Democratic and Popular Republic of Algeria

Ministry of Higher Education and Scientific Research
Tahri Mohamed University of Béchar
Faculty of Exact Sciences
Department of Mathematics and Computer Science

Introduction to multi-agent systems (MAS)

Course

Prepared by: *Mostefa Bendjima*

Avril 2025

Chapter 01: Introduction to Distributed Artificial Intelligence

Chapter 02: Multi-Agent Systems - General Principles

Chapter 03 Agent Models and Architectures

Chapter 04: Cognitive Agent Coordination Models

List of figures

Introduction

A multi-agent system (MAS) consists of multiple interactive agents that operate autonomously or cooperatively to solve complex problems that are often beyond the capability of single agents or centralized systems. Each agent in a MAS is an independent entity with its own perceptions, reasoning capabilities, and action mechanisms, allowing it to process information, learn from experience, and make decisions. This paradigm is inspired by natural systems such as human societies, animal groups, or biological systems, where individual components work together, balance self-interest with the collective good, and adapt to changing conditions.

The autonomy of agents is a key feature in multi-agent systems, which differentiates them from traditional centralized AI methods. Rather than relying on a central controller to dictate actions, agents independently analyze the environment, pursue goals, and respond to other agents' behaviors. This leads to significant benefits including scalability, robustness, and flexibility. For example, in a large network of agents acting as sensors or robots, a failure of one agent generally does not cripple the entire system because control and responsibility are distributed.

Coordination and communication protocols between agents are essential in MAS. Agents exchange messages containing information, intentions, or requests, enabling them to align their activities. The level of cooperation among agents varies widely—the interactions can be synergistic, where agents collaboratively maximize a shared objective, or competitive, where agents strive to optimize their own interests potentially at the expense of others. Mechanisms such as negotiation, coalition formation, auctions, and voting are often implemented to facilitate coordination, resolve conflicts, and reach agreements in a decentralized manner.

Multi-agent systems find applications in numerous real-world domains. In robotics, MAS enables cooperative missions by swarms of drones or robots in areas like environmental monitoring, search and rescue operations, and agricultural automation. In transportation systems, intelligent traffic lights coordinated by MAS can optimize traffic flow in congested urban areas, and autonomous vehicles can negotiate right-of-way and routing for safer and more efficient journeys. Financial markets and e-commerce platforms leverage MAS to simulate market behaviors, automate trading agents, and enhance decision-making processes. Smart grid technologies utilize MAS for managing decentralized energy resources, balancing supply and demand in real time while integrating renewable energy sources.

Designing multi-agent systems presents several challenges. Ensuring that agents act reliably and securely is crucial, especially when some agents may be faulty, malicious, or selfish. Establishing trust mechanisms and incentives promotes cooperation and deters malicious behavior. Additionally, MAS must handle uncertainty and incomplete information since agents operate with partial knowledge of the environment and other agents' states. Techniques like probabilistic reasoning, distributed problem-solving, and adaptive learning are incorporated to enable agents to make informed decisions under uncertainty.

From a theoretical perspective, MAS research integrates concepts from game theory, distributed artificial intelligence, and complex systems. Game theory provides analytical tools for modeling strategic interactions, helping predict behaviors and outcomes in competitive or cooperative settings. Distributed AI extends traditional AI methodologies by emphasizing decentralized decision-making and knowledge sharing. Complex systems theory investigates emergent phenomena, where simple interactions among agents lead to sophisticated global behaviors that are not explicitly programmed but evolve from local dynamics.

Learning and adaptation are increasingly important in MAS to improve performance over time. Agents use machine learning and artificial intelligence techniques, including reinforcement learning, evolutionary algorithms, and neural networks, to optimize their strategies based on feedback from the environment and other agents. This adaptability is vital in dynamic environments such as stock markets or disaster zones, where conditions evolve and pre-defined rules may no longer be effective.

The future of multi-agent systems is promising with the rise of technologies like the Internet of Things (IoT), blockchain, and cloud computing. MAS can manage massive networks of smart devices, facilitate transparent and secure distributed transactions, and enable large-scale collaborative artificial intelligence systems. The combination of these technologies opens new possibilities for intelligent, autonomous systems that support decision-making, resource management, and complex task execution on unprecedented scales.

In summary, multi-agent systems represent a powerful paradigm for designing distributed, intelligent, and adaptive systems. By leveraging the autonomy and cooperation of multiple agents, MAS solve intricate problems more efficiently and resiliently than single-agent systems. Their applications span a wide range of fields, and ongoing research continues to expand their capabilities, addressing challenges related to coordination, trust, learning, and scalability. As technology progresses, MAS are poised to play a central role in the next generation of intelligent systems that interact seamlessly with the physical and social world.

1. Introduction

Multi-agent systems (MAS) are currently a very active and widely applied field. This field is a branch of Distributed Artificial Intelligence (DAI). It therefore seems obvious to begin this course with an introduction to distributed artificial intelligence. So we begin this chapter with a brief introduction to artificial intelligence, showing the need to distribute the techniques of this branch. Next, we present distributed artificial intelligence, detailing the causes and forms of intelligence distribution. Since multi-agent systems represent only one branch of distributed artificial intelligence, we present an overview of the three branches that make up this field in section 3. In Section 4, we clarify the differences between multi-agent systems and the main paradigms and similar computer systems. Of course, multi-agent systems don't just offer advantages. We therefore present the advantages and challenges of these systems, before outlining their areas of application. Finally, we present a conclusion and some questions for further understanding.

2. From Artificial Intelligence to Distributed Artificial Intelligence

The creation of intelligent machines has been one of mankind's dreams since antiquity. There are various traces of this statement in ancient Egypt or in the Iliad. However, most of these traces only take on a mythological form. The formal deduction proposed by Aristotle (*the syllogism*) can be seen as the first step towards automatic reasoning. This idea consists in the possibility of validating the truth of a conclusion from two premises. For example, if we have the premises "*All men are mortal*" and "*Mohamed is a man*" we can then conclude that "*Mohamed is mortal*". Without doubt, formal reasoning is one of the fundamental issues in artificial intelligence.

The invention of computers is just one way of realizing this dream. In fact, after the invention of computers, many researchers assumed that this dream can be realized if we can process the symbols using computers. However, a number of objections have been raised, sometimes concerning the very nature of intelligence. In fact, many philosophers consider intelligence to be a human characteristic (a characteristic linked to the human mind) that cannot be achieved by machines.

In 1950, in a famous article, Alain Turing proposed an empirical test to overcome the debate on the feasibility of intelligent machines. This test is known as *the "Turing* test". In this way, one person can test the machine via an interface. The test person asks the machine questions. A machine is considered intelligent if its responses are indistinguishable from those of a human being.

In 1956, the field of artificial intelligence was officially born at the Dartmouth Conference, with the main aim of *getting machines to do what humans can do*. This is a multidisciplinary field. It encompasses computer science, philosophy, psychology, robotics and linguistics. This field is both engineering and science. On the one hand, the aim is to create an intelligent machine (engineering). On the other hand, the aim is to create models that can be used to understand and validate cognitive science theorie.

Participants at the Dartmouth conference were very optimistic. They estimated that the realization of intelligent machines is a matter of a few years. As an example, in 1970 Minsky (one of the founders of AI) announced that the realization of a machine with the general intelligence of an ordinary human being would take from three to eight years. However, the complexity of the task showed the falsity of these estimates. For example, Newell and Simon declared in 1958 that after ten years the world chess champion would be a computer.

Towards the end of the 1970s, a new trend emerged. It consists in exploiting the advances made by distributed systems in artificial intelligence to overcome the complexity of AI problems. This has given rise to the field of distributed artificial intelligence (DAI).

3. Distributed artificial intelligence

In the late 1970s, a system called "*Hearsay II*" was developed at Carnegie-Mellon University for automatic speech recognition. This system is based on the use of a blackboard. As shown in Figure 1.1, this system is made up of a set of knowledge sources (KS) that share the blackboard. It is organized into several levels. In fact, an automatic speech recognition problem is characterized by the manipulation of knowledge at different levels of abstraction (signal, lexical entity, syntactic entity, etc.). In addition, each knowledge source is specialized in a specific task (such as segmentation, syllable creation, word construction, etc.). So, each knowledge source manipulates knowledge of a specific level and generates knowledge for other levels. Changes made on a blackboard level can activate certain knowledge sources. These must be inserted into a scheduling list.

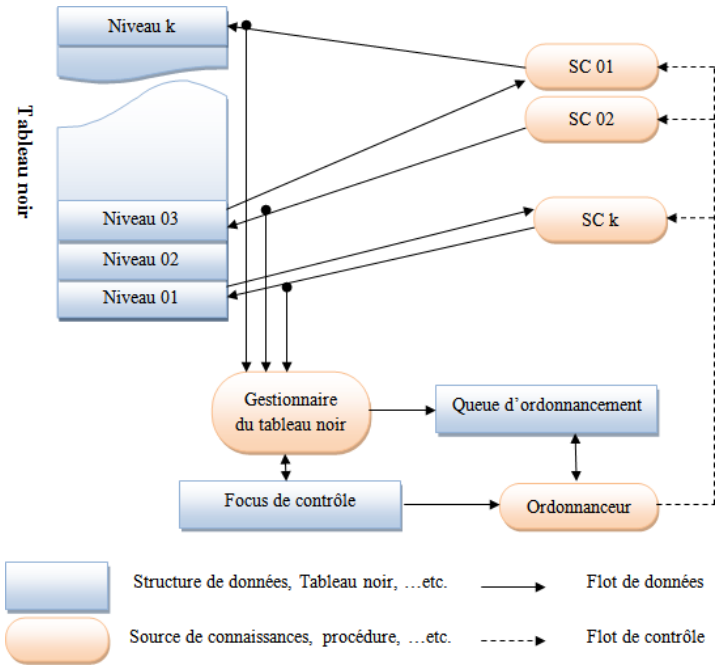

Figure 1.1: The HearsayII system.

In contrast to classical AI, this system is made up of a set of distributed knowledge sources. In fact, the Hearsay II system is one of the first IAD systems. As a result, we can define distributed artificial intelligence as a branch of classical artificial intelligence concerned with intelligent behavior produced by the activity of several entities. It is important to note that this branch does not deal with low-level parallelism, which are at the heart of the field of parallel and distributed systems.

The distribution of artificial intelligence can be justified by one of three reasons:

- **Functional distribution:** some tasks require multiple skills. Centralizing a wide range of expertise in a single entity complicates the task of designers. Taking the example of a space-exploring robot. Such a robot must be equipped with functions for navigation, obstacle avoidance, communication with the base, transporting found objects and analyzing them, etc. Clearly, designing a single entity with all these functions is a complex task. As a result, we can distribute tasks to more specialized entities. So we'll have a robot specialized in navigating and transporting objects, a robot for analyzing the objects found and a base for transferring the results to the ground base.

  This functional distribution is an essential feature of many real-world examples. In fact, we can see the distribution in design and diagnostic problems. In many cases, medical diagnosis requires the involvement of several experts. Each of these experts has his or her own knowledge and diagnostic methods. The general diagnosis is the conclusion of the diagnoses made by the various experts and the interactions between them. Consequently, such a system must be designed as a distributed system where each expert is represented by an entity.

- **Physical distribution:** There are several problems of a distributed physical nature. In this case, intelligence distribution becomes an obvious and natural solution. An example of such a problem is weather forecasting. In this problem, we need to analyze different parameters from several geographically distributed locations. In this way, each sensor can analyze the data collected locally and share this knowledge with other stations. Similarly, road traffic management is an example of a geographically distributed problem. In fact, a road traffic management system consists of analyzing and controlling the fluidity of road traffic in order to avoid traffic jams. Centralized control of this system is both a complex task and an inefficient solution, requiring the analysis of large quantities of data from several different sites. However, a driver in a given location (station, road, crossroads, etc.) concentrates on the places closest to his coordinates.

- **Computer distribution:** the success of distributed systems in the late 1970s made it possible to tackle problems initially considered unsolvable. A program capable of playing chess, for example, requires huge computing resources (memory space, computation time, etc.). However, the use of distributed systems means that the limits of machine capacity can be exceeded, because the solution can be run on several machines. In addition to efficiency, distribution offers the advantage of reliability. In a distributed system, the failure of one node has no influence on other parts of the system.

Intelligence distribution can take one of two forms:

- **Knowledge distribution:** in this case, knowledge is stored in different knowledge bases. In fact, some problems require the geographical distribution of knowledge. For example, sensors used to monitor atmospheric parameters can store the captured parameters in local knowledge bases to avoid the high cost of knowledge transfer. In addition, the distribution of knowledge can be justified by the difference in their levels of abstraction. Many problems require the manipulation of knowledge at different levels of abstraction. Automatic speech recognition systems (such as the Hearsay II system explained earlier) are typical examples. This type of application manipulates the signals picked up by microphones, syllables, words and phrases. This distribution increases efficiency by allowing knowledge sources to manipulate only the appropriate levels. Finally, knowledge distribution can influence system reliability by eliminating the need for a centralized knowledge base.
- **Distribution of control:** the Hearsay II system consists of a blackboard with several sources of knowledge. However, the execution of knowledge sources is controlled by a single module (the scheduler). On the other hand, Hweitt's actor model (MIT) proposes another model in which control is distributed over several entities called actors. Each actor can perform local processing and send messages to other actors. What's more, each player has control over its execution. Indeed, in this type of system, there is no scheduler responsible for planning the execution of the various players, but they execute in competition.

4. Branches of distributed artificial intelligence

The basic idea behind distributed artificial intelligence is to use several intelligent entities to solve a complex problem. However, solution design using these entities can be carried out in several ways. Each solving method has its own problems and challenges. Visit As a result, distributed artificial intelligence has been developed along three different tracks:

- **Distributed Problem Solving (DPS):** this involves dividing the problem into a set of sub-problems. Then we assign the sub-problems to different entities in order to solve them. Then we need to synthesize the partial solutions to find the final solution. Taking the example of a landscaping and grounds maintenance problem. This problem can be solved by breaking it down into a set of sub-problems such as maintaining the arable land, planting and maintaining trees (gardening), treating trees against disease and insects, protecting and maintaining tools (tensioners, tractors, shears, etc.), etc. Each sub-problem is assigned to a specialized unit. By carrying out the various tasks, we can solve the main problem, which is grounds maintenance. Note that this example is only a simplified representation of this approach. In reality, many problems can arise when adopting distributed problem solving. Thus, the problems addressed by this branch are essentially: How can we divide the problem into sub-problems? How can problem knowledge be shared between solvers? How can we synthesize the partial solutions to find the overall solution? Projecting these questions onto the previous example, we ask questions like: on what basis have we broken down the main problem into the sub-problems mentioned above? Assuming that diseases have been discovered during tree treatment that require specific gardening tasks to be carried out (weeding the garden, pruning trees, avoiding planting... etc.), how can this knowledge be shared between the various entities? how can the various tasks be organized to achieve the main objective?
- **Multi-Agent Systems (MAS):** in this branch we have a set of entities endowed with intelligent behaviors that must cooperate to solve a common problem. Each entity has its own goals and plans. For example, in the case of the landscaping and grounds maintenance problem mentioned above, we'll need to design the solution of the problem in the form of a set of entities performing specific tasks (one for land maintenance tasks, one for gardening tasks, one for prevention and treatment and one for tool maintenance). Of course, an entity specializing in tool maintenance, for example, has its own objective (keeping tools in good condition) and has plans for achieving this objective (regular maintenance, diagnosing malfunctions, troubleshooting tools).

