

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



Intelligent Systems Engineering (ISE)

Course

Prepared by: Dr. *Mostefa Bendjima*

February 2026

Chapter 01: Introduction to Distributed Artificial Intelligence

1. Introduction	03
2. From artificial intelligence to distributed artificial intelligence.....	03
3. Distributed artificial intelligence.....	05
4. Branches of distributed artificial intelligence.....	08
5. The differences between MAS and computer paradigms.....	11
5.1. The differences between MAS and object-oriented software.....	11
5.2. The differences between MAS and expert systems.....	12
5.3. The differences between MAS and distributed systems.....	13
6. The benefits of MAS.....	13
7. The challenges of MAS.....	15
8. MAS applications.....	15
9. Conclusion.....	16

Chapter 02: Multi-Agent Systems - General Principles

1. Introduction.....	17
2. Multi-agent systems (MAS).....	17
2.1. Definition of a multi-agent system.....	17
2.2. Specific cases of multi-agent systems.....	18
2.3. Control in multi-agent systems.....	19
3. The agents.....	20
3.1. Defining an agent.....	20
3.2. Agent characteristics.....	21
3.3. Agent types.....	23
4. The environment.....	24
5. The organization.....	26
6. Interaction.....	29
6.1. Types of interaction.....	30
6.2. Communication.....	32
7. Conclusion.....	33

Chapter 03 Agent Models and Architectures

1. Introduction.....	34
2. Abstract agent architecture.....	34
2.1. A purely reactive agent architecture.....	36
2.2. An agent with state.....	38
3. Concrete architectures.....	39
3.1. BDI architecture.....	39
3.2. Subsumption architecture.....	41
3.3. TuringMachine architecture.....	43
3.4. InteRRaP architecture.....	45
4. Specific architectures.....	46
4.1. DIMA architecture.....	46
4.2. ARTIS architecture.....	49
5. Conclusion.....	51

Chapter 04: Cognitive Agent Coordination Models

1. Introduction.....	52
2. Forms of interaction.....	52
3. Negotiation.....	54
3.1. Auction techniques.....	57
3.2. Task allocation by redistribution.....	61
4. Cooperation.....	64
4.1. Task allocation by contractual network.....	65
4.2. Cooperation through planning.....	66
5. Conclusion.....	67

List of figures

Figure 1.1: The HearsayII system.....	05
Figure 1.2: The multidisciplinary nature of MAS.....	05
Figure 2.1: The chalkboard model.....	19
Figure 2.2: The actor model.....	20
Figure 2.3: Agent properties by type.....	22
Figure 2.4: The layered architecture of an environment.....	25
Figure 2.5: The AGR model.....	28
Figure 3.1: description of a purely reactive agent.....	37
Figure 3.2: Architecture of a stateful agent.....	38
Figure 3.3: BDI architecture.....	39
Figure 3.4: The BDI agent control algorithm.....	40
Figure 3.5: Subsumption architecture.....	42
Figure 3.6: The TuringMachine architecture.....	44
Figure 3.7: The IneRRaP architecture.....	45
Figure 3.8: The DIMA agent.....	48
Figure 3.9: ARTIS agent architecture.....	50
Figure 3.10: How an in-agent works.....	51
Figure 4.1: The FIPA-Contract-Net protocol.....	66

Introduction

A multi-agent system is one of the core modeling paradigms in *Intelligent Systems Engineering (ISE)*, consisting 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 multi-agent system 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—a perspective strongly emphasized in *Intelligent Systems Engineering*.

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—properties that are explicitly targeted in the design principles of *Intelligent Systems Engineering*. 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, a principle central to resilient *Intelligent Systems Engineering* architectures.

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, all of which are fundamental topics within the field of *Intelligent Systems Engineering*.

Multi-agent systems are a major building block in *Intelligent Systems Engineering* and 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—an example of *Intelligent Systems Engineering* applied to urban mobility. 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, thus embodying *Intelligent Systems Engineering* in energy systems.

Designing multi-agent systems presents several challenges addressed within *Intelligent Systems Engineering*. 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 in *Intelligent Systems Engineering* to enable agents to make informed decisions under uncertainty.

From a theoretical perspective, MAS research in *Intelligent Systems Engineering* 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 central ideas in *Intelligent Systems Engineering*.

Learning and adaptation are increasingly important in MAS to improve performance over time a core theme of *Intelligent Systems Engineering*. 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, highlighting the role of *Intelligent Systems Engineering* in real-world, evolving systems.

The future of multi-agent systems is promising within the broader vision of *Intelligent Systems Engineering*, especially 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 line with the goals of *Intelligent Systems Engineering*.

In summary, multi-agent systems represent a powerful paradigm in *Intelligent Systems Engineering* 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 in *Intelligent Systems Engineering* 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 exactly the kind of systems that *Intelligent Systems Engineering* aims to conceive and deploy.

1. Introduction

In the context of Intelligent Systems Engineering (ISE), intelligent systems engineering are currently a very active and widely applied field. This field is a branch of Distributed Artificial Intelligence (DAI). It therefore seems logical to begin this course with an introduction to distributed artificial intelligence. We thus begin this chapter with a brief introduction to artificial intelligence, demonstrating the need to distribute the techniques within this branch. Next, we present distributed artificial intelligence, detailing the causes and forms of intelligence distribution. Since intelligent systems engineering represent only one branch of distributed artificial intelligence, we present an overview of the three branches that comprise this field in Section 3. In Section 4, we clarify the differences between intelligent systems engineering and the main paradigms and similar computer systems. Of course, intelligent systems engineering do not only offer advantages. We therefore present both the advantages and challenges of these systems before outlining their application areas. Finally, we present a conclusion and some questions for better 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 theories.

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.

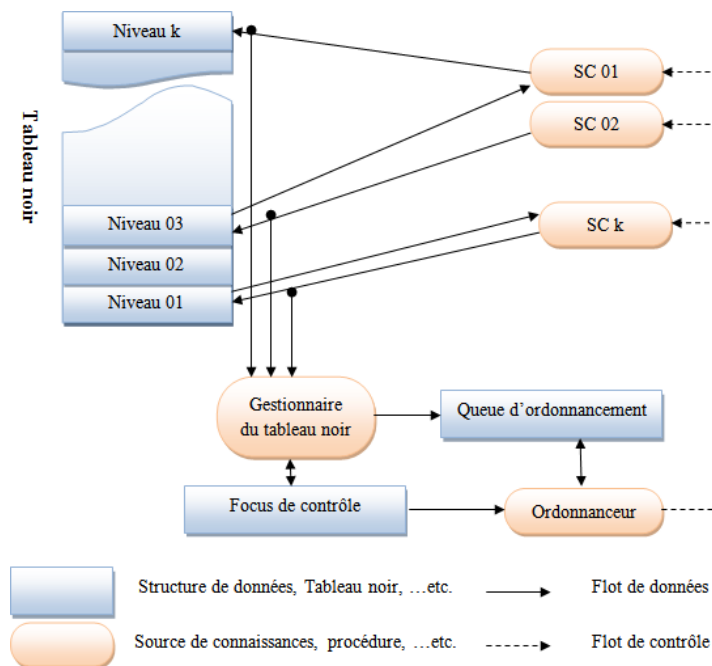


Figure 1.1: The HearsayII system.

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. 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?
- **Intelligent Systems Engineering (ISE):** 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 intelligent systems engineering is due, among other reasons, to their multidisciplinary nature. As we explained in In this chapter, ISE 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 intelligent systems engineering. Without doubt, intelligent systems engineering 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 ISE 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 ISE. This model will be detailed in chapter 03.

Considering intelligent systems engineering 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.

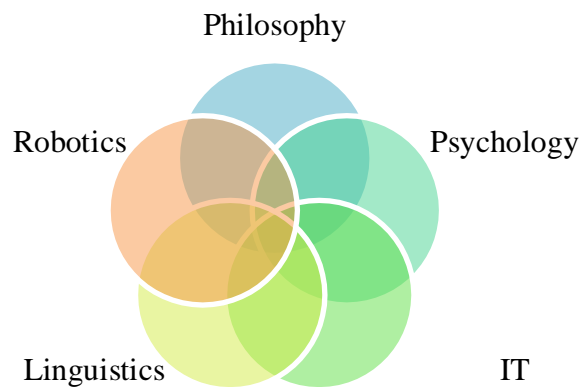


Figure 1.2: The multidisciplinary nature of intelligent systems engineering.

