

People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Abbes Laghrour University of Khenchela



Faculty of Sciences and Technology
Department of Computer Science ICOSI Laboratory

N° Order :

Serie :

PhD Thesis

An Organizational Multi-Agent Approach for Designing Ambient Intelligence Systems

Presented by :
SAFIR Samir

For the degree of : **Doctor of Philosophy (PhD)**
In : **Computer Science**
Specialization : **Software Engineering and Distributed
Systems**

Under the supervision of :
Prof : SIAM Abderrahim

Defense Date :

Jury Members :

Chair : MAAROUK Toufik Messaoud, Pr, University Abbes Laghrour, Khenchela
Supervisor : SIAM Abderrahim, Pr, University Abbes Laghrour, Khenchela
Co-supervisor SOUIDI Mohamed Elhabib, MCA, University Abbes Laghrour, Khenchela
Examiner 1 : BENMERZOUG Djamel, Pr, University of Constantine 2
Examiner 2 : BENABOUD Rouh Allah, MCA, University Larbi Ben Mhidi, Oum El-Bouaghi
Examiner 3 : LADMI Abdeldjalil, MCA, University Abbes Laghrour, Khenchela

الشكر والتقدير

في مستهل هذه الأطروحة، أتقدم بخالص آيات الشكر والتقدير إلى المشرف الأكاديمي البروفيسور عبد الرحيم صيام الذي رافقني بعلمه الغزير وخبرته القيمة وتوجيهاته السديدة ، طوال مساري لبحري، لقد كان لدعمه العلمي والمعنوي الأثر البالغ في إنجاز هذا العمل

كما أتوجه بجزيل الشكر والامتنان إلى المشرف المساعد، الدكتور سويدي محمد، على متابعته الدقيقة، وملاحظاته البناءة، ومساهماته القيمة التي أثرت مضمون هذه الأطروحة.

ولا يفوتني أن أعبر عن عميق امتناني إلى البروفيسور معروق توفيق، مدير مخبر ICOSI ، على دعمه الكريم وتوفيره لبيئة علمية محفزة ومناسبة لإنجاز هذا البحث.

كما أشكر جامعة عباس لغرور خنشلة، التي شرفني بمنحي فرصة ثمينة للانتماء إلى هذه المؤسسة العلمية المرموقة، وأتاح لي الإمكانيات المناسبة لمواصلة دراستي العليا في ظروف ملائمة.

أتقدم كذلك بجزيل الشكر إلى أعضاء لجنة المناقشة، الذين شرفوني بقبولهم تقييم هذه الأطروحة، وأثمن عالياً وقتهم وجهدهم المخصصين لقراءتها ومناقشتها.

وفي الختام، أتوجه بأسمى معاني الامتنان والعرفان إلى زوجتي العزيزة، التي كانت السند الحقيقي في مسيرتي، بصبرها وتشجيعها المستمر وتضحياتها التي لا تُقَدَّر بثمن، وإلى والديّ الكريمين، لما قدّماه لي من حبّ، ودعاء، وتضحية طوال مشواري العلمي والإنساني.

لكم جميعاً كل الشكر والتقدير، وأسأل الله أن يوفقني في ردّ الجميل، وأن يكون هذا العمل لبنة نافعة في خدمة البحث العلمي والمعرفة.

Abstract

This thesis explores the design and development of adaptive, intelligent environments using Multi-Agent Systems (MAS) within the paradigm of Ambient Intelligence (AmI). Focusing on organization-centered MAS development, we investigate how structured agent interactions, combined with artificial learning techniques, can enhance system robustness, scalability, and responsiveness to dynamic contexts. We leverage the *Moise* organizational modeling language and the *JaCaMo* framework to create a unified platform for developing and managing MAS, integrating agents, their environment, and organizational governance. This approach allows us to model complex social structures, define roles and norms, and specify collective goals, thereby facilitating coordinated agent behavior and adaptation to changing circumstances. The efficacy of our approach is demonstrated through two distinct case studies. The first addresses the timely prediction of *COVID-19* infections by leveraging Internet of Medical Things (IoMT) data and a distributed multi-agent architecture incorporating machine learning algorithms. The second applies our framework to the challenging problem of real-time course timetable scheduling, demonstrating its adaptability to dynamic constraints and resource allocation. Our findings highlight the transformative potential of combining organizational paradigms with artificial learning in MAS, laying the foundation for more intelligent and adaptive AmI systems across diverse domains.

المخلص

تتناول هذه الأطروحة تصميم وتطوير بيئات ذكية وقابلة للتكيف باستخدام أنظمة متعددة الوكلاء (MAS) ضمن نموذج الذكاء المحيطي (AmI). مع التركيز على تطوير MAS القائم على التنظيم، ندرس كيف يمكن لتفاعلات الوكلاء المنظمة، إلى جانب تقنيات التعلم الاصطناعي، أن تعزز من متانة النظام وقابليته للتوسع واستجابته للسياقات الديناميكية. نستفيد من لغة النمذجة التنظيمية *Moise* وإطار العمل *JaCaMo* لإنشاء منصة موحدة لتطوير وإدارة أنظمة الوكلاء المتعددين، من خلال دمج الوكلاء وبيئتهم وحوكمة التنظيم. يسمح لنا هذا النهج بنمذجة الهياكل الاجتماعية المعقدة، وتحديد الأدوار والقواعد، وتحديد الأهداف الجماعية، مما يسهل سلوك الوكلاء المنسق وقدرتهم على التكيف مع الظروف المتغيرة. تم إثبات فعالية هذا النهج من خلال دراستي حالة متميزتين. تتناول الأولى التنبؤ في الوقت المناسب بعدوى *COVID-19* من خلال الاستفادة من بيانات إنترنت الأشياء الطبية (IoMT) وهندسة متعددة الوكلاء موزعة مدعومة بخوارزميات تعلم آلي. أما الثانية، فتطبق إطار العمل الخاص بنا على مشكلة جدولة الجداول الزمنية للدروس في الوقت الفعلي، مبرزة قدرته على التكيف مع القيود الديناميكية وتوزيع الموارد. تسلط نتائجنا الضوء على الإمكانيات التحويلية لدمج النماذج التنظيمية مع التعلم الاصطناعي في أنظمة الوكلاء المتعددين، مما يمهّد الطريق لأنظمة AmI أكثر ذكاءً وتكيفاً في مختلف المجالات.

Contents

List of Figures	iv
Introduction	1
I Introduction and Background	3
1 Multi-Agent Systems in Ambient Intelligence	5
1.1 Introduction	6
1.2 DAI and MAS	6
1.2.1 Introduction to DAI	6
1.2.2 Types of Agents	6
1.2.3 Architecture of Cognitive Agents	7
1.2.4 Agent Society	8
1.2.5 Interaction and Cooperation Among Agents	9
1.2.6 Communication	10
1.2.7 Distribution	13
1.2.8 General Knowledge in AI	13
1.2.9 Formalizing Agent Knowledge	14
1.3 Ambient Intelligence and Multi-Agent Systems	16
1.3.1 Ambient Context	16
1.3.2 Application Scenarios of Ambient Intelligence (AmI)	17
1.3.3 Characterization of an Ambient Multi-Agent System	18
1.3.4 Context Sensitivity	19
1.3.5 Characterization of BDI Agents	20
1.3.6 BDI programming languages	22
1.3.7 BDI Agent Planning	23
1.4 Conclusion	23
2 Enhancing Multi-Agent Organizations with Artificial Learning Paradigms	25
2.1 Overview of Multi-Agent Organizations	25
2.1.1 Definition of Multi-Agent Organization	26
2.1.2 Organization Modeling Language (OML)	28
2.1.3 Organization Management Infrastructure (OMI)	29
2.1.4 Approach to Analyzing Openness in MAS Organizations	31
2.1.5 Non-Normative Organizations	33
2.1.6 Normative Organizations	34
2.2 Artificial Learning	35

2.2.1	Supervised Learning	35
2.2.2	Unsupervised Learning	47
2.2.3	Reinforcement Learning	48
2.3	Conclusion	49

II Methodology and Implementation 51

3 Organizational Modeling and Management Infrastructure: *Moise* & *JaCaMo* 53

3.1	<i>Moise</i> Meta-Model	53
3.1.1	Structural Specification	54
3.1.2	Functional Specification	58
3.1.3	Normative Specification	61
3.1.4	Summary of <i>Moise</i> Meta-Model	63
3.2	Organization Management Infrastructure : <i>JaCaMo</i>	63
3.2.1	Integrated View: The Synergy of Agents, Environments, and Organizations	65
3.2.2	The Agent Dimension	66
3.2.3	The Environment Dimension:	67
3.2.4	The Organization Dimension	69
3.2.5	Summary of <i>JaCaMo</i> OMI	72
3.3	Conclusion	72

4 Predicting Covid-19 Infections 75

4.1	Introduction	75
4.2	Motivation and overview	75
4.3	Related Work	76
4.4	Discussion	77
4.5	The Proposed Solution	77
4.5.1	Assistant Controller AC	78
4.5.2	Broker Agent	81
4.6	Some Implementation Aspects	86
4.6.1	UK Assistant Controller	86
4.6.2	ESI Assistant Controller	89
4.6.3	Biskra Assistant Controller	90
4.6.4	The Broker	91
4.7	Comparison and Discussion	92
4.7.1	Broker Performance Highlights	92
4.7.2	Key Observations	92
4.8	Conclusion and Future Work	93

5 Real-Time Course Timetable Scheduling 97

5.1	Introduction	97
5.2	Related Work	98
5.2.1	Discussion	98
5.3	The Proposed Solution	99

5.3.1	Constraints in Timetabling:	99
5.3.2	The organization Dimension	100
5.4	Implementation	103
5.4.1	System Evaluation	104
5.5	Summary	106
Conclusion		109
Bibliography		111

List of Figures

1.1	Communication through Information Sharing	12
1.2	The Architecture of a BDI Agent	21
1.3	A Functional View of the BDI Interpreter	21
2.1	General Architecture of OMIs	30
2.2	Different learnings of the separation of 2 classes in 2 dimensions, on a noisy sample	36
2.3	Example of a Decision Tree	37
2.4	Crossover Operators for Genetic Algorithms	41
2.5	Model of a Formal Neuron	42
2.6	Example of a Perceptron with Hidden Layer	43
2.7	The backache BN example	45
3.1	General Architecture of OMIs	55
3.2	Graphical Representation of the SS	56
3.3	Graphical Representation of the FS	59
3.4	Lifecycle of a Norm	62
3.5	Dimensions of Multi-Agent Oriented Programming [1].	64
3.6	Dynamic relations among instances of concepts of the MAOP dimensions	65
3.7	Main concepts in the agent dimension	66
3.8	Concepts in the agent dimension	67
3.9	Main concepts in the environment dimension.	68
3.10	Main concepts in the environment dimension.	68
3.11	Main concepts in the organization dimension.	70
3.12	Example of a simple structural specification	71
3.13	Example of a simple functional specification.	71
3.14	Example of a simple functional specification.	71
4.1	Global architecture of the MAS organizational approach for COVID-19 prediction	78
4.2	The global architecture of the MAS organizational approach to Covid-19 prediction.	78
4.3	Assistant Controller architecture	79
4.4	Broker Agent	85
4.5	UK architecture	86
4.6	UK interface	86
4.7	UK interface agent	87
4.8	UK Learner agent	88

4.9	UK Learner Curve	88
4.10	UK Test Covid-19 case	89
4.11	ESI Interface	89
4.12	BISKRA Interface	90
4.13	Broker Interface	91
4.14	Broker use Contract Net Protocol	91
4.15	Broker Accuracy	92
4.16	Broker Reponse Time	93
5.1	<i>Moise</i> structural specification for timetable schedule	101
5.2	<i>Moise</i> functional specification for timetable schedule	101
5.3	the Organization Entity of the timetabling project	102
5.4	XML Organization specification for timetable	103
5.5	the main parts and properties of <i>CARTAgO TimeTable</i> GUI artifact	104
5.6	Source code of the Jason <i>scheduler</i> agent in timetable project	104
5.7	Source code of the Jason <i>teacher</i> agent in timetable project	105
5.8	App Implementation	106
5.9	Scheduling Time.	106

Introduction

In recent years, the convergence of Multi-Agent Systems (MAS) and Ambient Intelligence (AmI) has opened new avenues for building intelligent, responsive, and adaptive environments. This thesis explores the organizational and learning dimensions of MAS within AmI, focusing on how structured agent coordination and artificial intelligence techniques enhance adaptability, scalability, and performance in complex, dynamic domains.

This research is grounded in the JaCaMo framework and the Moise organizational model, offering a unified platform for integrating agent behavior, environmental perception, and organizational governance. By integrating supervised and unsupervised learning techniques, agents can make context-aware decisions and collaborate more efficiently.

Two real-world case studies validate the proposed approach. The first involves the prediction of COVID-19 cases using a distributed agent system integrated with the Internet of Medical Things (IoMT). The second addresses the challenge of real-time academic course scheduling, demonstrating the adaptability of the organizational MAS framework to dynamic constraints and resource optimization.

This work not only underscores the theoretical underpinnings of organizational MAS in ambient systems but also provides concrete tools and architectures that can be adapted to sectors such as healthcare, education, and urban computing.

Part I

Introduction and Background

Multi-Agent Systems in Ambient Intelligence

The rapid evolution of computing has paved the way for new paradigms, transforming how intelligent systems interact with and adapt to their surroundings. Among these paradigms, Multi-Agent Systems **MAS** and Ambient Intelligence **AmI** stand out as promising approaches for creating dynamic, responsive environments. MAS, a field within Distributed Artificial Intelligence **DAI**, centers on the collaboration of autonomous agents, each with distinct goals, capabilities, and decision-making processes. By working within a shared environment, these agents address complex, distributed problems. Such tasks would be challenging or even impossible for a single agent to solve.

Complementing the power of MAS, the Ambient Intelligence envisions environments where technology is seamlessly embedded, context-aware, and responsive to human needs (e.g., adjusting lighting based on occupancy). AmI uses advances in ubiquitous computing, sensor networks, and AI to create spaces that adapt in real time, becoming almost invisible but intuitively supportive to users. This chapter examines how the integration of MAS and AmI creates intelligent environments that not only react to user needs but proactively meet them, fostering collaboration and enhancing safety and convenience across various domains.

The discussion covers the foundational principles of MAS and AmI, highlighting key concepts such as agent architectures, communication protocols, context awareness, and adaptive planning. We also examine real-world applications and address the technical challenges and research directions that continue to shape the development of context-aware, intelligent systems. Through this chapter, we aim to illustrate how the fusion of MAS and AmI creates transformative potential for adaptive, intelligent environments poised to impact areas ranging from smart homes and healthcare to urban management and personalized education.

1.1 Introduction

Distributed Artificial Intelligence (DAI) combines insights from cognitive psychology, biology, sociology, and computer science to design computing systems in which processing, representation, and reasoning are distributed across subsystems known as agents. Collectively, these agents form a society called a Multi-Agent System (MAS). Agents use their available resources to develop strategies for collaborative problem-solving.

In this chapter, we provide an overview of Distributed Artificial Intelligence (DAI) and Multi-Agent Systems (MAS), emphasizing key aspects such as agent architectures, interaction mechanisms, communication strategies, and organizational structures. We also focus on knowledge acquisition, formalization, and representation. Finally, we describe the contributions of MAS approaches to complex systems, such as productive organizations, and vice versa.

1.2 DAI and MAS

1.2.1 Introduction to DAI

Artificial Intelligence (**AI**) is a field of computer science concerned with modeling human intelligence, including perception, decision-making, understanding, and learning. Traditional AI emphasized centralized intelligence, developing computer programs capable of performing complex tasks within a single system. However, this centralized approach faced limitations, especially in integrating diverse expertise and enabling collaboration. In reality, agents must communicate and cooperate to accomplish shared goals.

Distributed Artificial Intelligence (DAI) [2, 3] emerged in the early 1980s to address these limitations and enrich the classical AI approach by distributing expertise across a group of agents operating without centralized control. These agents work and act within a common environment, handling potential conflicts. DAI focuses on modeling intelligent behaviors as outcomes of cooperative activity among multiple agents, which gave rise to the concept of multi-agent systems.

A Multi-Agent System **MAS** is defined as a tuple $\langle O, E, A \rangle$, where O represents a set of objects, A a set of agents, and both O and A are within an environment E [4].

Yves Demazeau [5] emphasizes two additional axes characterizing MAS: Interactions and Organizations. He formulates MAS as follows:

MAS = Agents + Environment + Interactions + Organizations (AEIO)

Following this approach, we present our view of multi-agent systems, using Y. Demazeau's "Vowels" concept and analyzing other essential aspects such as cooperation and conflict resolution.

1.2.2 Types of Agents

Multi-Agent Systems (MAS) are generally categorized into two types: cognitive agents and reactive agents. Cognitive agents, in contrast to reactive agents, resemble societies of experts, each possessing explicit representations of beliefs, intentions, communicative acts, and models of other agents.

Comparison of Cognitive and Reactive Agents

The term “agent” has been defined by various authors [3, 6–8], and the question “what is an agent?” can be as challenging as “what is intelligence?” Therefore, we synthesize several definitions rather than proposing our own.

According to Y. Demazeau and al, an agent is an intelligent entity that acts rationally and intentionally, based on its own goals and knowledge. J. Ferber describes an agent as a real or abstract entity capable of acting on itself and its environment, holding a partial representation of this environment. Within an MAS, it can communicate with other agents, with behavior shaped by its observations, knowledge, and interactions.

MAS distinguishes between “cognitive agents” and “reactive agents.” Cognitive agents possess a knowledge base comprising expertise in their domain and tools to manage interactions with other agents and the environment. Generally, these agents are “intentional,” meaning they have explicit goals and plans enabling them to achieve those goals. In this framework, cooperation resembles human group dynamics, requiring coordination and sometimes negotiation to resolve conflicts.

In contrast, reactive agents do not individually exhibit intelligence. They merely respond to basic stimuli from their environment, and their behavior is determined by interactions with their surroundings, without representing either other agents or the environment. Nonetheless, reactive agents can collectively solve complex problems. Research on reactive agents emphasizes modeling societies of agents rather than individuals, drawing analogies to artificial life and ethology, such as ant colonies, termite mounds, and beehives.

Cognitive	Reactive
Explicit representation	No explicit representation of the environment
Can account for past experiences	No local memory
Complex agent behavior	Simple stimulus-response behavior
Smaller number of agents	Large number of agents

Table 1.1: Comparison between Reactive and Cognitive Agents

As shown in Table 1.1, cognitive agents have an explicit representation of their environment, allowing them to consider past experiences, whereas reactive agents lack this feature and operate based on a stimulus-response mechanism.

1.2.3 Architecture of Cognitive Agents

Cognitive agents generally possess more elaborate architectures than reactive agents. However, this distinction does not fully capture the diversity of existing architectures, as a wide range has been developed for both types (see Table 1.2). It is even possible to design reactive agents using architectures originally intended for cognitive agents, and vice versa [3].

Most of these architectures can apply to both cognitive and reactive agents, but it’s important to recognize that some architectures—such as blackboard systems—are reserved for cognitive agents, whereas subsumption architectures are used for reactive agents.

Architecture Type	Approach	Component Type	Hierarchical Structure
Horizontal Modular	Function	Module	Hierarchical
Blackboard	Function	Task	Hierarchical
Subsumption	Vert. Function	Primitive Task	Hierarchical
Competitive Tasks	Vert. Function	Task + Primitive Actions	Competitive Hierarchical
Production Rules	Function	Rule	Hierarchical
Classifiers	Vert. Function	Rule	Hierarchical
Connectionist	Vert. Function	Formal Neuron	Egalitarian
Dynamic System	Vert. Function	Stimulus-Response Relation	Egalitarian
Multi-Agent	Object/Function	Agent	Egalitarian

Table 1.2: Main Agent Architectures [3]

The modular architecture is one of the most widely used in MAS. It consists of interconnected modules, each performing a specific horizontal function. Common modules include:

- Perceptual and motor functions, where applicable
- Emission and interpretation of communications
- Belief base, modeling the environment and other agents
- Management of commitments
- Domain expertise functions
- Goal management and decision-making
- Action planning

1.2.4 Agent Society

Regardless of whether agents are cognitive, reactive, or hybrid, implementing a multi-agent system raises two fundamental questions:

1. *How should agents communicate, interact, and cooperate?* What types of communication, protocols, or languages should be used? How can synchronizations and coherence of information flow between agents be managed?
2. *How should the system be organized?* How should tasks be distributed among agents to ensure effective global control of problem-solving? No universal theory exists for determining the optimal organizational structure or agent interaction pattern for a given application.

We attempt to answer these questions by presenting different views on the interactional and organizational aspects of agent societies.

1.2.5 Interaction and Cooperation Among Agents

Interaction is a dynamic relationship involving two or more agents through reciprocal actions. For an agent, interaction with others can provide capabilities but also introduce challenges [3]. Cooperation, the most general form of interaction, has been studied extensively in DAI. However, the literature is fragmented, and there is ambiguity regarding the meaning and implementation of cooperation [9]. Should it be defined in terms of goals [10], strategies [11], or structural forms [12]? Moreover, cooperation is often closely linked to related concepts such as coordination, organization, communication, and negotiation.

Research on cooperation covers diverse aspects and does not always align, using a variety of terminologies. Additionally, cooperation represents a point of convergence for many disciplines, making it challenging to extract a general definition. Consequently, we examine cooperation in relation to our specific problem context.

Sociologist K. Schmidt [13] argues that humans cooperate due to their limited capacities. We cooperate because we cannot accomplish a task individually or because individual performance is slower and less efficient. In psychology, two conditions define cooperation: “The subjects must share a common goal, and each subject must view the final product as the sum, combination, or combination of partial contributions of their actions and those of other subjects.”

Most research on cooperation focuses on whether there is an advantage in cooperating with others or on how to communicate within groups. While these studies emphasize collective structures and the affective relationships that lead individuals to cooperate and work together, J. Erceau highlights that the DAI-MAS approach reframes the issue by focusing on the cognitive and behavioral traits necessary for collective work. Durfee identifies four general goals of cooperation:

- Increasing task resolution speed through parallelization
- Extending the range or scope of achievable tasks through resource sharing
- Improving the likelihood of task completion through redundancy, using different methods if possible
- Reducing interference between tasks by avoiding negative interactions

J. R. Galliers [14] examines the nature and role of what she calls “propositional postures,” including conflict, cooperation, and indifference. These postures connote goals by characterizing social relationships among agents and their propositions. “Cooperation exists between two agents regarding a proposition when one agent recognizes that the other aims to realize this proposition and has committed to achieving it as a shared goal, in relation to other agents who also hold this goal and as a deliberate choice.” Cooperation thus requires three conditions: recognizing the goal of another agent, engaging in resolving it as a shared goal, and adopting it as a common objective.

Models of Cooperation

According to A. Vailly and M.A. Simon [15], agents can cooperate in the following ways:

- **Task and Result Sharing:** Task-sharing requires dividing the main problem into sub-problems that can be addressed independently with minimal inter-agent communication. This type of cooperation is well-suited to domains with task hierarchies or levels of abstraction. If sub-problems cannot be handled independently, result-sharing is necessary. Result-sharing does not require decomposition mechanisms and is especially useful in domains where one agent's results significantly impact others.
- **Command:** A supervising agent decomposes the problem into sub-problems and distributes them among other agents, who then solve these tasks and return partial solutions to the supervising agent.
- **Bidding:** A supervising agent breaks down the problem into sub-problems and broadcasts the list. Interested agents submit bids, from which the supervising agent selects and assigns tasks. The system then operates in command mode.
- **Competition:** In this cooperation mode, the supervising agent decomposes and distributes sub-problems, as in bidding. Each agent solves one or more sub-problems and sends results to the supervising agent, who then consolidates them.

Cooperation in Organizations

Cooperation in organizations takes on a specific meaning. When a decision center makes a decision, it influences constraints for other decision centers. Thus, coordination among decision centers can be viewed as a constraint negotiation process [16]. Decision systems require environmental assessment by a decision-maker and exchanges among decision-makers to address their inherent incompleteness. However, local choices can lead to decisions inconsistent with those made by other centers [17], and the interdependence of decisions may propagate this inconsistency across the system.

Once agents are capable of cooperating, the next challenge is establishing effective communication mechanisms.

1.2.6 Communication

Communication involves both physical and cognitive processes through which a sender establishes a connection with one or more receivers, aiming to achieve specific objectives. It is considered a special class of action that, rather than altering the physical environment, influences the mental representations of agents (e.g., their goals or beliefs). As noted by J. Erceau and J. Ferber, communication in a multi-agent context is not merely an input-output exchange; it must be modeled as an act capable of altering the internal states of other agents.

This involves both physical processes (for example, exchanging messages) and cognitive processes (such as changes in agents' goals and beliefs). According to Ferber, communication is a method of cooperation (interaction) alongside:

- **Collaboration:** Concerns how tasks and resources are allocated among multiple agents
- **Coordination of Actions:** Focuses on organizing actions among agents in time and space to achieve objectives
- **Conflict Resolution:** Entails arbitration and negotiation, such as through compromise

Available communication theories [17], as Bouron suggests [9], aim to explain and justify human communication. However, the complexity and high abstraction level of these theories prevent them from directly applying to computer-based agents. Communication facilities in most multi-agent systems, therefore, lack a theoretical foundation and are often developed based on the application domain.

Why Communicate?

Agents communicate to synchronize actions, resolve conflicts (such as those involving resources, goals, or interests), support each other, and, as J. Ferber notes, compensate for perceptual limitations. An agent cannot interact with all other agents, nor can it be equipped with all sensors necessary to fully understand the environment.

When and With Whom to Communicate?

Determining when agents should communicate requires identifying situations where communication is necessary. Generally, agents communicate when facing an unsolvable problem (due to skill or resource limitations), when action coordination is required, or when conflicts arise that cannot be deterministically resolved. Communications can be broadcast to all agents or targeted to specific agents likely to be interested.

How to Communicate?

Communication methods for transmitting messages (containing information or actions) among agents include message-passing and information-sharing.

Information-Sharing Communication

Historically, the first communication model, introduced in the early 1960s [18], was information-sharing. The “blackboard” model [19], a shared memory, was conceptualized as a board where agents can write, locate partial answers, and find information. The blackboard is divided into levels corresponding to different abstraction levels, and a control mechanism manages access conflicts, with agents autonomously requesting access.

The communication process using the blackboard model (see Figure 1.1) allows agents to access shared information at various abstraction levels.

The blackboard mechanism [20] consists of three main elements:

- **Knowledge Sources (KS):** The domain knowledge necessary for problem-solving is distributed across several modules called knowledge sources.

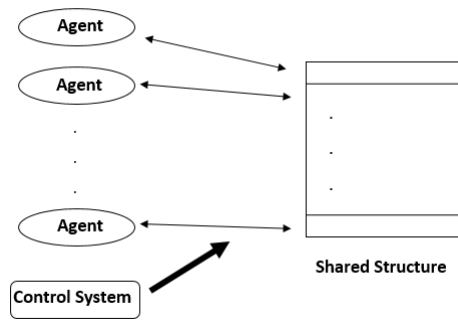


Figure 1.1: Communication through Information Sharing

- **Blackboard:** The blackboard contains a description of the solution state in entities called facts, hypotheses, or nodes. It is structured into levels, allowing for a decomposed description of the solution across abstraction levels.
- **Control :** The control mechanism selects which knowledge source to activate based on its condition part. The selected action part is then executed, creating new hypotheses on the blackboard and allowing the solution to progress. Activation of KS modules is determined by events generated through the creation or updating of hypotheses on the blackboard, processed sequentially by the control mechanism.

Message-Passing Communication

When agents communicate by sending messages, they align with the “actor” model proposed by Hewitt [21]. An actor is an active and autonomous entity with a limited view of the universe. Actors are defined by two elements: acquaintances (other actors known to them) and behaviors (scripts) that specify actions in response to messages. Actors may process a message, delegate tasks to acquaintances, create new actors, or disappear. Message-passing is the only communication mode between actors, with all messages sent asynchronously and placed in a memory buffer (with no response expected). Each message includes a “continuation” indicating where to send the reply. Knowledge and control are distributed, with each actor possessing a script to respond to received messages.

Communication Protocols

To communicate, agents need a protocol that structures and ensures continuity of exchanges from beginning to end. These exchanges depend on agents’ abilities to emit and process information. Each message begins and ends with a specific keyword, conveying orders, information, requests, promises, threats, etc. Protocols are expressed as a series of elementary processing steps (represented by states) implemented locally within agents, intercalating between every reception and transmission of interaction. Protocols rely on psychological, cognitive, and social principles, and often use mechanisms such as speech acts or bidding methods common in public contract awards [22].

1.2.7 Distribution

A key advantage of Multi-Agent Systems lies in their distributed nature, which enables them to manage complex problems by decomposing processing tasks. A distributed architecture enhances modularity, supports system expansion through the addition of new agents, improves reliability through fault isolation, and allows adaptation to environmental changes. It also permits multiple levels of authority among agents, such as control over resources, tasks, information, or knowledge.

Distribution is motivated by several factors:

- Traditional approaches, as demonstrated in industrial expert system development, have shown the challenges of creating knowledge bases due to the large scale, diverse sources, rule inconsistencies, and reasoning across multiple knowledge sources.
- A single (centralized) knowledge base cannot adequately represent conflicting reasoning.
- A knowledge base's search space is so vast that performance can suffer from combinatorial explosion.
- Centralized systems are less resistant to external disruptions.
- In reality, people typically work collaboratively, sharing experiences to achieve common goals, making it unrealistic to represent complex systems as a single expert.

Durfee, Lesser, and Cornill [23] proposed various distribution models: spatial distribution (agent location), functional distribution (agent roles), temporal distribution (availability of expertise at certain times), and logical distribution (degree of logical independence among knowledge parts).

A distributed architecture facilitates modularity and allows for system extension by adding agents. Separation of system components enhances overall reliability, allowing failure management without compromising the global structure.

1.2.8 General Knowledge in AI

Before addressing agent knowledge, let's cover some basics on knowledge in AI (acquisition and representation). In the 1970s, knowledge became a significant focus in AI research. Defining knowledge is challenging. In AI, it is often described as the collection of information—including skills, experience, memory, concepts, and facts—necessary for a human or machine to perform complex reasoning or problem-solving tasks.

The knowledge used in AI is predominantly symbolic and relates to this type of knowledge. When implementing a knowledge-based system (KBS) [24], two complementary aspects are essential: representing knowledge a priori and using that knowledge a posteriori.

Several key questions arise in this context:

- How can an expert's knowledge be acquired?
- In what form should this knowledge be expressed?
- How should this knowledge be represented within a system?

- Which algorithms or reasoning mechanisms can effectively use this limited set of knowledge and extract all implicit information?

Knowledge Acquisition

For a human to address a problem within a specific domain, they first need relevant knowledge of that domain, guiding their reasoning toward a goal. For automated problem-solving, systems must similarly be equipped with the necessary knowledge through a process known as knowledge acquisition. This process transfers knowledge from expert sources (experts, documents, databases, etc.) into the automated system.

Expert knowledge is challenging to extract and model within a Knowledge-Based System (KBS). Expertise results from long practice (experience), and experts may not have conscious access to all their knowledge. Consequently, making this knowledge explicit is not straightforward, making acquisition a delicate task.

Another issue KBS developers face is that experts may omit parts of their knowledge when sharing it. This can happen because experts may not always be in a real-life context when explaining their knowledge and might not break down each reasoning step to provide all relevant information.

To address these challenges, methodologies such as MACAO [25] and KADS [26] were developed to streamline the transfer of human knowledge to computer systems. These methods emphasize separating analysis and design phases to construct a reliable knowledge model.

Knowledge Representation

In an agent or a KBS, knowledge is viewed as a computable unit, an ontological entity within object taxonomies and relational hierarchies. Knowledge can be a fact, raw data, or an action characterizing domain-specific objects or behaviors.

For a KBS, creating, maintaining, reusing knowledge bases, acquiring knowledge, and learning all depend on effective knowledge representation, translating expert knowledge into a symbolic form the system can process.

Representation = Storage + Reasoning [27]

Knowledge representation involves designing data structures to store and manipulate knowledge, followed by implementing programs to effectively leverage the limited set of knowledge.

1.2.9 Formalizing Agent Knowledge

For years, researchers have explored using machines to solve complex problems, an effort achievable only by equipping these machines with human-like knowledge. Developers and users of intelligent systems understand that a system's problem-solving and explanatory capacities rely on its knowledge: concepts, relationships, representations, methods, and heuristics relevant to its domain.

Cognitive agents possess explicit knowledge that enables them to understand their environment, address challenges, make judgments, solve problems, and act autonomously [28]. Their behavior relies on both predefined knowledge and knowledge acquired through learning.

Agent knowledge extends beyond task knowledge to include information about the environment and other agents with whom it communicates and interacts. Interaction knowledge results from agent interactions, following the interactionist hypothesis which posits that an individual's knowledge can itself be seen as a multi-agent system. Here, concepts, ideas, and representations operate as particular types of agents within the individual, with reasoning viewed as the outcome of these inner interactions, akin to Minsky's "Society of Mind" theory [29].

