'environnement.

✗ **Modèle** :

$$f(P,E,B) \rightarrow (A) \quad \textit{où B représente le but}$$

✗ **Avantages**:
- ✓ **Comportement orienté objectif** : Permet un comportement plus intelligent et moins "réactif" que les types précédents.
- ✓ **Capacité à résoudre des problèmes complexes** : Peut trouver des solutions à des problèmes qui nécessitent une planification à long terme.
- ✓ **Flexibilité** : Peut s'adapter à des changements dans l'environnement en recalculant son plan.

✗ **Limites**:
- ✓ **Complexité computationnelle élevée** : La planification peut être très coûteuse en temps et en ressources, surtout dans de grands espaces d'états.
- ✓ **Dépendance à la complétude et à la précision du modèle du monde** : Un plan ne sera bon que si le modèle de l'environnement est précis.

✓ **Gestion de l'incertitude** : La planification dans des environnements incertains est un défi majeur.

✗ **Architecture** : Comme le montre la figure 3.6, l'architecture d'un agent basé sur les buts comprend :
- **Capteurs** : Pour percevoir l'environnement
- **État interne** : Modèle du monde
- **Objectifs définis** : Spécification des buts à atteindre
- **Module de planification** : Génération de séquences d'actions
- **Effecteurs** : Exécution des actions planifiées. Devant plusieurs actions possibles, ce type d'agent choisit celle qui lui permet d'atteindre son but.

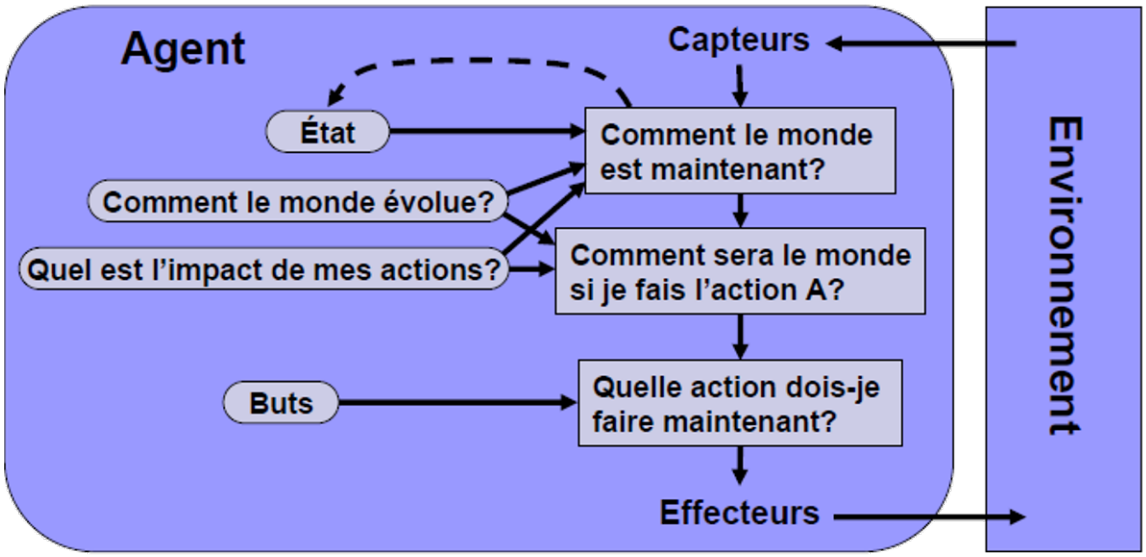


Figure 3.6: Architecture d'un Agent Basé sur les Buts.

▪ Exemple 3.5 — Taxi autonome.
À une intersection, un taxi peut tourner à gauche, à droite ou aller tout droit. Son choix dépend de sa destination. Un **agent basé sur les buts** utilise non seulement son état actuel mais aussi son objectif (ex. : atteindre la destination du passager) pour décider. Il planifie ses actions en fonction des résultats attendus :
- **Cas simple** : Une action suffit pour atteindre le but.
- **Cas complexe** : Une séquence d'actions (plusieurs virages) est nécessaire pour y parvenir.

L'**agent basé sur les buts présente une limitation majeure** : l'incapacité à quantifier et comparer différentes solutions lorsque plusieurs options permettent d'atteindre l'objectif comme le montre l'exemple de l'agent GPS suivant:

▪ Exemple 3.6 — un agent GPS.
Un agent de navigation doit choisir l'itinéraire pour atteindre une destination. Il dispose de plusieurs options :

1. **Route A** : Plus courte en distance mais avec beaucoup de feux rouges.
2. **Route B** : Plus rapide mais avec un péage.
3. **Route C** : Plus fluide mais plus longue.

Si l'agent est uniquement basé sur les buts, il sait qu'il doit arriver à destination mais ne peut pas quantifier laquelle de ces options est la meilleure. Il ne peut pas évaluer le compromis entre distance, coût et durée du trajet.
Solution: Un agent basé sur l'utilité serait plus efficace car il pourrait attribuer une valeur (score) à chaque option et choisir la plus avantageuse.

3.3.4 Agents Basés sur l'Utilité (Utility Based Agent)

L'agent basé sur l'utilité va au-delà de l'agent basé sur les buts en introduisant une mesure de satisfaction quantitative [11]. Il ne se contente pas d'atteindre un objectif, mais choisit la meilleure option pour le faire.

✗ **Principe de fonctionnement**: Les agents basés sur l'utilité sont les plus sophistiqués. Ils ne se contentent pas d'atteindre un but, ils cherchent à atteindre le but de la manière la plus optimale possible, en maximisant une fonction d'utilité.
- **Fonction d'utilité** : Mesure le degré de "***bonheur***" ou de satisfaction de l'agent dans un état donné. L'agent cherche à maximiser cette valeur.
- **Préférence** : Permet de choisir entre plusieurs états qui atteignent le but, mais avec des coûts ou des bénéfices différents.
- **Gestion de l'incertitude** : Particulièrement utiles dans des environnements incertains où différentes actions peuvent avoir des résultats probabilistes .

✗ **Modèle** :

$$f(P,E,B,U) \rightarrow (A) \textit{ où U représente l'utilité}$$

✗ **Avantages**:
- ✓ **Comportement optimal** : Permet à l'agent de prendre les meilleures décisions possibles, même en présence d'incertitude.
- ✓ **Gestion des compromis** : Peut évaluer les compromis entre différents objectifs ou risques. Il aide dans deux cas où les buts échouent:
  - Buts en conflit (ex: vitesse et sécurité)
  - Lorsqu'il y a plusieurs buts
- ✓ **Performance robuste** dans des environnements complexes et stochastiques.

✗ **Limites**:
- ✓ **Très haute complexité computationnelle** : Le calcul de l'utilité pour chaque action et chaque état peut être extrêmement coûteux.
- ✓ **Difficulté à définir la fonction d'utilité** : La définition d'une fonction d'utilité qui capture précisément les préférences de l'agent est un défi majeur.
- ✓ **Dépendance à la modélisation probabiliste** de l'environnement.

✗ **Architecture** : Comme le montre la figure 3.7, l'architecture d'un agent basé sur l'utilité comprend :
- **Capteurs** : perçoivent l'environnement (ex. : position, obstacles, trafic, etc.)
- **Fonction de perception** : transforme les perceptions en représentations internes.

- **Modèle du monde (état interne)** : garde en mémoire ce que l'agent sait de l'environnement (surtout les parties non visibles).
- **Ensemble de buts** : définit les états finaux souhaitables (ex. : arriver à destination, maximiser les ressources, etc.)
- **Fonction d'utilité** : attribue une **valeur numérique** à chaque état possible du monde. Plus la valeur est élevée, plus l'état est **préférable**.
- **Module de recherche/planification** : → génère les actions possibles, puis choisit celle qui **maximise l'utilité attendue**.
- Effecteurs : Exécutent l'action optimale choisie.

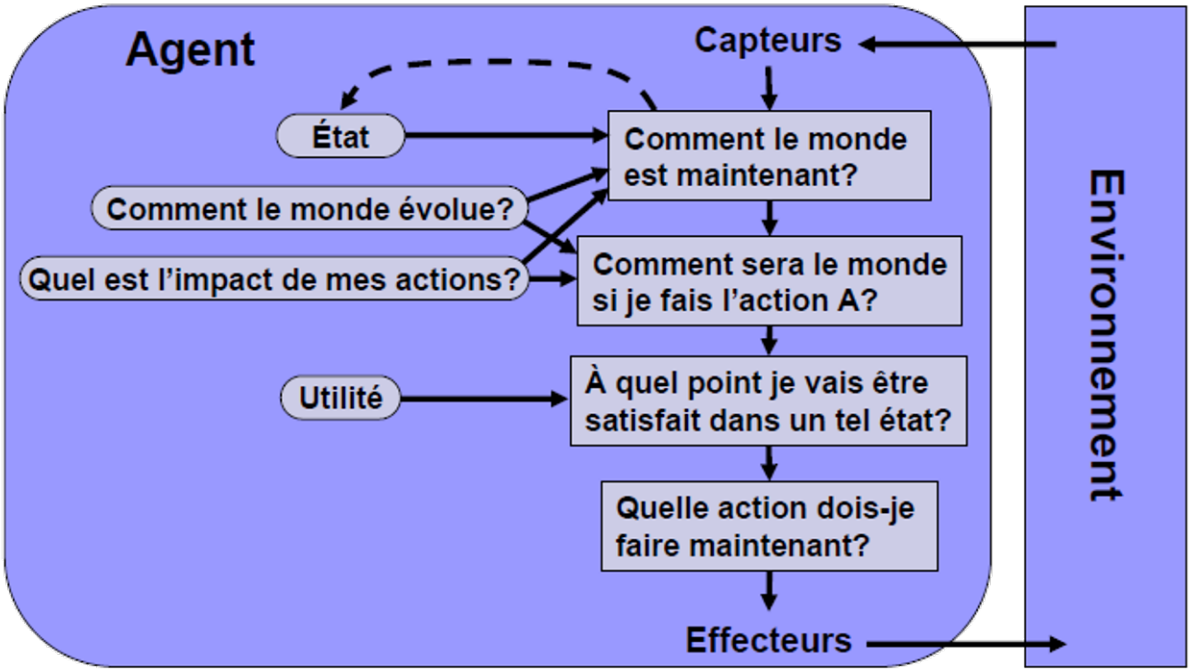


Figure 3.7: Architecture d'un Agent Basé sur l'Utilité.

R La différence clé entre un agent basé sur les buts et un agent basé sur l'utilité:

✓ **Agent basé sur les buts** : "Est-ce que cette action me mène à l'objectif ? Oui/Non.»

✓ **Agent basé sur l'utilité** : "Parmi toutes les actions qui mènent à l'objectif, laquelle maximise ma satisfaction?"

▪ Exemple 3.7 — un agent GPS orienté utilité.

Un agent de navigation doit choisir l'itinéraire pour atteindre une destination. Il dispose de plusieurs options :

1. **Route A** : 10 km, sans péage, mais 25 min à cause du trafic.
2. **Route B** : 12 km, fluide, mais avec un péage de 3€.
3. **Route C** : 15 km, rapide mais traverse une zone en travaux.

Si l'agent est **basé uniquement sur les buts**, il choisira n'importe laquelle de ces routes si elles

mènent toutes à la destination.
Un agent **basé sur l'utilité** va évaluer chaque itinéraire selon des critères comme Durée du trajet, Coût, Confort ou sécurité, Préférences de l'utilisateur (éviter les péages, arriver vite, etc.).
-> Il affecte à chaque option une valeur d'utilité (ex. : une note sur 10) et choisit celle qui maximise cette valeur.
✗ **Exemple de calcul simplifié** :
- **Route A** : 6/10 (gratuite mais lente)
- **Route B** : 8/10 (rapide + fluide mais payante)
- **Route C** : 5/10 (longue + zone à risque)

✗ **Décision finale** : ***Route B, car elle maximise l'utilité globale.***

■ **Exemple 3.8 — Un agent pour un système de gestion de portefeuille financier.**
- **Perceptions** : Cours des actions, taux d'intérêt, indicateurs économiques.
- **Modèle du monde** : Prévisions des marchés, volatilité des actifs.
- **But** : Maximiser le rendement du portefeuille.
- **Fonction d'utilité** : Évalue le rendement attendu et le risque associé à chaque investissement. Un état est jugé plus utile si le rendement est élevé et le risque faible.
- **Décision** : L'agent choisit les investissements qui maximisent son utilité, en tenant compte des probabilités de gain et de perte.

3.3.5 Agents Apprenants (Learning Agent)

L'agent apprenant représente une évolution majeure en introduisant la capacité d'améliorer ses performances au fil du temps [12]. Plutôt que d'être programmé avec toutes les connaissances nécessaires, il découvre progressivement la meilleure manière d'agir.

✗ **Modèle** :

$$f(P,E,B,U,L) \rightarrow A \textit{ où :}$$

- **P** = Perception actuelle (input sensoriel)
- **E** = Expérience passée
- **B** = Base de connaissances ou croyances (Beliefs)
- **U** = Fonction d'utilité (préférences ou valeur des états)
- **L** = Module d'apprentissage
- **A** = Action choisie

✗ **Avantages**:
- ✓ **Flexibilité** : Il permet de représenter aussi bien les agents simples que complexes.
- ✓ **Adaptabilité et Robustesse** : Les agents apprenants sont particulièrement efficaces dans des environnements dynamiques et inconnus. Ils peuvent découvrir de nouvelles stratégies et s'adapter aux évolutions imprévues.
  Un agent apprenant peut apprendre à gérer des données bruitées, des perceptions incomplètes ou des défaillances partielles de capteurs, améliorant ainsi sa robustesse.
- ✓ Décision intelligente : L'agent ne réagit pas seulement à la situation actuelle, il **raisonne**, **apprend** et **optimise**.

- ✓ **Découverte de Stratégies Optimales** : Les agents apprenants peuvent découvrir des stratégies qui surpassent celles conçues manuellement, atteignant des niveaux de performance élevés, parfois surhumains (ex: AlphaGo au jeu de Go).
- ✓ **Gestion de la Complexité**: Grâce à l'apprentissage automatique, les agents peuvent gérer des espaces d'états et d'actions très vastes et complexes.

✗ **Limites**:

- ✓ **Exigences en Données et en Expérience** : L'apprentissage, tel que le deep learning ou le renforcement, nécessite une quantité massive de données ou d'interactions avec l'environnement pour généraliser correctement. Acquérir ces données peut être coûteux ou difficile.
- ✓ **Coût Computationnel Élevé** : Le temps et les ressources d'entraînement requises peuvent être élevées.
- ✓ **Exploration risquée** : En phase d'apprentissage, il peut faire des erreurs coûteuses.

✗ **Architecture** : Comme le montre la figure 3.8, l'architecture modulaire d'un agent apprenant comprend :

- **Module d'apprentissage (Learning Element)**: Responsable de l'amélioration des performances de l'agent en modifiant le programme de performance. Il prend les retours du critique et les traduit en ajustements.
- **Module de performance (Performance Element)** : C'est le programme qui sélectionne les actions. Il se base sur les connaissances et procédures pour choisir les actions. Il prend les décisions et agit sur l'environnement.
- **Module de critique (Critic Element)** : Il fournit un feedback à l'élément d'apprentissage sur la qualité des actions de l'agent. Il observe les résultats réels et compare avec ce qui était attendu ou désiré.
- **Générateur de problèmes (Problem Generator)** : Suggère des actions exploratoires qui pourraient conduire à de nouvelles expériences et à une meilleure compréhension de l'environnement, même si ces actions ne sont pas optimales à court terme.

■ Exemple 3.9 — Robot d'arrosage intelligent.

- ✓ **P** = Il perçoit l'humidité du sol et la météo actuelle.
- ✓ **E** = Il a mémorisé qu'en fin d'après-midi, l'arrosage est plus efficace.
- ✓ **B** = Il croit que certaines plantes nécessitent plus d'eau que d'autres.
- ✓ **U** = Il vise à maximiser la santé des plantes tout en minimisant la consommation d'eau.
- ✓ **L** = Il apprend, à travers le temps, les meilleures heures d'arrosage et les préférences des plantes selon les saisons.
- ✓ **A** = Il décide s'il doit arroser maintenant, attendre, ou ne pas arroser.

3.3.6 Agent intentionnel BDI (Belief-Desire-Intention)

L'architecture BDI (Belief-Desire-Intention) s'inspire de la cognition humaine pour simuler la façon dont un agent intelligent **raisonne**, **décide** et **agit** en fonction de ce qu'il **croit**, **veut**, et **prévoit** de faire [13, 11]. Elle repose sur trois concepts fondamentaux [13, 14]:

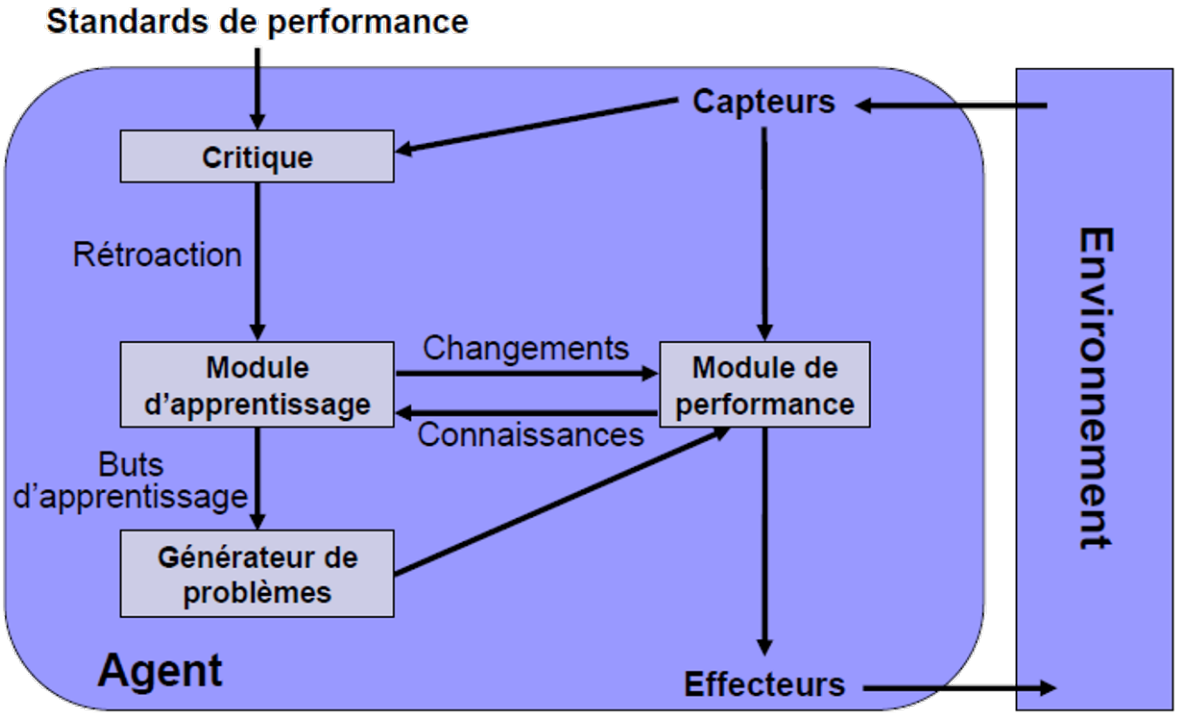


Figure 3.8: Architecture d'un Agent Apprenant.

- ✓ **Croyances (Beliefs)** : Représentation de l'état du monde selon l'agent. Ces croyances peuvent être incorrectes, incomplètes ou incertaines.
- ✓ **Désirs (Desires)** : Objectifs possibles que l'agent aimerait atteindre. Un agent peut avoir des désirs contradictoires, auquel cas il doit choisir un sous-ensemble cohérent. Ce sous-ensemble correspond aux buts de l'agent.
- ✓ **Intentions (Intentions)** : Objectifs choisis et plans d'actions que l'agent s'engage à poursuivre. Ce sont les actions que l'agent a décidé d'accomplir pour atteindre ses buts.

La figure 3.9 illustre l'architecture détaillée d'un agent BDI.

