UNIVERSITÀ DEGLI STUDI DI PALERMO

Dottorato in Ingegneria dell'Innovazione Tecnologica
Dipartimento di Ingegneria
SSD: ING-INF/05

Human-Robot Teaming Interaction: a Cognitive Architecture Solution

IL DOTTORE
FRANCESCO LANZA

IL COORDINATORE
CH.MO PROF. SALVATORE GAGLIO

IL TUTOR
CH.MO PROF. SALVATORE GAGLIO

CO TUTOR
CH.MO PROF. ANTONIO CHELLA
DOTT. ING. VALERIA SEIDITA

CICLO XXXIII
ANNO CONSEGUIMENTO TITOLO 2021

To my parents, my brother and my whole family.

Abstract

The relationship between artificial intelligence and human work from a sociological perspective leads to reflections on the future evolution of employment. People keep wondering if and when robots will replace us, taking our place even in those social environments that have always characterized our society, such as work. To the big question "*will robots replace humans?*" the answer is still uncertain, but what is certain is imagining a society where robots work alongside human beings. Today, it is also possible to think of artificial systems able to perform specific tasks faster and more efficiently than humans. Generally, society tends to imagine a robot as a machine with particular features and abilities, able to perform tasks on behalf of human beings. This vision of a robot *slave* of a man has gone and can go well, only in particular social and or work contexts, where the robot figure can be considered functional to the task to be performed, that is a valid alternative to the man himself. Unfortunately or fortunately, today more than ever, society needs robots that can work *with* humans and not *for* humans. The difference between the prepositions *for* and *with* is the turning point for this new industrial revolution, which invests our society making it an active part in this process of evolution/transition towards a social model that is forced to evolve together with scientific progress. The ambition of researchers, scientists, and engineers is to develop a new paradigm of interaction between man and machine that considers the process of integration of the robot in the working class society.

Born as a branch of the most common Human-Robot Interaction, Human-Robot Teaming Interaction is the research field that deals with the study of robotic systems able to cooperate and collaborate with humans. In this specific field of research, the research conducted in these years of the doctorate is placed. This dissertation proposes a cognitive architecture as a solution for implementing Human-Robot Teaming Interaction applications. The cognitive architecture was modeled and developed on the multi-agent system paradigm, implementing self-awareness capabilities for deliberating and performing actions. The proposed cognitive architecture can manage collaborative interactions between humans and robots in potentially unknown and dynamic environments. The objective was to generate a functional and straightforward cognitive architecture that managed robotic platforms and based on psychological theories derived from the study of human behavior. Fundamental characteristics of the proposed architecture are the ability to dynamically manage knowledge and compose new possible solutions at runtime to handle situations not foreseen at design-time by the programmer under the assumption of limited knowledge and resources. The intention is to endow a machine with innate abilities that reproduce the human behaviors without neglecting aspects of trust and reliability, typical of human beings that work in a team. The proposed cognitive architecture exploits the practical reasoning cycle, regular of the Beliefs-Desires-Intentions (BDI) model, in conjunction with other methods for handling knowledge, composing plans at runtime, and changing the way to select intentions for the agent through a mechanism of anticipation. These abilities were implemented through machine learning techniques for handling knowledge, Artificial General Intelligence (AGI) approach for composing plans at runtime, and extending the multi-agent system's reasoning cycle with the anticipation process. Also, to make transparent and trustworthy interactions between humans and robots, a model based on the trust theory was implemented to justify the robot's decision-making results. The justification process enables the robot to justify itself when something wrong occurs or cannot perform some operations. The cognitive architecture was developed using Jason and the CArtAgO frameworks. Jason is the framework for developing a multi-agent system based on the Beliefs-Desires-Intentions model, and CArtAgO is the framework for employing the Agent-Artifact meta-model into the multi-agent system (the cognitive architecture). Thus, the realized cognitive architecture extends and exploits the Jason reasoning cycle introducing a new situation-awareness method. In this way, it was possible to realize an architecture with all the strength of a BDI model and the other techniques used to realize architecture's modules. Results of this dissertation concern the creation of a new cognitive architecture that uses some technologies never used before in the Human-Robot Teaming Interaction (HRTI) research field.

Acknowledgments

First, I would like to sincerely thank Prof. Salvatore Gaglio for being my supervisor.

I wish to express my deeply-felt gratitude to Prof. Antonio Chella for everything he has done for me, for supporting me in every moment and circumstance. I will be forever in debt.

I have to deeply thank my mentor, Prof. Valeria Seidita, who in every moment has been close to me, constantly enduring and supporting me, passing on all her knowledge and willpower, inspiring and encouraging me day after day, and teaching me that in this world I must never give up and that probably the next step is the decisive one to reach the goal. Without her dedication and hard work, none of this could have been possible.

I wish to express my gratitude to my colleagues, my academic family, that have accompanied and supported me during this long and precious journey. I shared joy and sorrows, anxieties, and worries with them and a lot of satisfaction and happiness. In particular, I want to thank my sidekicks, Dr. Giuseppe Santaromita, Dr. Vito Monteleone, and Dr. Francesco Gugliuzza, for their support and their infinite patience and availability.

Thanks also to Prof. Orazio Gambino for the magnificent cappuccino breakfasts and his amazing cakes or plumcakes, but especially thanks for the tips and advice that he gave me.

Thanks to my British family, with whom I spent six indescribable months. A special thanks go to my main supporters, Dr. Jacopo de Berardinis and Dr. Gabriella Pizzuto. They made the experience in Manchester unforgettable.

And finally, thanks to my family who allowed all this. Without them, I would not be who I am today. Thanks to my father for the enormous sacrifices and the efforts made to fulfill my dream, thanks to my mother, the fixed reference point in my life, their advice has been precious to me. A thank you goes to my brother, “because of him” I fell in love with the fantastic world of computer science.

Finally, other special thanks go to my late grandfather *Francesco* that continues to protect and observe me. I am sure that he would have been happy and proud of the goals I have achieved if he had been here today.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Thanks to the maturity reached by the Artificial Intelligence (AI) today, it is possible to have more and more autonomous and intelligent machines. In the last few years, we assisted in a rapid increase in robots' deployment for filling roles previously assigned to human beings. In the past, robotics was seen as an engineering field apt to build machines able to substitute humans in heavy works to optimize industrial production and lighten the workers' workload.

Nowadays, industries, governments, and society reclaim robots able to satisfy more general and complex tasks. For instance, it is already possible to see robots deployed in domains in charge of performing operations hitherto entrusted to humans. It is the case of robots employed as caregivers or operators to the services of promotion and reception.

Hence, service robots have to satisfy users through interactive capabilities that involve language, sight, and those human characteristics relevant for fruitful interactions. Even if these robots perform jobs in place of humans, robots currently work `for` them and not `with` them. The concept of work `with` humans is at the base of the Human-Robot Teaming Interaction (HRTI) paradigm.

For doing that, the robot has to reproduce human behavior. Studying human-human interaction is necessary for understanding which abilities an *artificial system*, such as a robot, has to show.

In human-human interaction, a team member, first and foremost, knows the team's overall goal. He knows what he wants to do. Hence, he intentionally decides which purpose to commit. He is also aware of what he can do, i.e., his capabilities; accordingly, he selects the goals he can reach and the corresponding plans of actions.

A team member is also aware of the surrounding environment: he can associate any new element in the situation with the content of his knowledge base. A team member owns a knowledge base that includes a large number of items. He must select only items relevant to the context in which he is operating. A team member can communicate with the other teammates to update his knowledge base on the environment and explicitly or implicitly delegates actions to other mates; he asks how to do something he cannot do. He observes what other members are doing and, if the case, he anticipates actions or cooperates with other teammates. He sees what the other teammates are doing, decides what to do, and whether to do something based on his emotional and stress state, on the trust level he has in the other teammates and himself. A team member understands if the operative condition for pursuing actual objectives changes. In this case, he can re-plan or create a new plan from experience. He can explain to the other team members what he is doing and why and, if the case, why he cannot do something. Finally, he learns from experiences and stores all the information about the continuously changing world.

Challenges, in this domain, concern two aspects of a generic *intelligent system* that have to work in a team, the reasoning system and knowledge management. Other elements influence human-robot teaming interaction, such as establishing a level of trust in others, establishing a level of cooperation among team components, and feeling emotions, producing mental states.

In this domain, so, a reasoning system has to support abilities for realizing cognitive skills, such as the imaginary, self-modeling, theory of mind, and decision-making processes for supporting runtime planning and the capability to compose plans at runtime without the supervision of a programmer. The knowledge management system has to model the abilities to acquire and organize knowledge. In this context, knowledge representation, cognitive linguistics, and techniques of natural language processing and description logic could help face the challenges.

This dissertation aims to contribute to human-robot teaming interaction systems proposing a solution based on the development of cognitive architecture. Since the domain is complex and not easy to face, the architecture should offer the most suitable combination of reliability, feasibility, and robustness to simplify all mechanisms behind the development of intelligent systems, such as robots, for Human-Robot Teaming

Interaction.

1.1 Motivations and Goals

In the past years, robots were used as instruments to perform heavy tasks in place of human beings. Today, the interdependence between society and technology boosts the robot's concept to a higher level: to be more than a simple replacement. Designing a system for HRTI requires the study and application of psychological and scientific theories to produce computational models that are reliable both from a scientific and social point of view [191]. Thus, implementing HRTI applications has to start analyzing concepts and components of the human-robot teaming domain [138]. The idea behind this dissertation's development is related to what a team is and how it is possible to build machines that act as team-member.

Creating human-robot teams means realizing that hardware and software systems can adapt to the context to evolve with the advancing technology. In general, robots struggle with humans cause they do not own enhanced and sophisticated capabilities to let operation in real-time. Another big deal is the dialogue that could produce incorrect responses making interaction jittery. These issues make the interaction poor, unnatural, and inefficient, compromising the team's success. As authors said in [138],

> *High-performing teams operate with dynamic skillsets, including anticipation and prediction, to handle more complicated scenarios and workflow.*

So it is fundamental to endow robots with skills for acting in dynamic and changing environments, bridging the gap between the hardware/software resources and the theories of Human-Robot Teaming Interaction (HRTI) as well as paying attention *on teams, teamwork, and the team performance* [106, 180].

Typically, teammates share the same objectives and the same knowledge of the environment. They know the goal and have a plan to reach it, collaborating and cooperating with the others. Collaboration and cooperation amount to learning their abilities and others' ones, interacting each time something is unknown or is going wrong, or each time a task has to be delegated to another or adopted by another.

Also, collaboration and cooperation imply explaining why an action/task failed or was not performed. Moreover, the teammate may anticipate another member's actions due to continuing observing the other mate and the environment. Knowledge of the domain is continually updated to let teammates re-plan or create new plans if necessary. Knowledge of the environment also includes developing a model of the self, the inner state of the teammate changes over time and influences, and the general knowledge of the environment, the goal to pursue, and the decision-making process.

Generalizing, features of human-robot team systems require to investigate and analyze *(i)* knowledge acquisition, representation and updating, including memory management, *(ii)* environment representation, inner and outer one, *(iii)* plans selection or creation *(iv)* learning, *(v)* introspection, for allowing team members to be aware of himself and his capabilities.

To better explain the motivation of this dissertation, three scenarios are described.

In the first case, the working environment is known in advance, and a human-robot team has to carry out a task. It is static, and it does not change at runtime. This situation may be faced using standard approaches, well known and described in the literature, such as reasoning systems based on the Beliefs-Desires-Intentions (BDI) logic [168] and its implementation using Jason [27].

The second case foresees a robot that is not fully able to act autonomously by physical limitations or environmental constraints. This situation implies an intervention of the human under an explicit request of the robot. It is a collaborative work. Even this case, if properly treated, could be faced using the BDI logic and the Jason framework, the CArtAgO meta-model [171] could be added to improve results.

Whilst, the third case is more complex. This time, the robot partially know the environment, and probably at design time, not all plans and knowledge were given for handling occurring situations. Besides, the teammate could or could not face every unexpected event that the environment could produce. Thus, the interaction between humans and robots could be affected, as well as the team's success. This fact happens when interactions bring out new terms of operability that must be worked out to decide what action to take. Generally, when a team is made up of only humans, they choose actions from their experience, the knowledge they have of the other team member, the trust they place in the other team, their emotional state, or the anticipation of others' actions.

This dissertation proposes a solution for building robots to operate in a team even when some conditions lack. The solution is based on a cognitive architecture, designed and implemented for robotic purposes.

Before starting to develop a novel architecture, existing ones were studied. Each architecture shows a predominant and reliable ability for a context and not for another, so it is important to study them to understand their capabilities and limits.

A generic cognitive architecture is based on a cognitive model that implements a reasoning cycle. There is no common definition of what or whatnot is a cognitive architecture; the definition lacks a clear and sharp formulation. Making my own words in [132]:

> *The term "Cognitive Architectures" indicates both abstract models of cognition in natural and artificial agents, and the software instantiations of such models which are then employed in the field of Artificial Intelligence (AI).*

Most cognitive architectures are based on agent [74, 209, 124].

Differentiation of these cognitive architectures could be done looking at their characteristics, that in general are the memory organization and the type of learning [57].

The proposed cognitive architecture is based on a multi-agent system with centralized memory management and capabilities for learning plans at runtime as well as anticipating actions' results. Besides, trust theories and developmental models were analyzed to introduce the justification process and a system for measuring teammates' trust. These last contributes to making the decision-making process transparent and explainable.

1.2 A Cognitive Architecture for Human-Robot Teaming Interaction

In recent years, various cognitive architectures have been studied and developed. They served to explore the main characteristics of the human mind.

Figure 1.1 represents the cognitive architecture, the result of my thesis work, developed for supporting roboticists in writing application for Human-Robot Teaming Interaction (HRTI). The architecture was designed following the guidelines defined into the standard model of mind by J. Laird et al. [122].

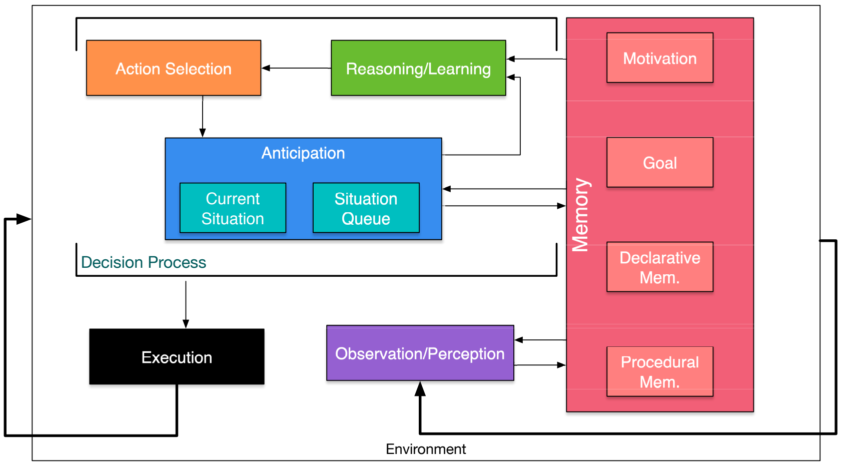


Figure 1.1: The cognitive architecture for Human-Robot Teaming Interaction [44].

Principal novelties and differences in this cognitive architecture are the ability to anticipate results and composing high-level plans at runtime under the assumption of insufficient knowledge and resources [205]. In addition, models for making transparent and explainable the reasoning system were realized. Besides, aspects of knowledge management, such as designing knowledge models and realizing a computational method for autonomously expanding knowledge at runtime, were given. Self-awareness was developed using standard cognitive approaches, well known in the literature, and employing mainly the practical reasoning system [28], typical of *beliefs-desires-intentions* agents [215].

The theoretical framework underlying the development of architecture is the Global Workspace Theory (GWT) theoretical framework, introduced by Baars [13, 15, 16, 17], and implemented in the LIDA cognitive system [186]. Thus, the proposed architecture follows the guidelines of Laird et al. [122] and exploits the GWT model and takes inspiration from LIDA.

The starting point is the generalization of all the characteristics of the cognitive architectures studied that led the representation in Figure 1.1. Before going into the details of the proposed architecture, discussed in the next chapters, it is worth to make a hint to some necessary starting hypotheses.

A team made up of humans and robots is a complex system. The global behavior and characteristics of the system emerge from interactions. The behavior of a complex system cannot be analyzed and implemented as *the sum of the single components, the* *system must be seen as a whole*. The cognitive architecture in Figure 1.1 is developed employing the Jason and CArtAgO framework. Thus, the cognitive architecture is based on the Beliefs-Desires-Intentions (BDI) agent model. In our vision, a robot may be managed by a set of agents that cooperate and collaborate to reach a common and shared objective.

1.3 Research Objectives and Questions: Hypothesis and Claims

Today we are invited to take part in an important appointment with history, a turning point for our society that sees intelligent machines and technologies, the future, and innovation useful to improve the living conditions of today's society and increase the well-being of the individual. A further technological and industrial revolution is underway. It is changing and will probably change our habits, replacing our current lifestyle in favor of a less demanding and tiring model.

All this will be thanks to the correct use of robots and artificial intelligence algorithms to allow these machines to operate correctly in our society to achieve a better world.

Researchers and scientists worldwide are working to make it possible, developing robotic platforms and algorithms that will blend and integrate, adequately, in every social tissue of our society.

This research work aims to endow programmers to develop a cognitive system that is more than a reasoner, like most cognitive architectures, since components of the system are written to be hosted by robots. Components integrate main important capabilities inherited from computer vision and speech recognition, or techniques for scheduling tasks.

Figure 1.2 shows an organization chart of this dissertation. The work done for pursuing the objective is grounded on two important pillars of Artificial Intelligence:

- Human-Robot Teaming Interaction (HRTI) — This research area is a fork of the most known Human-Robot Interaction (HRI). This research aims to realize a system able to interact with people during cooperative and collaborative tasks, involving their capabilities or showing proactive skills. For implementing an HRTI

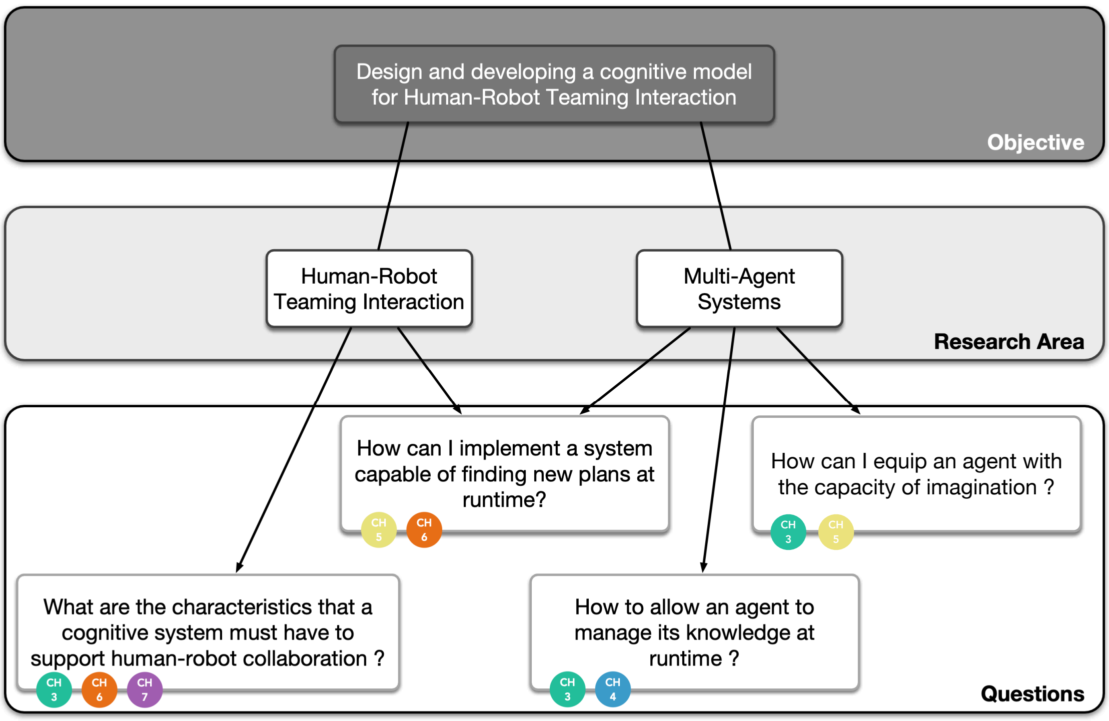

Design and developing a cognitive model for Human-Robot Teaming Interaction
Objective
Human-Robot Teaming Interaction
Multi-Agent Systems
Research Area
How can I implement a system capable of finding new plans at runtime?
CH 5
CH 6
How can I equip an agent with the capacity of imagination ?
CH 3
CH 5
What are the characteristics that a cognitive system must have to support human-robot collaboration ?
CH 3
CH 6
CH 7
How to allow an agent to manage its knowledge at runtime ?
CH 3
CH 4
Questions

Figure 1.2: The Thesis Outline.

system is necessary to move toward several disciplines from engineering to psychological subjects. The HRTI tries to model the teaming context's social behavior for building autonomous robots that work with humans and not for humans.

- Multi-Agent System (MAS) — This technological paradigm was born for handling complex systems and solving complex problems. The multi-agent paradigm uses a multi-agent system to design and develop software architecture on which agents communicate with each other to solve tasks. Implementing a multi-agent software architecture, a field of software engineering Agent-Oriented Software Engineering (AOSE), supports developers and programmers to project the architecture following rules and methodologies. Moreover, the multi-agent approach enables programmers to develop a distributed AI system through its native support.

The field of Human-Robot Teaming Interaction (HRTI) involves several research domains; nonetheless, the study of artificial intelligent systems for letting interaction between humans and robots asks for psychological support that helps model mechanisms that drive interaction among people. Before developing a system able to interact with humans, it is mandatory to investigate which behaviors this type of system has to show and which characteristics are essentials. For doing that, I searched which relevant questions are needed to answer to significantly contribute to the human-robot teaming interaction domain.

Here, the list of questions that I want to answer with this dissertation. The following chapters collect the results of the work done in these three years of doctorate.