Generally, cognitive agents have three types of knowledge [30]:

1. **Domain Knowledge** – Expertise in a specific field.
2. **Communication Knowledge** – Used for cooperative work and to exchange information or problems in pursuit of common objectives.
3. **Control Knowledge** – Helps agents coordinate actions and decide on tasks to complete a project.

Key Characteristics of a Cognitive Agent

Cognitive agents possess several characteristics, such as intention, rationality, belief, and commitment.

Intention is a crucial element of an agent's intelligent behavior, serving as an explicit declaration of its individual goals. This expresses an agent's conscious decision to achieve a goal or perform an action. Intentions are complex mental states, as Cohen and Levesque describe [31], and must satisfy temporal constraints (past beliefs and future goals) and goal-oriented constraints (breaking down complex goals into sub-goals).

Rationality In AI, Newell's [18] principle of rationality states that "if an agent knows an action will help it achieve its goals, it will select that action." Rational agents can evaluate actions and choose the best ones to reach decisions they can justify.

Beliefs represent an agent's understanding of the world, including itself, its environment, and other agents. Beliefs are subjective and may change over time. Chaib-Draa [32] describes beliefs as meaningful information for the agent.

Commitments play a crucial role in collective actions among cognitive MAS agents. Alan Bond [33] introduced this concept in AI, and Jacques Ferber [3] describes it as the duties and constraints binding cognitive agents to themselves and others when they decide to perform actions or offer services. Without commitments, agents would struggle to form adequate representations of future states, making it challenging to plan and anticipate.

Commitment Types

There are several types of commitments based on the nature of the commitment:

- **Relational Commitments** – When an agent commits to another to accomplish an action or ensure the validity of information.

- **Environmental Commitments** – Commitments related to resources.
- **Group Commitments** – When an agent promises to complete a task for the group or organization or accepts the constraints associated with a role in an organization.
- **Organizational Commitments to Members** – For example, a company's commitment to paying its employees.
- **Self-Commitment** – When an agent commits to a goal for itself.

These commitments represent the social and organizational bonds necessary for effective agent collaboration within a MAS.

1.3 Ambient Intelligence and Multi-Agent Systems

Ambient Intelligence (or AmI) represents the third wave of computing, following the mainframe and the personal computer (PC). Since the 2000s, computing devices have been rendered invisible and integrated into electronic terminals equipped with the ability to communicate with each other, known as connected objects.

Ambient Intelligence (AmI) envisions computerized systems seamlessly embedded in real-world environments. Its goal is to integrate computing so thoroughly into human settings that it becomes invisible. This contrasts with virtual reality, which immerses individuals in entirely computer-generated worlds.

The computerization of AmI, known as ambient systems, lies at the convergence of three domains [34]:

- **Ubiquitous Computing:** the integration of computing systems into everyday connected objects.
- **Ubiquitous Communication:** enabling these objects to communicate not only with each other but also with the user.
- **Intelligent User Interface:** allowing users to control and interact with these objects intuitively.

In recent years, artificial intelligence has been incorporated into these ubiquitous systems to make them smarter, enabling them to analyze the real environment and user activity, thus allowing ambient systems to respond contextually.

1.3.1 Ambient Context

One of the key components of ambient intelligence relies on the system's ability to interpret its environment based on the concept of context. Context is essential because it allows the system to understand and respond appropriately to situations. According to Kokinov [35], context is defined as the set of all entities within the system that influence the cognitive behavior of a human (or the system) in a given situation. He identifies four main characteristics of context:

- **Context as a "state of mind":** It influences the perception and reaction of an individual or system.

- **Lack of clearly defined boundaries:** Context does not have strict limits.
- **Association of relevant elements:** Context is formed by a set of elements that are meaningful when combined.
- **Dynamism:** Context evolves based on changes in the environment and interactions.

From another perspective, Pascoe [36] considers context as a subjective concept, depending on the perception of the entity experiencing it. For example, for one entity, context might be its spatial location, while for another, it could be its temporal position or even its emotional state.

In an ambient intelligence system, this notion of context is fundamental to enabling proactive and personalized adaptation. By understanding context dynamically and subjectively, the ambient system can respond to users' specific needs, anticipate their expectations, and offer a more intuitive interaction.

1.3.2 Application Scenarios of Ambient Intelligence (AmI)

cover a wide range of fields where intelligent environments can enhance daily life, optimize processes, and provide personalized services. Here are some typical scenarios [37, 38] :

1. Smart Home

- **Scenario:** A home equipped with sensors and connected devices automatically adjusts environmental conditions (temperature, lighting) based on the occupants' presence and preferences.
- **Functionality:** The system detects if someone is home, adjusts the temperature, turns off lights in unoccupied rooms, and even suggests music or videos based on residents' moods or activities.

2. Connected Healthcare

- **Scenario:** In a hospital or home setting, sensors continuously monitor patients' vital signs, detecting anomalies and automatically alerting caregivers when needed.
- **Functionality:** Wearable or implanted devices measure blood pressure, heart rate, and other parameters. In case of an emergency, the system can call for help, notify a doctor, or remind the patient to take medication.

3. Smart City

- **Scenario:** Intelligent systems manage traffic in real-time to reduce congestion, adjust street lighting according to brightness and pedestrian flow, and monitor air quality.
- **Functionality:** Smart traffic lights and signs adapt routes, guiding drivers along the best paths. Street sensors can also trigger safety alerts in case of unusual gatherings.

4. Smart Office

- **Scenario:** In an office environment, smart systems manage resources to create an optimized, comfortable workspace.
- **Functionality:** The system adjusts lighting and air conditioning based on the number of people present, organizes meetings according to participants' schedules, and enables contactless access to secure areas.

5. Personalized Education

- **Scenario:** In a school or university, AmI systems adapt educational resources and learning pace to meet individual students' needs.
- **Functionality:** By analyzing students' preferences and progress, the system suggests additional exercises, explanatory videos, or tailored review sessions, optimizing learning.

6. Smart Retail

- **Scenario:** In a store, sensors and cameras identify the products customers are looking at, suggesting complementary items or personalized promotions.
- **Functionality:** Using facial recognition and behavioral analysis, the store can adjust promotions in real time, optimize layout, and provide a more interactive shopping experience.

7. Elderly Care Assistance

- **Scenario:** In senior living facilities or at home, AmI systems monitor residents' activities and health, intervening if necessary.
- **Functionality:** Sensors detect if a person has fallen or is inactive for too long, sending alerts to family members or caregivers and reminding residents of medical appointments or medication.

8. Smart Transportation

- **Scenario:** Autonomous cars and connected public transportation adjust routes and schedules based on traffic and passenger needs.
- **Functionality:** Transportation systems monitor vehicle occupancy, adjusting schedules and routes in real-time. Smart cars communicate with each other to reduce collision risks and improve traffic flow.

These scenarios demonstrate how ambient intelligence can transform daily life, offering tailored services, improving efficiency, and enhancing comfort and safety across various domains.

1.3.3 Characterization of an Ambient Multi-Agent System

As in scenarios described in Section before, a group of individuals can be represented computationally by a set of software agents capable of interacting with each other and perceiving their surrounding environment. This system, composed of agents, forms a

Multi-Agent System (MAS). By extension, it is also possible to integrate additional software agents into this system, such as those acting as service or data servers.

There are often four main issues associated with MAS:

- The issue of action considers how a set of autonomous agents can act in a shared environment and how this environment interacts in return with the agents [39, 40].
- The modeling of the agent and its relationship to the world; the cognitive model of the agent must be able to implement actions that best achieve its goals. This decision-making capacity is tied to a "mental state" reflecting perceptions, representations, beliefs, along with certain "psychic" parameters (desires, tendencies, etc.) of the agent [41, 42].
- The issue of interactions between agents addresses central notions of coordination and cooperation [43].
- The issue of adaptation relates to the agent's behavioral adaptation according to context or learning. Adaptation can be purely individual or partially collective [44].

1.3.4 Context Sensitivity

Schilit et al. [45] were the first to introduce the notion of context sensitivity through the design of an active map service system, which provided information to a user based on their location. They defined context sensitivity as the ability of a mobile user's applications to detect and respond to changes occurring in the surrounding environment. Since then, several definitions have been proposed [46–48].

We can observe that two main aspects characterize context sensitivity. The first concerns the ability to perceive its context; this perception/detection notion is present in all definitions. The second aspect pertains to the capacity to leverage the perceived context. Therefore, being context-sensitive involves not only perceiving the context but also acting differently based on it.

Context sensitivity is a central issue in the field of ambient intelligence. A proactive and non-intrusive behavior would not be possible without a solid understanding of the user's context. Thus, the aim is to give an ambient agent the ability to dynamically adapt its functionality according to its current context.

Research on context sensitivity within ambient intelligence applications generally revolves around two aspects: first, the infrastructure enabling the capture and processing of contextual information, and second, the modeling and reasoning over this contextual information.

In the work of Olaru et al. [49], a hierarchical space system was proposed, allowing the direct specification of each agent's movement locality and other contextual elements. The global view of the system is abstracted, while the agent's context is modeled as a dynamic structure where coordination activities are captured using a pattern-matching technique. Furthermore, the authors highlighted several interesting projects implementing AmI systems.

In the smart home platform proposed by Guivarch et al. [50], a user's context is perceived within their own home. The user is assisted by a connected agent, whose architecture consists of four internal agents: a data agent, a user agent, a context agent,

and a controller agent. In this way, the agent's behavior can evolve according to the data collected from external sensors. Additionally, learning techniques are applied, which have demonstrated their value in a dynamic environment.

1.3.5 Characterization of BDI Agents

Many multi-agent architectures have been proposed. Some, such as finite state machines or motivational architectures, can be very useful for defining simple agents. However, they are not well-suited for defining complex cognitive agents, known for their proactivity and anticipation, as they have limited representational power.

The Belief-Desire-Intention (BDI) paradigm is one of the primary approaches dedicated to designing and building rational agents and multi-agent systems [31, 42]. Research, like that by [51], has highlighted the value of using this paradigm in simulation contexts. The theoretical foundations were initially described in the work of [52] on PRS1. Applications that utilize these concepts include the OASIS air traffic control system at Sydney Airport, Australia [53], and the SPOC business management system [54]. To our knowledge, the BDI approach is still infrequently used in practical applications, partly due to the apparent complexity of proposed architectures, such as JAM [39], compared to the more intuitive (purely algorithmic) programming generally preferred for specifying agent behavior.

Recently, the importance of the BDI approach has been reaffirmed through several specialized languages, such as Jadex [55], Jason [56], and 2APL [57].

The key idea of the BDI approach is to describe the internal state of agents, known as BDI agents, in terms of mental attitudes, and also to define a control architecture through which the agent can select its plan [58]. A BDI agent's reasoning mechanism, triggered by perceived events, is based on its mental attitudes represented by Beliefs, Desires, and Intentions:

- **Beliefs** represent what the agent believes about the current state of its environment.
- **Desires** (or goals) are an abstract notion specifying the agent's preferences. A crucial feature is that an agent can hold inconsistent desires and does not necessarily believe that its desires are achievable.
- **Intentions** represent the desires the agent commits to achieving. However, limited by its resources, the agent may not pursue all desires, even if they are consistent. It must therefore choose a subset of desires to commit to.

The agent has an explicit representation of its beliefs, desires, and intentions. We denote the agent's set of beliefs as **B**, its set of desires as **D**, and its set of intentions as **I**.

The sets **B**, **D**, and **I** can be represented using various knowledge representation models, such as first-order predicate logic, higher-order logic, production rule models, or even simple data structures.

To better understand the BDI paradigm theory initially proposed by Bratman [41], here's a realistic scenario that explains it: *"Agent Pierre believes that if someone spends their time studying, they can pursue a doctoral thesis. Additionally, Pierre has the desire to travel extensively, to complete a doctoral thesis, and to obtain an assistant position at*

the university. The desire to travel is inconsistent with the other two, and after some thought, Pierre decides to choose the latter two desires. Realizing he cannot fulfill both desires simultaneously, he decides to pursue the doctoral thesis first. At this point, Pierre has the intention to complete a thesis and will logically use all his resources to achieve this goal. It would be irrational for Pierre, once he has made his decision, to spend his time and energy, including his resources, on traveling around the world. By setting his intentions, Pierre has fewer choices to consider, as he has given up on exploring travel agencies for the best travel offer."

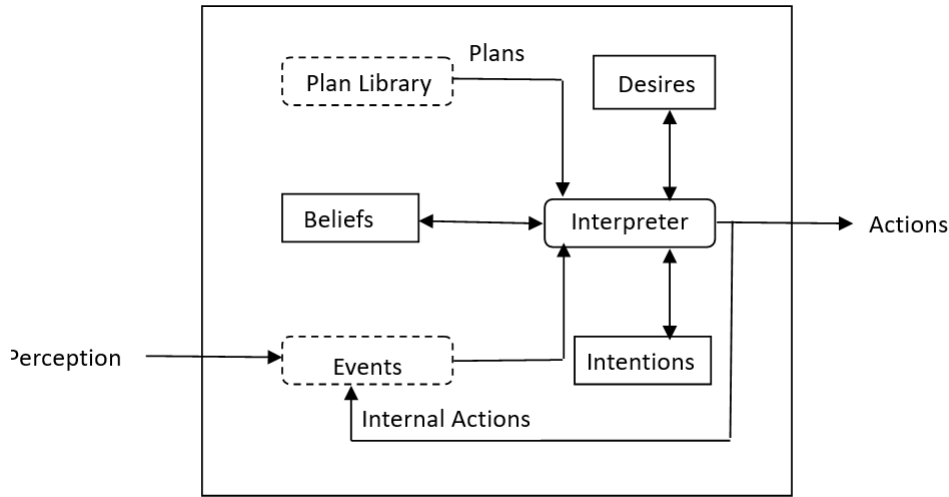


Figure 1.2: The Architecture of a BDI Agent

Belief-Desire-Intention (BDI) architecture

inspired by [59], Figure 1.2 illustrates the main components of the BDI architecture. Indeed, several studies have detailed the functioning of the BDI interpreter, such as [60,61].

The agent's reasoning mechanism, based on its Beliefs (B), Desires (D), and Intentions (I), is triggered by perceived events (Evt), enabling it to generate consistent action plans. These plans are drawn from a library of partial plans (Library of Plans), called LibP.

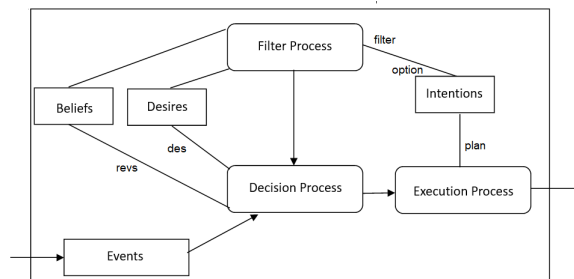


Figure 1.3: A Functional View of the BDI Interpreter

Figure 1.3 illustrates the operational scheme of the interpreter associated with the BDI (Belief-Desire-Intention) agent. The following functions describe its reasoning mechanism:

- **revs:** $2^B \times Evt \rightarrow 2^B$ is the belief revision function of the agent, triggered by new perceptions.
- **des:** $2^B \times 2^D \times 2^I \rightarrow 2^D$ is the function responsible for updating the agent's desires when its beliefs or intentions change, ensuring the consistency of the selected desires.
- **filter:** $2^B \times 2^D \times 2^I \times LibP \rightarrow 2^I$ is the function responsible for updating the agent's desires when its beliefs or intentions change, ensuring the consistency of the selected desires.
- **options:** $I \times LibP \rightarrow P$ is the function that associates each intention of the agent with specific partial plans from the plan library, $LibP$.
- **plan:** $2^I \rightarrow P$ is the function that enables the construction of an executable plan from partial plans using the options function.

Algorithm 1, designed by Michael Wooldridge, describes the reasoning mechanism of the BDI agent.

Algorithm 1: Reasoning Mechanism of the BDI Agent

```

1 repeat
2   Obtain new perceptions  $p$ ;
3    $B = \text{revs}(B, p)$  ;
4    $I = \text{options}(D, I)$  ;
5    $D = \text{des}(B, D, I)$  ;
6    $I = \text{filter}(B, D, I)$  ;
7    $P = \text{plan}(B, I)$  ;
8   Execute plan  $P$ ;
9 until true

```

1.3.6 BDI programming languages

In the literature, there are many programming languages dedicated to BDI agents [62], highlighting different aspects and development modules within the agent, such as goals, agent planning, and multi-agent organization (MAS) levels. Examples include Jadex [55], Jason [56], 2APL [57], and JIAC [63].

In 2006, agent-centered BDI approaches emerged with the goal of incorporating environmental dynamism into the expected behaviors. Specifically, the formal CanPlan model proposed in [64] extends the Can language to enhance action specification in plans by considering BDI-type mental attitudes. The paradigms used in the older languages have been reintroduced [62]:

- A plan is conditioned by the formal and explicit specification of application contexts.
- Plans are decomposed into goals, and goals into plans.
- The agent's mental process manipulates beliefs, desires, intentions, and even plans as a whole.

Everything seems designed to provide the programmer with a formal tool for global agent programming. However, the agent’s behavior is limited to the knowledge that the designer can embed at the development stage, specifically requiring known application contexts in advance. This represents a fundamental limitation in the context of ambient systems.

1.3.7 BDI Agent Planning

Early designs of distributed systems in the form of multi-agent systems (MAS) did not include anticipation in the planning process. Consequently, agent-centered BDI approaches have been proposed since the 2000s to address the dynamic environments in which agents operate.

In [64], a hierarchical model called Hierarchical Task Network (HTN) is proposed for cases where an agent’s plan fails, allowing for the selection of the next plan to execute. The execution approach is sequential; the next selected plan will generally be one that meets the agent’s contextual conditions. It may be either an alternative to the failed plan or another plan from the Goal-Plan hierarchy. In [65], the selection of an alternative plan is specified based on the successes and failures of past execution experiences. Thus, a plan’s applicability to a given agent context alone is insufficient for selection; it must also have a high likelihood of success based on experience. However, satisfactory selection is only achieved with a substantial amount of experience. In [66], a complementary approach is defined to estimate the propagation of failures in the Goal-Plan hierarchy, providing a broader basis for selecting the next plans to execute.

Orthogonally, the mutual consistency of an agent’s plans has been studied to enable concurrent executions. The GraphPlan planner by [67] builds on the concurrent planning work of [68] to apply it to BDI agents. It assesses conflicts among actions within plans, focusing on action execution preconditions and effects. GraphPlan can then efficiently construct a global plan as a flow of actions to fulfill a subset of desires (goals) that could be executed concurrently. However, GraphPlan only addresses specific cases of action scheduling, as the proposed concurrent execution follows a synchronous time-step approach. It is unclear whether an asynchronous approach would yield fewer achieved goals, yet synchronous and asynchronous approaches are often incomparable.

1.4 Conclusion

The exploration of Distributed Artificial Intelligence (DAI) and Multi-Agent Systems (MAS) reveals the transformative potential of these systems across various domains. MAS provide a robust framework for addressing complex, distributed problems by enabling autonomous agents to cooperate, communicate, and adapt dynamically to their environments. By integrating cognitive models, such as the Belief-Desire-Intention (BDI) architecture, MAS can achieve goal-oriented behaviors and improve decision-making in real-time scenarios.

The emergence of Ambient Intelligence (AmI) further enhances the utility of MAS by embedding intelligence into the environment, enabling seamless, context-aware interactions that anticipate and adapt to user needs. When combined with the dynamic capabilities of MAS, AmI systems represent a promising avenue for developing intelligent

environments across domains such as smart cities, healthcare, and education. Nonetheless, challenges remain in designing agents that can manage unforeseen contexts and concurrent interactions, which highlights the need for continued research in this field.

However, challenges remain, particularly in designing agents capable of handling unknown future contexts and managing concurrent interactions. Future research should focus on enhancing learning mechanisms within MAS to improve agents' adaptability and resilience in unforeseen circumstances. As technologies evolve, the synergy between MAS and AmI will likely yield even more innovative applications, pushing the boundaries of how autonomous systems can enhance human life in an increasingly connected world.

Enhancing Multi-Agent Organizations with Artificial Learning Paradigms

This chapter analyzes multi-agent organizations and the artificial learning paradigms that enhance their functionality. We begin by examining the fundamental characteristics of multi-agent organizations, covering key definitions and approaches to their representation and management. Next, we analyze a selection of existing organizational models, focusing on how they enable openness in the representation, deployment, and management of multi-agent systems.

The chapter then transitions to an in-depth examination of artificial learning techniques. The first section addresses supervised learning, a critical component that equips agents with classification and prediction capabilities. We then explore Unsupervised learning methods allow agents to autonomously identify patterns and clusters within data. Finally, Reinforcement learning enables agents to acquire optimal behaviors through trial-and-error interactions with the environment, guided by reward signals.

This structured examination presents a holistic view of how artificial learning techniques can be applied within multi-agent systems, setting a strong foundation for the practical applications discussed in subsequent chapters.

2.1 Overview of Multi-Agent Organizations

In the previous chapter, agents were defined as autonomous software entities situated within an environment and capable of exhibiting proactive and social behaviors [69]. Therefore, an agent can independently decide its actions, interact with other agents, and interact with its environment. The initial work on specifying and implementing these agent properties developed within closed multi-agent system (MAS) contexts, where the designers of MAS and the agent programmers were usually the same individuals [70, 71]. As a result, early research did not address issues such as data exchange among heterogeneous agents, integration with external entities, and control of potentially malicious behaviors. Agent heterogeneity encompasses various dimensions [72, 73], including:

- **diverse origins:** agents developed by different programmers.
- **varying architectures:** Differences in cognitive/reactive frameworks.

- different implementation languages.
- heterogeneous skill sets.

The absence of heterogeneity issues in early MAS work was mainly because these applications did not require interactions between agents from different programmers. As MAS technologies evolved toward more open applications, work on communication languages [74] emerged, enabling heterogeneous agents to interact and understand each other.

This initial step toward openness was followed by a desire to enhance the ability of agents to search for those with the skills or services that best meet the needs of a system or another agent. Thus, studies emerged on brokerage systems [75] and "yellow pages" [76].

The standardization of a communication language, FIPA ACL [76], and a MAS architecture with a yellow pages service by FIPA has contributed to establishing a foundation for openness. As a result, multi-agent applications can now develop independently of agent programming.

The availability of a yellow pages service and a shared communication language within a MAS allows for the independent development of agents and applications. These applications must then include mechanisms for recruiting agents from the yellow pages and coordinating their activities, as seen in systems like Karma [77]. However, the lack of modeling tools to address the increasing complexity of applications necessitated methodologies [78, 79] and models [80] that further promote the development of open MAS. Additionally, agent heterogeneity introduces a new problem: the possibility of malicious agents, a factor not previously considered. While benevolent agents generally perform the tasks assigned to them within an application without deviating from expectations, malicious agents often exhibit behaviors not anticipated by the system.

Organizational models thus provide MAS with tools for modeling and specifically structuring agents, representing their task distributions, defining the rules governing their behaviors, and the regulations that stem from these rules [81]. Modeling reduces the complexity of problems and facilitates modifications and adaptations of software aspects.

- **Structuring** is typically defined in organizational models—some of which we will examine later in this chapter—using concepts such as teams, groups, scenes, and roles. Generally, agents play one or more roles within one or more teams, scenes, or groups.
- **Task distribution** results from decomposing an organization's objectives into sub-goals and tasks, which are then assigned to roles. These tasks and sub-goals will be carried out by agents assigned to the corresponding roles.
- **Regulation rules** are defined by norms that establish the permissions, obligations, and prohibitions of roles, and consequently, of agents within an organization.

2.1.1 Definition of Multi-Agent Organization

Despite extensive research on modeling and managing organizations, there is no single, universally accepted definition of "organization." Some definitions in the literature include the following:

- An organization is a decision-making and communication framework applied to a group of actors who perform a set of tasks to achieve goals while maintaining a coherent system state [82].
- An organization is characterized by a decomposition of tasks, a role distribution, an authority system, a communication system, and a reward and penalty system [83].
- An organization is a structured system with distinct patterns of activity, knowledge, culture, memory, history, and capacity, separate from those of an individual agent [84].

While these definitions differ in appearance, they share similarities, as each generally includes elements of structuring, task distribution, and regulation in different formulations.

For the purposes of this thesis, a multi-agent organization is defined as follows:

Definition: A multi-agent organization is a set of agents that interact according to a predefined structure, cooperation patterns, and norms to achieve specific goals.

This definition reflects that our work primarily focuses on organization-centered development approaches for organizational models [85]. Within MAS, two main research communities address the issue of multi-agent organizations:

1. **SASO (Self-Adaptive and Self-Organizing Systems)** community [86], which uses agent-centered approaches [85]. These approaches focus on developing local behaviors for each agent and suitable peer-to-peer interactions, with the emergent result forming an organization. This organization-building process is bottom-up.
2. **COIN (Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems)** community uses organization-centered approaches. In COIN, organizations and the mechanisms for coordinating agent behaviors are specified beforehand. Unlike SASO, where the operation of organizations can lead to unpredictable phenomena that may not be verifiable *a priori*, COIN aims to ensure specific behavioral invariants among agents [86, 87].

After clarifying the distinction between these two primary approaches to multi-agent organization development, we can observe that our definition includes both aspects addressed by the top-down approaches. On one hand, explicit modeling of the structure, goals, and task allocation to agents is required. On the other, the concrete management of organizations throughout their lifecycle, from creation to goal achievement and through structural and functional evolution, is essential. Considering these two aspects has led to a distinction between two components of a multi-agent organizational model, called the **Organization Modeling Language (OML)** and the **Organization Management Infrastructure (OMI)** [85].

Consequently, various studies propose organizational models that provide either one of these two components or both, with different concepts and approaches, which will be examined further in section 2.4. To facilitate understanding, we will first present the concepts of OML and OMI.

2.1.2 Organization Modeling Language (OML)

An Organization Modeling Language (OML) is a language that relies on an organizational model to represent an organization's structure, the patterns of cooperation among agents, and, potentially, the norms that govern the assignment of roles, tasks, and the expected behaviors of agents [85].

Representing each of these elements constitutes a dimension of an organization [88,89]. The concept of dimensions was introduced by the Moise+ model [88], where an OML represents an organization across three main dimensions: structural, functional, and deontic. These dimensions allow for the description of the organization's structure, cooperation patterns, and norms, respectively. The idea of dimensions has since been generalized to categorize the decomposition methods used in various OMLs found in the literature. For instance, L. Coutinho et al. in [89] identified five primary dimensions proposed by different models: structural, functional, contextual, dialogical, and normative.

Structural Dimension

This dimension describes the decomposition of an organization into teams [90,91], groups [92], or scenes [93,94], in which roles are defined for agents to adopt.

Functional Dimension

In the functional dimension, an organization's objectives are defined and further broken down into sub-goals and tasks, along with the sequencing that establishes their dependencies and the order in which tasks are carried out [citeesteva2002islander, garcia2007distributed, tambe1997towards]. Through this dimension, agents assess their potential to cooperate and contribute to the achievement of an organization's goals.

Contextual Dimension

The contextual dimension allows for describing the various situations that agents may encounter as they evolve within an organization [90,93]. This dimension consists of contexts that help define these situations as well as the potential behaviors of roles that agents adopting them might exhibit [95].

Dialogical Dimension

The dialogical dimension describes communications among agents within an organization, including a set of protocols and an accompanying ontology that clarifies the semantics of the exchanged messages [93,96]. This includes the description of protocols and the ontology that agents use for coherent interaction.

Normative Dimension

Agent behaviors are constrained by norms that establish their rights (permissions), duties (obligations), and prohibitions within an organization. These norms collectively form the normative dimension [93,96,97].

Table 2.1 provides a synthesis of the dimensions proposed in various OMLs found in the literature. Each dimension’s expressiveness varies by OML, with symbols indicating whether a dimension is absent, has low, medium, or high expressiveness within a given OML.

Table 2.1: Summary of OML Dimensions [89]

OML	Structural	Functional	Contextual	Dialogical	Normative
AGRS [98]	++	–	–	+	–
BRAIN [99]	+	–	–	-	-
VE [100]	+	++	–	-	++
Islander [93]	+	–	++	+++	++
Moise+ [88]	+++	+++	–	-	++
MoiseInst [97]	+++	+++	++	-	+++
OperA [96]	++	++	+	++	+++
RIO [79]	+	+	–	+++	–
ROAD [101]	+	–	–	+	–
Teamcore [91]	++	++	–	-	–
THOMAS [102]	++	++	–	-	++

¹ – : Dimension does not exist in the OML

² + : Dimension exists with low expressiveness

³ ++: Dimension exists with medium expressiveness

⁴ +++: Dimension exists with high expressiveness

VE = Virtual Enterprise

Describing an organization with an OML constitutes an Organizational Specification (OS). Despite differences in expressiveness, each OML represents a language for describing organizations. From these OS, heterogeneous agents can reason about organizations and develop strategies for participating in the achievement of these organizations’ objectives. Thus, OMLs help to consider organizations as a means of fostering openness in MAS.

After explaining the role of OMLs in promoting openness, the next section introduces the second element of an organizational model, the Organization Management Infrastructure (OMI), which manages agent activities within organizations.

2.1.3 Organization Management Infrastructure (OMI)

The Organization Management Infrastructure (OMI) is a platform that enables the creation and management of organizational entities whose Organizational Specification (OS) is described using the OML for which the OMI was implemented [85]. An Organizational Entity (OE) is an instance of an organization, comprising an OS, agents, and all resources required for managing the state of the organization’s instance [97]. As presented in section 2.1.2, an OS outlines the structure and objectives of an organization, as well as the rules governing its evolution and the constraints placed on agent autonomy.

2.1.4 Approach to Analyzing Openness in MAS Organizations

R. Scott [91] defines an organization as an open system as follows: "An organization is a coalition of interest groups that achieve goals through negotiation; the structure of the coalition, its activities, and its outcomes are strongly influenced by environmental factors."

This definition highlights the influence of the environment on an organization, consistent with our analysis of openness in systems presented in the previous chapter. In that chapter, the environment of a system was defined as anything external to and not controlled by the system. Similarly, the environment of a multi-agent organization includes all agents and resources that are not part of the organization and, therefore, not under its control.

In this section, we will examine how specific elements of MAS organizations—namely structure, objectives, cooperation patterns, and norms, which are described by an OML and managed by an OMI—are influenced by openness properties, and how these elements affect the organization's environment. We will study these influences concerning the properties of interoperability, agent entry and exit, and control within a multi-agent organization.

Interoperability

The interoperability of a system characterizes its interactions with entities in its environment, particularly in terms of data exchange and the system's ability to use this data.

In the context of multi-agent organizations, interactions between external agents and an organization typically involve requests for an Organizational Specification (OS), procedures for entry/exit, and information related to the Organizational Entity's (OE) current state (e.g., organization members, ongoing objectives, managers). External agents need to understand the structure of the organization, for example, to determine which role they might adopt. If the organization is structured into groups, scenes, or teams, the OS also helps external agents identify the structural entity (e.g., team or group) they could join. Additionally, the OS allows agents to identify the organization's objectives and services, guiding their choices about cooperation before joining [108].

Thus, analyzing the syntactic interoperability of organizational models involves determining whether the models accommodate external agents' need for varying levels of access to the OS and OE of an organization, as well as access to explicit entry/exit procedures.

The exchanges between an organization and agents in its environment occur through exchange interfaces. Organizational models should enable external agents to request or access data that could potentially facilitate their entry into an organization. Conversely, organizations need tools to manage these exchanges with their environment. Structural interoperability in organizational models is therefore primarily managed by OMIs, which should provide components or mechanisms representing the exchange interfaces between an organization and its environment.

The third characteristic of interoperability, the capacity to use exchanged data, relies on a shared ontology between the agents and the organization to interpret the semantics of data. Analyzing this aspect in an organizational model involves determining whether the model employs a standard ontology or provides data exchange protocols, along with their semantics and interpretation domains, to ensure mutual understanding between external

agents and the organization.

Agent Entry and Exit

A multi-agent organization aims to achieve objectives that no single agent could accomplish alone. Therefore, allowing the entry of agents who can contribute to these objectives is beneficial. Consequently, agent entry into an organization is more complex than in an MAS platform.

Practically, an agent cannot simply register with the MAS's yellow pages service to become a member of an organization. An organization, whether structured with roles alone or also with groups, teams, or scenes where roles are defined, requires an agent to be integrated into its structure upon entry. Moreover, agents who enter an organization must participate in achieving its objectives.

Organizational models, however, allow for describing and managing organizations with various types of objectives. For instance, in e-commerce organizations, the goal is to facilitate proper transactions between agents [109]. In a scientific conference management organization, the goals might include allowing authors to submit papers and managing the article selection process [110]. Each type of organization has unique entry and integration procedures, depending on its objectives and structural representation (whether role-only or including structural entities and roles).