3.3.6.1 Fonctions Principales

L'architecture BDI s'articule autour de plusieurs fonctions clés:

- ✓ **Révision des croyances** : $revc : B \times P \rightarrow B$ est la fonction de **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; elle est réalisée par la composante **Révision des croyances**;

- ✓ **Génération d'options** : $options : D \times I \rightarrow I$ est la fonction qui représente le **processus de décision** de l'agent prenant en compte ses désirs et ses intentions courantes; Ces **options** représentent ses **désirs** possibles conformément à ses intentions. cette fonction est réalisée par la composante **Processus de décision**;

- ✓ **Gestion des désirs** : $des : B \times D \times I \rightarrow D$ est la fonction qui peut **changer les désirs**

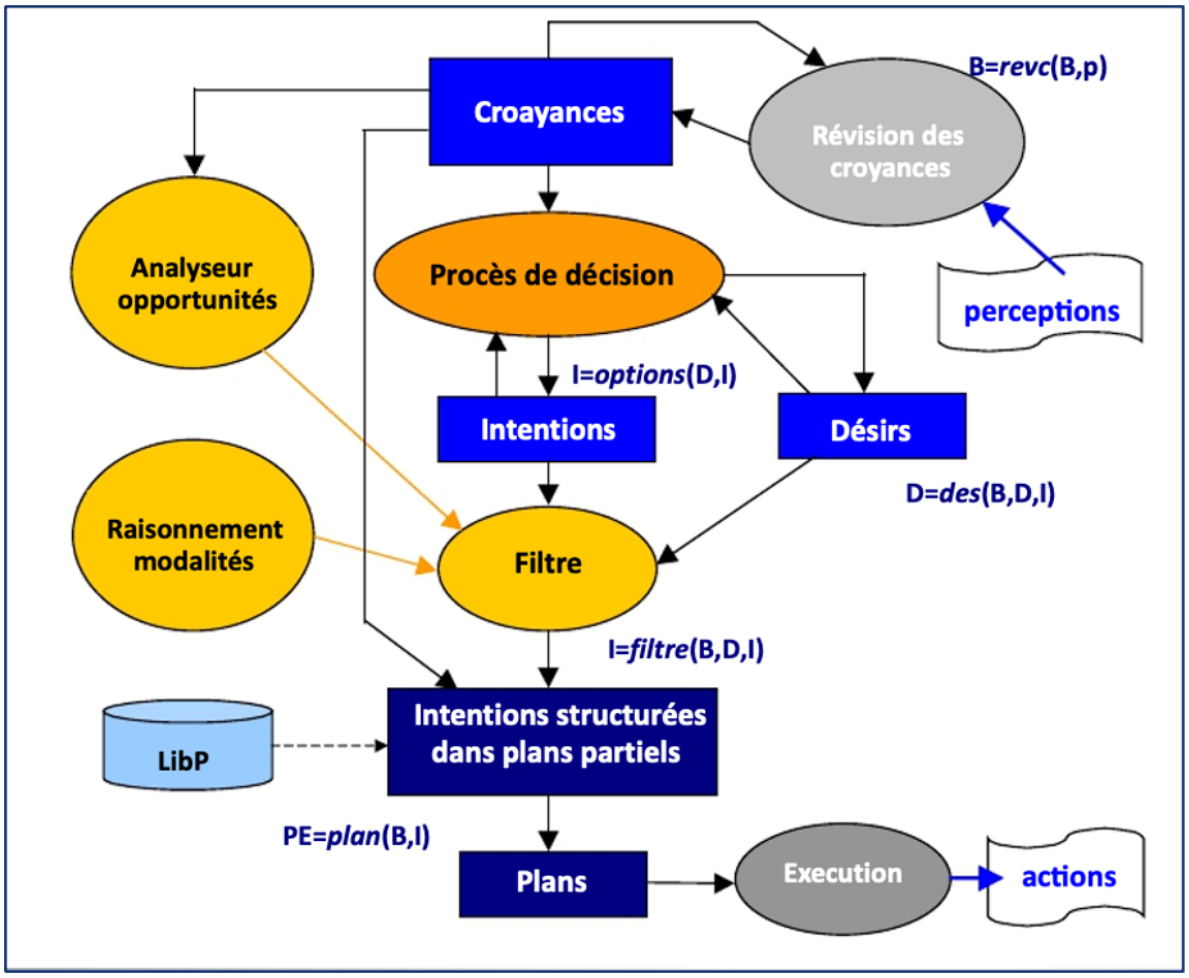


Figure 3.9: Architecture d'un agent BDI.

d'un agent si ses croyances ou intentions changent, pour maintenir la **consistance des désirs** de l'agent (on suppose dans notre modèle que l'agent a toujours des désirs consistants); cette fonction est également réalisée par la composante **Processus de décision;**

✓ **Filtrage des intentions** : $filtre : B \times D \times I \rightarrow I$ est la fonction la **plus importante** car elle décide des intentions à poursuivre; Elle active ses **nouvelles intentions** en fonction de ses croyances, options, et intentions courantes. Elle élimine les intentions devenues irréalistes ou incohérentes. Elle est réalisée par la **composante Filtre**;

✓ **Planification** : $plan : B \times I \rightarrow PE$ est la fonction qui transforme les **plans partiels** en plans exécutables, PE étant l'ensemble de ces plans; elle peut utiliser, par exemple, une bibliothèque de plans, représentée par le **module LibP** dans la figure 3.9. Un plan est une séquence d'actions à exécuter dans le temps;

✓ **Exécution**: est la fonction de sélection de l'**action à exécuter**, cette fonction renvoie une **intention exécutable** qui correspond à **une action**.

3.3.6.2 Cycle de Contrôle

L'algorithme de contrôle d'un agent BDI suit un cycle itératif :
Une version plus détaillée est décrite par l'algorithme BDI suivant:

Algorithm 1 Cycle de perception-décision-action d'un agent BDI

1: Initialiser B_0, D_0, I_0
2: **repeat**
3: Obtenir nouvelles perceptions p
4: $B \leftarrow \text{revc}(B, p)$
5: $I \leftarrow \text{options}(D, I)$
6: $D \leftarrow \text{des}(B, D, I)$
7: $I \leftarrow \text{filtre}(B, D, I)$
8: $PE \leftarrow \text{plan}(B, I)$
9: Exécuter(PE)
10: **until** condition d'arrêt

Algorithm 2 Cycle BDI (Belief-Desire-Intention)

Variables: b: beliefs, g: desires, i: intentions, eq: event queue

1: $(b, g, i) :=$ initialize() **Repeat**
2: $options :=$ option_generator(eq, b, g, i)
3: $selected :=$ deliberate$(options, b, g, i)$
4: $i := selected \cup i$
5: execute(i)
6: $eq :=$ see()
7: $b :=$ update_beliefs(b, eq)
8: $(g, i) :=$ drop_successful_attitudes(b, g, i)
9: $(g, i) :=$ drop_impossible_attitudes(b, g, i)
10: **forever**

■ **Exemple 3.10 — L'agent assistant personnel (type Google Assistant).**

1. Perception

Le robot capte via ses capteurs :

- Il est 19h00;
- Le sol est sale dans le salon;
- La batterie est à 25%;
- Il entend la voix de l'utilisateur dire : « *J'ai des invités qui arrivent bientôt.* »

2. Mise à jour des croyances (Beliefs) :

- **B1** : Le salon est sale.
- **B2** : Il est tard.
- **B3** : La batterie est faible.
- **B4** : L'utilisateur attend des invités → **la propreté est prioritaire.**

3. Génération des désirs (Desires) :
À partir de ces croyances, l'agent identifie plusieurs **objectifs possibles** :

- **D1** : Nettoyer le salon.
- **D2** : Recharger la batterie.
- **D3** : Accueillir les invités (musique d'ambiance, lumière...).
- **D4** : Fermer les volets automatiquement.

4. Filtrage des intentions (Intentions) : L'agent filtre les désirs en tenant compte des priorités, de l'état de la batterie et du contexte :

Le sol sale + les invités → **D1 devient prioritaire.**
La batterie à 25% → suffisant pour un petit nettoyage ⇒ pas besoin immédiat de D2.
D3 et D4 peuvent attendre.

Donc, l'intention retenue est : Nettoyer rapidement le salon.

5. Planification et exécution :

- Le robot choisit un plan court : nettoyage partiel du salon.
- Il décide ensuite d'aller se recharger.
- Enfin, il pourra allumer la musique douce à 19h45.

3.3.6.3 Avantages de l'Architecture BDI

On distigue plusieurs avantages, notamment [13, 12]:

- ✓ **Flexibilité** : Adaptation dynamique aux changements d'environnement
- ✓ **Transparence** : Structure claire et compréhensible du raisonnement
- ✓ **Robustesse** : Gestion cohérente des conflits et des priorités
- ✓ **Extensibilité** : Facilité d'ajout de nouvelles capacités.

3.3.7 Comparaison des Architectures

Le tableau 3.1 établit une comparaison entre les différentes architectures des agents en termes de mémoire et du processus de prise de décision.

Type d'Agent	Mémoire	Prise de décision	Exemple
Réactif	✗ Non	Basée sur des règles fixes	Aspirateur basique
Réflexe avec état interne	✓ Oui	Adaptation basique	Feux de circulation intelligents
Basé sur les buts	✓ Oui	Planification d'actions	GPS intelligent
Basé sur l'utilité	✓ Oui	Optimisation des actions	Système de recommandation
BDI	✓ Oui	Raisonnement et adaptation autonome	Robot assistant intelligent

Table 3.1: Comparaison des architectures d'agents selon leurs capacités

3.3.8 Agents Mobiles

La mobilité d'un agent correspond à sa capacité à se déplacer d'un nœud à un autre pour exécuter une tâche. L'agent "migre" vers un nouvel environnement pour continuer son exécution.
Exemple: Agents de surveillance migrant entre capteurs IoT pour optimiser la couverture.
Les agents mobiles sont utilisés dans divers domaines d'application tels que le commerce électronique, la recherche dans les bases de données, la gestion de réseau, le calcul mobile, etc.

3.3.8.1 Caractéristiques

- ✓ Les **agents mobiles** sont des entités logicielles qui peuvent se déplacer dans le réseau de leur propre **initiative**; ils se déplacent d'une machine à une autre et communiquent avec d'autres agents ou accèdent aux ressources du serveur.
- ✓ Ils permettent de réaliser une **meilleure exploitation** de ressources: En transférant des applications du client au serveur et en exécutant des appels locaux de procédure au lieu des appels extérieurs, le trafic réseau est réduit.
- ✓ La technologie d'agents mobiles permet de fournir une exécution asynchrone de tâche. Ainsi, la dépendance entre les clients et les applications de serveur peut être réduite et un traitement automatique de tâche est présenté.
- ✓ Les services ne sont plus liés à un certain environnement. Ils peuvent être dynamiquement installés et employés dans l'endroit exact où ils sont exigés.

■ Exemple 3.11 — Assistant virtuel personnel : .

- Un agent assistant (ex : Siri, Google Assistant) commence une tâche sur votre smartphone (ex : planifier un voyage).
- Il se déplace vers votre ordinateur portable pour continuer la tâche (ex : réserver un billet d'avion sur un écran plus grand).
- Avantage : Continuité transparente de l'expérience utilisateur.

3.3.8.2 Types de Mobilité d'un Agent

On distingue trois types de mobilité :

- ✗ **La mobilité du code** :
  La mobilité du **code** permet à un agent de **transférer son code exécutable** vers un autre nœud (machine, serveur, appareil) pour y exécuter une tâche spécifique. Le code est déplacé, mais pas l'état ou les

données de l'agent.

■ **Exemple 3.12 — Agent de collecte de données distribué :.**
- Un agent est programmé pour analyser la vulnérabilité des serveurs distants.
- Il envoie son code (ex : un script Python) vers un serveur A pour collecter des données.
- Le code s'exécute localement sur le serveur, analyse les fichiers, et renvoie uniquement les résultats pertinents (ex : liste des failles). Une fois terminé, il envoie le même code vers un serveur B.

✓**Avantage**:
- Réduit la charge réseau en évitant de transférer des données brutes.

✓**Application**:
- Crawlers web qui se déplacent sur différents sites pour indexer des pages.
- Mise à jour de logiciels à distance (ex : correction de bugs déployée via un agent mobile).

✗ **La mobilité des données** :
La mobilité des données consiste à transférer les données utilisées ou générées par un agent (fichiers, bases de données, capteurs), sans déplacer le code ou l'état de l'agent.

■ **Exemple 3.13 — Un agent météorologique collectant des données..**
- L'agent est installé sur un serveur central.
- Il récupère des données en temps réel depuis des capteurs distants (ex : température, humidité).
- Les données des capteurs sont transférées vers le serveur central pour analyse.
- L'agent reste sur le serveur central : seul le flux de données est mobile.

✓**Avantage**:
- Centralisation des données pour un traitement global.
- Pas besoin de déplacer le code ou l'état de l'agent.

✓**Application**:
- Surveillance IoT (ex : données de capteurs envoyées vers le cloud).
- Systèmes de stockage distribué (ex : Dropbox, Google Drive).

✗ **La mobilité de l'état (ou du contexte)** :
La mobilité de l'état implique le transfert de l'état courant d'un agent (sa mémoire, ses variables, son contexte d'exécution) d'un hôte à un autre, sans déplacer le code. Le code reste exécuté sur l'hôte d'origine, mais l'état est partagé.

■ **Exemple 3.14**
- Un PNJ (Personnage Non-Joueur) dans un jeu en ligne migre entre différents serveurs de jeu. Son état (position GPS, inventaire, interactions passées) est transféré avec lui pour assurer la continuité de son comportement.
- Un drone transfère son état (position GPS, batterie restante) à un autre drone pour prendre le relais dans une mission.

✓**Avantage**:
- Permet la continuité des tâches et l'équilibrage de charge entre serveurs,
- améliore la résilience et la scalabilité des systèmes.

✓**Application**:
- Systèmes de sauvegarde en temps réel (ex : reprise après sinistre).
- Réplication d'états dans les bases de données distribuées.
- Reprise d'une session de travail sur un autre appareil (ex : passage du smartphone à l'ordinateur).

Le tableau 3.2 présente une comparaison entre les trois types de mobilité.

Type de Mobilité	Élément Déplacé	Exemple	Avantage Clé
Code	Programme exécutable	Script de scan envoyé à un serveur distant.	Exécution locale, économie de bande passante.
État	Contexte (mémoire, variables)	PNJ migrant entre serveurs de jeu.	Continuité et équilibrage de charge.
Données	Informations brutes ou traitées	Capteurs envoyant des données météo au cloud.	Centralisation et analyse globale.

Table 3.2: Comparaison des types de mobilité des agents

3.3.8.3 Avantages de la Mobilité

1. **Optimisation des ressources**:
   - La mobilité du code évite de saturer le réseau avec des données brutes. C'est une exécution décentralisée.
   - La mobilité de l'état permet de répartir la charge entre serveurs: Continuité et équilibrage.
   - La mobilité des données centralise l'analyse. Centralisation et analyse.
2. **Adaptabilité** :
   Les agents peuvent s'adapter à des environnements dynamiques (ex : drones en mission).
3. **Résilience**:
   En cas de panne, l'état ou le code peut être migré vers un nœud fonctionnel.

3.4 Conclusion

Les modèles et architectures d'agents constituent le cœur de la conception des systèmes intelligents distribués [11, 12]. À travers les différentes approches étudiées , des plus simples aux plus sophistiqués, nous avons constaté que le choix d'une architecture dépend directement du niveau d'autonomie, de rationalité et de complexité de l'environnement dans lequel évolue l'agent [13].
Les architectures simples, basées sur des règles, conviennent aux environnements bien structurés et fortement déterministes, tandis que les architectures délibératives ou hybrides permettent une prise de décision plus souple, fondée sur la représentation interne du monde et des objectifs [14].
Ces modèles fournissent ainsi les bases nécessaires pour comprendre et concevoir des systèmes multi-agents, où plusieurs entités coopèrent, négocient ou apprennent ensemble pour atteindre des buts collectifs.

3.5 Exercices

Exercice 3.1 — Identifier et modéliser l'architecture d'un agent.

Contexte :

Vous êtes en charge de concevoir un **agent logiciel** pour un **réfrigérateur intelligent**. Celui-ci doit être capable de :

- Détecter quand un produit est **manquant** ou **périmé**.
- Décider s'il faut le **recommander**, et à quel moment.
- **Suggérer des recettes** en fonction des ingrédients présents.
- Tenir compte des **préférences alimentaires** des utilisateurs.
- **Apprendre** progressivement les habitudes alimentaires (ex : jours de courses, plats favoris, etc.).

Questions :

1. Quel **type d'architecture d'agent** conviendrait le mieux à ce système ?
2. Pour chaque fonctionnalité, **associez une architecture d'agent** adaptée et **justifiez** votre choix.
3. Que gagnerait-on à intégrer une **architecture hybride** comme **BDI** dans ce cas ?
4. Décrivez un **plan possible de l'agent** à l'aide du modèle BDI :
   (*Perceptions* $\rightarrow$ *Croyances* $\rightarrow$ *Désirs* $\rightarrow$ *Intentions* $\rightarrow$ *Plan*).
5. **Bonus** : Quels **défis** peut poser un tel agent dans un environnement réel ?

Exercice 3.2 Soit un système multi-agents de surveillance environnementale, Décrire les trois types de mobilite dans ce contexte?

✓ **Solutiom**

1. **Mobilité du Code**:
   - Un agent envoie son algorithme d'analyse de la qualité de l'air à un drone pour qu'il l'exécute en vol.
2. **Mobilité de l'État**:
   - Le drone transfère son état (position GPS, batterie restante) à un autre drone pour prendre le relais.
3. **Mobilité des Données**:
   - Les données collectées (taux de CO_2, particules fines) sont envoyées à un serveur central pour générer des cartes de pollution.

4. Systèmes Multi-Agents (SMA)

4.1 Introduction

L'évolution de l'Intelligence Artificielle nous a menés d'une approche centrée sur l'agent individuel vers des systèmes où plusieurs agents intelligents collaborent pour résoudre des problèmes complexes. Cette transition de l'Intelligence Artificielle (I.A.) vers l'Intelligence Artificielle Distribuée (I.A.D.), illustrée par la figure 4.1, représente un paradigme fondamental où l'interaction devient le cœur de la problématique des logiciels et systèmes complexes.

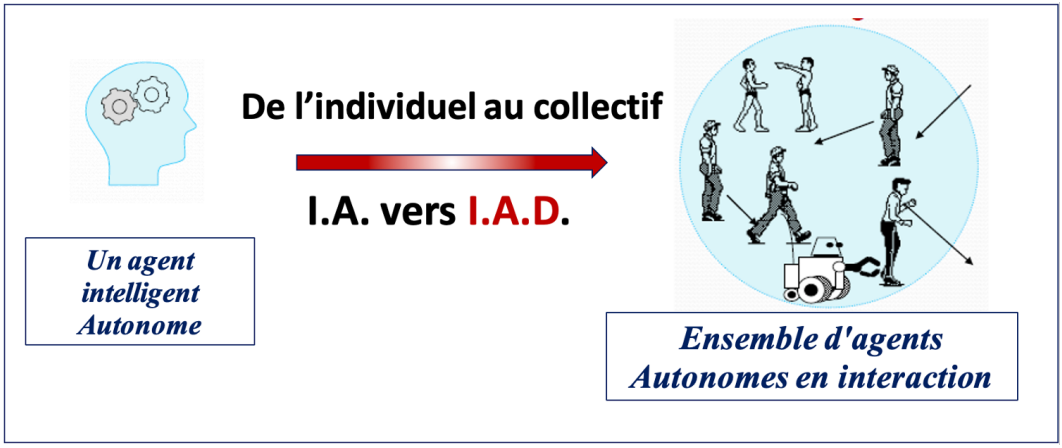


Figure 4.1: De l'Agent vers le Multi-Agents.

4.2 Fondements des Systèmes Multi-Agents

Avec l'émergence des agents intelligents autonomes, une question naturelle se pose :

Plusieurs agents peuvent-ils coopérer pour résoudre des problèmes complexes, mieux qu'un agent isolé ?

C'est précisément l'idée fondatrice des Systèmes Multi-Agents (SMA), où différents programmes d'IA collaborent comme une équipe coordonnée [16, 17]. Les SMA offrent ainsi une approche puissante pour aborder la complexité des systèmes modernes en décomposant un problème global en sous-problèmes gérables par des agents spécialisés [16, 10, 17]. Cette idée peut être illustrée par l'exemple suivant :

▪ Exemple 4.1 — Service clientèle d'une entreprise technologique.
Différents programmes d'intelligence artificielle (IA) spécialisés travaillent ensemble pour résoudre les problèmes des clients :

- ✓ L'un d'entre eux prend en charge la conversation initiale et achemine les demandes ;
- ✓ Un autre fouille dans la documentation technique pour trouver des solutions ;
- ✓ Un troisième élabore des réponses personnalisées pour chaque client.

En travaillant ensemble, ces agents peuvent s'attaquer à des problèmes qui dépasseraient largement les capacités d'un seul agent isolé, offrant une solution plus robuste, plus flexible et plus efficace.

4.2.1 Définition et Concepts Clés

Un Système Multi-Agent (SMA) est une organisation d'agents autonomes capables de **communiquer**, d'**interagir** et de **coopérer** pour résoudre collectivement un problème afin d'atteindre leurs buts (voir figure 4.2) [15, 6, 7, 17].
Les interactions dans un système multi-agents permettent d'exploiter l'intelligence collective des

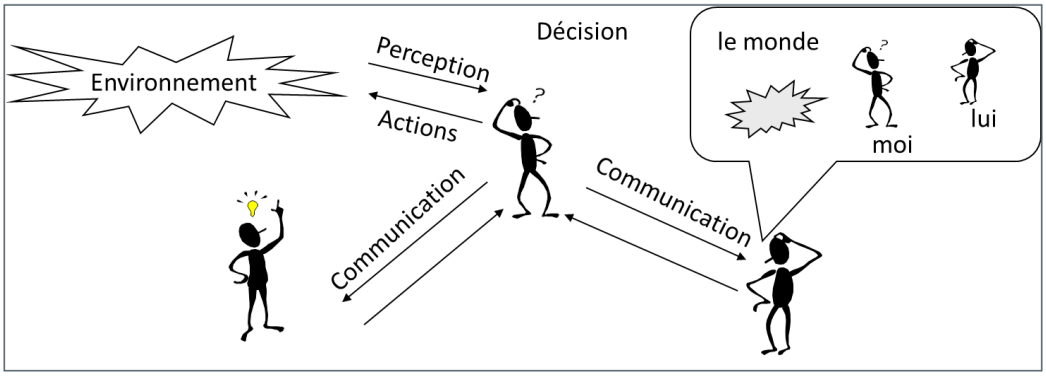


Figure 4.2: Système Multi-Agents.

agents pour résoudre des problèmes complexes.
Chaque agent, situé dans un environnement, n'agit que sur une zone d'influence limitée, comme le montre la figure 4.3. Formellement, un SMA est un système composé des éléments suivants :

- **Un Environnement E** : l'espace partagé où évoluent les agents;
- **Un ensemble d'objets O : les éléments manipulables et leurs interconnexions. Ces objets sont situés, c'est-à-dire que, pour tout objet, il est possible, à un moment donné, d'associer**

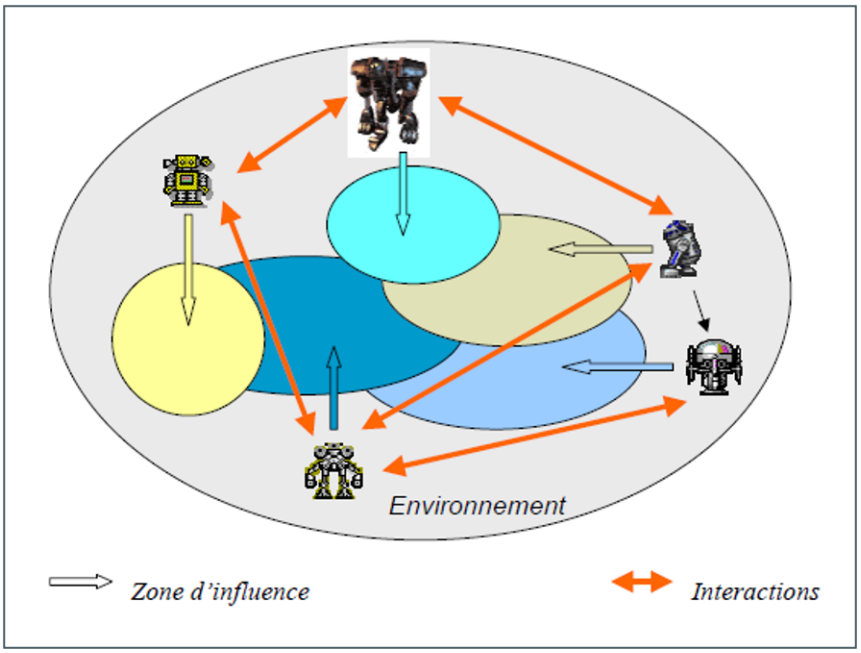


Figure 4.3: Zones d'influence des agents d'un SMA.

une position dans E. Ces objets sont passifs, c'est-à-dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents ;

- **Des Relations R** : qui unissent des objets (et donc des agents) entre eux ;
- **Un ensemble d'agents A (où $A \subset O$)** : les entités intelligentes du système
- Un ensemble d'opérations Op : permettant aux agents de percevoir, produire, consommer, transformer et manipuler les objets.