`Q1` — *What are the characteristics that a cognitive system must have to support human-robot collaboration ?*

`claim` — A cognitive system is an entity in charge of learning and reasoning about facts and goals that an artificial life has to reach during its evolution. Generally, it is deployed onto the physical system for functioning properly. Therefore, it needs physical support that lets the cognitive system acting in the environment through actuators and sensing the surrounding context through sensors. A cognitive system might be synthetic or biological, and it works as humans or animals. Another important characteristic of a cognitive system concerns the ability to learn and adapt to the changing environment; nonetheless, this ability regards non-trivial cognitive systems.

`hypothesis` — The main characteristics of a cognitive system can be extended and adapted to support a complex scenario. Exploiting this paradigm, performing cognitive architectures' functionalities let them follow teams' will, even at runtime.

`Q2` — *How can I implement a system capable of finding new plans at runtime?*

`claim` — The capability for a cognitive system to adapt its behavior for specific environment and context passes from the capability to search and apply the most suitable plan not given at design time.

`hypothesis` — For composing high-level plans at runtime in Jason is necessary to build a computational model that could overcome Jason's limitation without neglecting its nature. For instance, another logical formalism overcomes Jason's limitation and let the agent compose high-level plans at runtime.

`Q3` — *How can I equip an agent with the capacity of imagination ?*

`claim` — Imagination constitutes a trigger for the decision process. It can be realized using theories of self-modeling and theory of mind. Endowing the robot with self-modeling abilities makes it able to test operation before their real performing and anticipate the other team members' action.

`hypothesis` — Equipping a cognitive agent, or better, a robot with this capability requires to use of theories of self-modeling and theory of mind and let the agent to test operations before their real performing and to anticipate the action of the other members of the team.

`Q4` — *How to allow an agent to manage its knowledge at runtime ?*

`claim` — For modeling domain knowledge a robot uses tools for organizing it and retrieve information stored within the knowledge base. The knowledge base is handled using through knowledge management systems.

`hypothesis` — A knowledge management system uses architectural and computational aspects for handling a knowledge base. The cognitive architecture uses an ontology to represent the domain knowledge, and a centralized approach overwhelms the problem of syncing all agents with updated information to plan. Based on natural language processing techniques, a linguistic-computational model organizes and increases the knowledge at runtime in an unsupervised way.

1.3.1 Contributions

The main contributions of the work presented in this dissertation are:

- A cognitive architecture for Human-Robot Teaming Interaction (HRTI);
- A simulator prototype to implement an inner representation of the robot model for realizing self-modeling and anticipation capabilities;
- A knowledge management system for cognitive architecture;
- A method for letting agents composing high-level plans at runtime;

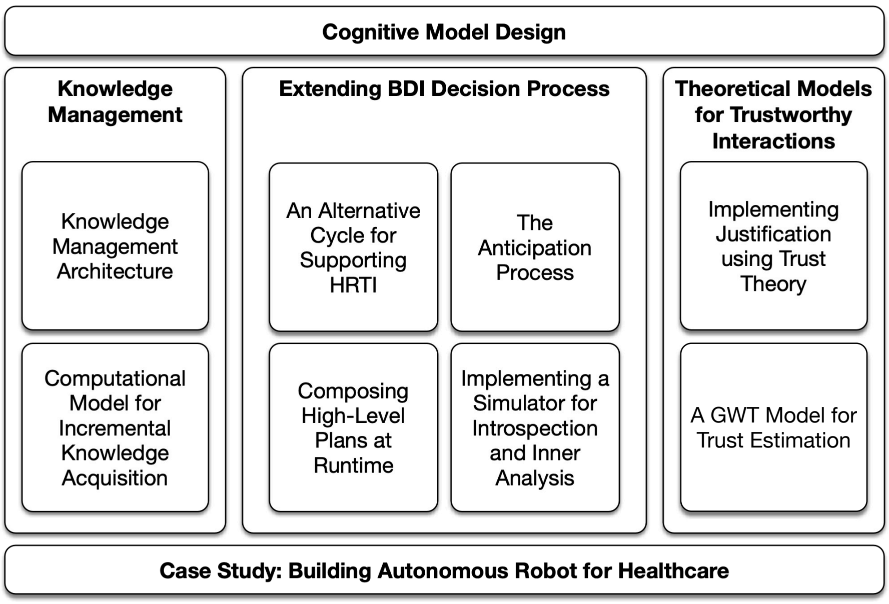

Cognitive Model Design
Knowledge Management
Knowledge Management Architecture
Computational Model for Incremental Knowledge Acquisition
Extending BDI Decision Process
An Alternative Cycle for Supporting HRTI
The Anticipation Process
Composing High-Level Plans at Runtime
Implementing a Simulator for Introspection and Inner Analysis
Theoretical Models for Trustworthy Interactions
Implementing Justification using Trust Theory
A GWT Model for Trust Estimation
Case Study: Building Autonomous Robot for Healthcare

Figure 1.3: An overview of the job done for writing this dissertation, subdivided into three pillars.

- A method for endowing robots with a justification process employing a model of trust;
- A theoretical model for trust estimation;

1.4 Dissertation Outline

The remainder of the dissertation is organized as follows.

Chapter 2 contains the background required for realizing this dissertation and the job done during these years; **Chapter 3** describes the cognitive architecture proposed for implementing Human-Robot Teaming Interaction applications, the BDI extension and the module to anticipate actions; **Chapter 4** is in charge to explain from a double point of view the techniques used for managing knowledge, the first part of the chapter is dedicated to the designing of the knowledge management system and the second part of the chapter introduce a new computational model for knowledge acquisition and upgrading; **Chapter 5** shows the implementation of a portion of the system for letting anticipation works and a computational method used for composing high-level plans at runtime. **Chapter 6** show a case study on which the architecture was used for implementing an application for human-robot interaction in the healthcare domain; **Chapter 7** introduces two theoretical frameworks for enabling trust prediction in the cognitive architecture and in general in human-robot interaction. In **Chapter 8** some conclusions are drawn.

1.5 Publications

Parts of the work in this thesis have been published in

International Conferences and Workshops

- Antonio Chella, **Francesco Lanza**, and Valeria Seidita. "A Cognitive Architecture for Human-Robot Teaming Interaction." AIC 2018 Artificial Intelligence and Cognition 2018 Proceedings of the 6th International Workshop on Artificial Intelligence and Cognition (pp. 82-89). CEUR-WS.

- Antonio Chella, **Francesco Lanza**, and Valeria Seidita. "Decision process in human-agent interaction: extending jason reasoning cycle." International Workshop on Engineering Multi-Agent Systems. Springer, Cham, 2018.

- Antonio Chella, **Francesco Lanza**, and Valeria Seidita. "Representing and Developing Knowledge using Jason, Cartago and OWL." 19th Workshop "From Objects to Agents", WOA 2018. In CEUR Workshop Proceedings (pp.147-152).

- Arianna Pipitone, **Francesco Lanza**, Valeria Seidita and Antonio Chella. "Inner Speech for a Self-Conscious Robot." AAAI Spring Symposium: Towards Conscious AI Systems. Stanford University, USA. 2019.

- Antonio Chella, **Francesco Lanza**, and Valeria Seidita. "Human-agent interaction, the system level using Jason." Proceedings of the 6th International Workshop on Engineering Multi-Agent Systems (EMAS 2018). Stockholm (July, 10-15 2018). 2018.

- Antonio Chella, **Francesco Lanza**, Arianna Pipitone, Valeria Seidita. "Human-Robot Teaming: Perspective on Analysis and Implementation Issues." AIRO@ AI*IA. CEUR Workshop Proceedings. CEUR-WS. 2018.

- Cristiano Castelfranchi, Antonio Chella, Rino Falcone, **Francesco Lanza** and Valeria Seidita. "Endowing robots with self-modeling abilities for trustful human-robot interactions." 20th Workshop "From Objects to Agents", WOA 2019 (2019): In CEUR Workshop Proceedings (pp. 22-28). CEUR-WS.

- **Francesco Lanza**, Samuele Vinanzi, Valeria Seidita, Angelo Cangelosi and Antonio Chella. "A Global Workspace Theory Model for Trust Estimation in Human-Robot Interaction." AIC 2019 Artificial Intelligence and Cognition 2019 Proceedings of the 7th International Workshop on Artificial Intelligence and Cognition. In CEUR Workshop Proceedings (pp. 104-112). CEUR-WS. (2019).

- **Francesco Lanza**, Valeria Seidita, Cristina Diliberto, Paolo Zanardi and Antonio Chella. "Inside the Robot's Mind During Human-Robot Interaction." AIC 2019 Artificial Intelligence and Cognition 2019 Proceedings of the 7th International Workshop on Artificial Intelligence and Cognition. In CEUR Workshop Proceedings (pp. 54-67). CEUR-WS.(2019).

- **Francesco Lanza**, Patrick Hammer, Valeria Seidita, Pei Wang and Antonio Chella. "Agents in dynamic contexts, a system for learning plans." In Proceedings of the ACM Symposium on Applied Computing (pp. 823-825). Association for Computing Machinery. doi: 10.1145/3341105.3374083. 2020.

- Antonio Chella, **Francesco Lanza**, Arianna Pipitone and Valeria Seidita. "The Inner Life of a Robot in Human-Robot Teaming". In 2020 IEEE International Conference on Human-Machine Systems (ICHMS) (pp. 1-4). IEEE.

- Jacopo de Berardinis, Gabriella Pizzuto, **Francesco Lanza**, Antonio Chella, Jorge Meira and Angelo Cangelosi (2020). "At Your Service: Coffee Beans Recommendation From a Robot Assistant". In Proceedings of the 8th International Conference on Human-Agent Interaction (HAI '20). Association for Computing Machinery, New York, NY, USA, 257–259.

International Journals

- Antonio Chella, **Francesco Lanza**, Arianna Pipitone, Valeria Seidita. “Knowledge acquisition through introspection in human-robot cooperation." Biologically inspired cognitive architectures 25 (2018): 1-7.
- **Francesco Lanza**, Valeria Seidita, and Antonio Chella. “Agents and robots for collaborating and supporting physicians in healthcare scenarios.” Journal of Biomedical Informatics (2020): 103483. DOI: https://doi.org/10.1016/j.jbi.2020.103483.
- Valeria Seidita, **Francesco Lanza**, Arianna Pipitone, Antonio Chella. “Robots and Intelligent Assistants to Face COVID-19 Pandemic.” Briefings in Bioinformatics 2020:bbaa361. DOI: https://doi.org/10.1093/bib/bbaa361

Chapter 2

Background

The Human-Robot Teaming Interaction (HRTI) is a research field that crosses over several domains.

In this chapter, the background for contextualizing the work done in this dissertation and the results reached during this journey are presented.

The chapter is organized in order to treat each content necessary to frame the problems and the solutions faced and used to accomplish the goals of this dissertation, accordingly with the state of the art.

2.1 The Role of Robotics into the Society

Cooperative robots are used in various domains: industrial [62], search and rescue [163], space exploration [113], social assistance [32] and healthcare [145, 172] are only a subset of the large number of robotic applications [198].

The manuscript written by Prof. Raj Reddy [169] highlights the potentiality and potential killer applications of robotics in several contexts. He identified several robotics applications that make the difference in term of well being for humanity.

At the moment, we often hear about the role of robotics in society. Discussions and evaluations of the actual use of robotics as a tool for solving critical problems that affect the population are under investigation by international commissions.

International Federation of Robotics (IFR)[1] and the Robotic Industries Association

[1]https://ifr.org/

(RIA)[2] analyzed the market in order to highlight the trend of robotic applications. From a rapid analysis, and searching on the web, currently, society claims for robot able to work side-by-side with humans.

This field, known as Human-Robot Teaming Interaction (HRTI), or also as Human-Robot Teaming (HRT) or yet Human-Robot Team Interaction (HRTI), is an advancement of the research field of human-robot interaction [89] moving the attention on human-robot collaboration.

Human-Robot Teams field studies the interaction among humans and robots during cooperative or collaborative tasks. The basic principle of the HRTI paradigm is to create robots able to work alongside humans. In this domain, research centers, universities, and industries look for solutions for facing the challenges related to this research area.

2.1.1 Human-Robot Teaming Interaction: a Collaborative Approach

The aim of HRTI is building robots that are more than autonomous systems able to execute actions and interact with humans following the guidelines of the HRI domain. A robotic teammate has to be an active member of a team, and not a mere tool for acting. Thus, the standpoint of teamwork for building human-robot teaming interaction applications is shared with the vision in [105], where robot has to work *jointly with others rather than acting upon others*. Hoffman and Breazeal [105] analyzed concepts and factors for working in a team made up of humans and robots, proposing a framework for realizing a humanoid robot envisioned for working with astronauts, *Robonaut*.

Theoretical foundations for the HRTI has to search in the *human-human teaming interaction* counterpart. In fact, humans are good in cooperation and collaborative tasks because they exchange continuously information and are able to infer using their senses all the knowledge required for pursuing an objective in a task. Questions, for building robots able to interact with humans in team that works in human-like fashion, may be searched in the mechanisms of human interaction.

A robot for acting in human-like fashion has to show abilities and cognitive skills used by humans [105]. A collaborative robot should be endowed with these characteristics:

- Shared Activity

[2]https://www.robotics.org/

- Joint Intention
- Common Ground
- Goals

Each of this term implements specific features in a robot. Theories are based on the principles defined by Bratman in [30], the shared cooperative activity. This last lets scientists model the HRTI exploiting the psychological behaviors at the base of the human reasoning.

For identifying concepts and components about human-robot teaming [138], we need to look at the human-human counterpart. As authors highlighted in [138] a teamwork has to endow mainly with the communication, coordination and collaboration components. The first represents the capability to exchange information among team-members. Communication could be driven by robotic skills such as dialogue systems [192, 129], vision-based interaction system [102, 159] or simply exchanging data over the network for remotely cooperative task [125]. The second components, the coordination, is in charge to orchestrate the robot with the aim to let it integrate in the team for creating interactions that are in unison with the other member. Characteristics introduced before are fundamental to implement coordination [105]. Team members has to share the common ground, has to show joint activity capabilities and implements method for predicting mate's behaviors. These ingredients are necessary to implement the effective coordination [110]. The latter components points to the collaboration. Collaboration is a joining activity task, in which teammates work side-by-side to reach a common goal. Collaborative tasks could be spontaneous or planned [89, 220]. The difference between these two type of interaction lies in the context, the environment, the objective and so, the situation.

Thus, it is important to recognize which theories contribute and analyze them to design a human-robot teaming system. A review manuscript emphasizes the relevance of theory to human-robot teaming research and development [191]. The paper summarized which factors contributes to enabling robots to be good in HRTI, as well as integrating theories of self-modeling and theory of mind [182].

Designing HRTI systems require a kind engineering process for representing and developing human capabilities in a robot. Current theories show a high-level potential on the usage of human social behavior theories and cognitive processes.

The next sections show the state of the art of the theories and frameworks used for implementing the solutions discussed in dissertation.

2.2 Cognitive Models for Artificial Intelligent Systems

Realizing intelligent systems that act as humans requires an in-depth analysis in modeling human-like cognitive capabilities. Artificial intelligence methods, sometimes, are not suitable for modeling them. The machine consciousness research field seems to be a solution for addressing this issue. Even, machine consciousness is rooted in the past, history is not so long, and we could almost say that it is a discipline that recently is evolving for supporting researchers and engineers in modeling cognition. Machine consciousness started to be a new way to realize consciousness. The manifesto proposed by Chella and Manzotti shows [48] a careful theoretical analysis about the characteristics of machine consciousness to be applied to robots. Summarizing the manifesto, issues to face are *embodiment, situatedness, emotions and motivations, unity, time, free will, representation, and, last but not least, qualitative experience* [48], each of them contributes equally in the development process.

A wide range of consciousness functions were suggested principally by Baars [12]. Baars proposed aspects such as definition and context setting, adaptation and learning, flagging and debugging, recruiting and control, prioritizing and access-control, decision-making, analogy-forming function, metacognitive and self-monitoring function, and autoprogramming and self-maintenance function. Aleksander, in [5], suggested other principles at the basis of the development of conscious systems, some of them are perceptual learning and memory, prediction, self-awareness, representation of meaning, will, learning utterances and language, instinct and emotion.

Let us consider machine consciousness research field as the underlying theory for developing cognitive architectures based on awareness, memory, learning and anticipation.

2.2.1 Overview

A cognitive architecture determines the structure of a computational cognitive model, applied in a generic domain, by underlying the infrastructure of an intelligent system.

Typically, cognitive architectures include modules representing the short and long-term memory and the functional processes operating on memory modules for realizing learning and action mechanisms. The literature proposes various cognitive architectures, a recent survey reports the last 40 years of cognitive architecture, detailing their characteristics and capabilities [119].

Cognitive architectures are in charge to create programs that are able to reason about problems across different domains, adapt to new situations and reflect on themselves and others. They are a part of the wide research area of general artificial intelligence. With the aim to model the human mind, cognitive architecture reveled to be good even for building artificial systems that show human-like capabilities. Cognitive models, at the base of the cognitive architectures, draws their skills from the cognitive science research fields, and that is why, designing and modeling cognitive architectures require a wide knowledge across several domain.

As mentioned by Russell and Norvig [177], an artificial intelligence may be realized designing systems that think and act like humans, rationally. Cognitive architectures are able to implement this type of artificial intelligence systems. Each cognitive architecture is built on a set of aspects that differentiates one architecture from another. Depending on which cognitive aspects the architecture implements, it is possible to classify this type of architecture. Another classification, the most common used in this field, is based on the type of representation and information that they implement. This classification is a three-paradigm classification, the paradigm are: symbolic, emergent and hybrid. The first, mainly, refers to cognitive approach, the second uses mainly a connectionist approach, the latter, as the name suggest, an hybrid approach. In other terms, a symbolic approach uses symbols for representing concept that can be manipulated through proper instruction set. This set could be, for instance, simple if-then rules applied to symbols for representing the facts known about the context on which the cognitive architecture is working. The emergent approach is more oriented to resolve learning and adaptability issues by building architectures similar to neural networks, where the information flow represent the propagation of the signals among nodes. Hybrid architectures attempt to combine elements of previous approaches, symbolic and emergent.

As the survey [119] highlights the major applications of cognitive architectures in robotics concerns the development of system for obstacle avoidance and navigation, and recently, for object manipulation.

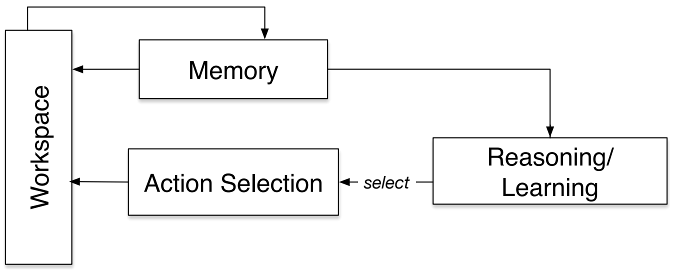


Figure 2.1: Common modules in cognitive architectures.

In literature, there exists various types and version of cognitive architecture, but very few architectures are well supported and maintained. The common cognitive architectures take into account some aspects neglecting others, depending on which context are designed and what they want to deal with.

Among the most known cognitive architectures, some are inspired by psychology and biology (for instance CLARION, SOAR, ACT-R, ...) [8, 123, 189] and other by agent technology, like for instance LIDA [75]. The fundamental principles they underpin are that every autonomous agent, be it human, animal, robot, software component/agent, etc., has to continuously sense the environment and reason on it to be able to select the appropriate action to reach an objective (action-perception cycle [79]). The previous scenario is the simplest one: an intelligent agent that has to be endowed with the capability of answering in a specific context exhibiting the right action. Thus, regardless of the context, a cognitive cycle can be schematically outlined as the module receiving all the sensorial data and the one processing them thus resulting in the corresponding action, as figured out in Figure 2.1.

In addition to these two modules, there is a basic module that manages everything related to data storage for further processing. This allows to represent, and therefore to implement, a system in which the decision-making process passes from the elaboration of a set of data, relative to everything that is perceived, memorized in the system. In existing architectures, the memory is mainly divided into short-term memory and long-term memory. The short-term memory stores all the facts and elements relating to the current moment. The long-term memory contains all data and events that are held indefinitely. Long-term memory is further divided into explicit and implicit memory in all its different forms (episodic, semantic, autobiographical and others) already widely debated in the literature [150, 6].

So, the memory module plays a fundamental role in reasoning on and learning the current situation to trigger the decision process.

Nevertheless, frequently human-robot teaming applications are not so simple to be sketched through these three simple modules but it is necessary, at least, that the memory mechanism also refers to all the evidence coming from perception and from knowledge of self and from the perception and knowledge of all the other agents present in the environment. This means that, in a human-robot teaming context, architectures must contain the modules useful for the representation of itself, of the (physical) world around and the other, including all the mental states that this fact entails.

To date, most architectures base their decision and learning process only on the concept of stored data or fact and not on the notion of mental state. So the implementation part of most known cognitive architectures is still missing. Our contribution lies in the creation of a cognitive architecture in which memory also contains all the information about the mental state so that the perceive-act cycle becomes what we call the perceive-proact cycle. Besides, the proposed architecture in Section 1.2 can be easily mapped into a BDI agent architecture [217, 84] for the effective implementation of the *cognitive system*, the software instance of a cognitive architecture.

2.2.2 Cognitive Architectures

Cognitive architectures are system arrangements for the production of human-like cognition. They try to combine and implement a large set of cognitive functionalities. Cognitive functions are in general: Perception, Attention, Prediction, Learning; Memory, Imagination, Planning, Judgment, Reasoning, General Intelligence, Emotions, Natural Language, Motor Action Control. A large number of cognitive architectures were developed during years. Samsonovich tried to list[3] them in [181].

This section shows relevant architectures studied during these years and that drove the designing process of the architecture shown in Figure 1.1.

[3]A comparative table of implemented cognitive architectures is available at this link: https://bicasociety.org/cogarch/ Last accessed: Jan, 23 2021.