Despite the multi-disciplinary nature of the intelligent 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 ISE and computer paradigms

intelligent systems engineering 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 ISE, such as expert systems, distributed systems and object-oriented software.

5.1 The differences between ISE 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, an object 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 intelligent systems engineering, 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 ISE and expert systems

intelligent systems engineering 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 intelligent systems engineering 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 ISE and distributed systems

As mentioned above, intelligent systems engineering 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 intelligent 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 ISE

intelligent systems engineering 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 intelligent systems engineering is a consequence of the interaction of different fields (artificial intelligence, software engineering and distributed systems).

The advantages of intelligent systems engineering include :

- **Modularity:** a intelligent systems engineering 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 intelligent systems engineering contribute to their modularity. One such concept is "organization".
- **Reusability:** reusability is a direct consequence of the modularity of intelligent systems engineering. In fact, modularity makes it possible to reuse certain components of one software package in another. In the case of intelligent systems engineering, 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 intelligent systems engineering 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 intelligent systems engineering 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. Intelligent systems engineering 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 intelligent 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. Intelligent systems engineering 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. Intelligent systems engineering 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:** Intelligent systems engineering 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 ISE

Despite the advantages of intelligent systems engineering, this paradigm faces several challenges. In fact, these challenges are results of agent features and intelligent systems engineering. 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 intelligent systems engineering 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).
- Intelligent systems engineering engineering, by developing methods, methodologies, techniques and tools to facilitate the development of these systems.

8. ISE applications

Intelligent systems engineering 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 intelligent systems engineering. In fact, in this course we've chosen to mention those areas of application in which real applications have seen the light of day. Intelligent systems engineering 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 intelligent systems engineering 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.

- **The medical field:** the medical field in general, and surgical intensive care in particular, is characterized by several attributes that favor the use of intelligent systems engineering. 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 intelligent systems engineering. Thus, the three essential tracks of this field were presented, focusing on intelligent systems engineering. 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 intelligent systems engineering applications.

1. Introduction

Distributed Artificial Intelligence (DAI) is a convergence of several disciplines, such as psychology, sociology, biology, and computer science. These fields are used, among other purposes, to model and build computer systems in which processing, representation, and reasoning capabilities are distributed among a set of subsystems called agents. The collection of these agents forms a society known as a Multi-Agent System (MAS). Depending on their respective resources, agents interact with one another to build knowledge in order to solve problems.

In this module, we present an overview of distributed artificial intelligence and multi-agent systems. We will emphasize important aspects of these systems such as agent architecture, interaction and cooperation, communication, organization, and more. Finally, we will provide an overview of the MAS approach in sensor networks to enhance their security.

2. Intelligence artificielle distribuee

Artificial Intelligence (AI) is recognized as a field of computer science whose objective is to model or simulate so-called intelligent human behaviors such as design, decision-making, understanding, learning, and so on. It focuses on building computer programs capable of performing complex tasks, relying on the centralization and concentration of intelligence within a single system. However, AI quickly encountered a number of difficulties, mostly due to the need to integrate, within a single knowledge base, the expertise, skills, and knowledge of different individuals who, in reality, communicate and collaborate to achieve a common goal.

Distributed Artificial Intelligence (DAI) emerged in the early 1980s from the desire to overcome these limitations and enrich the classical AI approach by proposing the distribution of expertise across a group of agents that are not subject to centralized control. These agents must be capable of working and acting within a shared environment and resolving potential conflicts. In summary, DAI is concerned, among other things, with modeling intelligent behaviors that result from cooperative activity among multiple agents, leading to the development of so-called “multi-agent systems.”

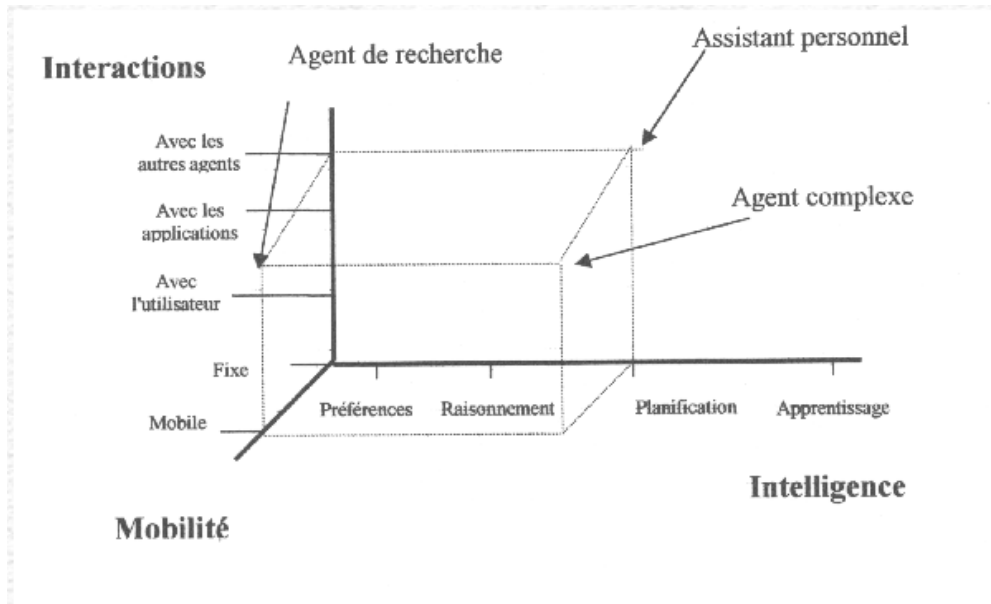
2.1 What Is an Agent?

The concept of an agent has been studied for several decades across different disciplines. It has been used not only in knowledge-based systems, robotics, natural language processing, and other areas of artificial intelligence, but also in disciplines such as philosophy and psychology. Today, with the advent of new technologies and the expansion of the Internet, this concept is associated with many new applications such as resource agents, broker agents, personal assistants, interface agents, ontological agents, and more. In the literature, there are numerous definitions of agents. While they are similar, they differ depending on the type of application for which the agent is designed.

- An agent is an autonomous entity, real or abstract, capable of acting upon itself and its environment. In a multi-agent world, it can communicate with other agents, and its behavior results from its observations, knowledge, and interactions with other agents. This definition highlights key properties such as autonomy, action, perception, and communication. Other properties may also be attributed to agents, particularly reactivity, rationality, commitment, and intention [Ferber, 1995].
- An agent is a computer system situated within an environment that acts autonomously in order to achieve the objectives (goals) for which it was designed.
- An agent is an entity that perceives its environment and acts upon it.
- Intelligent agents are software entities that perform operations on behalf of a user or another program, with a certain degree of independence or autonomy. To do so, they use some form of knowledge or representation of the user's goals or desires.
- An agent is an entity that operates continuously and autonomously within an environment where other processes are taking place and other agents exist.

2.2 Fundamental Characteristics of an Agent

The three main attributes of an agent are intelligence, interaction, and mobility. We will also mention a few other attributes. Each type of agent can be categorized according to its characteristics.



✓ Intelligence

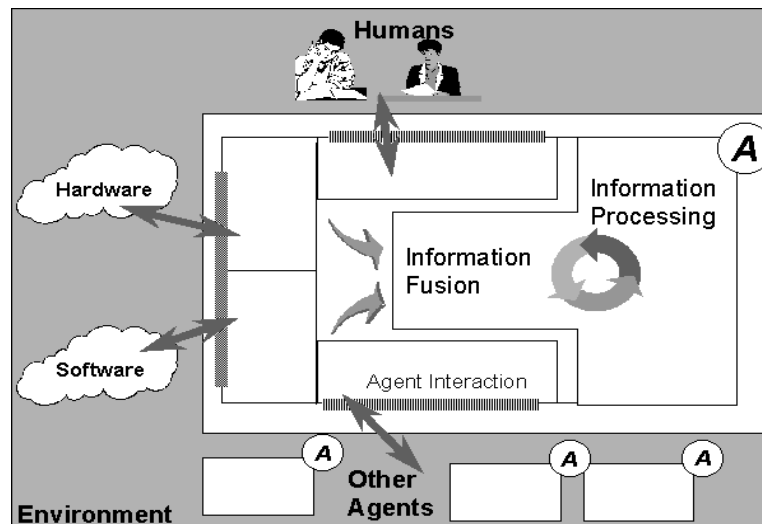
Intelligence is the faculty of reasoning and learning. It is the agent's ability to accept user requests and successfully complete the task delegated to it. At a minimum, the user must be able to specify their preferences, for example, in the form of rules, with a reasoning mechanism that acts based on these preferences. At a higher level of intelligence, the agent should understand what the user wants and plan the means to achieve this goal. On the intelligence axis of the graph above, the email sorting agent cited as an example could be at the preference level if the user must specify a number of parameters for the sorting to be carried out according to their wishes. If the agent, by observing the user's behavior with their emails, discovers their habits and acts accordingly, it will be at the learning level.