Les SMA permettent de réduire la complexité de la résolution d'un problème en le subdivisant en sous-ensembles, puis en associant un agent intelligent indépendant à chacun de ces sous-ensembles et en coordonnant l'activité de ces agents. On parle ainsi d'Intelligence Artificielle Distribuée (IAD). L'intérêt que suscitent les SMA est lié à leur capacité d'aborder les problèmes complexes d'une manière distribuée et de proposer des solutions réactives et robustes.
Les recherches dans les SMA se focalisent sur l'étude des comportements collectifs et sur la répartition de l'intelligence sur des agents plus ou moins autonomes, capables de s'organiser et d'interagir pour résoudre des problèmes, ce qui répond aux exigences des systèmes actuels qui sont de fait distribués et de plus en plus ouverts. Un système multi-agents peut ainsi être vu comme « *un réseau d'agents faiblement interconnectés qui travaillent ensemble dans le but de résoudre des problèmes qui ne peuvent pas être résolus individuellement par les agents* »

4.2.2 Architecture Générale

L'architecture d'un SMA se caractérise par trois composants principaux [16, 17] :

- **Agent** : entité autonome dotée de capacités de perception et d'action;
- **Environnement** : espace partagé d'interaction;
- **Organisation** : structure définissant les relations et interactions entre agents.

Comme illustré par la figure 4.4, cette architecture suit le modèle $MIND + BODY$, où chaque agent possède à la fois des capacités cognitives (*MIND*) et des moyens d'action dans l'environnement (*BODY*).

L'interaction est le cœur des Systèmes Multi-agents. Les agents autonomes dans un SMA doivent

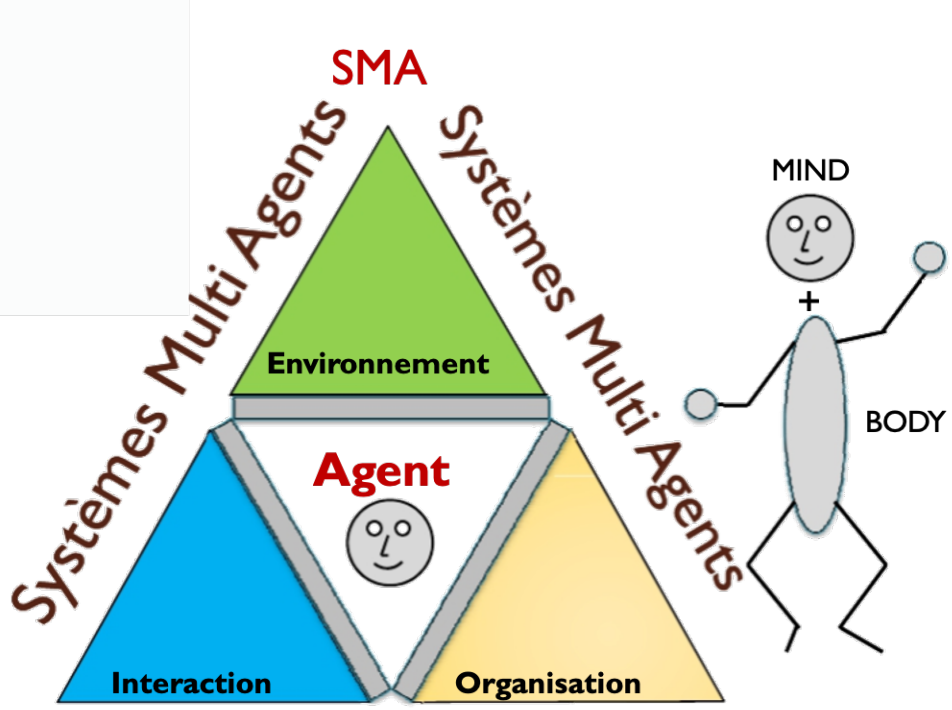


Figure 4.4: Architecture d'un SMA.

interagir pour :

- ✓ **Coopérer** : partager des objectifs communs et coordonner leurs actions
- ✓ **Collaborer** : se répartir les tâches de manière optimale;
- ✓ **Négocier** : résoudre les conflits d'intérêts ou de ressources;
- ✓ **Se coordonner** : synchroniser leurs activités dans le temps et l'espace.

4.2.3 Caractéristiques Distinctives des SMA

Les SMA se distinguent par les propriétés suivantes :

- **Autonomie** : Chaque agent prend ses propres décisions basées sur ses perceptions locales et ses objectifs propres.
- **Distribution** : Absence de contrôle centralisé, chaque agent agit selon sa logique interne tout en tenant compte des autres agents.
- **Hétérogénéité** : Les agents peuvent avoir des architectures, des rôles et des objectifs différents, permettant une spécialisation des fonctions.
- **Interaction** : Communication, collaboration ou compétition entre agents selon les besoins du système.
- **Comportement émergent** : Le comportement global du système résulte des interactions locales entre agents, créant des propriétés que ne possède aucun agent individuellement.

Le tableau 4.2.3 présente une comparaison entre un système mono-agent et un système multi-agents:

Aspect	Agent Isolé	Système Multi-Agents
Complexité des tâches	Gère des problèmes généralement moins complexes ou centralisés	Gère des problèmes complexes, distribués, dynamiques et souvent incertains
Coordination	Pas de coordination nécessaire	Coordination et communication critiques
Modularité	Souvent monolithique ou avec une décomposition fonctionnelle rigide	Hautement modulaire, avec des agents autonomes et spécialisés
Robustesse	La défaillance de l'agent peut entraîner la défaillance de tout le système	Plus robuste aux pannes partielles ; la défaillance d'un agent n'arrête pas nécessairement tout le système
Flexibilité	Moins adaptable aux changements de l'environnement ou aux nouvelles tâches	Plus adaptable et flexible grâce à la capacité d'ajouter, de supprimer ou de modifier des agents
Évolutivité	Difficile à faire évoluer pour gérer des problèmes de plus grande taille	Facilement évolutif en ajoutant davantage d'agents ou de types d'agents
Interaction	Interactions internes au système, ou avec l'utilisateur via une interface	Interactions dynamiques entre agents et avec l'environnement ; communication et coordination complexes.
Conception	Conception centralisée et top-down	Conception souvent distribuée, bottom-up, émergence de comportements collectifs
Scalabilité	Limitée aux capacités individuelles	Extensible par ajout d'agents
Exemple type	Aspirateur Roomba	Flotte de drones collaboratifs

4.2.4 Pourquoi les SMA ?

Les systèmes à agents deviennent des candidats aux qualités idéales lorsqu'il s'agit [18, 10]:

- de fournir une solution décentralisée pour les problèmes où l'expertise est répartie et qui ne pourraient pas l'être de façon centralisée ;

- de représenter une solution naturelle pour les problèmes qui font intervenir des éléments distribués et autonomes en favorisant l'interconnexion et la coopération entre ces différents composants ;

- d'offrir un support conceptuel simple et clair tout en tolérant un certain seuil d'incertitude et en améliorant la fiabilité et l'extensibilité du système conçu.

Les SMA présentent ainsi plusieurs avantages notamment :

✓ **Scalabilité** : Les SMA s'adaptent à des problèmes de grande envergure par l'ajout d'agents supplémentaires.
✓ **Robustesse** : L'absence de point de défaillance unique garantit la continuité de service même en cas de panne d'agents individuels.
✓ **Flexibilité** : La possibilité d'avoir des agents aux rôles et objectifs variés permet une adaptation dynamique aux changements.
✓ **Efficacité** : La répartition des tâches et le parallélisme optimisent les performances globales.

▪ Exemple 4.2 — Gestion d'un entrepôt Amazon : .
Des centaines de robots (agents) autonomes déplacent des colis dans l'entrepôt, en coopérant pour optimiser le traitement des commandes.

- **Scalabilité** : Il est possible d'ajouter facilement de nouveaux robots pour augmenter la capacité de traitement.
- **Robustesse** : Si un robot tombe en panne, les autres poursuivent le travail sans interruption.
- **Flexibilité** : Les robots s'adaptent dynamiquement aux changements de stock et de disposition des produits.

4.2.5 Organisation des Systèmes Multi-Agents

L'organisation des agents dans un système multi-agents établit un cadre pour les relations et les interactions entre les agents [16, 17]. Elle influence profondément leurs interactions, leur coordination et l'émergence de comportements collectifs. On distingue plusieurs modèles d'organisation, chacun adapté à des contextes spécifiques [7, 15] :

4.2.5.1 Organisation Hiérarchique

Dans ce modèle, les agents sont hiérarchisés selon la structure d'un arbre. Chaque nœud représente un agent, et possède un lien d'autorité sur ses nœuds-fils. Un ou plusieurs agents jouent un rôle de coordination centralisée. Ces agents "leaders" assignent les tâches, collectent les résultats, et prennent les décisions globales. Ce modèle s'adapte bien aux systèmes de décomposition de tâches et permet la résolution de problèmes de grande envergure.

✓ **Avantages** : prise de décision rapide, contrôle centralisé.

✗ **Inconvénients** : point de défaillance unique, faible robustesse.

▪ Exemple 4.3
une équipe de secours dirigée par un chef de mission qui répartit les rôles entre les agents-robots.

4.2.5.2 Organisation démocratique

Les agents participent collectivement aux décisions par des mécanismes de vote ou de consensus. Aucun agent n'a d'autorité supérieure ; la coordination émerge de la participation équitable de tous.

✓ **Avantages** : équité, résilience, adaptabilité.

✗ **Inconvénients** : décisions potentiellement lentes, conflits possibles.

> ■ **Exemple 4.4**
> un groupe d'agents logiciels dans un réseau pair-à-pair qui doivent choisir un chef temporaire ou une stratégie d'allocation.

4.2.5.3 Organisation de Marché

Dans une organisation basée sur le marché, les agents vendeurs proposent des objets à la vente, sur lesquels des agents acheteurs peuvent enchérir. Les agents négocient comme dans une économie, avec des mécanismes d'enchères et d'allocation de ressources. Chaque agent maximise son utilité en vendant ou achetant des ressources ou des tâches.

✓ **Avantages** : allocation efficace et dynamique des ressources.

✗ **Inconvénients** : Complexité des mécanismes, possibilité de comportements opportunistes, nécessite des modèles de coût.

> ■ **Exemple 4.5**
> des agents de cloud computing négociant dynamiquement l'accès à des serveurs ou à des bandes passantes.

4.2.5.4 Organisation en Coalitions

Une coalition est une alliance temporaire d'agents qui s'unissent et collaborent car leurs intérêts individuels se rencontrent. Lorsque l'objectif de la coalition est atteint, celle-ci est généralement dissoute. La difficulté de cette organisation réside essentiellement dans l'évaluation d'un intérêt partagé entre l'agent et celui de la coalition à former. Ainsi, le problème de cette organisation est de sélectionner l'ensemble approprié d'agents qui permettra de maximiser l'utilité de la coalition dans l'environnement. Notons qu'une coalition forme une seule entité où les agents qui la composent sont considérés comme des membres égalitaires (pas de niveaux hiérarchiques).

✓ **Avantages** : Mutualisation des compétences pour atteindre des objectifs plus ambitieux; Réduction des conflits internes, robustesse.

✗ **Inconvénients** : Risque d'instabilité, temps de formation long, conflits entre coalitions.

> ■ **Exemple 4.6**

Jeux multi-agents : les agents forment des alliances temporaires pour remporter une victoire collective.

4.2.5.5 Organisation stigmergique

C'est une coordination indirecte par l'environnement. Les agents ne communiquent pas directement, mais laissent des traces (indices, signaux, objets) dans l'environnement que les autres agents perçoivent et interprètent.
✓ **Avantages** : forte scalabilité, pas de communication explicite nécessaire, simplicité, émergence de comportements optimaux.

✗ **Inconvénients** : difficulté à contrôler ou prédire les comportements émergents, convergence parfois lente, difficile à contrôler directement.

■ Exemple 4.7
des robots ou des fourmis artificielles déposant des traces chimiques (ou numériques) pour guider les autres vers une ressource.

Le tableau 4.1 présente un résumé comparatif des différentes organisations d'agents.

Organisation	Coordination	Communication	Exemple
Hiérarchique	Centralisée	Directe	Chef de mission, superviseur
Démocratique	Distribuée (vote)	Directe	Agents votant une stratégie ou un leader
Marché	Par négociation	Directe (propositions, enchères)	Appels d'offres pour la répartition de tâches
Stigmergique	Indirecte via l'environnement	Indirecte	Traces chimiques, signaux dans l'environnement

Table 4.1: Résumé comparatif des types d'organisation dans les systèmes multi-agents

4.2.6 Ouverture, Homogénéité et Décentralisation dans les SMA

Les systèmes multi-agents (SMA) peuvent être caractérisés par des propriétés structurelles essentielles qui influencent leur fonctionnement et leur conception. Trois d'entre elles sont particulièrement importantes : l'ouverture, l'homogénéité et la décentralisation.

4.2.6.1 Ouverture

Un SMA est dit **ouvert** lorsque des agents peuvent rejoindre ou quitter le système dynamiquement, sans perturber son fonctionnement global.
À l'inverse, un système **fermé** comporte un ensemble fixe d'agents durant toute l'exécution.

- **Système ouvert** : évolutif et flexible, il tolère l'ajout ou la suppression d'agents à tout moment. *Exemple :* une plateforme e-commerce où de nouveaux agents (clients ou vendeurs) apparaissent ou disparaissent dynamiquement.

- **Système fermé** : plus facile à contrôler, mais moins adaptable aux changements.
  Exemple : une mission robotique planifiée avec des agents prédéfinis et fixes.

4.2.6.2 Homogénéité vs Hétérogénéité

Cela fait référence à la nature des agents dans le système.

- **Système homogène** : tous les agents sont similaires (mêmes rôles, capacités, comportements).
  Exemple : une flotte de drones identiques livrant des colis.
- **Système hétérogène** : les agents diffèrent par leurs rôles, compétences ou niveaux d'autonomie.
  Exemple : un système de secours composé de drones, ambulances et robots de reconnaissance.

4.2.7 Décentralisation

Les SMA sont typiquement **décentralisés**, c'est-à-dire qu'aucun agent ne possède de **vue globale** ni de **contrôle central**.

- La connaissance est distribuée : chaque agent agit selon une vision locale.
- Les décisions sont prises de manière autonome.
- La coordination émerge des interactions entre agents.

✓**Avantages :**

- Pas de point de défaillance unique (robustesse).
- Meilleure scalabilité.
- Réactivité accrue à l'environnement local.

✗**Inconvénients :**

- Difficulté de prédiction globale.
- Risque de conflits ou d'incohérences si la coordination est mal définie.

> ■ **Exemple 4.8**
> un système de feux de signalisation intelligents, où chaque feu est contrôlé localement par un agent, s'adaptant à son trafic et à celui de ses voisins sans supervision centrale.

4.2.8 Domaines d'Application des SMA

Les domaines d'application des SMA sont très vastes grâce à la généricité de leurs concepts. Néanmoins on ne peut pas pré- tendre qu'ils peuvent être appliqués quel que soit le contexte. Par contre, les SMA représentent un outil puissant et viable pour la modélisation et la simulation de systèmes complexes compte tenu des qualités des agents autonomes qui les composent.
Les SMA trouvent leur application dans de nombreux domaines critiques :

- ✓ **Systèmes Industriels**
  - Ordonnancement d'ateliers de production
  - Conduite de processus industriels complexes
  - Systèmes multi-capteurs pour la surveillance.
- ✓ **Systèmes de Contrôle**
  - Gestion du trafic routier urbain
  - Contrôle du trafic aérien
  - Distribution et équilibrage énergétique.
- ✓ **Télécommunications et Réseaux**
  - Routage adaptatif dans les réseaux
  - Équilibrage de charge dynamique

  - Recouvrement d'erreurs et surveillance réseau.

✓ **Applications Émergentes**
  - Commerce électronique et places de marché
  - Data Mining distribué
  - Robotique collaborative
  - Simulation de systèmes complexes.

D'autres domaines d'application sont illustrés par la figure 4.5.

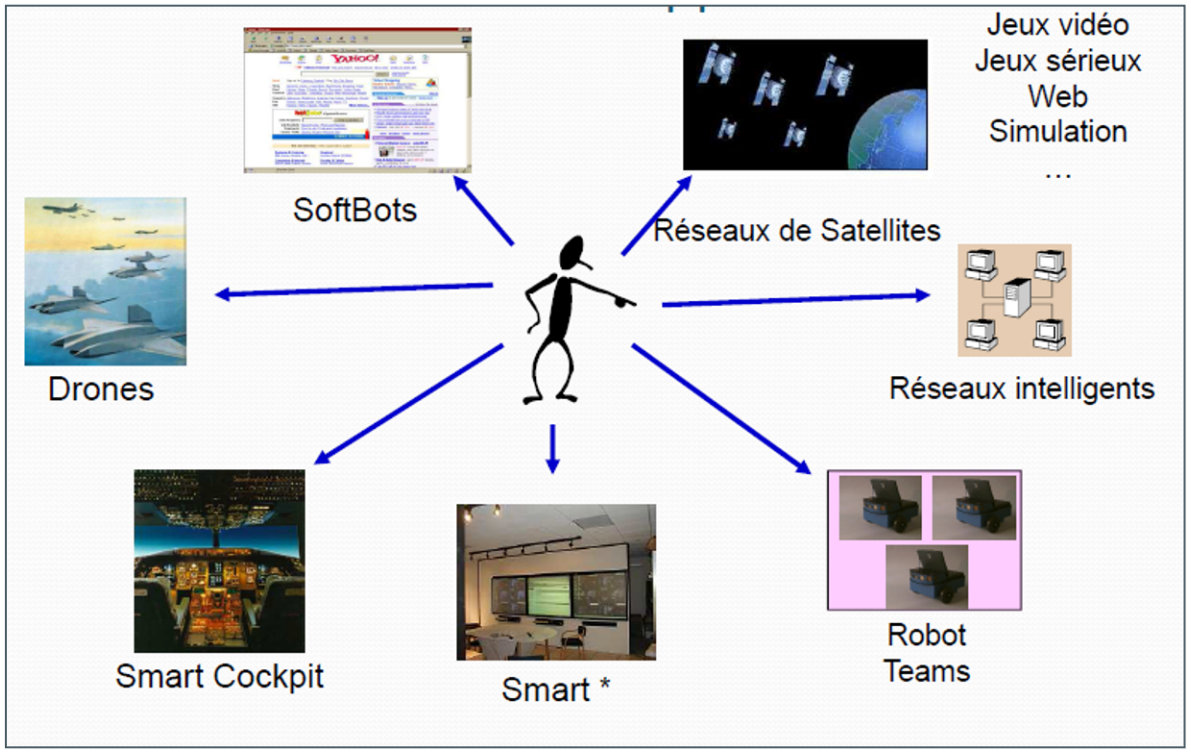


Figure 4.5: Domaines d'Application des SMA.

4.2.9 Études de cas

Cas 1 : Système de gestion de trafic urbain (ex : Singapour)

Problème : Embouteillages dans une grande ville.

Solution SMA :
- Feux de circulation intelligents communiquant entre eux.
- Adaptation en temps réel aux conditions de trafic.
- **Résultat** : Réduction de 20 % du temps de trajet moyen.

Cas 2 : Simulation de foule pour évacuation d'urgence

Problème : Évacuer un stade rapidement et sûrement.

Solution SMA :
- Agents (personnes virtuelles) suivant des règles de mouvement et d'évitement.
- Objectif : Identifier les goulots d'étranglement et optimiser les sorties.

Cas 3 : Trading haute fréquence

Problème : Maximiser les profits en bourse.