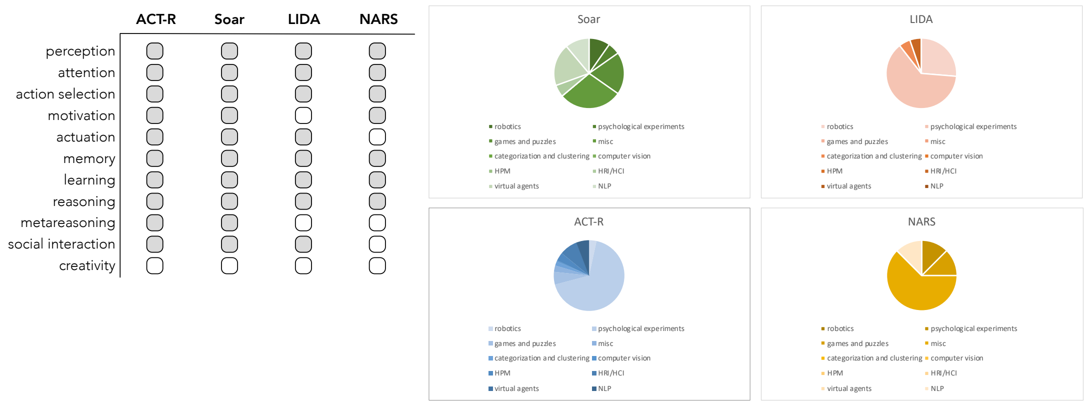


Figure 2.2: A list of cognitive capabilities implemented into the cognitive architectures that inspired the one proposed in this dissertation. The image figures out the usage of the selected cognitive architectures in specific domains. The pie charts were redrawn reading values from the website (see footnote 4), please refer to it for more details. Redrawn from [119].

Brief Considerations

An in-depth analysis about the usage of cognitive architectures for applications was done in [119]. Authors collected case studies and applications with which researchers and engineers used cognitive architectures for specific application domains and facing artificial intelligence challenges[4]. The manuscript shows a graphical representation of cognitive architecture applications related to their set of competency, accordingly with Adams et al. [1]. In Figure 2.2, a portion of the content shown in [119] is figured out.

Selected cognitive architectures are the most known and used in the scenario of artificial intelligence. These architectures realize, mainly, psychological experiments, games, and puzzles.

These architectures implement cognitive skills and capabilities through their cognitive models. Overall, they are not thought to be used for collaborative tasks and they do not take into account the mechanisms of anticipation for the HRTI domain, like the one developed in the proposed cognitive architecture.

Besides, the proposed cognitive architecture was developed for robotics and Human-Robot Teaming Interaction.

[4]URL available at http://jtl.lassonde.yorku.ca/project/cognitive_architectures_survey/applications.html. Last accessed: Jan, 23 2021.

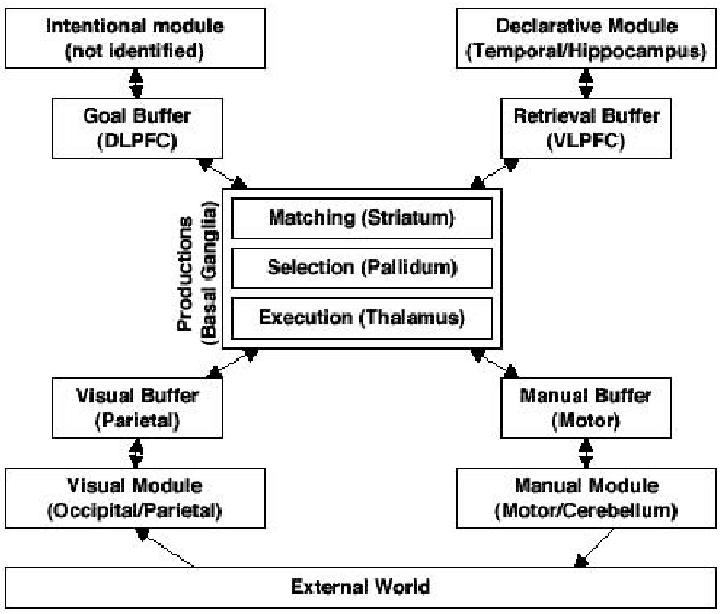


Figure 2.3: The ACT-R Cognitive Architecture.

The ACT-R Cognitive Architecture

ACT-R [9, 7, 173] was defined as the architecture for modeling cognition. Mainly developed by Anderson and Lebiere, ACT-R aims to define the cognitive and perceptual operations that enable the human mind, at the base of human reasoning.

ACT-R was inspired by the work of Newell and his idea of unified theories [149].

ACT-R, in Figure 2.3, is an architecture that is concerned with reproducing the most important characteristics of human behavior. ACT-R consists of a set of modules, each of which is intended for the creation of a different type of information:

- sensory modules, dedicated to visual processing;
- engine modules, dedicated to action;
- intentional module, dedicated to the pursuit of objectives;
- declarative module, dedicated to the maintenance of knowledge in the long term.

Each module coexists with a buffer that contains a declared relational structure (often called chunks) which, together with the buffers of the other modules, constitutes the short-term memory of ACT-R. In particular, a chunk consists of a vector representation of individual properties, each of which is accessible through a particular label. The coordination of the different modules is entrusted to the elaboration of certain production rules stored in a long-term memory, each of which has a cost (in terms of time needed to achieve the objectives) and a probability of success; each module, through the buffer associated to it, communicates to the outside and, therefore, their content will affect the choice of the central system on the next action to be performed. To each chunk is associated a basic activation that reflects its past use and influences the recovery from long-term memory. The operations within a module are carried out in parallel, while the communication between two modules must necessarily pass through the procedural module that operates in serial mode, launching only one production at a time and forming the main bottleneck of the system. One of the fundamental aspects of ACT-R is the subdivision of memory into two distinct parts: declarative memory, represented by the declarative module, keeps knowledge accessible from the consciousness on which one can reason and procedural memory, represented by the procedural module that keeps, instead, the capabilities that the human being generally does not use in a conscious way.

The Global Workspace Theory

The Global Workspace Theory (GWT) is a cognitive model proposed by Baars [15, 16] which is described metaphorically as a theater where several actors (the working memory, ensemble of expert systems) compete between them to earn the "spotlight of selective attention" on stage (the consciousness), while most of the background work remains invisible and behind the stages (the unconscious) [13].

Baars sees the consciousness as a biological adaptation that allow the brain to learn, understand and interact with the outer world [13]. Baars thought to this model as something that is distributed as the model of the brain that he imagined. Everything is contained in the consciousness, perceptions, inner speech, imaginary, feelings, emotion and other cognitive functions that help the system to reason. For Baars the brain has not a central command, instead each node of the network is in charge to produce some activities and they are controlled by their aim and contexts. The focus of the Baars global workspace theory is the namesake model. The Baars Global Workspace model [12, 13] uses the concept of a network of neural assemblies that would show the content of the consciousness. Based on a central working memory, called global workspace, it acts

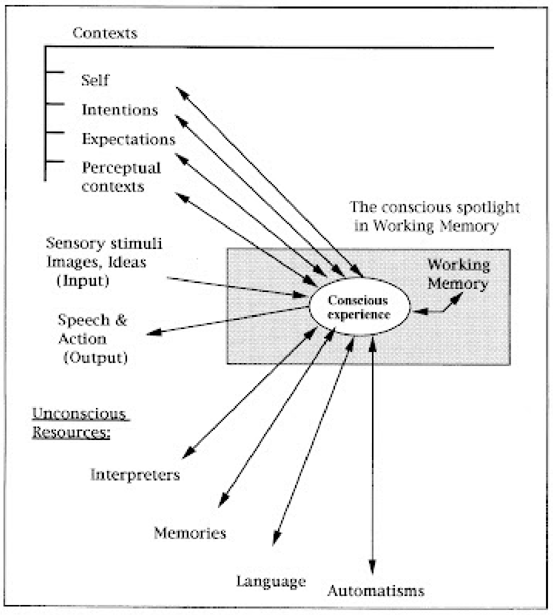


Figure 2.4: The Global Workspace Theory Model. Image from [15].

as a theater stage that displays and broadcast the content of conscious and unconscious audience or resources. This last consists of operators such as the memory system, action control, skills and other capabilities useful for performing actions or sensing. The model is shown in Figure 2.4. To simplify the description of the architecture, it is possible to thought to the GWT model as a blackboard model [15]. A blackboard model in Artificial Intelligence is a computational method in which a problem is written on the blackboard, available to be used by all specialist operators that works upon the problem. This operators are usually called expert systems. To summarize the GWT model, it is possible to assume that the GWT is a theater model derived from the blackboard models of Artificial Intelligence. It consists of a number of specialists modules and a common area, called working memory area, the global workspace broadcasts its contents to all its expert modules. Each expert system competes with others to transmit its contribution in the global workspace. With the GWT, it is possible to explain and simulate the conscious and non-conscious activities and it explains also the serial narrow bandwidth nature of the stream of consciousness. To conclude, each expert system operates in parallel with other and subconsciously. For more details, look at [17, 14, 16].

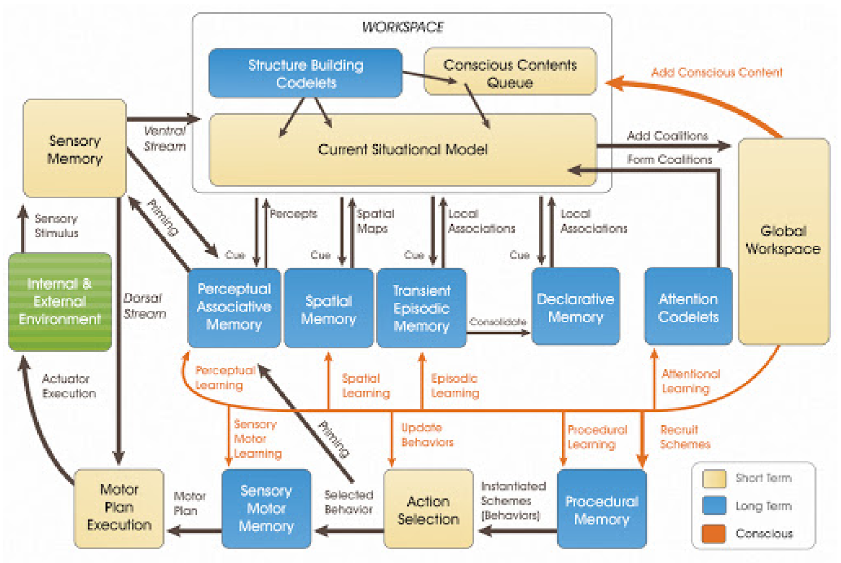


Figure 2.5: The LIDA Cognitive Model. Image from [76].

The LIDA Cognitive Architecture

LIDA [186] (Learning Intelligent Distribution Agent) is an architecture that uses the hybrid approach. The symbolic scheme coexists with the emerging connectionist approach in the LIDA memory organization. In contrast to biologically inspired cognitive architectures, LIDA models the mind from the perspective of an agent model [73] and not the brain.

The architecture provides separate modules for perception, emotions, selection of actions, learning and problem-solving, as well as a finely organized memory in specialized components. Most of the operations are performed by *codelet*: small, simple, and highly specialized programs that are waiting for certain conditions to occur to perform their task.

The LIDA architecture is derived from IDA (Intelligent Distribution Agent). IDA was developed to obtain an intelligent, autonomous and conscious agent [72], who could perform tasks in place of human beings and who could do it with the same results. LIDA extends it adding three learning modes: *perceptual*, *procedural* and *episodic*.

Cognition is based on the Global Workspace Theory [12].

In LIDA, Global Workspace (GW) is the area where the structures created during the attention phase of the cognitive cycle compete to become conscious. The winners are transmitted to all modules of the system via conscious *broadcast*.

The LIDA cognitive cycle, shown in Figure 2.5, can be seen as the fundamental constituent element of all higher level reasoning processes, divided into three phases: (*i*) perception and understanding; (*ii*) attention or interpretation; (*iii*) action and learning.

Perception starts the cognition via sensory stimuli, with the activation of low level characteristic detectors whose value is represented in the sensory memory. The *two-streams hypothesis* [88] lets understand which components of architecture are influenced by perception, the hypothesis divides in two flows: (*i*) the ventral flow is associated with the `thing`, that is the identification and recognition of shapes and objects. It is characterized by a higher degree of awareness; and (*ii*) the dorsal flow is associated with the `where` and `how` (such as, the awareness of spatial relations and related actions). It is mainly unconscious but more immediate.

The second phase has as an end the awareness of the observed data and the consequent updating of the long-term memory, for the purposes of subsequent learning. Each attention codelet examines the Current Situational Model (CSM) in search of elements inherent to its field of competence, established by the task.

As soon as an attention codelet finds in the CSM a structure of its interest, it includes it in a coalition [12] and copies it in the global workspace. An Attention Codelet (AC) can either add structures to its coalition or merge its coalition with that of another codelet. The coalitions created in GW compete to determine which one is more relevant and therefore deserves to become part of the conscious sphere with greater urgency. The competition is based on activation, as each coalition has been assigned a level of activation that depends on four factors:

- The basic activation level c from the AC that created it.
- The degree of correlation σ between the structures and the task of the codelet.
- The activation α of the structures that compose it.
- A scale factor related to the state of progress t in a possible refactoring period T.

The refactoring [76] is triggered when the winning coalition has a particularly high activation and requires to assign to new coalitions activation initially low and gradually increasing over the period. This mechanism allows to temporarily give precedence to pre-existing coalitions that although not winning had a high relevance, rather than the new ones.

The action phase starts as soon as the procedural memory receives the conscious data broadcast from GW. In the action selection, the behaviors are organized in a network on the model of the Maes architecture [139], in which each module is specified by pre and post conditions and by an activation level. In this case, given a behavior, a node is identified by the action, the links with the predecessors are defined according to the contexts and those with the successor nodes according to the results.

NARS: The Non-Axiomatic Reasoning System

In the field of consciousness for artificial systems, a novel paradigm leads the next generation of artificial intelligence. This paradigm, knows as Artificial General Intelligence (AGI), is the hypothetical intelligence [104] of a machine that shows human capacity in reasoning and understanding. The role and the strength of AGI are clear, some academic refers to it as the *strong AI*. The AGI is intended to perform cognitive abilities and not limited to specific problem solving or reasoning tasks. In the next paragraph, an AGI system is presented: NARS.

Non-Axiomatic Reasoning System (NARS) is a general-purpose reasoning system that uses the Non-Axiomatic Logic (NAL) [207]. NARS is able to reason under the assumption of insufficient knowledge and resources [205].

According to Wang, NARS is based on the belief that the essence of intelligence is the capability to adapt to the environment and to work with insufficient knowledge and resources, so in NARS, “intelligence” is usually taken as an ability to solve problems in a way similar to humans [203, 204].

NARS is a reasoning system with a unified reasoning-learning mechanism, using NAL inference rules, a concept-based memory structure, and attentional control. Narsese is the language to produce logical sentences in NARS, it is used as an internal language for handling I/O. NARS is equipped with self-related mechanisms to allow for introspective reasoning. [208].

Information acquired from the environment is processed and handled using sentences encoded in Narsese, the language that expresses NAL sentences. NAL [206] is a term logic with numeric truth values, where each truth value stands for a certain amount

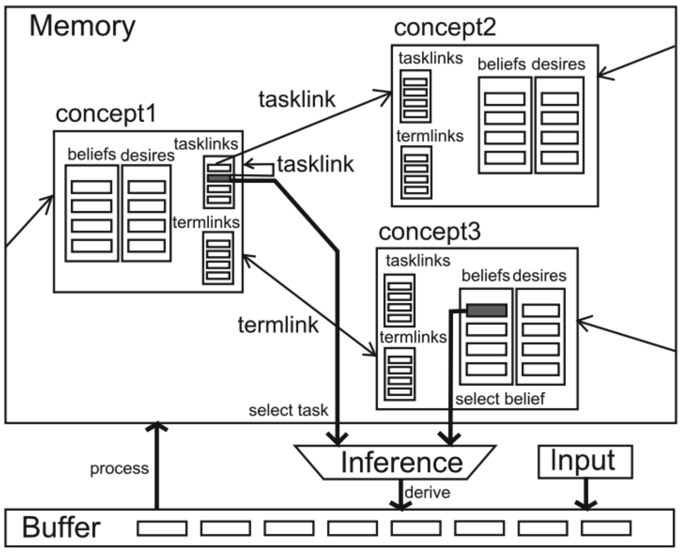


Figure 2.6: The OpenNARS architecture. Image taken from [98].

of positive and negative evidence.

NARS can be used where knowledge is outdated or generally insufficient. When circumstances change, plans can become ineffective. NARS can notice this to revise the truth value of the relevant knowledge. Also, it can build new procedure knowledge from observations as described in [96], allowing it to be used in cases where developers cannot prepare it for everything beforehand.

Figure 2.6 shows the design followed by the OpenNARS implementation [98]. As said before, NARS works under the assumption of insufficient knowledge and resource, this means that the architecture has to be *finite* and *open* and work in *real-time*. The Figure 2.6 shows the module of the framework, it is endowed with memory, logic, and a control component. The memory module supports three primary operations: *returning the best-ranked belief or goal for inference within concepts, providing a pair of contextually relevant and semantically related statements for inference between concepts, and adding statements to memory whilst maintaining the space constraints on the system* [98]. The logic component is based on NAL [207], it hosts two components, an inference rule domain-specific language, and an inference rule execution unit. NARS implements cognitive aspects shown in previous sections, such as anticipation and the sense of self [208]. Recently, a new design was released for implementing NARS rea-

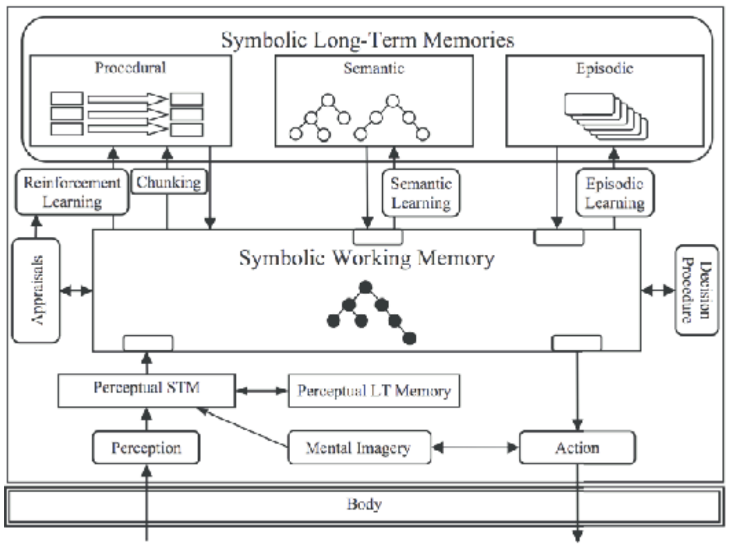


Figure 2.7: The Soar Cognitive Architecture.

soner in applications. The OpenNARS for Applications (ONA) system [97] shares the logic and conceptual ideas of OpenNARS, the event handling, and procedure learning capabilities of ANSNA [95, 96], and the control model from ALANN [136].

The principal difference between OpenNARS [98] and ONA [97] is the context in which it is used, while the first is related to academic and research context, the latter, as the name suggests, is oriented to the application domain, such as robotics application, smart-cities, ambient intelligence, and others.

The Soar Cognitive Architecture

Soar is a symbolic cognitive architecture created in 1983 by a work started by Laird and Newell [123]. Still today it is in continuous evolution including more and more capabilities. The main objective of the Soar project is to manage the full range of functionality of an“intelligent” agent, from the most trivial routine to the most complex and indeterminate problems. The main components of the computational theory of Soar are: (*i*) *goals*, (*ii*) problem space, (*iii*) states, and (*iv*) *operators*.

Figure 2.7 shows the larger blocks that present the primitive memories of Soar (procedural, semantic, episodic and working) and the smaller blocks that represent processes; all the connections between the units of the architecture are represented by arrows. Certain areas of Soar’s Working Memory activate calls to other memories, as

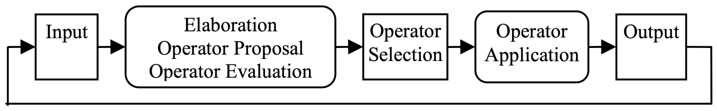


Figure 2.8: Soar execution cycle. Taken from [120].

can be seen from the presence in the figure of rectangles with rounded corners inside the central area. Perceptions from the outside world are acquired through the perception module and stored in the working memory in the form of symbolic structures (Symbolic Working Memory). The working memory can be considered a global short-term memory that manages the interaction with long-term memories, both for information and knowledge retrieval and for learning. The working memory activity can be influenced by the long-term memories and, in particular, by the procedural memory containing the productions that represent the core of Soar agents' reasoning. The productions are essentially composed of conditions and actions. The conditions match according to the content of the working memory and, if activated, produce effects described by their actions. Since all productions are controlled in parallel, some of them can be activated in the same control phase. The main work of procedural memory productions in Soar is to propose operators, whose actions permanently change the state of the architecture's memory. The semantic memory represents a real container of long-term training and knowledge, while the episodic memory deals with single episodes that collect inside them the information of certain events. The data of both memories can be retrieved through specific requests from the working memory. A characteristic of the architecture is the reasoning on impasse. Sometimes the production rules do not have enough knowledge to make a decision. This problem manifests itself when the decision-making phase fails to determine a single choice regarding how to change the current state. In this circumstance it is said that the system has reached an impasse. Soar activates a subgoaling mechanism that allows you to search for a solution and possibly learn a new decision that will be saved in the procedural memory.

As shown in Figure 2.8, the Soar execution cycle consists of the following five steps (operators) [121]:

1. Input: add new information retrieved from the sensors of Soar;
2. Proposal: activates the productions that produce effects based on the interpretation

of new data, or proposes new operators to evaluate. The productions continue to be activated until the state of quiescence is reached;

3. Decision: a new operator is chosen, otherwise a new impasse is generated;
4. Application: the actions triggered by the operator are applied. A new internal state is generated and the production activation cycle is repeated until the quiescence is reached.
5. Output: output commands to the external environment in Soar.

The execution cycle is repeated infinitely, unless the agent stops (*halt*) or the user interrupts it (*interrupt*).