✓ Autonomy

The agent must be able to take initiative and act without intervention from the end user. What strongly distinguishes an agent from other types of software is its objective, which allows it to be autonomous.

✓ Interactivity

The figure below shows the agent and the 4 components of the environment with which an agent can interact: hardware, humans, other agents and finally software which are not agents.



These interactions occur at different levels due to the varying complexity of the components.

✓ Interaction with hardware

This will largely be accomplished through software interaction, as software is often necessary to control physical characteristics. Nevertheless, the requirements imposed by the physical environment must be considered.

- ↗ Interfaces to sensors: These are essential for receiving information from the physical environment.
- ↗ Interfaces to actuators: These are essential for being able to affect the physical environment.

✓ Interaction with the software

Part of the problem is to be able to make agents collaborate with applications that have been designed with the expectation that a human being participates in the interaction.

✓ Interaction with Humans

This can be very complex; ideally, humans should be able to communicate with the agents in the same way they communicate with other humans. However, current technology is not expected to allow for the standardization of such interaction in the near future. Nevertheless, standardization is possible regarding the information that the agents receive from humans. Two main problems arise:

- ↻ Cooperation and information exchange with humans
- ↻ User authentication

✓ Interactions with other agents

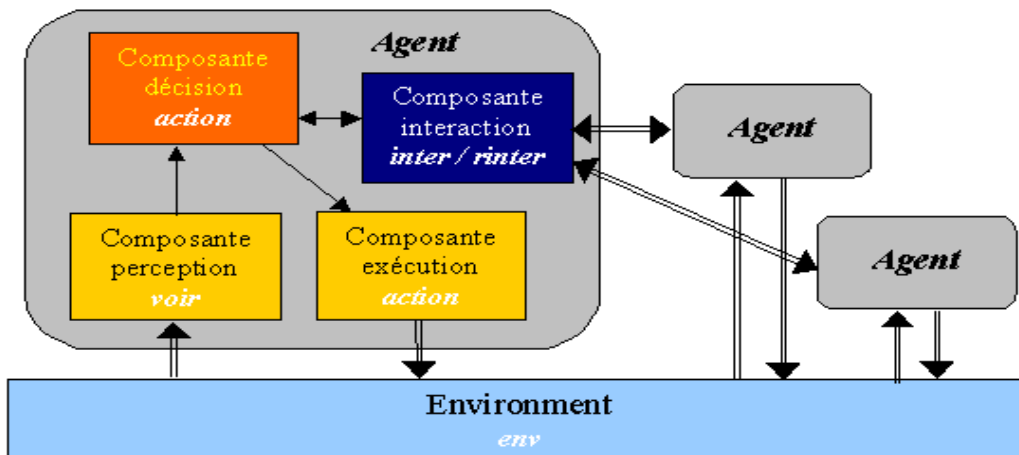
An agent is capable of interacting with other agents, in the same way that different people communicate.

To coordinate their tasks, two agents must negotiate to reach an agreement acceptable to both parties. The advantages provided by interaction between different agents include:

- ↻ the ability to solve problems too complex for the resources of a single agent;
- ↻ enabling cooperation between agents with different roles;
- ↻ providing solutions to naturally distributed problems, such as air traffic management;
- ↻ improving modularity, reliability, and flexibility.

To enable this interaction, a communication language between agents is necessary. Such languages are already under development, but standardization is needed if all agents are to be able to communicate and exchange information.

The air traffic control scenario illustrates the interactions between agents of the same type, to share work and available resources. The example of a user who needs to be contacted to be notified of a security issue illustrates the interaction between agents with different roles to successfully complete a complex task.



General structure of an agent that interacts with other agents

✓ Mobility

Some agents can be fixed, residing either on the user's machine or on the server. Other agents can be mobile, meaning they roam the network. They can move from machine to machine during execution, carrying the accumulated data with them. These agents can present themselves to agencies that can provide them with certain services or serve as a meeting point between different agents.

To illustrate the advantage of mobility, suppose you need to choose an image from a large number. Rather than downloading them all and making the selection locally, wouldn't it be better to send the agent to the machine that hosts the database to make the selection there?

Mobility involves addressing issues of security, privacy, confidential data protection, and management. Until a suitable infrastructure is in place, agents will primarily be fixed. Security personnel will be needed to monitor the activity of agents that install themselves on a machine. Currently, no agent is truly mobile. It's either "pseudo-mobility," meaning the agent uses the network to search for and access data and information, but it doesn't move; the code always runs on the user's machine. At this stage, to make an agent mobile, a specific mechanism must be designed to host the agent on the machines where it is intended to run. In the medium term, we can expect to see a number of network-connected machines dedicate resources (memory, CPU, communication) to mobile agents. For example, Java technology provides a foundation for mobility. Java is an interpreted language, which allows for easier control of its actions. It runs on a virtual machine, making it independent of the physical machine it resides on. This language is derived from C, but without pointers, which ensures a certain level of security, as the Java application cannot access restricted memory resources.

In the long term, it will be essential to design a comprehensive infrastructure to allow agents to move, communicate, and collaborate.

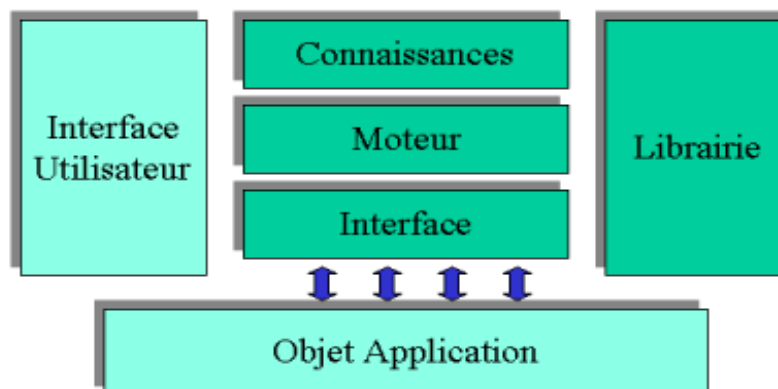
A personal assistant that sorts mail is an example of a stationary agent. An information retrieval agent would typically be a mobile agent.

✓ Ability to reason, to react to their environment

The agent must be able to adapt to its environment (which may consist of other agents, the web in general, or users) and to changes in that environment. This adaptation must be based on an analysis of the agents' external environment.

In the case of Firefly, for example, an agent's knowledge is partly dependent on the knowledge of other Firefly agents. As their knowledge grows, the knowledge of a particular agent also grows through exchanges between agents and each agent's ability to integrate the increased knowledge generated by the Firefly agent system that constitutes its environment.

3. Intelligent Agent Components



General structure of an agent

- The engine, or the "agent's brain."
- Knowledge: what the agent knows, believes, and thinks. Knowledge can be based on existing applications, such as a database or an email agent. Knowledge is stored in libraries.
- The interface with other applications, such as email or the web. This interface determines the scope of an agent's capabilities. L'interface avec l'utilisateur permet la communication between the agent and the user. The user gives instructions to the agent, which then returns the results or provides information. This interface can be a standard graphical interface, but the future will favor natural spoken language, as this allows the user to interact easily with the agent without having to learn a complex interface for each agent. This interface will therefore need to include a speech recognition, text-to-speech, and natural language understanding unit.

4. Types of Intelligent Agents

4.1. According to their nature

- **Reactive Agents:** These agents lack an explicit representation of their environment, a memory of their past, and an explicit goal. Their behavior is stimulus-response.
- **Cognitive Agents:** These agents possess an explicit representation of their environment. They rely on their knowledge of their environment and their ability to reason about that knowledge. Agents driven by goals or a utility function belong to this group.
- **Hybrid Agents:** Typically, an agent is neither entirely reactive nor purely cognitive. Rather, it is a combination of these two approaches.