Regarding exit, procedures also vary across models, though the resulting issues are similar within organizations. Openness requires allowing existing members of an organization to exit. However, since agents are integrated into an organization's structure and contribute to its objectives, managing an exit requires considering the potential impacts on the organization's structure and objectives to address any instability or dysfunction that may arise. This section will also review some proposals made by organizational models to handle these issues.

Control within an Organization

Control is often considered a core characteristic of organizations in general, and therefore of MAS organizations. An organization enables structuring a system and specifying the responsibilities, rights, and duties of its members. As a result, multi-agent organizations are frequently viewed as normative systems, which establish norms, ensure adherence to them, and, if needed, update them. However, not all multi-agent organizational models are normative (see Table 2.1). Models without a normative dimension assume agents are benevolent by default, which is a significant assumption given agent autonomy. Thus, two types of control approaches emerge with such models:

1. A control approach that nearly entirely restricts agent autonomy, meaning that no unexpected actions can be taken within the system.
2. An organizational management approach without any control mechanisms, where the organization relies on agents' benevolence, though this poses a high risk with openness.

For multi-agent organizational models, the key challenges in establishing control involve representing the aspects to be controlled, particularly agent actions, cooperation,

and interactions concerning the organization; defining rules for control management; and implementing techniques that suit organizational needs.

Having outlined the specific characteristics related to openness within multi-agent organizations, we now turn to a more detailed analysis of entry/exit and control properties based on specific models and their respective OML and OMI implementations.

2.1.5 Non-Normative Organizations

The growing interest in multi-agent applications has led to various non-normative organizational models. This section reviews a few of these models and highlights how the absence of a normative framework affects entry, exit, and control processes.

RIO (Role Interaction Organization)

The RIO model focuses on specifying interaction protocols among agents within open organizations, structuring interactions through "micro-roles" that determine message flows. Entry is centralized, with agents created and registered in a platform, acquiring roles dynamically without specific entry requirements. Exit is minimally addressed, and control is absent, assuming agent benevolence.

ROAD

In ROAD, an agent's autonomy level can be set based on the role it adopts, providing flexibility. Entry is based on the agent's skills for a given role, though specifics on skill verification are not provided. Control is minimal for agents with full autonomy, relying on contracts for partially autonomous agents.

BRAIN

The BRAIN model uses roles to define agent capacities and behaviors, which are implemented and provided to agents. Entry is based on role adoption through certificates, while exit simply involves certificate cancellation. Control is not specified, as agents lack autonomy to change roles.

AGRS

In AGRS, entry into groups is based on conditions like skills, group state, or architecture, though specific protocols are unspecified. AGRS lacks a normative dimension and relies on ad hoc control.

TeamCore

The TeamCore model organizes agents into hierarchical teams and roles, where agents are recruited by the organization itself. Control over agent behaviors is limited, with the OMI reassigning tasks when agents abandon roles.

2.1.6 Normative Organizations

Normative models enable explicit specifications of norms to constrain agent behaviors, influencing the entry, exit, and control within organizations.

THOMAS

The THOMAS model applies a service-oriented approach with norms linking roles and services. Entry is centrally managed by assigning roles, though verification mechanisms are unspecified. Control relies on norms specifying obligations, permissions, and sanctions, with sanctions and rewards managed by designated roles.

Islander

Islander structures electronic institutions into scenes, with norms defining speech acts per scene. Entry is managed through role conditions, while control is preventive, with mandatory adherence to norms.

OPERA

In OPERA, organizations are structured into groups and scenes with detailed objectives and norms for roles. Entry requires an agent to meet role goals and plan compatibility, forming a social contract. Control is based on contract adherence, allowing some negotiation of terms.

Virtual Enterprises

In Virtual Enterprise models, organizations consist of consortiums that outsource goals to member companies. Entry is based on companies meeting goal requirements, and control relies on contractual obligations with flexible deadlines, allowing companies to negotiate goal terms.

Moise+ and MoiseInst

The Moise+ model, along with its extension MoiseInst, provides an organizational structure for multi-agent systems by defining roles and groups. Each group consists of a set of roles, and the organization's goals are broken down into missions and sub-goals. A mission is a set of goals to be achieved, and these missions are associated with specific roles via norms. Therefore, an agent entering an organization described by these models will adopt one or more roles and is then expected to accomplish the missions assigned to those roles. In Moise+ and MoiseInst, control of agent behavior is heavily based on norms. Norms establish expected behaviors, permissions, and obligations for agents in their roles and across groups. This normative dimension allows the organization to manage how agents fulfill their missions, interact with other agents, and align with the organization's structural rules. The models provide a structured approach to organizing agents into a cohesive system with explicit responsibilities and constraints, ensuring agents act in ways that contribute to the overall goals of the organization. The reorganization features of MoiseInst, in particular, allow the organization to remain stable and functional even when roles and missions change dynamically.

Summary

Non-normative models generally lack formal mechanisms and assume agent compliance, limiting their application in scenarios with high openness. In contrast, normative models implement norms and rules to guide agent behavior, assigning roles for oversight, enforcement, and even sanctions or rewards. Moise+ and MoiseInst, for instance, employ structured control through roles and mission-based norms, while OPERA and Virtual Enterprises use contractual commitments to define expectations and consequences.

2.2 Artificial Learning

Artificial learning (or machine learning) refers to a program’s ability to improve its performance based on data acquired during its operation, that is, from its experience [111]. This type of system is used to solve tasks that are too complex for classical algorithms. Learning has been a central topic since the beginnings of artificial intelligence, and there are numerous approaches to it. These approaches are generally grouped into three categories, differentiated by the type of information available to the system for learning and the protocol with which it interacts with its environment: supervised learning, unsupervised learning, and reinforcement learning [112].

We provide a detailed examination of supervised learning techniques, given their pivotal role in our approaches. By thoroughly exploring these methods, we lay the groundwork for their application and refinement within our model. Additionally, we present an overview of unsupervised and reinforcement learning techniques. Unsupervised learning is discussed for its ability to identify patterns in unlabeled data, offering insights into clustering and data structure, while reinforcement learning introduces valuable strategies for real-time adaptability through feedback-driven approaches. Although not central to our method, these techniques provide a broader understanding of learning paradigms relevant to similar systems.

2.2.1 Supervised Learning

Supervised learning refers to techniques that rely on an oracle to guide the learning system. In the first stage, the oracle provides labeled examples, which consist of a set of m pairs comprising a data point x and the expected output for that data point u . This set is called the training sample and is denoted S .

$$S = (x_i, u_i)_{1 \leq i \leq m} = (x_i, f(x_i))_{1 \leq i \leq m} \quad (2.1)$$

The function f , sometimes referred to as the target function, is known only to the oracle and is the object of the learning process for the system. By utilizing this sample, the learner must find (or approximate) the correct output $u_n = f(x_n)$ corresponding to an input data point x_n that does not necessarily belong to S . The function estimated by the learning program is called a *hypothesis*.

A supervised learning algorithm searches the space of possible hypotheses for the one that is most suitable with respect to the training sample. However, there exists an infinite number of functions that can pass through a given set of points. And none of these functions is inherently better than another for describing the set of points in S and

interpolating the expected results. Choosing which one to retain is one of the fundamental problems of artificial learning. It is essential to ensure that the learner has a means to constrain it to converge towards a hypothesis that will be appropriate for the data it will be applied to after learning. This could be, for example, the maximum margin in the case of support vector machines. More generally, the principle of Occam's razor, which always favors the simplest solution, is a possible response.

In fact, it involves making a trade-off between complex hypotheses that fit the data better and simpler ones that are better for generalization [113]. If the hypothesis found is too complex, the system suffers from overfitting: it no longer makes errors on the training data but increasingly makes errors on new examples to identify, much like a student who memorizes their lessons without being able to generalize. Conversely, a hypothesis that is too simple does not adequately capture the reality of the data. This happens, for example, when using a linear separation method on a non-linear sample.

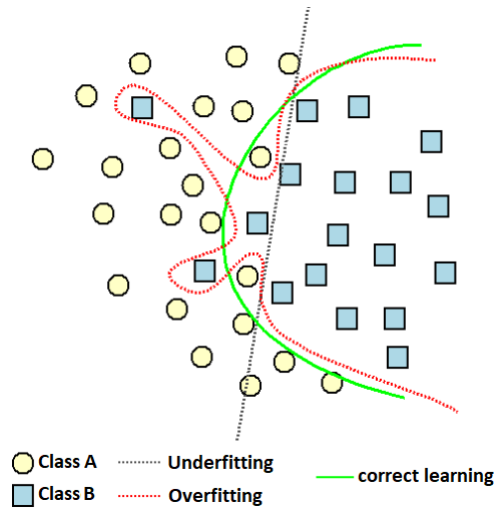


Figure 2.2: Different learnings of the separation of 2 classes in 2 dimensions, on a noisy sample

Figure 2.2 shows an example of overfitting, underfitting, and acceptable learning for separating two classes in two dimensions on a sample of noisy data. The green curve represents the expected separation. The red curve shows the result of overfitting: the data (including noise) are perfectly separated, but a new data point to classify is likely to be misclassified (there is no generalization). Finally, the gray line is the result of underfitting, where the hypothesis is too simple to match the data.

Supervised learning is commonly used for classification problems or pattern recognition. Seven of the main approaches are presented below.

Nearest Neighbors Method

Often abbreviated as kNN in English (or kPPV in French), the k-Nearest Neighbors method is simple yet effective in many cases. The principle is to assign the input data to the majority class among its closest neighbors in the training sample.

There are numerous variations of this method, depending on the distance function used or the weighting of neighbors relative to one another. Its main limitation is its high

computational cost, primarily due to the search for neighbors within a potentially large sample. Therefore, it is not suitable when dynamic learning is required.

One way to reduce computational costs is to build a model of the sample on which to base the learning [114], but this is not always sufficient. Additionally, the distribution of classes in the training sample can bias the final result. Indeed, if one class is significantly more represented than others, it is likely to dominate among the k nearest neighbors of the tested data [115]. This bias can be reduced by choosing an appropriate weighting (e.g., inversely proportional to distance). Finally, the choice of k is also a recurring issue: if it is too small, noise in the training sample strongly impacts the result, but if it is too large, class boundaries become less well-defined.

Decision Tree Inference

Decision trees aim to classify an object through a sequence of tests on its attributes. These tests are organized hierarchically, so that the outcome of one test indicates which test to perform next, and so on until the final test leads to the ultimate classification. This results in a tree structure where the nodes represent tests and the leaves represent classes (with several leaves potentially corresponding to the same class). In the example in Figure 2.3, objects are defined by the attributes surface and equipment and are classified into the categories basketball court, rugby field, and football field. Creating this tree requires considerable domain knowledge, such as knowing that rugby is not played on asphalt.

In addition to its graphical form, a decision tree can be represented as a set of logical formulas. A branch leading from the root to a leaf is essentially a conjunction of properties about the object, sufficient to associate it with a particular class. For instance, in Figure 2.3, the branch leading to the leaf labeled *rugbyfield* can be described by the formula $(surface = grass) \wedge \neg(equipment = cage)$. Similarly, a class can be represented by the disjunction of all branches leading to leaves of the same classification.

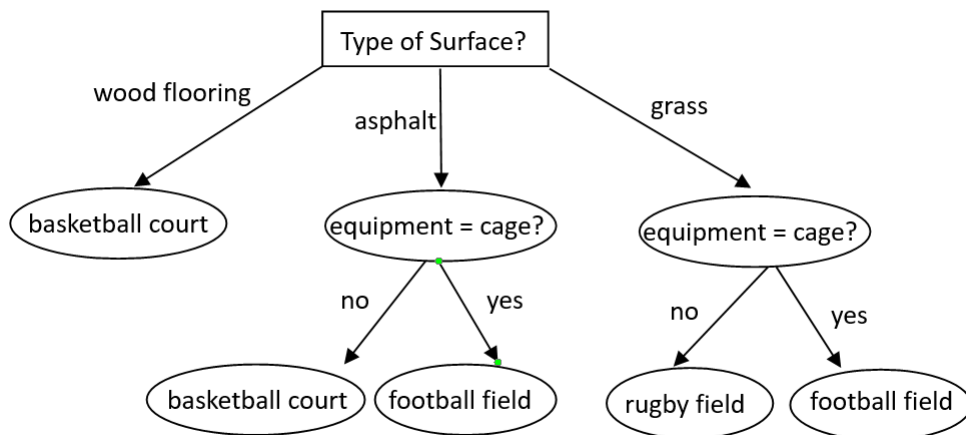


Figure 2.3: Example of a Decision Tree

Continuing with the same example, the class basketball court is represented by the formula $(surface = wood) \vee ((surface = asphalt) \wedge \neg(equipment = cage))$.

In supervised learning, the goal is to automatically build a decision tree from the training sample. A common approach is to recursively partition the sample based on the

value of each attribute, creating a node for each partition. This process continues until a partition contains only data associated with the same output value, forming a leaf. This approach is known as top-down induction of decision trees. Foundational algorithms based on this idea include CART [116] and ID3 [117]. Given the vast space of possible trees for a given sample, it's important to explore it intelligently to produce an effective tree—that is, one that is sufficiently accurate but does not contain too many leaves. The order of attribute selection for partitioning strongly impacts the result, so a metric is needed to rank and select the most appropriate attribute, such as cross-entropy, which measures the correlation level between an attribute and class distribution [118].

The advantage of decision trees is that they are often concise and understandable. Additionally, unlike the kNN method, decision-making is computationally inexpensive once the tree is built. However, using decision trees requires a compatible data structure, which can be difficult to obtain or costly to use, depending on the problem. Decision graphs are an extension of trees that help reduce node redundancy [119]. Recent refinements focus on accounting for the cost of decision errors [120].

In practice, however, there is the problem of overfitting. If a basic induction algorithm is allowed to run to completion, the generated tree becomes overly precise: it is too large, and each leaf is "pure" (meaning it corresponds to a single class). Noise in the training sample is particularly problematic in this case. This is why it's necessary to prune the tree. This can be done by adjusting the algorithm's stopping criteria (pre-pruning) or by gradually moving leaves toward the root while monitoring an error criterion (post-pruning) [121].

This remains an open research topic and is a challenge for the easy application of this method. Another major limitation to its application in real-world cases is its low robustness to missing data.

Support Vector Machines

Originating from the theoretical work of Vapnik [122], Support Vector Machines (SVMs) are also known as large margin separators. They primarily rely on two pre-existing concepts: kernel functions and the notion of maximum margin. A kernel function is a probability density that is symmetric with respect to the y-axis, such as the Gaussian distribution. In supervised learning, the challenge often lies in finding the separation between samples from different classes. The samples closest to the separating boundary are referred to as *support vectors*. The margin is the distance separating them from this boundary, and theory suggests that it should be maximized.

In the case of a linear separation of two classes, the goal is to find the hyperplane that separates the classes while maximizing the margin. The equation of a hyperplane is given by:

$$h(x) = w \cdot x + w_0 \tag{2.2}$$

where w is the normal vector to the hyperplane and w_0 is a constant representing its origin. A point (x_i, y) is correctly classified if and only if $y \cdot h(x_i) > 0$

A primal expression of the problem is then:

$$\begin{cases} \min(\frac{1}{2}\|w\|^2) \\ \forall i, u_i(w \cdot x_i + w_0) \geq 1 \end{cases} \quad (2.3)$$

where w (the normal vector of the hyperplane) and w_0 are the parameters to find, and x_i and u_i are the data from the training sample. As is, the problem is difficult (if not impossible) to solve when the dimensionality of the input data is large. It is preferable to express it in its dual form, which no longer depends on the dimensionality of the data but rather on the size of the training sample, Find the Lagrange multipliers α such that:

$$\begin{cases} \max\{\sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j u_i u_j (x_i \cdot x_j)\} \\ \alpha_i \geq 0, \quad i = 1, \dots, m \\ \sum_{i=1}^m \alpha_i u_i = 0 \end{cases} \quad (2.4)$$

where m is the size of the training sample. The *Kuhn-Tucker* theorem demonstrates that the solution to the dual problem and that to the primal problem are the same [123]. The solution hyperplane is then given by:

$$h(x) = (w^* \cdot x) + w_0^* = \sum_{i=1}^m \alpha_i^* u_i \cdot (x \cdot x_i) + w_0^* \quad (2.5)$$

where the α_i^* are the solutions of equation (2.4) and w_0^* can be calculated from a support vector. The support vectors are the only ones with a non-zero Lagrange multiplier, and thus they are the only ones defining the optimal hyperplane. This is why they are sometimes called "critical examples".

In the nonlinear case, the solution involves transforming the representation space of the training sample into a higher-dimensional space where there is a linear separation. But how do we find this nonlinear transformation Φ ? In practice, this often equates to knowing the solution in advance. This is where kernel functions come into play. Given Φ , the problem to solve would be to find the α such that:

$$\begin{cases} \max\{\sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j u_i u_j (\Phi(x_i) \cdot \Phi(x_j))\} \\ \alpha_i \geq 0, \quad i = 1, \dots, m \\ \sum_{i=1}^m \alpha_i u_i = 0 \end{cases} \quad (2.6)$$

It is noted that Φ only appears in the dot product $\Phi(x_i) \cdot \Phi(x_j)$. Instead of finding Φ , one can seek to find the function k such that:

$$k(x, x') = \Phi(x) \cdot \Phi(x')$$

This function k is a kernel function. When well chosen, it allows for the use of non-vector representations and avoids computing the representation of examples in the new space. Several kernels are commonly used and sometimes combined: linear, polynomial, Gaussian, or Laplacian kernels [124].

Finally, it is often necessary to relax the constraints, for example, to manage noise in the training sample. To do this, slack variables are introduced. The user must then set a constant to adjust the trade-off between maximizing the margin and minimizing classification errors.

Relying on optimization methods, SVMs can handle high-dimensional data and yield good results in practice. Furthermore, they provide strong theoretical guarantees due to their origins but remain challenging to implement in real-world cases.

Genetic Algorithms

Developed by Holland in [125], genetic algorithms are an optimization technique that models a simplified version of species evolution via natural selection. The adaptation achieved through this process can be seen as a learning mechanism. Here, we outline the basic principles of genetic algorithms along with their application in supervised learning.

The first requirement for applying an evolutionary algorithm (another name for genetic algorithms) is to have a decoupling of the hypothesis space. There must be a *phenotypic* space in which hypotheses can be evaluated, and a *genotypic* space in which they can be manipulated and transformed using specific operators. Thus, a hypothesis is represented by its genotype, a chain of values (such as a bit string). The goal is then to create an initial population of hypotheses and to crossbreed these individuals to evolve their genotype from one generation to the next until a satisfactory hypothesis is reached. Practically, the initial generation is typically random, and the evolution process unfolds in four steps, repeated until a stopping criterion is met.

- **Step 1: Evaluation.** Evaluating the current generation is the first step. This usually involves assessing each individual with a specific function, called a fitness function, which applies to a phenotype and returns a performance score. In supervised learning, this often means testing each hypothesis on a validation set, though various evaluation functions can be used depending on the problem and may even include simulations [126].
- **Step 2: Selection.** The second step involves selecting individuals as breeders, whose genotypes will be crossed to create new, potentially more effective phenotypes. This step mimics environmental pressure. The simplest solution is to select the top n individuals. However, maintaining diversity can prevent local minima. A common approach is to assign each individual a selection probability that increases with its evaluation score, allowing all individuals a chance to reproduce while favoring the most fit. To reduce sensitivity to potential *fitness* function errors and to save computational costs, another approach is tournament selection, where individuals are compared in small groups, retaining only the best [127].
- **Step 3: Crossover.** Once breeders are selected, their genotypes are crossed *cross-over*. This involves combining their genotypes with a crossover operator that generates two new genotypes from a parent pair. For example, a classic operator known as "single-point crossover" randomly selects a point in a genotype and swaps genes between the parents from this point to the end of the chain. A two-point crossover variation also exists, as illustrated in Figure 2.4. By crossing individuals presumed to be among the best, we hope to produce a new individual combining the advantages of both parents. If unsuccessful, the individual is likely to be eliminated in the next selection round. Along with crossover, each gene has a mutation probability, allowing it to randomly change value whenever a new individual is created. This can introduce new phenotypic traits, which will be retained by the selection process if beneficial, or discarded otherwise.

However, if the mutation rate is too high, preserving already acquired beneficial traits may be jeopardized.

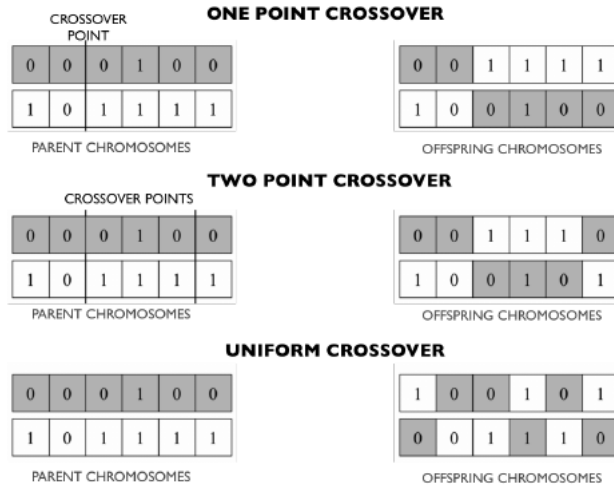


Figure 2.4: Crossover Operators for Genetic Algorithms

- **Step 4: Replacement** The final step involves replacing the current population with the new generation produced through crossover and mutation. These four steps are repeated in a loop, creating successive generations where the best individuals become increasingly well-adapted, as indicated by higher evaluation scores. The algorithm concludes when a predefined stopping criterion, specific to the problem, is met. In supervised learning, this could mean achieving error-free classification of a validation sample by one of the hypotheses in the current generation.

Among the specializations of genetic algorithms, one notable variant is the evolution of automata, which utilizes finite-state automata as genotypes and graph operations as mutation operators, foregoing crossover [128]. Another well-known variant, genetic programming, creates programs using a genetic algorithm, often by representing programs as trees. If the target programming language has a sufficiently simple grammar, crossover and mutation operations on a tree can maintain the validity of the resulting program [129].

A primary limitation of this approach is its applicability. Selecting parameters such as the size and composition of the initial population or the encoding of objects into genotypes is non-trivial. Similarly, choosing the right crossover operator and mutation rate is crucial for algorithm performance and is difficult to determine beforehand. Consequently, research has focused on mechanisms for adapting these parameters during runtime [130]. The best-known technique, CMA-ES, relies on adapting the covariance matrix of the probability distribution that governs mutation [131]. In a similar vein, coevolution refers to cases where the fitness function changes over time or even varies between individuals or groups of individuals. This approach is valuable for decomposing a problem into parts [132]. Finally, for genetic algorithms to yield good results, a sufficiently large population and numerous generations are essential, which makes them computationally expensive.

Artificial Neural Networks

First introduced by McCulloch and Pitts in 1943 [133], neural networks (sometimes called connectionist networks) are a cornerstone of artificial intelligence and machine learning. Widely studied and used in numerous applications, we will focus here on their application in supervised learning, with particular emphasis on a specific type known as *perceptrons*. Initially proposed by Rosenblatt in 1957 [134] and later extended to a multi-layer architecture by Fiesler in 1996 [135], this type of network is well-suited to the classification problems typical of supervised learning.

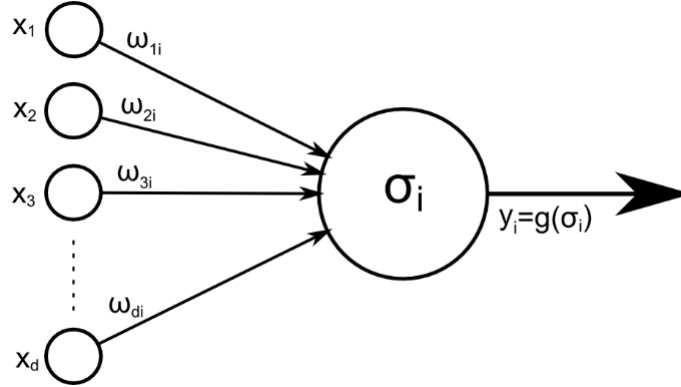


Figure 2.5: Model of a Formal Neuron

Figure 2.5 represents the basic unit of information processing in a connectionist network: the formal neuron. This neuron has a set of inputs called sources and a single output. It is represented by its state σ_i (where i is the neuron's index in the network) and by its output function g , which it uses to compute its output value y_i .

$$y_i = g(\sigma_i) \quad (2.7)$$

Most often, the function g is either the sign function or a sigmoid function (i.e., an S-shaped function), given by the formula:

$$g(x) = \frac{1}{1 + e^{-\lambda x}} \quad (2.8)$$

Each input j of a neuron i is associated with a weight ω_{ij} . The value of σ_i is calculated based on the d values x_j provided at the inputs and their associated weights ω_{ij} , generally using the simple formula:

$$\sigma_i = \sum_{j=1}^d \omega_{ji} x_j \quad (2.9)$$

Any architecture is conceivable depending on the task required of the network. Here, we are focused on multilayer perceptrons, where formal neurons are classified into three categories:

- **Input neurons** are used to transmit input data (both the training sample examples and future examples to classify). In this particular case, $\sigma_i = x_i$, where x_i is the component with index i of the data vector x .

- **Output neurons** are located at the end of the chain and provide the learning hypothesis. Each output neuron corresponds to a class.
- **Hidden neurons** are exclusively connected to other neurons and not to the network's inputs or outputs. They perform intermediate processing.

Perceptrons are built according to the feed-forward logic, meaning information flows only in one direction, from input neurons to hidden neurons and then to output neurons. They are thus organized in successive layers (input layer, hidden layers, and output layer), where each element is connected to all elements of the next layer and only to them. The input neurons are activated by each receiving a component of vector x (the input data). They calculate their output value (Equations (2.7) and (2.9)), and the result is then transmitted to the first hidden layer, which does the same, and so on until reaching the output layer. The result is then the class corresponding to the output neuron with the highest output value. The designer must define the number of hidden layers and neurons within these layers, as this will affect the complexity of the boundaries the network will identify.

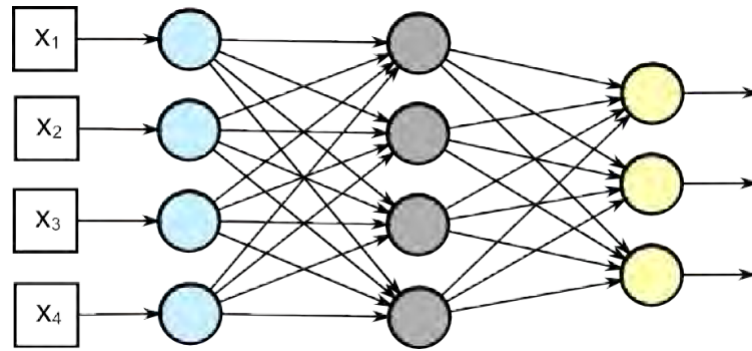


Figure 2.6: Example of a Perceptron with Hidden Layer

A simple example of such a network is illustrated in Figure 2.6. The inputs are represented by squares and the neurons by circles (blue for inputs, gray for hidden neurons, and yellow for outputs). This is a one-hidden-layer network for a three-class problem with four-dimensional data.

A neural network alone is not capable of learning. It must be complemented with a backpropagation mechanism of the error between the desired output and the calculated output, allowing it to adjust the weights of the connections, and thus learn the function it needs to perform [136]. The training sample data is passed sequentially through the network, which will adjust its weights with each pass. To enable the network to converge, this process must be repeated many times (typically around a hundred passes for each example). The weight adjustment is dictated by what is called the delta rule, which expresses the modification to be applied to the weight of the connection between neuron i and neuron j :

$$\Delta\omega_{ij} = \alpha\delta_j y_i \quad (2.10)$$

where y_i is the output of neuron i (one of the inputs to neuron j), α is a positive coefficient between 0 and 1, and δ_j is a value that depends on the error. For an output neuron,

it is expressed as follows (assuming the chosen function g is a sigmoid parameterized with $\lambda = 1$):

$$\delta_j = (u_j - y_j)y_j(1 - y_j) \quad (2.11)$$

where u_j is the desired output (corresponding to the label of the input example passed to the network), and y_j is the calculated output. The value of δ for the hidden neurons is obtained recursively:

$$\delta_j = y_j(1 - y_j) \sum_{k \in \text{dest}(j)} \delta_k \omega_{jk} \quad (2.12)$$

where $\text{dest}(j)$ is the set of neurons receiving the output from j . The calculation of ω comes directly from the derivative of the output function of the neurons. That is why this function must be differentiable and nonlinear. Furthermore, it should be noted that the weight adjustment of one layer is only possible if that of the lower layer has been performed. This adjustment method is thus called the backpropagation of the error gradient rule.

The learning of multilayer connectionist networks yields good results despite a certain slowness in the learning process. One of the open issues concerns the stopping criterion for learning since the network never completely stabilizes. The usual solution involves using a validation set of examples. Another problem arises from the initialization of weights, as certain initial conditions can lead to a local minimum.

Other types of networks are possible, including those with cycles. These are referred to as recurrent connectionist networks. However, such networks are difficult to understand, and finding a rule to adjust the weights for successful learning becomes challenging. Among those used in supervised mode are reservoir computing networks [137]. Discovered simultaneously under different names, namely Echo State Networks [138] and Liquid State Machines [139], the idea behind this type of network is to isolate the recurrent component (the one with cycles) into a separate part of the network, called the reservoir. Learning is then performed solely on a linear one-layer network whose inputs are neurons chosen from the reservoir. This approach is reminiscent of the idea of re-describing input data, as seen with SVMs (see 2.2.1). This approach is still relatively new and not thoroughly explored. However, certain characteristics necessary for reservoirs seem to emerge. They must be large enough and have sparse connections to exhibit rich dynamics. This means reservoirs with tens of thousands of neurons with a low connection rate (less than 20%) whose weights are randomly assigned. Nevertheless, it remains to be understood how to better adapt reservoirs to concrete learning situations.

This leads us to a crucial question regarding artificial neural networks: how to choose the architecture of a network? One answer comes from the use of genetic algorithms (see 2.2.1) with neuroevolution. This involves representing a neural network (both its topology and weights) in the form of a genotype and evolving it until an acceptable solution is reached. There are several techniques that differ mainly in their method of encoding networks into genes and the variant of genetic algorithm used, all of which share a very high cost [140].

Neural networks are still under extensive study, and while classic multilayer networks seem to have revealed most of their secrets, recurrent networks are still largely unexplored. In general, connectionist networks yield good results, but they are limited by the time required for learning and the challenges of implementing them on specific problems (e.g., choosing the topology, etc.). Nevertheless, they promote the idea that a collection of

simple yet judiciously connected entities can accomplish complex tasks. Finally, neural networks are often considered a particular case of Bayesian networks, which we will present in the next section [141].

Learning Bayesian Networks

Bayesian inference networks are probabilistic models that allow reasoning based on uncertain data. Formulated as directed acyclic graphs, they contain a representation of knowledge from which they can calculate conditional probabilities [142]. Each node corresponds to a random variable and contains a table of conditional probabilities, while each arc expresses a relationship of direct dependence.

Figure 2.10 presents an example from Ben-Gal [143] describing the situation of a person who may have a back injury (noted as B) and may experience pain (D). This injury could result from inadequate sports practice (S) or an uncomfortable chair at work (C). In the latter case, it can be assumed that a coworker also has a back injury (E). All these variables are binary and therefore take on the values true (T) or false (F). The graph shows the dependence relationships among these variables, from which two types of inferences can be calculated:

- **Top-down inference** when one seeks to calculate the conditional probabilities of a node based on its ancestry in the graph, also known as predictive support in English.
- **Bottom-up inference** when one seeks to calculate the conditional probabilities of a node based on its descendants in the graph, also known as diagnostic support in English.

These inferences are very costly to compute, and the problem is known to be NP-hard in general cases. However, there are efficient algorithms for particular cases, such as the message-passing algorithm [144] or the variable elimination algorithm [145], as well as others that focus on approximating the result using, for example, MCMC (Markov Chain Monte Carlo) methods [146].

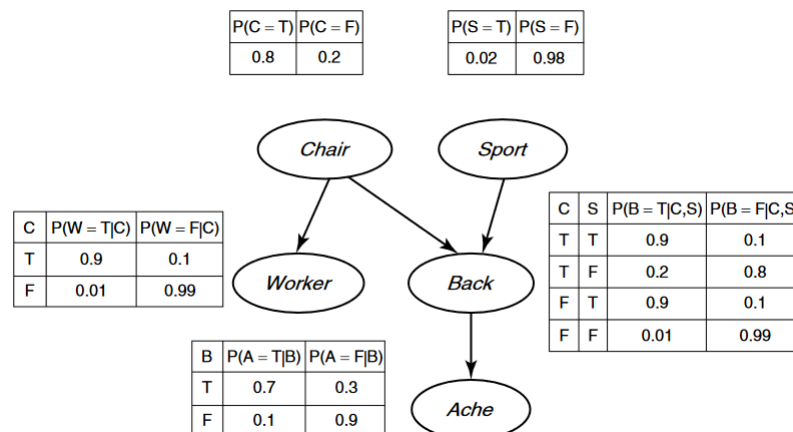


Figure 2.7: The backache BN example

What interests us here is how to infer a Bayesian network from a training sample. Several families of approaches can be distinguished based on two criteria: prior knowledge

of the graph's topology and consideration of partial observability of the data [147]. In the simplest case (where the structure is known in advance), the goal is to find the correct probability values in the tables from the examples. The most common techniques are based on the principle of maximum likelihood when all the data is observable, or on an expectation maximization algorithm if this is not the case.