The Standard Model of Mind

In 2017, Laird [122] proposed a common computational framework. He called this framework *Standard Model of Mind*. The hypothesis is to provide an appropriate computational abstraction for defining a standard model, and so this abstraction, in turn, is an architecture too. Authors started analyzing the state-of-the-art cognitive architecture and finding correspondence among them. The hope was to suggest some guidelines for designing cognitive architecture, in which the mind of the cognitive architecture behaves in a human-like fashion. Authors claims that proposing a standard model of mind could help scientists and researchers to speed up and enhance the quality of the research on this field, producing architectures that meet requirements and constraints in the development process of intelligent systems.

The Standard Model of Mind is the result of a long research work in the field of cognition. Authors, in fact, says that their work started in the past, and want to be a kind of *successor* of the *model human processor* proposed in 1983 by Card, Moran and Newell [36]. Other models influenced the Standard Model of Mind, such as the one proposed by Newell [149].

The Standard Model of Mind is grounded in: ACT-R, Soar and Sigma cognitive architectures. ACT-R decomposes the cognition process in five specialized modules and shows how to integrate the modules in order to create a complete cognitive process. SOAR is based on a cyclic process that includes the production process and the decision

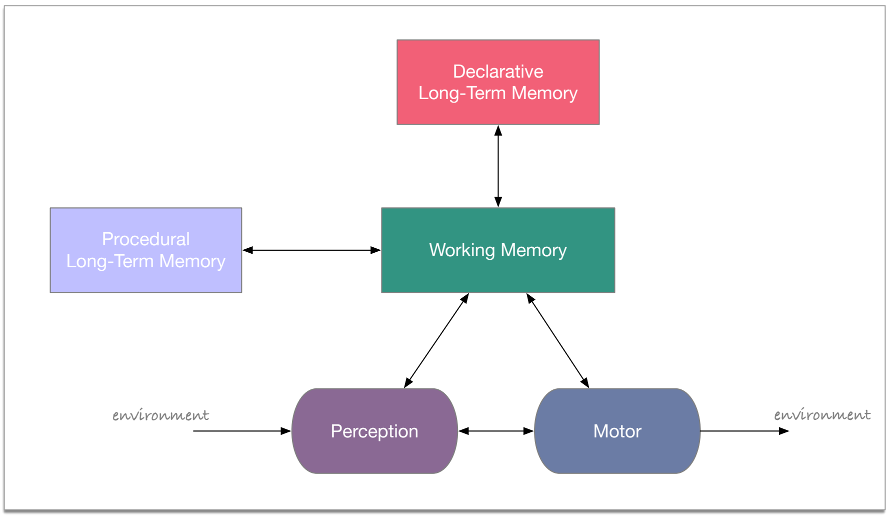


Figure 2.9: The Standard Model of Mind. Image redrawn from [122].

one. One decision cycle follows each production cycle. Sigma blends elements both from ACT-R and SOAR and provides just a single long-term memory.

Figure 2.9 shows the core of the standard model. It starts from perception module, to conclude with the motor module. The first is in charge to sense the inner and outer world of the cognitive entity that hosts the standard model, instead the latter performs operations in the inner and outer world too. The standard model uses a working memory and two different type of long-term memory, declarative and procedural. Both contribute to represent the knowledge necessary to decide and acting, each for its own competence.

2.3 Knowledge Representation

The problem of knowledge modeling is one of the biggest challenges in the development of software architecture based on inference systems or logical rules that use knowledge as information for deliberating actions. In [187], the term *knowledge engineering* was used for describing the field of Artificial Intelligence (AI) apt to design systems for handling Knowledge Base (KB) in Artificial Intelligence software system. Knowledge bases lend themselves well for implementing reasoning systems able to operate actions after a decision-making process. Thus the attention was moved onto the ability to design Knowledge-Based System (KBS). In literature, a possible approach discussed for modeling Knowledge Bases was treated in [210].

The formalism with which an agent codifies its knowledge is a central aspect in the development of cognitive architecture. Traditionally, representations are divided into declarative and procedural. The distinction between the two forms of representation is in the way knowledge is acquired, organized, and represented. Knowledge for an agent consists of facts and concepts about the world in which the agent operates. Thanks to these elements the agent can activate the mechanisms of reasoning to plan and "understand" the steps taken towards the goal. Designing of agents able to reason and act exploiting their inner capabilities, recall the concept of designing architectures for *representing knowledge* to manage and manipulate information that lets the reasoning system to act. One possible interface for representing knowledge in agent-based architecture was proposed in [19]. In architectures, such as the one proposed in this dissertation, a knowledge representation is used for storing, managing, and handling information. The knowledge, in general, could be represented by exploiting different approaches, but practically, programmers tend to use specific knowledge representation that is common for the specific agent programming framework.

Many frameworks, including Jason[5] [27], use a customized knowledge representation model.

Knowledge within this type of agent framework is not easy to handle, mainly, when the architecture has to be applied in complex scenarios. In general, the literature suggests using OWL [24] representation. This representation lets the agent access a large amount of information, organized semantically, and easy to handle. This type of representation is not standard for available frameworks and this requires for realizing interface able to blend their knowledge representation approach with the OWL one. In [19], for instance, authors proposed an interface for interfacing a Prolog-based application with the OWL knowledge base. Other approaches in literature were proposed, for instance, agent framework such as JACK [211] and Jadex [160] implement beliefs and goals of an agent exploiting an object-oriented technology [53], combined with XML [100]. This approach is a bit different from the standard approach on which knowledge is represented using knowledge technology. For example, another framework such as Jason, solutions based on OWL was realizing such in the case of JASDL extension [116], that lets agents incorporate OWL knowledge into their belief base. Another solution is

[5]The framework used for developing the cognitive architecture here proposed.

shown in JIAC [103], where OWL is used for representing the agent's beliefs. To conclude, literature offers various solutions to represent knowledge in agent-based software architectures, and mainly, semantic web technologies (above all OWL) are widely used for enhancing the knowledge representation in this type of architecture. Other solutions are based on Bayesian Network [3] and Conceptual Space [83].

2.3.1 Ontologies for Modeling Knowledge

The use of ontologies for effective knowledge modeling and information retrieval was treated by Munir and Anjum [147]. In this paper, the authors discussed approaches for facing several application domain issues in the field of information retrieval based on ontology, techniques for modeling ontology as well as the processing and the translation of ontological knowledge into queries and search requests for a database. The field of knowledge representation through ontology is rich in ontology and query languages. Most of them are based on the eXtensible Markup Language (XML) [100], easy to read and manipulate by computers. In this category fall languages such as the *Resource Description Framework* (RDF) and the namesake schema [117], the DARPA Agent Markup Language and the Ontology Inference Layer (DAML+OIL) [87] and the Ontology Web Language (OWL) [155] and OWL2 [91]. An ontology is more than a simple taxonomy because it allows representing further semantics relations beyond the *is-a* subsumption. Formally, the ontology is a set of concepts, individuals, and roles. The concepts represent abstract entities, and can be considered the symbolic representation of the knowledge; the individuals are instances of concepts and represent concrete entities in the environment. The roles are properties, that can be relational or datatype; the former define abstract relationships among concepts, latter define properties of a concept with datatype values.

The choice of the type of ontology depends on what the programmer wants to do and the needs of the domain, to correctly evaluate what features to look for in ontology modeling languages. A first evaluation may be done in term of their *expressive power*, *tools* and *reasoning support*. Applications that widely use ontology for representing knowledge, usually, use the RDF/RDFS XML based language rules and ranking structure.

The OWL Ontology is a semantic ontology-based on a markup language generally used to publish and share ontologies on the web [24]. Using OWL allows representing means and semantic of terms using vocabulary and relations between them. It uses three sub-languages known as OWL-Lite, OWL-DL, and OWL-Full. OWL is built upon the RDF and DAML + OIL [85] and it provides expressiveness power respects its predecessors. Given the success and efficiency of OWL, recently, researchers and knowledge engineers released the OWL2 [91] that inherits all qualities, building blocks, and structure from its predecessor OWL. This choice ensures the compatibility among ontology written in OWL and parser or reasoner that expects OWL2 ontologies. OWL2 [91] integrates other concepts that enhance the expressive power of OWL. These are OWL2-EL, OWL2-QL, and OWL2-RL. The comparative table between most used dialects for handling knowledge [147] shows the difference between RDF/RDFS, OWL, and OWL2. To summarize the comparison made by Munir and Anjum, as we can expect, RDF and OWL share lots of features and capabilities but OWL has more expressiveness than RDF, this means that in terms of interpretability OWL is greater than RDF. Besides, the larger vocabulary than RDF lets define complex ontology concept restrictions and to formulate ontology-based relational database queries. The same consideration may be done for OWL2 and OWL, as a more natural consequence of more expressive power language.

2.3.2 Perspective and Analysis on Knowledge Acquisition

Recently, the interest on growing knowledge base affects various research areas and it is considered as solution to treat issues in important tasks, such as fields of: computer vision [115, 128, 71], task planning for social robot [49], reasoning [82] [86], symbol grounding for cognitive robot [51].

Letting knowledge grow means increasing and discovering new knowledge during agent/robot activities. The discovering process, usually, concerns the use of domain patterns and rules or machine learning processes. For instance, it is the case of NELL [37] and ELLA [178], that by using machine learning methods are able to generate new knowledge at the expense of transparency. This phenomena does not make the decision making process transparent. For some domain, transparency is required, such in the case on which an intelligent system has to justify itself for errors or fails. Generally, incremental knowledge acquisition approaches are from the logic programming [68] and the usage of this type of models arouses a strong interest in the field of robotics [148, 161]. Acquiring knowledge for the field of robotics is a symbol grounding problem [52], and it aims to anchor knowledge from the physical world in the robot knowledge model.

Generally, this type of problem in robotics is faced using ontologies, for instance KnowRob [190], OpenRobots [130] are based to OpenCyc ontology (a free and open version of CYC [131]). These ontologies include a wide set of concepts ranging from domain and common-sense. Approaches for ontology processing require manual pre-annotation tasks, leading to efforts for rules definitions about the domain representation [146, 94]. The existence of well-structured knowledge sources and memories is also considered fundamental [78, 174].

Novel knowledge management techniques are investigating on the feasibility to acquire the knowledge at runtime from tasks made by the robot. For instance, [135] proposed a system where the incremental acquisition is implemented by integrating different kinds of knowledge from sensory data, context and domain information, internal states, possible actions and so on. In conclusion, ontologies and semantic representations offer several advantages in building cognitive architectures and intelligent systems [135, 147, 151, 111, 77, 152, 162]. This type of knowledge management arouses a strong interest among knowledge engineering experts and the scientific community that uses it.

2.4 Multi-Agent System Engineering for Robotics

Recently, the prodigious steps did toward the development of intelligent software systems moved the interest of scientists and engineers to create novel paradigms apt to model and design them. In these years, academics proposed novel paradigms and solutions for facing challenges in this technological sector, such as agent-oriented software engineering, a valid paradigm for building autonomous and complex systems. Essentially, the agent paradigm [217] was used for decades as a solution, both from a theoretical and an implementation standpoint, for systems providing aid to humans in their everyday life. An agent is thought and programmed *to act on behalf of humans*.

The fundamental concept underneath the job done in these years is contained in a possible definition of a multi-agent system:

Multi-agent systems are systems composed of multiple interacting computing elements, known as agents [216].

The *computing element* cited in the definition above refers to an agent. So, an agent is a computational entity able to interact with other agents and capable of autonomous actions. The interaction among agents in a multi-agent system is the analog of the kind of social activity that humans engage in every day of their life. These two features, communication, and autonomous actions let consider multi-agent systems to be a natural metaphor for building and understanding *artificial social systems*. For describing a multi-agent system, it is suggested to split the description among three levels: (i) a micro-level (ii) the middle-level; and (iii) the macro-level. The micro-level treats the description of an individual agent, the middle-level is in charge to consider cooperation, communication, and interaction among a society of agents and the macro-level is related to decision-making in multi-agent systems.

So, for this dissertation, it is important to make clear the overview of mechanisms underneath the reasoning of individual agents to comprehend novelties introduces in future chapters and at the end understanding the idea at the basis of the cognitive architecture introduced in Chapter 1.

2.4.1 Agents and Multi-Agent Systems

An agent is an autonomous entity able to act in response to stimuli coming from the environment and proactively act towards a specific goal [214, 215, 27].

As defined in [217, 216]:

An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objective.

The agent paradigm was born to better understand and model complexity in software. Interaction among components is the main characteristic of complex software. Correctly engineering and implementing systems where complex components interact with each other is harder than engineering systems requiring the computation of single or simple functions. Agent theory is suitable for facing this kind of situation.

An agent has the following important properties: *autonomy*, *reactivity*, *proactiveness* and *social ability*. To understand what autonomy means when talking about agents let refer to a functional program, a Java class, a compiler, or something like that. All these kinds of software may be modeled by a function, they receive input and produce an output as the result of elaborating that input. Everything that happens in this application is because we (the programmers or the designers) want it to happen and to happen exactly in that way.

An autonomous agent is conceived to be the exact opposite of the applications above. Research in the field of agents and multi-agent systems is going towards means for building agents to which we can delegate tasks. It is up to the agent to decide how to reach the objectives, they act on the base of plans we give them. A plan defines the set of actions an agent may perform to pursue an objective.

An agent is *reactive* in the sense that it is able to perceive the environment and act in response to changes coming from it, actions are directed towards the agent's objective. One of the key points related to agent autonomy is that an agent may put together plans to achieve the goals. So they are making able to operate autonomously on the behalf of humans that delegate them their goals [216]. *Proactiveness* is the ability "*to exhibit a goal-directed behavior by taking the initiative*" [216]. *Social ability* is the ability of an agent to interact with other agents and humans to purposefully reach its objective [216]. This latest ability involves an important skill belonging to humans, i.e. communication abilities. To establish a society, the agent cooperates and coordinates activities with other agents, and therefore it is able to communicate to the other (also humans) its beliefs, its goals, and its plans.

A possible approach adopted by agent systems to carry out what proposed above is the *intentional stance*. In this way, programmers endow agents with *mental states*. A mental state may be *beliefs*, *desires*, *wishes*, *hopes* and so on. Thus an agent differs from an object because an object executes actions for free, an agent does it because it wants to do it. It is possible to refer to agents' systems as intentional systems, or better, entities "*whose behavior can be predicted by the method of attributing belief, desires, and rational acumen*" [55].

Agents lend themselves perfectly for the development of classical artificial intelligence systems, often defined as symbolic AI systems. In this typology of systems, the symbolic representation suggests the behavior to execute through logical formulas or

```
$B \leftarrow B_0$;      /* $B_0$ are initial beliefs */
$I \leftarrow I_0$;      /* $I_0$ are initial intentions */
while true do
  get next percept $\rho$ via sensors;
  $B \leftarrow brf(B, \rho)$;
  $D \leftarrow options(B, I)$;
  $I \leftarrow filter(B, D, I)$;
  $\pi \leftarrow plan(B, I, Ac)$; /* $Ac$ is the set of actions */
  while not ($empty(\pi)$ or $succeeded(I, B)$ or $impossible(I, B)$) do
    $\alpha \leftarrow$ first element of $\pi$;
    $execute(\alpha)$;
    $\pi \leftarrow$ tail of $\pi$;
    observe environment to get next percept $\rho$;
    $B \leftarrow brf(B, \rho)$;
    if $reconsider(I, B)$ then
      $D \leftarrow options(B, I)$;
      $I \leftarrow filter(B, D, I)$;
    end-if
    if not $sound(\pi, I, B)$ then
      $\pi \leftarrow plan(B, I, Ac)$
    end-if
  end-while
end-while
```

Figure 2.10: Practical reasoning taken from [27].

syntactical manipulations of the type logical deduction or theorem proving.

Some agents implement their internal states through a database of first-order logical formulas, these formulas are analog to human's beliefs, hence, the database of these formulas is generally called belief base. The agent's decision-making process is so implemented over these logical formulas through a set of deduction rules, also called the plan library.

Programming agents' architecture requires new guidelines and how-to. The idea at the basis of the Agent-Oriented Programming (AOP) is to develop systems in term of *mentalistic* notion, such as beliefs, desires and intentions [28, 185]. A method for developing this type of system is through practical reasoning. Practical reasoning is a way to reason for acting. Accordingly with Bratman [29]:

> *Practical reasoning is a matter of weighing conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what the agent believes* [29].

The practical reasoning combines two types of *reasoning*: the deliberation and the *means-ends reasoning* (see Figure 2.10). The former process what states of affairs the agent has to reach, the latter decides how to achieve these states of affairs. Before introducing the BDI paradigm and the framework, as a possible type for developing agent architectures, today, it is important to clear some differences in the development of intelligent systems. The choice, about the possible agent architecture and its decision-making system, has to consider the level of adaptivity that the problem requires. Qureshi et al. [165] show four levels of adaptivity proposed below:

- Type I - *self-adaptive systems of the first type are able to react to changes following the behavioral model given by designers, well specific decision points trigger actions on the basis of perceptions and available knowledge on the environment*;
- Type II - *systems of this type own many strategies to face changes, the best strategy is chosen at runtime and may impact on non-functional requirements and is the result of a compromise that considers perceptions and available knowledge also on special requirements*;
- Type III - *is applied to systems operating in a not totally known environment, strategies are not established at design time but assembled at runtime according to the actual execution context*;
- Type IV - *is the most similar to biological systems, these kind of systems are able to self-modifying their specification and generating strategies from scratch to respond to changes*.

Each level implies how much adaptivity agents have to acquire and learn from the surrounding world to operate their intentions into the environment. Various solutions and approaches dealt with aspects of the development of self-adaptive systems [112, 54, 35], and a meta-model [179] let deduce main abstractions for designing intelligent systems with a specified level of adaptivity.

2.4.2 BDI Agents: Introducing the Jason Framework

A widely recognized agency model in literature is the Beliefs-Desires-Intentions (BDI) model. This AI-based agent paradigm [168] involves a deliberation ability of the agent, based on a continuous sense-action loop and the evaluation of existing beliefs. It allows the agent to realize a desire with a plan available in its library.

Thus, the computational model implementing deliberation and means-ends reasoning (practical reasoning) contains four elements: B (beliefs), I (intentions), D (desires), and π (plans). At the beginning an agent is endowed with a belief base, hence its knowledge about the environment, and a plan library, a set of possible plans for reaching objectives, and the goals to pursue. A plan is selected and then activated only if some preconditions are true. The model develops on a loop for which an agent starts by updating the belief base, it is something like getting aware of the world around and choosing an intention. During the loop, the agent continuously perceives the environment, if necessary it updates the belief base, determines desires and intentions by computing current beliefs, and then generates a plan to reach the intention. Generating a plan means to select the best plan from the plan library, the one that fits with all the intention's precondition and executing all the actions it contains. After the execution of each action, the agent pauses and observes the environment. The agent might need to update the belief base and reconsider the intention. In the worst case, if no plans or no actions exist in the plan library for reaching an intention the agent should be provided with other plans, hence its plan library has to be enriched.

Due to its features, the BDI model well fits the need to model and implement the modules of the proposed cognitive architecture introduced in Chapter 1 and detailed in the rest of the thesis. Indeed, *Belief-Desire-Intention model* has been recognized as the most useful paradigm to implement human-like agents.

Thus, an agent is characterized by beliefs, desires, and intentions:

- *beliefs* - are information that the agent has about the surrounding area or the world in general.
- *desires* - are all the possible *states of affairs* that an agent perform. The desire is not a *must do action* but it could be seen as a condition that influences other actions.
- *intentions* - are the *states of affairs* that an agent decides really to perform. The intentions can be normally intended as a particular operation that can be delegated to the agent or agent's consideration. This latter definition comes from practical reasoning systems inside the agent.

A framework for building systems based on BDI agents is Jason [26, 27]. Jason is a framework for building multi-agent systems based on the BDI model [28, 31, 29, 30] employing the AgentSpeak language [167, 168]. It does not yet have the characteristics to perfectly replicate how a human acts, but it can autonomously process the knowledge it possesses about how to do things and implements the practical reasoning agents. The basic idea behind Jason is to define what is called the program's know-how in the form of a set of beliefs, goals, and plans. The Jason platform allows executing the deliberation process of a BDI agent that leads to choosing the intention to pursue within a set of possible states of affairs.

An agent can decide what to do, the set of actions in its repertoire to be undertaken starting from a set of data obtained through sensing and to modify the surrounding environment. In AgentSpeak, and therefore in Jason, deciding what to do means manipulating plans and the environment. Typically, a Jason agent has partial control over the environment in which it lives because it is also populated by other agents having control over their part of the environment. It can autonomously work because it is structurally defined to do this but cannot adapt itself in a dynamic environment; especially if the dynamicity of the environment derives from interactions with humans and other agents. The procedure for handling agent-agent interaction is standardized and mainly established at design time. Human-Agent, as well as Human-Robot, interactions have to be still explored, especially in the context of cooperation between humans and agents/robots which presupposes delegation and/or selection of actions to be undertaken even by observing the human actions and skills.

Hence, even though Jason is one of the most widely used frameworks to realize multi-agent systems, it does not support essential features such as refining itself to enhance the pro-activeness level. This capability is vital to realize efficient cooperation or collaboration during scenarios of human-robot teaming interaction as mentioned in the introduction. Let us suppose that, during the design phase, the agent programmer has endowed the Jason agent with a set of plans available in the plan library. Normally, the Jason agent reasoning cycle prescribes to find an *applicable* plan in the stack of plans of the library. A plan is applicable when the context (the set of pre-conditions for the plan) is true.

A plan in Jason is composed of two components, the *head* and the *body*. A *trigger event* and a *context* compose the *head*; generally, a plan is executed starting from another plan call or when an event raises and a plan is triggered for that event. A plan needs

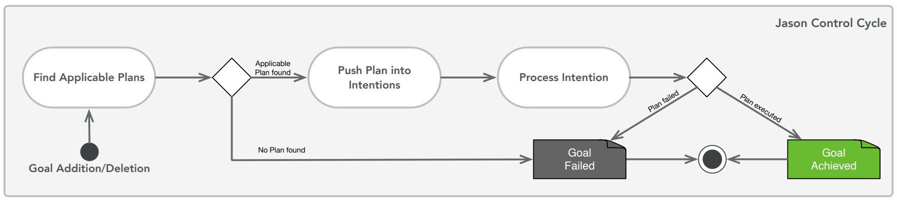


Figure 2.11: AgentSpeak control cycle. Redrawn from [142]

to check the context before the execution. The context is checked using a unification process on the agent's beliefs with some logical operators. If the context is true, the plan can be considered and the intention associated with the plan can be processed (see the standard control cycle in Figure 2.11).