- **Parallel Artificial Intelligence (PAI):** This branch involves proposing solutions to take advantage of the benefits of parallel machines in the field of artificial intelligence. A number of techniques have emerged in this area, such as the creation of hardware adapted to the execution of intelligent software. Several machines have been designed for this purpose, such as *CM-2 (Connection Machine), Semantic Network Array Processor* and *IXM2 (Associative Memory Processor)*. What's more, this path proposes new paradigms for the development of intelligent systems. In fact, parallel machines offer the possibility of processing large-scale knowledge bases in real time. Case-based reasoning, for example, requires the power of parallel machines to save and process the large number of stored cases.

In our opinion, the success of multi-agent systems is due, among other reasons, to their multidisciplinary nature. As we explained in In this chapter, MAS represent a branch of DAI. Naturally, these systems exploit both artificial intelligence and parallel and distributed systems techniques. In addition, fields such as software engineering, psychology, sociology, economics and linguistics contribute to the development and success of multi-agent systems. Without doubt, multi-agent systems represent a new way of developing software (the agent paradigm). As a result, software engineering needs to propose new methods, methodologies and techniques that support the novelties of this paradigm.

Psychology helps MAS designers understand the nature of human reasoning. Modeling the latter represents an important building block in agent development. The BDI model is undoubtedly the archetypal example of how to exploit advances in psychology in the field of MAS. This model will be detailed in chapter 03.

Considering a multi-agent system as a community of interacting agents directly implies the study of these communities at the macro level (i.e. at the level of society). Two aspects can be studied at this level: the organization of the company and the interactions between agents. Studies of agent organizations are generally inspired by sociology. On the other hand, interactions between agents are based on communication. Linguistics is generally used to formalize the concepts linked to communication between agents. These two aspects will be explained in Chapter 02. In addition, interaction between agents enables them to make collective decisions in cooperative and conflictual situations. Such decisions are based on models that are well known in the field of economics. We'll explain coordination models in Chapter 04.

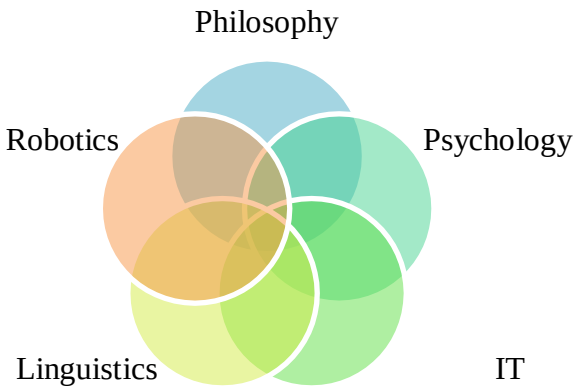

Figure 1.2: The multidisciplinary nature of multi-agent systems.

Despite the multi-disciplinary nature of the multi-agent field, there are fundamental differences between this new paradigm and other paradigms and computer systems. In the next section, we examine these differences.

5. The differences between MAS and computer paradigms

Multi-agent systems represent a new software development paradigm. This paradigm is based on new concepts different from those known in the various fields of computing. However, the similarities between the concepts of this new paradigm and other concepts represent a source of confusion. This confusion is exacerbated when we talk about concepts from fields directly related to MAS, such as expert systems, distributed systems and object-oriented software.

5.1 The differences between MAS and object-oriented software

The key concept in object-oriented programming is the notion of object. The latter is considered the ancestor of the agent. The similarity between the two concepts has made it possible to extend object-oriented programming languages to support agents. However, by overlooking the differences between the two concepts, we run the risk of under-utilizing the power of agents. In fact, the first difference between agents and objects is ***autonomy***. An agent is autonomous from the object. Autonomy represents the agent's ability to achieve his or her objective without the intervention of others. In object-oriented programming, a executes a method in response to a message sent by another object. When an object receives a message, it must respond to it by executing the method. On the other hand, an agent is not obliged to execute a method as a response to another agent. The decision to execute a method is taken solely by the executing agent.

In addition to autonomy, an agent is characterized by ***flexibility*** in relation to objects. Agent flexibility is inherited from artificial intelligence. This characteristic refers to the agent's ability to change his behavior according to the situation. In object-oriented software, an object always executes the same method in response to a received message. Consequently, generating the same situation (receiving a message) implies the same behavior (executing a specific method). On the other hand, an agent performs different behaviors depending on the situation. For example, when an agent is asked to execute a given behavior, it may execute the behavior, or it may refuse to execute it if the situation does not allow it. An object-oriented program has a single thread of **control**. As a result, at any given time, only one method can be found running. When a method calls on a method of another object, the new method is executed. In multi-agent systems, agents run concurrently. Each agent has its own control wire.

Despite the existence of certain extensions to object-oriented programming, where these differences are not fully identified (such as the case of control in concurrent object-oriented programming), we note that these extensions are not inherent features of basic object-oriented programming.

5.2 The differences between MAS and expert systems

Multi-agent systems share with expert systems the same domain of origin (artificial intelligence). In fact, an agent can be thought of as an expert specialized in the execution of a given task. However, the differences between the two technologies can be summed up in two essential aspects: ***the environment*** and ***sociability***. In fact, an expert system interacts with a domain expert. It is the latter that introduces knowledge to systems (facts). In addition, the system presents the results of its execution to the expert. An agent, on the other hand, interacts directly with the environment. Considering the example of an expert system in the medical field, the doctor must enter the patient's symptoms into the system, and the system presents the doctor with the diagnosis and/or medical prescription. The philosophy of an agent designed for the same problem is totally different. An agent must be equipped with sensors capable of perceiving the patient's symptoms. What's more, the agent must act directly on the patient's body (by injecting the medication, for example). The second aspect that differentiates multi-agent systems from expert systems is the social aspect. In expert systems, on the other hand, the agents interact with each other to solve the problem.

5.3 The differences between MAS and distributed systems

As mentioned above, multi-agent systems are a paradigm for the development of distributed systems. However, this new paradigm has brought some novelties compared to other distributed systems development paradigms. The differences between multi-agent and distributed systems essentially boil down to ***autonomy*** and ***flexibility***. Ordinary distributed systems are made up of a set of interacting conventional computer systems. As a result, these systems will not be autonomous or flexible.

6. The benefits of MAS

Multi-agent systems offer many advantages in the development of complex systems. In fact, the agent paradigm is considered the ideal paradigm for developing these systems. The power of multi-agent systems is a consequence of the interaction of different fields (artificial intelligence, software engineering and distributed systems).

The advantages of multi-agent systems include :

- **Modularity:** a multi-agent system is made up of a set of entities (agents). Each agent is conceived as an autonomous entity independent of the others. Agent autonomy means that each agent has control over its own state and behavior. Interaction between agents is based essentially on the exchange of messages. As a result, coupling between agents is weak. In addition, several basic concepts of multi-agent systems contribute to their modularity. One such concept is "organization".
- **Reusability:** reusability is a direct consequence of the modularity of multi-agent systems. In fact, modularity makes it possible to reuse certain components of one software package in another. In the case of multi-agent systems, reusability can be applied at several levels. Naturally, the agent is designed to be a *brick* that can be reused as much as possible. For example, designing a workshop as a multi-agent system composed of producer, transporter and manager agents offers the possibility of reusing certain agents in other systems. A transport agent, in this case, can be used in the development of logistics chains or road traffic simulation systems.
- **Ease of maintenance:** the modular development of multi-agent systems simplifies maintenance. There are several types of software maintenance: perfective maintenance, corrective maintenance and evolutionary maintenance. We believe that the agent paradigm has a positive impact on all these types of maintenance. Multi-agent systems are conceived as independent entities with a single weak coupling. Consequently, a change in one entity has little impact on the others. In the case of corrective maintenance, the purpose of this modification is to correct an existing system. For example, the discovery of an error in an agent only implies the modification of that agent. Similarly, the evolution of models and multi-agent approaches involves modifying existing systems.

- **Reliability:** reliability refers to a system's ability to continue operating despite the occurrence of failures or errors in one or more of its parts. Multi-agent systems can contribute to software reliability through agent autonomy. In fact, an agent is an autonomous entity in the sense that it can achieve its objectives without the intervention of other entities. As a result, an agent can achieve its objectives even if another agent has broken down.
- **Efficiency:** the agent paradigm offers the possibility of developing efficient solutions. Software efficiency is measured in terms of resources consumed to solve a problem. Multi-agent systems represent a distributed execution model where resources are distributed. This sharing enables rational use of resources. What's more, certain types of agent can make better use of certain resources. For example, mobile agents reduce the bandwidth used to reach the target. It is a well-known fact that data size is more important than processing size.
- **Adaptation to reality:** the agent paradigm enables faithful modeling of real phenomena. Agent characteristics such as autonomy, flexibility and sociability are intrinsic to many real systems (such as human societies, biological systems, insect colonies, etc.). Consequently, the adoption of this paradigm represents the essential characteristics of the systems mentioned above.
- **Sophisticated interaction modes:** multi-agent systems support sophisticated interaction modes such as negotiation, cooperation and collaboration. The object paradigm, on the other hand, is based on simple interactions that only allow methods to be invoked. The complexity of interactions enables the modeling and development of complex systems where interaction is a fundamental characteristic. Examples of such systems include e-commerce systems, social phenomenon simulation and supply chains.

7. The challenges of MAS

Despite the advantages of multi-agent systems, this paradigm faces several challenges. In fact, these challenges are results of agent features and multi-agent systems. Taking the example of autonomy as an intrinsic characteristic of agents, this characteristic makes the agent's behavior unpredictable. As a result, the validation of the systems developed represents a real problem. Thus, the challenges of multi-agent systems lie in :

- Formulation of problems, allocation of resources to the various entities and synthesis of results.
- Interaction between agents (when and how) and reasoning about other agents during the interaction process.
- Ensuring behavioral consistency by ensuring a compromise between local actions and distributed processing, and eliminating undesirable effects (such as chaotic behavior).
- Multi-agent systems engineering, by developing methods, methodologies, techniques and tools to facilitate the development of these systems.

8. MAS applications

Multi-agent systems are generally suitable for the development **of complex, distributed and heterogeneous systems,** where the solution is the result of the **interaction of several entities**. As a result, several areas are considered ideal for application. It goes without saying that. It would be difficult to list all the fields of application for multi-agent systems. In fact, in this course we've chosen to mention those areas of application in which real applications have seen the light of day. Multi-agent systems have been applied in the following areas:

- **Business process management:** decisions taken in large companies are based on information from a variety of sources. The real challenge is to make correct decisions with incomplete or obsolete information. Many researchers are finding that multi-agent systems are ideal for working in these conditions. The ADEPT system has been developed for this purpose. This system is made up of a set of agents that model the roles and departments of a company. Naturally, each agent offers a range of services.

- **The medical field:** the medical field in general, and surgical intensive care in particular, is characterized by several attributes that favor the use of multi-agent systems. Critical care systems are organized into teams with several experts who cooperate and share information to achieve their goals. What's more, the level of expertise involved is not the same. In fact, some of our staff do not have the necessary skills to deal with specific situations. For example, information workers cannot interpret the data collected. The GUARDIEN system perfectly models these systems.
- **Cooperative information systems:** at present, information is distributed everywhere (in local networks or the Internet) with heterogeneous structures and natures. Cooperative information systems (CIS) are systems capable of searching, filtering and merging information. These systems can be used in the financial sector (to find share prices, stock market information, etc.), the industrial sector (for example, in the automotive sector) and the industrial sector (for example, in the construction sector). tourism (finding itineraries for a given destination, information on accommodation, means of transport, etc.), supply chains, etc.

9. Conclusion

In this chapter, we've given a general introduction to distributed artificial intelligence, the original domain of multi-agent systems. Thus, the three essential tracks of this field were presented, focusing on multi-agent systems. The differences between these systems and other paradigms were detailed before presenting the advantages and challenges of this new paradigm. Next, we present a few examples of multi-agent system applications.

1. Introduction

Multi-agent systems (MAS) is a new programming paradigm that is increasingly taking its place among the technologies for developing complex, distributed systems. This importance may be reflected in the evolution of the global market for software agents.

The aim of this chapter is to answer this question by presenting the general concepts of the multi-agent domain. In fact, these concepts can be presented using either a bottom-up or a top-down approach. The first approach is to build the overall picture (the MAS concept) from the basic concepts. The second approach, on the other hand, involves breaking down the overall concept to find the basic concepts. We're going to opt for the second approach because, in our opinion, it's more educational.

2. Multi-Agent Systems (MAS)

Before decomposing the notion of multi-agent systems to its basic components, it seems important to specify the notion of the system to be studied. In this section, we present the definition of a multi-agent system, before describing their types. Next, we briefly address the problem of control in MAS.

2.1 Definition of a multi-agent system

The concept of multi-agent systems can be popularized by defining the term "system". It seems obvious that a multi-agent system is a system composed of a set of agents. A multi-agent system is a set of interacting agents. In fact, in this definition we have only adopted one of the definitions of the concept "system" (according to the systems approach), which consists of a set of interacting elements. However, a closer analysis shows us the limits of this definition.

Despite the importance of Ferber's definition, which is considered a benchmark in the field, it's important to note that the components proposed by this definition do not have the same importance. It is even possible to neglect certain components in some multi-agent systems. We can, therefore, distinguish specific types of multi-agent systems according to the characteristics of certain components.

2.2 Specific cases of multi-agent systems

Ferber's definition gave specific importance to the multi-agent system environment. However, he characterized the environment of an MAS as a metric space. However, in some multi-agent systems, it is impossible to identify the environment in this way. Software agents are examples of such cases. In fact, a software agent does not exist in a Euclidean geometric space.