Whether the data is fully accessible or not, learning the structure of the graph is extremely costly. Two approaches stand out. On one hand, there are methods that assign a score to each candidate graph to measure the compatibility between the dependencies represented in the network and those observed in the sample data. Search techniques in a possibility space are then employed to find the best candidate, such as those described by Elidan, Nachman, and Friedman [148].

The drawback of this method is that searching for the best candidate is an NP-complete problem. On the other hand, there are constraint-based methods, which are less commonly used. These methods start by constructing an undirected graph from dependencies identified through statistical tests on the training sample. The edges of this graph are then progressively directed by constraint propagation. Here, the main limitations are the explosion in the number of tests that need to be performed and the reliability of the statistical test when the sample does not contain enough examples. Heuristics are then employed to circumvent these difficulties [149].

Before summarizing this section on supervised learning, it now seems interesting to discuss a set of somewhat special techniques aimed at improving the performance of so-called "weak" programs by combining them.

Meta-learning

Meta-learning is the learning of both the choice between multiple learning programs and their combination [150]. This can involve different algorithms or several instances of the same algorithm that have benefited from different data samples. Various methods have been proposed, among which we briefly discuss here: two-stage learning, bagging, boosting, and finally cascade learning.

Voting among a set of classifiers is certainly the simplest way to combine them: the final decision for an input point is the weighted average of each learning program's decision. The question then becomes how to assign weight to each vote. This is the purpose of two-stage learning [151]. This technique uses a particular classifier (called a meta-classifier) tasked with learning the appropriate weight values for each base classifier's votes. It involves splitting the training sample S into two parts S_1 and S_2 and training the base classifiers only on S_1 before testing them on S_2 . The parameters of the meta-classifier are then learned based on a new sample formed from the results of the tests on S_2 , labeled by the desired actual output.

Bagging (short for bootstrap aggregation) utilizes multiple instances of the same algorithm. Each instance is trained on a random fraction of the training sample drawn with replacement. For a given input, each draw corresponds to a hypothesis, and the final hypothesis is simply their average [152].

Boosting encompasses techniques that produce accurate decisions from so-called weak learners (i.e., those whose performances are barely better than random guessing). The first algorithm proposed was the subset algorithm. An initial hypothesis is obtained from a subsample S_1 . A second hypothesis is learned from another subsample S_2 drawn from

$S - S_1$, in which half of the data points are misclassified by the first hypothesis. Finally, a third hypothesis is learned from a third subsample drawn from $S - S_1 - S_2$, where the first two hypotheses disagree. The final hypothesis is then the result of a vote among the three learned hypotheses [153]. This algorithm can be applied recursively, allowing for the use of nine subsets, then twenty-seven, and so on. AdaBoost is a probabilistic generalization of boosting by subsets. It is based on the idea of applying a probability distribution to the examples in the training sample according to the results of the previous hypotheses. These probabilities are then used to form the subsets [154]. There are several variations of this algorithm, such as BrownBoost [155], Gentle AdaBoost [156], and Multi-class [157].

In contrast to boosting and bagging, cascading learning places the base classifiers in series. An example to be classified is given to the first classifier, and the result is evaluated. If it is deemed reliable, the algorithm stops. Otherwise, the example is passed to the second classifier, and so on. The advantage of this method is that it optimizes computation time by placing the simplest classifiers first. The more costly classifiers will only be invoked in the case of difficult examples. However, this approach requires an evaluation function for each type of classifier present in the chain [158].

The idea behind meta-learning, which aims to achieve better performance than the individual programs by combining them wisely, is particularly interesting.

Overview of Supervised Learning

Supervised learning is a key machine learning paradigm where an algorithm learns from labeled training data to make predictions or decisions. In this approach, each training example consists of input features and the corresponding output label. The goal is to create a model that can accurately map inputs to outputs, enabling it to predict outcomes for unseen data. Supervised learning encompasses various algorithms, including regression techniques for continuous outcomes and classification methods for categorical outcomes. It is widely applied across numerous domains, including finance, healthcare, and marketing, where predictive accuracy is crucial. We will equip agents in our organization with various supervised learning techniques to enhance their ability to predict pandemics in healthcare. By leveraging algorithms trained on historical health data, these agents will analyze patterns and trends, allowing for timely predictions and informed decision-making. This approach aims to improve our response strategies and resource allocation during health crises.

2.2.2 Unsupervised Learning

Unsupervised learning methods aim to classify data or discover patterns without labeled guidance. These techniques identify clusters, correlations, or components within unstructured data and include:

1. **Semi-Supervised Learning:** Uses partially labeled data to improve learning. Co-learning is an example, where classifiers collaborate to label unlabeled data by validating each other's predictions.
2. **k-Means Clustering:** This common clustering algorithm partitions data into k clusters to minimize variance. Variants like k-means++ improve the algorithm's initialization for more optimal clusters.

3. **Component Analysis:** Techniques such as Principal Component Analysis (PCA) reduce dimensionality, seeking to represent data in an uncorrelated feature space. Multilinear subspace learning (MSL) extends PCA for high-dimensional data, though it is sensitive to initial conditions.
4. **Kohonen Maps (Self-Organizing Maps):** These neural networks generate simplified data representations by adjusting neuron weights to reflect data distribution, useful for high-dimensional data visualization.
5. **Hopfield Networks:** These networks function as associative memories, storing and recalling patterns based on partial inputs, and are particularly useful for noise filtering and shape recognition.
6. **Boltzmann Machines:** Stochastic versions of Hopfield networks, employing probabilistic principles, with Restricted Boltzmann Machines (RBMs) widely used in applications like image recognition. However, they are computationally intensive and unsuitable for real-time learning.

Evaluation

Unsupervised learning methods effectively classify and find associations in data without *a priori* knowledge. However, most operate offline, requiring retraining if new data is introduced. They offer insight into the structure of static datasets but fall short for real-time adaptive needs.

2.2.3 Reinforcement Learning

Reinforcement learning (RL) addresses scenarios where an agent learns optimal behaviors through feedback from the environment. RL balances exploration (trying new actions) with exploitation (choosing known rewarding actions). Key approaches include:

1. **Q-Learning:** An off-policy algorithm where actions are chosen based on the highest estimated reward. Q-learning is simple to implement but converges slowly since it requires repeated exploration of each state.
2. **SARSA:** A policy-driven variant of Q-learning that updates based on a predefined policy, adapting to the agent's consistent interaction with its environment.
3. **Learning Classifier Systems (LCS):** These systems combine reinforcement learning with genetic algorithms to refine action-selection rules. LCS are adaptive over time and well-suited for complex control environments.
4. **Case-Based Reasoning (CBR):** This problem-solving method reuses solutions from similar past cases. While CBR systems improve over time by accumulating cases, their performance relies heavily on prior data quality.

Evaluation

Reinforcement learning techniques are particularly effective for problems requiring real-time adaptability, as they continually refine behavior in response to feedback. The main limitations are the computational expense and the extensive cycles required for complex, real-world applications.

2.3 Conclusion

In this chapter, we explored the foundational aspects of multi-agent organizations and the role of artificial learning paradigms in enhancing their capabilities. We first established a framework for understanding multi-agent organizations, emphasizing how structuring, cooperation, and normative behaviors are essential for effective organizational modeling and management. The analysis included various types of organizations, examining both normative and non-normative models to highlight the diverse approaches to agent roles, group dynamics, and regulatory structures.

We also investigated artificial learning techniques—supervised, unsupervised, and reinforcement learning—as vital tools for augmenting multi-agent system functionality. Supervised learning was highlighted for its capacity to equip agents with prediction and classification abilities, critical for responsive and adaptive behavior. Unsupervised learning techniques were presented as means for agents to autonomously recognize patterns and structures in unstructured data, while reinforcement learning offers agents the ability to learn through interaction and feedback, optimizing their decision-making in dynamic environments.

Overall, this chapter provides a comprehensive view of how multi-agent organizations benefit from structured learning methodologies. These insights set a strong foundation for the practical applications of artificial learning paradigms in subsequent discussions, allowing for enhanced adaptability, openness, and robustness in multi-agent systems.

Part II

Methodology and Implementation

Organizational Modeling and Management Infrastructure: *Moise* & *JaCaMo*

Organizational modeling is fundamental in multi-agent systems (MAS) for structuring and managing agent interactions to achieve collective goals. The *Moise* organization modeling language and *JaCaMo* framework together offer a comprehensive approach for implementing and managing MAS, providing both structural and functional components for organizational design.

Moise, an Organizational Modeling Language (OML), focuses on defining structures, functional objectives, and norms within MAS, enabling organized interaction and cooperation among agents. *JaCaMo* complements *Moise* by offering a powerful management infrastructure that unifies agents, their environment, and organizational governance under one framework. The combination of *Moise*'s modeling capabilities and *JaCaMo*'s infrastructural support allows for the creation of adaptable, scalable MAS with complex behavior.

The study of organizational models presented in Chapter 2 allowed us to observe how organizations serve as a means of modeling using organizational modeling languages (OML) and their management through organizational management infrastructures (OMI).

Among the OMLs studied, we briefly introduced the [88] and *MoiseInst* [97] models, which form the foundation of our research. These models have the advantage of enabling a modular representation of organizations, facilitating model management.

3.1 *Moise* Meta-Model

We define a multi-agent organization as a collection of agents that interact according to a structure, cooperation schemes, and predefined norms to achieve specific objectives. An organizational modeling language (OML) is a language that enables the representation of organizational data, such as goals, structure, cooperation schemes, and norms. We also note that this representation is divided into multiple dimensions in organizational models. The OML *Moise* used in this thesis encompasses three dimensions: structural, functional, and normative.

- **Structural Dimension (*SS*):** Specifies the groups and roles that define the organization's structure.

- **Functional Dimension (*FS*):** Defines social schemes that specify cooperative functional activities within an organization. These social schemes consist of goals and missions.
- **Normative Dimension (*N*):** Establishes constraints in the form of obligations, permissions, and prohibitions between roles defined in the Structural Scheme (*SS*) and missions defined in the Functional Scheme (*FS*).

Thus, the organizational specification of an organization defined with the *Moise* OML is represented by the following triplet :

$$OS = \langle SS, FS, N \rangle \quad (3.1)$$

The OML *Moise* addresses several levels "individual, social, and collective" through its *OS*. Specifically, the roles specified in the structural dimension (*SS*) define an individual level relative to the agents. Moreover, in an *SS*, relationships can be specified between roles. These relationships define the social level that exists between agents who play roles within an organization. The groups to which roles belong represent the collective level of structuring [97].

In an *FS* , goals define the individual level of an organization's functioning, meaning the objectives that agents must achieve individually according to their roles. The goals are grouped into missions, which represent the social level, as a mission may be assigned to multiple agents who must then coordinate to accomplish it. Finally, the social schemes represent the collective functional organization of an organization, thus reflecting the collective level of interpretation of the *FS*. The normative dimension (*NS*) allows for defining the norms that establish the behaviors expected by the organization from its agents.

As with the *Moise+* and *MoiseInst* models, the dimensions of an organizational specification proposed can be represented in several formalisms, As shown in Figure 3.1

- A graphical formalism that offers a visual representation easily understandable and manipulable by organizational designers. The graphical representation elements of the structural and functional dimensions are presented in Figures 3.2 and 3.3.
- A BNF formalism for a more formal representation of the specifications and the relationships that exist between them, as well as among their elements [97].
- For programming OMI, an XML formalism and a logical formalism (NOPL [159]) have also been defined.

3.1.1 Structural Specification

The Structural Specification (*SS*) of the *Moise* Organizational Model Language (OML) defines the structural elements of an organization. The concept of role, introduced in the original *Moise+* [88], is used to express the individual level of an organization's structure. In other words, a role can be adopted by agents. The concept of group was also defined in *Moise+* to express the collective level within an organization's *SS*. Agents may belong

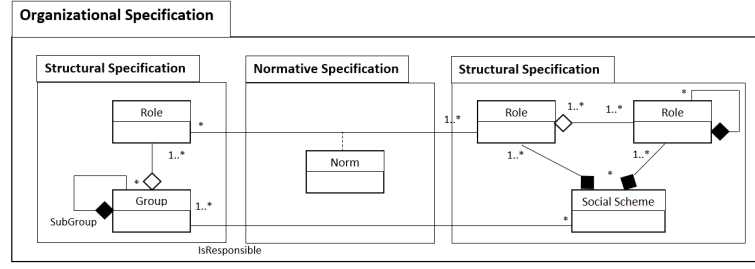


Figure 3.1: General Architecture of OMIs

to one or more collectives within the organizational structure. Each collective is defined by a group.

Thus, an SS is defined by Equation 3.2. It comprises the set of its roles, the set of its groups, and an inheritance relationship among roles.

$$SS = \langle R, Gr, \prec \rangle \quad (3.2)$$

- R : The set of roles in the SS , which includes abstract roles.
- Gr : The set of groups and sub-groups in the SS .
- $\prec$: The inheritance relationship among the roles in R .

In the following sections, we present the three levels "individual, social, and collective" of an SS .

Individual Level: "Role"

The roles defined in the Structural Specification of the *Moise* OML explicitly represent the individual level of agents within an organization's structure.

A role is the abstraction of an agent in the SS. It serves as an anchor point for an agent to a set of responsibilities and constraints that they must adhere to from the moment they agree to assume this role [97].

In an SS , some roles may be abstract. This type of role cannot be assigned to an agent within an organization. Generally, an inheritance relationship is specified between an abstract role and one or more non-abstract roles. Abstract roles are used to define and factorize common properties shared by several roles, which the non-abstract roles will inherit. Unlike an abstract role, a non-abstract role is a role that an agent can play within an organization.

Social Level: "Link"

The social level in an SS is expressed through links that can be defined between roles within a group.

A link is a relationship between two roles within a group. It is defined by a source role, a destination role, and a type [97].

Let L_{gr} denote the set of links in a group gr . A link belonging to L_{gr} is defined by the equation (3.3) below.

$$\forall l \in L_{gr}, l = \langle r_s, r_d, t \rangle \quad (3.3)$$

- r_s : is the source role of the link, $r_s \in R_{gr}$, where R_{gr} is the set of roles in gr and $R_{gr} \subset R$.
- r_d : is the destination role of the link, $r_d \in R_{gr}$.
- $t \in \{acq, com, aut\}$: is the type of the link.

The types of links between the roles of groups in a multi-agent system (MAS) defined below are the same as those in the *Moise+* [88] and *MoiseInst* [97] models.

- **Acquaintance link (acq):** This specifies that agents playing the source role r_s are allowed to represent agents playing the target role r_d .
- **Communication link (com):** This expresses that agents playing the source role r_s are permitted to communicate with agents playing the target role r_d .
- **Authority link (aut):** This last type of link allows agents playing the source role r_s to control agents playing the target role r_d .

The graphical representation of the links in a SS, along with its other elements, is presented in Figure 3.2.

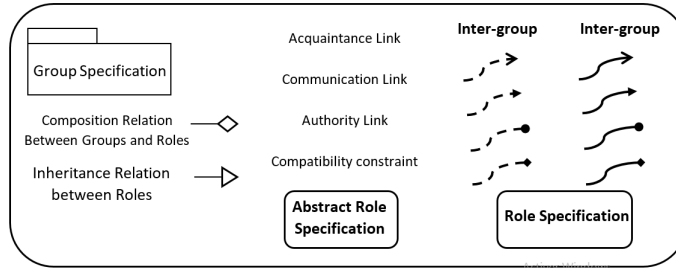


Figure 3.2: Graphical Representation of the SS

Collective Level: "Group"

The collective level of constraint in a structural specification is defined through the groups in the organizational specification. The concept of a group is proposed to express the framework within which agents can play roles.

A group is an abstraction of a collective of agents [97]. It is defined by a septuplet presented in Equation (3.4).

$$\forall gr \in Gr, gr = \langle R_{gr}, SG_{gr}, L_{gr}^{intra}, L_{gr}^{inter}, C_{gr}^{intra}, C_{gr}^{inter}, ng_{gr}, nr_{gr} \rangle \quad (3.4)$$

- R_{gr} : The set of roles for gr , $R_{gr} \subset R$
- SG_{gr} : The set of subgroups of gr .

- $L_{gr}^{intra}, L_{gr}^{inter}$: respectively, the set of links as defined above, categorized into intra-group and inter-group links defined for the group gr . We denote $L_{gr} = L_{gr}^{intra} \cup L_{gr}^{inter}$.
- $C_{gr}^{intra}, C_{gr}^{inter}$: respectively, the set of *intra-group* and *inter-group* compatibility constraints defined for the group gr .
- ng_{gr} : the function that associates each subgroup defined for the group $gr' \in SG_{gr}$ its cardinality in terms of the minimum and maximum number of instances of gr' that can be created within an organization.
- nr_{gr} : the function that assigns to each role r of the group gr its cardinality in terms of the minimum and maximum number of instances of r that can exist within the group gr in an organization.

Subgroup Relation

- A group gr' is a subgroup of a group gr if $gr' \in SG_{gr}$; we say that there exists a composition link from gr to gr' . The graphical representation of a composition link is shown in Figure 3.2.
- If a group is not a subgroup of any other group, it represents the root group of the SS (structural specification).
- A group can only be a subgroup of a single group.

In the following sections, we will refer to a group instance to designate a collective entity within an organization, created from a group specification defined in the SS. Agents can join and adopt roles within a group instance.

Intra- and Inter-Group Links

Two categories of links exist between the roles within groups, which define the scope of the link relative to the group gr in which they are defined:

- **Intra-group links (noted as L_{gr}^{intra})** are links whose scope is limited to an instance of gr and to the instances of its subgroups. Thus, for any $l \in L_{gr}^{intra}, l = \langle rs, rd; type \rangle$; any agent playing the role rs in an instance of gr has a link of type $type$ with all agents playing the role rd in the same instance or in instances of subgroups $gr' \in SG_{gr}$.
- **Inter-group links (noted as L_{gr}^{inter})** are links whose scope extends to any instance of gr . Thus, for any $l \in L_{gr}^{inter}, l = \langle rs, rd; type \rangle$; any agent playing the role rs in an instance of gr has a link of type $type$ with all agents playing the role rd , regardless of the specific instance of gr in which the agents are playing the role rd .

Constraints

Since multiple roles can be defined within the same group, the concepts of compatibility constraints and cardinality constraints were introduced in the OML *Moise+* framework to enforce constraints on agents at the collective level of the structural specification (SS), meaning within groups defined in a structural specification.

1. **A compatibility constraint** is defined between two roles and allows for explicitly specifying the possibility for an agent to adopt both compatible roles. This is represented by Equation 3.5. A compatibility constraint also has a scope that can be either intra-group or inter-group. Therefore, we denote the set of all intra- and inter-group constraints defined for a group $gr \in Gr$ as $C = C_{gr}^{intra} \cup C_{gr}^{inter}$. An element of C is defined as follows:

$$\forall c \in C, c = rs \bowtie rd \quad (3.5)$$

- **Intra-Group Compatibility:** $\forall c \in C_{gr}^{intra}$, all agents playing the source role rs in an instance of gr can also play the role rd in the same instance or in one of its subgroup instances.
- **Inter-Group Compatibility:** $\forall c \in C_{gr}^{inter}$, all agents playing the source role rs in an instance of gr can also play the role rd in any instance of gr .

The inheritance relationship between roles also affects the compatibility constraints. Thus, we have:

$$\begin{aligned} rs \bowtie rd \wedge rs \sqsubset rs' &\implies rs' \bowtie rd \\ rs \bowtie rd \wedge rd \sqsubset rd' &\implies rs \bowtie rd' \end{aligned} \quad (3.6)$$

2. **Cardinality Constraint:** This constraint allows us to express the minimum and maximum number of instances of subgroups within a group gr and the minimum and maximum number of instances of each role within a group gr .

The cardinality constraint on the subgroups is defined by the function ng_{gr} , represented by Equation (3.7). Meanwhile, the function nr_{gr} , represented by Equation (3.8), defines the cardinality constraint on the roles.

$$\begin{aligned} ng_{gr} : SG_{gr} &\rightarrow \mathbb{N} \times \mathbb{N} \\ gr &\mapsto (min, max) \end{aligned} \quad (3.7)$$

$$\begin{aligned} nr_{gr} : R_{gr} &\rightarrow \mathbb{N} \times \mathbb{N} \\ r &\mapsto (min, max) \end{aligned} \quad (3.8)$$

3.1.2 Functional Specification

The functional specification (FS) defines a set of social schemas considered as collective objectives to be achieved by an organization [97]. More specifically, it is within the FS of the OML *Moise* that the collective objectives of an organization are specified, the decomposition of these objectives into goals that agents must achieve, the ordering of goals that defines the coordination and cooperation among agents through *social schemas*, and the grouping of goals into missions that agents can undertake. The concepts of goals, missions, and social schemas thus allow for the definition of the individual, social, and collective levels of the functional dimension of an organization.

Equation (3.9) represents the definition of a functional specification, which consists of two elements: the set S of social schemas and a function Pr that defines a preference order for executing the missions defined in the social schemas of S .

$$FS = \langle R_{gr}S, Pr \rangle \quad (3.9)$$

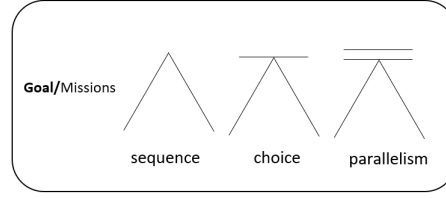


Figure 3.3: Graphical Representation of the FS

Collective Level: "Social Schema"

A social schema is the structure of an organization's overarching objective, broken down into goals, structured into plans, and grouped into missions [97]. A social schema thus defines the process that agents must follow to achieve a collective goal or objective. It is represented by a tree, called a plan, whose root is the collective goal to be achieved, with intermediate nodes and leaves representing the breakdown of the collective goal into sub-goals.

The formal representation of a social schema is given by the following equation:

$$\forall s \in S, s = \langle G_s, M_s, P_s, RespGr_s, nm_s, ng_s, mo_s \rangle \quad (3.10)$$

where:

- G_s : The set of goals of the social schema. The concept of a goal is presented in section 5.4.3.
- M_s : The set of missions of the social schema. The concept of a mission is presented in section 5.4.2.
- P_s : The set of plans of the social schema.
- $RespGr_s$: The set of groups that can be responsible for the social schema.
- $nm_s \& ng_s$: Functions that define cardinality constraints on the missions and goals of the social schema.
- mo_s : The function that, for each mission m in M_s , associates a subset of goals SG_m from G_s , where $SG_m \subset G_s$.

Plans of a Social Schema

Each element in the set of plans P_s of a social schema is defined as follows:

$$\forall p \in P, p = \langle head, op, G_p \rangle \quad (3.11)$$

- $head$: The root goal of the plan, with $head \in G_s$
- G_p : The set of goals of the plan.
- op : The operator of the plan, with $op \in \{;, |, \parallel\}$. The graphical representation of these operators is shown in Figure 3.3.

1. The operator $;$ indicates that the goals in G_p must be executed sequentially.
2. The operator $|$ indicates that the goals in G_p should be selected for execution. This choice can be made according to the preferences defined in P_r .
3. The operator $\parallel$ indicates that the goals in G_p must be executed in parallel.

Responsible Group(s) of a Social Schema

To specify the type of group from the structural specification (SS) whose agents, playing roles within it, can participate in achieving the goals of a social schema, we have enhanced each social schema $s \in S$ with the concept of a responsible group for a social schema (*ResponsibleGroup*). Several groups can be responsible for the same social schema. Any agent committed to accomplishing a goal or mission of s must be a member of at least one of the responsible groups for s . Thus, the set $RespGr_s$ of responsible groups for a social schema s is calculated by the function *respGroup* defined as follows:

$$respGroup : S \rightarrow P(Gr) \quad (3.12)$$

Cardinality Constraints

Two types of cardinality constraints are defined in a social schema. They are expressed by the functions *nms* and *ngs* presented below.

1. **Mission Cardinality Constraint:** This specifies, for each mission m in M_s , the minimum and maximum number of agents who can commit to m within the social schema. It is defined by the function *nms*.

$$\begin{aligned} nms : M_s &\rightarrow \mathbb{N} \times \mathbb{N} \\ m &\mapsto (min, max) \end{aligned} \quad (3.13)$$

2. **Goal Cardinality Constraint:** This defines, for each goal g in G_s , the minimum and maximum number of agents required to achieve goal g within the social schema according to the function *ngs*.

$$\begin{aligned} ngs : G_s &\rightarrow \mathbb{N} \times \mathbb{N} \\ g &\mapsto (min, max) \end{aligned} \quad (3.14)$$

A goal g will only be considered as effectively achieved if the number of agents who have validated its achievement is at least equal to the minimum *min* defined by *ngs*(g).

Social Level: "Mission"

A mission is a set of coherent goals belonging to the same social schema, to which an agent can commit [97]. As introduced in Section 3.1, a mission represents the social level of interpreting an FS (Functional System) in the sense that multiple agents can commit to a mission and therefore must coordinate their activities to accomplish the mission as effectively as possible. The specification of a mission in *Moise* has been enhanced compared to that in the OML *Moise* + and *MoiseInst*, and it includes sets of adoption and abandonment requirements as presented by the equation below.

$$\forall m \in M_s, m = \langle AdoptReq_m, LeaveReq_m \rangle \quad (3.15)$$

- $AdoptReq_m$: the set of mission adoption requirements.
- $LeaveReq_m$: the set of mission abandonment requirements.

Individual Level: "Goal"

Within a social schema, a goal is an end state that constitutes an objective toward which an organization aspires. A goal can be defined by a plan that breaks it down into sub-goals [97].

In the *Moise* model, a goal is also interpreted as the individual level for the FS (Functional System). Goals are achieved or fulfilled individually by agents. However, according to the cardinality constraints defined for the goals of a social schema, some goals may require multiple agents to accomplish them. In such cases, each agent involved in achieving the goal must individually contribute to its realization and to validating that realization.

Equation (3.16) presents the specification of a goal.

$$\forall g \in G, g = \langle AdoptReq_g, LeaveReq_g, ttf \rangle \quad (3.16)$$

- $AdoptReq_g$: and $LeaveReq_g$: are, respectively, the sets of adoption and abandonment requirements for the goal.
- ttf : is the time duration defined for achieving the goal.

We can note that, as in the specification of a role presented in Section 3.1.1, the specification of a mission and that of a goal allow for the explicit definition of adoption and abandonment requirements: $AdoptReq_m$: and $LeaveReq_m$: for a mission, and $AdoptReq_g$: and $LeaveReq_g$: for a goal. These requirements are verified when an agent wishes to commit to achieving a goal or multiple goals within a mission, and when an agent wishes to abandon a mission or a goal they had committed .

3.1.3 Normative Specification

The Normative Specification (NS) is the set of norms that establish relationships among roles in the SS (Structural Specification), missions in the FS (Functional Specification), and a deontic operator.

The NS in the current version of the *Moise* model results from the developments in the definition and representation of norms in the *Moise+* and *MoiseInst* models.

Regardless of the model, the NS represents the dimension of an organization that links the other dimensions of an OS (Organizational Specification). It is therefore defined as follows:

$$NS = \langle N \rangle \quad (3.17)$$

Norm

A norm specifies the permissions, obligations, and prohibitions of the various roles in a structural specification with regard to the missions in a functional specification.

A norm thus defines the expected behaviors for an agent playing a particular role within an organization. In *Moise+*, it consists of a simple link between roles in an SS (Structural Specification) and missions in an FS (Functional Specification) through a deontic operator. The *MoiseInst* version is more expressive than *Moise+* because it allows for specifying an activation condition for the norm, an application context, an associated sanction and reward, and a validity period for the norm.

The new version, represented by Equation (3.18), is derived from the normative formalism called the Normative Organization Programming Language (NOPL) [159]. The objective of this formalism is to simplify the representation of a norm as proposed in *MoiseInst* so that the expression of a reward or a sanction is itself a norm, whose activation depends on a condition, and in which the other elements of the norm specify, among other things, the role of the agent responsible for managing the reward or sanction and the validity period of the reward or sanction norm.

$$\forall n \in N, n = \langle condition, role, type, mission, ttf \rangle \quad (3.18)$$

- *condition* : as the name suggests, this is a condition that defines the activation of the norm, i.e., its transition to the active state (see Figure 3.4).
- *role* : denotes the role associated with the norm.
- *type* : the type of the norm, corresponding to one of the following deontic operations: obligation, permission, or forbidden.
- *mission* : an element of the FS (Functional Specification) representing the mission to which the deontic operator applies.
- *ttf(TimeToFulfil)* : the duration after which an active norm must transition to either the fulfilled state or the unfulfilled state (see Figure 3.4).

Lifecycle of a Norm

The lifecycle of a norm n in the NS (Normative Specification) is depicted in Figure 3.4. According to the specification of a norm presented in the previous section, a norm has an activation condition denoted as ρ . When this condition is met, the norm transitions to the active state, noted as active. In this state, the agent or agents concerned by this norm can carry out the associated mission m .

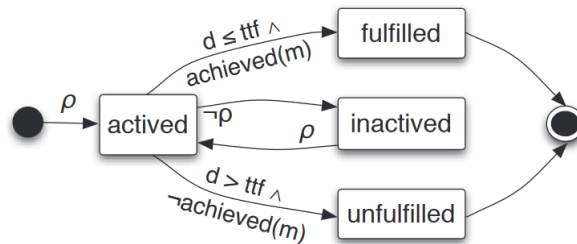


Figure 3.4: Lifecycle of a Norm

We also noted that the ttf duration of a norm defines the time within which the norm, specifically the associated mission, must be fulfilled. If the mission is accomplished within a duration d that is less than or equal to ttf , the norm transitions to the fulfilled state. Otherwise, the norm transitions to the unfulfilled state. These two outcomes are represented in Figure 3.4 by the transitions $(d \leq ttf \wedge achieved(m))$ and $(d > ttf \wedge \neg achieved(m))$.

After a norm is activated, it may happen that the activation condition ρ changes and is no longer valid. In this case, the norm transitions to the inactive state (see transition $\neg\rho$ in Figure 3.4).

Thus, throughout the lifecycle of a norm, three factors influence its state changes: the activation condition, the validity duration, and the accomplishment of the mission associated with the norm.

3.1.4 Summary of *Moise* Meta-Model

the Organization Modeling Language (OML) *Moise* provides a structured approach to defining and organizing multi-agent systems through its multi-level framework that encompasses the individual, social, and collective dimensions. The Structural Specification (SS) establishes the fundamental roles, groups, and relationships among agents, enabling a clear structure for individual responsibilities, social interactions, and collective organization. The Functional Specification (FS) extends this by defining the goals, missions, and social schemas that guide agents' activities, fostering coordination and goal alignment across different organizational levels. Finally, the Normative Specification (NS) introduces norms that govern agents' behaviors, ensuring adherence to organizational standards and facilitating a controlled, predictable system dynamic.

Together, these specifications present a comprehensive model for multi-agent organization, combining flexibility in agent roles with structured frameworks for interaction and goal achievement. By leveraging these three dimensions, *Moise* supports complex, adaptable, and scalable organizational modeling, making it a robust tool for designing sophisticated multi-agent systems.

3.2 Organization Management Infrastructure : *JaCaMo*

JaCaMo is a powerful framework purpose-built for developing complex, distributed, and self-adaptive multi-agent systems (MAS). Its strength lies in its unique integrated approach, combining the traditionally separate concerns of agents, their environment, and their organization into a unified programming model called Multi-Agent Oriented Programming (MAOP) [1]. This holistic perspective differentiates *JaCaMo* from traditional MAS frameworks that often prioritize agent programming while treating the environment and organization as secondary aspects. By elevating these dimensions to first-class entities, *JaCaMo* enables developers to model and implement sophisticated MAS applications with a focus on autonomy, coordination, adaptation, and openness [160].

The *JaCaMo* framework is built upon the solid foundation of three well-established technologies [161]

- **Jason** [56] A widely used platform for agent programming based on the Belief-Desire-Intention (BDI) architecture. Jason provides a declarative agent programming lan-

guage (AgentSpeak) and robust tools for managing agent beliefs, goals, plans, and practical reasoning.

- **CARTAgO** [162] A framework specifically designed for environment programming within MAS. CARTAgO introduces the concept of artifacts as first-class abstractions representing resources, tools, and shareable entities within the environment, facilitating agent interaction and environmental dynamics.
- **Moise** [163] A platform for organization programming, offering abstractions for defining groups, roles, norms, and organizational goals. *Moise* enables the creation and management of complex social structures within the MAS, providing a powerful means for coordinating and regulating agent behavior.