Solution SMA :
- Des milliers d'agents négociant des actions en temps réel.
- Coordination via des algorithmes d'enchères.
- Adaptation rapide aux variations de marché.

4.3 Interactions et Communications entre Agents

Les SMA présentent un processus de résolution qui combine à la fois les qualités des techniques de résolution distribuée (parallélisation, redondance) et aussi celles des méthodes d'IA en particulier par l'intégration de mécanismes d'**interaction sophistiqués** comme la **coopération**, la **coordination** et la **négociation** [16, 18]. De ce fait, résoudre les problèmes inhérents à la construction des SMA et des mécanismes d'interactions représente le défi majeur pour exploiter pleinement ces systèmes [17]. La conception, l'implémentation et la validation des SMA incluent évidemment la mise en œuvre de :

- Stratégies de coordination qui permettent effectivement à des groupes d'agents de résoudre des problèmes ;
- Mécanismes de négociation qui permettent à des agents à intérêts conflictuels de parvenir à un accord mutuellement acceptable ;
- Techniques pour détecter et résoudre des conflits ;
- Protocoles par l'intermédiaire desquels les agents peuvent communiquer et raisonner sur leurs communications ;
- Mécanismes par lesquels les agents peuvent maintenir leur autonomie tout en contribuant au fonctionnement général du système.

Il en découle que le comportement global du SMA n'est pas une sommation directe des comportements locaux des agents le composant. L'interaction est, par conséquent, l'essence même des SMA car les agents ne travaillent pas de manière isolée. Elle constitue à la fois la source de sa puissance et l'origine de ses problèmes. Jacques Ferber [15] donne la définition suivante de l'interaction : « *Une interaction est la mise en relation dynamique de deux ou plusieurs agents par le biais d'un*

ensemble d'actions réciproques, etc. ».

Les agents qui agissent localement et de manière autonome doivent souvent **échanger**, **coordonner**, **coopérer** ou même entrer en **compétition** pour atteindre des objectifs communs ou individuels dans un environnement partagé, partiellement observable, non déterministe et changeant. Cette dynamique crée une intelligence collective supérieure à la somme des intelligences individuelles mais introduit une complexité inhérente qui doit être gérée par des mécanismes d'interaction sophistiqués (voir Figure 4.6).
Les interactions entre agents visent principalement à :

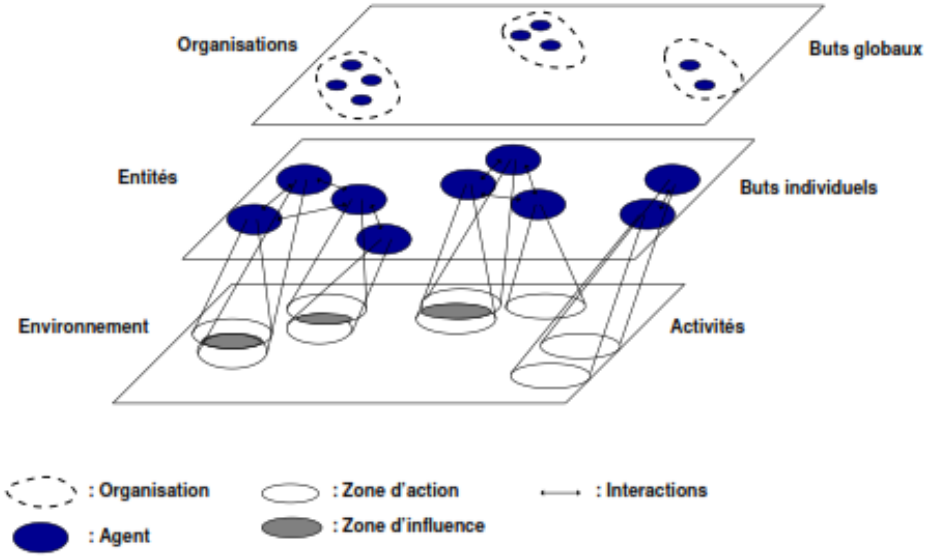


Figure 4.6: Représentation des entités et de leurs interactions

- ✓ **Atteindre des buts communs** : Coordination vers un objectif partagé;
- ✓ **Résoudre des conflits** : Gestion des divergences d'intérêts ou de ressources;
- ✓ **Optimiser les performances collectives** : Amélioration de l'efficacité globale du système;
- ✓ **Partager les connaissances** : Distribution et enrichissement mutuel de l'information;
- ✓ **Adapter le comportement** : Évolution en fonction du contexte et des autres agents.

4.3.1 Formes d'Interaction

Les interactions peuvent être de forme basique très simple, se résumant à un simple échange intentionnel et explicite d'informations, comme elles peuvent être de forme plus évoluée selon que les entités aient des connaissances plus étendues, des capacités de planifier ou encore de faire évoluer leurs raisonnements par apprentissage en observant les actions des autres.
La figure 4.7 résume les différents formes de coordination entre les agents.

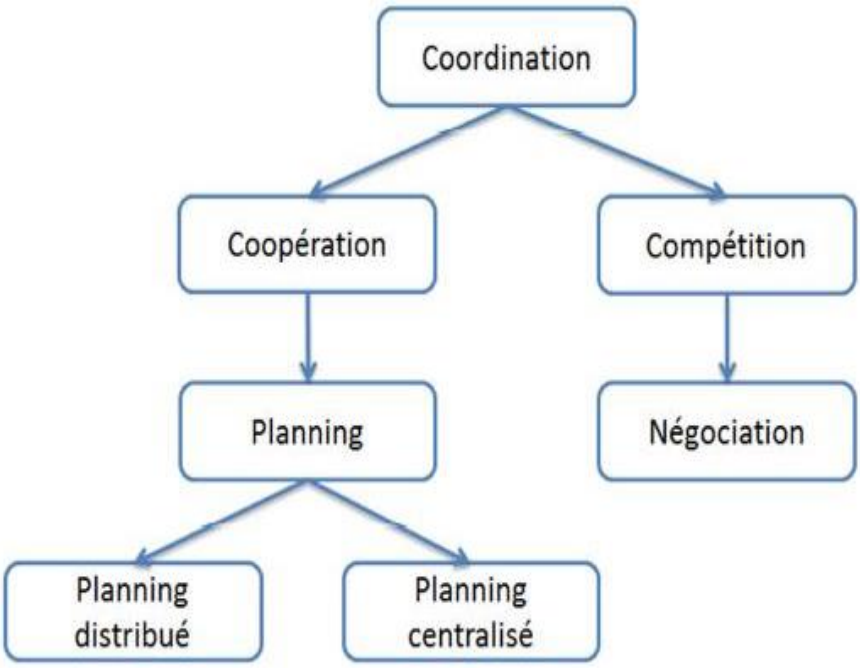


Figure 4.7: Différentes formes d'interactions entre agents

4.3.1.1 Coopération

La coopération représente la forme d'interaction la plus positive où les agents partagent des objectifs communs et collaborent pour atteindre des résultats supérieurs à leurs capacités individuelles. Dans ce cas, le but des agents n'est plus seulement de maximiser leurs propres satisfactions mais aussi de contribuer à la réussite du groupe.

✓ **Caractéristiques** :
- Objectifs alignés ou complémentaires;
- Partage volontaire de ressources et d'informations;
- Bénéfice mutuel recherché;
- Absence de comportements opportunistes.

✓ **Mécanismes de Coopération** :
- **Partage de Connaissances** :

  Agent_A possède : $Expertise\_Domaine\_X$ Agent_B possède : $Expertise\_Domaine\_Y$
  Coopération $\rightarrow Agent\_A + Agent\_B = Solution\_Domaine\_(X \cup Y)$

- **Division Optimale des Tâches** :
  * Allocation basée sur les compétences spécifiques;
  * Parallélisation des activités;
  * Réduction du temps global d'exécution.
- **Planification Conjointe** :
  * Construction collaborative de plans d'action;
  * Synchronisation des calendriers;

* Gestion partagée des ressources.

✓ **Défis de la Coopération** :
- **Free-riding** : Risque qu'un agent bénéficie sans contribuer;
- **Overhead de la coordination** : Coût de communication et synchronisation;
- **Vulnérabilité** : Dépendance aux performances des partenaires.

▪ Exemple 4.9 — Système de Production Collaborative.
Dans une usine automobile moderne, plusieurs robots spécialisés coopèrent pour assembler un véhicule :
- Robot-Soudure : Expertise en assemblage métallique;
- Robot-Peinture : Expertise en finition de surface ;
- Robot-Assemblage : Expertise en montage de composants;
- Robot-Contrôle : Expertise en vérification qualité.

Séquence coopérative :
1. Robot-Soudure → assemblage châssis;
2. Robot-Peinture → application couches protectrices;
3. Robot-Assemblage → installation composants;
4. Robot-Contrôle → validation qualité finale.

4.3.1.2 Planification Collaborative

La planification collaborative est une forme spécialisée de coopération focalisée sur la construction conjointe de séquences d'actions. Les agents doivent s'accorder sur *qui fait quoi ? quand ? et comment ?* pour atteindre un objectif global.

✓ **Planification Centralisée :**
- **Architecture** :

Agent_Planificateur_Central

↓ *(collecte des capacités)*

[Agent_1, Agent_2, ..., Agent_n]

↓ *(distribution du plan)*

Exécution_Coordonnée

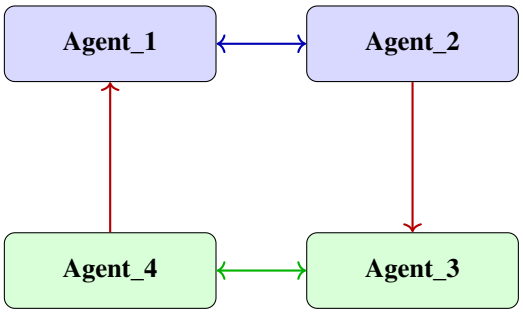
- **Avantages** :
  * Optimalité globale garantie;
  * Résolution efficace des conflits;
  * Vue d'ensemble complète.
- **Inconvénients** :
  * Point de défaillance unique;
  * Scalabilité limitée;

∗ Surcharge de communication.

Exemple : Système de gestion de flotte aérienne où un contrôleur central optimise toutes les trajectoires.

✓ **Planification Distribuée** :

– **Architecture** :



– **Processus** :

Processus de Planification Distribuée

∗ **Phase d'Initialisation** : Chaque agent formule ses objectifs locaux (construit sa partie du plan localement.
∗ **Phase de Négociation** : Échange d'intentions et de contraintes.
∗ **Phase de Raffinement** : Ajustement itératif des plans partiels.
∗ **Phase de Convergence** : Validation du plan global distribué.

Algorithme Type

∗ Pour chaque Agent_i : générer un plan initial basé sur ses objectifs.
∗ Répéter jusqu'à convergence :
  · Chaque agent diffuse son plan aux voisins.
  · Il reçoit les plans des voisins.
  · Il ajuste son plan en conséquence.
∗ Vérifier la cohérence globale.

■ **Exemple 4.10 — Exploration Spatiale Multi-Rovers.** Trois rovers martiens doivent explorer une région géologique :

- **Rover_Alpha** : analyse géochimique
- **Rover_Beta** : cartographie 3D
- **Rover_Gamma** : détection biologique

Planification distribuée :

- Rover_Alpha : « Je peux analyser 5 échantillons/jour »
- Rover_Beta : « Je cartographie 2 km²/jour »
- Rover_Gamma : « J'ai besoin des cartes de Beta »

Plan global :
- Jours 1–3 : Beta cartographie la zone
- Jours 4–8 : Alpha et Gamma explorent en parallèle
- Jours 9–10 : Synthèse des découvertes

4.3.1.3 Coordination

La coordination vise à synchroniser les comportements pour éviter les conflits, optimiser l'utilisation des ressources et améliorer les performances globales.
On distingue plusieurs types de coordiantion:

✓ **Coordination Temporelle** :
- Synchronisation des actions dans le temps;
- Évitement des conflits d'accès simultané;
- Optimisation des séquences d'activation;

✓ **Coordination Spatiale** :
- Gestion de l'occupation de l'espace;
- Évitement de collisions physiques;
- Optimisation des déplacements;

✓ **Coordination des Ressources** :
- Allocation efficace des ressources limitées;
- Prévention des blocages (deadlocks);
- Équilibrage de charge;

Les mécanismes de coordination peuvent être soit préventifs ou réactifs:

✓ **Mécanismes Préventifs** :
- **Réservation de Ressources** :

Calcul des priorités

```
Agent_A : demande_réservation(Ressource_X, [t1, t2])
Gestionnaire :
   if disponible(Ressource_X, [t1, t2]):
     accorder_réservation(Agent_A, Ressource_X, [t1,
t2])
   else:
     proposer_alternative(Agent_A, alternatives)
```

- **Ordonnancement par Priorités** :

Calcul des priorités

```
Priorité_Agent = f(Urgence, Importance, Ancienneté)
File_Attente = trier_par_priorité(Demandes_Pendantes)
```

– Optimisation des séquences d'activation;

✓ **Mécanismes Réactifs** :

– **Détection et Résolution de Conflits** :

Gestion des conflits entre agents

```
si conflit_détecté(Agent_i, Agent_j):
   si Priorité(Agent_i) > Priorité(Agent_j):
     Agent_j.attendre()
   sinon:
     négocier_solution(Agent_i, Agent_j)
```

▪ **Exemple 4.11 — Intersection Intelligente.**
Véhicule_Nord, **Véhicule_Sud**, **Véhicule_Est**, **Véhicule_Ouest**

Algorithme de coordination :

1. **Annonce d'intention :**
   - Véhicule_Nord : « Arrivée dans 10s, direction Sud »
   - Véhicule_Est : « Arrivée dans 12s, direction Ouest »
2. **Détection de conflit :**
   - `Trajectoires_intersectent(Nord→Sud, Est→Ouest) = True`
3. **Résolution :**
   - Priorité temporelle : Nord arrive avant Est
   - Décision : Nord passe, Est attend
4. **Exécution coordonnée :**
   - Nord traverse [t = 10s, t = 13s]
   - Est attend jusqu'à t = 13s, puis traverse.

4.3.1.4 Négociation

La négociation est un processus d'interaction où les agents aux intérêts divergents cherchent à parvenir à un accord mutuellement acceptable. Dans un SMA, la négociation vise plusieurs buts généraux à savoir l'allocation de tâches ou de ressources, la résolution des buts conflictuels ou encore la modification des plans des agents (voir figure 5.1.

On distingue trois axes de réflexion pour la négociation :

- L'**objet de la négociation** : ceci concerne, en général, les alternatives possibles formant l'espace des solutions, et en particulier la structuration de l'objet ;
- **Les protocoles de négociation** : ce sont les règles qui régissent l'interaction entre les parties prenantes de la négociation et déterminent l'état du processus ;

- **Raisonnement et modèles décisionnels des agents** : il s'agit des stratégies mises en œuvre par les agents pour atteindre le meilleur accord possible tout en suivant le protocole.

La négociation automatique s'articule autour de trois phases typiques:

- Offre initiale;
- Contre-offres;
- Accord ou échec. *Exemple : Deux robots négocient l'accès à un couloir étroit en proposant des horaires alternés .*

Plus de détails concernant le processus de négociation seront abordés dans le chapitre suivant.

4.3.1.5 Compétition

La compétition survient lorsque les agents ont des objectifs conflictuels ou rivalisent pour des ressources limitées.
Exemple : Plusieurs agents immobiliers rivalisent pour vendre un appartement à un client intéressé.
La compétition peut être ***pure*** ou ***coopérative*** :

✓ **Compétition Pure :**
- Jeu à somme nulle;
- Le gain d'un agent implique la perte d'un autre;
- Exemple : Enchères exclusives.

✓ **Compétition Coopérative :**
- Mélange de compétition et coopération;
- Bénéfices mutuels possibles malgré la rivalité;
- Exemple : Standards technologiques.

4.3.2 Modes et Mécanismes de Communication

La communication inter-agent est fondamentale à la réalisation du paradigme agent. En effet, pour se coordonner, coopérer ou négocier les agents ont généralement besoin de communiquer leurs intentions, buts, résultats et états. les communications peuvent se dérouler selon un mode **direct** ou **indirect**.

4.3.2.1 Communication Directe

La communication directe entre agents s'appuie sur l'**échange explicite de messages structurés**, respectant le principe fondamental que l'action directe sur un autre agent est interdite (voir figure 4.8): un agent **ne peut pas agir directement** sur l'état et le comportement d'un autre Agent. L'intéraction doit passer par l'envoi de **message ACL** (**A**gent **C**ommunication **L**anguage).

La communication directe est **caractérisée** par :

✓ **Explicité** : Les intentions et informations sont verbalisées clairement;

✓ **Structuration** : Messages suivant des formats standardisés;

✓ **Adressage** : Communication point-à-point ou multicast;

✓ **Synchronisme** : Échange synchrone (message bloquant) ou asynchrone (message non bloquant) selon les besoins.

> ■ **Exemple 4.12 — Communication Synchrone vs Asynchrone.**
> **Communication Synchrone :**

Figure 4.8: Communication directe dans les systèmes multi-agents

- Agent_A : `envoyer_message_bloquant(Agent_B, Message)`
  `attendre_réponse()`
- Agent_B : `traiter_message_immédiat()`
  `envoyer_réponse_immédiate(Agent_A, Réponse)`

Communication Asynchrone :

- Agent_A : `envoyer_message_non_bloquant(Agent_B, Message)`
  `continuer_traitement()`
- Agent_B : `recevoir_message_quand_disponible()`
  `traiter_selon_priorité()`
  `envoyer_réponse_ultérieure(Agent_A, Réponse)`

Comme le montre la figure 4.9, la **topologie** de la communication directe peut être point-à-point, multicast ou broadcast.

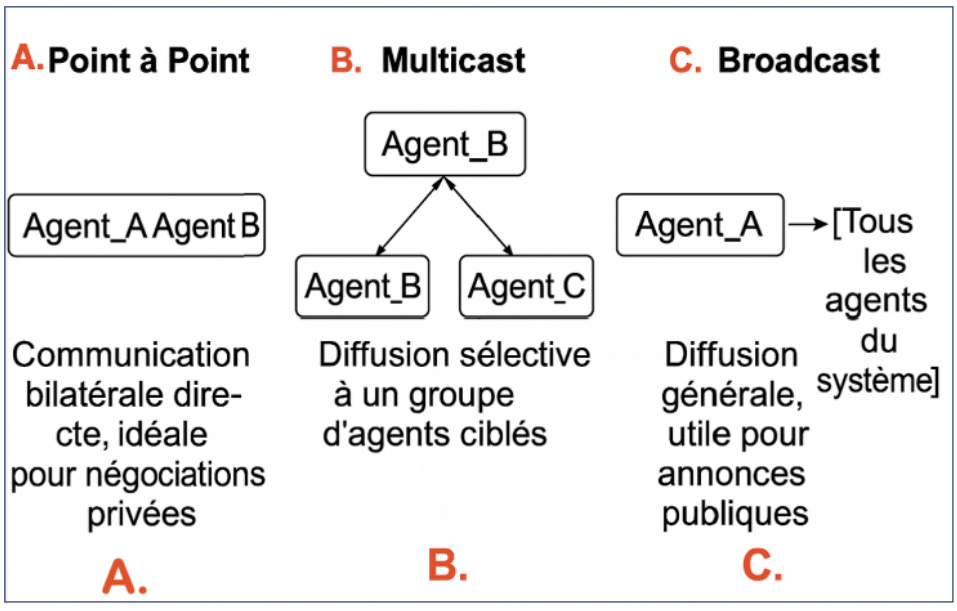


Figure 4.9: Topologie des Communication directe dans les systèmes multi-agents

4.3.2.2 Communication Indirecte (Stigmergie)

La stigmergie représente un mode de communication indirecte où les agents coordonnent leurs actions en modifiant et observant l'environnement partagé [16, 17].
On distingue plusieurs mécanismes de la communication indirecte:

✓ **Communication par Traces** : La coordination se fait via la modification de l'environnement (ex: phéromones). L'algorithme permet de Renforcer les traces par feedback positif. Ce mécanisme est utilisé dans, par exemple, l'optimisation de trajets logistiques.

▪ **Exemple 4.13 — Phéromones Digitales :.**

– **déposer_phéromone(position, intensité):** Augmente le niveau de phéromones à la position spécifiée:

$$\text{nouveau\_niveau} = \text{ancien\_niveau} + \text{intensité}$$

– **suivre_gradient_phéromone():** Choisit la direction avec la plus forte concentration:

$$\text{direction} = \underset{\text{voisins}}{\text{argmax}} \left(\sum \text{phéromones}_i \right)$$

– **évaporer_phéromone(p, taux):** Réduit progressivement les phéromones:

$$p_{\text{new}} = p_{\text{current}} \times (1 - \text{taux})$$

✓ **Communication par Blackboard** : Le **Blackboard** (tableau noir) est une mémoire partagée où les agents déposent et consultent des informations de manière asynchrone (voir figure 4.10.