In the case of purely communicating *multi-agent* systems, there is no environment in the sense of a metric space that encompasses objects (i.e.
$E = \emptyset \; et \; A = 0$). As a result, agent actions boil down to communication between agents. Typically, the agents in this case are software computing entities. On the other hand, purely situated multi-agent *systems* are MAS made up of agents that do not communicate directly with each other using messages, but only by manipulating objects in the environment. In this case, the agents (which are generally physical entities) take up real positions in the environment.

It's important to note that the environment on which this classification is based represents the logical environment of MAS. Of course, purely communicating software agents exist in IT environments. The difference between the two concepts will be explained later in this chapter.

A multi-agent system composed of a set of agents that are heterogeneous in their structures, capabilities and communication languages, and that can join and leave the system, is called an *open* multi-agent system. In this category of multi-agent systems, several problems still require the attention of researchers, such as agent heterogeneity, trust, organization search and failure management. *Closed multi-agent systems, on the other hand,* are made up of a set of homogeneous agents operating in organizations with restricted admission policies. Despite the advantages of open multi-agent systems, closed systems are the most widely used in real-life applications. Of course, a fundamental issue in all types of systems (open or closed) is the control of a system's operation. The following section explains two approaches used to solve this problem.

2.3 Control in multi-agent systems

Although multi-agent systems are a model of distributed systems, the execution of agents must be consistent. Thus, a fundamental question in this field is the choice of a type of control for the various agents that make up the system. In the literature, there are two different approaches to agent control:

- **The blackboard model (the centralized approach):** in this approach, the system is made up of a set of agents (called knowledge sources), a blackboard for agent interaction and a control module. The control module ensures that the appropriate knowledge source is executed. In fact, this module orders knowledge sources according to the content of the blackboard. Agents use the blackboard as a shared space for interaction. This means that every agent has access to the blackboard for reading or writing. Changes in the blackboard can trigger the execution of a knowledge source. As a result, the control module schedules the execution of this knowledge source according to its priority level. Figure 2.1 shows a simplified view of this model. The HearSay II architecture mentioned in the previous chapter is a typical example of this model. It is important to note that the distributed nature of multi-agent systems has a negative impact on the applicability of this model. However, the model is still applicable to the development of agents composed of other agents (for example, the ARTIS agent, which we'll introduce in Chapter 03).

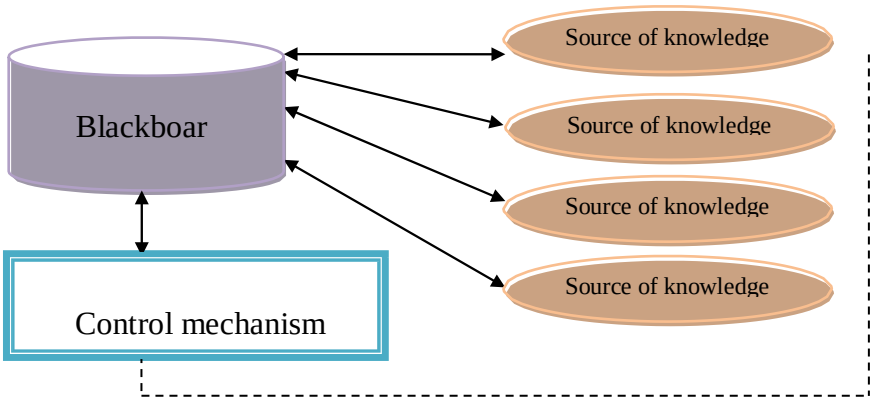

Figure 2.1: The chalkboard model.

- **The actor model (distributed control):** the actor model is another IAD model. Actors are autonomous entities that communicate via asynchronous messages. Thus, there is no centralized control, but each player has control over itself. As a result, players find it difficult to reach global objectives with local knowledge. Figure 2.2 shows this IAD model.

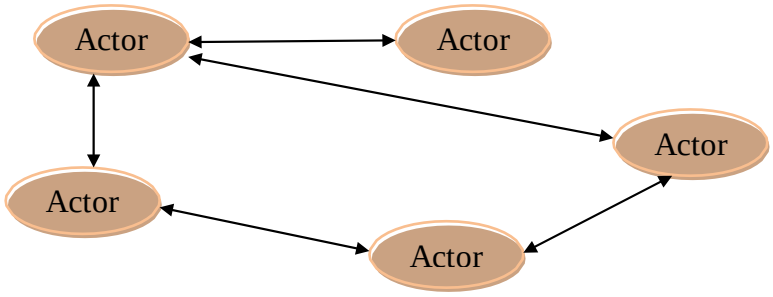

Figure 2.2: The actor model.

As mentioned above, multi-agent systems introduce several new concepts. The Voyelles approach is one of several that present the concepts of an MAS in a single framework. According to this approach, a multi-agent system is made up of agents (A), interactions (I), the environment (E) and organizations (O). In the remainder of this chapter, we present these essential elements of MAS.

3. The agents

Agents are undoubtedly the basic building block used to create MAS. What's more, we can consider the agent to be the most polemical notion in this field. This section is dedicated to the presentation of agents, their characteristics and types.

3.1 Defining an agent

Not everyone agrees on the definition of an agent. Indeed, several definitions can be found in specialized literature. In a famous scientific article entitled (***Is it an Agent, or just a Program? A Taxonomy for Autonomous Agents***), Stan Franklin and Art Graesser present a list of the definitions most widely accepted by specialists in the field. Of course, some of the definitions presented in this article are no longer cited. As a result, we select definitions that are still valid today. Thus, an agent can be defined as :

- An entity that perceives and acts on its environment;
- A computer system existing in a dynamic environment that can perceive and control it autonomously in executing the tasks or achieving the objectives for which the system was designed;
- An entity that continuously performs three functions: perceiving the dynamic conditions of an environment, acting to affect the conditions of that environment, and reasoning to interpret perceptions and determine actions;
- An autonomous entity, real or abstract, which is capable of acting on itself and on its environment, which, in a multi-agent universe, can communicate with other agents, and whose behavior is the consequence of its observations, knowledge and interactions with other agents;
- An autonomous, reactive and proactive IT system.

Despite the diversity of definitions of the agent concept, we can note the existence of certain common characteristics between the definitions presented. In fact, we can identify agents by their characteristics.

3.2 Agent characteristics

The diversity of agent definitions is due to the diversity of application areas in this field. However, a closer reading of these definitions enables us to identify certain common features in most of the proposals. On the other hand, we can find certain characteristics that exist in definitions without others. Consequently, we can ask how important these characteristics are in defining the agent concept. In other words, are all the features needed to design an agent?

To answer this question, we can use Ferber's definition as an example. In fact, Ferber emphasized that an agent's behavior is the result of its ***observations***, ***knowledge*** and ***interactions***.

Consequently, we distinguish between agent characteristics, intelligent agent characteristics and specific agent characteristics.

- **Agent characteristics :** From the definitions above, we can identify the essential characteristics of agents. Thus, two important characteristics give rise to the agent concept: situation in an environment and autonomy.
- **Characteristics of intelligent agents:** agent intelligence is a reflection of the flexibility that characterizes agent behavior. Indeed, an intelligent agent is one that ensures the flexibility of its behavior. Flexibility refers to the agent's ability to change its behavior or structure in order to achieve its objectives. Taking the example of a soccer player, this agent can have attacking and/or defensive capabilities. Knowing that the player's primary objective is to help his team defeat the opposition, the attacker's aim is to score goals, and the defender's aim is to prevent the opposition from scoring a goal.
- **Specific agent characteristics:** as we noted earlier, MAS are used in a wide variety of applications. Indeed, some applications require optional features in order to achieve their objectives. For example, agents operating on the *Internet* have the ability to move between machines in order to execute their requests. Mobility is one of the hallmarks of this type of agent. Different agent properties are often used as classification criteria. In the literature, we find classifications based on agents' mobility, adaptability or rationality.

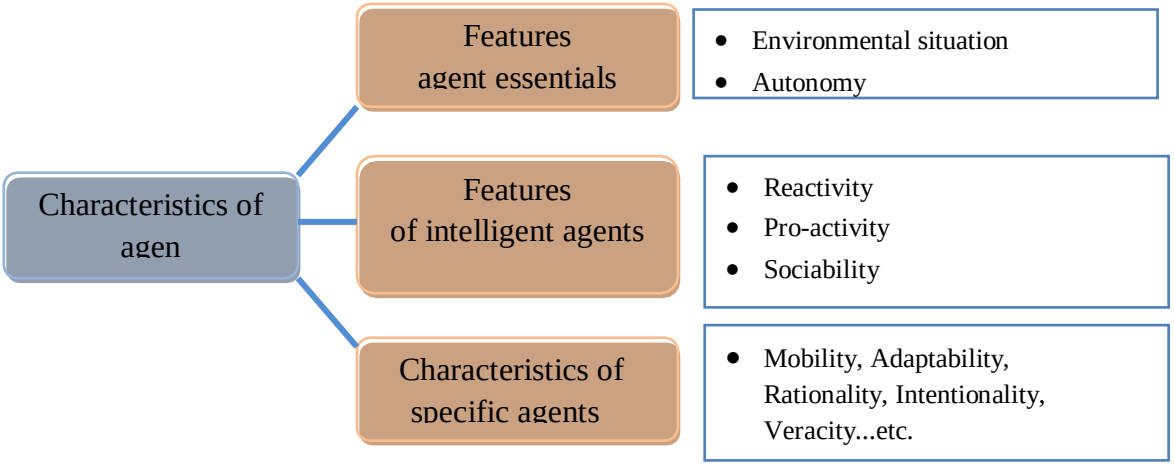

Figure 2.3: Agent properties by type.

3.3 Agent types

The agent characteristics mentioned in the previous section can be considered as agent classification criteria. In fact, agents can be classified into mobile **and fixed agents** according to the criterion of **mobility**. In addition, the **communication criterion** can be used to classify agents into agents with direct **communication** or agents with indirect **communication** (called situated agents). We can also choose the level of **agent** cooperation as a classification criterion. In this case, we find classes of agents ranging from **totally cooperative** to **totally antagonistic**. In this section we will detail a classification based on the criterion of **agent** granularity, as this is the classification most widely adopted by domain researchers. By granularity, we mean the complexity of agent behavior and knowledge. In this section, we only present the principles of the various agent classes. The presentation of agent models for each class will be the subject of the third chapter. Under this criterion, we find three classes of agents:

- **Cognitive agents:** this class of agents is part of symbolic artificial intelligence. In this way, the agent has a symbolic representation of its environment, of other agents, and of the agent's actions and internal states. Knowing that the agent's goals are modeled as internal states, the agent's reasoning consists in finding the set of possible actions to satisfy its goals. Thus, the essential problem in this category of agents becomes a planning problem. Of course, several attributes can be taken into account in the reasoning process, such as the agent's beliefs about the beliefs of other agents, or the agent's execution history. However, the integration of various attributes into the reasoning process can complicate the latter. As a result, performance becomes a relevant issue in this class of agents. Finally, we note that a multi-agent system in this category is composed of a small number of heterogeneous agents.
- **Reactive agents:** reactive agents are based on reactive artificial intelligence. This school of artificial intelligence is based on the ability to design intelligent behavior from simple behavior. Indeed, intelligence is a phenomenon that emerges from the interaction of simple entities. Ants are a case in point. In fact, the behavior of ants is simple and lacks any real intelligence. However, ant interaction produces intelligent behavior. Consequently, reactive agents are agents that possess neither a representation of their environment nor a true reasoning

mechanism. What's more, their behavior is modeled by stimulus-response rules that generate actions from perceptions. This behavior is often modeled by finite-state machines. Unlike cognitive agents, a multi-agent system based on reactive agents is made up of a large number of homogeneous agents.

- **Hybrid agents:** the division of agents into two classes is merely an idealistic view. In fact, these two classes present only the extreme cases. In reality, an agent requires reactive capabilities and cognitive skills. So it's important to combine the two approaches mentioned above. A hybrid agent is an agent designed in layers. The lower layers ensure reactive behavior. On the other hand, the upper layers are responsible for complex cognitive abilities such as social aspects. Of course, higher layers manipulate knowledge, while lower layers manipulate data directly. As a result, intermediate layers are responsible for transforming data into knowledge.

4. The environment

An agent, by definition, is an entity that acts on an environment. Indeed, the environment is an important component in multi-agent systems. By environment we mean the passive objectives that make up the world of the environment. Generally, the agent evolves in this world. For example, a player's environment is the terrain in which the game takes place. From an abstract point of view, this pitch can be modeled by the set of passive objects that make it up (the stands, the ball, etc.). It's important to note that the passivity of objects in the environment doesn't mean that the environment itself is passive. In fact, the environment can be seen as an active, dynamic entity. For example, environmental laws may allow the state of the environment to change without the influence of an agent. In our example (*soccer*), the ball can change its state (change its movement or direction) without the influence of a player.

The third layer represents the system's logical environment. This logical environment is made up of all the passive objects that model the system. Figure 2.4 gives an overview of this layered modeling of multi-agent environments.

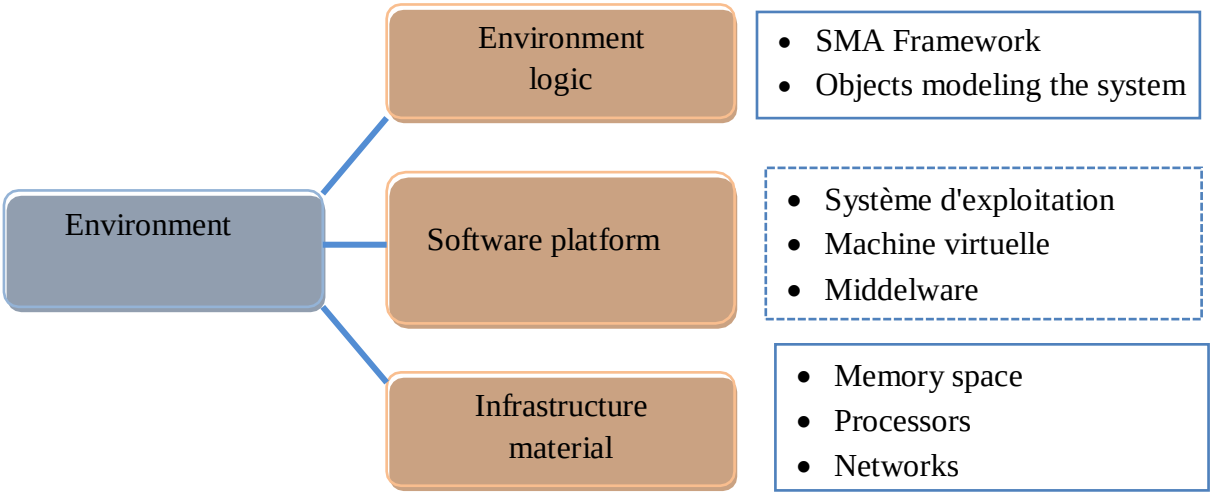

Figure 2.4: The layered architecture of an environment.