If the context is false, the plan cannot be applied. If no other applicable plan can be found, the goal fails. The reasoning cycle of Jason is based on a static plan library, written by developers in the *agent definition file*. Consequently, Jason cannot redefine itself from within, or simply revise its plans nor create new ones or compose by existing ones.

A plan is performed by using a portion of AgentSpeak code called *action*. Actions may change the state of resources in the environment or the internal state of the agent. In this latter case, an action is named *Internal Action*, and it represents an inner agent's capability. Another major part of the plan is the *Recovery Plan*; it is a part of the agent code that takes control only if all the evaluated plans are not applicable.

For understating the reasoning process of a Jason agent, the next section will discuss the Jason reasoning cycle.

2.4.3 The Jason Reasoning Cycle

The main concepts in the Jason reasoning cycle are illustrated and described in the following table (Table 2.1). Table 2.1 and Table 2.2 are based on chapter 4 in the book by Bordini et al. [27].

AgentSpeak programs use agents to set the initial state of beliefs, the events, and the set of the plans that an agent could execute during its life-cycle.

The agent reasoning cycle, in Figure 2.12, consists in three main processes, (i) a knowledge updater, (ii) an event handler and (iii) a module to act.

According to [27], *rectangles* represent the principal components that determine the

Table 2.1: A complete summary of Jason's components and elements, more details in [27].

Agent	An agent is an entity with several capabilities. An agent is able to perceive and act in environment, communicate with others and reason about possible events. Agents have several skills and offer services.
Belief	Beliefs are information about the world.
Belief Base	A Belief Base is the structure where all beliefs are organized.
Plan	A Plan is composed by three parts: the triggering event, the context, and the body. The body contains other plans or actions.
Plan Library	A Plan Library is where all plans are stored and lets agent choosing which plan is more applicable or not to reach the goal.
Event	An event is a couple where the first component denotes the change that are taking place, and the second is the associated intention. It may be internal or external, the first is related to the goal the second is related to environment's changes.
Intention	Intentions are the states of affairs that the agent has decided to commit
Percepts	Percepts are referred to state of affairs in the surrounding world.
Context	As mentioned in [38] a context is the place where agents take into account others and/or where the others act to realize tasks.

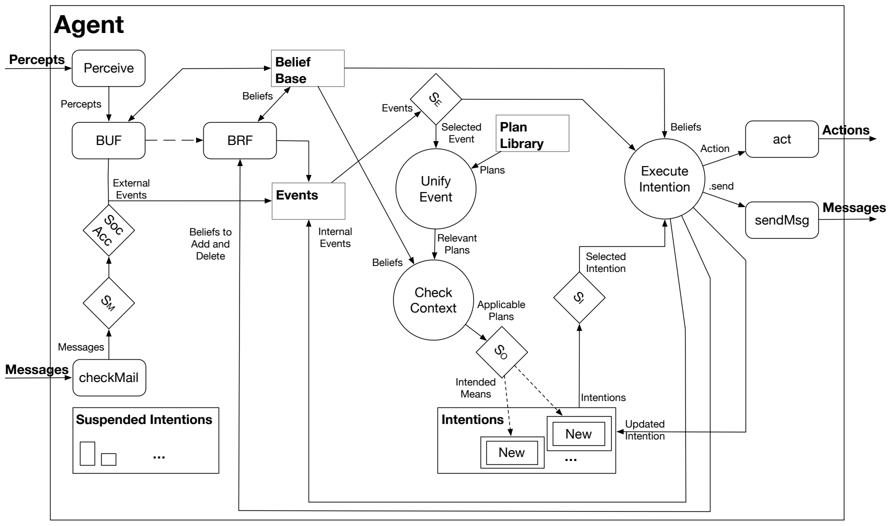


Figure 2.12: Jason reasoning cycle redrawn from [27].

agent's state, such as belief base component and all the components that handle the set of events, plan library and intentions. *Diamonds*, *circles* and *rounded boxes* are used for describing the functions used in the reasoning cycle. In particular, *circles* model the application processes and *diamonds* represent the selection functions. Jason allows to modify and customize the functions represented by *round boxes* and *diamonds*.

The first thing that an agent does at the beginning of each reasoning cycle is perceiving the environment through its senses. This operation involves the belief base and the related function. Beliefs are represented by using a symbolic form, the architecture has an internal *Literal* component that converts perceptions into a list of Literals; each of these is a single *percept* and this is a symbolic representation of a specific property perceived from the environment. This *percept* is detected by *perceive method* that implements the process. Once the agent has perceived the environment, it *updates* its internal belief base to reflect changes in the environment. The method able to do this is the *belief update function*, also known as *buf*. Jason's *buf* presumes that every perceptible thing could be included in the list of percepts generated by the perceive module.

Generally, the *buf method* updates the belief base in a simple way; this method consists of two points: (i) each literal l in p not currently in b is added to b; (ii) each literal l in b no longer in p is deleted from b; where the symbol p is the set of current percepts and b is the set of literals in the belief base that the agent obtained from the last sensing process.

An *event* in Jason is considered as a couple, where the first component is the change occurred and the second is the associated intention. Furthermore, the event could be divided mainly into two categories, *internal* and *external events*. Each change produced by *buf method* invokes an *external event*. This kind of event is created when the source of belief is related to the percept. In this case, the associated couple contains as the first member the change produced and as the second member an *empty intention* denoted by a $\top$.

Another important module is the communication system. Within the cycle, an agent could retrieve information about the environment or other agents through others in the same system. At this point, the cycle checks for new messages that might have been sent from others. This distribution system is just integrated into Jason and it works such as a mailbox for each agent.

The method within the reasoning cycle is called *checkMail* that simply checks for received messages and makes them available at the level of AgentSpeak interpreter to be handled by agents.

The *message selection function*, denoted by $\mathcal{S}_{\mathcal{M}}$ in Figure 2.12, selects the first message in the list in the default implementation. By security reason a *social acceptance function* is defined into the system, the aim is ignoring potential malicious attack that others could do or bypass wrong messages that could spoil the reasoning loop, the default implementation accepts all messages from all agents.

A relevant role is taken by events since they represent the effective changes in the environment or the agent's own goal. The principal cognitive skill of the BDI Jason Agent is handling events, this happens only by managing them one at a time; for instance, in a hypothetical cooperative scenario, the agent could have the necessity to evaluate other events before the first in the set of events; to solve this problem Jason lets the user customize a specific function called *event selection function* denoted with $\mathcal{S}_{\varepsilon}$ in Figure 2.12. The default implementation selects the first event in the list of events; it works as a FIFO structure if not customized.

Once a relevant event has been selected, the agent needs to select a set of reliable plans from a collection called *Plan Library* that will permit to react to a specific event with a designed action. The agent searches into the *library* all necessary plans for each event. After selecting the relevant plans, a unification process helps the reasoning cycle

Table 2.2: Description of functions for the Jason reasoning cycle. More details in [27].

Function	Description
perceive	The perceive method lets agents sense the environment and retrieve from it information about things. It implements the perceive function in Figure 2.12.
socAcc	Social Acceptance Function establishes the reliability of other agents, it implements the *socAcc* function in Figure 2.12.
selectEvent	Selects the events that will be handled in the reasoning cycle; it implements the $\mathcal{S}_\mathcal{E}$ function in Figure 2.12.
selectIntention	Selects an Intention to be further executed in the reasoning cycle; it implements the $\mathcal{S}_\mathcal{I}$ function in Figure 2.12. The default implementation executes intentions with a *round robin* scheduling process.
selectOption	This method is used to select one among several options (an applicable plan and an unification) to handle an event. It implements the $\mathcal{S}_\mathcal{O}$ function in Figure 2.12.
selectMessage	Selects the message that will be handled in the current reasoning cycle; it implements the $\mathcal{S}_\mathcal{M}$ function in Figure 2.12.
buf	Updates the belief base with the given percepts and adds all changes that were actually carried out as new events in the set of events.
brf	Revises the belief base with a literal to be added (if any), a literal to be deleted (if any), and the Intention structure that required the belief change.
act	Act function lets agents execute action in the environment.

to catch a set of relevant plans on which a check context process will be applied. The output of the latter process lets deleting all the plans that do not match with the current context and only a subset of all relevant plans will be marked as *applicable plans*.

Given the set of applicable plans and all the knowledge acquired by the agent through perception and communication skills, converted into a set of literals represented as beliefs, the agent is able to choose one of the selected applicable plans through an internal method: *Option Selection Function* denoted with the symbol $\mathcal{S}_{\mathcal{O}}$ in the Figure 2.12. The output of $\mathcal{S}_{\mathcal{O}}$ produces what is called *intended means* and this name is associated to the mean that the agent intends to execute to handle the related event. It is worth to remind that each plan in the set of applicable plans represents an alternative to reach the goal. The default implementation of this function considers as applicable plan the first in order of appearance in the plan library; the position of each plan is defined in the agent definition file written in AgentSpeak. Thus in the default implementation, the agent attempts to choose the applicable plan according to the agent's developer.

To select which intention to be computed during the cycle, an *intention selection function* denoted with the symbol $\mathcal{S}_{\mathcal{I}}$ is used. $\mathcal{S}_{\mathcal{I}}$ has a default implementation like a *round-robin scheduler*. Moreover every time an intention is extracted, it is removed from the list of intentions after the execution, and when this will be inserted back into the list, this will be added at the end of the list. In this way, every intention has the same portion of the agent's attention.

The last stage of the reasoning cycle performs an action in the agent environment or produces a message to communicate with other agents. Generally, the action that an agent performs are (i) *environment action*, (ii) *achievement action*, (iii) *test goal*, (iv) *mental notes*, (v) *internal actions* or (vi) *expressions*.

The framework revealed useful for implementing the cognitive architecture and was extended accordingly as will be described in Chapter 3.

2.5 The Trust Theory and Agents

Trust is a general term to explain what a human has in mind about how to rely on others or himself. In literature, we can retrieve more than one definition of trust. These definitions often are partially or entirely related to the others.

One of the most accepted definitions of trust is the one by Gambetta [80]:

Trust is the subjective probability by which an individual, A, expects that another individual, B, performs a given action on which its welfare depends.

Trust is strongly related to the knowledge one has on the environment and on the other. Knowledge of the environment is often the result of some kind of measure of trust. Trust is seen both as a mental state and as a social attitude. Trust is related to the mental process leading to the delegation. The degree of trust is used to rationally decide whether or not to delegate an action to another entity, the classic "on the behalf of". A software agent [216, 217] is born to act in place of the human; all the theories and technologies about agents are born and have evolved around this pivotal point.

The concept of trust in this dissertation is referred to the work of Falcone and Castelfranchi [38, 65, 66, 41]. Authors [38] define:

- *trust* as *mental attitude* allowing to predict and evaluate other agents' behaviors;
- *trust* as a *decision* to rely on in other agent's abilities;
- *trust* as a *behaviour*, or an intentional act of entrusting.

Moreover, in [38], trust is considered as a composition of a set of different figures that take part in a trust model:

- the *trustor* - is an "intentional entity" like a cognitive agent based on the BDI agent model that has to pursues a specific goal.
- the *trustee* - is an agent that can operate into the environment.
- the *context* - is a context where the trustee performs actions.
- τ - is a "causal process". It is performed by the trustee and is composed of a couple of act α and result p, g_X is surely included in p and sometimes it coincides with p.
- the *goal* g_X - is defined as $Goal_X(\mathbf{g})$.

The trust function can be defined as *the trust of a trustor agent in a trustee agent for a specific context to perform acts to realize the outcome result.* The trust model is described as a five-part figures relation:

$$TRUST(\text{X Y C } \tau \ g_X) \tag{2.1}$$

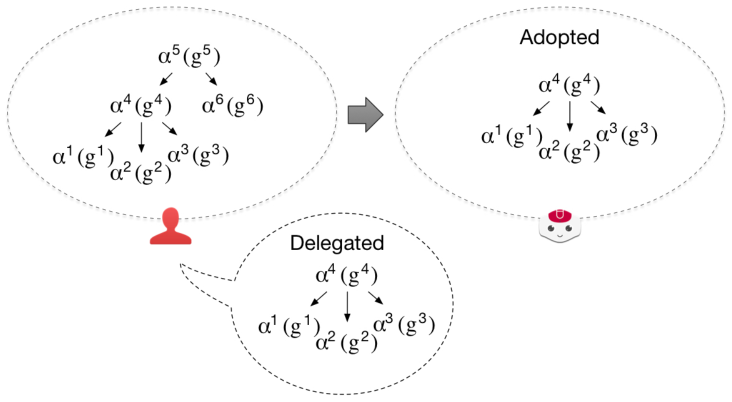


Figure 2.13: Level of Delegation/Adoption: The *Literal Help*

where X is the trustor agent, Y is the trustee agent. X's goal or briefly g_X is the most important element of this model. In some cases, the outcome result can be identified with the goal. For more insights on the *model of trust* and *the trust theory* refer to [38].

In this theory, trust is the mental counterpart of delegation. In the sense that trust denotes a specific mental state mainly composed of beliefs and goals, but it may be realized only through actions. Delegation is the result of a decision taken by the trustor to achieve a result by involving the trustee.

Several different levels of the delegation have been proposed in [40] and [64], they range from a situation in which the trustor directly delegates the trustee to a case in which the trustee autonomously acts on behalf of the trustor.

For instance, Figure 2.13 shows a literal help case study, on which a human delegates a sub-goal and the list of plans to realize it.

In the literal help, a client (trustor) and a contractor (trustee) act together to solve a problem, the trustor asks the trustee to solve a sub-goal by communicating to the trustee the set of actions (plan) and the related result. In the *literal help* approach, the *trustee* strictly adopts all the sub-goals the *trustor* assigns to him [40, 64].

This corresponds to the notion of behaving "on behalf of" that, as said, is one of the key ideas in the multi-agent systems paradigm. Agents' features, such as autonomy, proactivity, and rationality are a powerful means that make trust-based agents ideal candidates to be used in applications such as human-robot interaction.

The idea is to develop multi-agent system using the *belief-desire-intention* (BDI) paradigm [28] and so, the practical reasoning [29]. As said, the practical reasoning, in human-terms, consists of two activities:

- *deliberation and intentions*;
- *means-ends reasoning*.

Each activity can be expressed as the ability to fix a behavior related to some intentions and deciding how to behave. These features of a BDI agent shall faithfully reflect all we need to realize a system based on the trust theory.

Figure 2.10 shows the standard practical reasoning cycle of a BDI agent. Chapter 7 will show how it was changed the reasoning cycle and a theoretical model for supporting self-modeling capabilities to implement trust.

2.6 Summary

This chapter shows the theories that drove the work done in these years. To summarize all the concepts described here was used agent-oriented programming to develop a cognitive architecture adopt to model cognitive abilities in HRTI systems. The choice of the BDI model as agent programming lies in the characteristics that the paradigm owns. By using the BDI model it is possible to describe all the modules representing the system cognitive abilities in terms of computation entities. A practical reasoning agent lets define intelligent software modules simply by describing them under a logical standpoint and represent all the knowledge useful for reasoning. Moreover, for enhancing the quality of interactions between robots and humans the concept of the transparent reasoning process, or better explainable AI should not be undervalued. With this choice, the cognitive architecture is transparent and able to justify its decisions. In the following chapters, all the decisions among model designing and implementation as well as developed modules are given.

Chapter 3

A Cognitive Architecture for Human-Robot Teaming Interaction

Human-Robot Interaction (HRI) aimed at cooperation and teamwork is a demanding research task. The research work carried out during these years of my doctorate is part of a longer-term research objective in which my laboratory is focused. The performed work focuses on analyzing and developing a cognitive architecture for Human-Robot Teaming Interaction (HRTI). The aim is to let the robot act as a human during cooperative tasks.

Developing systems for HRI is mandatory for enabling robots to perform operations in a society where interactions and communications are essential. For rapidly enabling machines to be human-friendly, it is necessary to study human interactions and try to reproduce them to create intelligent systems that behave like humans.

Cognitive architectures enable machines to be intelligent implementing reasoning systems for deliberating and acting in environments, which are partially known and dynamic; they also represent a good starting point for developing systems for HRI [23], even for teaming approach.

This chapter shows the developed model for supporting Human-Robot Teaming Interaction (HRTI) applications through a cognitive architecture [44, 43]. The architecture is built employing the Multi-Agent Oriented Paradigm (MAOP) with the support of the Beliefs-Desires-Intentions (BDI) model as agent's model [45, 47].

3.1 Perspectives and Analysis on Human-Robot Teaming Interaction

Let consider a team made up of a human and a robot who has to carry out a task, known to both of them, interacting with each other. During the design phase, all the actions that each of the two must perform and all the communications they must exchange during the interaction can be set. During the execution phase, if the task is to be carried out in a dynamic environment, the interactions between a robot and human and the environment inevitably change the state of the world that can thus provide new constraints or requirements for achieving the initial goal. The whole system's actual behavior comes out at runtime, and the team members need to be able to respond to changes efficiently. The most significant difficulty in designing such a kind of system is equipping the robot with the ability to select the best action to perform at runtime to achieve the team's goal. A possible solution may be looking at a human-human team [92].

Hence, the question is:

> *How does a human being act in a team whose goals are known and shared in a changing environment? And, how can this condition be managed in a team made by humans and robots?*

Human grounds his decision-making process on a set of factors, such as knowledge and actions, to achieve a goal. These factors, which could be described as reactive, occur when the situation being addressed is known, and the changes are foreseeable before an action plan is established. Also, when one is engaged in a cooperative and dynamic context, some psychological factors intervene. These factors represent the whole set of internal states used for triggering decisions. Generally, they are configured into the agent's knowledge as mental states, and they contain all mental representations and propositional attitudes for pursuing an objective.

Some possible representations or capabilities are:

- understanding to be not able to perform a certain action that can lead to the achievement of the common goal;
- refusing to perform a certain task, and delegate another team member based on a certain level of trust on his skills;

- feeling an emotional state (for example, euphoria) to want to do more than what was prescribed during the design phase and propose to the other component to carry out some actions or even decide to act independently on behalf of the other.

All these examples recall the state of mind for the agent that could contribute to the success of the agent's desires.

In recent years, cognitive architectures have been developed for reproducing biological systems to replicate human behaviors like decision-making processes in artificial agents. To the best of my knowledge, they have been applied to robots several times even they are not developed just for robotics. Thus we need to adapt the theory of cognitive architecture for robotics.

The architecture, realized for this research project, is focused on the use of self-modeling capabilities [90], elements of Theory of Mind (ToM) [182] and employs Multi-Agent System (MAS) [216] as software paradigm for robotics applications. Generally, in a team for pursuing an objective, a series of activities are performed by mates that compose a team. Each team member performs different processes concerning vision, speech understanding, and learning and reasoning processes during each one of these activities, such as decision-making.

Taking into account these aspects in human-robot teaming means to analyze and implement the processes of:

- knowledge acquisition and representation, including memory management;
- representation of the external environment;
- plans selection and creation;
- learning.

In [122], authors propose a standard model of mind (Figure 2.9). The standard model of mind is intended to be a reference point and a driving theoretical approach for developing and implementing cognitive architectures in several research areas. It is based on three well known cognitive architectures (ACT-R, SOAR and Sigma [8, 123, 118]) and presents a core composed of the cognitive cycle: perception, working memory, and action.

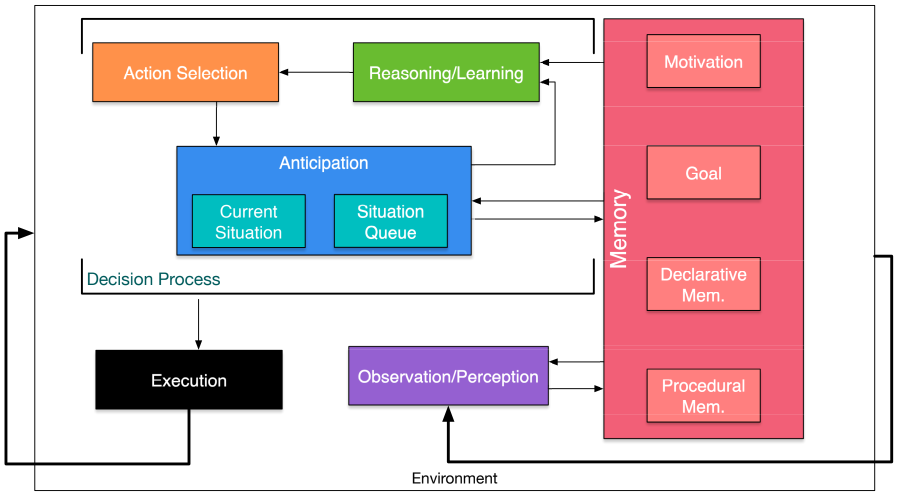


Figure 3.1: The Proposed Cognitive Architecture for Human-Robot Teaming.

The standard model of mind [122] was developed along with three different levels, from the purely biological level of mind to the more complex deliberative one. Hence, from simple behavior to complex one. The standard model is conceived to allow extending modules in each level; the lower ones constrain the higher levels.

The standard model of mind inspired the system hypothesized through a set of modules of a cognitive architecture for a robot teammate [44]: the *knowledge* and *memory module*, the *perception module*, the *communication system* and the *reasoner* that allows the robot to choose by taking into account the retrieved data. The robot's behavior is deliberated by the *planner* module, which interacts with the context in which the robot is plunged. Thus, the architecture is inspired to the standard model for creating a *self-adaptive agent-oriented architecture* for human-robot teaming interaction.

In the next section, the architecture is detailed.

3.2 The Cognitive Architecture

Figure 3.1 shows the developed cognitive architecture for human-robot teaming interaction.

The architecture is built over the standard model of mind without neglecting the basic system shown in Figure 2.1. The architecture is enriched with the modules devoted to sense, decide, and act for a robotics platform.

Based on the standard model of mind [122] and influenced by LIDA [74, 186, 75] the architecture in Figure 3.1 follows the theoretical framework of the Global Workspace Theory (GWT) [12, 13, 14]. The theoretical model underlying LIDA, and its adapted modules, drive the designing process. The strength of this architecture lies in the usage of self-modeling capabilities. Consciousness in LIDA is the mental counterpart of attention, that is, of the perception process. LIDA gives a complete representation of a cognitive cycle. However, it is not suitable to represent teaming interactions due to its internal features that do not fully support self-representation and theory of mind and make them complex to represent [166, 17] mainly for human-robot teaming interaction applications.