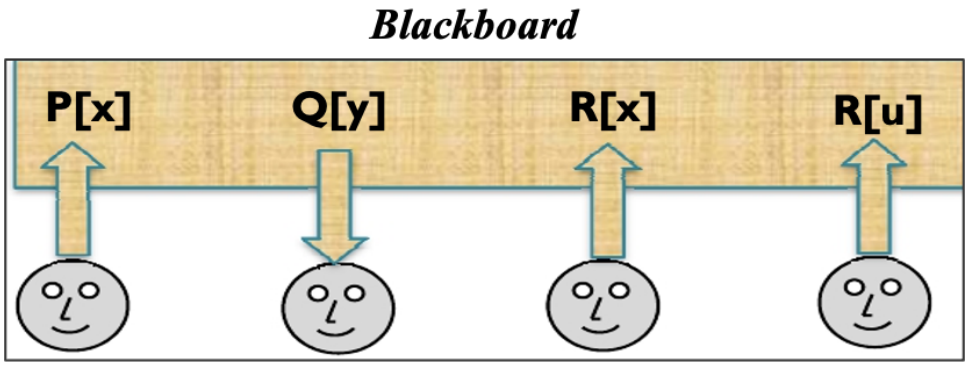


Figure 4.10: Communication Indirecte par Blackboard

✓ **Communication par Signaux** : Un agent emet un signal via l'environnement qui sera intercepté et interprété par les agents récepteurs qui vont adapter leurs comportements en conséquence.
Il existe plusieurs types de signaux:

– **Signaux d'Alarme** : *Agent_Sentinelle détecte menace → émet_signal_alarme(intensité élevée, rayon_large) → Agents_voisins reçoivent signal → adoptent_comportement défensif()*

- **Signaux de Ressource** : *Agent_Découvreur trouve ressource → émet signal_attractif (intensité * qualité_ressource) → Agents_chercheurs suivent gradient signal → convergent vers ressource().*
- **Signaux de Répulsion** : *Agent_en_difficulté émet signal détresse → Agents_compétiteurs évitent zone problématique → réduction_congestion locale().*

4.3.3 Langages de Communication Agents (ACL)

Les modélisations de la communication entre agents s'appuient sur la **théorie des actes de langages**, développée par les linguistes et qui vise à s'inspirer des dialogues entre humains. Cette théorie est à l'origine des langages de communication multi-agents **ACL**.
ACL est un langage standardisé utilisé dans les systèmes multi-agents (SMA) pour permettre aux agents intelligents de communiquer entre eux de manière structurée et interprétable.
Il définit des règles syntaxiques, sémantiques et des protocoles pour échanger des informations, des requêtes, des promesses ou des actions.
Les principaux standards sont KQML(Knowledge Query and Manipulation Language) et FIPA-ACL (Foundation for Intelligent Physical Agents ACL).

4.3.3.1 KQML (Knowledge Query and Manipulation Language)

C'est le premier standard de communication entre agents, orienté messages et indépendant de la syntaxe du contenu.

✓ **Structure d'un message KQML** :
Un **message KQML** (Knowledge Query and Manipulation Language) suit une structure standardisée avec des champs obligatoires et optionnels. La syntaxe générale est :

```
(performative-name
  :sender agent-emetteur
  :receiver agent-destinataire
  :reply-with identifiant-unique
  :in-reply-to reference-message
  :language langage-contenu
  :ontology domaine-vocabulaire
  :content contenu-message
)\\
```

✓ **Description des champs**:

Champ	Description
`performative-name`	Type d'action communicationnelle (`tell`, `ask-one`, `reply`, etc.)
`:sender`	Identifiant de l'agent émetteur du message
`:receiver`	Identifiant de l'agent destinataire du message
`:reply-with`	Identifiant unique pour référencer cette conversation
`:in-reply-to`	Référence au message précédent dans la conversation
`:language`	Langage utilisé pour le contenu (`KIF`, `Prolog`, `SQL`, etc.)
`:ontology`	Domaine sémantique pour interpréter le contenu
`:content`	Contenu informationnel du message

Exemple 4.14

```
(ask-one
  :sender Agent-Client
  :receiver Agent-BaseDeDonnees
  :reply-with req-125
  :language SQL
  :ontology e-commerce
  :content "SELECT prix FROM produits WHERE id = 'P-789'"
)\\
```

✓ Performatives courantes

- `ask-one`: Demande d'une seule réponse à une requête
- `ask-all`: Demande de toutes les réponses possibles
- `tell`: Transmission d'une information
- `reply`: Réponse à une requête précédente
- `subscribe`: Abonnement à des mises à jour
- `error`: Signalement d'une erreur dans le message précédent

✓ Caractéristiques Techniques

- **Indépendance syntaxique**: Le contenu peut utiliser n'importe quel langage (KIF, SQL, Prolog, etc.)
- **Sémantique déclarative**: Basée sur la théorie des actes de langage
- **36 performatives** différentes définies dans le standard
- **Conversations suivies** via les champs `reply-with` et `in-reply-to`

✓ Plates-formes des systèmes multi-agents & Communcation avec KQML

Le tableau 4.2 présente les plateformes les plus usuelles qui supportent KQML comme langage de communication.

✓Comparatif des Caractéristiques

Fonctionnalité	AgentBuilder	JAT	Java Agent Library
Support KQML	✓	✓	✗
Architecture BDI	✗	✓	✗
Interface graphique	✓	✗	✗
Communication TCP/IP	✓	✓	✓
Licence	Commerciale	Open Source	Commerciale

Note : Ces plateformes historiques ont été importantes dans l'évolution des SMA.
Aujourd'hui, des solutions plus modernes comme JADE (Java Agent DEvelopment Framework) leur ont succédé.

Table 4.2: Plateformes historiques de développement SMA

Plateforme	Lien	Description
AgentBuilder	www.agentbuilder.com	Environnement intégré de développement d'agents avec support visuel.
		• Création graphique d'agents • Support KQML/FIPA-ACL • Débogueur intégré • Historique : Développé par Reticular Systems (1995-2005)
JAT (Java Agent Template)	www-cdr.stanford.edu/ABE/JavaAgent.html	Template pour créer des agents Java avec architecture BDI.
		• Développé à Stanford University • Modèle basé sur les croyances, désirs et intentions • Intégration facile avec systèmes existants • Actif de 1998 à 2008
Java Intelligent Agent Library	www-bitpix.com/business/main/bitpix.htm	Bibliothèque légère pour agents intelligents en Java.
		• Architecture événementielle • Communication TCP/IP • Support XML-RPC • Développé par BitPix (1999-2010)

4.3.3.2 FIPA-ACL (Foundation for Intelligent Physical Agents)

C'est un standard moderne basé sur KQML mais avec une **sémantique formelle** et une infrastructure de gestion des agents. Ce langage a été spécifié pour promouvoir l'usage des agents logiciels dans l'industrie. Il est syntaxiquement similaire à KQML, cependant il a été conçu directement avec une sémantique formelle plus rigoureuse. Il peut être ainsi perçu comme une extension de KQML. En effet, FIPA-ACL est aussi basé sur les la théorie des actes de langages mais au lieu de définir des performatifs, il définit des **actes de communications** associées à diverses finalités.

✓ Structure Générale d'un Message FIPA-ACL

La norme FIPA-ACL (Foundation for Intelligent Physical Agents - Agent Communication Language) définit un format standard pour les échanges entre agents. Un message se compose de plusieurs champs obligatoires et optionnels comme illustré par le tableau 4.3.

Table 4.3: Champs d'un message FIPA-ACL

Champ	Description
`performative`	Type d'acte de communication (requête, information, proposition...)
`sender`	Identifiant unique de l'agent émetteur
`receiver`	Identifiant unique de l'agent destinataire
`content`	Contenu informationnel du message
`language`	Langage de représentation du contenu (KIF, Prolog, XML...)
`ontology`	Vocabulaire partagé pour l'interprétation sémantique
`conversation-id`	Identifiant unique de la conversation
`protocol`	Contexte de l'échange (fipa-request, fipa-contract-net...)
`reply-with`	Identifiant pour référencer ce message
`in-reply-to`	Référence au message précédent
`reply-by`	Délai maximal pour la réponse

■ **Exemple 4.15**

```
(cfp
  :sender agent-manager
  :receiver (agent-fournisseur1 agent-fournisseur2)
  :content 'Besoin de 100 unites produit X avant 2023-12-31'
  :language XML
  :ontology e-commerce
  :protocol fipa-contract-net
  :conversation-id cmd-789
  :reply-by "2023-12-15T12:00:00Z"
)
```

✓ Catégories de Performatives

✓ Détails des Performatives Courantes

- `query_if`: Demande si une proposition est vraie
- `inform`: Transmission d'une information
- `cfp` (Call For Proposal): Appel à propositions
- `propose`: Soumission d'une proposition
- `accept-proposal`: Acceptation d'une offre
- `not-understood`: Indique qu'un message n'a pas été compris

✓ Caractéristiques Techniques

- **Théorie des actes de langage**: Basée sur les travaux de Austin et Searle
- **20 performatives** standardisées
- **Interopérabilité**: Permet la communication entre agents hétérogènes
- **Sémantique formelle**: Définie par des règles d'interprétation
- **Protocoles standard**: fipa-request, fipa-query, fipa-contract-net

Table 4.4: Catégories de performatives FIPA-ACL

Catégorie	Performatives	Description
Information	query_if, inform, confirm, subscribe	Messages d'échange d'information et de requête de données
Gestion d'erreurs	not-understood, failure	Signalement d'erreurs de communication ou d'exécution
Négociation	cfp, propose, accept_proposal, reject_proposal	Protocoles de négociation et d'enchères
Actions	request, agree, cancel	Demande d'exécution d'actions et réponses associées

✓ Plates-formes utilisant Fipa-ACL

Le tabeau 4.5 présente les plateformes les plus usuelles utilisant le langage Fipa-ACL.

✓ Comparatif Technique

Caractéristique	JADE	SPADE	FIPA-OS	JACK	Cougaar	JIAC
Langage	Java	Python	Java	Java	Java	Java
Licence	LGPL	MIT	LGPL	Commerciale	Apache	LGPL
Architecture	BDI/Event	BDI/Event	Agentic	BDI	Cognitive	Service
IDE	Oui	Non	Non	Oui	Non	Oui
Scalabilité	+++	++	+	++	++++	+++
Cloud	Oui	Oui	Non	Limited	Oui	Oui
RDF/DF	Oui	Oui	Oui	Oui	Oui	Oui
Messagerie	Propriétaire	XMPP	Propriétaire	Propriétaire	Propriétaire	Propriétaire

✓ Caractéristiques FIPA Implémentées

- **AMS (Agent Management System)** : Gestion du cycle de vie des agents
- **DF (Directory Facilitator)** : Service d'annuaire des agents
- **ACC (Agent Communication Channel)** : Transport des messages ACL
- **Protocoles standard** : Request, Contract-Net, Subscribe, etc.
- **Sémantique formelle** : Implémentation des performatives et protocoles

✓ Applications Typiques

- **Télécommunications** : Gestion dynamique de réseaux (JADE, JIAC)
- **Logistique** : Optimisation de chaînes d'approvisionnement (Cougaar)
- **Systèmes embarqués** : Contrôle de véhicules autonomes (JACK)
- **Énergie** : Gestion de smart grids (JIAC, SPADE)

Table 4.5: Plates-formes compatibles FIPA-ACL

Plate-forme	Lien	Statut	Caractéristiques
JADE (Java Agent DEvelopment Framework)	jade.tilab.com	Actif	• Implémentation de référence FIPA • Gestion complète du cycle de vie des agents • Outils de débogage (Sniffer, Introspector) • Support des protocoles FIPA standard • Plateforme distribuée avec AMS et DF
SPADE (Smart Python Agent Development Environment)	spade-mas.readthedocs.i	Actif	• Basé sur Python et XMPP • Support natif de FIPA-ACL • Architecture asynchrone • Facile à intégrer avec l'écosystème Python • Support des comportements complexes
FIPA-OS (FIPA Open Source)	fipa-os.sourceforge.net	Arrêté	• Première implémentation open source FIPA • Architecture modulaire • Support des services d'annuaire (DF) • Historiquement importante (1998-2006) • Remplacée par JADE
JACK Intelligent Agents	aosgrp.com	Actif	• Plateforme commerciale • Support complet de FIPA-ACL • Architecture BDI avancée • Utilisée dans les systèmes critiques (défense, aviation) • Environnement de développement intégré
Cougaar (Cognitive Agent Architecture)	cougaar.org	Maintenanc	• Développée par DARPA • Support FIPA-ACL étendu • Optimisée pour les grandes applications distribuées • Utilisée dans la logistique militaire • Scalabilité extrême
Java Intelligent Agent Componentware (JIAC)	jiac.de	Actif	• Développé par l'agence allemande des télécoms • Certification FIPA complète • Framework orienté service • Outils de gestion et de monitoring • Utilisé dans les réseaux intelligents

- **Santé** : Systèmes de coordination médicale (JADE)

✓ Évolution et Tendances

- **Intégration Web** : Passage à des architectures REST/gRPC
- **Conteneurisation** : Déploiement via Docker/Kubernetes

- **Hybridation** : Combinaison SMA/Blockchain/IoT
- **IA intégrée** : Utilisation de modèles de LLM pour la négociation

4.4 Protocoles d'Interaction FIPA

Un **protocole** est ensemble de **règles** définissant le comportement des agents lors d'échanges de messages (*qui peut parler, dans quel ordre, quel type de message, que répondre*).
Une **conversation** est une **séquence concrète** de messages échangés entre agents, conforme à un protocole donné.
Un **protocole d'interaction** permet de décrire explicitement les enchaînements conversationnels lors des communications entre les agents.

4.4.1 Protocoles FIPA Standards

FIPA est une organisation qui a crée des standards pour les systèmes multi-agents.elle définit des protocoles d'interaction (ex : demande, enchère, souscription) en utilisant le langage de modélisation **AUML** (extension d'UML pour modéliser les interactions entre agents).
Le standard FIPA-ACL définit plusieurs protocoles d'interaction spécialisés :

Protocoles Fondamentaux

Request	Demande d'exécution d'action (`request` → `agree/inform`)
Query	Requête d'information (`query-if` → `inform`)
Subscribe	Abonnement à notifications (`subscribe` → `inform` périodique)

Protocoles de Négociation

Contract Net	Allocation de tâches par enchères inversées (`cfp` → `propose` → `accept`)
Auction English	Enchères ascendantes publiques
Auction Dutch	Enchères descendantes
Propose	Négociation directe d'offres (`propose` → `accept/reject`)

Protocoles Collaboratifs

Brokering	Médiation via agent tiers (`request` → `proxy` → `inform`)
Recruiting	Recrutement d'agents (`recruit` → `agree`)
Recommend	Recommandation de services

Protocoles Spécialisés

Iterated Contract Net	Enchères itératives pour tâches complexes
Forward	Transfert de requêtes à d'autres agents
Propagate	Diffusion d'information en cascade

Caractéristiques Commune

- **Chaînage** : Références via `conversation-id` et `in-reply-to`
- **Temporalité** : Gestion des délais avec `reply-by`
- **Sémantique** : Interprétation via ontologies partagées
- **Extensibilité** : Adaptation aux besoins spécifiques

Dans ce qui suit, nous allons détailler deux types de protocoles : Request Protocol et Contract Net Protocol.

4.4.1.1 Request Protocol (Protocole Demande)

Protocole Request (Demande)

Objectif : Un agent (initiateur) demande à un autre agent (participant) d'exécuter une action
Caractéristiques :
- Protocole simple et fondamental
- Identifiant FIPA : `fipa-request`
- Utilisé pour les interactions client-serveur
- Supporte différentes réponses (acceptation/refus)

Performatives clés : `request`, `agree`, `refuse`, `inform`, `failure`

– Phases du Protocole :
Comme le montre la figure 4.11, le protocole de demande suit les **étapes suivantes** :

1. **Demande** - Initiateur envoie une requête (`request`)
2. **Réponse** - Participant accepte (`agree`) ou refuse (`refuse`)
3. **Exécution** - Si accepté, participant exécute l'action
4. **Résultat** - Participant informe du résultat (`inform`) ou échec (`failure`)

– États Possibles :

État	Description
En attente	Requête envoyée, en attente de réponse
Accepté	Le participant a accepté la requête
Refusé	Le participant a refusé la requête
En cours	Action en cours d'exécution
Terminé	Action complétée avec succès
Échec	Échec lors de l'exécution

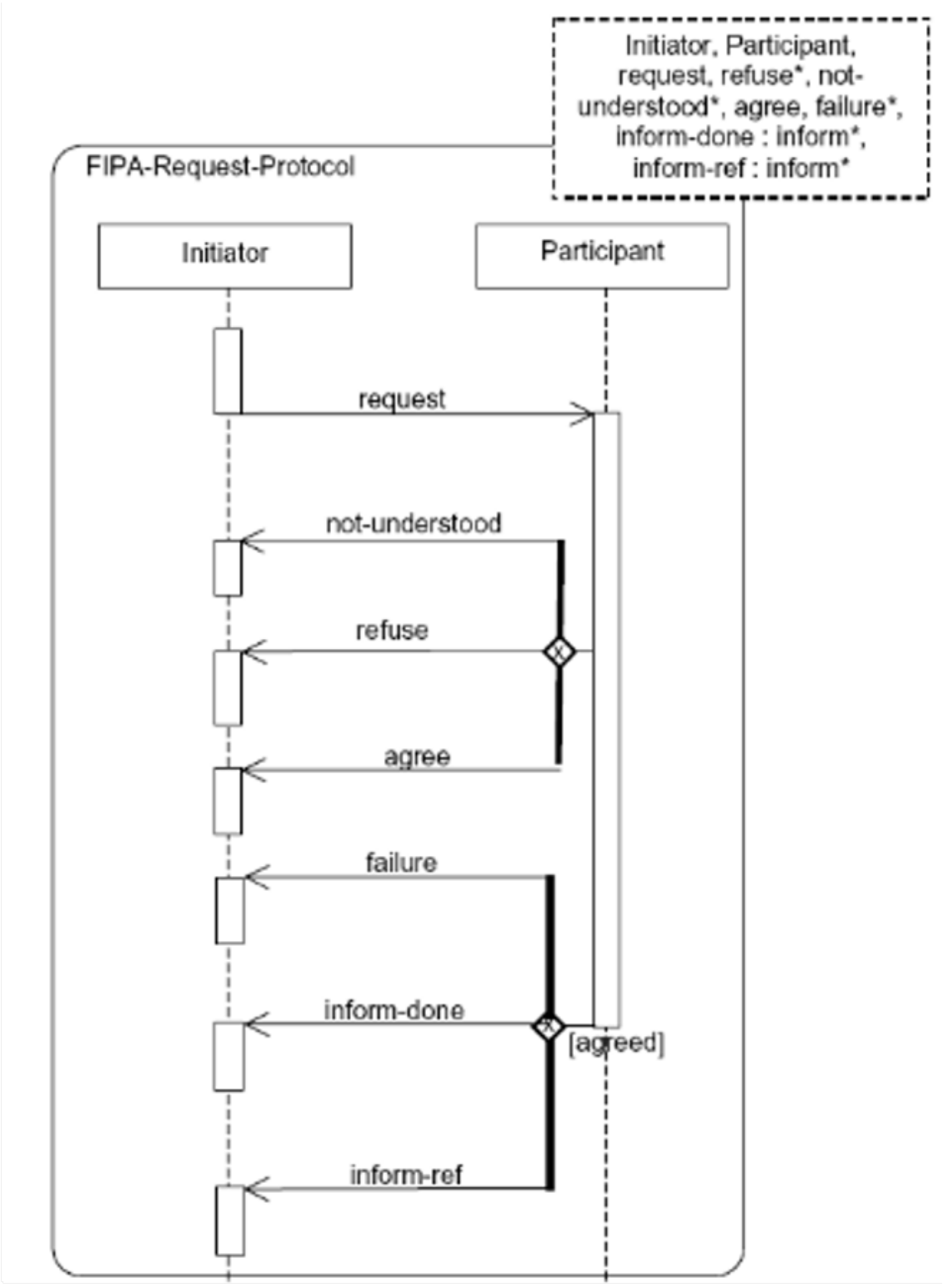


Figure 4.11: Diagramme de séquence AUML du Protocole Request

– Exemple d'Échange

1. Demande (Initiateur → Participant)

```
(request
  :sender client@system
  :receiver weather-service
  :content 'Fournir temperature a Bejaia'
  :language KIF
  :ontology weather
  :protocol fipa-request
  :conversation-id req-789
  :reply-by '2025-11-20T12:00:00Z'
)
```

Analyse

- `request` : Performative standard pour les demandes
- **Contenu clair** : Action demandée explicitement
- **Délai** : `:reply-by` fixe un timeout
- **Contexte** : `protocol` spécifie le protocole

2. Acceptation (Participant → Initiateur)

```
(agree
  :sender weather-service
  :receiver client@system
  :content 'Requete acceptee'
  :language KIF
  :ontology weather
  :protocol fipa-request
  :conversation-id req-789
  :in-reply-to req-789
)
```

Analyse

- `agree` : Confirmation d'acceptation
- **Chaînage** : `:in-reply-to` référence la requête
- **Engagement** : Le participant s'engage à exécuter l'action

3. Résultat (Participant → Initiateur)

```
(inform
  :sender weather-service
  :receiver client@system
  :content 'Temperature a Paris: 22 C'
  :language KIF
  :ontology weather
  :protocol fipa-request
  :conversation-id req-789
  :in-reply-to req-789
)
```

Analyse

- `inform` : Communication du résultat
- **Donnée utile** : Contenu répond à la requête initiale
- **Clôture** : Même `conversation-id` et référence

Scénarios Alternatifs

Refus (Participant → Initiateur)

```
(refuse
  :sender weather-service
  :receiver client@system
  :content "Service indisponible"
  :language KIF
  :ontology weather
  :protocol fipa-request
  :conversation-id req-789
  :in-reply-to req-789
)
```

Causes possibles : Ressources insuffisantes, autorisation manquante, erreur interne

Échec (Participant → Initiateur)

```
(failure
  :sender weather-service
  :receiver client@system
  :content "Echec de la requete: capteur defectueux"
  :language KIF
  :ontology weather
  :protocol fipa-request
  :conversation-id req-789
  :in-reply-to req-789
)
```

Causes possibles : Erreur d'exécution, timeout, ressource non trouvée

Bonnes Pratiques

- **Timeout** : Toujours spécifier un `:reply-by`
- **Identifiants** : Utiliser des `conversation-id` uniques
- **Chaînage** : Maintenir les références avec `:in-reply-to`
- **Ontologies** : Utiliser des vocabulaires partagés
- **Erreurs** : Toujours gérer les réponses négatives

Applications Typiques

Domaine	Utilisation
Services Web	Accès à des API distantes
Bases de données	Requêtes SQL/NoSQL
IoT	Contrôle de capteurs/actionneurs
Robotique	Commande de dispositifs physiques
Systèmes experts	Requêtes à des moteurs de règles

Implémentation dans JADE

Code Java :

```
ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
request.addReceiver(new AID("weather-service", AID.ISLOCALNAME));
request.setContent("Fournir temperature a Paris");
request.setProtocol(FIPAProtocolNames.FIPA_REQUEST);
request.setReplyByDate(new Date(System.currentTimeMillis() + 60000)); //
    60s timeout
myAgent.send(request);
```

4.4.1.2 Contract-Net Protocol (Réseau Contractuel)

Le Contract Net Protocol (CNP) repose sur un mécanisme d'allocation décentralisée de tâches régi par le protocole d'appel d'offres qui est utilisé dans les organisations humaines. Ce protocole de haut niveau est probablement le protocole le plus largement utilisé dans les contextes de négociations multi-agents.