A multi-agent environment is characterized by several attributes such as :

- **Accessibility:** in an accessible environment, agents can access its full state. On the other hand, the scope of the agent's actions and perception is local in an inaccessible environment.
- **Determinism:** the state of a deterministic environment is linked only to its previous state and to action. In an indeterministic environment, the results of the same action in the same context can be different.
- **Dynamism:** a change of state in a static environment occurs exclusively through the actions of agents. However, the state of a dynamic environment can be modified without the intervention of agents.
- **Continuity:** if the number of possible perceptions and actions of the agent in an environment is unlimited, the environment is said to be continuous. If not, we consider it discreet.

Finally, it's important to note the difference between an agent's environment and the environment of a multi-agent system. In fact, the environment of a multi-agent system consists, as we mentioned earlier, of the passive objects that make up the system. An agent's environment, on the other hand, is made up of these objects and the other agents in the system.

5. The organization

When studying multi-agent systems, it's impossible to ignore the society in which the agent is embedded. Thus, we generally study agents from two different dimensions: the micro level and the macro level. The micro level refers to the agent's internal architecture and reasoning mechanisms. At the macro level, we focus on the multi-agent system as an organization. Sociologist Edgar Morin defines an organization as "*an arrangement of relationships between components or individuals that produces a unit, or system, endowed with qualities unknown at the level of the components or individuals*. ". A macro-level study of multi-agent systems must consider agents as black boxes. However, the problem is *how to study system behavior while ignoring individual* behavior. In fact, such a study should focus on the agents' functions, not their behavior. In other In the long term, by adopting the macro view, we will have to deal with the question of *who does* what, but not with the question of who does what.

As a consequence of the points discussed above, multi-agent organizations can be treated along the following lines:

- **Structural analysis:** the role of this analysis is to identify the shape of the organization. In fact, a multi-agent system can be organized as a *hierarchy*, a *group* or a *coalition*. A correct structural analysis should not be limited to the structure of the system. Other criteria need to be taken into account to properly identify the category of organizational structure, such as the exchange of information and control flows, the structure's objectives, the process of creating the organization and its lifespan. For example, a group and a coalition can share the same structure. However, coalitions are formed for well-defined objectives and short periods of time. Short-term goals and the coalition-building process are the key differences between a group and a coalition. Finally, we distinguish two classes of organizational structures: predefined structures and emergent structures.

- **Functional analysis:** this analysis determines the functions of the organization's members. Although the roles that exist in a system are closely related to it, some generic roles can be commonly identified as *facilitator*, *broker* and *mediator*. Its roles provide communication, directory and/or coordination services in a multi-agent system.
- **Realization parameters:** this dimension deals with the realization of SMA organizations, moving from structural organization to concrete organization. The fundamental question in this area is how to assign roles to the various agents that make up a multi-agent system. Two criteria guide designers in addressing this issue: *specialization* and *redundancy*. By means of the specialization criterion, the designer must decide whether a given agent should be specialized in a single role or be generic in the sense of being able to perform several roles. Redundancy, on the other hand, refers to the possibility of assigning a single role to a set of agents. In fact, there is no single answer to these two choices. In principle, the designer must address the issues of specialization and redundancy by taking into account the specific features of the system in question.

To simplify the designers' tasks, several organizational models have been proposed. An organizational model is an abstract representation of the essential characteristics of multi-agent organizations. So designers need only design the system according to the characteristics of the chosen model. A model can answer questions relating to the structure of organizations, the generic functions that exist in an organization, the criteria for assigning roles and/or the management of organizations. In this section, we present the AGR model and the MOISE+ model.

The AGR model is based on three central concepts: Agent, Group and Role. This model places no constraints on the architecture of an agent. An agent is simply an autonomous interacting entity that performs actions in a social context. This agent belongs to one or more communities called groups. A group. A group is the generic term used to designate a community of related agents. In addition, an agent can play a role that is an abstract representation of a group function.

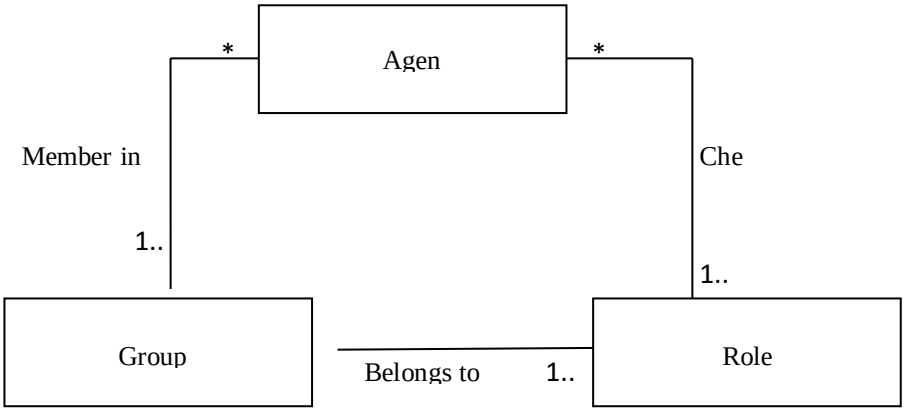

Figure 2.5: The AGR model.

The MOISE+ model is a fast-growing organizational model. Since its first version, the model has undergone several extensions. The MOISE+ model focuses on the concepts of role, group and organizational links. A role represents the authorized behavior of an agent within the organization. Indeed, a role can be specified by a set of missions. A mission is formally defined by the quadruplet <G, P, A, R> which represent, respectively, the goals, plans, actions and resources authorized within the mission. In addition, the model proposes organizational links to structure the relationships between the agents playing these roles. Three organizational links can exist between the different roles: communication, authority and bridging. The communication relationship refers to the possibility of communication between two roles. The authority relationship expresses the subordination of one role to another. The accointance relationship gives a given agent the right to represent and use a subset of the organization in its reasoning.

6. Interaction

Agents are generally characterized by a limited set of skills. Consequently, agent interaction is an important condition for effectively achieving an agent's goals. In fact, an isolated agent is only a handicapped entity in the sense that it cannot achieve its objectives. Because of the importance of agent interaction, some authors consider it to be the raison d'être of multi-agent systems. You can't talk about a multi-agent system simply by assembling agents without interaction (a multi-agent system is by definition a set of interacting agents). However, interaction is also the source of problems in multi-agent systems. In fact, interaction forces agents to reason about the capabilities of other agents, their beliefs, their goals, and so on. For example, to ask another agent for help in achieving its goals, an agent needs to know which agents share its goals (it's unreasonable to ask an agent to behave in a way that contradicts its goals), or which agents are capable of performing the requested task. Of course, an agent's autonomy prevents others from accessing its own capabilities and goals. What's more, poorly managed interaction situations can lead to undesirable or even chaotic situations. To understand the chaotic situations created by poorly managed interactions, just imagine a city where cars don't obey the rules of the road. In this example, the car drivers represent the agents. Each driver needs to think not only about his or her own objectives, but also about the intentions of other drivers.

Interaction represents the influence of an agent on the behavior of other agents through a series of actions. In this section we will focus on two important aspects: interaction types and communication in multi-agent systems. In fact, interaction cannot be reduced to the simple specification of elementary mechanisms enabling agents to influence the behavior of other agents. On the contrary, it allows us to analyze and design forms of interaction that enable agents to achieve their objectives. Of course, the forms of interaction can be as simple as they are complex. In the second part of this section, we present communication as the ideal medium for agent interaction. It is important to note that interaction models will be the subject of Chapter 04 of this course.

6.1 Types of interaction

The concept of interaction is generally associated with several other concepts such as: coordination, cooperation, negotiation and collaboration. Coming up with a classification that everyone can agree on is no simple task. We believe that the lack of an accepted classification is due, among other reasons, to the ambiguous nature of these terms (cooperation, coordination, collaboration, etc.). Most classifications in this field take the purpose of the interaction as a classification criterion. Thus, we find forms of interaction where agents share compatible goals and forms where agents possess incompatible goals.

In his book, Ferber presents a detailed description of different forms of interaction. Its classification is based on three criteria: compatibility of goals, sufficiency of resources and sufficiency of skills. Goal compatibility refers to the possibility of achieving the agents' different goals. On the other hand, if two goals are incompatible, agents can achieve only one goal. What's more, this classification differentiates between resources and skills. Resources are the different means used to carry out a task. On the other hand, the Competencies are the attitudes of an agent for carrying out a task.

Goals	Resources	Skills	Form of interaction
Compatible	**Sufficient**	**Sufficient**	**Independence**
Compatible	**Sufficient**	**Insufficient**	**Simple collaboration**
Compatible	**Insufficient**	**Sufficient**	**Dimensions**
Compatible	**Insufficient**	**Insufficient**	**Coordinated collaboration**
Incompatible	**Sufficient**	**Sufficient**	**Individual competition**
Incompatible	**Sufficient**	**Insufficient**	**Group competition**
Incompatible	**Insufficient**	**Sufficient**	**Individual conflicts**
Incompatible	**Insufficient**	**Insufficient**	**Collective conflicts**

Table 2.1: Forms of interaction.

In Table 2.1, we can see the existence of several forms of interaction, ranging from primitive to more complex. The first form of interaction is a very special one. In such a system, agents have objectives that are compatible with sufficient resources and skills. In this way, each agent is independent of the other agents because it simply seeks to satisfy its goals (there is no interaction). If the goals are compatible, the agents enter into cooperative interactions, because each agent will help the other agents achieve their goals. In this case, we differentiate according to the sufficiency of skills and resources. If skills are lacking, agents must work together to achieve their goals. However, collaboration is simple if the agents have sufficient resources; otherwise the agents have to coordinate in addition to collaborating. The example of a system for manufacturing a product is perhaps a good example to explain the difference between the two situations. Manufacturing a product is a process that requires several agents with different skills (design, production of various parts, assembly, etc.). If each agent in the workshop has his or her own tools for carrying out the task, the agents just have to work together to make the final product. However, the lack of tools forces agents to share existing tools. As a result, agents need to coordinate and collaborate at the same time. Overcrowding occurs when agents have sufficient skills but insufficient resources. An example of this situation is a practical training room where the number of PCs (resources) is lower than the number of students (agents), who are able to solve their problems.

Competitive or conflict situations are characterized by incompatible goals. Strategic games, wars and economic situations are all examples of competitive situations. For example, the magic cube game is a game with incompatible goals where each player has the skills needed to solve the problem. In addition, each player has his own cube (resource). In this case, the situation is one of pure individual competition. On the other hand, there are situations where agents do not possess all the skills needed to solve a problem, despite the existence of sufficient resources. In this case, agents group together and the situation becomes one of pure collective competition.

If resources are insufficient and goals are incompatible, the situation becomes one of conflict. The game of tennis is an example where the two players have incompatible goals and there is only one tennis ball (insufficient resource). In this case, the situation is one of individual conflict over resources. The soccer game, on the other hand, is a situation of conflict (insufficient resources and incompatible goals), but with insufficient skills. In fact, in the game of soccer, no single player possesses all the skills. As a result, the situation is one of collective conflict over resources.

6.2 Communication

Communication is the basis of all types of interaction. By communication we mean the transformation of information from one agent to one or more other agents, using articulated language or other codes. However, communication is not always associated with the explicit exchange of messages. In fact, communication can be direct or indirect. In direct communication, agents explicitly exchange messages. In indirect communication, on the other hand, agents manipulate their environment, which represents the medium of communication.

This type of communication is generally applied in reactive agents. Ants use their tracks as a means of communication. In addition, communication can be either point-to-point or broadcast. In point-to-point communication, the sender sends the message to a single receiver. In broadcast communication, on the other hand, a sender sends the message to a set of agents. A final classification of communication between agents is based on agent intentionality. Communication can be an intentional process in which agents are aware of the act of communication, or it can be an incident. Communication in some animal communities is considered incidental because the animals are not shouting to communicate with other agents in their community. In the same way, a newborn baby communicates with its mother through cries.

We have defined interaction as the influence of one agent on the behavior of another. But the question is, how can we influence the behavior of another agent by sending messages? The answer to this question is provided by the theory of the language act proposed by Austin in 1962 in his article *How To Do Things With Words*. This theory studies the effect of a statement on the recipient.

Of course, the interaction between agents is not limited to the simple exchange of a message. Interaction represents a combined conversational sequence of several messages. The modeling of this sequence is based on one of two approaches:

7. Conclusion

This chapter introduces the basic concepts of multi-agent systems. In fact, multi-agent systems, as an interaction of several domains, offer several concepts. The basic problem is how to represent these concepts in a single framework. The vowel approach adopted here allows multi-agent systems to be represented using four basic concepts: agent, environment, interaction and organization. This chapter is devoted to presenting these concepts.

The next chapter is devoted to a detailed presentation of the agent concept.

1. Introduction

The agent is a central concept in multi-agent systems. However, this concept is characterized by several features with different degrees of importance. In some cases, these characteristics can even be contradictory. For example, pro-activity refers to the agent's ability to seek the realization of his objectives. However, an agent with this characteristic can ignore important events in its environment. Responsiveness is the key to handling these events. Consequently, integrating these characteristics into a single entity (the agent) is not a simple matter. We generally use models (or architectures) to present the essential components of an agent.

An agent architecture is the software (or hardware) structure which, from a set of inputs, produces a set of actions on the environment or on other agents. In other words, an agent architecture is a particular methodology for building an agent. It specifies how agents can be decomposed into a set of modules and how these modules interact. The total set of modules and their interactions determine how the agent's perceived data and current internal state determine its future actions and states.

This chapter is dedicated to the presentation of agent architectures. We begin this chapter by presenting an abstract architecture, before moving on to some concrete ones. At the end of this chapter, we present two examples of specific architectures. These are the ARTIS agent for real-time agents and the DIMA architecture for adaptive agents.

2. Abstract agent architecture

By abstract architecture, we mean an architecture that does not present concrete modules for agent implementation. These architectures are generally represented by algebraic or logical specifications.