The seamless integration of Jason, CARTAgO, and *Moise* within *JaCaMo* provides a cohesive environment for developing MAS applications where intelligent agents, situated in a dynamic environment and governed by organizational structures (shown in figure 3.5), can effectively collaborate and achieve complex goals. This multi-dimensional perspective allows for a more natural and intuitive representation of intricate real-world scenarios, facilitating the development of robust solutions [164].

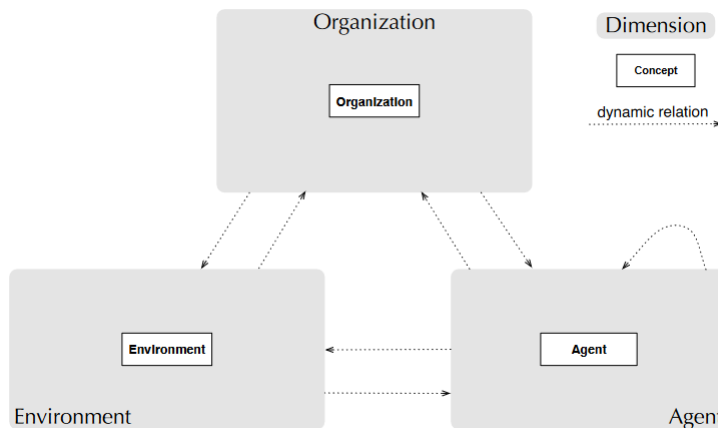


Figure 3.5: Dimensions of Multi-Agent Oriented Programming [1].

Key features of JaCaMo that streamline MAS development include:

- **Artifact-based Programming:** Leveraging CARTAgO’s artifact model enables modular design, dynamic environment modifications, and seamless integration with external resources [162].
- **Organization Management:** *Moise*’s integration offers built-in mechanisms for defining and managing organizations, groups, roles, norms, and organizational goals, empowering developers to model complex social interactions and control agent behavior within these structures [163].
- **Distributed Deployment:** *JaCaMo* readily supports distributed deployments, enabling the construction of scalable MAS applications across multiple machines, particularly through integration with the JADE agent platform [161].

- **Integration with Other Technologies:** *JaCaMo*’s flexible architecture facilitates integration with external libraries, frameworks, and platforms, broadening its applicability to diverse domains, including web services and robotics middleware [1].

JaCaMo’s integrated and multi-dimensional approach distinguishes it as a compelling framework for a wide spectrum of MAS applications, ranging from academic research and simulations to complex real-world systems [165]. By offering dedicated abstractions for agents, environments, and organizations, *JaCaMo* empowers developers to build robust, adaptable, and scalable MAS solutions, addressing the core challenges of modern software engineering.

3.2.1 Integrated View: The Synergy of Agents, Environments, and Organizations

The true power of Multi-Agent Oriented Programming (MAOP) with *JaCaMo* emerges when its three dimensions – agent, environment, and organization – are integrated to create a complete multi-agent system. This system isn’t merely a collection of isolated components but a dynamic, interacting network where individual agent autonomy is interwoven with environmental constraints and organizational governance. Figure 3.6 visually captures this interconnectedness, showcasing the dynamic relationships that bind the dimensions together.

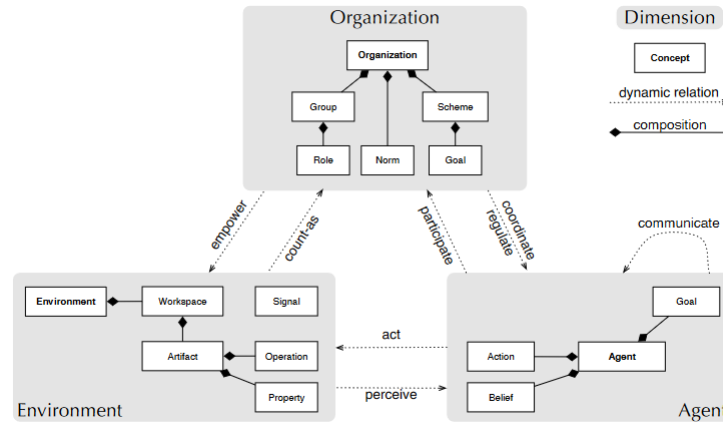


Figure 3.6: Dynamic relations among instances of concepts of the MAOP dimensions

Interaction as the Cornerstone:

Interaction lies at the heart of MAOP. Agents don’t exist in a vacuum; they continuously communicate with each other, exchanging information and coordinating their actions through speech acts and agent communication languages (ACLs). Simultaneously, they act upon and perceive their environment, mediated by artifacts. These interactions drive the system’s dynamics and shape its evolution.

Bridging the Individual and the Collective:

The organization dimension adds a crucial layer of structure and control. Agents participate in organizational schemes by adopting roles and committing to missions. Organizations coordinate agent activities by assigning goals and managing dependencies between tasks. Furthermore, organizations regulate agent behavior through norms, ensuring that individual actions align with collective objectives.

The Dynamic Feedback Loop:

The interactions between the dimensions create a dynamic feedback loop. Agent actions, mediated by the environment, can trigger changes in the organizational state. Conversely, organizational structures and norms influence agent decisions and actions, shaping how they interact with the environment. This continuous feedback loop allows the system to adapt to changing circumstances and achieve its goals in a flexible and robust manner.

3.2.2 The Agent Dimension

At the heart of the agent dimension lies the concept of an agent, an autonomous entity capable of independent action. Figure 3.7 illustrates the fundamental components that drive agent behavior:

- **Beliefs:** An agent's internal representation of its knowledge about the world. These beliefs, expressed as logical statements, can be incomplete or inaccurate, reflecting the uncertainty often present in complex environments.
- **Goals:** Represent desired states of the world that the agent aims to achieve. Goals motivate agent actions and provide direction to their behavior.
- **Actions:** The means by which agents interact with the environment and other agents. Actions can be physical (e.g., moving a robot arm) or communicative (e.g., sending a message). The figure highlights communication as a crucial action, enabling agent interaction and coordination.

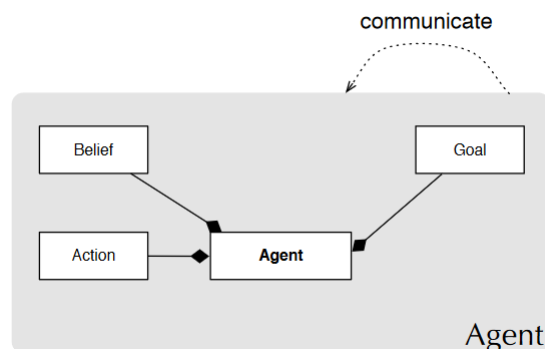


Figure 3.7: Main concepts in the agent dimension

Figure 3.8 expands upon these core components, showing the full range of concepts within the agent dimension:

- **Events:** Triggered by changes in an agent's beliefs or goals. Events drive the reactive and proactive behavior of agents.
- **Plans:** Recipes for actions that prescribe how an agent should behave in response to specific events. Plans consist of a trigger (the event that activates the plan), a context (conditions that must be met for the plan to be applicable), and a body (the sequence of actions to be performed).
- **Internal Actions:** Actions performed within the agent's internal reasoning cycle, such as updating beliefs or dropping goals.
- **External Actions:** Actions that affect the external environment, often involving interaction with artifacts.
- **Communicative Actions:** Actions that involve sending messages to other agents, typically using performative verbs to convey intent.

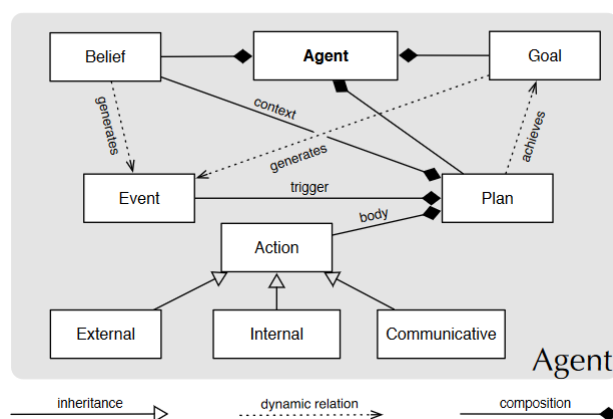


Figure 3.8: Concepts in the agent dimension

3.2.3 The Environment Dimension:

The environment dimension in MAOP provides the arena where agents act and interact, offering a structured way to represent resources, tools, and shared spaces. Unlike agents, environment entities are not autonomous; they provide functionality but don't initiate actions on their own.

Figure 3.9 shows the central concepts within the environment dimension:

- **Environment:** The overall space where agents and artifacts exist. It provides the context for agent actions and interactions.
- **Workspace:** A logical region within the environment. Workspaces can represent physical locations (e.g., rooms) or conceptual spaces (e.g., a shared database). They provide a way to structure the environment and manage the scope of interactions.
- **Artifact:** Represents a resource or tool within a workspace. Artifacts offer operations (actions agents can perform) and observable properties (attributes agents can perceive).

- **Signal:** An event emitted by an artifact, allowing for asynchronous communication with observing agents.

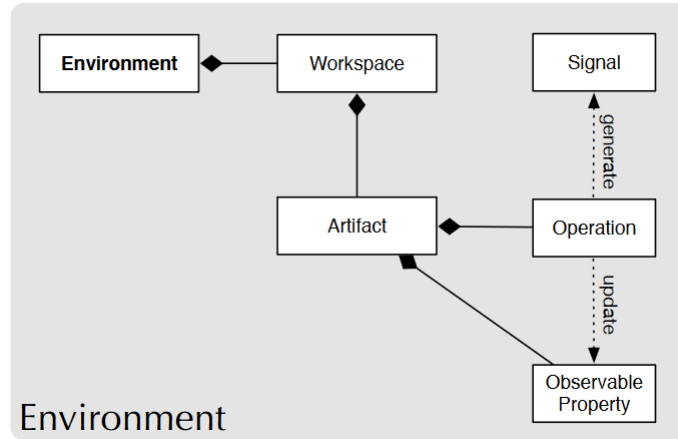


Figure 3.9: Main concepts in the environment dimension.

Figure 3.10 illustrates the relationship between environment concepts and the agent dimension:

- **Actions:** Agents interact with the environment through actions, which are mapped to artifact operations. This allows agents to affect the state of the environment.
- **Beliefs:** Agents gain beliefs about the environment by perceiving artifact observable properties. This allows agents to update their knowledge about the state of the world.
- **Events:** Changes in the environment, such as updates to observable properties or the emission of signals, generate events that trigger agent reactions.

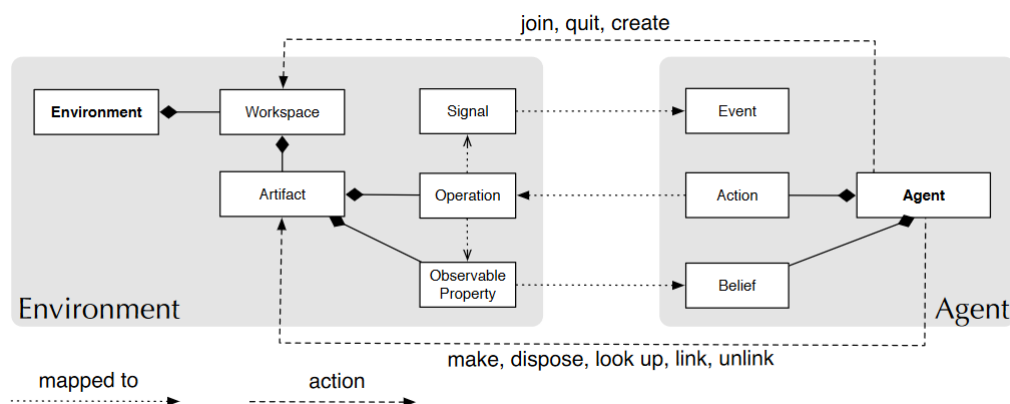


Figure 3.10: Main concepts in the environment dimension.

Key Features and Benefits:

- **Modularity and Encapsulation:** Artifacts promote modularity by encapsulating environment functionality.
- **Dynamic Extensibility:** Agents can create and dispose of artifacts during runtime, allowing for dynamic environments.
- **Scalability and Distribution:** Distributed workspaces enable the creation of scalable MAS applications.
- **Integration:** Artifacts can be used to integrate with external systems and resources.

The environment dimension in *JaCaMo* provides a rich set of tools and abstractions for modeling and managing the shared context in which agents operate. By understanding these concepts and utilizing the features provided by *JaCaMo*, developers can create dynamic, flexible, and scalable MAS applications. The figures mentioned above are essential for visualizing these concepts and their relationships. Always refer to the original text for comprehensive explanations and code examples.

3.2.4 The Organization Dimension

The organization dimension in *JaCaMo* provides the high-level structures and mechanisms for governing agent interactions and coordinating their behavior within a multi-agent system. It enables the creation of structured groups, the definition of roles and norms, and the specification of collective goals.

Figure 3.11 illustrates the core concepts of the organization dimension:

- **Organization:** The overarching structure that governs agent interactions and defines the rules of engagement within the system.
- **Group:** A collection of agents with defined relationships, forming a social unit within the organization.
- **Role:** A specific function or behavior associated with agents within a group.
- **Norm:** A rule governing agent behavior, including obligations, permissions, and prohibitions.
- **Norm:** A structured plan that defines how organizational goals are decomposed into subgoals and assigned to agents.
- **Mission:** A set of goals assigned to an individual agent within a scheme.
- **Goal:** A desired state of the world that the organization or an agent within the organization aims to achieve.

The organization dimension comprises three interconnected specifications:

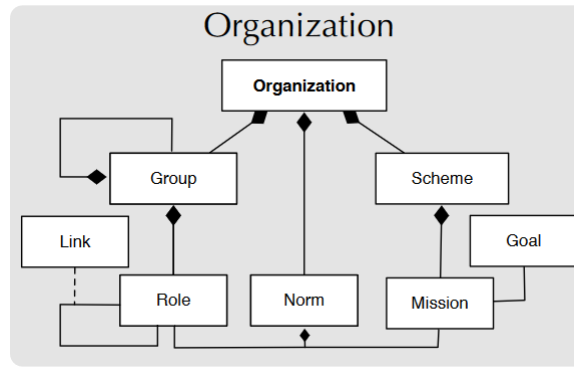


Figure 3.11: Main concepts in the organization dimension.

- **Structural Specification (Figure 3.12):** Defines the structure of the organization in terms of groups, roles, and links between roles. This specification determines the social structure of the MAS. Links define the types of interactions allowed between roles (e.g., communication, authority). Group Formation Constraints further refine organizational structure through role compatibility (which roles can be played together by an agent) and cardinalities (number of agents in a role).
- **Functional Specification (Figure 3.13):** Specifies the collective goals of the organization and how they are decomposed into individual agent missions through schemes. This specification determines the tasks and workflow of the MAS. Goals represent desired outcomes. Missions assign responsibilities to individual agents. The hierarchical structure of goals, plans, missions, and schemes represents the overall system functionality.
- **Normative Specification (Figure 3.14)** Defines the norms that govern agent behavior within the organization. This specification regulates the interactions and ensures compliance with organizational rules. Norms typically include a deontic modality (obligation or permission), a role (the norm bearer), and a mission (the scope of the norm).

Key Aspects & Benefits:

- **Coordination and Regulation:** Organizations provide mechanisms for coordinating agent actions and regulating their behavior.
- **Flexibility and Adaptation:** Dynamic role adoption and organization reorganization allow the system to adapt to changing circumstances.
- **Decentralized Control:** Organizations distribute control among agents, avoiding single points of failure.
- **Transparency and Reasoning:** Agents can reason about organizational specifications to understand their roles and responsibilities.

```
1 <structural-specification>
2
3 <role-definitions>
4   <role id="role0"/>
5   <role id="role1"> <extends role="role0"/> </role>
6   <role id="role2"> <extends role="role0"/> </role>
7 </role-definitions>
8
9 <group-specification id="group1">
10  <roles>
11    <role id="role1" min="1" max="2"/>
12    <role id="role2" min="1" max="1"/>
13  </roles>
14
15  <links>
16    <link from="role1" to="role2" type="authority"
17          scope="intra-group" bi-dir="false" />
18    <link from="role0" to="role0" type="communication"
19          scope="intra-group" bi-dir="true" />
20  </links>
21  <formation-constraints>
22    <compatibility from="role1" to="role2" bi-dir="true"/>
23  </formation-constraints>
24 </group-specification>
25 </structural-specification>
```

Figure 3.12: Example of a simple structural specification

```
1 <functional-specification>
2   <scheme id="scheme1">
3     <goal id="goal1" ds="description of goal1">
4       <plan operator="sequence">
5         <goal id="goal2" ttf="20 minutes" ds="description of goal2"/>
6         <goal id="goal3" ds="description of goal3"/>
7         <goal id="goal4" ds="description of goal4"/>
8       </plan>
9     </goal>
10
11     <mission id="mission1" min="1" max="2">
12       <goal id="goal2" />
13       <goal id="goal4" />
14     </mission>
15     <mission id="mission2" min="1" max="1">
16       <goal id="goal3"/>
17     </mission>
18   </scheme>
19 </functional-specification>
```

Figure 3.13: Example of a simple functional specification.

```
<normative-specification>
  <!-- the norms of the application -->
  <norm id="norm1" type="obligation"
        role="role1" mission="mission1"/>
  <norm id="norm2" type="permission"
        role="role2" mission="mission2"/>
</normative-specification>
```

Figure 3.14: Example of a simple functional specification.

The organization dimension in *JaCaMo* empowers developers to model and manage complex interactions and coordination patterns within MAS. By understanding these concepts and leveraging the visualizations provided by the figures, developers can create more robust, adaptable, and organized multi-agent systems. For detailed explanations, code examples, and a deeper understanding of organizational dynamics within *JaCaMo*, refer to [1].

3.2.5 Summary of *JaCaMo* OMI

JaCaMo stands as a significant advancement in the field of multi-agent systems development. By unifying the agent, environment, and organization dimensions within a single, cohesive framework, *JaCaMo* empowers developers to tackle the complexities of building truly robust, adaptive, and scalable MAS applications. Its integrated approach, based on the principles of Multi-Agent Oriented Programming (MAOP), promotes a more holistic understanding of system design, facilitating the development of systems that can effectively respond to dynamic circumstances and achieve collective goals in a decentralized and distributed manner.

The strength of *JaCaMo* lies not only in its conceptual underpinnings but also in its practical implementation. By leveraging the mature technologies of Jason, CArTAgO, and *Moise*, *JaCaMo* offers a robust and feature-rich environment for building real-world MAS applications. Features like artifact-based programming, organizational management, distributed deployment, and seamless integration with other technologies significantly reduce development time and effort, allowing developers to focus on the core logic of their applications. Moreover, the ability to reason about organizations and dynamically adapt to changing conditions provides a powerful mechanism for creating self-adaptive systems capable of autonomous operation in complex environments.

While *JaCaMo* has demonstrated its effectiveness in various domains, ongoing research and development continue to enhance its capabilities and broaden its applicability. Exploring new agent architectures, refining organization semantics, and improving integration with other AI techniques like machine learning and argumentation are just some of the exciting avenues being pursued. Furthermore, the growing community of *JaCaMo* users and developers contributes to a rich ecosystem of tools, libraries, and best practices, further solidifying its position as a leading MAS framework.

JaCaMo offers a compelling paradigm for engineering complex software systems. Its integrated, multi-dimensional approach, combined with its practical features and active community, makes it a valuable tool for both researchers and practitioners seeking to build the next generation of intelligent, distributed, and self-adaptive systems. *JaCaMo* is not just a framework; it represents a shift towards a more holistic and powerful way of thinking about and designing complex systems, paving the way for innovative solutions in a wide range of domains.

3.3 Conclusion

In this chapter, we have explored the integration of the *Moise* organization modeling language and the *JaCaMo* framework to design and manage multi-agent systems (MAS). Through the systematic analysis and implementation of *Moise*'s structural, functional, and

normative dimensions within *JaCaMo*’s agent-oriented environment, we demonstrated a robust, adaptive approach to MAS development that balances autonomy, cooperation, and rule-based behavior.

Our work contributes to the MAS field in several significant ways. First, we demonstrated that *Moise*’s layered approach to organizational modeling provides a clear and modular structure that simplifies the definition and management of roles, groups, goals, and norms. By embedding *Moise* within *JaCaMo*’s integrated environment, we also showed that MAS can be designed to adapt dynamically to real-world constraints, facilitating the creation of scalable and resilient systems.

Another key contribution is our detailed examination of how *JaCaMo*’s agent, environment, and organizational dimensions enable seamless coordination within MAS. By aligning *Moise*’s organizational rules with *JaCaMo*’s multi-agent infrastructure, we ensured that agents can autonomously interact within structured, resource-sharing environments. This fusion not only promotes robust communication and cooperation but also allows agents to flexibly respond to environmental changes and organizational shifts, making this approach highly suited for complex applications, including pandemic prediction, educational scheduling, and adaptive planning.

Predicting Covid-19 Infections

4.1 Introduction

This chapter presents the first contribution of multi-agent organization in the healthcare domain: a strategy for designing and implementing a Multi-Agent System (MAS) based on organizational paradigms. The proposed system introduces a healthcare-oriented approach that uses the Internet of Medical Things (IoMT) to support public health authorities in predicting COVID-19 infections. By leveraging autonomous agents, the system manages dynamic data from various sources within a well-defined organizational structure. These agents collaborate to generate real-time predictions, and as they continuously learn from incoming cases, the accuracy of the predictions improves over time. The system is implemented using the JaCaMo framework, which integrates three essential MAS programming layers: organization, environment, and agent programming. The methodology achieves a prediction accuracy of over 90%, surpassing state-of-the-art (SOTA) methods by enabling faster real-time decision-making. This efficiency in processing real-time big data contributes significantly to advancing predictive healthcare systems.

4.2 Motivation and overview

To demonstrate the importance of incorporating organizational perspectives into the design of Multi-Agent Systems (MAS), we present a scenario envisioning a solution for combating the Coronavirus pandemic. This study introduces an architecture for a Multi-Agent System that leverages machine learning algorithms to rapidly identify Covid-19-infected patients. Our approach is based on a real-time investigation system that collects physiological data from patients, including body temperature, ECG, heart rate, oxygen levels, blood pressure, glucose levels, and more.

The system relies on accurate information from hospitals connected to it, necessitating an efficient method for storing and processing large volumes of data. Big Data technology is utilized to digitally store comprehensive information on all Covid-19 cases, including those currently infected, recovered, or deceased. The stored data can be continuously analyzed to develop future preventive measures. The system applies a Multi-Agent Organizational model, enhanced with data analytics powered by machine learning (ML), to analyze the collected data and improve prediction models in real-time.

This solution aids local health authorities in monitoring a large number of users, promptly alerting them if symptoms are reported. Health officials can then reach out to the affected users, instructing them to report to the hospital for testing. Patients are admitted for observation until test results are confirmed. Additionally, the system tracks individuals in close contact with the infected patient, including family members, friends, and coworkers, and monitors them for any signs of infection.

By assisting health authorities in controlling the spread of Covid-19, this approach helps alleviate the burden on medical staff. Furthermore, the solution is adaptable for use in other hazardous pandemics or public health crises, offering a versatile and scalable tool for managing health emergencies.

Unlike traditional centralized systems, our distributed MAS architecture offers greater scalability, resilience, and localized decision-making capacity.

4.3 Related Work

As noted by Ilana et al. (2021) [166], the majority of AI research aimed at combating the coronavirus can be classified into four main categories: diagnosis and prognosis, treatments and vaccines, social control and tracking, and prediction. This paper focuses primarily on the latter category—prediction.

Otoom et al. (2020) [167] proposed a system for detecting and monitoring Covid-19 cases in real-time. During quarantine, IoT devices were deployed to gather real-time physiological data, and machine learning algorithms were used to enhance predictive accuracy. Their study compared seven machine learning algorithms, with five showing improvements in prediction accuracy. This demonstrates the effectiveness of using IoT in combination with machine learning for real-time pandemic response.

A survey by Thanh (2020) [168] analyzed various AI techniques applied to combat Covid-19, focusing on data analytics, natural language processing, and data mining. The work highlighted the diversity of AI applications in pandemic management, particularly in addressing key challenges such as big data processing and decision-making efficiency.

Carrillo et al. (2020) [169] employed unsupervised machine learning techniques, including k-means clustering, to classify countries based on similar Covid-19 infection patterns. This study emphasized the importance of using machine learning to understand the geographic spread of the virus and revealed significant insights into transmission trends across different regions.

Janko et al. (2021) [170] explored how non-countermeasure factors, such as culture, development, and travel, contributed to the early spread of Covid-19 before strict interventions were implemented. Using machine learning and statistical models, the study achieved approximately 80% prediction accuracy, showing how interconnected societal factors affect viral transmission in diverse contexts.

Recently, agent-based systems have emerged as a promising approach to addressing limitations in the healthcare sector. By integrating Multi-Agent Systems (MAS) into medical applications, healthcare costs can be reduced, and the burden on medical professionals can be alleviated. These systems shift the focus toward preventive, long-term care, which is patient-centered rather than hospital-centered. This transformation includes the use of remote monitoring systems that enable patients to play a more active

role in managing their health and treatment, especially during extended care periods in both hospitals and homes.

Isern et al.(2016) [171] developed a MAS-based platform for managing patient care during hospital stays. This platform collects real-time data from various sources, facilitating dynamic bed occupancy allocation, doctor assignments, medical procedure planning, and automated billing by tracking the behavior of hospital actors in real time. The system demonstrates the potential of MAS to optimize healthcare operations and improve hospital management efficiency.

Lanzola et al.(1999) [172] proposed a framework for developing interoperable Multi-Agent Systems for medical applications. Their work highlighted the need for MAS in enhancing collaboration across diverse healthcare systems. Additionally, Juan et al.(2006) [173] in an Ambient Intelligence (AmI) ecosystem for Alzheimer patients, another study by Gonzalez et al.(2002) [174] developed BDI agents that integrate context-aware technologies to gather real-time data from users, further advancing patient-centered care.

4.4 Discussion

To summarize the key methodologies, results, and contributions from the aforementioned studies, we present a comparative table (Table .4.1). This table provides a clear overview of the different approaches to Covid-19 prediction, their key features, and their relative performances. This table highlights how different methods approach COVID-19 prediction, focusing on collaboration efficiency, machine learning techniques, and accuracy. It also demonstrates the strength of our proposed MAS-based system, particularly its adaptability, collaboration-driven improvements, and real-time monitoring capabilities, which are critical for managing future pandemics and healthcare crises.

In summary, previous research demonstrates the effectiveness of AI and MAS in addressing various healthcare challenges, particularly in pandemic response and patient monitoring. Our study builds on these approaches by integrating Multi-Agent Organizational paradigms with IoMT devices and machine learning for real-time Covid-19 prediction, providing a novel solution that enhances predictive accuracy and supports public health efforts.

4.5 The Proposed Solution

Several studies conducted in the field of Multi-Agent learning have emphasized the importance of large datasets [175, 176]. Additionally, extensive research has been performed on various models, such as ensembles of classifiers [177–179]. The goal of this research is to develop a predictive Covid-19 case detection system using a Multi-Agent approach based on an organizational model. The system operates in a distributed environment and is composed of different sites referred to as "Assistant Controllers" (ACs). Each AC consists of two agents: a Learner Agent and an Interface Agent. It is crucial to note that different sites may have distinct instances of datasets. The knowledge generated by the independent ACs at each site will be consolidated into a single knowledge repository. The global system comprises of a collection of ACs and a special agent called a Broker agent or Mediator, as illustrated in (Fig. 5.1).

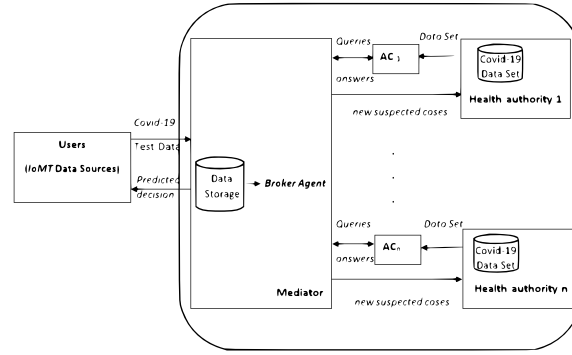


Figure 4.1: Global architecture of the MAS organizational approach for COVID-19 prediction

The agents communicate and share information about the selected datasets at each node. Subsequently, the Broker Agent selects the dataset that provides the most significant information. After this, the prediction decisions are sent to health authorities and users for preventive action (see Fig. 5.2). In our proposed approach, the agents work together within a structured organization to coordinate their behavior and cooperate to achieve the global objective of improving Covid-19 predictions.

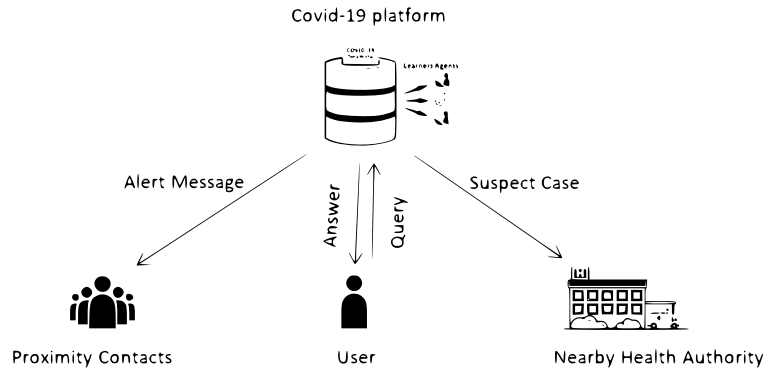


Figure 4.2: The global architecture of the MAS organizational approach to Covid-19 prediction.

4.5.1 Assistant Controller AC

An Assistant Controller (AC) is a Multi-Agent System (MAS) designed to categorize and predict Covid-19 patients using streaming datasets provided by health authorities. The structure of the AC is illustrated in Fig. 5.3. The Assistant Controller (AC) operates within the framework of the Agent and Artifact (A&A) paradigm, as discussed in Section 4.5.2. This paradigm serves as a conceptual model that outlines the interactions between autonomous agents and their environment, facilitating effective communication and collaboration.

At the heart of the AC's operation are two primary agents:

- *Interface Agent*: This agent is responsible for capturing every incoming dataset at the Assistant Controller (AC) and processing bid queries from the Broker Agent, ensuring that relevant data is available for decision-making. Upon receiving new data or

queries, the Interface Agent immediately notifies the Learner Agent by updating its environment through the Agent and Artifact (A&A) mechanism, where artifacts act as intermediaries to facilitate communication, coordination, and collaboration among agents. This process initiates the prediction process, ensuring continuous and adaptive updates to the predictive model.

- *Learner Agent*: This agent plays a crucial role in updating the predictive model. Upon receiving notifications from the Interface Agent, the Learner Agent incrementally updates the model based on the incoming data. It can also predict the queried bids as the expected results. This capability allows the Learner Agent to refine its predictions continuously, ensuring that the model remains current and relevant.

The real-time coordination between the Interface Agent and the Learner Agent enables the AC to continuously rebuild and dynamically update the predictive model. As a result, our system effectively manages highly dynamic healthcare datasets, adapting seamlessly to new information as it becomes available. This flexibility is essential in a rapidly changing context like Covid-19, where timely and accurate predictions can significantly impact public health responses.

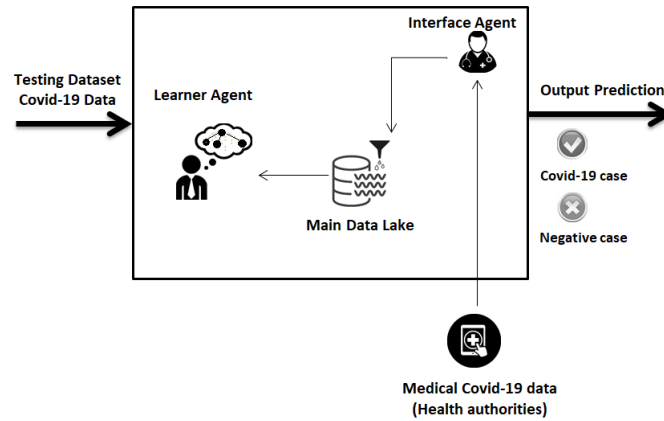


Figure 4.3: Assistant Controller architecture

Interface Agent

The Interface Agent plays a crucial role in cleaning, standardizing, and transforming data to produce high-quality datasets. Preparing these training datasets is essential for machine learning and statistical analysis, as it allows a computer to acquire knowledge and effectively learn problem-solving techniques. Both machine learning models and statistical analysis rely heavily on these well-prepared datasets.