Protocole Contract Net (Appel d'offres)

Objectif : Un agent (Manager) alloue une tâche à travers un processus d'enchères inversées
Caractéristiques :

- Protocole standard FIPA : `fipa-contract-net`
- Modèle économique : Enchères inversées
- Trois rôles : Manager, Participant, Gagnant
- Flexible : Supporte les offres multiples, refus et annulations

Performatives clés : `cfp`, `propose`, `accept-proposal`, `reject-proposal`, `inform`

Phases du Protocole

comme illustré par la figure 4.12, le protocole contract net compte les phases suivantes:

1. **Appel d'offres (CFP)** - Manager diffuse la tâche
2. **Soumission** - Participants proposent des offres
3. **Sélection** - Manager choisit et notifie le gagnant
4. **Exécution** - Participant gagnant exécute la tâche
5. **Notification** - Résultat communiqué au Manager

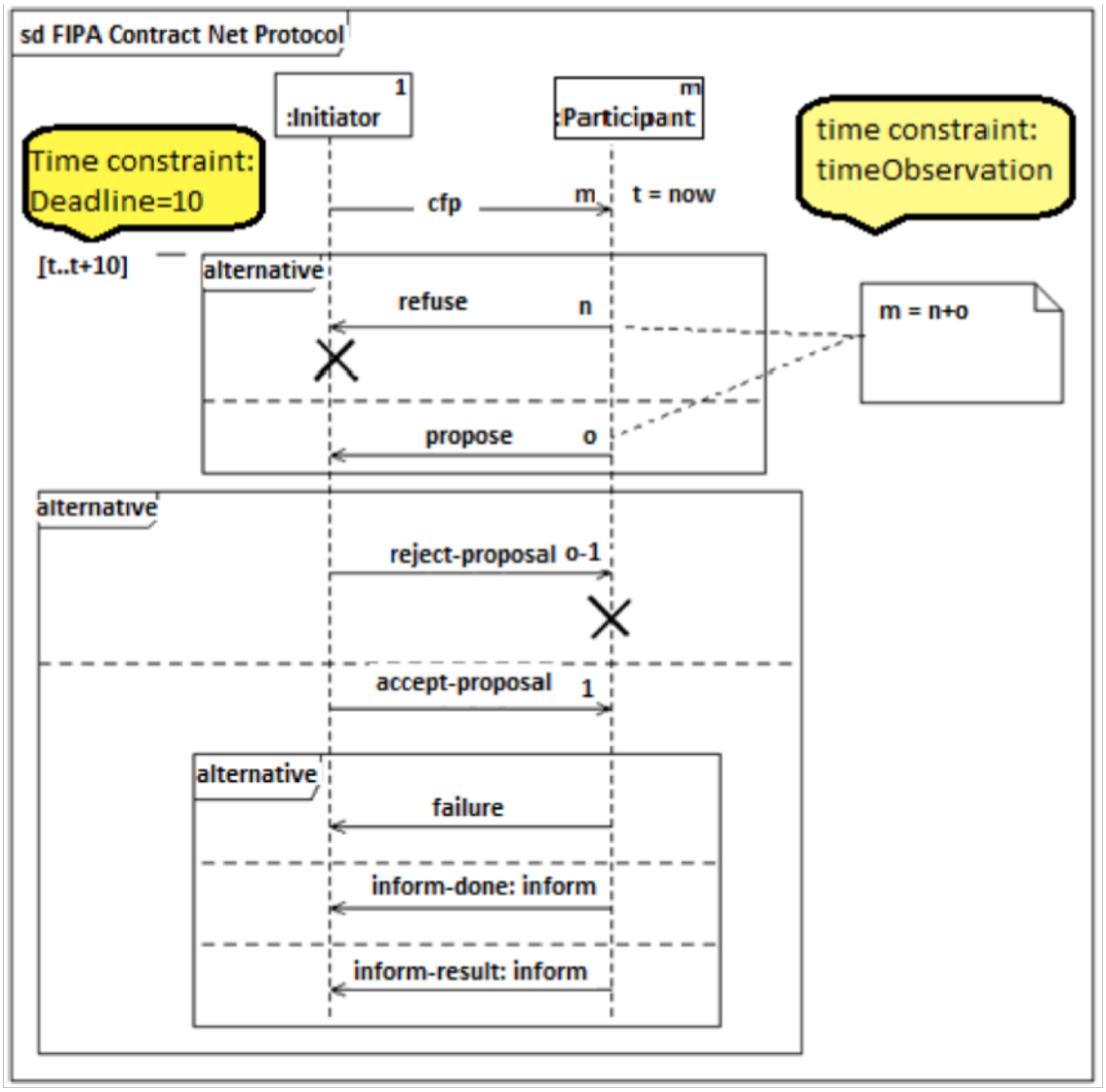


Figure 4.12: Diagramme de Séquence AUML du Protocole Contract Net

Rôles et Responsabilités

Rôle	Description
Manager	• Initie le processus • Définit les critères de sélection • Sélectionne la meilleure offre • Surveille l'exécution
Participant	• Évalue la faisabilité • Soumet une proposition compétitive • Exécute la tâche si sélectionné • Peut refuser l'appel d'offres
Gagnant	• Participant sélectionné • Doit exécuter la tâche • Rend compte du résultat • Peut échouer et notifier

Exemple d'Échange

Dans cet exemple, un agent Manager envoie un appel à proposition à plusieurs agents drones pour livrer de la marchandise à un client X.

1. Appel d'Offres (Manager → Participants)

```
(cfp
  :sender manager@logistics
  :receiver (drone1@fleet drone2@fleet drone3@fleet)
  :content
    "<delivery id='D456'>
       <from>Warehouse A</from>
       <to>Client X</to>
       <weight>2.5kg</weight>
       <deadline>2023-12-05T15:00:00Z</deadline>
     </delivery>"
  :language XML
  :ontology delivery-services
  :protocol fipa-contract-net
  :conversation-id cn-2023-789
  :reply-by "2023-12-01T12:00:00Z"
)
```

Analyse

- `cfp` : Performative d'appel à propositions
- **Contenu structuré** : Détails de la livraison en XML
- **Multicast** : Envoyé à plusieurs participants
- **Délai** : `:reply-by` fixe une deadline

2. Proposition (Drone1 → Manager)

```
(propose
  :sender drone1@fleet
  :receiver manager@logistics
  :content
    "<offer id='O789'>
       <cost>15.00 USD</cost>
       <eta>45 minutes</eta>
       <conditions>Weather permitting</conditions>
     </offer>"
  :language XML
  :ontology delivery-services
  :protocol fipa-contract-net
  :conversation-id cn-2023-789
  :reply-with offer-789
)
```

Analyse

- `propose` : Performative de soumission d'offre
- **Proposition compétitive** : Coût, délai et conditions
- **Identifiant** : `:reply-with` pour référence future

3. Acceptation (Manager → Drone1)

```
(accept-proposal
  :sender manager@logistics
  :receiver drone1@fleet
  :content "Acceptation offre O789"
  :language KIF
  :ontology delivery-services
  :protocol fipa-contract-net
  :conversation-id cn-2023-789
  :in-reply-to offer-789
)
```

Analyse

- `accept-proposal` : Confirmation de sélection
- **Chaînage** : `:in-reply-to` référence l'offre
- **Engagement** : Crée un contrat entre les parties

🏁 4. Résultat (Drone1 → Manager)

```
(inform
  :sender drone1@fleet
  :receiver manager@logistics
  :content
    "<result id='D456'>
       <status>delivered</status>
       <timestamp>2023-12-05T14:30:00Z</timestamp>
       <signature>client_X_signature</signature>
     </result>"
  :language XML
  :ontology delivery-services
  :protocol fipa-contract-net
  :conversation-id cn-2023-789
  :in-reply-to offer-789
)
```

Analyse

- `inform` : Notification de complétion
- **Preuve d'exécution** : Signature et timestamp
- **Clôture** : Termine le cycle contractuel

– Scénarios Alternatifs

❌ Refus de Participation (Drone2 → Manager)

```
(refuse
  :sender drone2@fleet
  :receiver manager@logistics
  :content "Batterie insuffisante"
  :language KIF
  :ontology delivery-services
  :protocol fipa-contract-net
  :conversation-id cn-2023-789
  :in-reply-to cn-2023-789
)
```

Causes possibles : Capacité insuffisante, conflit d'horaire, désintérêt

Rejet d'Offre (Manager → Drone3)

```
(reject-proposal
  :sender manager@logistics
  :receiver drone3@fleet
  :content "Offre non competitive"
  :language KIF
  :ontology delivery-services
  :protocol fipa-contract-net
  :conversation-id cn-2023-789
  :in-reply-to offer-791
)
```

Causes possibles : Coût trop élevé, délai trop long, manque de fiabilité

– Algorithmes de Sélection

Critères de Décision

$$\text{Décision} = \begin{cases} \min(C_i) & \text{Optimisation de coût} \\ \min(T_i) & \text{Optimisation de temps} \\ \alpha C_i + \beta T_i & \text{Combinaison linéaire} \\ U_i(C_i, T_i, R_i) & \text{Fonction d'utilité} \end{cases}$$

- C_i : Coût de la proposition
- T_i : Temps d'exécution estimé
- R_i : Fiabilité du participant
- α, β : Poids relatifs

– Applications Typiques

Domaine	Utilisation
Logistique	Allocation de livraisons en temps réel
Calcul distribué	Attribution de tâches de calcul
Réseaux électriques	Équilibrage de charge dans les smart grids
Robotique collective	Coordination de robots dans un entrepôt
Cloud computing	Allocation dynamique de ressources

– Bonnes Pratiques

Recommandations

- **Délais clairs** : Spécifier des deadlines réalistes
- **Critères transparents** : Définir des règles de sélection objectives
- **Feedback** : Toujours répondre aux participants (acceptation/rejet)
- **Pénalités** : Prévoir des mécanismes pour les échecs
- **Historique** : Conserver les performances passées des participants

Implémentation dans JADE

Code Java :

```
// Manager - Envoi du CFP
ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
for (AID participant : participants) {
    cfp.addReceiver(participant);
}
cfp.setContent(deliveryTask.toXML());
cfp.setProtocol(FIPAProtocolNames.FIPA_CONTRACT_NET);
cfp.setReplyByDate(deadline);
myAgent.send(cfp);

// Participant - Traitement du CFP
public void handleCfp(ACLMessage cfp) {
    if (canPerformTask(cfp)) {
        ACLMessage propose = cfp.createReply(ACLMessage.PROPOSE);
        propose.setContent(myOffer.toXML());
        send(propose);
    } else {
        send(cfp.createReply(ACLMessage.REFUSE));
    }
}

// Manager - Traitement des propositions
public void handleProposals(Vector<ACLMessage> proposals) {
    ACLMessage bestOffer = selectBestOffer(proposals);
    ACLMessage accept = bestOffer.createReply(ACLMessage.ACCEPT_PROPOSAL)
        ;
    send(accept);

    // Rejeter les autres offres
    for (ACLMessage proposal : proposals) {
        if (proposal != bestOffer) {
            send(proposal.createReply(ACLMessage.REJECT_PROPOSAL));
        }
    }
}
```

4.5 Conclusion

Les Systèmes Multi-Agents représentent une évolution naturelle de l'Intelligence Artificielle vers des approches distribuées et collaboratives [16, 18]. Leur capacité à résoudre des problèmes complexes par l'intelligence collective, tout en maintenant l'autonomie des agents individuels, en fait un paradigme incontournable pour les applications modernes [17, 19].
L'évolution continue des standards de communication, des protocoles d'interaction et des outils de développement confirme la prédiction selon laquelle "dans 10 ans, la plupart des nouveaux développements informatiques seront affectés, et de nombreux produits consommateurs contiendront des systèmes intégrés basés sur des agents" [18].
La maîtrise des concepts, méthodes et outils présentés dans ce chapitre constitue un fondement essentiel pour tout ingénieur en Intelligence Artificielle souhaitant concevoir et développer des systèmes intelligents distribués performants et robustes [19].

4.6 Exercices

Études de cas sur les Systèmes Multi-Agents

Les études de cas suivants illustrent concrètement les concepts théoriques présentés dans ce chapitre. Elles montrent comment les agents interagissent, coopèrent ou entrent en compétition au sein de systèmes distribués, dans divers domaines d'application.

Exercice 4.1 Étude de cas 1 : Gestion collaborative de robots logistiques

Contexte : Dans un entrepôt automatisé, plusieurs robots (agents) transportent des colis vers différentes zones. Chaque robot dispose d'une capacité limitée, d'une autonomie énergétique et d'un itinéraire préféré.
Problématique : Comment répartir les tâches de livraison sans supervision centrale, tout en évitant les collisions et en optimisant les trajets ?
Questions :

Quels sont les rôles et objectifs de chaque agent ?

- Quelle architecture (centralisée, distribuée, hiérarchique) est la plus adaptée ?
- Quel protocole d'interaction utiliser (ex. : *Contract Net Protocol*) ?
- Comment gérer les conflits d'itinéraires ?

Concepts illustrés : Coordination décentralisée, communication directe, auto-organisation.

Exercice 4.2 Étude de cas 3 : Marché électronique multi-agents

Contexte : Une plateforme d'e-commerce est modélisée comme un SMA où des agents vendeurs et acheteurs négocient des prix.
Problématique : Concevoir un mécanisme de négociation automatique équitable.
Questions :

Comment représenter les préférences des agents ?

- Quels protocoles de négociation (enchères, offres successives, etc.) peuvent être utilisés ?
- Comment garantir l'équilibre ou la Pareto-optimalité ?

Concepts illustrés : Interaction compétitive, négociation automatique, théorie des jeux.

Exercice 4.3

Étude de cas 4 : Agents pédagogiques dans un environnement d'apprentissage

Contexte : Une plateforme d'e-learning comprend :

- des **agents tuteurs** : adaptent le contenu aux besoins des étudiants ;
- des **agents étudiants** : représentent les profils et préférences ;
- des **agents coordinateurs** : équilibrent la charge entre tuteurs.

Problématique : Comment ces agents coopèrent-ils pour fournir une assistance personnalisée ?

Questions :

- Quelle structure de communication adopter ?
- Comment un agent tuteur décide-t-il d'intervenir ?
- Quelles informations doivent être partagées ?

Concepts illustrés : Coopération entre agents hétérogènes, systèmes adaptatifs, coordination distribuée.

Exercice 4.4 **Étude de cas 5 : Gestion énergétique distribuée**

Contexte : Dans un micro-réseau électrique, chaque maison est représentée par un agent intelligent capable de produire, consommer ou stocker de l'énergie.

Problématique : Optimiser la consommation et l'échange d'énergie sans supervision centrale.

Questions :

Quels sont les objectifs individuels et collectifs ?

- Comment échanger les informations énergétiques entre agents ?
- Quel protocole de coordination adopter ?

Concepts illustrés : Coopération, durabilité, optimisation décentralisée.

Exercice 4.5 **Étude de cas 6 : Simulation de sauvetage d'urgence**

Contexte : À la suite d'une catastrophe naturelle, plusieurs équipes de secours (agents) doivent explorer des zones, localiser les victimes et coordonner les interventions.

Problématique : Comment répartir efficacement les zones d'intervention et coordonner les secours sans supervision centrale ?

Questions :

Quels rôles attribuer aux agents (scout, secouriste, coordinateur) ?

- Quels messages échanger pour signaler des découvertes ou des besoins ?
- Comment adapter la stratégie si certains agents deviennent inactifs ?

Concepts illustrés : SMA coopératif, tolérance aux pannes, communication adaptative.

5. Négociation Automatique

5.1 Introduction

Dans les systèmes multi-agents, la négociation constitue un mécanisme fondamental permettant aux agents autonomes de résoudre leurs conflits, de coordonner leurs actions et d'atteindre des accords mutuellement acceptables [20, 21]. Par exemple, dans un marché électronique, des agents acheteurs et vendeurs peuvent négocier les prix ; dans une smart grid, des agents producteurs et consommateurs d'énergie négocient la consommation et la distribution en temps réel [22]. Ce chapitre explore les différentes facettes de la négociation entre agents, depuis ses motivations jusqu'à ses applications concrètes dans un domaine tel que le smart grid.

5.2 Définition

La négociation automatique est un processus informatique permettant à des agents logiciels de mener des transactions de manière autonome, sans intervention humaine [23, 24]. Elle s'appuie sur des protocoles d'échanges et des algorithmes de prise de décision pour déterminer les meilleurs termes de la transaction qui permettent d'atteindre un accord entre les différentes parties en négociation.

5.3 Pourquoi négocier ?

Dans un système multi-agents, les entités logicielles interagissent dans un environnement commun, mais poursuivent souvent des objectifs différents, voire contradictoires [20]. La négociation s'impose alors comme une nécessité dans ces environnements pour plusieurs raisons [23] :

- **Résolution de conflits d'intérêts** : Les agents évoluent souvent dans des contextes où les ressources sont limitées, où les objectifs peuvent être contradictoires, et où la coordination devient indispensable pour éviter les blocages ou les inefficacités. La négociation représente alors une approche privilégiée pour la résolution de ces conflits . Elle permet de trouver des solutions acceptables pour toutes les parties impliquées plutôt que d'entrer dans une compétition inefficace ou destructrice.

- **Maximisation de l'utilité individuelle et collective** : Chaque agent cherche à optimiser son propre gain. En négociant, les agents peuvent parvenir à des accords équilibrés qui respectent leurs préférences tout en augmentant l'efficacité globale du système (accords dits Pareto-optimaux).
- **Préservation de l'autonomie des agents** : Plutôt que d'imposer une décision collective, la négociation permet à chaque agent de défendre ses intérêts tout en acceptant des compromis. Cela respecte la nature autonome des agents intelligents.

▪ Exemple 5.1 Agents sur une place de marché

Considérons un scénario concret de négociation dans le contexte de la création d'une entreprise virtuelle. Le processus se déroule en plusieurs étapes :

1. **Initialisation** : Le créateur d'entreprise virtuelle identifie les rôles et les processus de fabrication à distribuer, puis envoie un agent courtier sur la place de marché virtuelle.
2. **Rencontre des agents** : Pendant ce temps, d'autres compagnies envoient leurs propres agents représentants sur la place de marché, créant ainsi un environnement multi-agents dynamique.
3. Négociations sophistiquées : Les agents engagent des négociations complexes, échangeant des propositions, des contre-propositions et des arguments.
4. **Sélection** : L'agent courtier analyse les différentes offres et choisit un partenaire qui correspond le mieux à ses critères.
5. **Continuation ou retour** : L'agent peut soit continuer à chercher d'autres partenaires pour d'autres besoins, soit retourner auprès de son mandant pour présenter les résultats obtenus.

Figure 5.1 illustre les étapes d'une négociation automatique entre des agents sur une place de marché: Initialisation du processus de négociation -> déploiement des agents sur le marché -> rencontre des agents -> négociation sophistiquée -> sélection -> continuation ou retour.

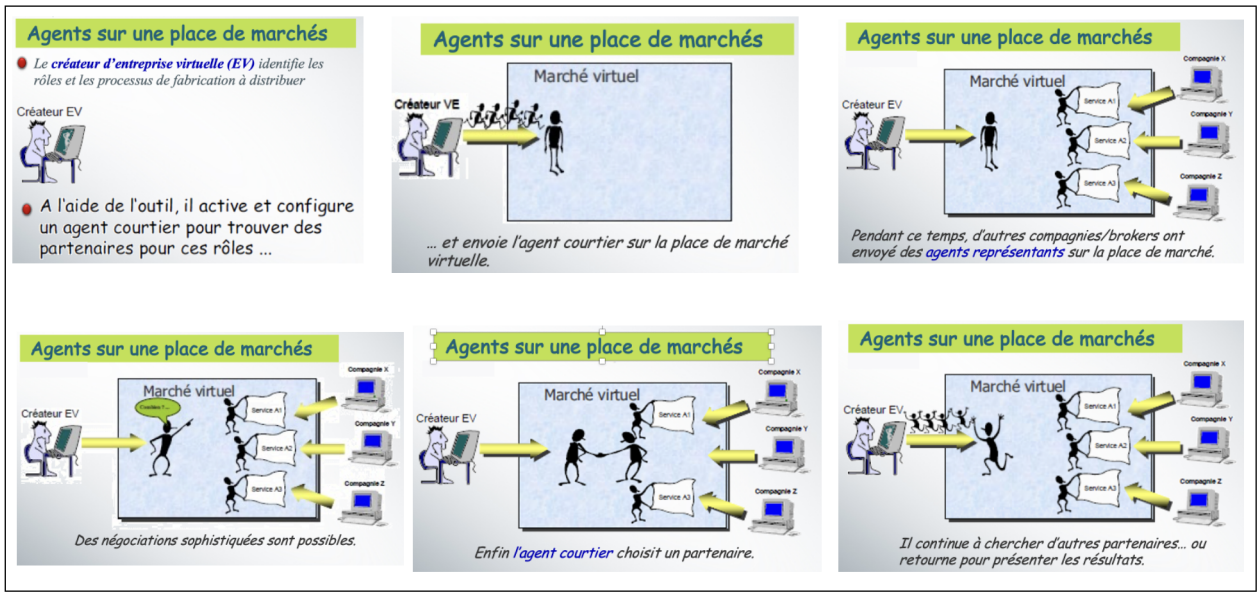


Figure 5.1: Scénarios d'une Négociation Automatique

5.4 Fondements de la Négociation dans les SMA

Dans les processus de négociation, on distingue principalement deux types d'agents négociateurs [21], comme le montre la figure 5.2:

- **Agents coopératifs** : Ces agents partagent des buts communs et coopèrent pour satisfaire leur objectif collectif. La négociation vise ici à optimiser la satisfaction globale du groupe.