Knowing that the interaction between the agent and the environment is a fundamental characteristic of the agent (see Chapter 02), we begin the specification of the agent by specifying this interaction. Thus, the environment "E" is modeled by a set of discrete states. Therefore, $E = \{e_1, e_2, \ldots, e_n\}$.. In addition, the agent's set of possible actions is represented by the set $Ac = \{a_0, a_1, \ldots a_n\}$. Consequently, the interaction between the agent and its environment is represented by a transition between two states due to the execution of an action. For example, the agent's

current state is e_i, and executing action a_i will set the environment to state e_j.

An agent's execution in its environment is the sequence ***r*** that represents changes in the environment's states due to the execution of the agent's actions.

The set of possible executions of an agent is represented by a set of **R.** Within this set, we can distinguish two specific sub-sets: ^{RAC} (set of execution sequences ending in actions) and ^{RE} (set of execution sequences ending in states).

The transformation of the state of the environment can be represented by a mathematical function (τ) which, from a set of execution sequences ending in actions, obtains a subset of states of the environment.

$$r = R^{AC} \rightarrow \varphi(E)$$

This specification of the state-of-the-environment transformation function represents an extremely general case. In fact, in this case, the transformation is history-dependent because it is linked to an execution sequence (^{RAC}) and not just an action. On the other hand, this transformation is indeterministic because the execution sequence gives us a possible set of states ((E)) and not a single state.

The environment is thus defined by the triple $Env=(E, e_0, \tau)$ where E is the set of states of the environment, e_0 is the initial state and τ is the transformation function of the environment.

Formally, an agent is defined as a function that gives an action from a sequence that ends with a state.

$$Ag = R^{E} \rightarrow Ac.$$

By associating an agent with an environment, we can obtain a system. The latter is formally specified by a set of possible executions of an agent ***Ag*** in its environment ***Env.*** This system is represented by ***R(Ag, Env).***

Formally, an execution sequence ***(e_0, a_0, e_1,….)*** is a possible execution sequence of the agent ***Ag*** in the environment ***Env*** **if**:

e_0 is the initial state of the environment ***Env,***

a_0=Ag(e_0)

,

for u > 0, u > 0, $e_u \in r((e_0, a_0, \ldots, a_{u-1}))$

$$a_u = Ag((e_0, a_0, \ldots e_u))$$

Now we'll introduce our agent. Assuming that the designer's objective is to avoid situations where both lamps are switched off or on at the same time. In this way, the agent controls the state of the room, and manipulates the switches according to this state. Consequently, the agent is a function that transforms a sequence ending in a state into an action. Of course, we've said that the agent will transform a sequence that ends in a state, not just a state, because the agent's action may be dependent on its history. For example, if the current state of the room is (T, R) and a user has switched on the first lamp (so the state becomes (A, R)), then the agent can manipulate the first lamp (to arrive at the previous state (T, R)) just as it can manipulate the second lamp (to arrive, after an execution sequence, at state (A, T)).rst lamp (to reach the previous state (T, R)) just as it can manipulate the second lamp (to reach state (A, T) after a sequence of executions). In this case, rational execution must take into account the state before user manipulation. Otherwise, the agent will perform actions that contradict the user's objectives.

2.1 A purely reactive agent architecture

A purely reactive agent is one whose behavior is independent of its history. Consequently, this agent can be specified as a function from a state of the environment to an action ***(Ag : E→Ac)***. However, this level of abstraction does not show how an agent can make decisions. We'll need to ***refine*** this specification to show the agent's internal mechanisms.

The agent's decision module can be broken down into two sub-modules: ***perception*** and ***action***. Figure 3.1 shows the architecture of this agent. The module "***Seeing"*** is responsible for ensuring perception. On the other hand, the "***action***" function is used to assign actions to the environment.

Formally, an agent can be defined by the composition of two functions: ***see*** and ***action***. So, ***Ag=(see, action)***. The "***seeing***" function transforms environmental states into perception (***see: E → Per***). Perceptions represent an interpretation of the state of the environment. The action generates an action to be executed from a set of environmental perceptions (action: ***Per*→Ac).***

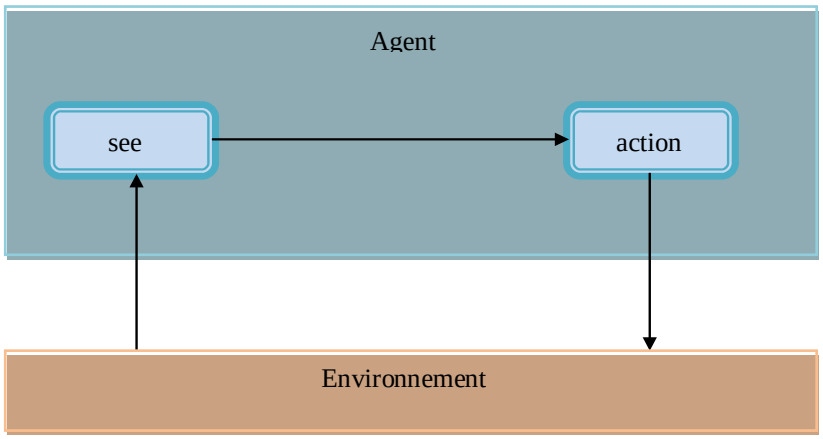

Figure 3.1: description of a purely reactive agent.

It is important to distinguish between perception and the state of the environment. Perception represents an agent's interpretation of the state of the environment. It is therefore possible to find a change in the state of the environment, but the agent does not perceive this change. For example, people suffering from color blindness cannot distinguish certain colors. So, even if we change the colors in the environment, the agents' perceptions don't change.

2.2 An agent with state

Assuming now that an agent takes into account its history when making decisions. How can we model the abstract architecture of this agent?

In fact, this agent's history can be saved in an internal report. As a result, perceptions and the current state influence the agent, generating a new internal state. The "action" decision function therefore becomes based on this internal state. Figure 3.2 shows this architecture.

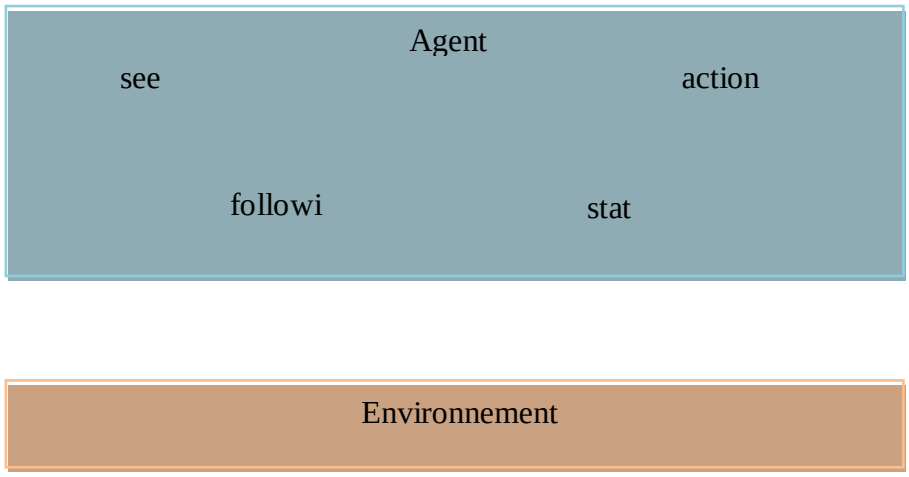

Figure 3.2: Architecture of a stateful agent.

Agent execution is based on the following cycle:

1. The agent starts in internal state i,
2. It observes a state (e) of the environment,
3. It generates perception (Per1 = *see*(e))
4. It executes the *following function* to update its internal state i'=.

   next (i, Per1) = *next* (i, *see* (e))

   It selects an action a = *action*(i') = *action*(*next* (i, Per1)) =Action (*following* (i, *see* (e))),

3. Concrete architectures

Following the presentation of abstract agent architectures, we present some concrete architectures in this section. Concrete architectures present the various modules for agent development. Of course, several architectures have been proposed in the literature. Since granularity-based agent classification is the most widespread classification, we have chosen to present an architecture for each class of this classification. A cognitive agent architecture, a reactive agent architecture and two hybrid agent architectures will be presented.

3.1 BDI architecture

The **BDI (Belief-Desire-Intention)** architecture is a cognitive agent architecture inspired by intention theory. This theory of psychology explains how decisions are made. Thus, an agent uses his beliefs and objectives to choose the goals he must reach and the strategies he must apply. As a result, this architecture is based on three essential components:

- **Beliefs:** beliefs represent the information an agent possesses about its environment and about other agents. In fact, the agent's beliefs are obtained from its perceptions and its ability to interact with other agents. Of course, the agent's beliefs may be incomplete, uncertain and incorrect. We have already explained in previous chapters that the agent's perception capabilities are limited. What's more, the agent's environment can be dynamic. Consequently, the truthfulness and correctness of beliefs are not absolute.
- **Desires:** represent the agent's goals. In particular, desires represent the states of the environment and of oneself that the agent seeks to achieve. Of course, the agent's desires can be represented by the same formalise used to represent beliefs, because both concepts represent states. Naturally, an agent cannot achieve all his or her objectives. On the one hand, objectives can be contradictory. On the other hand, the agent's resources, especially time, may prevent him from achieving all his goals.

The algorithm shown in figure 3.4 represents the control cycle of the BDI agent. In this algorithm, symbols ***B***, ***D*** and ***I*** represent beliefs, desires and intentions respectively. In addition, ***B0***, ***D0*** and ***I0*** represent initial beliefs, desires and intentions respectively. The ***revision function*** is responsible for revising beliefs. The decision process is represented by the two functions ***des*** and ***option***. The ***filter function*** can be used to filter intentions. Once an intention has been selected, the ***plan function*** transforms it into an executable plan (***EP***).

Several real-life examples can be applied to BDI agents. We can think of all human beings as cognitive agents. For example, a professional soccer player is a BDI agent. The latter has several objectives: basic human objectives (guaranteeing one's health, safety, etc.), social objectives (guaranteeing the needs of one's family, preserving and enriching human relations, etc.) and professional objectives (developing one's skills, devoting time to one's team, playing for one's national team, etc.). Of course, these objectives do not have the same degree of importance. At every moment, the player will take into account new beliefs in order to choose the most relevant goals from among his objectives. Then he'll build plans to achieve these goals.

BDI agent control algorithm

B ← B0
D ←
D0 I
← I0
Repeat
Get new perceptions **p B ←**
revision(B, p)
I ← options*(D,*
I)* D ← *des(B, D
,I)* I ← *filter(B,
***D, I)* PE ←**
plan(B, I)
Execute(**PE)**
until the agent is arrested
end.

Figure 3.4: The BDI agent control algorithm.

Suppose we have an attacking player who believes his intention is to exploit an attacking attempt to score a goal. The relevant question in the BDI model is how long should the agent hang on to this attempt or intention? Assuming that the opponent has exploited this attack attempt to make a counter-attack, does the attacker in this case have to return to his team's zone to help the defenders or does he wait in his opponent's zone believing that there is hope of completing his initial attempt. The answer to this question is linked to **the bond** strategy. In fact, an agent who follows a **strategy of blind obligation** maintains his intentions until they are realized. On the other hand, an agent with a limited obligation **strategy** will maintain his intentions until they are realized, or until they are impossible. In the third strategy (open **obligation** strategy), the agent will maintain his intentions as long as they are still desires.

3.2 Subsumption architecture

We explained in the previous chapter that reactive agents are entities with simple behaviors. These behaviors are generally stimulus-response in nature. Intelligence emerges from the interaction between agents and the environment. Brooks, one of the best-known researchers in the reactive school of artificial intelligence, has proposed an architecture of reactive agents called "subsumption architecture".

This architecture is made up of a set of modules, each responsible for carrying out a simple task. These modules are called competency modules because each module is responsible for executing a specific behavior. Figure 3.5 shows this architecture. Note that the modules are organized in layers, with each lower layer having a higher priority than the layers above. In fact, each layer has a simpler, more urgent task compared to the tasks of higher layers. To ensure such layer control, a lower layer can suppress inputs from a higher layer to inhibit its execution.

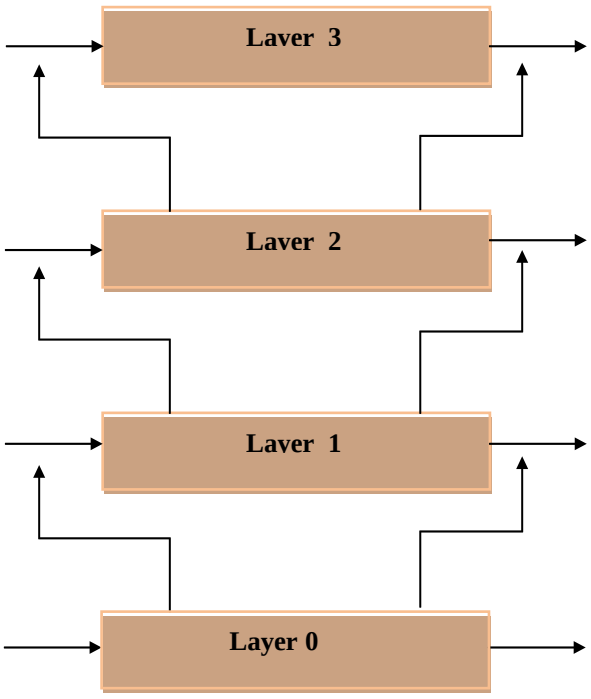

Figure 3.5: Subsumption architecture.

Each task is described by a set of behavior rules. A behavior rule is specified by a condition and an action. The condition is activated by the perception of the environment, and the action changes the state of this environment.

The reactive school of artificial intelligence (sometimes called the actionist school) is applied particularly in the field of robotics. As an example, we can propose an architecture for an explorer robot based on this architecture. The robot will be composed of the following layers:

- **Layer 0:** this is responsible for dynamic obstacle avoidance behavior. So, if the robot detects an object moving towards it, it will move rapidly in a vertical direction with the object's movement.
- **Layer 1:** this layer is responsible for avoiding static objects. If an object detects a static object, it will stop moving. Of course, if the robot detects two objects (one moving towards it and the other static), then it will disable the operation of this layer (according to the principle of subsumption architecture) in order to avoid the dynamic object in the first place (the operation of layer 0 takes priority).
- **Layer 2:** This layer is responsible for avoiding a static obstacle, given that the agent is dynamic. In this case, the robot will change direction and continue moving. As in the case of the previous rule, if the agent is moving and detects a static object, it will deactivate this rule to allow the previous rule to be executed, because the agent cannot brake and turn at the same time.