In some circumstances, an agent must act very quickly, while in others, it may have the opportunity to take more time for a more thorough cognitive process. In this case, the agents are called hybrid.

4.2. Based on use

- **Collaborative agents:** These agents have cooperative capabilities. Grouping these agents allows, among other things, the reduction of a complex problem into less complex sub-problems.
- **Interface agents:** These agents collaborate with the user to perform certain tasks.
- **Information retrieval agents:** These agents first search for information within a data collection and then analyze the useful information found to discover new knowledge.
- **E-commerce agents:** The rise of the Internet has, of course, created new needs. Agents resulting from this trend enable the promotion, sale, and purchase of products and services through computer networks, etc.

4.3. Based on the technology used

- **Stationary agents:** This refers to the case where the agent always runs on the same machine.
- **Mobile agents:** These agents run on different machines, moving from one host to another. Typically, they follow what is called a "route".

5. Intelligent agent architectures

An agent's architecture can be described as a description of its internal organization: the agent's data and knowledge, the operations that can be performed on its components, and the control flow for those operations. The choice of one architecture over another is, of course, linked to the agent's conceptual structure and represents the designer's decision on how to build the artificial agent. The various types of agents are examples of agent architectures, albeit at a very general level. We will examine more detailed examples of agent architectures below and systematically study their operation

5.1 Architecture BDI

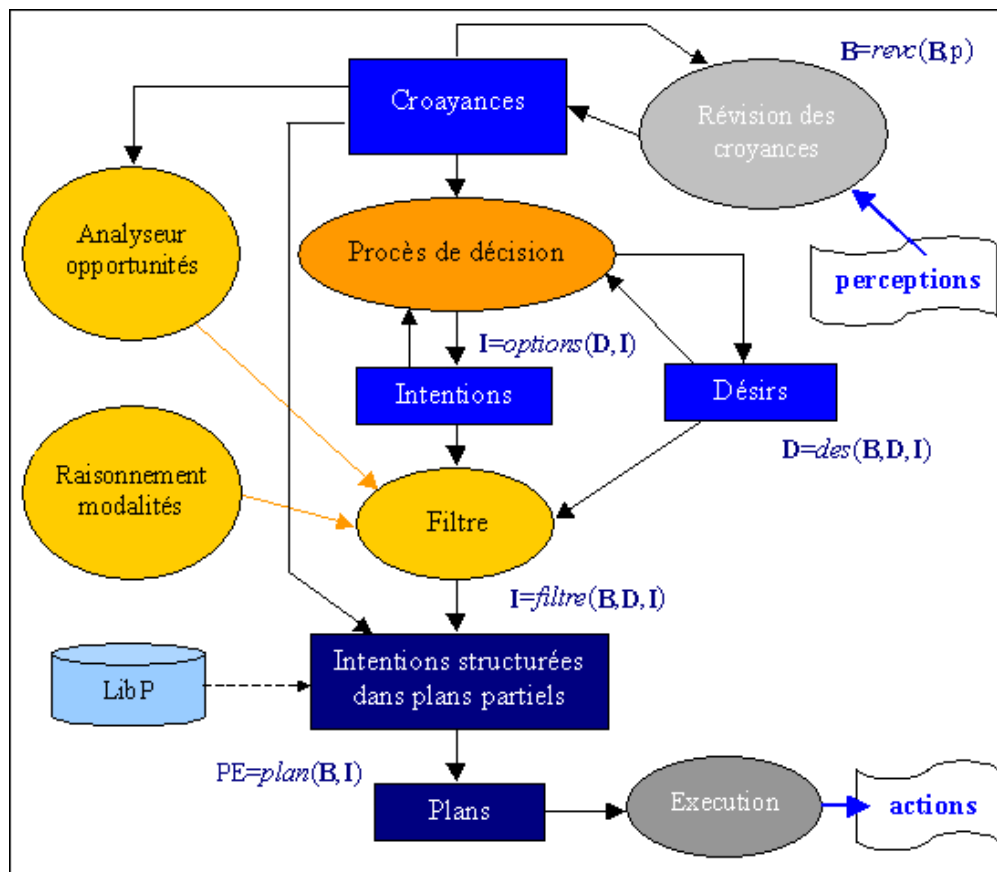
BDI architecture is designed based on the Belief-Desire-Intention model.

- **B = Belief:** An agent's beliefs are the information they possess about the environment and other agents within that environment. Beliefs can be incorrect, incomplete, or uncertain, and therefore differ from the agent's knowledge, which is always true. Beliefs can change as the agent gathers more information through their perception or interaction with other agents.
- **D = Desire:** An agent's desires represent the states of the environment, and sometimes of themselves, that they would like to see realized. An agent can have conflicting desires; in this case, they must choose a consistent subset of their desires. This consistent subset of his desires is sometimes identified with the agent's goals.
- **I = Intention = Intention:** An agent's intentions are the desires that the agent has decided to fulfill or the actions they have decided to take to fulfill their desires. Even if all of an agent's desires are consistent, the agent may not be able to fulfill all of their desires at once.

Exemple : The following example will provide more clarification to this model.

Agent X believes that if someone dedicates their time to studying, they can complete a doctoral thesis. Furthermore, X desires to travel extensively, pursue a doctorate, and obtain a teaching assistant position at a university. The desire to travel is inconsistent with the other two, and after reflection, X decides to choose the latter two from among these seemingly inconsistent desires. Realizing he cannot fulfill both simultaneously, he decides to complete his doctorate first. At this point, X intends to write a thesis and, naturally, will use all available resources to achieve this goal. It would be irrational for X, once he has made his decision, to then dedicate his time, energy, and resources to traveling the world. By focusing on these objectives, X has fewer options to consider because he has already decided against visiting travel agencies to find the most suitable

travel package.



BDI architecture of an agent

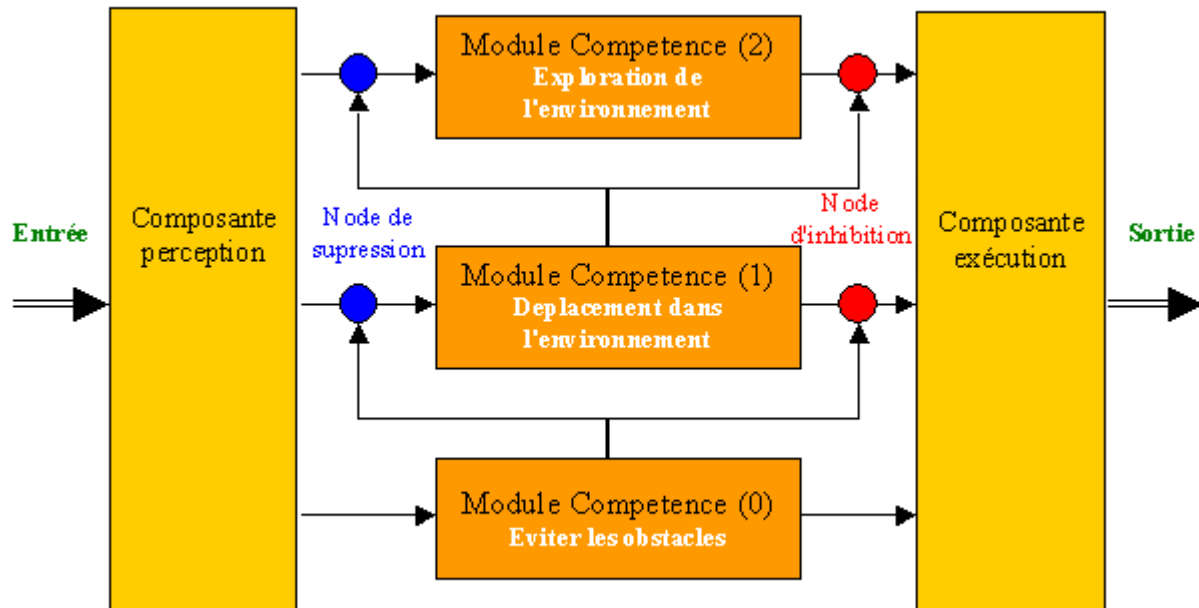
The agent has an explicit representation of their beliefs, desires, and intentions. We denote by B the set of the agent's beliefs, by D the set of their desires, and by I the set of their intentions, and by B , D , and I the agent's current beliefs, desires, and intentions. The sets B , D , and I can be represented using various knowledge representation models, for example, using first-order predicate logic, higher-order logic, the production rule model, or simply as data structures.