Besides, lessons learned from the literature suggest to build architecture on which an agent may learn from experience [197, 196] and at the same time, anticipate the outcome of its action and adapt to changing situation in the environment by acting driven by its motivation. We refer to these abilities in general as *pro-activeness* [109, 216, 27]).

These abilities imply specific architectural modules when we study robots that collaboratively interact with humans.

As shown in Figure 3.1, the cycle Monitor, Analyzing, Planning and Executing (MAPE), which is the basis of all the implementations of complex autonomous systems [10], is realized within the four main modules, *Observation/Perception*, *Memory*, *Decision Process* and *Execution*.

3.2.1 The Architecture Overview

The proposed cognitive architecture is composed of a set of agents that shares the environment and inanimate objects. Analysis and design of such a system must begin by choosing how we look at the system. For example, if the perspective is that of the agent (or the robot), it will see the system as the set of inanimate objects on which it can act. It will see itself with everything it has inside (its goals, capabilities, mental states), and it will see the other (any other agent) with everything the other has inside and with the set of objects on which the other can act.

So the human-robot system is, to all intents and purposes, a system of systems in which the environment is not external, like something with which it only interacts but is an integral part of the system itself. This is the main difference between our architecture and the existing ones. Such a feature allows us to represent and then implement a decision-making process based on a series of factors similar to human ones, such as the sense of self, elements of the theory of the mind, trust, emotions, and everything that can generate a mental state.

The most important change lies in the *decision process* developed within the *Anticipation* module and the representations of team desires that exploit the *Motivation* module.

The paradigm used for implementing the reasoning system into the cognitive architecture was the Beliefs-Desires-Intentions (BDI) model [28, 31, 29]. Agent's desires are commonly known as goals or objectives. In decision-making based on the practical reasoning system (PRS) [31] goals are necessary for triggering actions. For reaching an objective, goals were linked to the memory as well as the required knowledge for letting agents continuously performing operations within the environment autonomously. Agent's skills, environmental knowledge, pre-conditions are a part of the required concepts that have to be declared and stored into the agent's memory as beliefs; these factors are known at design-time by the programmer or acquired at runtime by the agent (or the robot).

The architecture is composed of different modules. Each one has characteristics and specifications that distinguish its activity.

As shown in Figure 3.1, the model starts acquiring OBSERVATION/PERCEPTION (in purple); this part of the system is in charge to sense and monitor the overall environment. Perceptions and observations are realized through a set of agents on which their goal is set up to continuously discover changes. New perceptions are categorized by source, and information is represented, firstly as agent's beliefs, and then organized into an ontology on which the concepts and the instances of concepts are stored. The second module concerns the MEMORY (in red). It contains declarative and procedural memory as well as goals and motivations for triggering plans and letting agents pursue their desires. A custom ontology implements both types of memories on which episodes and behaviors (including actions) for handling objects are stored. This process links the high-level concepts to the agent's abilities, letting the agent manage them as rules or beliefs, depending on their nature. An agent is in charge of handling this module. The role of the knowledge management agent is fundamental to the system. The next chapter shows a method and an algorithm developed for solving potential issues related to knowledge management in the architecture. The following chapter will provide more details about knowledge and its management system with architecture.

Thus, the architecture aims to endow robots with the *imagination* skill, or in other words, with the ability to anticipate the evolution of the scenario.

The MOTIVATION module (in red) triggers the anticipation and the action selection in the architecture during cooperation. It is a portion of knowledge that drives the decision process for handling human-robot teaming interaction; here, all the information and process for elaborating mental states about the team reside. It is the module devoted to the representation of inner and outer world beliefs of each agent in the architecture. Through this module, therefore, it is possible to make decisions about the actions being conveyed by the sense of self, by the ability to attribute mental states (belief, desire, intention, knowledge, capabilities) to oneself and others, and by the understanding that others have different mental states, by emotions, by the level of trust in the abilities (or more generally by the trust) of others and oneself.

The third module, DECISION PROCESS, is on charge to deliberate action to be executed. It is composed of a *Reasoning/Learning* module (in green), the *Action Selection* module (in orange) and the *Anticipation* module (in blue). The robot acts upon the results produced by the anticipation process. This process is in charge of producing a queue of situations representing the evolution of the faced task. Once the current situation is selected, and the agent knows how context it is acting, the system can produce a situation queue evaluating the current scenario. The current situation is computed based on motivations, goals, and all those elements in the memory, thus getting the execution launched. The queue of possible situations is also created. This queue is intended as a set of pre-conditions, objectives, and knowledge to achieve them, and post-conditions on the objectives. The robot can draw on all these elements at any time to respond to changes and still maintain its initial target.

The architecture is mapped into a multi-agent architecture that employs the BDI paradigm as a general reasoning system, thus giving the possibility to implement a human-robot interaction system with agents that may act according to their beliefs, desires, and intentions.

The last module is EXECUTION module (in black). It represents the latest part of the architecture. It hosts orchestrating agents' mechanisms to let them move toward their goals using actions or plans. In this module, the agent's action consists of phys-

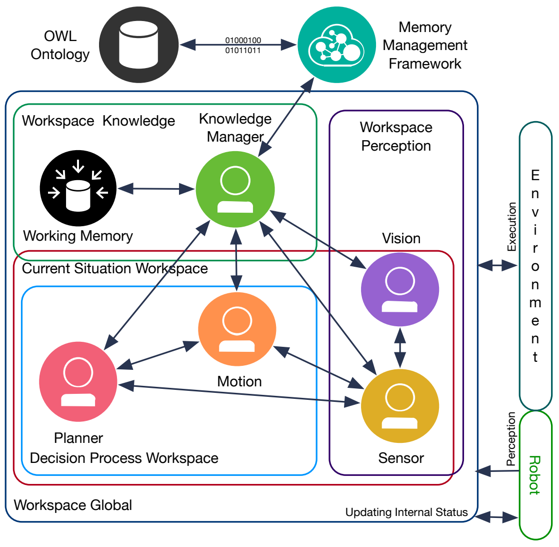


Figure 3.2: This diagram shows the multi-agent system in its part. It hosts agents for implementing basic behaviors for robots. Each of them could be distributed in more computational systems and replied in more than one single agent, according the needs of the programmer.

ical activity, such as enabling the motors of a robot for moving toward its direction or exchanging messages among the agent society.

3.2.2 Looking Under The Hood: The Framework Underneath The System

Since the beginning of this dissertation, it was claimed the ambition to develop a system able to perform interactions among humans and robots employing a cognitive architecture that is structurally different from ones proposed in the literature. In order to create a cognitive system that implements the architecture shown in Figure 3.1 was employed the multi-agent oriented paradigm and the Beliefs-Desires-Intentions (BDI) model [28, 29, 168]. Jason [26, 27] is the multi-agent framework used for programming the architecture. Jason uses BDI agents. A BDI agent in Jason uses the *practical reasoning* as a reasoning system.

Figure 3.2 figures out the agent architecture proposed through a *deployment diagram* where the agent society and its relationship are shown.

The architecture uses an OWL ontology [11, 24] for storing the knowledge. An

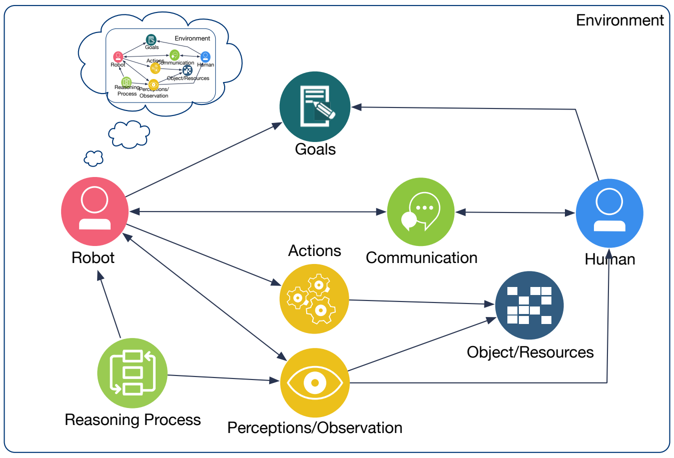


Figure 3.3: In figure a typical scenario in which robot and human collaborate. The cloud over the robot's icon indicates the level of awareness desired [188].

agent handles the architecture's knowledge through the framework Apache Jena[1]. Data and information to save are retrieved from the robot's sensors.

To let the system works, a set of agents cooperates to reach a common target. The reasoning system underlying the intelligent part of the agent's software is based on AgentSpeak language. This language implements the BDI model with theories and features described by Bratman [28]. The framework uses Java, and then it is fully compatible with libraries and other frameworks necessary to implement intelligent systems around decision-making processes and artificial intelligence models. The architecture is ready to support ROS [164] for managing robots. Each agent has an AgentSpeak file that describes its basic activity. It is possible to change agents' behavior, but in general, only the *planner agent* is in charge to define the mission of the system. Other agents support the architecture implementing modules of the cognitive systems.

The architecture employs artificial intelligent methods for implementing high-level capabilities that involve operations for classifications and recognition through state-of-the-art algorithms.

The multi-agent system supports both types of agents, the BDI and the extended one. They live together into a unique platform, and they exchange their knowledge using networking communications.

This is an overall description of the system; in the next section and chapters, an in-

[1]https://jena.apache.org/

depth analysis through the extension, models, and algorithms implemented for enabling the architecture to support machine consciousness abilities will be treated.

3.3 Extending the BDI Model

Today we want software able to anticipate our needs and to cooperate and to coordinate its activities with us. We also wish to have software that can autonomously and intelligently intervene and act in dynamic and changing contexts, operating as humans do. This section treats Jason's implementation of the aforementioned cognitive architecture, whose modules allow structuring the decision-making process by the agents' internal states, thus combining aspects of self-modeling and theory of mind.

First of all, let me recall the usual scenario:

> *An agent-human team has to cooperate to achieve a common goal in an environment not fully known, as shown in Figure 3.3.*

Preliminary Analysis From a human perspective, the cooperation process in which partners are called to share common goals presupposes a preliminary phase on which teammates plan activities and highlight potential sub-goals to reach. Even in a human-robot team, this operation is mandatory. All team members have to decompose the overall goal into a series of sub-goals. They should then be able to understand or learn which actions are needed to reach the objective. Finally, they should match their skills with the correct steps to perform, and eventually, they could also delegate some tasks to others.

Considerations This scenario concerns fully autonomous cooperative work that requires a complex software system with runtime adaptation to new situations that may lead to new requirements and constraints. Everything injected and evaluated at runtime cannot be defined during design phases, and therefore the system has to be handled as a self-adaptive system. In brief, a self-adaptive system [54, 35] must be aware of its objectives; it must monitor the working environment and understand how far it is and if it is deviating from the objective. Moreover, it must be able to adopt alternative plans, and it must also be able to generate new plans when necessary.

Important challenges in this field concern knowledge representation and updating, the selection and creation of plans at runtime, the invention of techniques for purposefully and efficiently conveying the (runtime) decision process. These challenges lead to different solutions depending on whether we look at the *architectural level* or *the system level*. At the *architectural level*, it is necessary to identify a set of cognitive modules for modeling the cognitive process behind decisions in the before said scenario. At the *system level*, it is necessary to employ the right technological solutions for coding and implementing a system working in changing conditions and continuous interaction with the human. This section's scope is to show how it is possible designing and developing the system-level counterpart of the decision process.

As you can expect, for achieving this result, it was employed the BDI model [84] and the Jason framework [25, 27]. It is clear that for implementing a HRI system able to operate in a dynamic and unknown scenario, the mere monitoring, analyzing, planning, acting (MAPE) cycle [10] is not sufficient.

As I said before, a possible approach for enhancing the quality of HRI system is endowing them with models of consciousness. This architecture is based on two important pillars of machine consciousness: *self-modeling* and *theory of mind*. In this system implementation, the most important role is covered by the decision-making process and knowledge management, and organization.

The decision-making process uses the agent's beliefs. This belief concerns information retrieved that could represent the agent's internal states, the surrounding environment, and human data. For implementing self-awareness and theory of mind, the human is also represented as a system's element, characterized by attitudes, capabilities, and constraints. The architecture aims to consider, as a crucial part of the decision process, the data coming from the capability of attributing mental states (beliefs, desires, emotions, knowledge, abilities) to itself and the other.

3.3.1 The Architectural Level for the Decision Process

The architecture in Figure 3.1 recalls the deliberative behavior of an agent that perceives, reasons, and acts. This architecture may be implemented, at the system level, by using Jason agents and their reasoning cycles (Figure 2.12).

Jason does not account for internal states as well as the fact that the environment may continually change in a not prescribed way or that the representation of the agents' environment also includes their inner world. Thus, the Jason reasoning cycle was extended to support these requirements without modifying its core functionality.

Figure 3.1 shows an extended version of the perception-action cycle in which modules are added for handling decision process and memory in the most useful way for the proposed purposes. It can be seen that an agent uses, for deciding which action to execute, inputs coming from the environment perception and from memory. The agent decides actions to perform after a reasoning process, and then it executes and continuously observes the results of its action on the environment.

The decision to include into the knowledge environment's concepts, goals, and motivations, and the need to split motivations and goals is necessary for handling the agent's reasoning process and implement the situation handler.

Besides, this lets the model consider the environment has a composition of objects, agents, and all that is inside each agent. These latter elements, such as, for instance, the awareness to be able to do something, constitutes the agent's self-model and then trigger the decision process along with the knowledge on the *static* environment.

Continuous observing and perceiving allow to constantly update and increase knowledge even during the execution phase. As said before, the *anticipation* module gives an agent the possibility of creating anticipation of an action result, some kind of post-condition on the state of affairs (the whole environment) at the end of each action.

In the *anticipation* module, it is included the elements for generating the *current situation* and a *queue* of possible situations, generated starting from the knowledge base and gained when the current situation is not applicable. The module, called *Motivation*, includes all the elements related to a human-like mental state; these elements have to be considered during the decision process. Examples of some of these elements are the emotional state, the level of trust in the other and in itself. Motivations, along with knowledge on goals, environment, including the self and own capabilities, lead to making purposeful decisions.

In summary, in this architecture, the design process has the same input of the standard model of mind [122], but it also has an intermediate part. Executing an action has effects on the environment and also on the internal state. The working memory, so as the reasoning/learning process and action selection, is affected by the ability to generate anticipation. The following section shows the revised Jason reasoning cycle, extended

Component	**Description**
Motivation	A Motivation is a form of belief for self-awareness modeling. Motivations are information that the agent has about itself. Motivation contains knowledge about the consciousness of the agent.
Motivation Base	A Motivation Base is where all motivations are organized.
Situation	A Situation is an extension of *Intention*. It represents the future state of affairs after the execution of agent's action.

Table 3.1: Definitions of new elements integrated into Jason.

and adapted for the target.

3.3.2 Extending the Jason Reasoning Cycle

The reasoning cycle, shown in Figure 2.12, develops three general processes: the knowledge update process, the handle event and the acting module.

As you can see, elements such as, motivation, goal, anticipation are not handled by the traditional Jason reasoning cycle. In some way, goals are implicitly treated in the plan and the plan library plus the context and the intention. However, it is not enough for our objectives. We need a structure to match goals with the anticipated results of single actions. This provides a cornerstone for realizing self-modeling.

Motivation serves for modeling all that is related to the internal state, the inner world. It is a kind of belief (the state of the environment) but they differ from a conceptual point of view. In fact, beliefs are values providing pre-conditions for a plan to be activated or selected. Motivation is, in some senses, a *superstructure* of beliefs including the agent's mental state.

Anticipation, in the two forms, current situation and situation queue, is used for generating a sort of simulation of the scene, the state of the world, if everything would go well. During the reasoning cycle, the agent decides which action to commit also after having evaluated the anticipation against the first component of the event, thus reinforcing the decision process.

In Table 3.1 and Table 3.2 a summary of definition of concepts used in the extended cycle (Figure 3.4) is figured out.

Jason reasoning cycle does not allow to implement these elements, above all the

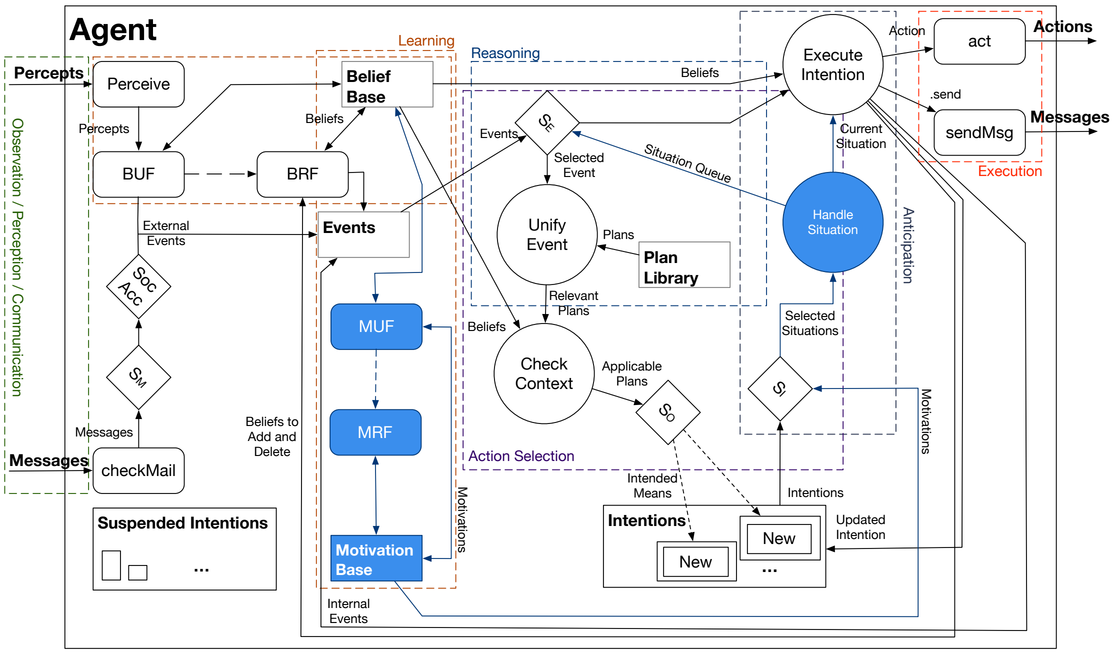


Figure 3.4: The extended reasoning cycle for Jason's agents.

anticipation. To support them, two new functions have been added to support them: the MUF and MRF with their related methods `muf` and `mrf` in the agent class (Figure 3.5), one other principal component, the *Motivation Base* and one new application process, the *Handle Situation*.

Beyond the changes made in the reasoning cycle, also selection functions were modified, as will be illustrated later in this section. Through motivations, it is possible to solve other complex plans and to force to select an intention to accomplish agent's desires. The deliberative process of actions is not limited to simple execution of a plan assigned to agents at design time and stored into the Plan Library, but the extension provides a valid alternative built on the motivation base. With this knowledge, the agent tries to define new alternative *applicable plans* that should be successful plans cause these are generated by checking the status of the agent internal state and the context.

Thus, Anticipation has the aim to produce a *queue of situations* where the head of the queue is the situation that the cycle is scheduling to handle for acting. The tail of the queue contains all possible future situations in according to the inner state of the agent including its motivation. The mechanism of the anticipation module recalls the perception loop described in [184].

Function	**Description**
perceive	The perceive method lets agents sense the environment and retrieve from it information; in our approach, the function gets information about itself through sensing the status of internal parts and the status of the mind. It implements the perceive function in Figure 3.4.
selectEvent	Our implementation takes in input a queue of events and motivations to select the plan that accomplish agent's desires (see Algorithm 2).
selectIntention	Selects an Intention to be further executed in the current reasoning cycle; it implements the $\mathcal{S}_{\mathcal{I}}$ function in Figure 3.4. Our implementation uses an intelligent scheduling that selects the right intention considering not only the queue of intentions but also a queue of situations that helps the algorithm to choice the better situation to compute in the handle situation. This function looks in future scenario to accomplish the desires and respects the agent's motivation (see Algorithm 4).
muf	*Motivation update function* updates the motivation base with given perception.
mrf	*Motivation revision function* revises the motivation base with a literal to be added (if any), a literal to be deleted (if any).
buf, brf, $\mathcal{S}_{\mathcal{M}}$, socAcc, $\mathcal{S}_{\mathcal{O}}$	These functions are not modified. For a description read the Table 2.2.

Table 3.2: Description of functions for the extended Jason reasoning cycle.

The reasoning process starts with a *perceive method*. In Figure 3.4 percept is handled by a *perceive function*, this latter handles two kinds of percept, the first is external perception or better, percepts that come from the environment and the second is the internal perception. In this last, the agent sense itself looking for features such as for instance execution time, stress-level, emotive state and other motivational features. This is the first significant difference with the original cycle. Inner perceptions are not defined as normal beliefs (*Literal*) but once agent perceives the feature this is converted into motivation. Each motivation is organized in a MotivationBase that is an extension of the BeliefBase (see in Figure 3.5). In the same UML diagram, it is possible to find some changes that we did to implement the cycle. As described we extended the *AgArch class* by creating another derived class called *AgArchMotivated* that implements the handling of internal perceptions. Knowledge is updated by means of *belief update function* and *motivation update function*. As said for the *buf method* in the original Jason reasoning cycle, the operating mechanism is not modified but we added a new branch to handle the inner state.

Initially every perception acquired is a belief with some operations the belief update function and the belief revision function convert internal perceptions in motivations and using the *motivation update function*, the system saves internal perceptions (aka motivations) in the *MotivationBase*. Processing motivations involves also motivation revision function to revise the MotivationBase. These functions are implemented in *AgentMotivated* class, this class extend the Jason *Agent* class. Information about the state of affairs of the environment is also perceived by the communication module. The reasoning contains methods to communicate with other agents.