3.3 Turing Machine architecture

A hybrid architecture is one that combines the agent's proactive and reactive behaviors. Pro-active behaviors are behaviors performed when the agent takes the initiative. They are therefore goal-directed behaviors. Reactive behaviors, on the other hand, are behaviors directed by changes in the environment. Combining the two behaviors can give us an agent with more capabilities. In fact, the decomposition of agents into cognitive agents and reactive agents is a unrealistic decomposition. A human being is an example of an agent that is both cognitive and reactive.

Hybrid architectures are layered architectures. There are two types of hybrid architecture: vertical and horizontal. Horizontal architecture is similar to subsumption architecture. It's made up of a set of layers, with perceptions transmitted to all layers. In addition, all layers are connected to effectors. Each layer is responsible for a specific behavior. The problem with this architecture is consistency of results. In fact, compared with subsumption architecture, a layer in horizontal hybrid architectures does not have the ability to inhibit the operation of higher layers. As a result, all layers can generate actions that may be inconsistent for the same perceptions. To avoid this problem, these architectures feature a control module.

The role of this module is to choose the action to take in relation to each perception captured. Of course, centralized control poses problems of complexity, because the designer has to consider all possible cases. What's more, this control module represents a bottleneck in this architecture. The ***TuringMachine architecture*** is an example of an architecture in this category. As shown in Figure 3.6, this architecture is composed of three layers:

- **The reactive layer:** which ensures reactive behavior using stimulus-response rules (as in the case of subsumption architecture);
- **The planning layer:** ensures proactive agent behavior. To achieve the agent's objectives, this layer generates plans from a plan library. As in the case of the BDI architecture, the plan library saves skeletons of plans that the agent must refine to achieve its own goals;
- **The modeling layer:** in this layer, the agent has a representation of the system's agents (including itself). This layer is responsible for anticipating and managing conflicts between agents. Using this layer, an agent can generate new goals to avoid conflicting situations. Of course, the newly adopted goals will be transmitted to the planning layer to generate their appropriate plans.

The operation of these three layers is managed by a control module as explained above. This module is implemented in the form of control rules that hide perceptions captured from specific layers or suppress actions suggested by certain layers.

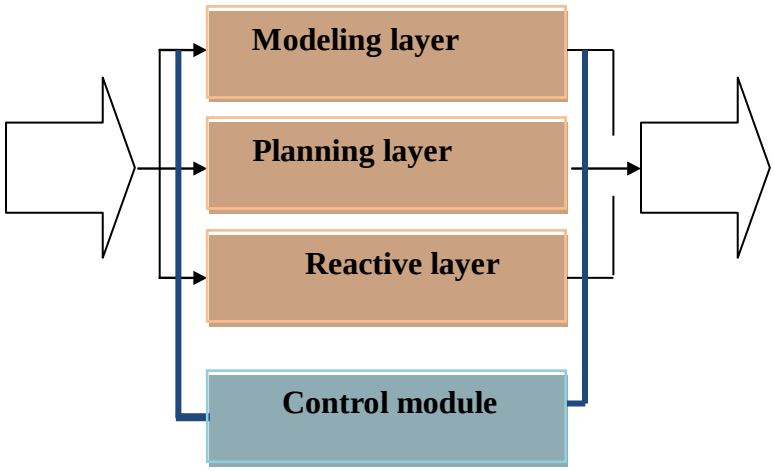

Figure 3.6: The TuringMachine architecture.

3.4 InteRRaP architecture

In contrast to horizontal hybrid architectures, vertical architectures capture perceptions through a single layer. The captured information can then be transmitted from one layer to the next. If the actions are generated by the uppermost layer, we speak of a single-pass vertical architecture. On the other hand, if the information is transmitted to the higher layers, then the decision is transmitted to the lower layers, so that the action can be carried out by the same layer that provides the perception - we speak of a two-pass layer. The **IneRRaP (*Integration of Reactive Behavior and Rational Planning***) system shown in figure 3.7 is a typical example of a two-pass vertical architecture. This architecture is made up of three layers of control, each serving a different purpose. Thus, the first layer ensures reactive objectives by executing simple actions according to the perceptual information. The second layer ensures local objectives (i.e. the agent has the skills to achieve these objectives). Finally, the top layer is responsible for cooperative planning (i.e. objectives requiring the intervention of several agents). Each layer uses a knowledge base layer that encompasses the knowledge required for its operation. Consequently, the reactive goals layer uses knowledge about the agent's environment (World Knowledge Base). The second layer (the local planning layer) uses a planning knowledge base that encompasses the agent's goals and plans. The third layer uses a social knowledge base that encompasses knowledge about other agents.

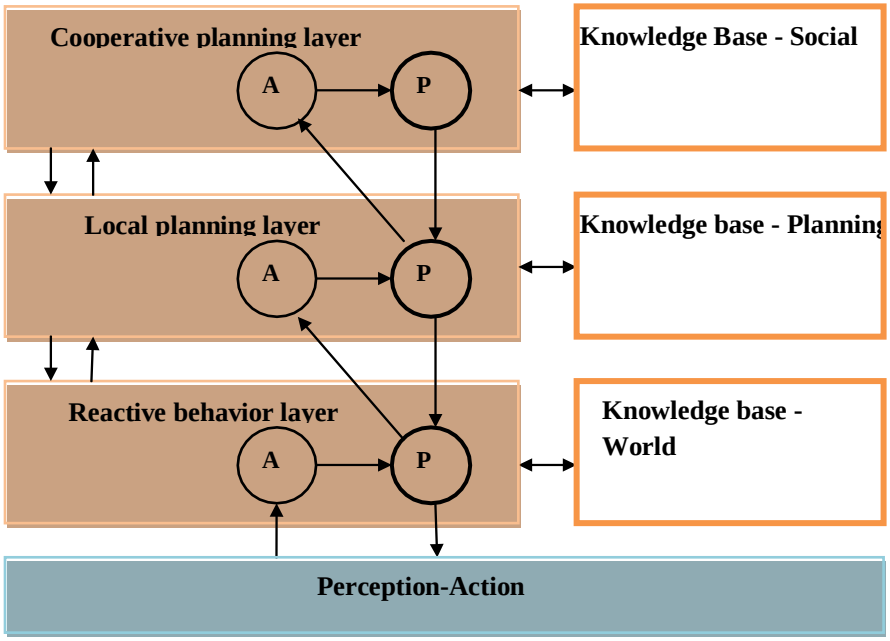

Figure 3.7: The IneRRaP architecture.

By capturing perceptions, the agent tries to respond to the situation using the lower layer. If this layer is able to resolve the situation, then it will generate a control flow to execute the appropriate action. Otherwise, the second layer will be activated to draw up a plan. Similarly, if the second layer is able to resolve the situation, it will generate a control flow to activate the shares. Otherwise, control will be passed to the next layer. The start of a layer's execution is ensured by a specific module that activates the goals and recognizes the situation (AR). This module receives information about the situation from the lower layer using the execution planning (EP) module, if the lower layer is unable to resolve the situation. Otherwise, the execution planning module will directly control the agent's actions.

4. Specific architectures

As we saw in the previous chapter, some agent characteristics are not intrinsic. In fact, these features may only have appeared in specific architectures. So we're talking about specific architectures. In this section, we present two specific architectures: the DIMA architecture and the ARTIS architecture.

4.1 DIMA architecture

Adaptability is an optional feature of agents. In fact, as we explained in the previous chapter, an agent can achieve its objectives without being adaptive. However, in certain fields of application, adaptability is an important feature. For example, the high level of the dynamism of an environment, or its unpredictability, requires the existence of an entity capable of changing its behavior or structure in response to these unpredictable changes. In the case of the robot explorer we explained in the previous sections, is it possible for designers to foresee all possible execution scenarios in the case of an unknown environment (e.g. Mars exploration)? Clearly, controlling all scenarios is a complex, if not impossible, task.

Adaptability enables agents to change their structure, their behavior or even their objectives. Changing the structure involves, for example, modifying its relationships (acquaintances) in order to achieve its objectives. In the case of an explorer robot, it can establish communications with space stations if unforeseen failures prevent it from communicating with the earth station. It's impossible for designers to foresee every possible failure situation, just as it's impossible for them to predict in advance the best space station to receive the robot's communications. So the best solution is to equip the robot with the ability to choose the best station for the situation. In the same way, an agent can change his behavior and/or goals according to his situation. Assuming that the robot's objective is to explore the planet in order to send plans to the ground station, and that the robot has detected traces of water on this planet (bearing in mind that the existence of water is the ultimate objective of all studies targeting the planet Mars); in this case, changing the robot's objective will be the ideal solution.

The DIMA (Développement et Implémentation de Systèmes Multi-Agents) agent model is an example of an adaptive agent. This agent is part of a project initiated by Guessoum at the University of Paris 6 in 1993. The main objective of the project is the modular development of hybrid agents. The DIMA model is designed around a main component called the proactive component. This component ensures the agent's autonomy and pro-activity, because it describes the agent's actions and goals. Thus, the agent will execute these actions until it has reached its goals. This component can then be enriched by modules that represent the agent's capabilities, such as the communication component. This component, for example, offers message box descriptions and message formats and the functions that ensure communication between agents.

Although DIMA was initially proposed as a hybrid agent, it was later extended with adaptive capabilities. The approach adopted in this model consists of using a meta-behavior level capable of controlling the execution of agent behaviors. This module manages interactions between the modules that make up the agent, and adapts the agent's behavior in real time to changes in the environment. Figure 3.8 shows an abstract view of the DIMA agent.

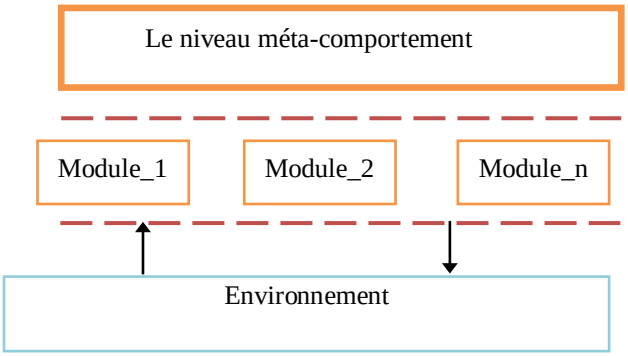

Figure 3.8: The DIMA agent.

The meta-level of the DIMA agent can be described using several artificial intelligence formalisms, such as rule-based systems, case-based reasoning, ATN (***Augmented Transition Network***), etc. For example, in the case of a NTD-based agent, states represent decision points. In each state, the agent must choose a transition. The transition is described as a condition and an action. The condition represents an event that may occur in the system (for example, the reception of a message). If this event has occurred, then the agent will execute the action to switch to another state.

4.2 ARTIS architecture

Another specific field of application for multi-agent systems is real-time intelligent systems. A real-time system is a system which requires a response by a certain ***deadline***. In order to satisfy real-time constraints, we generally estimate the program execution time on a worst-case basis. We'll need to design a system whose worst-case performance is lower than the deadline. However, this technique is unfeasible in systems based on artificial intelligence, due to the impossibility of predicting the execution time of an intelligent system. Multi-agent systems as a paradigm of artificial intelligence also suffer from this drawback. Several agent models have been proposed to take real-time constraints into account. The ARTIS agent is one such model.

The ARTIS model is an extension of the blackboard model, suitable for strict real-time environments. The agent architecture consists of two levels of agents: agents called ***ARTIS Agent*** (*AA*) and a set of agents called ***in-agent*** (for *Internal Agent*).

In keeping with Russell's definition of agents, ARTIS is an autonomous, reactive, proactive agent with temporal continuity. Other characteristics (such as communication) can influence the agent's strict real-time behavior. However, it's the designer who decides on the agent's characteristics, depending on the intended application.

An ARTIS agent consists of the following components (Figure 3.9):

- A set of sensors and effectors to ensure interaction with the environment. Perceptions and actions are time-bound processes.
- A control module responsible for the real-time execution of ARTIS components. It is divided into two parts: a reflex server for controlling reactive components and a deliberative server for controlling deliberative components. A detailed description can be found in.

Reflex level: ensures a response to any event. It is made up of the reflex levels of the in-agents that make up ARTIS.

- *Intelligent Server IS*: gives improved responses if there's enough time. Like the reflex level, it is formed by the cognitive levels of the in-agents.

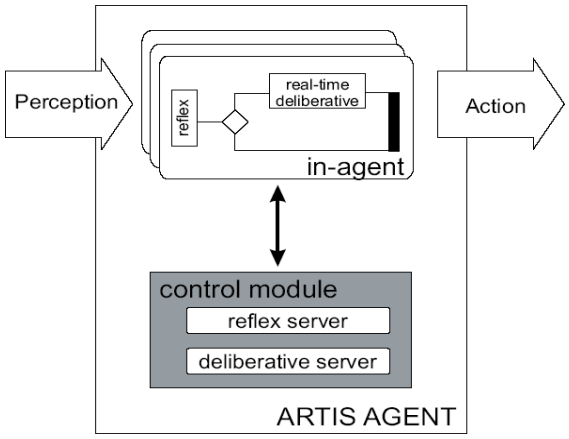

Figure 3.9: ARTIS agent architecture.

In the lower level, the in-agent represents an entity internal to ARTIS and encompasses the knowledge required to solve a particular problem. The main reason for breaking down the problem-solving method into smaller entities is to offer an abstraction whose solving knowledge can be organized in a modular, step-by-step way. This agent is characterized by: reactivity, strict real-time activity and in-agent collaboration via a blackboard. An in-agent consists of :

- **A reflex level:** to ensure a minimum response in a short time.
- **A cognitive or intelligent level:** where a response is produced by a deliberative process.
- **The action level:** this involves executing the responses produced.

Figure 3.10 shows how an in-agent works. This is because, after perceiving new information from the environment, an internal agent will update this knowledge. Next, it will activate the reflex control level to select a reflex action to execute. After specifying the reflex action, the agent will try to improve its reasoning by executing the cognitive control. The aim of this level is to select better quality cognitive actions. At the end of deadline, the agent will execute the cognitive action if the reasoning at this level has been completed, otherwise it will execute the reflex action. Knowing that the execution of internal agents is periodic, the internal agent will resume this cycle after a period D.