5.2. Reactive architectures

Reactive architectures represent the agent's operation using components with a simple control structure and without an advanced representation of the agent's knowledge. The agent's intelligence is seen as the result of interactions between these components and the environment. This means that such an architecture can solve complex problems, which normally require intelligent behavior, without treating intelligence from the classical perspective of artificial intelligence. Intelligence is said to emerge from the interaction between simple components, and between reactive agents and the environment. This approach, which differs significantly from the design of intelligent agents, emerged as a solution to criticisms of symbolic approaches, particularly the computational complexity they

require, which seems incompatible with the limited resources of agents, and the difficulty of always finding the right cognitive model for certain applications.

The best-known and most influential reactive architecture is that proposed by Rodney Brooks; it is called subsumption architecture. A subsumption architecture comprises several modules, each responsible for performing a simple task. These modules correspond to specific behaviors for accomplishing a particular task and are called competence modules



Reactive architectures

5.3. Hybrid architecture

A hybrid intelligent agent architecture is composed of a set of modules organized hierarchically. Each module is either a cognitive component, representing knowledge and reasoning capabilities symbolically, or a reactive component. This approach combines the agent's proactive, goal-driven behavior with reactive behavior in response to environmental changes. Furthermore, it aims to simultaneously achieve the advantages of both cognitive and reactive architectures while eliminating their limitations.

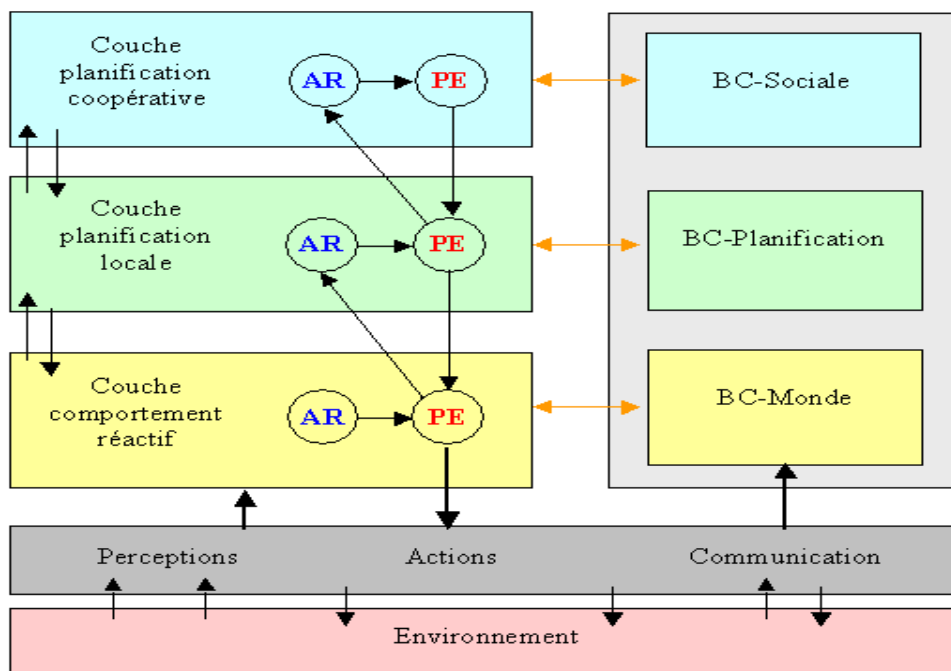
Several hybrid architectures have been designed and implemented. One of the best-known is the InteRRaP system ("Integration of Reactive Behavior and Rational Planning").

An InteRRaP agent is a BDI agent that has goals to achieve (these goals are the same as desires) and is capable of cooperating with other InteRRaP agents to accomplish these goals.

An agent's goals are divided into three categories:

- ✓ Reactive goals: these are simple goals to achieve based on perceptions of the environment;
- ✓ Local goals: these are goals that the agent can achieve independently;
- ✓ Cooperative goals: these are goals that can only be achieved through cooperation with other agents in the system.

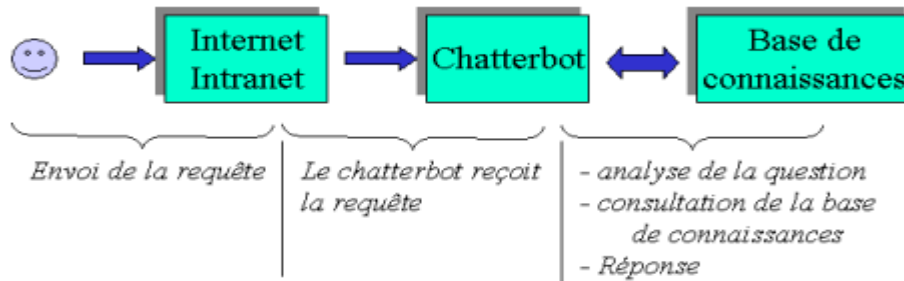
It is interesting to note that, in this model, the agent's reactivity is always considered a goal, that is, at the cognitive level, but a very simple one to achieve.



Architecture InteRRaP (hybride)

- ✓ Competitive intelligence agents: As part of its competitive intelligence strategy, a company can use an intelligent agent to monitor and analyze the strategies and tactics developed by its competitors. The agent will scrutinize their resources, development methods, actions, and results.
- ✓ Conversational agents: A conversational agent (i.e., a chatbot) is an intelligent agent with a dialogue system that allows it to establish verbal communication with its user, offering a more user-friendly and natural interaction. Dialogue refers to the use of various speech acts such as:

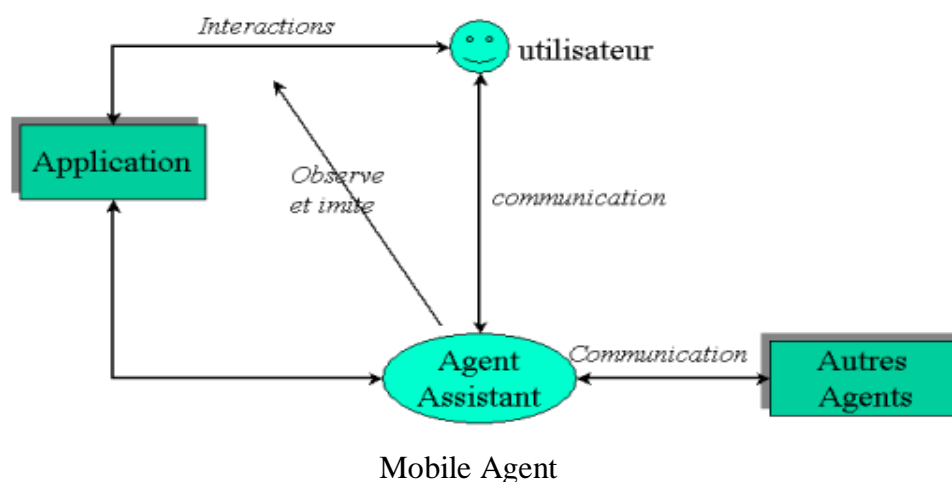
- a series of questions and answers
- expressing an opinion or comment
- stating facts
- being declarative, assertive, or expressive



Information retrieval agents: An information agent is an agent that has access to one or more information sources and is capable of collecting and manipulating this information to respond to requests from users or other agents.

Example: If a user has heard that a researcher is working on a particular subject at a specific university, they ask the agent to perform a search, and the agent returns a technical report along with the contact information for the researcher in question.

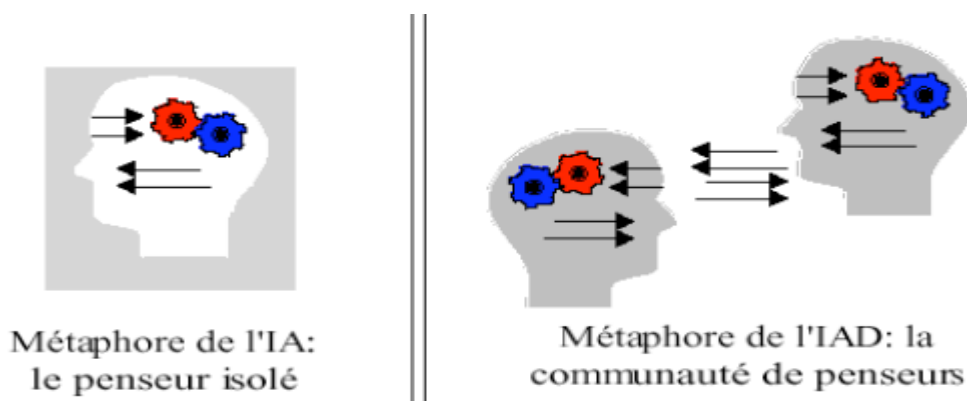
Interface assistant agents: The Intelligent Personal Assistant (IPA) is an agent designed to help us with our daily work, much like a virtual assistant. It helps us perform tasks and can also perform them for us. Thanks to this assistant, the computer becomes an intelligent, active, and, above all, personalized collaborator.