- **Agents égocentrés** : Ces agents poursuivent des buts individuels et utilisent la négociation comme un mécanisme de coordination pour maintenir un comportement cohérent au sein du système, tout en cherchant à maximiser leur propre utilité.

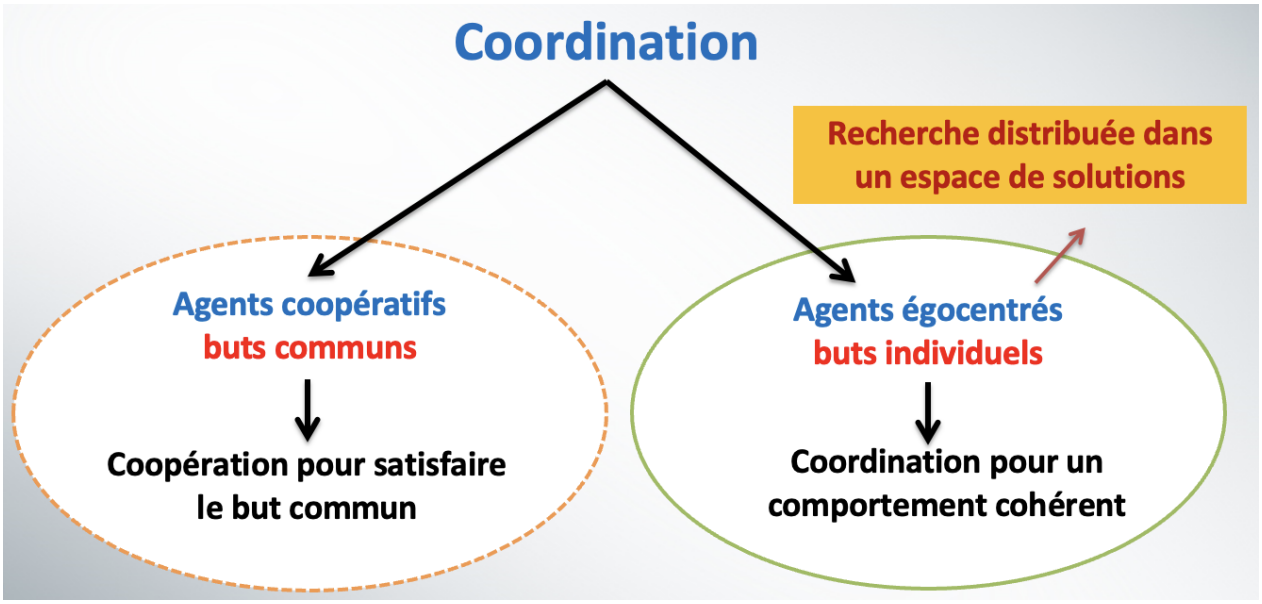


Figure 5.2: types d'Agents dans une Négociation Automatique

5.4.1 Composantes d'une Négociation

Une négociation automatique repose sur plusieurs composantes essentielles. Chaque aspect représente un domaine de recherche à part entière, mais tous contribuent à la richesse et à l'efficacité du processus de négociation.

5.4.1.1 Protocoles de Négociation

Ils représentent l'ensemble des règles qui structurent l'échange, déterminant qui peut parler, quand, et comment. Ils spécifient les actions valides possibles et déterminent comment les interactions peuvent prendre place.
– **Éléments constitutifs d'un protocole** :
- **Participants possibles** : acheteurs, offreurs, tierces parties (médiateurs, facilitateurs);
- **États de négociation** : offering (en cours d'offre), accepting proposal (acceptation de proposition), negotiation closed (négociation terminée);
- **Événements de transition** : fin des offres, expiration d'un délai, acceptation ou rejet;
- **Actions valides** : accept (accepter), reject (rejeter), counter-propose (contre-proposer).

■ **Exemple 5.2 Protocole d'échange de propositions**
Dans ce protocole, les agents échangent des propositions qui peuvent prendre différentes formes : une solution complète, une partie de solution, un ensemble de solutions acceptables, ou un ensemble de parties de solutions. Une proposition peut être générée de plusieurs manières :

- Indépendamment, sur la base des connaissances propres de l'agent
- À partir des propositions précédentes de l'agent lui-même ou des autres agents
- À partir des critiques formulées par d'autres agents sur ses propositions ou sur d'autres propositions.

La figure 5.3 illustre les étapes d'un protocole de négociation automatique.

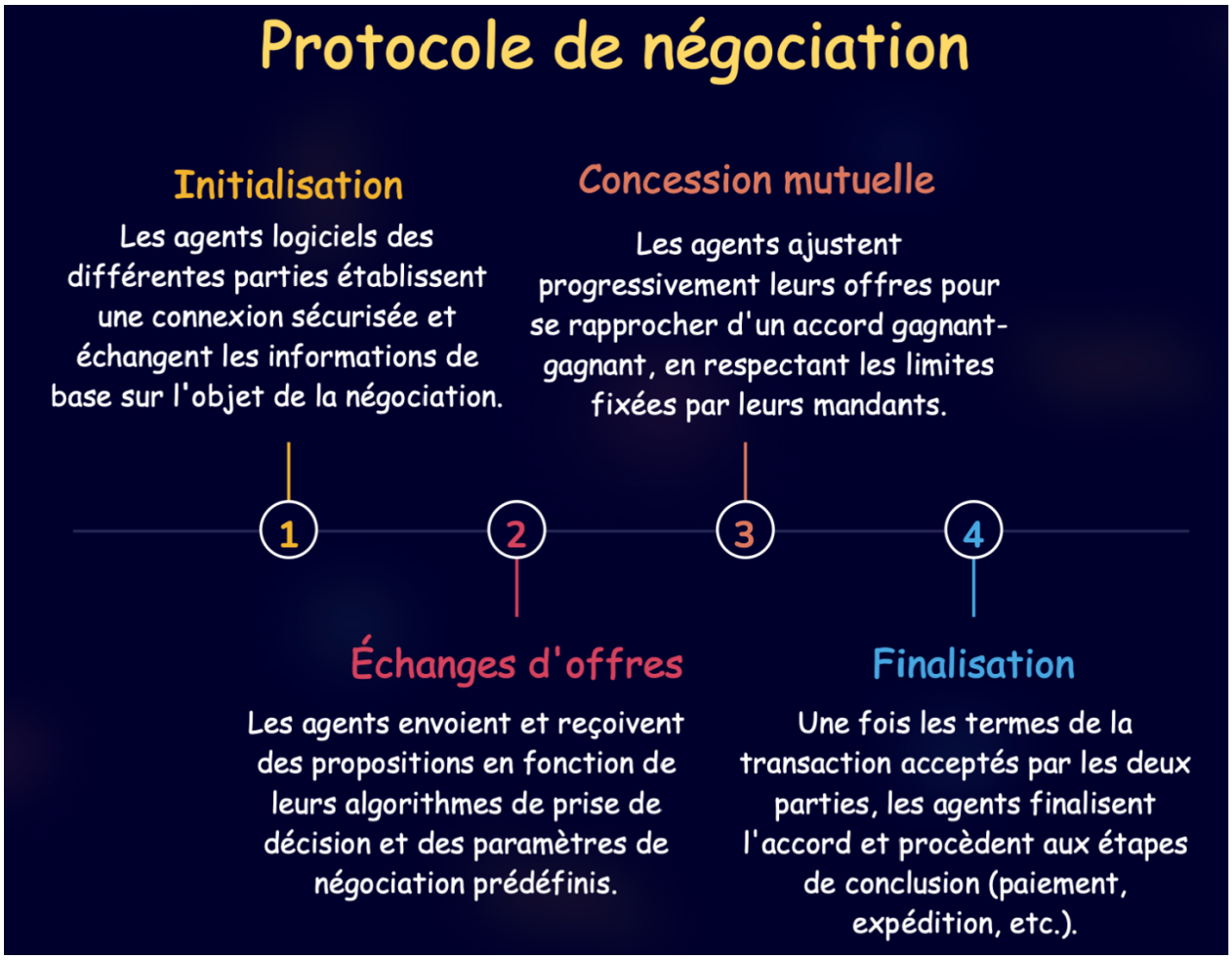


Figure 5.3: Etapes d'un Protocole de Négociation Automatique

5.4.1.2 Types de Protocoles

Il existe plusieurs types de protocoles :

- **Négociation bilatérale (un à un)** : deux agents échangent des offres et contre-offres jusqu'à un accord (ex. un acheteur et un vendeur).

- **Négociation multilatérale (plusieurs à plusieurs)** : plusieurs agents interagissent simultanément (ex. enchères).
- **Protocole du marchandage** : les agents se font des concessions successives, souvent basé sur un délai limite.
- **Contract Net Protocol (CNP)** : un agent "**manager**" annonce une tâche, et d'autres agents soumettent des propositions ; le manager choisit la meilleure. Ce protocole est très utilisé dans la planification distribuée.
- **Enchères** :
  - Anglaise : prix croissant jusqu'à ce qu'un seul enchérisseur reste.
  - Hollandaise : prix décroissant jusqu'à acceptation.
  - Vickrey (second prix scellé) : chaque agent fait une offre, le plus haut gagne mais paie le prix du second.
  - Enchères combinatoires : les agents enchérissent sur des lots de ressources.

5.4.1.3 Objets de Négociation

L'objet de la négociation représente ce sur quoi porte l'interaction (prix, quantité, ressources, délais, qualité de service). Il peut être de différentes natures:

- **Valeurs discrètes ou continues** : prix fixes ou dans un intervalle;
- **Éléments complexes** : plans d'action, intervalles de temps, répartition de tâches;
- **Issues simples ou multiples** : négociation sur un seul critère (temps) ou sur plusieurs critères simultanément (temps, prix, qualité, pénalités).

Les opérations applicables sur les objets sont :

- **Niveau rigide** : la structure et le contenu de l'accord sont fixes, seule l'acceptation ou le refus est possible (négociation binaire);
- **Niveau flexible** : la structure et le contenu peuvent être modifiés via des contre-propositions;
- **Niveau extensible** : il y a possibilité d'étendre la structure même de l'objet de négociation en cours de processus.

5.4.2 Stratégies de Négociation dans les Systèmes Multi-Agents

Il s'agit des méthodes et des algorithmes utilisés par les agents logiciels pour prendre des décisions optimales [24]. Les agents logiciels utilisent des algorithmes de négociation avancés pour développer une stratégie optimale [25]. Ils analysent les offres des différentes parties, évaluent les concessions possibles et ajustent leurs propositions en temps réel pour parvenir à un accord mutuellement bénéfique.

Ces algorithmes s'appuient sur des modèles économiques, des théories des jeux et de l'intelligence artificielle pour simuler les comportements des négociateurs humains et anticiper les meilleures décisions à prendre. Ainsi, la négociation automatique permet d'obtenir des résultats plus objectifs et équitables.
Il existe plusieurs stratégies dont les plus utilisées et les plus répandues seront detaillées dans ce qui suit:

5.4.2.1 Stratégies basées sur le temps (Time-dependent)

Dans ces stratégies, le niveau de concession est principalement ajusté en fonction du temps écoulé. Leur grand avantage est leur simplicité et leur prévisibilité. La fonction de concession est souvent

régie par une formule mathématique incluant un paramètre β qui définit le profil de la stratégie :

- **Boulware (ou Tough)** : Avec $\beta < 1$, l'agent fait très peu de concessions jusqu'à ce que le temps soit presque écoulé, puis concède brusquement. Cette stratégie est risquée mais peut conduire à des accords très favorables si l'adversaire est pressé.
- **Concéder (ou Soft)** : Avec $\beta > 1$, l'agent concède rapidement dès le début de la négociation pour montrer sa bonne volonté et maximiser les chances d'accord. Elle est utilisée lorsqu'un accord rapide est prioritaire.

5.4.2.2 Stratégies basées sur le comportement (Behaviour-dependent ou Imitative)

Ces stratégies se basent sur le comportement de l'opposant. Elles réagissent directement aux actions de l'adversaire. Elles ne nécessitent pas de modèle a priori de ses préférences. Parmi ces stratégies, on distingue:

- **Tit-for-Tat (Œil pour œil)** : L'agent reproduit le niveau de concession que l'adversaire a montré dans son dernier mouvement. Si l'autre concède, il concède ; s'il reste ferme, il reste ferme. C'est une stratégie robuste, réputée pour favoriser la coopération.

  ▪ **Exemple 5.3** Exemple : Si l'autre fait une concession de 10 %, je concède aussi 10 %.

- **Average TFT** : Une variante qui calcule la concession sur la moyenne des derniers comportements de l'adversaire, lissant ainsi les réactions.
- **Absolute TFT** : L'agent imite le changement absolu de l'utilité de l'offre de l'adversaire, plutôt que le changement relatif.

5.4.2.3 Stratégies basées sur les modèles (Model-based)

Ces stratégies sont les plus sophistiquées. L'agent tente d'apprendre et de construire un modèle des préférences ou de la stratégie de son adversaire à partir de l'historique des offres. Une fois le modèle estimé, l'agent peut calculer l'offre qui maximise sa propre utilité tout en ayant une forte probabilité d'être acceptée. Ces approches font souvent appel à des techniques d'apprentissage bayésien ou d'apprentissage par renforcement pour affiner leurs prédictions en temps réel.

▪ **Exemple 5.4** Un agent acheteur apprend que le vendeur ne descend jamais en dessous de 80 €, et arrête de proposer des prix inférieurs.

5.4.2.4 Stratégies basées l'Apprentissage par Renforcement (RL)

Les agents considèrent la négociation comme une séquence d'actions (proposer, accepter, refuser) [25]. Ils reçoivent une récompense en fonction de l'accord final. À force d'interactions, ils apprennent la meilleure politique pour maximiser leurs gains [26].

▪ **Exemple 5.5** Dans un marché de l'énergie, un agent apprenant ajuste ses offres en observant les réponses des autres consommateurs/producteurs [22].

Il convient de noter que le choix et le réglage d'une stratégie dépendent étroitement du contexte de la négociation (compétitif, collaboratif, pression temporelle, etc.). Une tendance actuelle consiste à hybrider ces approches, par exemple en utilisant une stratégie basée sur le temps comme comportement par défaut, qui est combinée avec une stratégie d'imitation si l'agent détecte un pattern coopératif chez son adversaire.

■ **Exemple 5.6** Utiliser une concession linéaire comme stratégie de base, mais ajuster le rythme de concession grâce à un modèle d'apprentissage supervisé.

5.5 Modèles de Négociation

Il existe trois paradigmes majeurs pour modéliser une négociation: la théorie des jeux, la négociation par argumentation, et les mécanismes d'enchères [20, 21]. Dans ce qui suit, nous présentons brièvement chacun de ces paradigmes.

5.5.1 Négociation et Théorie des jeux

La théorie des jeux fournit le cadre formel fondamental pour modéliser et analyser les interactions stratégiques entre agents rationnels [21, 23]. Dans ce paradigme, la négociation peut être vue comme un jeu où chaque agent cherche à maximiser son utilité.
Les concepts clés de ce modèle sont :

- **L'utilité** : Chaque agent possède une fonction d'utilité privée qui quantifie sa satisfaction pour chaque issue possible. **La rationalité** : Les agents cherchent systématiquement à maximiser leur utilité espérée.
- **Jeux coopératifs et non coopératifs** : dans un **jeu coopératif,** les agents peuvent former des coalitions et partager les gains (ex. deux entreprises s'allient pour un projet). Cependant, dans un **jeu non coopératif**, chaque agent agit seul pour maximiser son utilité, sans possibilité de coalition formelle (ex. négociation bilatérale entre un acheteur et un vendeur).
- Jeux à somme nulle et somme non nulle :
  - Dans un jeu à somme nulle, le gain d'un agent équivaut à la perte de l'autre (ex. un duel aux enchères).
  - Dans un jeu à somme non nulle, un compromis peut bénéficier aux deux parties (ex. négociation d'un contrat commercial).

  ■ **Exemple 5.7** Deux entreprises négocient la co-utilisation d'un entrepôt. En partageant l'espace intelligemment, elles peuvent toutes deux réduire leurs coûts → situation gagnant-gagnant.

- **Équilibre de Nash** : c'est une situation stable où aucun agent ne peut améliorer son sort en modifiant unilatéralement sa stratégie.
- **Optimalité de Pareto** : Un accord est **Pareto-optimal** s'il est impossible d'améliorer l'utilité d'un agent sans détériorer celle d'un autre.

  ■ **Exemple 5.8**
  – Accord A : Agent 1 = 80, Agent 2 = 60
  – Accord B : Agent 1 = 70, Agent 2 = 70
  – L'accord B est Pareto-meilleur car il améliore le gain de l'Agent 2 sans pénaliser l'Agent 1 de façon excessive.

Les modèles basés sur la théorie des jeux sont très utilisés dans le partage de ressources, la négociation de prix ou dans les systèmes robotiques. Cependant, ils présentent plusieurs limites dues

essentiellement au fait qu'ils supposent souvent des agents parfaitement rationnels avec une connaissance complète, ce qui est rarement réaliste. Or, dans la pratique, les agents disposent d'informations incomplètes et incertaines. Ceci a suscité l'intérêt d'approches hybrides qui combinent théorie des jeux et apprentissage automatique pour ajuster dynamiquement les stratégies.

5.5.2 La Négociation par Argumentation

Dans ce paradigme, la négociation ne se limite pas à seulement un échange d'offres, mais aussi de justifications (arguments) [27, 28]. Un agent tente de persuader l'opposant en ajoutant des arguments pour supporter une offre ou une contre-offre. On distingue plusieurs types d'arguments tels que les menaces ou les récompenses [28].

■ **Exemple 5.9** « Si vous acceptez ce prix, je m'engage à vous donner la priorité sur mes prochaines ventes ». Ou au contraire : « Si vous refusez, je me tournerai vers un autre fournisseur ».

L'argument permet :

- de modifier la région de l'acceptabilité d'un accord, les seuils de l'opposant, etc.
- d'augmenter la probabilité d'un accord, la vitesse d'obtention d'un accord, etc.

■ **Exemple 5.10**

- **Offre classique** : « Je propose 100 € ».
- **Offre avec argumentation** : « Je propose 100 €, car la livraison inclut un service supplémentaire de garantie d'un an ».

Pour construire un négociateur capable d'argumenter, il faut disposer de :

- Mécanismes pour l'échange de propositions et d'arguments;
- Techniques pour générer des propositions, des contre-propositions, des critiques et fournir les arguments correspondants.
- Techniques pour accepter les propositions (contre-propositions, critiques) et les arguments associés.
- Techniques pour répondre aux propositions (contre-propositions, critiques) et aux arguments associés.

Un protocole de négociation par argumentation définit quand et comment les arguments peuvent être échangés. Un cycle typique de ce protocole :

1. Un agent fait une offre.
2. L'autre l'accepte, la rejette ou propose une contre-offre.
3. En cas de refus, un argument est fourni pour justifier le rejet.
4. L'autre agent adapte sa prochaine proposition en fonction de cet argument.

■ **Exemple 5.11**

- Agent A : « Je propose 120 € »
- Agent B : « Non, car mon budget maximal est 100 € »
- Agent A : « D'accord, je peux baisser à 100 €, mais uniquement si la livraison est en deux semaines ».

La négociation par argumentation présente plusieurs avantages entre autres :

- **Plus grande transparence** : les agents comprennent mieux les contraintes de l'autre.
- **Accords de meilleure qualité** : en tenant compte des justifications, on obtient souvent des solutions Pareto-optimales.
- **Réduction du temps de négociation** : moins d'essais et d'erreurs, car les refus sont expliqués.
- **Compatibilité avec les environnements humains** : les arguments sont proches de la communication humaine, facilitant les interactions homme-agent.

Ce paradigme est très utilisé dans plusieurs domaines tels que : le e-commerce, la santé, le transport, les smart grid, etc. Cependant ils présentent plusieurs limites notamment la complexité des modèles, la standardisation ou la charge computationnelle élevée.

5.5.3 La Négociation par Mécanismes d'Enchères

Les enchères sont un processus de négociation compétitif où des acheteurs et vendeurs échangent des offres successives jusqu'à ce que le meilleur prix soit atteint [20]. Le but d'une enchère est de maximiser le profit du vendeur et de minimiser les coûts d'un acheteur [21]. Ce mécanisme permet d'optimiser les transactions en identifiant la valeur réelle du bien ou du service négocié.
Ce modèle formalise les interactions selon un protocole prédéfini où les participants expriment leurs préférences via des engagements fermes.
Un mécanisme d'enchères se définit par quatre éléments fondamentaux :

- Un ensemble de biens à allouer (ressources, services ou objets)
- Un ensemble d'enchérisseurs (agents acheteurs) possédant des valuations privées
- Un commissaire-priseur (agent vendeur ou allocateur) supervisant le processus
- Des règles déterminant le format des offres (cri public, sceau fermé), la fonction d'allocation (qui gagne) et la règle de paiement (à quel prix).