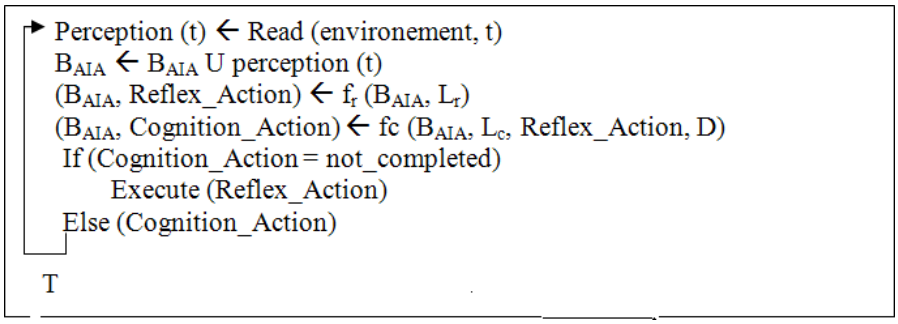

Figure 3.10: How an in-agent works.

5. Conclusion

This chapter is devoted to presenting examples of agent architectures. An agent architecture is a description of the internal components of an agent and how they work together to provide the agent's features. We have presented these architectures along several axes. Firstly, we have presented an abstract architecture in the sense that the agent is represented only as algebraic and logical functions. Next, we present some concrete architectures classified along the granularity axis. Finally, we presented two agent architectures with specific characteristics, namely, adaptation and real-time. In the next chapter, we'll look at the interaction between agents. Thus, we will present coordination models for cognitive agents.

1. Introduction

Interaction between agents is an important dimension in multi-agent systems. In fact, an agent is often designed with limited capabilities. Consequently, an agent without interaction will be a handicapped entity in the sense that it cannot achieve all its objectives. However, integrating the interaction between agents is not a simple task. Since the objectives of the agents in a system can be incompatible or contradictory, their interaction can create chaotic situations or harmful behavior. This chapter is dedicated to presenting the forms of interaction and the techniques that can ensure better interaction by eliminating undesirable situations. In the first section of this chapter, we present the different forms of interaction between agents. Then, in sections 3 and 4, we present the techniques of negotiation and cooperation respectively. Finally, we'll end this chapter with a conclusion.

2. Forms of interaction

Despite the importance of interaction between agents, but like almost all concepts in this field, there is no single classification for forms of agent interaction. In fact, we presented several interaction situations in the second chapter of this course, taking into account three criteria: compatibility of goals, sufficiency of resources and sufficiency of skills. However, these shapes can be grouped into a limited number of classes. A simple analysis of these situations shows that some shapes are in fact complex shapes made up of more primitive shapes. By way of example, a situation of pure collective competition is in fact one in which the agent is in a conflictual relationship with certain agents on the one hand, and in a cooperative situation with other agents on the other. The example of the military battle presented in Chapter 2 illustrates this situation. The agent is in fact in two different relationships with other agents. As a result, it is possible to classify interactions between agents into primitive forms.

Presenting primitive forms of interaction is only a partial solution to the debate, because even these primitive forms are not accepted by all domain specialists. For example, some specialists see communication as a form of interaction. We explained in Chapter 2 that communication is actually the medium of interaction. There is no interaction without communication, especially if we consider indirect communication via the environment to be a type of communication. In addition, skills can be considered as resources (internal resources). In this case, inadequate skills are just one specific type of resource shortage. Consequently, the most relevant criterion for classifying interactions between agents is goal compatibility.

In his book "*An Introduction to Multi-Agent Systems*", Wooldridge considers coordination to be the most generic form of interaction. This form is divided into two classes according to the criterion of goal compatibility: competition and cooperation. Competition represents the case where agents have incompatible or contradictory goals or interests. In such situations, negotiation is generally the best means of resolving conflicts known to human beings. In the case of cooperation, the agents have compatible goals. As a result, each agent will help other agents satisfy their goals.

It is important to note that this chapter is dedicated to the coordination of cognitive agents. Consequently, interaction is a conscious process carried out by the agents involved. Following this criterion, this chapter will omit forms of interaction in reactive agents. In fact, the interaction between reactive agents is not intentional, but the result of traces left involuntarily on their environment.

3. Negotiation

As we saw in the previous chapters, agents are autonomous, interacting entities. As a result, each agent will try to satisfy its goals. However, the incompatibility of agent goals may prevent all agents achieve their goals. For example, in an economic system, the seller's aim is to sell his goods at the highest price. At the same time, the customer is looking to buy this merchandise at the lowest possible price. In these cases, it's impossible to satisfy both customer and seller at the same time. This is known as a conflict situation. **Negotiation** is a mechanism inspired by the human model of conflict resolution.

In order to achieve the goal of negotiation (i.e. reaching a common agreement), process modeling must take several aspects into account. These aspects must be clearly and explicitly specified. These aspects are :

- **The language of negotiation**: we explained earlier that communication is the basis of interaction. As a result, agents cannot negotiate without exchanging messages. In addition, the language of communication must include performatives that express the primitives of communication. For example, using the language of communication, an agent must be able to express his or her opinion concerning a proposal (acceptance or refusal), just as he or she must be able to propose or refuse participation in a negotiation.
- **Negotiation protocol:** a negotiation protocol is a set of rules used to manage the negotiation process. Since negotiation protocols represent a subset of interaction protocols, these negotiation protocols specify a subset of valid responses or interactions for each instant in the negotiation process. For example, on receipt of a request to participate in a negotiation, the agent has the right to participate or refuse participation. It is unacceptable and invalid to send the initiator a failure message at this level, by for example. In addition, the protocol determines whether the negotiation is terminated successfully or unsuccessfully.

- **The object of negotiation**: it is important to specify the object of negotiation clearly and explicitly before starting the negotiation process. Otherwise, the agent will be a lost entity with no purpose. A selling agent who minimizes the price of product "X" to reach an agreement with an agent who is looking for product "Y" and who is not interested in product "X" is an irrational agent. Note that the object of negotiation can be specified by a single attribute (e.g. price) or several attributes (e.g. product type, color, quality and price).
- **The decision-making process:** of course, the agent's decisions during the negotiation process will not be random. An agent must follow a specific strategy during this process, called the negotiation strategy. This strategy enables an agent to determine the right decision at any given time. For example, if a buyer is negotiating to buy a product at the lowest possible price, when does he decide that the price offered by the seller at a given moment is the minimum price? To make these decisions, the agent must be able to reason about the reasoning of other agents. It must also be able to identify the strategies followed by other agents. A chess player reasons not only about his possible moves, but also about his opponent's possible moves.

Negotiation is a well-known mechanism in many fields, such as economics, politics and game theory. As a result, several negotiation methods and protocols have been proposed. Of course, most of these techniques can be adopted in the field of intelligent agents. However, a designer must choose the best protocol for each system or application domain. In order to decide on the best protocol, we need to be able to evaluate the different protocols on offer. In fact, several criteria have been proposed in the literature for evaluating negotiation protocols, such as :

- **Individual rationality:** a protocol is considered rational for an agent if participation in the negotiation implies a gain for that agent. In other words, the agent loses nothing by taking part in the negotiation. It is obvious that an autonomous agent seeks to increase its utility function through participation in negotiation. As a result, if participation in a protocol does not increase its earnings, there is no reason to participate in the protocol.

- **Pareto efficiency:** the problem posed by the previous criterion is the inconsistency of agents' earnings. Consequently, it is impossible to compare or calculate the sum of agents' gains in order to decide the community's utility. For example, considering the players on a soccer team as agents competing to secure a place on the team taking part in matches. These players have heterogeneous goals, such as money, fame, career advancement, and so on. It's impossible to compare these goals to decide on the best strategy for the team. Consequently, the Pareto efficiency criterion is a solution to this drawback. In this case, comparisons are made in relation to each agent's earnings. A solution to a trading problem is said to be Pareto-optimal (Pareto-efficient), if the gains of one agent cannot be improved without a negative effect on the other agents. In other words, if we try to to improve one agent's gains in an alternative strategy, then another agent will lose out on his previous gain.
- **Balance:** we said earlier that an agent in an interaction situation must be able to reason about the possible actions of other agents. In wars, strategic games, economics or soccer, an agent has to consider his opponent's possible strategies to increase his profits. However, the pertinent question is how can we identify the opponents' strategies? The answer to this question is proposed in the field of game theory in the Nash equilibrium theorem. Two strategies are said to be in Nash equilibrium, if one agent's choice of the first strategy makes the second strategy the best choice for the second agent. On the other hand, if one agent has chosen the second strategy, then the first strategy will be the best choice for the other agent.
- **Computational simplicity and efficiency:** a protocol is implemented in the form of a distributed program. Reasoning about all possible situations and strategies can complicate calculations. It's even possible to generate combinatorial explosion situations in the search for an optimal strategy. Simplicity is one of the criteria used to evaluate negotiation protocols. By simplicity we mean the ease with which agents can identify the best strategy. In addition, the computational resources (computing time, memory space, etc.) used to determine the optimal strategy must be reasonable.

Several negotiation techniques have been applied to multi-agent systems. In this chapter, we present two techniques: auctions and the task redistribution protocol.

3.1 Bidding techniques

An auction is a negotiation mechanism in which an agent puts several other agents in competition to buy or sell an item. This competition will enable us to determine the price of the item objectively. This mechanism has been known since ancient times. In fact, Herodotus describes this mechanism existing in Babylon. Today, this mechanism is applied in several online sales sites such as eBay.

The general auction philosophy is the proposal of prices by participants. Then, the initiator will decide on the best proposal. Of course, the number of rounds (i.e. the number of times the agent can propose a price), the method of announcing proposals (public or private) and the method of choosing the winner are the features that make the difference between auction protocols. However, some features are common to most protocols, such as the existence of a reserve price. This price is specified by the initiator as the object's minimum price. Thus, participants are not allowed to advertise a price below the reserve price. If all participants offer prices at the reserve price, the protocol will fail.

An agent generally applies a strategy to decide on a price to offer. There are auction protocols with dominant strategies (i.e. strategies that ensure the object is obtained at a better price), just as there are protocols where there are no dominant strategies. However, the choice of price is not only linked to the agent's strategy, but also to the value of the object in relation to the agent. There are objects with common values. In this case, the object's value is unique in relation to all participants. For example, to buy a computer, all participants will offer similar prices because the value of the computer is unique to all participants. Generally speaking, you won't find participants offering inflated prices for this product. On the other hand, an object can have a private value in relation to an agent. We sometimes hear in the media that a supporter has bought his favorite player's shoes for millions of dollars. In reality, the value of this object is estimated in this fair way by the agent concerned. Other agents may even consider this behavior absurd.

In multi-agent systems, there are four main types of auction:

- **English auction (first-bid-public):** as the name of the protocol indicates, agent proposals are public. In this way, each agent is aware of the proposals made by the others. What's more, the winning agent is the one offering the highest price. He'll buy the object and offer the proposed price. Consequently, the protocol is as follows. The bidder starts the auction by announcing the reservation price. Then, each agent will publicly announce a higher price than the last one offered. The protocol is successive rounds. Agents can propose prizes until a winning prize is proposed. A prize is a winner if no agent can offer an increase over it. Let's take the example of a cell phone purchase. The initiator will announce the start of the protocol by announcing the reservation price (e.g. 1000). Two participants are assumed. A participant (P1) will offer a price higher than the reservation price (1100 for example). Next, participant P2 will offer a higher price than the previous one (e.g. 1150). The protocol will continue with prizes in each round. If one of the agents proposes a price X (e.g. 2000), knowing that the other agent has not been able to propose a price higher than this price, then the agent who proposes price X will be considered the winner. In this case, he has to pay price X to get the cell phone. The best strategy in this protocol is to increase the price by a small amount. In the case of a private value, the agent must increase the price by a small value as long as the proposals are lower than his private value. If the proposals exceed his private value, he withdraws from participation. Thus, this strategy is considered to be a dominant strategy because, at the end of the protocol, the agent can either acquire the object at a price lower than or equal to its private value (i.e. he wins), or he arrives at a state where he can't buy the product because the object's value is higher than his private value.gal to his private value (so he wins) or he reaches a state where he can't buy the product because the object's value is higher than his private value. Of course, even the second case is relatively a win-win situation, because the agent has not dispensed more than the estimated value of the object. However, in the case of a private value, this protocol suffers from the disadvantage of a winner's curse. This situation arises because the agent assumes that he is offering a price higher than the object's value.

- **First bid-hidden auction:** in this type of auction, the bidder begins by announcing the item and its reservation price. Participants then start sending in their proposals. Participant bids are hidden offers. In this way, one participant cannot know the proposals of the others. After a deadline, the initiator begins evaluating different proposals, then announces the name of the winning agent and his/her proposal. The winning agent is the agent who offers the highest price. As a result, the winning agent will win the item by paying this price. For example, an initiator wants to sell a cell phone. It sends participants the phone type and reservation price (for example, "type = Galaxy S4, reservation price = 1000"). Participants then start sending in their proposals. Three participants p1, p2 and p3 are assumed to offer prices 1100, 1150 and 1050 respectively. After receiving the various proposals, the initiator announces that the winning participant is p2 with prize 1150.

A participating agent estimates the object's value according to its private value or resources. Of course, he doesn't offer a proposal higher than its value, because even if he wins with this proposal, he won't be able to pay the price of the object (because it exceeds his resources) or because he finds the price of the object higher than its real value. Consequently, the best strategy is to offer a price lower than or equal to the private value (e.g. price = private value - e). However, there is no single strategy for setting the right price. In the previous example, assuming that participant p1 can pay up to 1200, but has chosen to offer 1100. After the results were announced, this participant (p1) considered that he had been able to obtain the item at a price of 1160. Similarly, participant p2, after the results are announced, considers that he could have won with a prize of 1101 instead of 1150.

- **Dutch auction (descending):** in this case, the bidder begins by announcing a high price. Then, he will lower the price each round until there are no proposals. If a participant believes that the value proposed by the initiator in a given round represents the value of the object or is within its capabilities, then it announces that it is buying the item. Thus, the initiator declares this agent the winner with the prize for this round. This protocol is equivalent to a first-price-bid-hidden auction. In our previous example of a cell phone deal, the initiator starts by announcing the item and a price (e.g. "type = Galaxy S4, price = 2000"). Then, it will progressively decrease the price with each turn (for example, 1900, 1850, 1800, 1700, 1600, etc.). When the initiator offers an attractive price to a participant, the latter announces that he or she is buying the item. For example, at turn "*i*", the initiator proposes a price of "1200", which is attractive to participant p2. As a result, p2 announces that he has bought the item and pays 1,200. As with the previous protocol, there is no single dominant strategy. A participant listening to an interesting price can behave in two different ways. Firstly, he announces that he is buying the item, but in this case the attractive prices offered by other agents may still be a long way off. So, he will have been able to wait to buy the item at a lower price.
- **Vickery auction (second-bid-hidden):** this protocol is similar to the first-bid-hidden protocol, except that the winner doesn't pay the price he offers, but may pay the price of the second bid received by the initiator. Thus, an initiator starts with the object announcement and the reservation value (for example, "type= Galaxy S4, price = 1000"). Then it starts receiving proposals. Assuming that after the deadline the initiator has received proposals from p1 participants, p2 and p3 with prices 1100, 1150 and 1050. In this case, the initiator announces that p2 is the winner because he has offered the highest price. But p2 has to pay 1100 (the second proposed price) instead of paying his proposal. Although this protocol may seem illogical for human communities, it has an important advantage that makes it the most suitable for agents.