Since datasets may contain erroneous or missing information, data cleaning is necessary to remove or correct these issues. The data is partitioned into separate sets for testing and training purposes, with the machine learning (ML) model being trained on 75% of the dataset. The efficiency of the proposed solution is then evaluated using the remaining 25%. Additionally, the Interface Agent serves as a connection point between the Learner Agent and the Broker Agent, facilitating seamless communication and coordination within the system.

Medical data streams: Whenever a new Covid-19 case is discovered, it must be reported promptly to the relevant Assistant Controller system. This allows the intelligent learning agents to update the model using only a single iteration through the data, leading to better and more accurate real-time predictions.

Learner Agent

The Learner Agent has two vital roles to fulfill. The first function is responsible for creating the training model, referred to as the Classify function. The second function is tasked with making predictions, known as the Predict function. This function utilizes the model to predict unknown outcomes. Notably, the goals of these two missions can be achieved in parallel, leveraging the strengths of a Multi-Agent System (MAS) and incremental machine learning techniques.

Supervised machine learning algorithms include a classification type, where a model is developed from a set of labeled training data. As a result, the model can predict the class or label for newly discovered data, known as testing data. Metrics such as accuracy, which are calculated based on the testing results, can be used to evaluate classification performance [180]. Numerous classification algorithms exist, including decision trees, random forests, neural networks, rule-based algorithms (such as conjunctive rules and PRISM), logistic regression, naive Bayes, and others. Each algorithm has its distinct advantages and limitations, and the selection of an appropriate algorithm depends on various factors, including the data's attributes and the desired objectives [181].

Decision trees are highly regarded classification techniques in the field of data mining and have applications across various domains, including business intelligence, biomedicine, and healthcare. The conventional method of creating a decision tree is known as the Greedy Search approach. This process involves loading the complete dataset into memory and organizing it into a series of nodes and leaves that form a hierarchical structure. However, a significant drawback of this method is that once the decision tree has been constructed, it cannot be easily modified or updated, even in the presence of new data. Incorporating newly acquired information necessitates the complete reconstruction of the tree by loading both historical and recent data, a process that may be time-consuming and computationally intensive. Additionally, this approach risks losing critical information or patterns that were present in the original tree. To address these limitations, various alternative methods for decision-tree learning have been developed to enhance the scalability and adaptability of decision trees to changing data [117].

The conventional approach is inadequate for handling limitless data inputs, such as data streams, where information arrives in real-time. To tackle this challenge, an incremental approach has been introduced, enabling dynamic model construction so that the tree expands as new data is inputted. The Very Fast Decision-Tree (VFDT) algorithm, proposed by [182], employs the Hoeffding Bound [183] for incremental node splitting. This approach constructs a decision tree by continuously observing the features of incoming data and maintaining a record of their statistics. Once sufficient statistics are collected from every leaf, a node-splitting algorithm is applied to determine whether there is adequate statistical evidence to justify a node split. If a node split is warranted, the tree is expanded, and a new decision node is added in place of the leaf. This technique allows the decision tree to learn continuously as the data stream is processed in real-time [184]. The primary advantage of this approach is its ability for real-time data mining, eliminating the

need to store all data in advance since data streams are potentially unlimited. Unlike traditional decision trees that require retraining with every new data arrival, the incremental approach allows for dynamic updates to the tree, enabling more efficient and effective data analysis. The process of building a VFDT is demonstrated in Algorithm 2 [185]. Notably, node divisions across multiple leaves can occur concurrently and autonomously [186].

While several libraries in the literature support incremental learning—particularly for the Very Fast Decision Tree (VFDT), such as MOA, Apache Spark MLlib, Scikit-Multiflow, and River—we have chosen WEKA for the following reasons:

- *Compatibility with Java Eclipse:* WEKA is fully compatible with Java, allowing for seamless integration with the Java Eclipse IDE. Its rich package and API facilitate the easy incorporation of WEKA’s machine learning capabilities into larger Java applications, making it a versatile tool for implementing machine learning within the JaCaMo framework.
- *User-Friendly Interface:* WEKA provides an intuitive interface for data preprocessing, model evaluation, and visualization. This streamlines the machine learning workflow, making it accessible for researchers who are new to the field.
- *Community Support and Documentation:* WEKA boasts a strong community and extensive documentation, which facilitate troubleshooting and ensure the availability of resources for further experimentation.
- *Suitability for Multi-Agent Systems:* WEKA’s capabilities align well with the requirements of our multi-agent system, particularly in handling large datasets, performing real-time predictions, and integrating seamlessly with the JaCaMo framework.
- *Runtime Metrics:* We will present detailed runtime metrics for WEKA’s machine learning tools to demonstrate their efficiency, including average training and testing times for various algorithms implemented within WEKA, such as Random Forest, VFDT, and others utilized in our study.

We selected the Very Fast Decision Tree (VFDT) algorithm for several compelling reasons relevant to healthcare applications. First, VFDT is designed for high scalability [?], enabling it to efficiently manage large-scale data streams typical in healthcare settings. Unlike ensemble-based approaches, which often require full retraining, VFDT allows for incremental updates, making it well-suited for the rapid influx of patient data. Additionally, VFDT’s real-time adaptation capabilities [187] ensure that the model can quickly adjust to dynamic changes in healthcare datasets. Finally, its memory efficiency [188] enables effective resource management, which is crucial in healthcare environments where computational power may be limited. Thus, the combination of efficiency, adaptability, and memory management makes VFDT a more suitable choice for our predictive modeling needs compared to ensemble-based incremental algorithms.

4.5.2 Broker Agent

The system continuously receives incoming queries related to Covid-19 testing datasets for prediction. To facilitate communication between the collection of Assistant Controller

Algorithm 2: Very Fast Decision Tree Induction

Input: S : A sequence of examples; X : The set of attributes; γ : One minus the desired probability; ϵ : The Hoeffding bound

Output: τ : very fast decision tree learned from S

```

1 Initialize  $\tau$  with a single root node.
2 Initialize the statistics for tree growth.
3 foreach  $s \in S$  do
4   Sort  $s$  into leaf node  $l$  using  $\tau$ .
5   Update the statistics at  $l$  for tree growth.
6   if Examples at  $l$  are not from the same class then
7     foreach attribute  $X_a \in X$  do
8       Calculate the Hoeffding bound using the formula:  $\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2|S|}}$ 
          (where  $R$  is the range of the attribute values);
9       Select the attribute with the highest Hoeffding bound and split  $l$  on
          that attribute.
10      if multiple attributes have the same highest Hoeffding bound then
11        select the one with the highest information gain.
```

(AC) agents and external entities, An intermediary agent, the Broker Agent, facilitates communication between Assistant Controller (AC) agents and external entities. The Broker Agent is responsible for mediating interactions between the AC agents and the external environment.

Once it receives the queries, the Broker Agent utilizes Random Forest techniques to select the most appropriate predictions based on the aggregated data from the AC agents. This approach ensures that the predictions are robust and take advantage of the diverse insights provided by the various AC agents.

Random forest

The Random Forest algorithm is an ensemble learning method that involves building multiple decision trees and combining their predictions to enhance accuracy. Each decision tree is constructed using a randomly selected subset of the training data and a random subset of features. The final prediction is determined by averaging the predictions from all the trees in the forest. This approach helps reduce overfitting and improves the overall accuracy of the model. Based on the principles of Random Forest classification [189], the proposed approach involves selecting the most relevant majority vote decision from various Assistant Controller (AC) groups. This method operates under the assumption that different AC groups may have access to different sources of information and may employ distinct decision-making criteria. By combining the decisions from multiple AC groups, we can leverage their diverse perspectives and expertise to arrive at a more informed and accurate conclusion.

We chose Random Forest as the primary decision-making technique at the broker agent level due to its strong performance in managing complex, high-dimensional datasets

typical of healthcare applications. Although Random Forest presents challenges in interpretability—especially when dealing with distinct datasets from each AC—its robustness and ability to handle missing data (particularly when one AC does not respond), along with its capacity to provide actionable insights, make it a strong candidate for our predictive modeling efforts in the healthcare domain.

To implement this approach, we first need to identify the relevant AC groups that can contribute input to the decision-making process. Various criteria can be employed to select these groups, including their area of expertise, past performance, or level of authority.

Once the relevant AC groups are identified, we can gather their decisions and use a voting mechanism to determine the most pertinent majority decision. This can involve assigning weights to the decisions based on the expertise or performance of each group or utilizing a more complex algorithm to combine decisions in a meaningful manner.

Overall, this approach has the potential to improve the accuracy and robustness of the decision-making process by leveraging the collective intelligence of multiple AC groups. However, it also requires careful coordination and communication between the different groups to ensure that their decisions are aligned and consistent with the overall goals and objectives of the organization.

Contract Net

To prevent potential bottlenecks associated with the Broker Agent, the system employs the Contract Net Protocol, originally proposed by [190] and later refined by [191]. This protocol is designed to facilitate the announcement of transferable tasks and to solicit bids from Interface Agents representing each Assistant Controller (AC) group capable of executing these tasks.

When a task is announced, the Interface Agents respond with bids that indicate their perceived capability to fulfill the task. The Broker Agent collects these bids and allocates the task to the bidder with the highest offer. Although the Contract Net Protocol is often viewed as a negotiation technique, its primary function is to act as a coordination mechanism for task allocation.

Task Announcement and Bid Solicitation: When a specific task arises—such as processing a new Covid-19 dataset for prediction—the Broker Agent announces this task to the relevant AC groups within the system. Each Interface Agent receives the announcement and evaluates its own capabilities, resources, and current workload before deciding whether to respond with a bid. The bid submitted by each Interface Agent includes not only an estimate of the time and resources required to complete the task but also their confidence level in executing it successfully.

Bid Evaluation and Task Allocation: Once the Broker Agent has received all the bids from the Interface Agents, it conducts a thorough evaluation to determine which bid offers the best combination of capability and cost. The Broker Agent allocates the task to the bidder with the highest offer, taking into account the quality of the bid and the capabilities of the Interface Agent. This allocation process ensures that tasks are assigned to agents that are most likely to perform them effectively and efficiently.

Coordination and Flexibility: Although the Contract Net Protocol is often viewed primarily as a negotiation technique, its primary function is to serve as a coordination mechanism for task allocation. By allowing dynamic task allocation and enabling agents

to bid for multiple tasks simultaneously, the protocol enhances the system's flexibility and adaptability. This is particularly important in a healthcare setting where the demands on the system can vary significantly and unpredictably.

Workload Balancing The protocol also promotes workload balancing within the system. Since agents that are already busy with other tasks are less likely to place bids, the distribution of tasks tends to be more even across the available agents. This self-regulating mechanism helps to prevent any single agent from becoming overwhelmed with too many tasks, thereby maintaining overall system performance.

Limitations of the Contract Net Protocol: However, the Contract Net Protocol does have its limitations:

- *Conflict Detection and Resolution:* One notable drawback is that the protocol lacks built-in mechanisms for conflict detection and resolution. If two agents bid for the same task or if their bids conflict in terms of resource availability, the protocol does not provide a framework for resolving these conflicts, which may lead to inefficiencies or delays.
- *Heavy Reliance on Communication:* The protocol heavily relies on communication between agents. This reliance can introduce delays or inefficiencies in task allocation, especially in scenarios where network latency or communication failures occur. If the communication channels are congested or disrupted, it could hinder the timely allocation of tasks.
- *Scalability Concerns:* In larger systems with numerous agents and tasks, the process of soliciting bids and evaluating them can become computationally expensive and time-consuming. This could potentially lead to performance bottlenecks as the system scales.
- *Limited Bidder Pool:* The effectiveness of the protocol is contingent upon having a sufficient number of agents willing and able to submit bids. If too few agents respond to a task announcement, it could result in suboptimal task allocation.

Overall, while the Contract Net Protocol offers a structured and efficient means of task allocation within the Broker Agent framework, it is important to be aware of these limitations and to consider supplementary mechanisms or strategies that could mitigate potential issues and enhance the overall robustness of the system.

Agents and Artifacts (A&A) Environment

Our proposal involves utilizing the Agents and Artifacts meta-model to create a coordination mechanism for developing a shared environment, as described by [192]. This model addresses some of the shortcomings of the Contract Net Protocol by representing the environment as workspaces, where Agents and Artifacts coexist to provide various services. In this environment, agents interact with artifacts through a mechanism known as the focus action. When an agent focuses on an artifact, it gains access to the observable properties of that artifact as perceptions. Additionally, artifacts can offer a range of operations or actions that agents can perform, enhancing collaboration and interaction within the system. *Bidding Process for Task Allocation:* The Broker Agent plays a crucial

role in locating Interface Agent contractors for each Assistant Controller (AC) group (Fig. 5.4). This is achieved through a structured bidding process, which follows these steps:

- The Broker Agent announces the task (the task is considered an artifact in the system).
- Interface Agents evaluate the task based on their capabilities and commitments.
- Interface Agents submit bids to the Broker Agent.
- The Broker Agent evaluates the received bids and selects the majority vote from the competing predictions.

Decentralized and Collaborative Approach: This bidding scheme is completely distributed, meaning that each agent operates independently without a centralized controller. The Broker Agent facilitates the announcement of tasks and the collection of bids, but the decision-making process relies on the collective actions of multiple agents. Each Interface Agent autonomously assesses the task based on its own capabilities and commitments, leading to a diverse range of bids that reflect varying perspectives. The final decision regarding task allocation is made based on the majority of predictions from the Interface Agents. This approach not only ensures a decentralized and collaborative handling of tasks but also leverages the diverse expertise of multiple agents to enhance the accuracy and reliability of the outcomes.

By employing the Agents and Artifacts framework, we create a flexible and adaptive environment that can effectively manage the dynamic and complex nature of healthcare data processing tasks. The interaction between agents and artifacts allows for improved coordination, resource utilization, and task execution in the context of predicting Covid-19 patient outcomes.

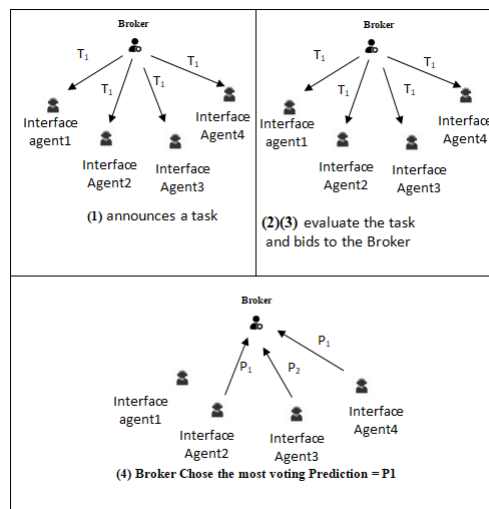


Figure 4.4: Broker Agent

4.6 Some Implementation Aspects

The integration process described in the previous sections was implemented using the Eclipse Modeling and JaCaMo framework. The JaCaMo¹ framework is a multi-agent-oriented programming platform (MAOP) designed to facilitate the development of complex multi-agent systems [1]. This framework utilizes autonomous agents programmed in Jason², which operate within environments defined as artifacts in CArtaGO³. Additionally, these agents are organized by the Moise framework⁴, and they interact with each other using a specified interaction language.

All experiments were conducted on a system equipped with an Intel Core i5-1035 G1 processor running at 1.00 GHz and 16 GB of RAM. The software application AlertCovid employs the proposed model, consisting of three Assistant Controllers named UK, Biskra, and ESI. The Broker Agent serves as the intermediary, facilitating communication and coordination among the different Assistant Controllers.

4.6.1 UK Assistant Controller

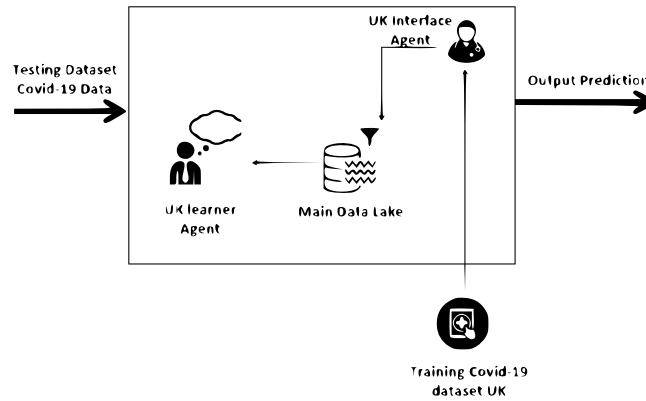


Figure 4.5: UK architecture

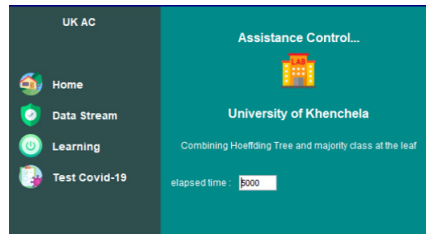


Figure 4.6: UK interface

¹<http://jacamo.sourceforge.net>

²<http://jason.sourceforge.net/wp>

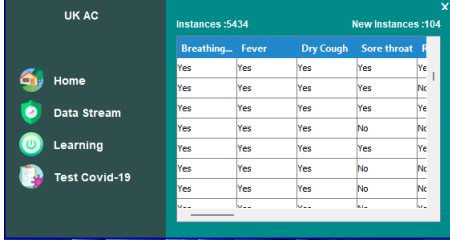
³<http://cartago.sourceforge.net>

⁴<http://moise.sourceforge.net/>

UK Interface Agent

For the training data at the UK Assistant Controller site (Fig 5.6), we utilize the dataset titled Covid-19 Symptoms and Presence, which is publicly available on Kaggle [193]. This dataset comprises 20 attributes that represent potential factors associated with contracting Covid-19, including symptoms such as cough and fever, along with other relevant indicators. Additionally, it contains a class attribute indicating the presence of the virus.

The dataset is sourced from reputable organizations, including the World Health Organization and the All India Institute of Medical Sciences, with the aim of facilitating research into the prevalence of symptoms and their correlation with Covid-19 diagnoses. Its comprehensive nature makes it a valuable resource for predictive modeling in the healthcare domain.



The screenshot shows a web application titled 'UK AC'. On the left is a dark sidebar with navigation links: Home, Data Stream, Learning, and Test Covid-19. The main content area has a teal header with 'Instances :5434' and 'New Instances :104'. Below the header is a table with columns: Breathing..., Fever, Dry Cough, Sore throat, and a partially visible 'R' column. The table contains several rows of 'Yes' and 'No' values.

Breathing...	Fever	Dry Cough	Sore throat	R
Yes	Yes	Yes	Yes	Ye
Yes	Yes	Yes	Yes	Nc
Yes	Yes	Yes	Yes	Ye
Yes	Yes	Yes	No	Nc
Yes	Yes	Yes	Yes	Ye
Yes	Yes	Yes	No	Nc
Yes	Yes	Yes	No	Nc
Yes	Yes	Yes	No	Nc

Figure 4.7: UK interface agent

UK learner Agent

The UK Assistant Controller's learning agent implement the Hoeffding tree algorithm, which is integrated into the open-source software WEKA, to obtain knowledge from data streams. WEKA is a suite of machine learning algorithms developed at the University of Waikato in New Zealand [194]. It facilitates data mining by providing a wide range of tools for clustering, pre-processing, visualization, association, regression, and classification.

The VFDT implementation in the UK learner agent uses the following specific parameters Fig. 5.7:

- *Splitting Criterion*: Gini Index, which is a measure of the impurity or variance of a node. The algorithm selects the attribute with the lowest Gini Index to split the node, leading to purer child nodes.
- *Allowable Error δ* : $1 \cdot 10^{-7}$. This parameter controls the confidence level for deciding whether to split a node. A lower error threshold ensures higher confidence in the decision to split, minimizing the likelihood of unnecessary splits, which helps in maintaining the efficiency of the model.
- *Grace Period*: 100 instances. This setting ensures that the algorithm waits for 100 instances before evaluating a potential split at any node. This helps to accumulate sufficient data to make statistically sound splitting decisions, avoiding premature splits based on insufficient data.

These parameters help the UK learner agent achieve a balance between accuracy and efficiency, making it capable of handling continuous data streams while ensuring the decision tree adapts quickly to new data without overfitting.

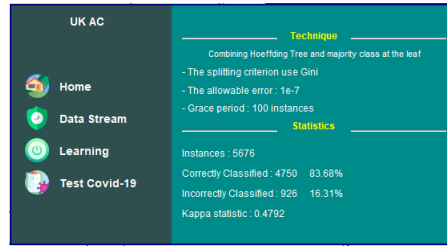


Figure 4.8: UK Learner agent

A learning curve plots accuracy against the number of instances, as illustrated in (Fig. 4.9), showing how the model's performance improves with more training data.

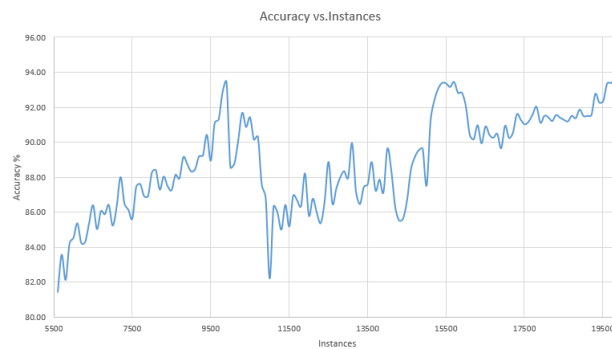


Figure 4.9: UK Learner Curve

- The accuracy starts at 81.42% at 5602 instances and fluctuates slightly as more data is processed.
- Over time, there is a general upward trend, with accuracy reaching 93.65% at 1982 instances.
- The model demonstrates significant improvement, achieving over 90% accuracy as it processes larger datasets.

Analysis:

- The learning curve likely depicts an overall positive trend with some small fluctuations. These fluctuations suggest that the model is learning from new data but may occasionally encounter challenges from certain instances.
- Despite these variations, the general increase in accuracy indicates the model's capacity to learn and improve its predictions as more instances are introduced.

The UK learner model shows a general improvement in accuracy as more data is processed, starting at 81.42% and reaching over 90%. Despite minor fluctuations, the overall trend is upward, indicating the model's ability to adapt and learn effectively from increasing data.

UK test Covid-19 Case

With the implementation of our proposed architecture, we developed a GUI module for testing the UK Assistant Controller, enabling functional testing at the local level. Fig.4.10 presents an example of a positive Covid-19 case.

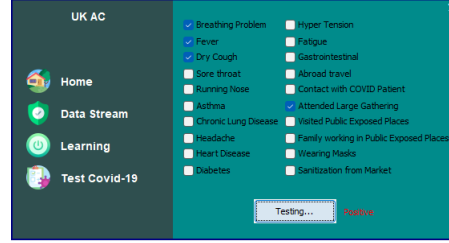


Figure 4.10: UK Test Covid-19 case

4.6.2 ESI Assistant Controller

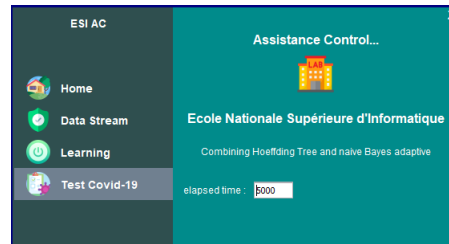


Figure 4.11: ESI Interface

ESI Interface Agent

The ESI Interface Agent operates as a crucial component within the multi-agent system, specifically tasked with handling and processing COVID-19-related health data from a localized data stream. To enable effective COVID-19 prediction, the ESI Interface Agent utilizes a dataset sourced from a research repository [195]. This combined dataset, titled "Covid Symptoms," features 3,021,444 entries and ten essential characteristics related to COVID-19 symptoms and test outcomes. The dataset includes key information on symptoms such as cough, fever, shortness of breath, sore throat, and headache, which are critical indicators for COVID-19 detection.

Given the dataset's comprehensive nature, the ESI Interface Agent processes and cleans the data to ensure its usability for predictive modeling. The cleaned and preprocessed data is then forwarded to the Learner Agent, which uses the Very Fast Decision Tree (VFDT) algorithm for incremental learning. This approach allows the agent to continuously update the prediction model in real time as new COVID-19 cases are added to the stream, without the need for retraining from scratch.

The choice of this dataset aligns perfectly with the goals of the ESI Interface Agent for several reasons:

- *Rich Symptom Features:* The dataset contains detailed records of COVID-19 symptoms, allowing the ESI Interface Agent to make highly accurate predictions. By processing a diverse range of symptoms such as cough, fever, and sore throat, the agent can feed comprehensive input into the learning model.
- *Large Scale Data:* With over 3 million entries, the dataset provides the necessary volume for evaluating the robustness and scalability of the system. This is especially useful in a real-world scenario where the ESI Interface Agent must process a continuous influx of new COVID-19 cases.
- *Real-Time Updates:* The incremental learning method employed by the Learner Agent ensures that the ESI Interface Agent can handle real-time data without delays. The large dataset simulates real-world conditions, where updates to the model must happen quickly to reflect new information.
- *Binary Class Labels:* The presence of a class attribute indicating a positive or negative COVID-19 test result is crucial for the classification tasks performed by the agent. This simplifies the prediction process, allowing for straightforward binary classification and evaluation of prediction accuracy.

In summary, the "Covid Symptoms" dataset not only aligns with the core tasks of the ESI Interface Agent but also ensures the agent can process high-volume data efficiently, update models in real time, and make accurate predictions based on symptom analysis. By incorporating this dataset into the system, the ESI Interface Agent plays a vital role in ensuring the overall success of the multi-agent COVID-19 prediction architecture.

4.6.3 Biskra Assistant Controller

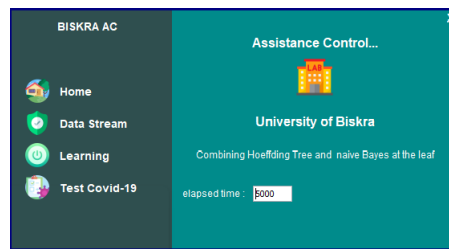


Figure 4.12: BISKRA Interface

Biskra Interface Agent

The dataset titled 'Covid-19 Symptoms Checker,' sourced from Kaggle [196], contains 27 attributes related to Covid-19 symptoms and demographic information that may affect whether an individual has contracted the Coronavirus. Key attributes include:

- *Symptoms:* Fever, Tiredness, Dry Cough, Difficulty in Breathing, Sore Throat, and more.
- *Severity Levels:* Indicators for Mild, Moderate, Severe, and None.

- *Contact Information*: Details on contact with confirmed Covid-19 cases.
- *Demographics*: Gender (Male, Transgender) and Country.

This dataset is suitable for building a robust model to better predict Covid-19-infected patients for several reasons:

- *Comprehensive Symptoms*: The dataset includes a wide array of Covid-19 symptoms, making it highly relevant for early detection and diagnosis in real-time applications.
- *Severity Assessment*: Having labels for different severity levels helps in predicting the potential progression of the infection, allowing healthcare authorities to prioritize responses based on predicted outcomes.
- *Contact History*: Information on whether individuals were in contact with confirmed cases significantly improves the predictive power of the model, aiding in identifying likely transmissions.
- *Demographic Factors*: Attributes like gender and country allow for a detailed analysis of symptom presentation and the potential impact of demographics on infection susceptibility and disease outcomes.

4.6.4 The Broker

The role of the Broker is to mediate between different Assistant Controllers (UK, ESI, Biskra) (Fig .4.13), demonstrating the benefits of collaboration despite varying interests and goals. Autonomous and collaborative decision-making among the three Assistant Controllers is facilitated through the use of the Contract Net Interaction Protocol (Fig .4.14).

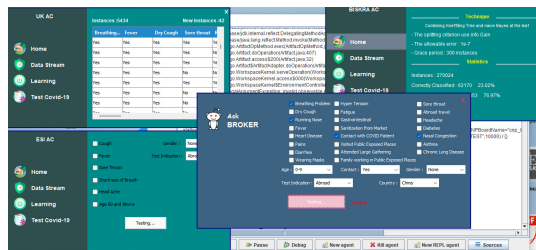


Figure 4.13: Broker Interface

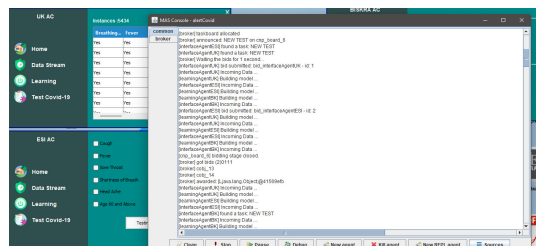


Figure 4.14: Broker use Contract Net Protocol

4.7 Comparison and Discussion

Fig .4.15 illustrates a comparison of the accuracy results from a portion of the dataset, starting from instance 7500 onward. This timeframe allows the system to build its model effectively. The comparison includes the UK, Biskra, and ESI agents, as well as the Broker Agent.

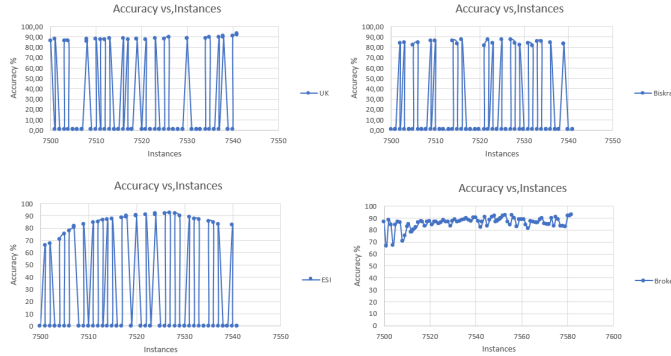


Figure 4.15: Broker Accuracy

The performance of the UK, Biskra, and ESI ACs is summarized as follows:

- *UK AC:* Active in roughly half of the instances, with an accuracy ranging from 85% to 92% when involved.
- *Biskra AC:* Sporadically active, performing well when engaged, with an accuracy between 80% and 90%.
- *ESI AC:* The least active but consistently reliable when used, often achieving 85% or higher accuracy.

4.7.1 Broker Performance Highlights

- *Consistent Activity:* Unlike the individual agents, the broker is always active and never shows an accuracy of 0. It ensures that predictions are made even when some agents are inactive.
- *Accuracy Selection::* The broker typically mirrors the highest-performing agent's accuracy, selecting or aggregating the best prediction.
- *Aggregation Role:* When multiple agents are active, the broker's performance reflects its ability to select the most accurate prediction or combine insights, ensuring higher accuracy.

4.7.2 Key Observations

- *Broker Stability:* The broker's accuracy remains consistently strong, never dropping to 0, unlike the agents. This reliability makes the broker a crucial decision-maker, especially when certain agents are inactive.

- *Dependence on Agents:* The broker performs better when multiple agents contribute. When only one agent is active, the broker’s accuracy mirrors that agent’s performance. For example:

- In instance 7541, only the UK agent contributes, resulting in the broker’s accuracy being equal to the UK agent’s at 92.86

The response times of the Broker Agent (in nanoseconds) across various instances are illustrated in (Fig .4.16). The broker’s response times range from 5,600 ns to a peak of 113,700 ns in instance 7540, with another notable spike at 50,200 ns in instance 7574. These variations indicate occasional spikes in response times, possibly due to increased processing complexity or load.

The average response time of the Broker Agent is approximately 11,716 ns. Despite occasional delays, which may depend on the materials used, the broker maintains a relatively low average response time, indicating that it can efficiently manage predictions under typical conditions.

The broker plays a critical role in ensuring continuous and accurate predictions, achieving over 90% accuracy consistently over time. Utilizing the Random Forest technique, the broker consistently selects the best-performing agent’s result, enhancing the overall system’s reliability and performance. To further improve accuracy, increasing the participation of AC, would provide the broker with more data to work with and potentially boost overall performance.

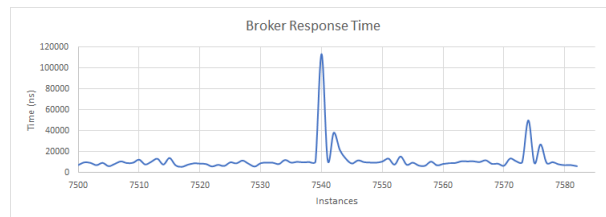


Figure 4.16: Broker Reponse Time

4.8 Conclusion and Future Work

In conclusion, this study successfully demonstrates the effectiveness of a multi-agent system (MAS) utilizing the Internet of Medical Things (IoMT) in predicting COVID-19-infected patients. The approach presented is grounded in an organizational structure within the MAS, defining the expected behaviors of agents collaborating towards a common healthcare objective. Through the implementation of experimental studies with various datasets, the proposed approach illustrates the contributions and benefits of organizational frameworks in MAS systems, achieving significant improvements in prediction accuracy, reaching over 90%.

Furthermore, we propose a flexible organizational model that allows for dynamic reorganization of the MAS in response to environmental changes. To enhance system performance, future work will focus on increasing the number of broker agents to mitigate potential bottlenecks. Additionally, rather than relying solely on random forests, we plan to explore alternative techniques for decision-making among different Assistant Controllers.