For this purpose, Jason has two important functions to handle the communication between agents. The *checkMail method* is the first function that the agent does as shown in Figure 3.4 and it gets external messages sent by agents and organizes them into a mailbox assigned to the agent. Here, the function remains the same of the original cycle (*AgArch* class in Figure 3.5). The *social acceptance function*(aka *socAcc*) and its associated method are modified for our purposes and their functionalities are described in Table 2.2. After beliefs and motivations are updated, new events are generated. At this point, in the original cycle the *selectEvent* function, given as parameter a queue of events, returns the related *poll function*. Algorithm 1 shows as the original *selectEvent* function works. Algorithm 2 shows how it was modified the *selectEvent* function in order to consider the situation.

Algorithm 1: The *selectEvent* algorithmn implemented in Jason.

```
1: procedure $\mathcal{S}_{\mathcal{E}}$(Queue<Event> events)
2:     queue $\leftarrow$ events
3:     return queue.poll()
4: end procedure
```

So $\mathcal{S}_{\mathcal{E}}$ function implements an algorithm that, given as parameters the event queue and the situation queue generated at the previous cycle, executes a *generateEventsQueue function* to obtain a queue of events that are reformulated considering the queue of possible future scenarios carried by the queue of situations (if any). Once the algorithm has

Algorithm 2: The *selectEvent* algorithmn implemented in the extended Jason reasoning cycle.

```
1: procedure $\mathcal{S}_\mathcal{E}$(Queue<Event> events, Queue<Situation> situations)
2:     Queue<Event> eventsQueue ← events
3:     Queue<Situation> situationsQueue ← situations
4:     Queue<Event> queue := generateEventsQueue(eventsQueue, events
5:                                Situations)
6:     Event selectedEvent := queue.poll()
7:     return selectedEvent
8: end procedure
```

generated the *Selected Event* as shown in Figure 3.4, such a result of the *diamond* $\mathcal{S}_\mathcal{E}$ function, a *unification process* is fired. It unifies events evaluating the previous results and a list of plans generated by a *plan library* module. The result of this process is a list of relevant plans. Once plans are selected another process is started.

The following process *check context* (Figure 3.4) is the same of the original cycle. This process is not influenced by the *motivation module* or *anticipation module* because the relative check context for our scope is included in the *select event function* with the *generateEventQueue* function as you can see in Algorithm 2. After the context is verified, the process generates *applicable plans* that are given as input to the function *applicable plan selection function* denoted as $\mathcal{S}_\mathcal{O}$ in Figure 3.4. The output of this function is called *intended means* that is the chosen applicable plan because the actions executed by that plan is the means that the agent intends to execute to handle the chosen event. The intended means is converted into intentions and will be handled by the *intention select function*. The list of intentions, this is given as argument for the *intention select function* and through a handle situation process our reasoning cycle extracts from this latter process the current situation, that is shown in the UML diagram a derived class of Intention. The original *intention selection function* $\mathcal{S}_\mathcal{I}$ implements a poll function as argument as described in Algorithm 3. Our implementation, instead, involves two functions (Algorithm 4). The first creates a simulation process that implements a structure similar to the one described in [184] and the second verify the queue of situations generated at the step before. In Algorithm 4, it is proposed our pseudo-code for the reasoning cycle.

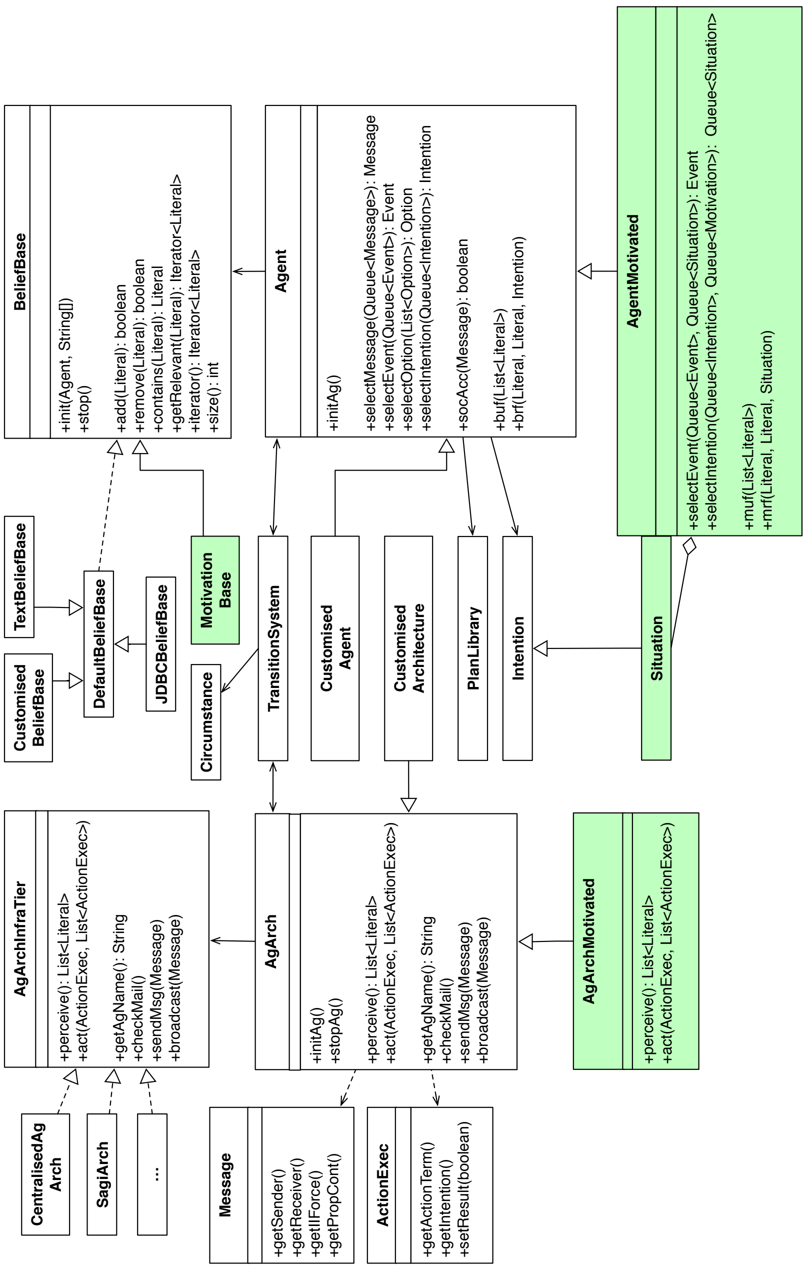

BeliefBase
+init(Agent, String[])
+stop()
+add(Literal): boolean
+remove(Literal): boolean
+contains(Literal): Literal
+getRelevant(Literal): Iterator<Literal>
+iterator(): Iterator<Literal>
+size(): int
Agent
+initAg()
+selectMessage(Queue<Message>): Message
+selectEvent(Queue<Event>): Event
+selectOption(List<Option>): Option
+selectIntention(Queue<Intention>): Intention
+socAcc(Message): boolean
+buf(List<Literal>)
+brf(Literal, Literal, Intention)
AgentMotivated
+selectEvent(Queue<Event>, Queue<Situation>): Event
+selectIntention(Queue<Intention>, Queue<Motivation>): Queue<Situation>
+muf(List<Literal>)
+mrf(Literal, Literal, Situation)
TextBeliefBase
Customised BeliefBase
DefaultBeliefBase
JDBCBeliefBase
Motivation Base
Circumstance
TransitionSystem
Customised Agent
Customised Architecture
PlanLibrary
Intention
Situation
AgArchInfraTier
+perceive(): List<Literal>
+act(ActionExec, List<ActionExec>)
+getAgName(): String
+checkMail()
+sendMsg(Message)
+broadcast(Message)
AgArch
+initAg()
+stopAg()
+perceive(): List<Literal>
+act(ActionExec, List<ActionExec>)
+getAgName(): String
+checkMail()
+sendMsg(Message)
+broadcast(Message)
AgArchMotivated
+perceive(): List<Literal>
+act(ActionExec, List<ActionExec>)
CentralisedAg Arch
SagiArch
...
Message
+getSender()
+getReceiver()
+getIlForce()
+getPropCont()
ActionExec
+getActionTerm()
+getIntention()
+setResult(boolean)

Figure 3.5: UML Class Diagram for integrating Jason extension.

Algorithm 3: The *selectIntention* algorithm implemented in Jason.

```
1: procedure $\mathcal{S}_{\mathcal{I}}$(Queue<Intention> intentions)
2:     queue ← intentions
3:     return queue.poll()
4: end procedure
```

Algorithm 4: The *selectIntention* algorithm implemented in the extended Jason reasoning cycle.

```
1: procedure $\mathcal{S}_{\mathcal{I}}$(Queue<Intention> intentions, Queue<Motivation> motivations)
2:     intentionsQueue ← intentions
3:     motivationsQueue ← motivations
4:     Queue<Situation> situations
   := simulate(intentionsQueue, motivationsQueue)
5:     Queue<Situation> selectedSituations := verify(situations)
6:     return selectedSituations
7: end procedure
```

Regarding code, the situation queue is internal to the agent so the code structure is not changed because a poll function is always called on the result of the previous function. So the execute intention process has as input a situation that is an intention as evidenced in the UML diagram in Figure 3.5. This process tries to perform all actions included in the current situation and the agent is able to produce the *act* function, after which the reasoning cycle could be restart.

3.4 Summary

This chapter has shown the realized model for an Human-Robot Teaming Interaction (HRTI) scenario. The process to create the cognitive architecture starts with the model's definition and later implements it, exploiting paradigms and frameworks at the state-of-the-art. Perspective on analysis and implementation issues were treated to do this. Those analyses drove the model's definition as well as the abilities required for realizing interactions. The cognitive architecture was developed employing the multi-agent paradigm and followed guidelines and approaches common in cognitive systems' literature. The architecture's core lies in the deliberation process, revised and extended to support abilities for the target. As discussed in the previous section, each module faces a specific issue, and it employs perceived data for selecting which plan is more reliable to select to be executed. Plans may be written at design time by an agent programmer or, as shown in this dissertation, generated at runtime, exploiting the algorithm developed for this purpose. The extended version of the Jason reasoning cycle, shown in this chapter, assists the teams' decision-making process. The alternative cycle added to the standard cycle is an alternative path to reason, employing logical models that the programmer thinks for letting cooperation. The representation of the agent's knowledge of motivation does not corrupt the normal cycle. However, it helps the agent select the proper plan to reach the team goal faster than usual and the programmer to identify and split information, properly cataloging them. For doing that, Jason's classes were created and modified to support it. This enhancement was possible by implementing Jason's interface and extending Jason's classes. To model and implement motivations, also belief concept was extended instead of creating a new element, introducing a motivation base with functions for revising it. In next chapters will be presented with other important modules developed in this 3-years job. This dissertation and the work during these years were oriented onto a definition of a cognitive model and architecture for supporting the Human-Robot Teaming Interaction (HRTI), algorithms, methods and produced learning systems aid and will support the continuous building and integration of this cognitive system letting the entire architecture behave as a human through the creation of new operation at runtime.

Chapter 4

Knowledge Acquisition and Management

The Chapter 3 introduced the cognitive architecture for Human-Robot Teaming Interaction (HRTI) applications.

Contexts where agents and humans are required to collaborate and cooperate in a human-like fashion are complex systems where a high degree of self-adaptability of every component is demanding. A fundamental ingredient when developing and implementing this kind of system is *knowledge*. It is crucial in deciding which action to perform to reach an objective and to behave in a self-adaptive way. The problem of knowledge modeling and representation becomes more and more urgent if the agents' operation domain changes at runtime. Knowledge has to be updated and handled while the system is in execution.

This chapter is organized as follows, in the Section 4.1 a model for designing a knowledge management system for a multi-agent system is shown, the objective is to show how to handle knowledge in a system where multiple agents may access the same memory resource [46]. Multiple access at different times could produce a phenomenon of asynchronous memory, and so, agents could reason on expired knowledge.

The Section 4.2 introduces a new computational model on which the knowledge is organized automatically by the cognitive architecture. The idea is to realize a method able to link unknown concepts with their *semantic relatives* that are stored in the robot's knowledge [42]. The model works in an unsupervised way.

4.1 Designing a Memory Architecture for Representing Knowledge

Storing data, information, and relevant perceptions let humans learn, reason on facts, and decide the right action to pursue an objective. Similarly, to build agents and robots able to work in specific environments, it is useful to give them the required know-how for doing actions and getting the jobs done. Implementing this feature on robots and agents involves defining a suitable data structure that can emulate a memory architecture.

For instance, in a complex system such as a multi-agent system, a memory architecture is necessary for the set of agents for planning and acting operations. The memory architecture is used for managing the knowledge of cognitive architecture. As shown in Chapter 3, cognitive architecture is developed using a set of agents that have to cooperate and collaborate to pursue the system's goals. For letting agents share their perceptions, it is necessary to organize how to represent the knowledge, organize it, and share it with other modules of the architecture.

Before starting the memory architecture description, the capabilities for acquiring new perceptions and operating in the environment have to be described. Chapter 2 described the multi-agent framework used for developing this cognitive architecture, and Chapter 3 presented the cognitive architecture and discussed changes and extensions proposed for implementing it as a solution for the HRTI domain. We see that the agent class uses other classes to perform functionalities (to recall see the Figure 3.5 in Chapter 3). The AgArch class is responsible to *perceive* and to *act* (for a description of these functions see Chapter 2). With these methods, the agent can perceive the environment and act on it after a decision-making process. It is possible to endow an agent with other capabilities; for instance, a possible way is the CArtAgO meta-model [171] through the namesake framework. This meta-model is also know as Agent & Artifact meta-model [153].

Notwithstanding the focus of this chapter is on the part of the cognitive architecture in charge to handle the knowledge, for this type of cognitive system based on the BDI model, the knowledge is generally contained into the Belief Base (BB). The implementation of belief base in Jason is shown in Figure 4.1.

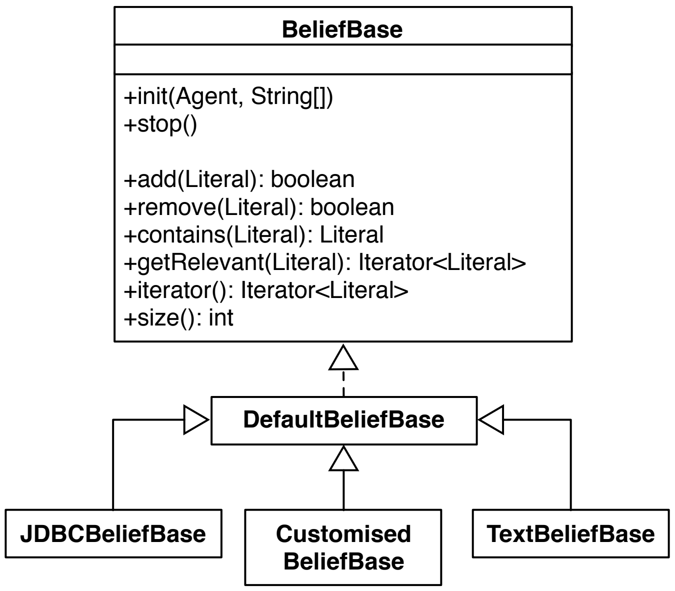


Figure 4.1: Jason Belief Base UML redrawn from [27].

4.1.1 Modeling Knowledge

As seen, cognitive architectures own a memory system for handling knowledge. For building a knowledge management system that implements a human-like memory architecture, Tulving states in [194] the fundamentals. The memory should be divided into two different types, the *episodic memory* and the *semantic memory*. The *semantic memory* allows retrieving information that has not been directly stored in the global knowledge base system. Semantic memory does not save perceptions but instead cognitive links to input signals. The semantic memory architecture is much less susceptible to losing information concerning the episodic one. *Episodic memory* stores temporal information about occurring episodes or specific events and merges them, when possible, to the temporal-spatial relation. Episodic memory architecture may store perceptible events in terms of properties, characteristics, or attributes. Hence, to handle system knowledge efficiently, we need architecture letting the agents storing all the information about perceived data and, at the same time, create and/or maintain semantic mapping among them.

The realized memory architecture tries to emulate the memory proposed by Tulving [194] through a semantic representation based on ontology and the agent's beliefs. Agents use the ontology to store all information, organized by their relations, and the agent's beliefs to plan during the decision-making phase.

The memory architecture employs the OWL ontology [24] for storing and representing information with all the semantic relationship. The framework used for managing OWL ontology is Apache Jena [108].

The idea is to implement a long-term memory, procedural, declarative, and episodic, and working memory within a short-term memory for acting and pursuing objectives employing only the memory relevant for the current task.

In the following sections, the memory architecture and properties, methods, and functionalities will be discussed to show how the problem of out-of-date or expired beliefs was faced and how it was possible to organize an ontology at runtime with new stimuli following their semantic nature.

4.1.2 Handling Knowledge in a Multi-Agent System with OWL

A cognitive architecture is endowed with a semantic and episodic memory to support the decision-making process and other cognitive capabilities. Organizing the memory in this way is suggested in models that inspired the designing of the proposed cognitive architecture: the standard model of mind [122], and LIDA [186].

The usage of ontologies as OWL for representing knowledge in a multi-agent system is a common practice [111] for implementing the knowledge representation module. Agent's beliefs are handled to be saved in the ontology, *handlers* are written as implementation or extension of existing Jason's methods [26, 27]. Jason capabilities were extended with the support of CArtAgO [171] framework by using artifacts and the definition of a specific workspace.

Briefly, the CArtAgO framework realizes artifact-based working environments in a Multi-Agent System (MAS) [171]. The framework makes extensive use of *artifacts* as abstractions to handle tools, objects, and resources belonging to the surrounding environment. Tools, objects, and resources may be used and manipulated by agents to solve several tasks such as cooperation, actions, or perceptions. An artifact is a `.java` file to implement methods that could be used in AgentSpeak plans by the Jason agent.

Figure 4.2 shows a generic approach used for handling knowledge with CArtAgO artifacts by an agent developed in Jason.

Agents deployed into the knowledge management workspace can use these artifacts.

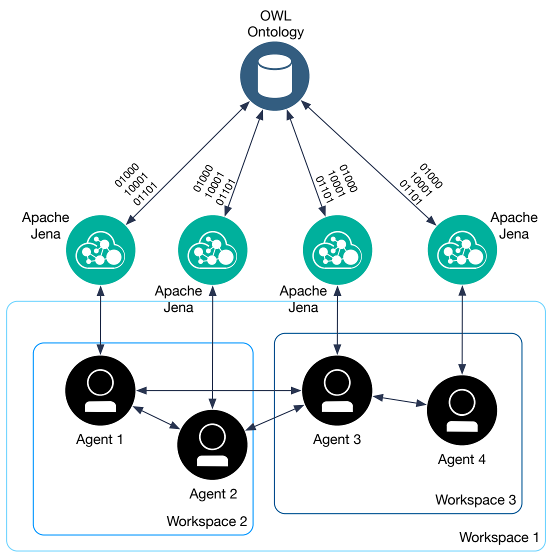


Figure 4.2: Using an OWL Ontology within a Multi-Agent System (MAS).

Each artifact is used for managing resources and for organizing concepts using semantic relationships. So, a first general approach is figured out in Figure 4.2, here all the agents can interface with OWL ontology using the Apache Jena framework. The issue in this type of memory architecture is that each belief may be revised at any moment, and so, an agent could get an expired belief, so the reasoning process could fail the scheduled plan.

Figure 4.2 contains three different workspace. Workspace 1 contains tools and methods that could be used by every agent in the system; this means that the knowledge acquired using these tools could be accessible by all or could be acquired from every agent that uses the artifact in the plans defined in `.asl` file. The proposed scheme implements, in each agent, the functions interfacing with Apache Jena; concerning its complexity, this means that a connection through the ontology for each agent will exist. The only form of communication is related to the transmission and the execution of the regular plan. In workspace 2, such as in workspace 3, agents communicate with each other with internal communicative action as `.broadcast` and `.send` to catch or retrieve information from and to the agent society, and artifacts of each workspace could be used only from agents deployed into the related workspace.

The way proposed above maybe not reliable for this type of application. Cognitive architecture needs a memory system that can perform operations during the execution

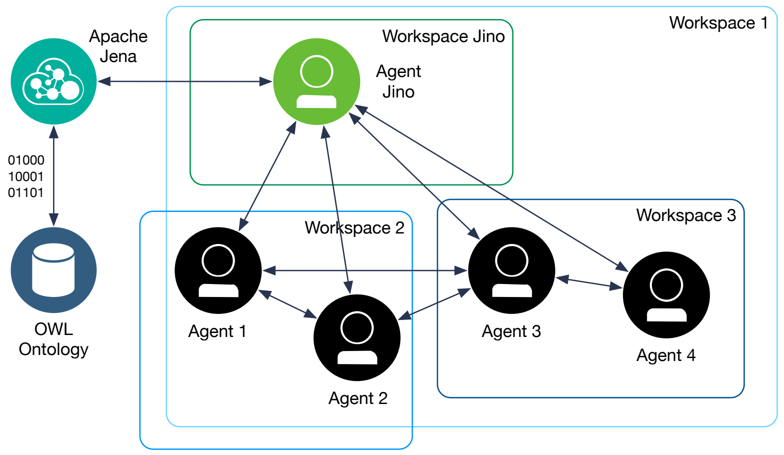


Figure 4.3: Proposed approach to implement semantic and episodic memory system using Jason-CArtAgO e Jena-OWL.

time.

Thus, a possible solution is developing a centralized approach, as you can see in Figure 4.3. Developed knowledge system can emulate the mechanism of a generic belief base such as the Jason one; it works well at runtime and at the same time guarantees inferring logical consequences from a set of axioms.

This approach aims to connect the entire system to an ontology through another agent responsible for operating actions for keeping synced and updated beliefs inside the *agents society*.

This method was revealed to be useful and simple to realize a centralized system for distributing knowledge. The next section will treat in more detail the approach.

4.1.3 The Jino Approach: A Proof of Concept

Let us suppose agents with the goal of understanding objects not present in their knowledge base. As humans do, agents start from whatever they know and through interactions with humans.

The agent system developed using the first approach (Figure 4.2) revealed to be slow and hard to implement the cause of the need to handle all possible occurring cases.

Also, agents' connection latency and the delay on possible plans actuation entail a series of problems that could create inconsistency in the decision process.