3.2 Task allocation by redistribution

This technique models the agent interaction problem as a task allocation problem. This problem was studied by Rosenschein and Zlotkin in 1994. Take the following example: two neighbors have children who attend different schools. Every neighbor needs to take his children to school. Assuming the first neighbor has two children, C1 and C2, who attend schools S1 and S2 respectively. The second neighbor has three children C3, C4 and C5 who attend schools S1, S2 and S3 respectively. Although both neighbors are able to accompany their children, this situation can lead to a loss of resources. For example, the journey will be long and the children may arrive late. Naturally, both neighbors can see the situation and decide to coordinate their actions to preserve their resources. For example, one neighbor will be responsible for accompanying children C1 and C3 to school S1, and the other will be responsible for accompanying children C2 and C4 to school S2 and C5 to school S3. Assuming now that the journey to school S1 is longer than the journey to schools S2 and S3. Naturally, each neighbor will seek to be the one responsible for accompanying the S2 and S5 schoolchildren. As a result, both neighbors will be in a competitive situation. The problem is how to allocate tasks to different neighbors.

Rosenschein and Zlotkin call a task allocation problem a task-oriented domain. The latter is formalized by the triple **<T, Ag, C>** such that:

- **T** is a set of tasks (**$T = \{T_1, T_2, \ldots T_m\}$**) ;
- **Ag** is the set of agents participating in the interaction situation (**Ag ={1, ...,n}**);
- **C:** is a cost function that defines a positive real value for each subset of tasks **($C : 2^T \rightarrow R^+$).** This function must satisfy two conditions:
    - The cost of not performing a task is always zero (C(Ø)=0);
    - The cost function is a monotonic function. In other words, if a task is added to a set of tasks, the new cost must be greater than the cost of the initial set.

In order to distribute tasks across all agents, we have defined a meeting in a task-oriented domain. A meeting is a set of task sets, each of which represents the tasks allocated to a specific agent. Formally, we write an encounter $\boldsymbol{R = \{E_1, E_2, \ldots E_n\}}$ such that $\boldsymbol{E_i}$ is the set of tasks allocated to agent $\boldsymbol{Ag_i}$.

The problem posed in such an encounter is the possibility of redistributing tasks to find a more interesting distribution for the agents. For example, in the neighbor example, the encounter is R $\{_{E1=\{T1,\ T3\}},\ _{E2}\{\ _{T2},\ _{T4},\ _{T5}\}\}$ such that $_{Ti}$ represents the accompaniment of child $_{Ci}$. Assuming that school C3 is on the same route as school C1. In this case, it will be more interesting for the agents to redistribute tasks so that one neighbor accompanies children C1, C3 and C5 to schools S1 and S3 respectively, and the second child accompanies the rest of the children to school S2. So agents have to negotiate among themselves to achieve this reallocation.

A reallocation of tasks is called a deal. Thus, a case in a two-agent system is A = $\{_{D1},\ _{D2}\}$, with the proviso that this redistribution must not omit any task. Consequently, $_{D1}$ U $_{D2}$ = $_{E1}$ U $_{E2}$.

Since a case or a meeting is a subset of a task, we can apply the cost function to it. We write $_{Ci}(_{Dj})$ to designate the cost of case $_{Dj}$ in relation to agent *i*. We can calculate an agent's profit by adopting a deal versus a meeting by comparing the costs of the deal and the meeting. This profit can be calculated using the **utility function.** Thus, this function is calculated by: utilityi(A) = $_{Ci}(_{Ei})$ - $_{Ci}(_{Di})$. For example, if cost represents the travel distance to accompany children to a school, then utility represents the difference in journeys between the meeting (the first distribution) and the business (the result of the redistribution of tasks). If this difference is negative, then the agent will lose out by adopting the business. In this case, the agent will prefer to run its initial set.

A case $_{A1}$ is said to **dominate** another case $_{A2}$ **<u>if and only if :</u>**

- For both agents, the benefit of adopting case $_{A1}$ is at least equal to the benefit of adopting case $_{A2}$.

  $i \in \{1, 2\}, utilité_i(A_1) \geq utilité_i(A_2)$

- For at least one agent, the benefit of adopting case $_{A1}$ is strictly greater than the benefit of adopting case $_{A2}$.

  E $i \in \{1, 2\}, utilité_i(A_1) > utilité_i(A_2)$

The monotonic concession protocol ensures better task allocation. In this protocol, negotiation takes place in a series of rounds, with the number of rounds being limited. As a result, if no business has been adopted in round N, the protocol will be terminated and the conflict will persist. In a given round, each agent will propose a deal ($_{A1}$ by agent 1 and $_{A2}$ by agent 2). Agents reach an agreement if the benefit of an agent *i* brought by adopting the deal proposed by the other agent is greater than or equal to the benefit of adopting its own deal.

Of course, an agent doesn't offer a deal that isn't useful to him or her. So, if the other agent finds this proposal more beneficial to him than his own business, he'll adopt it. If each deal proposed by an agent is more beneficial to the other agent than its own proposal, then one will be adopted at random. If no agreement is reached, we move on to the next round. In this new round, no agent is allowed to make a proposal that is less preferred by the other agent than the deal he proposed in the previous round.

4. Cooperation

Relations between agents are not always conflictual. On the contrary, agent relationships can also be positive relationships. By positive relationships, we mean increasing agent synergy or managing the interdependencies of agent tasks. In fact, the existence of these relationships is justified by the non-existence of an agent who possesses all the skills and knowledge needed to solve the problem. Agents in these situations resort to cooperative techniques. Generally speaking, we speak of cooperation when there is no conflict. For example, to write a paper scientific, the co-authors have a single objective: to write a quality paper. Consequently, the co-authors are in a cooperative relationship. Each agent can help other agents with their tasks (e.g., proofreading and correcting paragraphs) or managing interdependencies (e.g., one agent is in the process of preparing the results of experiments, another must wait for the completion of this part to write the corresponding part in the paper).

Cooperation problems were dealt with in the field of concurrent systems before the advent of multi-agent systems. However, there are two crucial differences between cooperation in multi-agent and distributed systems. On the one hand, systems distributed around a shared objective. In other words, all the components of a system have the same objective. In multi-agent systems, agents are not always required to have the same goal. Each can have its own objectives. What's important is that the agents' objectives are compatible. For example, in the case of soccer players, each player may have his own objective (money, fame, being called up to the national team, etc.), but these are compatible. On the other hand, in distributed systems, cooperation situations are identified at the design stage. It's up to the designers to decide when agents should cooperate and what technique they should use (for example, where to put process synchronization code). In the case of agents, on the other hand, decisions are made at runtime, because the system is dynamic. It's impossible to foresee every possible cooperation situation. It's up to the agent to decide how he or she can work to help other agents achieve the goal. In the case of soccer players, it's impossible to foresee every possible interaction situation to win the match. A player in possession of the ball must decide at this moment (execution time) whether his best action is to attack or pass the ball to his colleague.

Several techniques have been proposed to ensure agent cooperation. In the remainder of this chapter, we present two categories of techniques widely applied in this field: task allocation by contractual networks and cooperation by planning.

4.1 Task allocation by contractual network

Contract network protocols are high-level protocols for efficient cooperation through task allocation. The proposal of these protocols is based on the metaphor of organizations during the realization of contracts. In fact, in an organization, there is usually a manager who breaks down the task to be carried out into a set of sub-tasks. It then assigns these tasks to a set of contractors according to a specific protocol. Once the task has been completed, the contractor informs the manager of the result. For example, in the context of doctoral training, a research supervisor can break down a research project into a set of sub-problems. It then assigns these sub-problems to the doctoral set. Each PhD student will then work on his or her own sub-problem. After proposing different solutions, the research manager can synthesize the results to form the solution to the initial research problem.

In the context of multi-agent systems, Contract Net is considered one of the first techniques for agent cooperation. The FIPA organization has standardized the FIPA-Contract-Net protocol. As shown in figure 4.1, the manager begins by announcing the task using the performative CFP (*Call-For Proposal*). Then, the contractors can accept participation in the protocol by making a proposal (price of the task, quality of the task, duration of execution, etc.) or refuse participation. After receiving responses from various contractors, the manager will evaluate their bids. As a result, it will choose the best proposal by sending a proposal acceptance message to the proposal owner contractor, and it sends a proposal rejection message to the other contractors. Once the contractor has carried out the task, he will send the result of the task's execution, or simply inform the manager that he has completed the requested task. In some cases, the contractor may encounter problems that prevent him from completing the task. He must inform the manager of this situation.

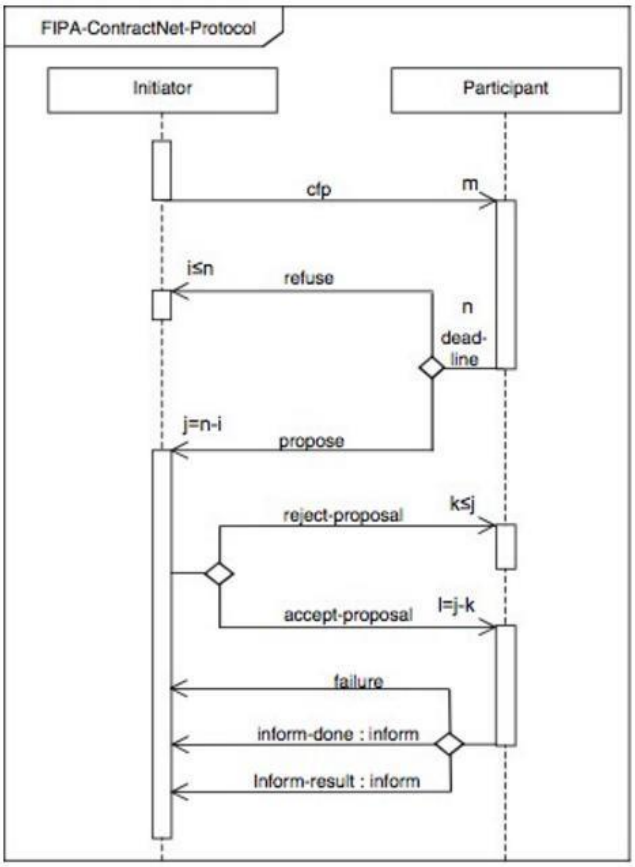

Figure 4.1: The FIPA-Contract-Net protocol.

4.2 Cooperation through planning

Cooperation means that agents have compatible goals, or share the same goal. Thus, agents must cooperate to ensure rational use of resources or to avoid unnecessary actions. For example, in the case of a scientific paper, the authors have the same objective. In this way, they cooperate to avoid unnecessary actions. For example, a single agent can handle paper formatting. There's no point in having several agents perform the same action, because the result is the same. Also, working on a single copy requires cooperation, because it's impossible for all agents to work on the same copy. Consequently, the cooperation problem is in fact a problem of scheduling agent actions over time. Indeed, agent cooperation can be treated as an agent planning problem. The aim is to organize agents' tasks according to time.

In the example of writing a scientific paper, assuming we have three authors responsible for the following actions: $A_1 = T_1, T_2, T_3$; $A_2 = T_4, T_5$; $A_3 = T_6, T_7, T_8$; so the aim is to specify a global order for these actions, specifying which actions can be executed in parallel and which require synchronization.

In multi-agent planning, two classes of techniques can be distinguished: centralized *planning* and distributed planning. In centralized planning, a single agent manages the planning process. This agent creates a plan, then organizes it into potentially synchronized sub-plans. He then transmits these plans to all the agents who execute them in competition. For example, to write a scientific paper, the project manager draws up an overall plan and then divides it into sub-plans. Each sub-plan must be executed by a specific author. Tasks can be run in parallel, e.g. each author must write a section. It is also possible to find tasks that require synchronization, e.g. an author cannot correct a section unless that section is written by another author. Each author will execute his own plan. Of course, the formalism used to describe plans must support the description of parallelism and synchronization. For example, it's important to use formalisms such as Petri nets.

5. Conclusion

This chapter is devoted to the presentation of coordination models for cognitive agents. Coordination encompasses interaction situations with compatible or incompatible goals. Negotiation is the most widely applied interaction mechanism for dealing with situations involving incompatible goals. Under this heading, we have presented bidding techniques and task allocation through redistribution. On the other hand, we presented two cooperation techniques, namely contractual networks and cooperation by planning. Of course, the field of agent interaction is a broad one, with a growing number of techniques. The techniques presented in this chapter represent just a sample of those available in the literature.

Books

1. Jacques Ferber, *Multi-agent systems: towards collective intelligence,* InterEditions, 1997.
2. Michael Wooldridge, *An Introduction to MultiAgent Systems,* John Wiley & Sons Ltd, 2002.
3. Nikos Vlassis, *A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence,* Morgan & Claypool, 2007.
4. Jean-Pierre Briot, Yves Damazeau (Eds), *Principles and architectures of multi-agent* systems, Hermes Science Publications, 2001.
5. Gerhard Weiss, Multiagent Systems - *A Modern Approach to Distributed Artificial Intelligence* , MIT Press, 1999.
6. Paulo Leitão, Stamatis Karnouskos (Eds), *Industrial Agents - Emerging Applications of Software Agents in Industry* , Elsevier, 2015.
7. Stuart Russell, Peter Norvig, *Artificial Intelligence - A Modern Approach*, Prentice Hall, 2010.

Articles

1. Stan Franklin, Art Graesser, *Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents* , Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.
2. Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, Raj Reddy, *The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty* , ACM Comput. Surv. 12(2): 213-253 (1980).
3. Seyed H. Roosta, *Artificial Intelligence and Parallel Processing* , Parallel Processing and Parallel Algorithms, pp 501-534, 2000.

Theses

1. Marir Toufik, *Une démarche d'assurance de qualité pour les* systèmes *multi-agents*, PhD thesis in science, University of Annaba, 2015.
2. *Olivier Gutknecht, Proposition d'un modèle organisationnel générique de systèmes multi-agents et examen de ses conséquences formelles,*