This research contributes to the growing body of knowledge in health informatics, offering viable solutions for public health authorities to monitor and respond effectively to pandemics. Future investigations could extend this model to other infectious diseases and further integrate it with existing healthcare infrastructures. The findings advocate for continued investment in smart health technologies that can transform healthcare delivery and improve patient outcomes.

Table 4.1: Performance Comparison of Predictive Methods

Article Title	Key Metrics/Variables	Methods	Results/Finding
[167]:An IoT-based Framework for Early Identification and Monitoring of Covid-19 Cases	Accuracy, real-time health data	IoT, Ensemble Learning (RF GBTs), Apache Spark	Achieved over 90% accuracy with various algorithms; proposed a real-time framework for monitoring Covid-19 using IoMT and WSNs.
[169]:Using Country-Level Variables to Classify Countries According to Confirmed Covid-19 Cases	Covid-19 case counts, country variables	Unsupervised Machine Learning	Identified distinct clusters of countries based on Covid-19 case counts using unsupervised learning techniques.
[172]:A Framework for Building Cooperative Software Agents in Medical Applications	Collaboration efficiency, application areas	Framework Development, Agent-Based Modeling	Proposed a cooperative agent framework for healthcare applications to enhance collaboration among medical agents.
[173]:Intelligent Environment for Monitoring Alzheimer Patients, Agent Technology for Health Care.	Patient monitoring metrics, AI integration	Agent Technology, Sensor Networks	Developed an intelligent monitoring environment for Alzheimer's patients to enhance care and support for families.
[170]:Machine Learning for Analyzing Non-Countermeasure Factors Affecting Early Spread of Covid-19	Spread factors, machine learning accuracy	Machine Learning, Data Analysis	Analyzed non-countermeasure factors affecting Covid-19 spread; developed predictive models with promising accuracy in identifying influential factors.
The proposed approach	Collaboration efficiency ,Decision-making efficiency , machine learning accuracy , real-time health data	Multi Agent based on an Organizational Modeling, Incremental Machine & ensemble learning Techniques	Support for Remote Real-time Monitoring Alerts, Improved Decision-Making Customizable for Large Scale Pandemics

Real-Time Course Timetable Scheduling

Effective planning of courses in educational institutions requires the capacity to adjust in real time to changing constraints. Conventional approaches frequently find it difficult to handle these complexity successfully. The *JaCaMo* framework is used in this study to implement an innovative Multi-Agent Organization *MAO* approach that makes it easier to program Multi-Agent Systems *MAS* by organizing cognitive agents that operate within an artifact-based environment. The suggested Approach uses autonomous agents to handle resource allocation, preference management, scheduling, and other course scheduling-related tasks. Within a structured organization, these agents work together to dynamically optimize schedules. By utilizing *JaCaMo*'s capabilities, the built approach allows for effective real-time change adaption. By means of comprehensive testing and analysis, we prove the efficiency and expandability of our methodology in generating better schedules. The results demonstrate how organization-based multi-agent systems, which provide improved flexibility, adaptability, and efficiency with real-time processing, have the potential to completely transform course scheduling techniques.

5.1 Introduction

Many researchers had been unable to complete the difficult process of creating a university course timetable. It's considered an *NP-hard* problem, which means that solving it perfectly is extremely hard. Manual timetabling is time-consuming and error-prone, particularly as student and course numbers grow. The study offers a timetable scheduling approach based on multi-agent architecture intended for educational institutions in order to overcome these problems. By automating the scheduling process, this system seeks to reduce the need for human labor while increasing speed and accuracy. It also aims to avoid problems brought on by parameter adjustments without necessitating a full rescheduling procedure and reacts to these adjustments instantly.

When creating complicated software systems, an effective approach called agent-oriented engineering is employed. Because autonomous agents serve as its core, it operates in dynamic contexts. Alignment with global aims may be hampered by issues like distributed agent behavior brought on by this autonomy [197]. In crucial applications such as timetable scheduling, it is necessary to take into account the behavior of the entire system. The goal of multi-agent system research is to comprehend how agents can work

together to accomplish goals that are greater than what they could accomplish separately. *Organizations* assist in the execution of solutions in order to deal with the dispersing effects of agent autonomy. A social construct called a multi-agent organization is made up of several agents placed in particular topologies and communication systems. Together, these agents complete a variety of obligations that are in line with the organization's main goals.

5.2 Related Work

Timetable scheduling has been a focus of numerous studies across academia, industry, and business domains. Multi-agent systems (MAS) have emerged as a promising approach due to their adaptability and distributed decision-making capabilities. *Cheema et al.* [198] proposed a JADE-based multi-agent framework for course timetable scheduling, which includes capturing, monitoring, and distributing agents. Although effective in resource allocation, it lacks scalability and real-time adaptability. Similarly, *Klimov et al.* [199] introduced a MAS for self-regulated learning, incorporating student agents and personalized scheduling. However, hard constraints like resource availability are not thoroughly addressed.

Houhamdi et al. [200] presented a decision support system that integrates multi-agent features to consider teacher preferences and minimize conflicts. However, the focus on specific constraints limits its ability to adapt to dynamic changes. *Syskov et al.* [201] developed a MAS model to support individualization in education through automated scheduling, while *Junn et al.* [202] combined hyper-heuristics with MAS for optimization, leveraging simulated annealing and cooperative agents. Despite their innovations, these approaches encounter challenges related to scalability and computational resources.

Nourmohammadi-Khiarak et al. [203] proposed a hybrid MAS using genetic and imperialist competitive algorithms to handle scheduling conflicts. Similarly, *Henry-Obit et al.* [204] developed a distributed MAS for timetable negotiation, resolving conflicts via mediator agents. While effective, these approaches have limited capacity for managing dynamic constraints and real-time resource reallocation.

Recent work by the Atlantis Press team [205] introduces a dynamic timetable scheduling framework combining multi-agent systems with federated learning. While demonstrating real-time adaptability and preserving privacy, their approach differs from ours in its centralized coordination of agents, whereas our solution emphasizes decentralized organizational structures enabled by JaCaMo.

The proposed Multi-Agent Organizational (MAO) approach builds upon these works by leveraging the JaCaMo framework's modularity and real-time responsiveness. It combines decentralized collaboration with efficient hard and soft constraint handling, overcoming limitations in scalability and adaptability found in existing methods.

5.2.1 Discussion

The proposed Multi-Agent Organizational (MAO) approach introduces several advantages over existing methods for course timetable scheduling:

1. *Real-Time Adaptability:* The system dynamically adjusts schedules and reallocates

resources in response to changing constraints, ensuring uninterrupted operations even in dynamic environments. This feature significantly enhances scheduling reliability and flexibility.

2. *Comprehensive Constraint Handling:* The approach balances hard constraints, such as room availability and teacher schedules, with soft constraints, such as preferences for specific time slots. By managing these constraints effectively, the system produces conflict-free and efficient timetables.
3. *Decentralized Collaboration:* The use of autonomous agents with specialized roles for resource management, preference evaluation, and schedule optimization ensures high efficiency and scalability. This decentralized structure enables the system to handle large-scale scheduling scenarios effectively.
4. *Optimized Computational Performance:* The modular architecture minimizes computational overhead, enabling the system to address the challenges of complex scheduling tasks without excessive resource consumption. This makes it well-suited for institutions requiring efficient solutions for large datasets.

A comparative analysis of SOTA methods “Table. 5.1” highlights these advantages. While hierarchical or hybrid MAS frameworks have demonstrated success, they often require significant computational resources or fail to address dynamic adaptability. The proposed approach, through its modular architecture, ensures flexibility, efficiency, and practical applicability across diverse scheduling scenarios.

5.3 The Proposed Solution

Course scheduling is a complex NP-hard optimization problem that requires distributing resources such as courses, instructors, and rooms among timeslots while adhering to numerous constraints. We propose a MAS approach to handle both hard and soft constraints and create conflict-free timetables using organizational strategies.

5.3.1 Constraints in Timetabling:

Hard constraints

these need to be met and are not negotiable. As an illustration:

- No teacher should be assigned to teach two classes concurrently.
- Never make two bookings for the same room.
- The only room that should be used for the course is one that can accommodate everyone.

Soft constraints

these are preferences that, in theory, should be honored but though, in practice, may be disregarded. As an illustration:

- Students might like fewer breaks in between classes.
- Instructors might have preferred specific times for their classes.

The difficulty in resolving scheduling issues is striking a balance between these limitations, making effective use of available resources, and guaranteeing a workable, conflict-free timetable. We provide a *MAS* that addresses both hard and soft constraints and creates feasible timetables using organizational strategies. In this system, autonomous agents that are assigned certain duties cooperate to find solutions to challenging scheduling issues. The system effectively manages both stringent requirements, like resource availability, and flexible preferences by assigning the work to several agents and facilitating communication and negotiation between them to coordinate their efforts. This decentralized method improves efficiency, scalability, and flexibility, which makes it perfect for complex and dynamic scheduling problems.

The first step in setting up a *MAS* is for players involved in its creation to carry out a definition phase. After that, agents must obey the rules established by the designated organization during the execution phase. As agents collaborate to make decisions about how to behave, this process could consist of the iterative weaving of these phases. Agents actively participate in actions, perceptions, and collaboration inside their organization, defining and modifying it along the way. Two descriptions are produced by this organizing process: an organization entity and an organization specification [206]. *JaCaMo* makes it easier to implement organizational programming, in which the organizational dimension structures the coordination of team agents rather than depending just on the abilities of individual agents. Within this framework, agents have the ability to choose how best to attain system goals; the organization provides parameters, but they still maintain autonomy.

5.3.2 The organization Dimension

To organize our agents, we adopt the *Moise* model integrated in *JaCaMo* platform, Beginning with Defining the organization for timetable scheduling, the specification of the organization has three parts: structural(groups, roles, ...)“Fig. 5.1”, functional(social schemes, goals, ...) “Fig. 5.2”, and normative(norms)“Tab. 5.2”.in the structural part of this organization, we have the following roles to be played by agents:

1. *ResourceManager*: An agent responsible for overseeing academic resources at educational institutions may have this crucial role in the *ResourcesStaff* group’s course schedule scheduling. Academic administrators, registrars, or scheduling coordinators are examples of these agents; they manage the allocation of resources such as classrooms, the tools needed for course scheduling, and time slots to create an efficient and productive course schedule.

The agents in charge of overseeing instructor preferences, course scheduling, and evaluation procedures are members of the *SchedulersStaff* group. These intermediaries organize with interested parties, compile, and rank the preferences of teachers.

2. *Teacher*: Agents in this capacity gather details regarding teachers’ availability, including preferred teaching times and any scheduling constraints due to other commitments. Additionally, they collect preferences relating to specific courses, class

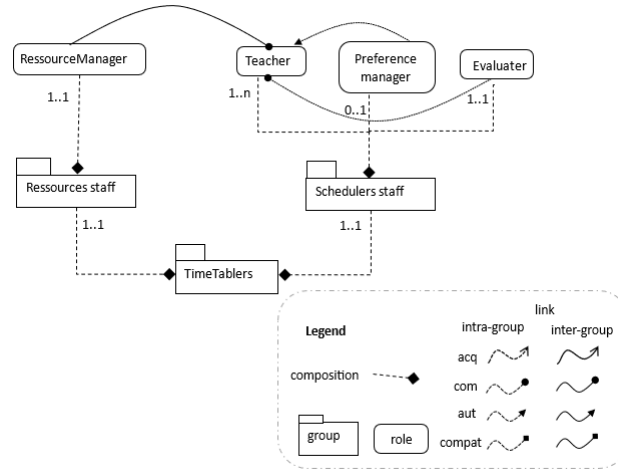


Figure 5.1: *Moise* structural specification for timetable schedule

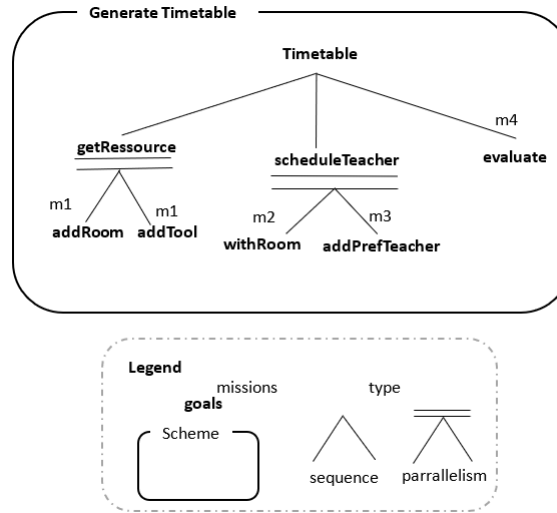


Figure 5.2: *Moise* functional specification for timetable schedule

sizes, or teaching environments, which can influence the scheduling process. Furthermore, agents ensure flexibility and adaptability among teachers, enabling them to accommodate changes in the timetable, such as room changes or class rescheduling due to unforeseen circumstances.

3. *PreferencesManager*: This is an important position in course timetabling scheduling because it ensures that stakeholders' preferences are considered and properly integrated into the planning process. By efficiently handling preferences, the Preference Manager aids to the establishment of schedules that are both realistic and satisfying to all concerned parties.
4. *Evaluator*: Agents in the job of Evaluator examine timetable performance criteria such as room usage, teacher workload, student satisfaction, and overall schedule quality. They evaluate the created timeline to the objectives, trying to maximize resource utilization, reduce conflicts, and meet stakeholder preferences.

“Fig. 5.3” depicts the *JaCaMo MAS* file’s source code for the timetable project. This code defines the agents, their duties, and group memberships to provide effective coordination and collaboration when managing the project’s timeline. Figure 4 depicts an expanded version of the *MOISE++* XML organization specification *OS* built for the project. This standard most likely defines the structure and formatting guidelines for the XML data required to describe different components of the timetable, such as groups and responsibilities.

```

1  /*
2     JaCaMo timeTabling File
3     This file defines the initial state of the MAS (initial agents,
4         environment, organisation, ....)
5  */
6  mas timeTabling {
7     agent s : scheduler.asl
8     agent r : ressource.asl
9     agent p : preference.asl
10    agent e : evaluate.asl
11
12    organisation timetable : timetable_org.xml{
13        group r1 : ressources_staff{
14            players: r ressource_manager
15        }
16        group r2 : schedulers_staff{
17            players: s teacher_scheduler
18                p preference_manager
19                e evaluator
20
21        responsible-for : genTT
22        }
23        scheme genTT: generatTT
24    }
25
26
27    asl-path: src/agt,
28            src/agt/inc
29 }
30

```

Figure 5.3: the Organization Entity of the timetabling project

The Environment Dimension

“Fig. 5.5” illustrates an abstract representation of timetable CArtaGO artifact, according to the A&A meta-model. It highlights properties, and operations pertaining to the primary GUI artifact within the timetable project.

The Agent Dimension

The agent dimension describes the internal architecture of agents, including their mental states and interactions. It facilitates the creation of software entities with their unique logical control flow. These agents possess the capability to autonomously and proactively pursue their objectives, engaging in the environment, other agents, and the governing organization. Communication and information sharing among agents occur through two primary mechanisms: messages and artifacts (environment), as elaborated in Section 3.2. The JaCaMo agent plan within the organization is designed to delineate how an agent can fulfill both its individual goals and the overarching objectives of the organization. For instance, “Figs. 5.6 and 5.7” exemplify Jason source code of teacher and scheduler agents in our approach.

```

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="http://moise.sourceforge.net/xml/os.xml" type="text/
xml" ?>

<organisational-specification
  id="timeTabling_org"
  os-version="0.8"

  xmlns='http://moise.sourceforge.net/os'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xsi:schemaLocation='http://moise.sourceforge.net/os
    http://moise.sourceforge.net/xml/os.xsd' >

  <structural-specification>

    <role-definitions>
      <role id="ressource_manager" />
      <role id="teacher_scheduler" />
      <role id="preference_manager" />
      <role id="evaluator" />
    </role-definitions>

    <group-specification id="timetablers">
      <subgroups>
        <group-specification id="ressources_staff">
          <roles>
            <role id="ressource_manager" min="1" max="1"/>
          </roles>
        </group-specification>
        <group-specification id="schedulers_staff">
          <roles>
            <role id="teacher_scheduler" min="1" max="10"/>
            <role id="preference_manager" min="1" max="1"/>
            <role id="evaluator" min="1" max="1"/>
          </roles>
        </group-specification>
      </subgroups>
    </group-specification>
  </structural-specification>

```

Figure 5.4: XML Organization specification for timetable

5.4 Implementation

We implemented the proposed solution using the JaCaMo framework and Eclipse Modeling “Figs. 5.8”. The system efficiently schedules teachers and rooms in real time based on the availability of resources and constraints. The implementation was tested on an Intel Core i5 system with 16 GB RAM. The system dynamically allocates rooms and adjusts timetables as new constraints and agents are introduced.

The example in “Figs. 5.8” illustrates a timetable involving four teachers and two rooms. In the first step, the system creates agents such as the resource manager r , scheduler s , preference p and evaluator e agents. The scenario involves two teachers, $T1$ and $T2$, each teaching two distinct groups: $G1$ and $G3$, with two courses, $C1$ and $C2$. Group $G1$ has 10 students, while group $G3$ has 5 students. The scheduler agent creates the teacher agents and waits for available rooms. When the GUI creates room $R1$, with a capacity of 5 (as set by the resource manager agent), it satisfies the capacity requirement for teacher $T2$ and is allocated to the first available time slot (Room $R1$, Slot 0) in 12838 milliseconds. Later, a new teacher named $T3$ joins in, $T3$, along with group $G4$ and course $C3$, with a capacity of 10 students. The scheduler agent creates the $T3$ agent according to the timetable organization scheme and searches for an open room. When room $R2$ becomes available, and it can fit 10 students, it is allocated to $T3$ in time slot 1 (Room $R2$, Slot 1) in 16103 MS. Afterward, room $R2$ is also allocated to $T1$ in time slot 0 (Room $R2$, Slot 0) in 65137 MS, as $T1$ had been waiting for a room. Finally, teacher agent $T4$ enters the system, teaching course $C4$ to group $G5$, which consists

TimeTableGUI extends GUIArtifact	
Observable Properties	
rooms : Rooms //classe List of room	
teachers : Teachers	
preferences : Preferences	
...	
Artifact Operations	
+addRoom	
+findRoom	
+addTeacher	
+teacherName	
+scheduleTeacherRoom //Schedule teacher at the specific room	
+scheduleTeacher	
+unscheduleTeacher	
+addPreference	
+preferenceRomTeacher	
...	

Figure 5.5: the main parts and properties of *CARTAGO TimeTable* GUI artifact

```

1 // Agent scheduler in project timeTabling
2
3 !startSchedul. /* initial goal */
4
5 +!startSchedul <- !setup;
6     //focusing scheme genTT
7     lookupArtifact(genTT,SchArtId);
8     focus(SchArtId);
9     //play teacher_scheduler for the first time
10    adoptRole(teacher_scheduler).
11
12 +!setup <- //Create GUI artifact
13    makeArtifact("SchedulGUI", "artifcats.TimeTableGUI", [], GUIId);
14    focus(GUIId).
15
16 // when a scheme genTT is focusing I commit to missions for each goal
17 +goalState(_,withRoom,_,_,waiting) <- commitMission(mission2).
18
19 +teacherAdd(Id) <- teacherName(Id, NamTech);
20    .print("creating agent teacher : ", NamTech);
21    // Create new teacher agent and start schedule
22    .create_agent(NamTech, "teacher.asl");
23    .send(NamTech, achieve, schedul(Id)).
24
25 +teacherSignal <- addTeacher(Id).
26
27 +closed <- .print("Scheduler agent leave the organization: ");
28    .drop_all_intentions;
29    .my_name(Me);
30    .kill_agent(Me).
31
32 { include("$jacamoJar/templates/common-cartago.asl") }
33 { include("$jacamoJar/templates/common-moise.asl") }
34 { include("$moiseJar/asl/org-obedient.asl") }

```

Figure 5.6: Source code of the Jason *scheduler* agent in timetable project

of 5 students. The scheduler initially allocates room $R1$ to $T4$ in the second time slot (Room $R1$, Slot 1) in 0 *MS*. However, when the user modifies the room preference to room $R2$, the scheduler agent reschedules $T4$ to the preferred room, as it satisfies the capacity constraint, allocating it to slot 2 (Room $R2$, Slot 2). The system works fast! Teachers get scheduled as they arrive. It makes sure each teacher finds a free room and time slot based on their schedule needs.

5.4.1 System Evaluation

The graph “Fig. 5.9” represents Scheduling Time for 400 teachers and 80 rooms arriving every 100 milliseconds. The approach guarantees that only available rooms are allocated, providing a measure of system efficiency, scalability, and performance benchmarking against alternatives. The analysis includes:

- *Initial Phase*: Low scheduling time due to fewer conflicts.

```

1 // Agent teacher in project timeTabling
2 +!withRoom <- .print("satisfying withRoom goal in scheme: gentTT").
3 +!addPrefTeacher <- .print("satisfying addPrefTeacher goal in scheme: gentTT").
4
5 // when a scheme gentTT is focused I commit to missions for each goal
6 +goalState(.,withRoom,.,.,waiting) <- commitMission(mission2).
7 +goalState(.,addPrefTeacher,.,.,waiting) <- commitMission(mission3).
8
9 +!schedul(Id) <-
10   ?discover("SchedulGUI", ArtId);
11   teacherName(Id, N);
12   .print("i am looking for free room");
13   scheduleTeacher(Id, TimeSlot, Room);
14   .print("i found room Id: ", Room, " at time slot N° :", TimeSlot);
15   !withRoom.
16
17 +?discover(N, I) <- //focusing timetable Organisation
18   joinWorkspace(timetable,WspId);
19   println("joined workspace :", WspId);
20   //focusing r2 group
21   lookupArtifact(r2, GrArtId)[wid(WspId)];
22   focus(GrArtId)[wid(WspId)];
23   println("focusing on artifact :", GrArtId);
24   // check role cardinality
25   if (not max_players_for(teacher_scheduler)){
26
27       //focusing scheme gentTT
28       lookupArtifact(gentTT,SchArtId)[wid(WspId)];
29       focus(SchArtId)[wid(WspId)];
30       //play teacher_scheduler for the first time
31       adoptRole(teacher_scheduler)[artifact_id(GrArtId)];
32       debug(inspector_gui(on))[artifact_id(WspId)];
33   } else {
34       .print("There are enough assistants already!");
35       ?kill_an_agent;
36   }
37   //focusing GUI artifact

```

Figure 5.7: Source code of the Jason *teacher* agent in timetable project

- *Rising Complexity*: Increased scheduling time as conflicts arise and are managed.
- *Fluctuations*: Dynamic adjustments typical of multi-agent systems.
- *Steady State*: Stabilization as most resources are allocated.

Key Highlights:

1. *Real-Time Adaptation*: Processes arrivals every 100 ms, handling dynamic changes efficiently.
2. *Scalability and Robustness*: The algorithm maintains effective performance as the number of agents scales.
3. *Efficiency in Constraint Handling*: Ensures free rooms are allocated, minimizing idle resources.
4. *Practical Impact*: Applicable in environments requiring rapid, adaptive decision-making, like education and healthcare.

Overall, the approach demonstrates significant efficiency and scalability, proving its viability for large-scale, complex scheduling needs. It validates that decentralized, organization-based multi-agent systems can outperform traditional centralized methods.

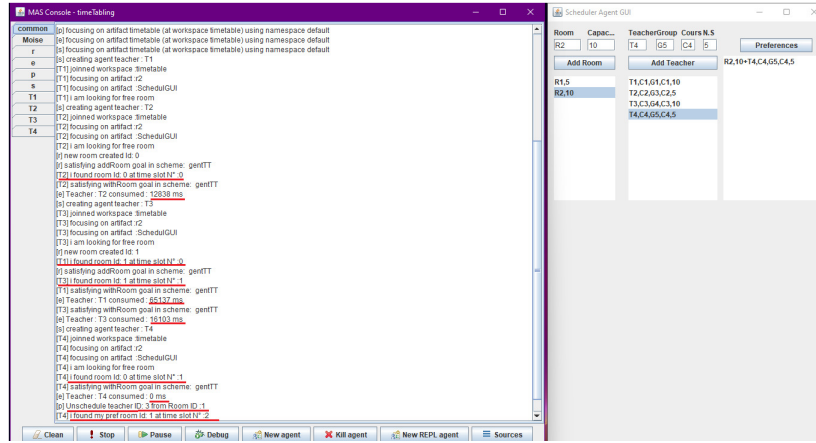


Figure 5.8: App Implementation



Figure 5.9: Scheduling Time.

5.5 Summary

The proposed system effectively automates course scheduling, reducing manual effort, minimizing conflicts, and optimizing resource utilization. Its real-time adaptability ensures seamless operation under dynamic constraints, making it highly suitable for large-scale educational institutions and other scheduling-intensive domains. By handling both hard and soft constraints, the system ensures efficient resource allocation and stakeholder satisfaction.

However, the approach presents some limitations. The reliance on the JaCaMo platform introduces a learning curve for developers unfamiliar with agent-oriented paradigms. In addition, while the system has demonstrated promising results on synthetic test scenarios, real-world deployment and testing on institutional datasets are necessary to confirm robustness, especially under unpredictable user behavior and network conditions.

Future work will involve validating scalability on larger datasets to ensure robustness, extending the application to dynamic environments such as healthcare and logistics, and incorporating advanced decision-making techniques like reinforcement learning and evolutionary algorithms to enhance adaptability and long-term performance.

Table 5.1: Comparison of State-of-the-Art Approaches for Timetable Scheduling

Study	Approach	Key Features	Limitations
SM Cheema et al.	Multi-Agent System (JADE)	Hierarchical roles, web-based with Android support.	Limited scalability; lacks real-time adaptability.
Z. Houhamdi et al.	Decision Support System	Handles preferences and minimizes conflicts.	Focuses on specific constraints; lacks dynamic adaptability.
KY Junn et al.	Hyper-Heuristics (Custom)	Simulated annealing, cooperative agents.	High computational cost; scalability issues.
J. Henry-Obit et al.	Distributed MAS with Mediation	Resolves shared timeslot conflicts via negotiation.	Limited handling of dynamic constraints.
D. Klimov et al.	Individualized Scheduling (MAS)	Personalized timetables, learning analytics tools.	Hard constraints like room allocation neglected.
J. Nourmohammadi-Khiarak et al.	Hybrid MAS (Genetic Algorithms)	Combines MAS with genetic optimization techniques.	High complexity and computational overhead.
V. Sharmila et al. (2025)	MAS + Federated Learning	Real-time adaptability, privacy-preserving learning, agent collaboration at scale	Centralized coordination; lacks explicit organizational modeling
Proposed Approach	Multi-Agent Organizational System (JaCaMo)	Real-time adaptability, decentralized collaboration, efficient handling of constraints.	Scalability testing required for very large datasets.

Table 5.2: *Moise* deontic specification for timetable schedule

<i>Role</i>	<i>Mission</i>	<i>Link</i>
RessourceManager	m1	obligation
Teacher	m2	obligation
PreferenceManager	m3	permission
Evaluator	m4	obligation

Conclusion

This thesis contributes to the field of Ambient Intelligence (AmI) by demonstrating a novel approach to developing adaptive and intelligent environments using organization-centered Multi-Agent Systems (MAS). We successfully integrated the Moise organizational modeling language and the JaCaMo framework to create a unified platform for designing, implementing, and managing complex MAS. By modeling social structures, defining roles and norms, and specifying collective goals, our approach facilitates coordinated agent behavior, efficient resource allocation, and dynamic adaptation to changing contexts. The efficacy of this methodology was validated through two distinct case studies. The first demonstrated a significant improvement in real-time COVID-19 infection prediction by leveraging IoMT data and a distributed multi-agent architecture enhanced with machine learning. This showcases the potential for more timely and accurate responses to public health crises. The second case study, applying the framework to real-time course timetable scheduling, highlighted its ability to handle dynamic constraints and optimize resource allocation in complex scheduling scenarios. These findings underscore the power of combining organizational paradigms with artificial learning within MAS. Future research avenues include exploring more sophisticated learning algorithms, refining organizational semantics for increased autonomy, and investigating the integration of other AI techniques like argumentation. Ultimately, this work provides a robust foundation for building more intelligent, adaptable, and scalable AmI systems capable of addressing complex challenges across various domains, from healthcare to education and beyond.

Bibliography

- [1] O. Boissier, R. H. Bordini, J. Hubner, and A. Ricci, *Multi-agent oriented programming: programming multi-agent systems using JaCaMo*. Mit Press, 2020.
- [2] M. N. Huhns, *Distributed Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987.
- [3] J. Ferber and M.-A. LES SYSTEMES, “Vers une intelligence collective,” *Paris InterEdition*, 1995.
- [4] J. Erceau, “Intelligence artificielle distribuée et systèmes multi-agents de la théorie aux applications,” *23ème Ecole Internationale d’Informatique de l’AFCET, Neuchâtel*, 1993.
- [5] Y. Demazeau, “From interactions to collective behaviour in agent-based systems,” in *European Conference on Cognitive Science*, vol. 95, 1995.
- [6] Y. Demazeau and J.-P. Müller, “From reactive to intentional agents,” *Decentralized AI*, vol. 2, pp. 3–10, 1991.
- [7] M. Minsky and D. Riecken, “A conversation with marvin minsky about agents,” *Communications of the ACM*, vol. 37, no. 7, pp. 22–29, 1994.
- [8] Y. Shoham, “Agent-oriented programming,” *Artificial intelligence*, vol. 60, no. 1, pp. 51–92, 1993.
- [9] T. Bouron, “Structures de communication et d’organisation pour la coopération dans un univers multi-agents,” *Université Pierre et Marie Curie (Paris VI)*, 1992.
- [10] E. H. Durfee and T. A. Montgomery, “Coordination as distributed search in a hierarchical behavior space,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 6, pp. 1363–1378, 1991.
- [11] S. Cammarata, D. McArthur, and R. Steeb, “Strategies of cooperation in distributed problem solving,” in *Readings in Distributed Artificial Intelligence*, pp. 102–105, Elsevier, 1988.
- [12] R. Davis, “Report on the workshop on distributed ai,” 1980.

- [13] K. Schmidt, J. Rasmussen, B. Brehmer, and J. Leplat, "Cooperative work: A conceptual framework," *Distributed decision making: Cognitive models for cooperative work*, pp. 75–110, 1991.
- [14] J. R. Galliers, *A theoretical framework for computer models of cooperative dialogue, acknowledging multi-agent conflict*. Open University (United Kingdom), 1989.
- [15] A. Vailly and M. Simon, "Des systèmes experts coopérants, pourquoi, comment?," *Cognition*, vol. 87, 1987.
- [16] J. Erschler, "Approche par contraintes pour l'aide à la décision et à la coopération: une nouvelle logique d'utilisation des modèles formels," *Coopération et Conception sous la direction de G. de Terssac et E. Friedberg*, Octares Editions, pp. 137–147, 1996.
- [17] P. Cohendet and P. Llerena, "Flexibilité et évaluation des systèmes de production," *Gestion Industrielle et Mesure Economique, ECOSIP, Economica, Paris*, 1990.
- [18] A. Newell, "The knowledge level," *Artificial intelligence*, vol. 18, no. 1, pp. 87–127, 1982.
- [19] B. Hayes-Roth, "A blackboard architecture for control," *Artificial intelligence*, vol. 26, no. 3, pp. 251–321, 1985.
- [20] H. P. Nii, "Cage and polygon: two frameworks for blackboard-based concurrent problem solving," *THIS DOCUMENT IS THE BEST QUALITY AVAILABLE. COPY FURNISHED CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT*, p. 142, 1986.
- [21] C. Hewitt, "Viewing control structures as patterns of passing messages," *Artificial intelligence*, vol. 8, no. 3, pp. 323–364, 1977.
- [22] J. Erceau and J. Ferber, "L'intelligence artificielle distribuée," *La recherche*, no. 233, pp. 750–758, 1991.
- [23] E. H. Durfee, V. R. Lesser, and D. D. Corkill, "Coherent cooperation among communicating problem solvers," *IEEE Transactions on computers*, vol. 100, no. 11, pp. 1275–1291, 1987.
- [24] D. Kayser, "Examen de diverses méthodes utilisées en représentation des connaissances," *Actes du 4ème CARFIA*, pp. 115–144, 1984.
- [25] N. Aussenac, "Conception d'une méthodologie et d'un outil d'acquisition de connaissances expertes," *These de doctorat en informatique, Université Paul Sabatier, Toulouse. Cité par (Dieng1993)*, 1989.
- [26] J. Breuker and B. Wielinga, "Kads—structured knowledge acquisition for expert systems," in *5th International Workshop Vol. 2 on Expert systems & their applications*, pp. 887–900, 1987.