Data exchange and task distribution in a distributed application require interaction between different entities across the network. Currently, the client/server model, where exchanges occur by sending messages across the network, is the most widely used. This model has the drawback of increasing network traffic and requires a permanent connection, which is not the case for mobile devices. A new approach uses mobile agents. These agents are entities that move from one machine to another on the network without losing their code or state. Thus, by sending agents to where the tasks are performed, the exchanged messages become local, thereby reducing the network load. Although the size of the agent adds overhead to communication via mobile agents, this is offset by the fact that the agent performs local interactions after its movement, thereby reducing the amount of information exchanged. Mobile agent communication functions well when it is not permanently established. The client connects to create a mobile agent. This agent then acts as the client and transmits results back to the client upon subsequent connections. Mobility increases the reliability and smooth operation of distributed systems. If a machine fails, the agent on that machine can switch to another to complete the current task.

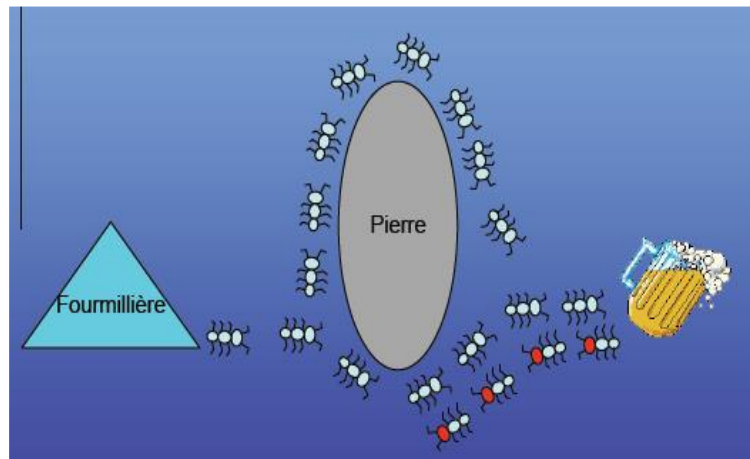
6. Multi-agent systems

6.1 Definition: A multi-agent system is considered a distributed system composed of a number of autonomous entities (agents) that work according to complex modes of interaction to achieve their own goals and thereby reach the desired overall objective. This differs from AI systems, which simulate human reasoning capabilities to some extent. Agents can interact by communicating directly with each other, through another agent, or by acting on their environment.



6.2. Origin: One of the major sources of inspiration for multi-agent systems was the study of the social behaviors of certain insect families (e.g., ants). Instead of considering only the cognitive abilities of an individual, Intelligent Design (ID) posits that intelligence "emerges" from the interactions that agents maintain with each other and with the environment.

The concept of collective intelligence originated in the 1960s, from the application to robotics of ethologists' analyses of insect society behavior.

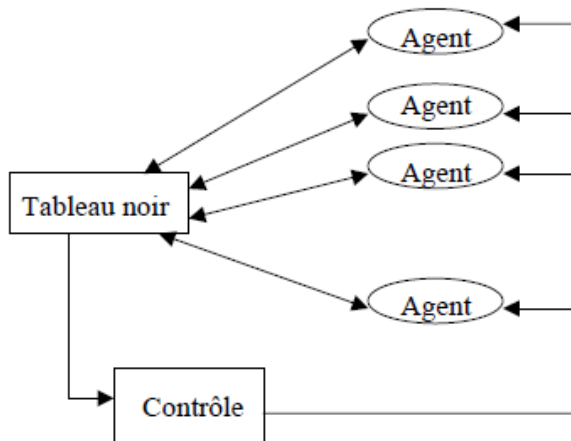


6.3. Interactions : One of the main properties of an agent in a multi-agent system (MAS) is its ability to interact with other agents. These interactions are generally defined as any action performed within the system of agents that results in a change in the behavior of another agent. They allow agents to participate in achieving an overall goal. This participation enables the system to evolve toward one of its objectives and to exhibit intelligent behavior regardless of the complexity of the agents that compose it.

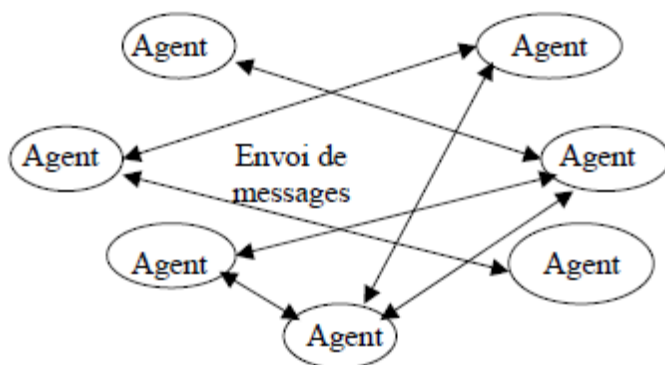
In general, interactions are implemented through the transfer of information between agents or between the environment and the agents, either through perception or communication. Perception is one example.

Agents become aware of a change in a third party's behavior through their environment. Through communication, an agent deliberately transfers information to one or more other agents.

6.4. Agent Communication: Agents communicate with each other using communication protocols. Their communication can be of two types: information sharing or message sending. Information Sharing Communication: In information sharing mode, all system information is centralized in a global data structure. Agents read from and write to this database to update the system, which initially contains the problem data. This type of communication, called "blackboard," is a centralized communication mode that means agents make a large number of requests to a single central location. Furthermore, the overall system operation is not purely multi-agent, since the behavior of one agent depends on the content of a shared knowledge.



Message-based communication: Multi-agent systems based on message-based communication are characterized by the fact that each agent possesses its own local representation of its environment. Each agent then queries other agents about this environment or sends them information about its own perception of it. Communication occurs either point-to-point or broadcast. This type of communication enables the creation of a true multi-agent system since each agent has its own knowledge base. However, ensuring global system convergence is more difficult. Furthermore, in a distributed environment, communication between a large number of agents can quickly lead to network saturation due to the large volume of messages.



6.5. Agent Cooperation: Cooperation between different agents is a difficult problem. Agents have varying degrees of knowledge about the other agents in the system. They must be able to understand the other agents' skills, as well as the tasks they are currently performing.

Methods for distributing tasks among the different agents are a major problem that has been the subject of much research, particularly in the field of distributed computing. Each problem must be broken down and distributed among the different agents, which produce partial solutions that must then be combined to obtain the overall solution.

Task allocation models can be either centralized, where one agent is responsible for decomposing the problem and distributing it, or distributed, where each agent is able to decompose the problem and distribute the resulting tasks. The distributed allocation model allows for better use of workstation resources, but the overall consistency of the system is more difficult to maintain.

Collaboration between agents can sometimes be conflictual. In this case, agents may have identical goals but divergent or even completely opposing views. Conflicts between different agents can be resolved in two different ways:

- ✓ Through centralized control, which ultimately makes the decision.
- ✓ Through negotiation between the various conflicting parties.

While negotiation may seem more promising for generating innovative solutions, this approach presents significant challenges, particularly due to the linguistic limitations of the participants and the excessively long negotiation process when conflicts of interest are significant.

Inter-agent communication languages:

- ✓ The advantage of agent interaction languages is that they reduce communication by avoiding exhaustive descriptions of ad hoc messages and an unnecessarily extensive range of protocols. These languages focus primarily on how to comprehensively describe communication acts from a syntactic and semantic perspective, supporting a knowledge representation language.
- ✓ KQML (Knowledge and Query Manipulation Language) is a high-level communication language and protocol for information exchange. It is message-oriented and independent of the content syntax and applicable ontology.