– Typologie des Enchères Fondamentales :
Les principaux types d'enchères sont :

- **Enchère Anglaise (montante)** : Les enchères anglaises sont un format d'enchères traditionnelles où les offres des acheteurs augmentent progressivement jusqu'à ce que le prix le plus élevé soit atteint. Dans ce processus, les participants enchérissent ouvertement les uns contre les autres, permettant à tous de suivre l'évolution des offres en temps réel.
  Ce type d'enchères se déroule généralement dans un cadre formel, comme une salle des ventes aux enchères, où l'enchérisseur le plus offrant remporte le lot. Ce mécanisme favorise la transparence et la compétition saine entre les acheteurs, garantissant ainsi l'obtention du juste prix du marché.
- **Enchères hollandaises (descendante)** : Les enchères hollandaises, également connues sous le nom d'enchères à la baisse, sont un modèle d'enchères inverse où le prix commence à un niveau élevé et diminue progressivement jusqu'à ce qu'un acheteur accepte l'offre. Ce mécanisme crée une compétition entre les acheteurs potentiels pour obtenir le bien au meilleur prix.
  Contrairement aux enchères anglaises, les enchères hollandaises sont généralement utilisées pour la vente de produits périssables comme les fleurs. Ce processus permet de vider rapidement les stocks et d'optimiser la valeur des produits sur le marché. La figure 5.4 illustre le principe des enchères anglaises et hollandaises.
- **Enchère au premier prix scellé** : Les offres sont secrètes et simultanées. Le bien revient à l'agent ayant fait la meilleure proposition, qui paie le prix de son offre.

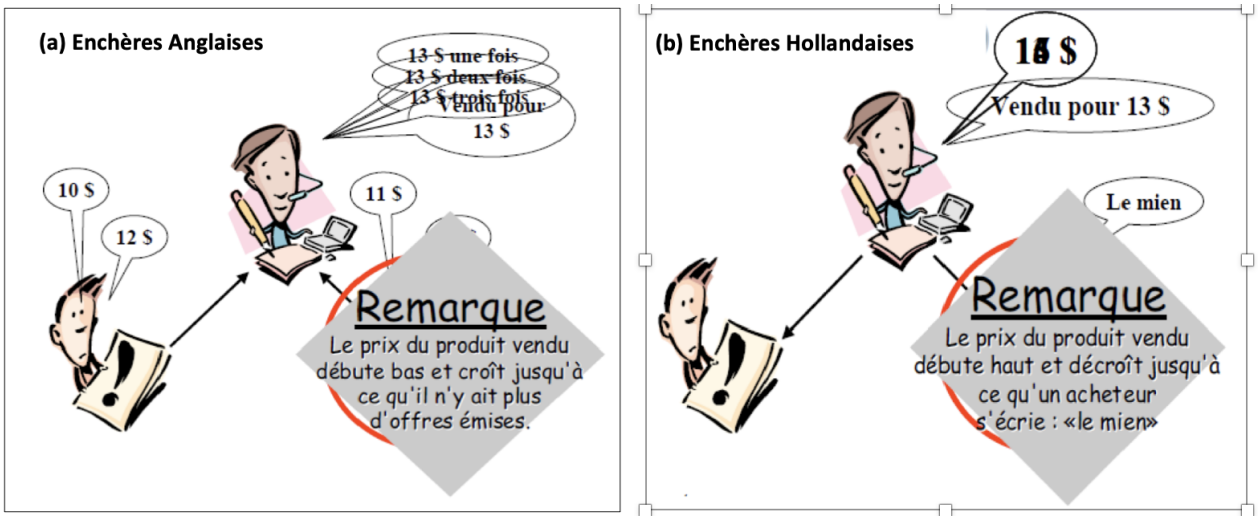


Figure 5.4: Enchères Anglaises vs Enchères Hollandaises.

- **Enchère de Vickrey (Second Prix scellé)** : Chaque agent soumet secrètement une offre. Le bien revient au plus offrant, mais il paie le prix de la deuxième meilleure offre. Le but est d'encourager les agents à révéler leur vraie valorisation du bien.

Les enchères sont très utilisées notamment dans le commerce électronique (plateformes comme eBay, Amazon Marketplace), l'allocation de ressources dans les réseaux (bande passante, calcul distribué, énergie), les smart grids pour la répartition d'électricité et les marchés financiers artificiels simulés entre agents.

5.6 Conclusion

La négociation automatique occupe une place centrale dans les systèmes multi-agents, car elle offre un cadre structuré pour résoudre les conflits d'intérêts, coordonner les comportements et parvenir à des accords mutuellement acceptables dans des environnements distribués et dynamiques [20, 24]. Différentes approches – de la négociation bilatérale aux enchères, en passant par l'argumentation et les mécanismes issus de la théorie des jeux – permettent d'adresser une large variété de contextes et de besoins [23, 21]. L'étude de cas sur la gestion énergétique dans les Smart Grids a montré comment ces mécanismes trouvent des applications concrètes [22] et souligne l'utilité réelle de ces mécanismes dans des environnements informatiques distribués.

5.7 Exercices

Exercice 5.1 Etude de cas: Allocation d'énergie dans un Smart Grid basé sur la négociation multi-agents

1. **Contexte:** Dans les réseaux électriques intelligents (Smart Grids), la production et la consommation d'énergie doivent être équilibrées en temps réel. Avec l'intégration des énergies renouvelables (solaire, éolien), le réseau devient plus dynamique et incertain. Plusieurs acteurs interviennent :

- Producteurs d'énergie (centrales, panneaux solaires, éoliennes),
- Consommateurs (foyers, entreprises, véhicules électriques),
- Opérateurs de stockage (batteries, stations hydrauliques).

Ces acteurs ont des intérêts différents : maximiser leur gain, minimiser leurs coûts ou garantir leur autonomie énergétique.

2. **Problématique** : Comment répartir efficacement l'énergie disponible entre plusieurs consommateurs et producteurs, en respectant les contraintes (capacité limitée, pics de demande, coûts variables), tout en maintenant un réseau stable et équilibré ?
3. **Approche par négociation multi-agents** : Les agents représentent :
   - **Agents producteurs** : proposent de l'énergie à un certain prix.
   - **Agents consommateurs** : soumettent des offres en fonction de leur besoin et de leur budget.
   - **Agent régulateur (facultatif)** : supervise le processus et s'assure que les accords respectent la stabilité du réseau.

   Deux mécanismes de négociation peuvent être utilisés :
   (a) **Négociation bilatérale** → Un consommateur peut négocier directement avec un producteur pour obtenir de l'énergie à un prix acceptable.
   (b) **Enchères multi-agents** → Les producteurs organisent une enchère (ex. enchère ascendante ou Vickrey) pour vendre leur surplus d'énergie aux consommateurs.
4. **Exemple pratique** :
   - À 19h, plusieurs foyers augmentent leur demande d'électricité (cuisine, chauffage, recharge de véhicules électriques).
   - Les producteurs d'énergie éolienne n'ont qu'une capacité limitée à cause d'un vent faible.
   - Une enchère est lancée : chaque foyer-agent soumet une offre.
   - Le foyer prêt à payer le plus obtient la priorité, mais d'autres accords bilatéraux se négocient avec les batteries de stockage.
5. **Résultats attendus**
   - Allocation efficace de l'énergie disponible,
   - Réduction des coupures de courant,
   - Optimisation des coûts pour les producteurs et consommateurs,
   - Satisfaction globale améliorée grâce à une négociation décentralisée.

Exercice 5.2 **Partage d'une ressource entre deux agents**.

1. **Contexte** : Deux agents autonomes, A et B, doivent se partager une ressource indivisible (par exemple : un canal de communication unique, un créneau horaire, ou une unité d'énergie). Ils ont le choix entre deux stratégies :
   - **Haut (H)** : réclamer fortement la ressource (comportement agressif).
   - **Bas (B)** : réclamer faiblement / coopérer (comportement conciliant).

   Les gains (utilités) résultant de leurs choix sont donnés par la matrice suivante — la première valeur est l'utilité d'A, la seconde celle de B :

A \ B	H	B
H	(2, 2)	(4, 1)
B	(1, 4)	(3, 3)

Interprétation rapide :

- Si les deux jouent H (conflit), chacun obtient 2 (coût dû à la compétition).
- Si A joue H et B joue B, A « gagne » (4) tandis que B obtient 1.
- Si les deux jouent B (coopération), chacun obtient 3 (bon compromis).
- Symétrie entre A et B.

2. **Questions :**
   - Identifie les stratégies dominantes (s'il y en a) pour chaque joueur.
   - Trouve tous les équilibres de Nash purs.
   - Parmi les quatre issues, lesquelles sont Pareto-optimales ?
   - Suppose maintenant que les joueurs peuvent randomiser (stratégies mixtes). Trouve une stratégie mixte (probabilités) pour A et B de sorte que chaque joueur soit indifférent entre H et B. Calcule ces probabilités.
   - Discussion : quel résultat (équilibre pur ou mixte) est le plus souhaitable du point de vue collectif ? Comment imposer/encourager ce résultat dans un SMA (idées pratiques) ?

Corrigé de l'exercice:

1. **Stratégies dominantes** :
   - **Pour A :**
     - Si B joue H : utilité(A|H)=2, utilité(A|B)=1 → H > B.
     - Si B joue B : utilité(A|H)=4, utilité(A|B)=3 → H > B.

     → A a H dominant (H domine strictement B).
   - **Pour B (symétrie) :**
     - Si A joue H : utilité(B|H)=2, utilité(B|B)=1 → H > B.
     - Si A joue B : utilité(B|H)=4, utilité(B|B)=3 → H > B. → B a H dominant aussi.

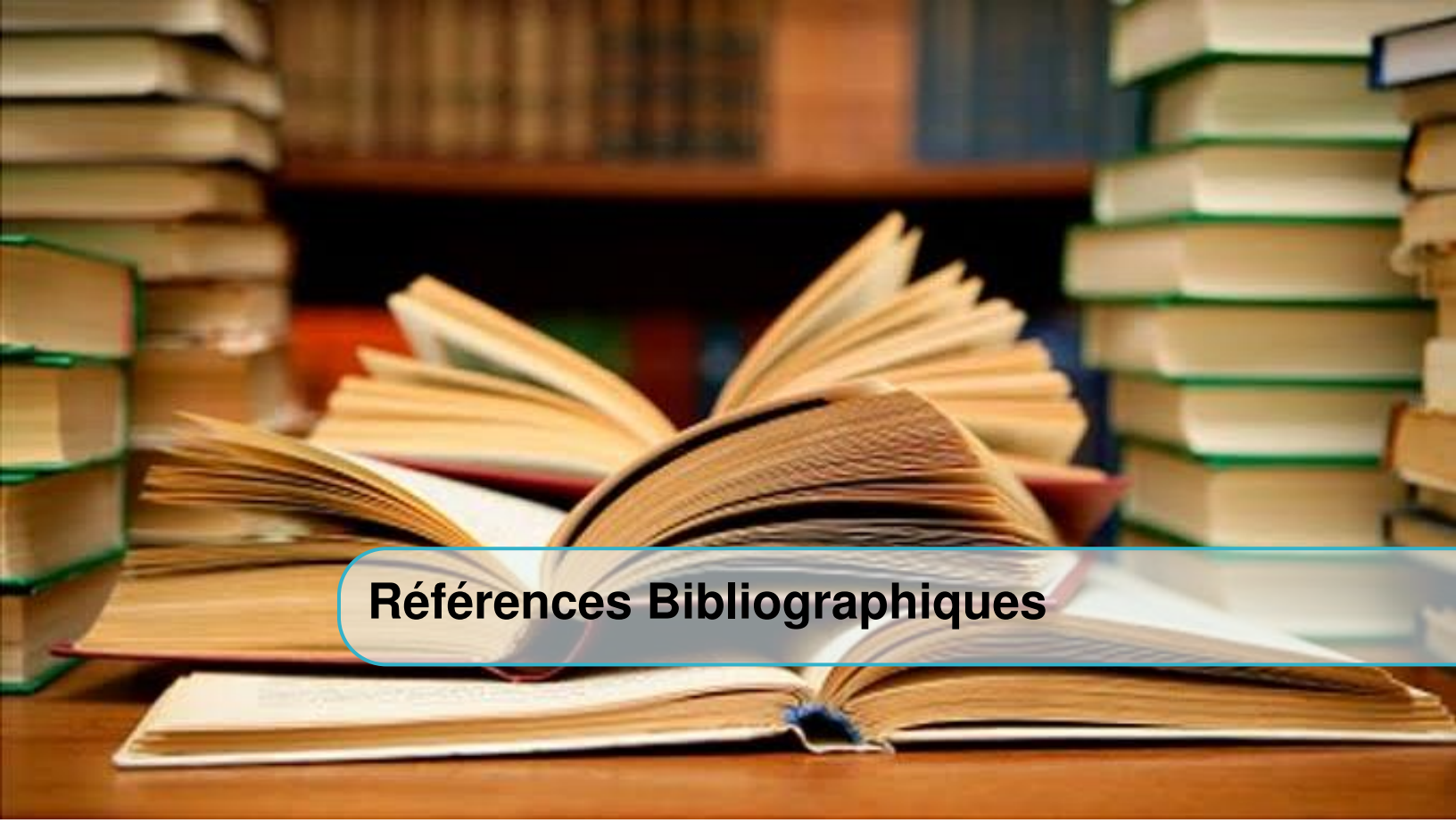
   Conclusion Q1 : Les deux ont une stratégie dominante H.
2. **Équilibres de Nash purs**
   Un profil est un équilibre de Nash si aucun joueur ne peut augmenter son utilité en déviant unilatéralement.
   Puisque H est dominant pour A et pour B, le profil (H, H) est un équilibre (chacun joue sa stratégie dominante). Vérifions les autres profils rapidement :
   - (H, B) : B peut dévier de B→H et passer de util(B)=1 → util(B|H)=2 (améliore), donc (H,B) n'est pas un EN.
   - (B, H) : symétrique, pas EN.
   - (B, B) : A peut dévier B→H : util(A) passe 3 → 4, donc (B,B) n'est pas EN.

     Conclusion Q2 : L'unique équilibre de Nash pur est (H, H).
3. **Optimalité de Pareto** :
   Une issue est Pareto-optimale si on ne peut augmenter l'utilité d'un joueur sans diminuer celle de l'autre. Regardons les 4 résultats :
   - (H,H) → (2,2)

- (H,B) $\rightarrow$ (4,1)
- (B,H) $\rightarrow$ (1,4)
- (B,B) $\rightarrow$ (3,3)

Comparer :

- **(B,B) = (3,3)** : on ne peut pas améliorer l'un sans réduire l'autre $\Rightarrow$ **Pareto-optimal**.
- **(H,B) = (4,1)** : on peut passer à (3,3) et améliorer B ($1 \rightarrow 3$) tout en diminuant A ($4 \rightarrow 3$) — ce n'est pas une amélioration conjointe.
  Est-ce que (H,B) est dominé par un autre profil ? Comparons avec (B,B) donnant (3,3):
  $(4,1)$ vs $(3,3)$ $\Rightarrow$ B s'améliore, A diminue $\Rightarrow$ pas de domination stricte.
  Mais (H,B) n'est pas Pareto-dominant car (B,B) améliore B même si A diminue.
  La Pareto-optimalité implique l'absence d'un autre profil strictement meilleur pour les deux joueurs. Aucun profil ne domine (H,B) simultanément pour A et B.
  Testons : existe-t-il un profil (x,y) tel que $x \geq 4$ et $y \geq 1$ avec au moins une inégalité stricte ? Non.
  Donc **(H,B) est Pareto-optimal**. (Même raisonnement pour (B,H)).
- **(H,H) = (2,2)** : le profil (B,B) = (3,3) domine (H,H), car $3 \geq 2$ et $3 > 2$.
  $\Rightarrow$ **(H,H) n'est pas Pareto-optimal.**
  Conclusion Q3 : Pareto-optimaux : (B,B), (H,B), (B,H).



Références Bibliographiques

Ces références bibliographiques répertorient les ouvrages de référence, articles fondateurs et ressources clés qui ont inspiré la rédaction de ce support de cours. Elle est destinée aux étudiants souhaitant approfondir leurs connaissances sur les sujets abordés.

Chapitre 1 : Introduction : de l'IA à l'IAD

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Pearson, 2016 (cited on pages 9, 11, 40).

[2] David Poole and Alan Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, 2010 (cited on pages 9, 14).

[3] Elaine Rich and Kevin Knight. *Artificial Intelligence*. McGraw-Hill, 1991 (cited on page 11).

[4] Allen Newell. "Physical Symbol Systems". In: *Cognitive Science* 4.2 (1980), pages 135–183 (cited on pages 11, 18).

[5] Stuart Russell. "Rationality and intelligence". In: *Foundations of Rational Agency*. Springer, 1999, pages 11–33 (cited on pages 13, 18, 21, 27).

Chapitre 2 : Concepts fondamentaux des agents

[6] Michael Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2009 (cited on pages 17, 21, 22, 25–27, 34–36, 60).

[7] Jacques Ferber. *Les systèmes multi-agents: Vers une intelligence collective*. InterEditions, 1999 (cited on pages 17, 21, 25, 34, 36, 60, 64).

[8] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009 (cited on pages 21, 22, 37).

[9] Michael Wooldridge and Nicholas R Jennings. “Intelligent Agents: Theory and Practice”. In: *The Knowledge Engineering Review* 10.2 (1995), pages 115–152 (cited on pages 21, 22, 24, 26, 34, 36).

[10] Nicholas R. Jennings and Michael Wooldridge. “Applications of Intelligent Agents”. In: *Agent Technology*. Springer, 1998, pages 3–28 (cited on pages 22, 35, 60, 63).

Chapitre 3 : Modèles et architectures des agents

[11] Gerhard Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999 (cited on pages 39, 40, 46, 49, 56).

[12] Michael Wooldridge. “Agent-Based Software Engineering”. In: *IEEE Transactions on Software Engineering* 144.1 (2002), pages 26–37 (cited on pages 39, 40, 48, 53, 56).

[13] Anand S Rao and Michael P Georgeff. “BDI Agents: From Theory to Practice”. In: *ICMAS*. Volume 95. 1995, pages 312–319 (cited on pages 40, 49, 53, 56).

[14] Patrick Taillandier, Olivier Therond, and Benoit Gaudou. “Une architecture d’agent BDI basée sur la théorie des fonctions de croyance: application à la simulation du comportement des agriculteurs”. In: *Journées Francophones sur les Systèmes Multi-Agents*. Cépaduès. 2012, pages 107–116 (cited on pages 49, 56).

Chapitre 4 : Les systèmes multi-agents

[15] Jacques Ferber. “Les systèmes multi-agents, vers une intelligence collective”. In: *InterEditions, Paris* 322 (1995) (cited on pages 22, 60, 64, 69).

[16] Jacques Ferber. “Les systèmes multi-agents: un aperçu général”. In: *Techniques et Sciences Informatiques* 16.8 (1997) (cited on pages 60, 61, 64, 69, 78, 97).

[17] Jean-Pierre Briot. *Principes et architecture des systèmes multi-agents*. Hermès Science, 2001 (cited on pages 60, 61, 64, 69, 78, 97).

[18] Nicholas R. Jennings, Katia Sycara, and Michael Wooldridge. “A Roadmap of Agent Research and Development”. In: *Autonomous Agents and Multi-Agent Systems* 1.1 (2000), pages 7–38 (cited on pages 63, 69, 97).

[19] Nicolas Ravier and Jean-Paul Jamont. *Multi-Agent Systems: Foundations and Development*. Wiley, 2023 (cited on page 97).

Chapitre 5 : Négociation automatique

[20] Sarit Kraus. *Strategic Negotiation in Multiagent Environments*. MIT Press, 2001 (cited on pages 99, 105, 107, 108).

[21] Noam Nisan et al. *Algorithmic Game Theory*. Cambridge University Press, 2007 (cited on pages 99, 101, 105, 107, 108).

[22] Xiaofeng Wang, Jing Li, and Wei Zhang. “Smart Grid Energy Trading: A Multi-Agent Negotiation Framework”. In: *Applied Energy* 350 (2023), pages 121–135 (cited on pages 99, 104, 108).

[23] Jeffrey S. Rosenschein and Gilad Zlotkin. “Rules of Encounter: Designing Conventions for Automated Negotiation among Computers”. In: *MIT Press* (1994) (cited on pages 99, 105, 108).

[24] Tim Baarslag et al. *Automated Negotiation: A Guide for Researchers and Practitioners*. Springer, 2022 (cited on pages 99, 103, 108).

[25] Martin N. Achenmpong, Hongyang Wang, and Fei Li. “Deep Reinforcement Learning for Automated Negotiation: A Systematic Review”. In: *IEEE Transactions on Artificial Intelligence* 4.2 (2023), pages 45–62 (cited on pages 103, 104).

[26] Dhruv Shah and Jaspreet Kaur. “Transformer-based Negotiation Agents for Multi-Issue Bilateral Negotiation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36 (2022), pages 12589–12597 (cited on page 104).

[27] Iyad Rahwan and Liz Sonenberg. “The Role of Argumentation in Negotiation among Intelligent Agents”. In: *Artificial Intelligence* 22.2 (2003), pages 1–16 (cited on page 106).

[28] Sanjay Negi, Nir Oren, and Wamberto Vasconcelos. “Explainable AI in Argumentation-Based Negotiation Systems”. In: *Autonomous Agents and Multi-Agent Systems* 38.1 (2024), pages 1–28 (cited on page 106).