- [27] A. Napoli, *Représentations à objets et raisonnement par classification en intelligence artificielle*. PhD thesis, Henri Poincaré University, Nancy, France, 1992.
- [28] C. Braesch and A. Haurat, “La modélisation systémique en entreprise,” (*No Title*), 1995.
- [29] M. Minsky and J. Henry, *La société de l’esprit*. Nouveaux horizons, InterEditions, 1988.
- [30] C. Iffenecker, “Un système multi-agents pour le support des activités de conception de produits,” *These de doctorat, Université de Pierre et Marie Curie*, 1992.
- [31] P. R. Cohen and H. J. Levesque, “Intention is choice with commitment,” *Artificial intelligence*, vol. 42, no. 2-3, pp. 213–261, 1990.
- [32] B. Chaib-Draa, *Contribution à la résolution distribuée de problème : une approche basée sur les états intentionnels*. PhD thesis, 1990. Thèse de doctorat dirigée par Millot, Patrick Automatique industrielle et humaine Valenciennes 1990.
- [33] R. E. Fikes, “A commitment-based framework for describing informal cooperative work,” *Cognitive Science*, vol. 6, no. 4, pp. 331–347, 1982.
- [34] J. C. Augusto and P. McCullagh, “Ambient intelligence: Concepts and applications,” *Computer Science and Information Systems*, vol. 4, no. 1, pp. 1–27, 2007.
- [35] B. Kokinov, “A dynamic approach to context modeling,” in *Proceedings of the IJCAI-95 workshop on modeling context in knowledge representation and reasoning. LAFORIA*, vol. 95, Citeseer, 1995.
- [36] N. Ryan, J. Pascoe, and D. Morse, “Enhanced reality fieldwork: the context aware archaeological assistant,” 1999.
- [37] K. Ducatel, U. européenne. Technologies de la société de l’information, U. européenne. Institut d’études de perspectives technologiques, and U. européenne. Société de l’information conviviale, “Scenarios for ambient intelligence in 2010,” 2001.
- [38] M. Friedewald, E. Vildjiounaite, Y. Punie, and D. Wright, “The brave new world of ambient intelligence: An analysis of scenarios regarding privacy, identity and security issues,” in *Security in Pervasive Computing: Third International Conference, SPC 2006, York, UK, April 18-21, 2006. Proceedings 3*, pp. 119–133, Springer, 2006.
- [39] M. J. Huber, “Jam: A bdi-theoretic mobile agent architecture,” in *Proceedings of the third annual conference on Autonomous Agents*, pp. 236–243, 1999.
- [40] R. Z. Mili and R. Steiner, “Modeling agent-environment interactions in adaptive mas,” in *International Workshop on Engineering Environment-Mediated Multi-Agent Systems*, pp. 135–147, Springer, 2007.
- [41] M. Bratman, “Intention, plans, and practical reason,” 1987.
- [42] A. S. Rao, M. P. Georgeff, *et al.*, “Bdi agents: from theory to practice.,” in *Icmas*, vol. 95, pp. 312–319, 1995.

- [43] T. Balke, F. Dignum, M. RIEMSDIJK, and A. K. Chopra, *Coordination, Organizations, Institutions, and Norms in Agent Systems Ix*. Springer, 2013.
- [44] V. Guivarch, V. Camps, and A. Péninou, “Amadeus: an adaptive multi-agent system to learn a user’s recurring actions in ambient systems,” *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, vol. 1, no. 3, p. 1, 2012.
- [45] B. Schilit, N. Adams, and R. Want, “Context-aware computing applications,” in *1994 first workshop on mobile computing systems and applications*, pp. 85–90, IEEE, 1994.
- [46] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, “Towards a better understanding of context and context-awareness,” in *Handheld and Ubiquitous Computing: First International Symposium, HUC’99 Karlsruhe, Germany, September 27–29, 1999 Proceedings 1*, pp. 304–307, Springer, 1999.
- [47] J. Brézillon and P. Brézillon, “Context modeling: Context as a dressing of a focus,” in *Modeling and Using Context: 6th International and Interdisciplinary Conference, CONTEXT 2007, Roskilde, Denmark, August 20–24, 2007. Proceedings 6*, pp. 136–149, Springer, 2007.
- [48] K. Henriksen and J. Indulska, “Developing context-aware pervasive computing applications: Models and approach,” *Pervasive and mobile computing*, vol. 2, no. 1, pp. 37–64, 2006.
- [49] A. Olaru, A. M. Florea, and A. El Fallah Seghrouchni, “Graphs and patterns for context-awareness,” in *Ambient Intelligence-Software and Applications: 2nd International Symposium on Ambient Intelligence (ISAmI 2011)*, pp. 165–172, Springer, 2011.
- [50] V. Guivarch, V. Camps, and A. Péninou, “Context awareness in ambient systems by an adaptive multi-agent approach,” in *Ambient Intelligence: Third International Joint Conference, AmI 2012, Pisa, Italy, November 13–15, 2012. Proceedings 3*, pp. 129–144, Springer, 2012.
- [51] A. Shendarkar, K. Vasudevan, S. Lee, and Y.-J. Son, “Crowd simulation for emergency response using bdi agents based on immersive virtual reality,” *Simulation Modelling Practice and Theory*, vol. 16, no. 9, pp. 1415–1429, 2008.
- [52] M. P. Georgeff and F. F. Ingrand, *Decision-making in an embedded reasoning system*. Australian Artificial Intelligence Institute, 1989.
- [53] M. Ljungberg and A. Lucas, “The oasis air-traffic management system,” in *Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence, PRICAI*, vol. 92, pp. 185–190, 1992.
- [54] M. P. Georgeff and A. S. Rao, “A profile of the Australian artificial intelligence institute,” *IEEE Intelligent Systems*, vol. 11, no. 06, pp. 89–92, 1996.

- [55] A. Pokahr, L. Braubach, and W. Lamersdorf, “Jadex: A bdi reasoning engine,” *Multi-agent programming: Languages, platforms and applications*, pp. 149–174, 2005.
- [56] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*, vol. 15. John Wiley & Sons, 2007.
- [57] M. Dastani, “2apl: a practical agent programming language,” *Autonomous agents and multi-agent systems*, vol. 16, pp. 214–248, 2008.
- [58] J. P. Müller, *The design of intelligent agents: a layered approach*, vol. 1177. Springer Science & Business Media, 1996.
- [59] M. d’Inverno, D. Kinny, M. Luck, and M. Wooldridge, “A formal specification of dmars,” in *Intelligent Agents IV Agent Theories, Architectures, and Languages: 4th International Workshop, ATAL’97 Providence, Rhode Island, USA, July 24–26, 1997 Proceedings 4*, pp. 155–176, Springer, 1998.
- [60] M. E. Bratman, D. J. Israel, and M. E. Pollack, “Plans and resource-bounded practical reasoning,” *Computational intelligence*, vol. 4, no. 3, pp. 349–355, 1988.
- [61] M. E. Pollack and J. F. Horty, “There’s more to life than making plans: plan management in dynamic, multiagent environments,” *AI Magazine*, vol. 20, no. 4, pp. 71–71, 1999.
- [62] K. Hindriks, “Twenty years of engineering multiagent systems,” in *Keynote of the EMAS workshop, part of AAMAS*, vol. 14, 2014.
- [63] M. Lützenberger, T. Küster, T. Konnerth, A. Thiele, N. Masuch, A. Heßler, J. Keiser, M. Burkhardt, S. Kaiser, and S. Albayrak, “Jiac v: A mas framework for industrial applications,” in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pp. 1189–1190, 2013.
- [64] S. Sardina, L. De Silva, and L. Padgham, “Hierarchical planning in bdi agent programming languages: A formal approach,” in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pp. 1001–1008, 2006.
- [65] S. Shapiro, S. Sardina, J. Thangarajah, L. Cavedon, and L. Padgham, “Revising conflicting intention sets in bdi agents,” in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 1081–1088, 2012.
- [66] M. Waters, L. Padgham, and S. Sardina, “Evaluating coverage based intention selection,” in *AAMAS*, vol. 14, pp. 957–964, 2014.
- [67] F. R. Meneguzzi, A. F. Zorzo, M. da Costa Móra, and M. Luck, “Incorporating planning into bdi systems,” *Scalable Computing: Practice and Experience*, vol. 8, no. 1, 2007.

- [68] A. L. Blum and M. L. Furst, “Fast planning through planning graph analysis,” *Artificial intelligence*, vol. 90, no. 1-2, pp. 281–300, 1997.
- [69] N. R. Jennings, K. Sycara, and M. Wooldridge, “A roadmap of agent research and development,” *Autonomous agents and multi-agent systems*, vol. 1, pp. 7–38, 1998.
- [70] A. L. Lansky, *Behavioral specification and planning for multiagent domains*. Artificial Intelligence Center, SRI International, 1985.
- [71] H. J. Levesque, P. R. Cohen, and J. H. Nunes, *On acting together*, vol. 485. SRI International Menlo Park, CA 94025-3493, 1990.
- [72] O. Gutknecht and J. Ferber, “Madkit: Organizing heterogeneity with groups in a platform for multiple multi-agent systems,” *Rapport Interne LIRMM*, vol. 97188, p. 1997, 1997.
- [73] K. S. Barber and D. N. Lam, “Specifying and analyzing agent architectures using the agent competency framework.,” in *SEKE*, pp. 232–239, 2003.
- [74] Y. Labrou and T. Finin, “A proposal for a new kqml specification,” tech. rep., Technical Report Technical Report TR-CS-97-03, University of Maryland . . . , 1997.
- [75] P. R. Cohen, A. Cheyer, M. Wang, and S. C. Baeg, “An open agent architecture,” in *AAAI Spring Symposium*, vol. 1, 1994.
- [76] H. Suguri, “A standardization effort for agent technologies: The foundation for intelligent physical agents and its activities,” in *Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences. 1999. HICSS-32. Abstracts and CD-ROM of Full Papers*, pp. 10–pp, IEEE, 1999.
- [77] D. V. Pynadath, M. Tambe, N. Chauvat, and L. Cavedon, “Toward team-oriented programming,” in *Intelligent Agents VI. Agent Theories, Architectures, and Languages: 6th International Workshop, ATAL’99, Orlando, Florida, USA, July 15-17, 1999. Proceedings 6*, pp. 233–247, Springer, 2000.
- [78] F. Zambonelli, N. R. Jennings, and M. Wooldridge, “Developing multiagent systems: The gaia methodology,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 12, no. 3, pp. 317–370, 2003.
- [79] Y. B. D. Secq, *RIO: Rôles, Interactions et Organisations: une méthodologie pour les systèmes multi-agents ouverts*. PhD thesis, Lille 1, 2003.
- [80] A. Omicini, “Soda: Societies and infrastructures in the analysis and design of agent-based systems,” in *International workshop on agent-oriented software engineering*, pp. 185–193, Springer, 2000.
- [81] M. Hannoun, “Moise: un modele organisationnel pour les systemes multi-agents,” *These (Doctorat), École Nationale Supérieure des Mines de Saint-Etienne*, 2002.
- [82] W. Malone Thomas, C. Kevin, and L. Jintae, “Tools for inventing organizations,” *Management Science*, vol. 45, no. 3, pp. 425–443, 1999.

-
- [83] B. Philippe, “La sociologie des organisations,” *Paris, éditions du Seuil, Essais Points, 5ème éd*, vol. 1990, p. 382, 1985.
- [84] L. Gasser, “Perspectives on organizations in multi-agent systems,” *Multi-Agent Systems and Applications: 9th ECCAI Advanced Course, ACAI 2001 and Agent Link’s 3rd European Agent Systems Summer School, EASSS 2001 Prague, Czech Republic, July 2–13, 2001 Selected Tutorial Papers 9*, pp. 1–16, 2001.
- [85] O. Boissier, J. F. Hübner, and J. S. Sichman, “Organization oriented programming: From closed to open organizations,” in *Engineering Societies in the Agents World VII: 7th International Workshop, ESAW 2006 Dublin, Ireland, September 6–8, 2006 Revised Selected and Invited Papers 7*, pp. 86–105, Springer, 2007.
- [86] G. Picard, M.-P. Gleizes, and P. Glize, “Distributed frequency assignment using cooperative self-organization,” in *First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)*, pp. 183–192, IEEE, 2007.
- [87] G. Picard, J. F. Hübner, O. Boissier, and M.-P. Gleizes, “Reorganisation and self-organisation in multi-agent systems,” in *1st International Workshop on Organizational Modeling, ORGMOD*, pp. 66–80, 2009.
- [88] J. F. Hübner, J. S. Sichman, and O. Boissier, “Moise+ towards a structural, functional, and deontic model for mas organization,” in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pp. 501–502, 2002.
- [89] L. R. Coutinho, J. S. Sichman, O. Boissier, *et al.*, “Modeling organization in mas: a comparison of models,” in *First Workshop on Software Engineering for Agent-oriented Systems*, pp. 1–10, Brazilian Computer Society Uberlândia, Brazil, 2005.
- [90] M. Tambe, “Towards flexible teamwork,” *Journal of artificial intelligence research*, vol. 7, pp. 83–124, 1997.
- [91] D. V. Pynadath and M. Tambe, “An automated teamwork infrastructure for heterogeneous software agents and humans,” *Autonomous Agents and Multi-Agent Systems*, vol. 7, pp. 71–100, 2003.
- [92] J. Ferber, O. Gutknecht, and F. Michel, “From agents to organizations: an organizational view of multi-agent systems,” in *International workshop on agent-oriented software engineering*, pp. 214–230, Springer, 2003.
- [93] M. Esteva, D. De La Cruz, and C. Sierra, “Islander: an electronic institutions editor,” in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3*, pp. 1045–1052, 2002.
- [94] A. Garcia-Camino, J. A. Rodríguez-Aguilar, and W. Vasconcelos, “A distributed architecture for norm management in multi-agent systems,” in *International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems*, pp. 275–286, Springer, 2007.

- [95] R. M. Turner, “Context-mediated behavior for intelligent agents,” *International Journal of Human-Computer Studies*, vol. 48, no. 3, pp. 307–330, 1998.
- [96] V. Dignum, J.-J. Meyer, and H. Weigand, “Towards an organizational model for agent societies using contracts,” in *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, pp. 694–695, 2002.
- [97] B. Gâteau, *Modélisation et Supervision d’Institutions Multi-Agents*. PhD thesis, Ecole Nationale Supérieure des Mines de Saint-Etienne, 2007.
- [98] S. Mansour, *Un modèle de gestion distribuée de groupes ouverts et dynamiques d’agents mobiles*. PhD thesis, Université Montpellier II, 2007.
- [99] G. Cabri, L. Leonardi, and F. Zambonelli, “Brain: a framework for flexible role-based interactions in multiagent systems,” in *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Proceedings*, pp. 145–161, Springer, 2003.
- [100] H. L. Cardoso and E. Oliveira, “Virtual enterprise normative framework within electronic institutions,” in *International Workshop on Engineering Societies in the Agents World*, pp. 14–32, Springer, 2004.
- [101] A. Colman and J. Han, “Organizational roles and players,” 2005.
- [102] E. Argente, V. Botti, C. Carrascosa, A. Giret, V. Julián, and M. Rebollo, “An abstract architecture for virtual organizations: the thomas project,” tech. rep., Technical report, DSIC, Universidad Politécnica de Valencia, 2008.
- [103] O. Gutknecht and J. Ferber, “Madkit: A generic multi-agent platform,” in *Proceedings of the fourth international conference on Autonomous agents*, pp. 78–79, 2000.
- [104] J. F. Hübner, J. S. Sichman, and O. Boissier, “: a middleware for developing organised multi-agent systems,” in *International Conference on Autonomous Agents and Multiagent Systems*, pp. 64–77, Springer, 2005.
- [105] M. Esteva, B. Rosell, J. A. Rodriguez-Aguilar, and J. L. Arcos, “Ameli: An agent-based middleware for electronic institutions,” in *AAMAS*, vol. 4, pp. 236–243, 2004.
- [106] R. Kitio, O. Boissier, J. F. Hübner, and A. Ricci, “Organisational artifacts and agents for open multi-agent organisations: “giving the power back to the agents”,” in *International Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems*, pp. 171–186, Springer, 2007.
- [107] O. Boissier, R. Kitio, J. F. Hübner, A. Ricci, *et al.*, “Instrumentation d’organisations multi-agents avec des artefact organisationnels,” in *Actes des Journées Franco-phones sur les Systèmes Multi-Agents (JFSMA’08)*, pp. 107–116, sl, 2008.

-
- [108] C. Carabelea, *Raisonner sur l'autonomie d'un agent au sein de systèmes multi-agents ouverts: une approche basée sur les relations de pouvoir*. PhD thesis, Ecole Nationale Supérieure des Mines de Saint-Etienne, 2007.
- [109] B. M. Balachandran and M. Enkhsaikhan, "Developing multi-agent e-commerce applications with jade," in *Knowledge-Based Intelligent Information and Engineering Systems: 11th International Conference, KES 2007, XVII Italian Workshop on Neural Networks, Vietri sul Mare, Italy, September 12-14, 2007, Proceedings, Part III 11*, pp. 941–949, Springer, 2007.
- [110] M. Dastani, V. Dignum, and F. Dignum, "Role-assignment in open agent societies," in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 489–496, 2003.
- [111] T. M. Mitchell, *The discipline of machine learning*, vol. 9. Carnegie Mellon University, School of Computer Science, Machine Learning . . . , 2006.
- [112] A. Cornuéjols and L. Miclet, *Apprentissage artificiel: concepts et algorithmes*. Editions Eyrolles, 2011.
- [113] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson, 2016.
- [114] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer, "Knn model-based approach in classification," in *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Proceedings*, pp. 986–996, Springer, 2003.
- [115] D. Coomans and D. L. Massart, "Alternative k-nearest neighbour rules in supervised pattern recognition: Part 1. k-nearest neighbour classification by using alternative voting rules," *Analytica Chimica Acta*, vol. 136, pp. 15–27, 1982.
- [116] L. Breiman, *Classification and regression trees*. Routledge, 2017.
- [117] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, pp. 81–106, 1986.
- [118] T. M. Cover, "Universal portfolios," *Mathematical finance*, vol. 1, no. 1, pp. 1–29, 1991.
- [119] J. Oliver, *Decision graphs: an extension of decision trees*. Citeseer, 1992.
- [120] S. Lomax and S. Vadera, "A survey of cost-sensitive decision tree induction algorithms," *ACM Computing Surveys (CSUR)*, vol. 45, no. 2, pp. 1–35, 2013.
- [121] N. Patel and S. Upadhyay, "Study of various decision tree pruning methods with their empirical comparison in weka," *International journal of computer applications*, vol. 60, no. 12, 2012.

- [122] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 2013.
- [123] H. W. Kuhn and A. W. Tucker, “Nonlinear programming,” in *Traces and emergence of nonlinear programming*, pp. 247–258, Springer, 2013.
- [124] B. Schölkopf, “Learning with kernels: support vector machines, regularization, optimization, and beyond,” 2002.
- [125] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.
- [126] Y. Jin, “A comprehensive survey of fitness approximation in evolutionary computation,” *Soft computing*, vol. 9, no. 1, pp. 3–12, 2005.
- [127] V. Filipović, “Fine-grained tournament selection operator in genetic algorithms,” *Computing and Informatics*, vol. 22, no. 2, pp. 143–161, 2003.
- [128] S. A. Kauffman and R. G. Smith, “Adaptive automata based on darwinian selection,” *Physica D: Nonlinear Phenomena*, vol. 22, no. 1-3, pp. 68–82, 1986.
- [129] R. I. McKay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O’neill, “Grammar-based genetic programming: a survey,” *Genetic Programming and Evolvable Machines*, vol. 11, pp. 365–396, 2010.
- [130] O. Kramer, “Evolutionary self-adaptation: a survey of operators and strategy parameters,” *Evolutionary Intelligence*, vol. 3, pp. 51–65, 2010.
- [131] A. Auger and N. Hansen, “A restart cma evolution strategy with increasing population size,” in *2005 IEEE congress on evolutionary computation*, vol. 2, pp. 1769–1776, IEEE, 2005.
- [132] H.-S. Park and C.-H. Jun, “A simple and fast algorithm for k-medoids clustering,” *Expert systems with applications*, vol. 36, no. 2, pp. 3336–3341, 2009.
- [133] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.
- [134] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.
- [135] E. Fiesler and R. Beale, “Neural network topologies,” *The Handbook of Neural Computation*, E. Fiesler and R. Beale (Editors-in-Chief), Oxford University Press and IOP Publishing, 1996.
- [136] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [137] M. Lukoševičius and H. Jaeger, “Reservoir computing approaches to recurrent neural network training,” *Computer science review*, vol. 3, no. 3, pp. 127–149, 2009.

-
- [138] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, no. 34, p. 13, 2001.
- [139] W. Maass, T. Natschlager, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural computation*, vol. 14, no. 11, pp. 2531–2560, 2002.
- [140] D. Floreano, P. Durr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary intelligence*, vol. 1, pp. 47–62, 2008.
- [141] T. Griffiths and A. Yuille, "A primer on probabilistic inference," *The probabilistic mind: Prospects for Bayesian cognitive science*, pp. 33–57, 2008.
- [142] J. Pearl, "Bayesian networks: A model of self-activated memory for evidential reasoning," in *Proceedings of the 7th conference of the Cognitive Science Society, University of California, Irvine, CA, USA*, pp. 15–17, 1985.
- [143] I. Ben-Gal, *Bayesian Networks*. 03 2008.
- [144] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 50, no. 2, pp. 157–194, 1988.
- [145] N. L. Zhang and D. Poole, "Exploiting causal independence in bayesian network inference," *Journal of Artificial Intelligence Research*, vol. 5, pp. 301–328, 1996.
- [146] G. Altekari, S. Dwarkadas, J. P. Huelsenbeck, and F. Ronquist, "Parallel metropolis coupled markov chain monte carlo for bayesian phylogenetic inference," *Bioinformatics*, vol. 20, no. 3, pp. 407–415, 2004.
- [147] D. Heckerman, "A tutorial on learning with bayesian networks," *Learning in graphical models*, pp. 301–354, 1998.
- [148] G. Elidan, I. Nachman, and N. Friedman, " " ideal parent " structure learning for continuous variable bayesian networks.," *Journal of Machine Learning Research*, vol. 8, no. 8, 2007.
- [149] H. Steck, *Constraint-based structural learning in Bayesian networks using finite data sets*. PhD thesis, Technische Universitat Munchen, 2001.
- [150] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artificial intelligence review*, vol. 18, pp. 77–95, 2002.
- [151] D. H. Wolpert, "Stacked generalization," *Neural networks*, vol. 5, no. 2, pp. 241–259, 1992.
- [152] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, pp. 123–140, 1996.
- [153] R. E. Schapire, "The strength of weak learnability," *Machine learning*, vol. 5, pp. 197–227, 1990.

- [154] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.
- [155] Y. Freund, “An adaptive version of the boost by majority algorithm,” in *Proceedings of the twelfth annual conference on Computational learning theory*, pp. 102–113, 1999.
- [156] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, “The elements of statistical learning: data mining, inference and prediction,” *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.
- [157] T. Hastie, S. Rosset, J. Zhu, and H. Zou, “Multi-class adaboost,” *Statistics and its Interface*, vol. 2, no. 3, pp. 349–360, 2009.
- [158] C. Kaynak and E. Alpaydin, “Multistage cascading of multiple classifiers: One man’s noise is another man’s data,” in *ICML*, pp. 455–462, Citeseer, 2000.
- [159] J. F. Hübner, O. Boissier, and R. H. Bordini, “Normative programming for organisation management infrastructures,” *Multi-Agent Logics, Languages, and Organisations*, p. 77, 2009.
- [160] O. Boissier, R. H. Bordini, J. F. Hübner, and A. Ricci, “Dimensions in programming multi-agent systems,” *The Knowledge Engineering Review*, vol. 34, p. e2, 2019.
- [161] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi, “Multi-agent oriented programming with jacamo,” *Science of Computer Programming*, vol. 78, no. 6, pp. 747–761, 2013.
- [162] A. Ricci, M. Piunti, M. Viroli, and A. Omicini, “Environment programming in cartago,” *Multi-agent programming: Languages, tools and applications*, pp. 259–288, 2009.
- [163] J. F. Hubner, J. S. Sichman, and O. Boissier, “Developing organised multiagent systems using the noise+ model: programming issues at the system and agent levels,” *International Journal of Agent-Oriented Software Engineering*, vol. 1, no. 3-4, pp. 370–395, 2007.
- [164] A. Ricci, A. Omicini, M. Viroli, L. Gardelli, and E. Oliva, “Cognitive stigmergy: Towards a framework based on agents and artifacts,” in *Environments for Multi-Agent Systems III: Third International Workshop, E4MAS 2006, Hakodate, Japan, May 8, 2006, Selected Revised and Invited Papers 3*, pp. 124–140, Springer, 2007.
- [165] D. Weyns, *Environments for Multi-Agent Systems III: Third International Workshop, E4MAS 2006, Hakodate, Japan, May 8, 2006, Selected Revised and Invited Papers*, vol. 4389. Springer Science & Business Media, 2007.
- [166] J. W. Ilana Harrus, “Artificial intelligence and covid-19: Applications and impact assessment,” 2021.

-
- [167] Otoom, Mwaffaq, Otoum, Nesreen, Alzubaidi, Mohammad A., Etoom, Yousef, and Banihani, Rudaina, "An iot-based framework for early identification and monitoring of covid-19 cases," *Biomedical Signal Processing and Control*, 2020.
- [168] T. T. Nguyen, "Artificial intelligence in the battle against coronavirus (COVID-19): A survey and future research directions," *CoRR*, vol. abs/2008.07343, 2020.
- [169] R. M. Carrillo-Larco and M. Castillo-Cara, "Using country-level variables to classify countries according to the number of confirmed covid-19 cases: An unsupervised machine learning approach," *Wellcome Open Research*, vol. 5, 2020.
- [170] V. Janko, G. Slapničar, E. Dovgan, N. Reščič, T. Kolenik, M. Gjoreski, M. Smerkol, M. Gams, and M. Luštrek, "Machine learning for analyzing non-countermeasure factors affecting early spread of covid-19," *International Journal of Environmental Research and Public Health*, vol. 18, no. 13, p. 6750, 2021.
- [171] D. Isern and A. Moreno, "A systematic literature review of agents applied in health-care," *Journal of medical systems*, vol. 40, no. 2, pp. 1–14, 2016.
- [172] G. Lanzola, L. Gatti, S. Falasconi, and M. Stefanelli, "A framework for building cooperative software agents in medical applications," *Artificial intelligence in medicine*, vol. 16, no. 3, pp. 223–249, 1999.
- [173] J. M. Corchado, J. Bajo, Y. de Paz, and D. I. Tapia, "Intelligent environment for monitoring alzheimer patients, agent technology for health care," vol. 44, no. 2, 2008.
- [174] M. González Bedia, J. M. Corchado Rodríguez, *et al.*, "A planning strategy based on variational calculus for deliberative agents," 2002.
- [175] A. Srivastava, E. H. S. Han, V. Kumar, and V. Singh, "Parallel formulations of decision-tree classification algorithms," in *Proceedings of the International Conference on Parallel Processing*, pp. 237–244, Springer, 1998.
- [176] S. Orlando, P. Palmerini, R. Perego, and F. Silvestri, "Scheduling high performance data mining tasks on a data grid environment," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2400, pp. 375–384, Springer, 2002.
- [177] K. Ali, "On explaining degree of error reduction due to combining multiple decision trees," in *Integrating Multiple Learning Model Workshop, AAAI-96, Portland, Oregon*, 1996.
- [178] F. Provost and V. Kolluri, "A survey of methods for scaling up inductive algorithms," *Data Mining and Knowledge Discovery*, vol. 3, no. 2, pp. 131–169, 1999.
- [179] C. Mastroianni, D. Talia, and P. Trunfio, "Managing heterogeneous resources in data mining applications on grids using XML-based metadata," in *Proceedings - International Parallel and Distributed Processing Symposium, IPDPS 2003*, pp. 11—pp, IEEE, 2003.

- [180] A. Murdopo *et al.*, “Distributed decision tree learning for mining big data streams,” *Master of Science Thesis, European Master in Distributed Computing*, 2013.
- [181] R. Caruana and A. Niculescu-Mizil, “An empirical comparison of supervised learning algorithms,” in *Proceedings of the 23rd international conference on Machine learning*, pp. 161–168, 2006.
- [182] P. Domingos and G. Hulten, “Mining high-speed data streams,” in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 71–80, 2000.
- [183] W. Hoeffding, “Probability Inequalities for Sums of Bounded Random Variables,” in *Journal of the American Statistical Association*, vol. 58, pp. 13–30, Springer, 1963.
- [184] H. Yang and S. Fong, “Incremental optimization mechanism for constructing a decision tree in data stream mining,” *Mathematical Problems in Engineering*, vol. 2013, 2013.
- [185] A. Das, J. Wang, S. M. Gandhi, J. Lee, W. Wang, and C. Zaniolo, “Learn smart with less: Building better online decision trees with fewer training examples,” in *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2019-Augus, pp. 2209–2215, 2019.
- [186] R. Jin and G. Agrawal, “Communication and Memory Efficient Parallel Decision Tree Construction,” in *Proceedings of the 2003 SIAM International Conference on Data Mining*, pp. 119–129, SIAM, 2003.
- [187] A. Bifet and R. Gavalda, “Learning from time-changing data with adaptive windowing,” in *Proceedings of the 2007 SIAM international conference on data mining*, pp. 443–448, SIAM, 2007.
- [188] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation,” *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.
- [189] A. Liaw and M. Wiener, “Classification and Regression by randomForest,” *R news*, vol. 2, no. December, pp. 18–22, 2002.
- [190] R. G. Smith, “Correction to “The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver”,” *IEEE Transactions on Computers*, vol. C-30, no. 5, p. 372, 1981.
- [191] R. Davis and R. G. Smith, “Negotiation as a metaphor for distributed problem solving,” *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, vol. 2650, no. 1, pp. 51–97, 2003.
- [192] A. Omicini, A. Ricci, and M. Viroli, “Artifacts in the A&A meta-model for multi-agent systems,” *Autonomous agents and multi-agent systems*, vol. 17, no. 3, pp. 432–456, 2008.
- [193] H. Hari, “Symptoms and covid presence (may 2020 data),” Aug 2020.

- [194] W. The Departments of Computer Science, “Weka 3: Machine learning software in java,” 2002.
- [195] Nshomron, “Nshomron/covidpred: Machine learning-based prediction of covid-19 diagnosis based on symptoms.”
- [196] B. Hungund, “Covid-19 symptoms checker,” Mar 2020.
- [197] A. Siam and R. Maamri, “A Category-theoretic Approach to Organization-based Modeling of Multi Agent Systems on the Basis of Collective Phenomena and Organizations in Human Societies,” *Informatica (Slovenia)*, vol. 42, no. 4, pp. 563–576, 2018.
- [198] S. M. Cheema, R. Shafiq, S. Saleem, S. Z. H. Shah, and A. Baqir, “Multi-agent oriented framework for university course timetable scheduling,” in *Intelligent Technologies and Applications: Second International Conference, INTAP 2019, Bahawalpur, Pakistan, November 6–8, 2019, Revised Selected Papers 2*, pp. 791–799, Springer, 2020.
- [199] D. Klimov, M. Rabovskaya, V. Kozlov, and A. Syskov, “Timetable model for self regulated learning based on student agent level restrictions resolving,” in *2021 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBREIT)*, pp. 0383–0386, IEEE, 2021.
- [200] Z. Houhamdi, B. Athamena, R. Abuzaineddin, and M. Muhairat, “A multi-agent system for course timetable generation,” *TEM Journal*, vol. 8, no. 1, p. 211, 2019.
- [201] A. Syskov, M. Rabovskaya, and V. Makhneva, “Timetable model for individualization of education based on multi-agent approach,” in *2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBREIT)*, pp. 467–470, IEEE, 2020.
- [202] K. Y. Junn, J. H. Obit, R. Alfred, and J. Bolongkikit, “A formal model of multi-agent system for university course timetabling problems,” in *Computational Science and Technology: 5th ICCST 2018, Kota Kinabalu, Malaysia, 29-30 August 2018*, pp. 215–225, Springer, 2019.
- [203] J. Nourmohammadi-Khiarak, Y. Zamani-Harghalani, and M.-R. Feizi-Derakhshi, “Combined multi-agent method to control inter-department common events collision for university courses timetabling,” *Journal of Intelligent Systems*, vol. 29, no. 1, pp. 110–126, 2020.
- [204] J. Henry-Obit, D. Landa-Silva, D. Ouelhadj, T. Khan-Vun, and A. Rayner, “Designing a multi-agent approach system for distributed course timetabling,” in *11th International Conference on Hybrid Intelligent Systems (HIS)*. IEEE, 2011.
- [205] V. Sharmila, A. Rajivkannan, M. Venkatesan, S. Sangeetha, S. Sivani, and V. Sumukhi, “Dynamic timetable scheduling using multi-agent systems and federated learning,” in *International Conference on Sustainability Innovation in Computing and Engineering (ICSICE 2024)*, pp. 1485–1499, Atlantis Press, 2025.

- [206] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, and A. Santi, “Multi-agent oriented programming with jacamo,” *Science of Computer Programming*, vol. 78, no. 6, pp. 747–761, 2013.