- ✓ Furthermore, KQML is independent of the transport mechanism (TCP/IP, SMTP, IIOP, or other), the language of the exchanged content (KIF, SQL, Prolog, or other), and the ontology used.
- ✓ FIPA ACL can be considered an extension of KQML with two distinct languages. The external language defines the intended meaning of the message. The internal language (or content) describes the expression to which the beliefs, desires, and intentions of the agents, as described in the communication primitive, are applied.
- ✓ Unlike KQML, FIPA ACL is based on the logical semantics of communication. This facilitates the description of communication formats. However, agents do not always possess the necessary logical capabilities. It is worth noting that the semantics of FIPA ACL are largely based on the beliefs of agents, which may be unknown to other agents.
- ✓ Another major difference between FIPA ACL and KQML concerns communicative acts. FIPA ACL contains a set of normative communicative acts, which can be primitive or compound. New communicative acts can only be defined by combining existing acts and using predefined operators. This helps maintain the semantic integrity of the language..

SMA development environment:

- ✓ PERL: PERL is an interpreted language optimized for browsing text files, extracting information, and printing reports based on that information. It's a practical language similar to C. PERL can also handle binary files. This language is typically used for search tools.
- ✓ JAVA: Java was designed to be mobile, as we mentioned earlier.
- ✓ LISP: LISP is the main language used in artificial intelligence. It's a very powerful and flexible functional language. LISP will be useful for implementing the agent's deduction mechanisms. Its many versions have been combined into a common language, Common LISP, which allows programs to be exchanged between computers without modification.

- ✓ C/C++: These two languages remain effective for implementing certain agent functionalities.
- ✓ Sensor: A sensor is a small, self-contained device capable of performing simple measurements on its immediate environment. Self-contained versions can communicate wirelessly (via wireless networks) with other nearby sensors (within a few meters), and some even include processing power (a processor) and memory. This allows for the creation of a wireless sensor network (WSN) that collaborates over a fairly large area.

WSN Description:

- ✓ Wireless Sensor Networks (WSNs) are considered a special type of ad hoc network. The nodes in this type of network consist of a large number of micro-sensors capable of autonomously collecting and transmitting environmental data. The position of these nodes is not necessarily predetermined. They are randomly distributed across a geographical area, called the sensing field, which defines the area of interest for the phenomenon being detected. The captured data is routed via multi-hop routing to a node considered a "collection point," called a sink node. This sink node can be connected to the network user via the internet or satellite. The user can then send requests to other nodes in the network, specifying the type of data required, and collect the environmental data captured through the sink node.
- ✓ Many problems are inherent in managing wireless networks. However, wireless sensor networks present different characteristics from other types of ad hoc networks.

Generally, the network elements are energy independent and cannot be easily recharged as in cellular networks. This significant constraint must be considered at all stages of design and implementation. Particular attention is required when addressing the issue of information routing (the energy expended during communications represents a significant portion of overall energy consumption). Low energy levels can lead to internal faults at the sensor level and can affect various parameters such as the range of transmitted messages, the quality and accuracy of measurements, etc.

- Nodes are generally stationary after a deployment period, except for a few. The harsh environment (in terms of temperature, pressure, humidity, etc.) in which the devices are typically immersed can also contribute to unexpected faults and require automatic and dynamic reconfigurations.
- The entire system can also be subject to time constraints. In the case of mobile entity networks, the system's dynamics are even more pronounced.

These networks are of particular interest for military, environmental, home automation, medical, and, of course, critical infrastructure monitoring applications. These applications often require a high level of security.

However, due to their characteristics (lack of infrastructure, energy constraints, dynamic topology, large number of sensors, limited physical security, reduced node capacity, etc.), securing sensor networks is currently the source of many scientific and technical challenges. Multi-agent systems applied to wireless sensor networks to improve their security

Intelligent components are found in several areas of network administration:

Configuration, security, fault handling, and accounting. This intelligence generally comes from distributed artificial intelligence.

Recently, there has been a growing number of attempts to combine the two research areas of multi-agent systems and wireless sensor networks. The distributed and open nature of sensor networks makes the multi-agent approach particularly well-suited. The fact that this approach provides an external representation of interactions and organization offers several functionalities, such as control by an external observer. Multi-agent systems have several application areas in the context of wireless networks.

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, $E = \{e_1, e_2, \dots, e_n\}$. In addition, the agent's set of possible actions is represented by the set $Ac = \{a_0, a_1, \dots, a_n\}$. 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: R^{AC} (set of execution sequences ending in actions) and R^E (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 (R^{AC}) 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, \dots) 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 $u > 0, u > 0, e_u \in r((e_0, a_0, \dots, a_{u-1}))$

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

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 ($Ag : E \rightarrow Ac$). 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 ($see: E \rightarrow Per$). 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: $Per^* \rightarrow Ac$).

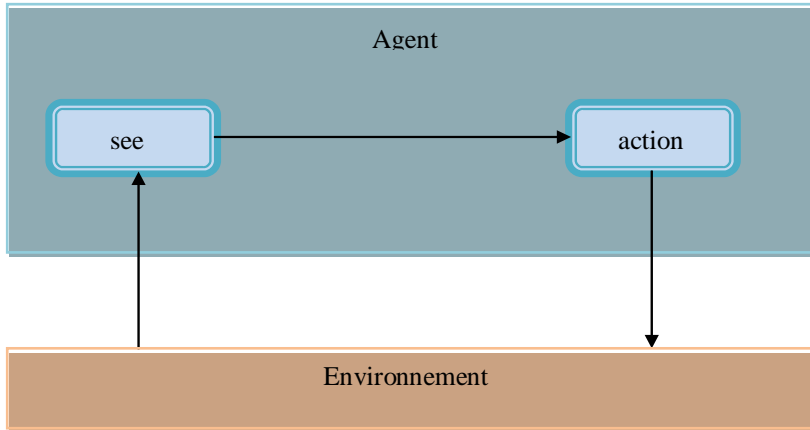


Figure 3.1: description of a purely reactive agent.

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.

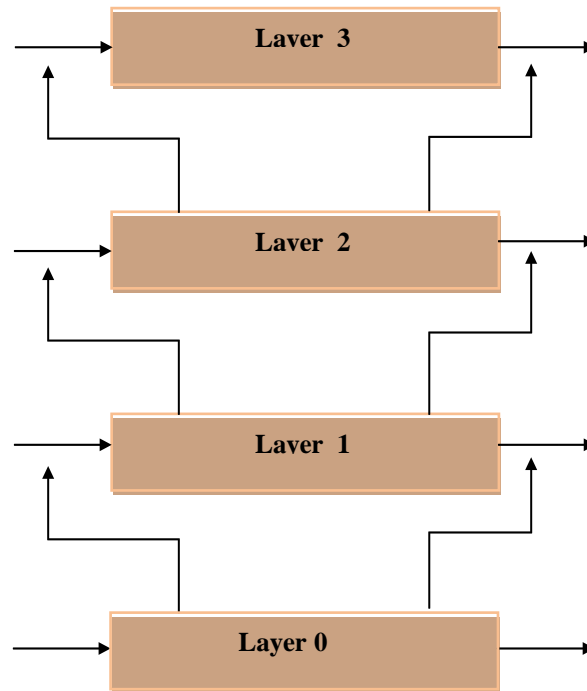


Figure 3.5: Subsumption architecture.

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. Proactive 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.

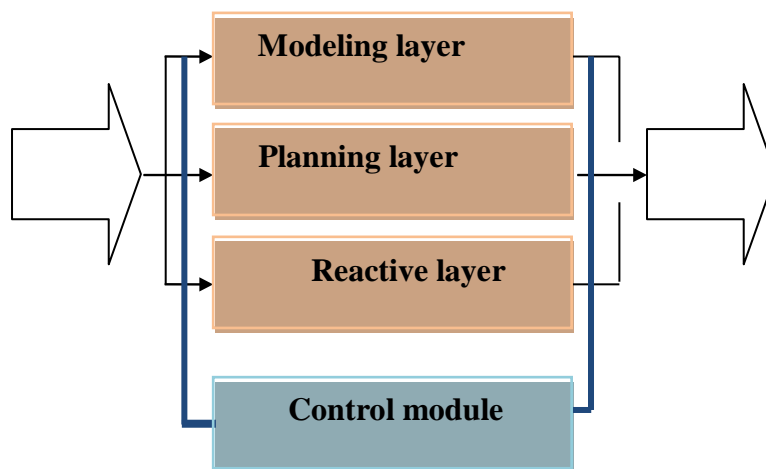


Figure 3.6: The TuringMachine architecture.

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.

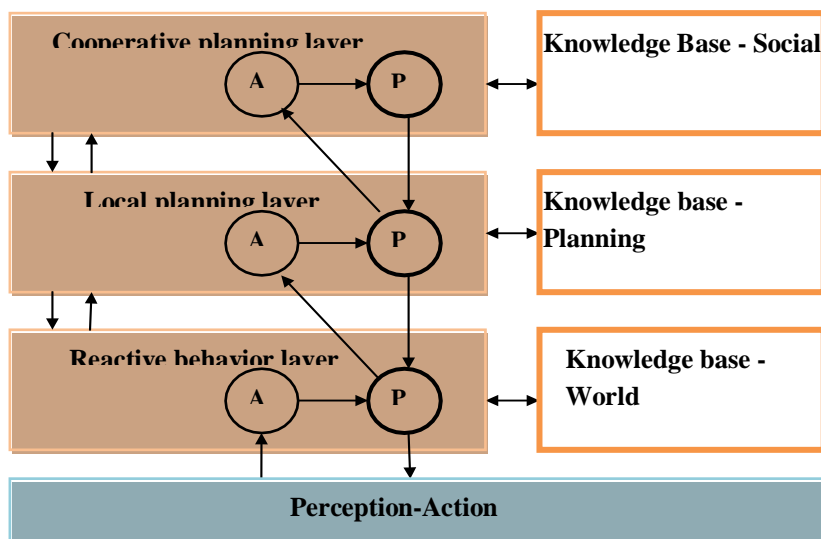


Figure 3.7: The IneRRaP architecture.

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.

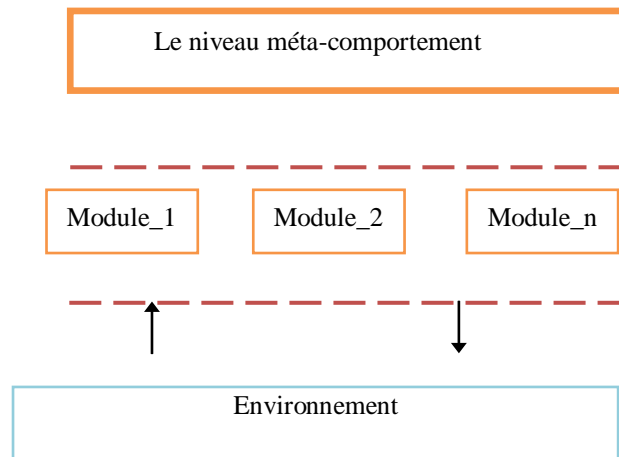


Figure 3.8: The DIMA agent.

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.

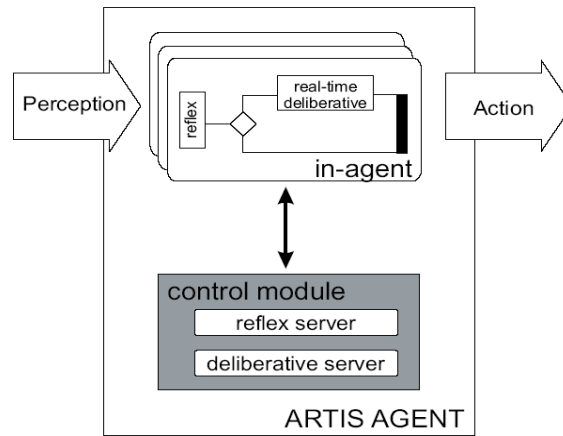


Figure 3.9: ARTIS agent architecture.

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 .

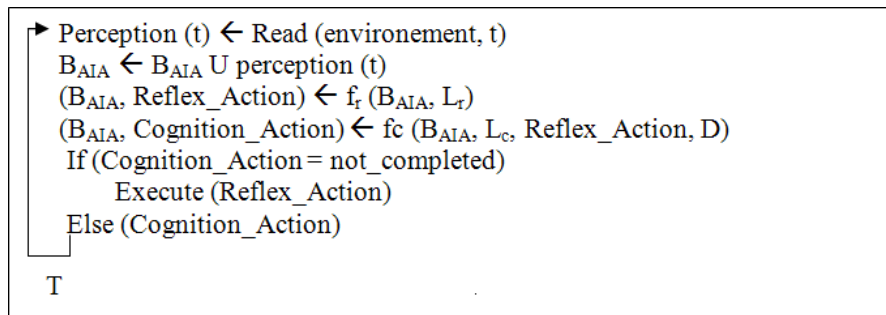


Figure 3.10: How an in-agent works.

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 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. 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 - ϵ). 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 $\langle \mathbf{T}, \mathbf{Ag}, \mathbf{C} \rangle$ such that:

- \mathbf{T} is a set of tasks ($\mathbf{T} = \{\mathbf{T}_1, \mathbf{T}_2, \dots, \mathbf{T}_m\}$);
- \mathbf{Ag} is the set of agents participating in the interaction situation ($\mathbf{Ag} = \{\mathbf{1}, \dots, \mathbf{n}\}$);
- \mathbf{C} : is a cost function that defines a positive real value for each subset of tasks ($\mathbf{C} : 2^{\mathbf{T}} \rightarrow \mathbf{R}^+$). This function must satisfy two conditions:
 - The cost of not performing a task is always zero ($\mathbf{C}(\emptyset)=0$);
 - 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 $R = \{E_1, E_2, \dots, E_n\}$ such that E_i is the set of tasks allocated to agent 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 $R = \{E_1 = \{T_1, T_3\}, E_2 = \{T_2, T_4, T_5\}\}$ such that T_i represents the accompaniment of child C_i . 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 $A = \{D_1, D_2\}$, with the proviso that this redistribution must not omit any task. Consequently, $D_1 \cup D_2 = E_1 \cup E_2$.

Since a case or a meeting is a subset of a task, we can apply the cost function to it. We write $c_i(D_j)$ to designate the cost of case D_j 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: $utility_i(A) = c_i(E_i) - c_i(D_i)$. 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 A_1 is said to **dominate** another case A_2 **if and only if**:

- For both agents, the benefit of adopting case A_1 is at least equal to the benefit of adopting case A_2 .

$$i \in \{1, 2\}, \text{utilit  }_i(A_1) \geq \text{utilit  }_i(A_2)$$

- For at least one agent, the benefit of adopting case A_1 is strictly greater than the benefit of adopting case A_2 .

$$\exists i \in \{1, 2\}, \text{utilit  }_i(A_1) > \text{utilit  }_i(A_2)$$

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 (A_1 by agent 1 and A_2 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.

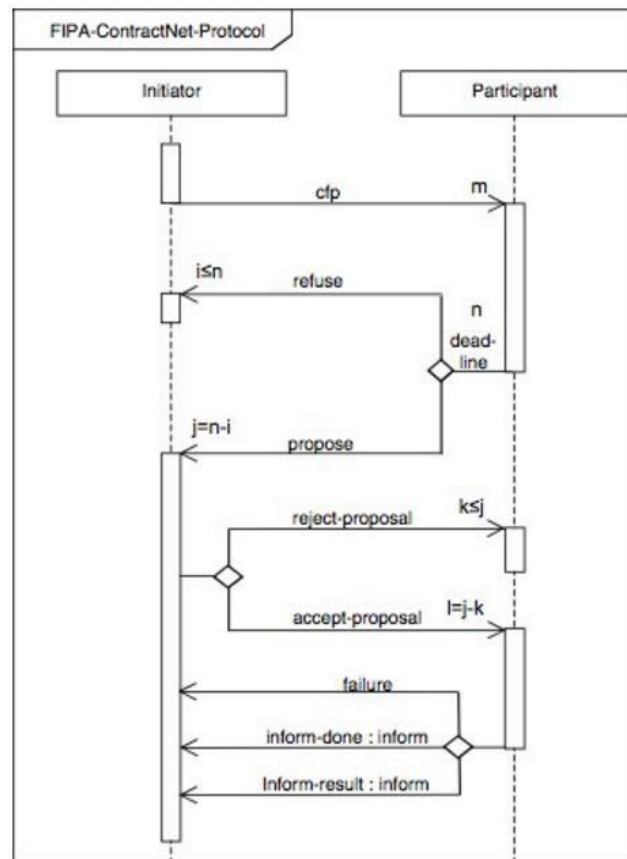


Figure 4.1: The FIPA-Contract-Net protocol.

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*,