Some errors propagated in some semantic structure functionalities because of the high latency that severely decreased performances and did not let agents adapt to changing situations. Indeed, each agent that got a deprecated belief could not know which version of belief was correct. Besides, CArtAgO produces *object properties* that represent perceived beliefs for the agent; this type of beliefs are used for deciding the plan to schedule.

The crucial difference between the two approaches is in the memory management system, which was implemented by introducing a specialized agent called Jino.

Jino uses a customized agent-type that considers a series of plans composed to handle the memory system. Jino lets us manage the episodic and the semantic memory but also all the information acquired by beliefs.

Through the communication system, Jino maintains the system synced with all the members. The value of beliefs is updated at runtime. When an external agent asks for a specific belief, Jino knows whether this is reliable or is better to check into the ontology to retrieve a newer belief version.

Jino's workspace owns all artifacts necessary to manage the perceptions coming from agents. The communication system always guarantees the direct connection between Jino and each agent. To acquire runtime knowledge, Jino gets started each time a belief changes or an event occurs; this fact lets Jino add, remove, or update some beliefs and manage others simultaneously. Thus, Jino manages to change knowledge using its working memory without any external ontology reasoner; this lets the system use resources at runtime and have beliefs updated and synchronized.

Jino, across the communication system, informs all agent-society about changes using the broadcast method. Whenever Jino believes that a certain belief is necessary only for a specific agent or a group of agents, it takes over to communicate only to the interested agent/group using the `.send` method. Moreover, when new knowledge is acquired, Jino firstly saves the knowledge base's result from constructing the long-term memory and then lets OWL's reasoner infer new concepts.

Only one connection with the ontology across Jena Framework is present. The reasoner is launched only when it is essential and not when a single agent desires a specific belief. Jino takes advantage of an internal memory representation that makes belief usable when someone asks.

4.2 Robot Incremental Knowledge Acquisition at Runtime

In the human-robot teaming cooperation scenario, robots interact and cooperate with humans and the environment to reach a shared and common objective.

Whether human or robot, a team member possesses a prior knowledge apt to support him to carry out his activities. An Artificial Intelligence system for working in a team needs a knowledge management system to organize his knowledge. Besides, collaborative tasks with humans could manifest unexpected events; this makes the environment and the context dynamic and unpredictable.

To the five fundamental lines proposed by J.S. Albus [4], other guidelines were added to the road-map for creating a human-level artificial intelligence. For developing a Human-Robot Teaming Interaction (HRTI) architecture, it was considered a sub-set of them, accordingly, with the requirements dictates by the principal research area (the HRTI): (i) attention awareness and intentionality, (ii) believable behaviors, (iii) emotional intelligence, (iv) imaginary.

The cognitive architecture described Chapter 3 tries to serve the lines proposed in the road-map. For exploring everything related to the meta-cognitive level, a first step is developing a knowledge manager in charge to organize efficiently new concepts acquired at runtime by classifying them.

The previous section (Section 4.1) shows how the knowledge management system was designed for adapting to the cognitive architecture requirements. This section shows a computational model to increment the robot's knowledge at runtime through an energy-based model built with natural language processing support.

4.2.1 The Kewap Model

The model proposed here is based on the principles of cognitive semantics. The knowledge is handled, representing concepts as mental schema [141]. The mental schema plays a fundamental role in the conceptualization of a new perception by a robot, and hence for knowledge discovering. The robot becomes capable of mental situations about a single perception. The computational model lets the robot acquire *imaginery* and *critical mass* skills [4] because it becomes able to grow in environmental awareness as a human could do under the same condition. This model aims to permit the development of environmental awareness and self-consciousness for knowledge acquisition. The solution adopted for doing that is in employing the ontological structure to represent the mental schema for the robot. Distribution over the space of possible mental schema from the ontology allows estimating the probability of each possible schema given a perception.

Once the schema for perception is inferred, the perception is acquired by blending the correspondent new concept and the related instance (if the concept does not already exist in the knowledge base) or only the instance (if the concept already exists), and the knowledge grows.

The acquisition of new concepts is at runtime by perceptions. Each time that new one has detected, the knowledge grows, expanding the concepts space and instance space if the concept is unknown, or only the instance space if the concept is known. The model integrates sub-symbolic and symbolic approaches, letting it be transparent for humans.

The computational method, called *Kewap* (KnowledgE WrAPping), is modeled using two processes: (*i*) the mapping process, and (*ii*) the merging process. The method starts taking as a parameter a label. The label is a string that describes the new perception perceived by a robot, or an agent, with its sensing module.

Within the cognitive architecture in Chapter 3, the Perception/Observation module senses the environment, and when a new perception occurs, the module in charge to manage the knowledge tries to map the concept in the knowledge using *Kewap*. The perception is handled as a label: a string related to the meaning of the perception. It is important to mention that the model meets the assumption of insufficient knowledge and resource [205] philosophy because it does not need ontological bases rich in concepts and relationships. It works even when the ontology is limited.

Once the string corresponding to a vocal stream is detected or an object is recognized, the *mapping process* allows to infer if such an entity is already modeled in the knowledge, and in this case, the perception is instantiated. Otherwise, the concept has to be acquired and correctly allocated in the ontology by the *merging process*.

For the purpose, the surface form of the ontological label and the surface form of the perception has to be mapped; the similarity measure used is the one proposed in [157] that computes how the ontological labels and an external word are close.

This measure keeps into account both semantic and syntactic components. The measure represents a similarity distance between the labels (words) w_1 and w_2.

It is the weighted sum of the Jaro-Winkler distance [212] and the Wu-Palmer distance [219], then:

$$sim(w_1,\ w_2) = \delta * jaro(w_1,\ w_2) + \gamma * wup(w_1,\ w_2). \tag{4.1}$$

The main motivation to consider the Jaro-Winkler distance is the characteristic of the strings to compare, that are short words as the labels of the ontology, instead, the Wu-Palmer is considered one of the best semantic measures in literature.

The parameters δ and γ represent the quantity of syntactic or semantic similarity to consider. The choice of these parameters depends on the type of application domain. The constraint to determine these values is $\delta + \gamma = 1$.

To describe the model, let consider the set of ontological labels O (the ontology) and the perception p.

The mapping process is modeled by the map function, that is:

$$map(p) = \begin{cases} o & if\ \ max_o > \tau \\ mer(p) & otherwise \end{cases}$$

where the mer function starts the *merging process* next defined, while the max_o is the maximum value of the set S_p where $S_p = \{sim(p, o) \mid o \in O\}$.

The map function returns the concept o in the ontology if o matches with the perception p according to the similarity value, which has to be greater than the threshold value τ[1]. It means that the perception p is similar to o, so it already exists in the knowledge, and p becomes an instance of o.

The *merging process* starts when the max_o value is less than τ; to merge a new concept in the ontology requires to compute its correct allocation.

The model inspired by a probabilistic approach already proposed for the *Probabilistic Tree Substitution Grammar (PTSG) induction* [50]; it aims to define a power-low distribution over a space of production rules that combine the grammatical linguistic structures.

For estimating the probability of each production rule, the statistics for linguistic

[1]An higher value of τ, means that the two concepts have to be very similar, a lower level, means that concepts not completely similar will be swapped.

structures they represent has to be learned from text corpora. Parsing a string by using a PTSG means to find the most probable combination of rules for the given string. A Pitman-Yor Process (PYP) [158] is used for this purpose. The PYP represents a rich and flexible class of random probability measures used as the prior distribution in Bayesian nonparametric inference.

The key idea is that the ontology of the robot is a set of ontological structures, that are the nodes and the triples representing properties and relations. Thus, a PYP process is defined for estimating the correct linking between these structures and a given perception in the same way as the previous string parsing.

It is worth noting that *Kewap* does not need sufficient statistics for computing such probabilities because the process works over the text representing the perception and the label of the ontology.

The PYP process assigns a probability to each ontological structure and then finds the most probable combination of these structures for a given perception.

Formally, the merging process is modeled by the PYP distribution $mer(p)$ over the ontological structures, that is:

$$mer(p) \propto PYP(\alpha, \beta, G_o) \tag{4.2}$$

Where α and β are the hyper-parameters of the process that influence the shape of the distribution, while G_o is the base distribution which determines which fragment trees will fall in support of $mer(p)$. A *fragment tree* is an excerpt of the enriched taxonomy corresponding to O.

In particular, G_o is a function that, similarly to $map(p)$, involves the symbolic linguistic properties of p over space $o \in O$, and allows to choose the more plausibly fragment tree given p.

In this sense, the method is hybrid, involving a sub-symbolic process integrated with symbolic properties.

The following section shows proof of concept to demonstrate the method's functionalities and obtained results. The proof of concept describes a task of *self-repairing* for a robot. The objective is endowing a robot with the ability to understand, looking at its log, which component (hardware or software) has issues at runtime. If some log description lacks, the *Kewap* method is used to predict with which concept the perception

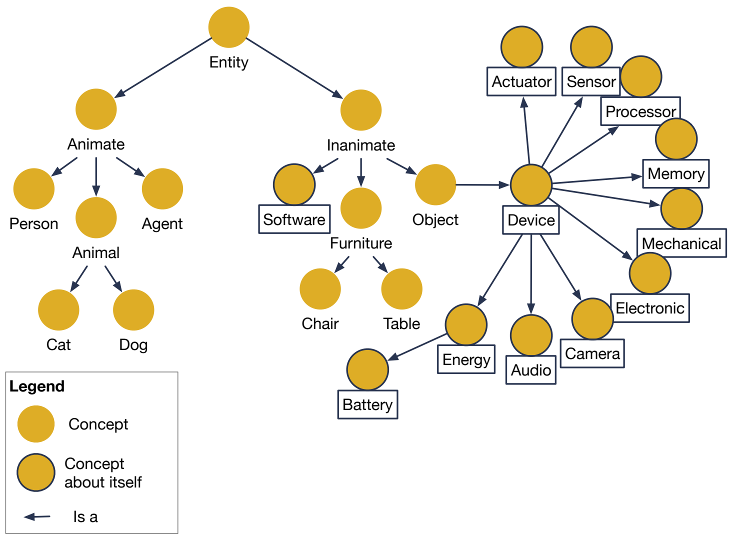


Figure 4.4: The fragment of the ontology including all the concepts about itself. These concepts are framed.

establishes the semantic relation.

4.2.2 Self-Repairing Task: a Robotic Proof of Concept

This work aims to make a robot *aware* of objects in dynamic environments and *self-conscious* of its knowledge by automatically updating it when a new entity is perceived. If the robot cannot "understand" an object, it would not be able to use and refer to it during task execution, leading to compromise in the team's collaboration.

The ontology is one of the possible strategies to follow for equipping the knowledge level of cognitive agents, and in some cases, it can present little limitations [134]. The problem of modeling dynamic knowledge acquisition processes can be plausibly enriched by using conceptual spaces [83] representations to align with the ontological representation [133].

Ontological representation was considered because it is the most widespread and also allows standardization by the foundation ontology. The ontology is often manually encoded before the robots are deployed. If the robot is situated in a dynamic environment, a dynamic ontology is expected to grow when it perceives new objects.

Let consider a simple scenario where a robot is plunged into an environment whose high-level knowledge is represented in Figure 4.4. The goal of the robot is self-repair.

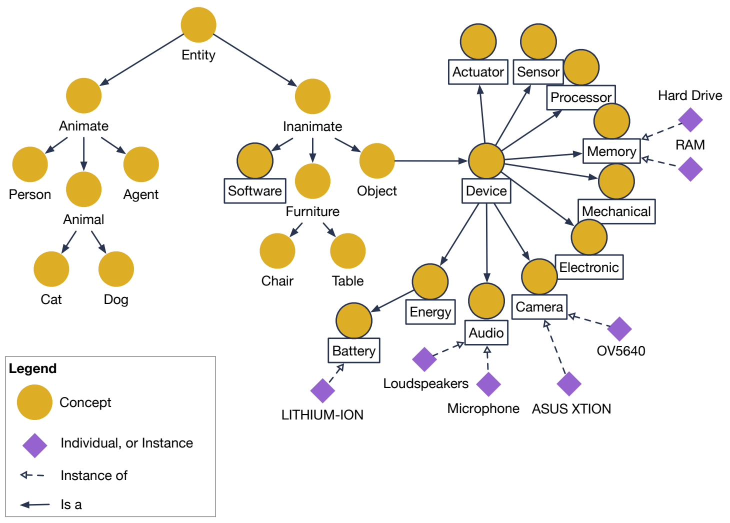


Figure 4.5: The fragment of the ontology includes all the concepts about the environment and some instances.

Elements in ontology concern the physical components of the robot, which it perceives as integral parts of the environment. In Figure 4.5, the same ontology is enriched with some concepts the robot has perceived and then instantiated; in some sense, it “knows” these concepts and recognizes one or more instances of them in the environment. Physical self-repair could become critical in applications where no humans are around to assist or repair the robot, which has to heal itself. Also, this can be useful for cooperating with humans to repair other machines.

Knowing the damaged resource is the first step for self-repairing; the robot acquires awareness of such a resource and becomes able to understand how to repair it, for example, by identifying what object can be used in place or the set of necessary actions for replacing.

Generally, the robots can self-diagnose and detect if and what component is in trouble. The main goal of diagnosis is to check every device and its functionalities and publish whether the device has or has not an error. Until this moment, the robot has no awareness about this device. It passively communicates to the human the internal state, but it cannot understand such a state.

The robot has to semantically conceptualize the device for the opportune intervention to start the self-repair task.

In other words, by self-diagnosis, the robot knows that a device is in trouble, but it

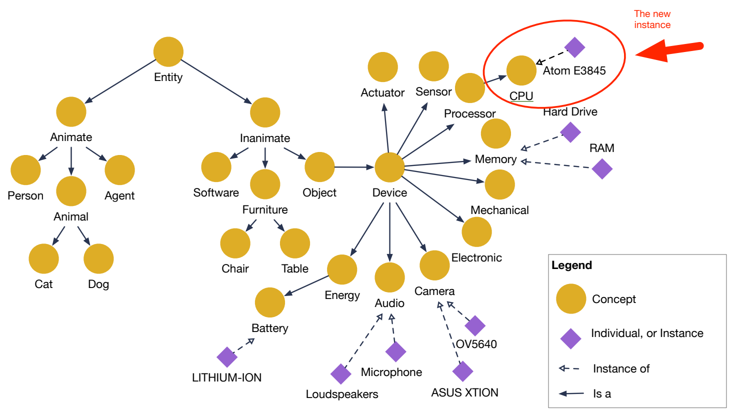


Figure 4.6: The knowledge acquisition related to the *CPU* concept with its instance represented by the diamond shape.

does not know such a resource, which remains an abstract concept until it is not acquired in the knowledge. The robot might not know anything in advance about its devices, or it could have partial knowledge about itself.

Experimental Setup

The simulation was developed using Pepper Robot[2] and the diagnostic module simulated hardware trouble mimicking the native naoqi library. As a robot's vocabulary was used WordNet [144], a large electronic lexical database for English developed and maintained at Princeton University since 1986. WordNet was used to calculate the similarity, using *Kewap*, between the perceived label and the concept's label. The simulation started listening to the diagnostic module.

This diagnosis returns a new perception of d with the name of the damaged resource. Let suppose that the `CPU` is this resource, and so the perception is $d = cpu$.

As it can be seen from Figure 4.5, the CPU is an element not known to the robot. The robot is able to process the new perceived element, to link it to a known one at to produce a new instance (see Figure 4.6).

When a new entity is perceived, two different cases can be considered:

1. the abstract concept is already modeled in the knowledge; a new instance for that concept has to be created corresponding to the perceived entity. We will refer to

[2]https://www.softbankrobotics.com/emea/it/pepper

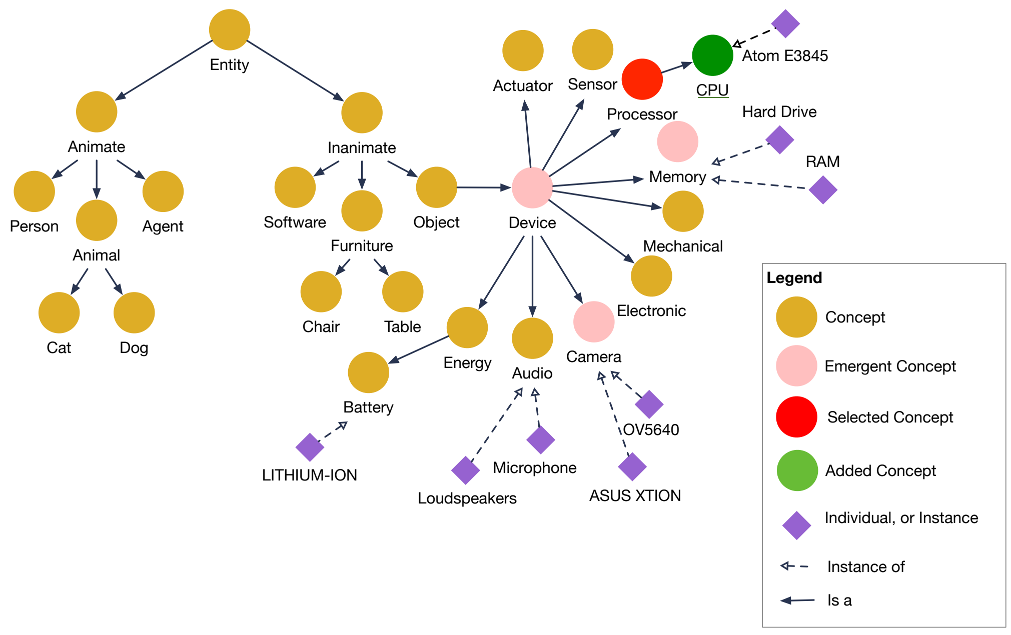


Figure 4.7: The knowledge acquisition related to the *CPU* concept with its instance represented by the diamond shape. The emergent concepts are highlighted, among them the more probable is the candidate parent.

this process as *mapping process*;

2. the abstract concept is not modeled in the knowledge; both the concept and the instance have to be created. The concept becomes persistent. We will refer to this process as *merging process*. This, it is the case of the previous example that considers the CPU resource.

In both cases, the problem is related to infer the presence/absence of the concept in the knowledge and then identify the correct allocation of a new concept/instance in the ontology, leading to the *introspection* of the robot.

For this case study, the parameters of equation 4.1 are set with the following values: $\delta = 0.7$ and $\gamma = 0.3$. The PYP was initialized with $\alpha = 2$ and $\beta = 0.3$. These parameters values were determined experimentally running the experiment several times in order to find the optimal solution for the problem.

The mapping process *map* will invoke the *merg* function because the similarity measure computed by the equation 4.1 is under the threshold for each concept in the ontology.

Ontological Structure	*Result*
[Processor]	**0.04662**
[Camera]	0.03195
[Memory]	0.01704
[Device, Processor]	0.01251

Table 4.1: Kewap results for the CPU resource.

The Table 4.1 shows the results of the merging process. The concept *Processor* is more probable than the other structures in the ontology; a new concept *CPU* is created as children of *Processor*, with the correspondent instance. Since this moment, the robot has conceptualized the resource, and it can refer to the knowledge about the processor for self-repairing it or for identifying the possible new processor among a set of available spare parts.

New knowledge is discovered, and the ontology is updated as drawn in Figure 4.7. The same figure shows the *emergent concepts* for the resource, that is, in the structures which fall in the PYP support; it is important to notice that these concepts are semantically similar to the perception. Hence the base distribution defines good support by excluding concepts that have a completely different meaning.

Tests were done using different types of concepts. In Table 4.2 each row represents a new concept to be allocated in the ontology and the corresponding ontological structures with the results classified by the mapping and the merging processes. Even if the emergent structures could be trivially evaluated, they were validated by domain experts.

4.3 Summary

This chapter shows the progress made in developing a knowledge management system for cognitive architectures developed using multi-agent systems.

Engineering knowledge management systems for complex software architectures is an activity that involves several areas of research. Knowledge engineering has always been involved in the study of knowledge representation systems for various research areas, depending on the demands and problems of the domain in which the knowledge management systems must work. It has been necessary to evaluate the application domain and its constraints to model and develop knowledge management systems.

In the solution proposed here, the main problem arises from the fact that agents co-

Concept	**Ontological Structure - Result**		
Kiwi	[Fruit] - 0.04250	[Fruit, Pear] - 0.0295	[Fruit, Apple] - 0.02318
Watermelon	[Fruit] - 0.04685	[Fruit, Apple] - 0.02563	[Fruit, Pear] - 0.01664
Voltage	[Energy] - 0.08136	[Device, Energy] - 0.02436	[] - nan
Linux	[Software] - 0.13556	[Object, Software] - 0.04744	[] - nan
Frog	[Animal] - 0.03827	[Animal, Fish] - 0.01140	[Animal, Cat] - 0.00913
Cat	[Cat] - 1.0	[] - nan	[] - nan
Sonar	[Sensor] - 0.04739	[Device, Sensor] - 0.03176	[Device, Memory] - 0.02247
Pencil	[Pencil] - 1.0	[] - nan	[] - nan

Table 4.2: Table collects results obtained during the experiments for testing the method. Domain experts validated obtained results. Each row contains the new concept and the related ontological structures computed using *Kewap*. The label and the correspondent result are represented for each ontological structure. The structures are ordered according to the results. The concepts are from different domains for demonstrating the robustness and the generality of the proposed approach. In the case the concept is already in the knowledge base, only a structure emerges that is the node in the ontology corresponding to such a concept, for which only the instance has to be created (for example, the *Cat* and *Pencil* concepts in the table). The cells with empty square brackets and dashes mean not detected structures.

operate and collaborate to perform functions of the multi-agent system. In this case, the knowledge must be a repository of information linked semantically and a system of knowledge control and updating, which keeps the agents belonging to the architecture always synchronized. Besides, the cognitive architecture is designed for realizing HRTI tasks. In this regard, the environment in which humans and robots collaborate could be dynamic and or partially unknown. Therefore a method for acquisition and update at runtime is necessary. The method must be based on design-time knowledge of the cognitive architecture and categorize new concepts perceived unsupervised quickly and precisely. To address these issues, the first section of the chapter proposes a centralized knowledge management architecture, implemented in the cognitive architecture of Chapter 3. The second section instead tells about the computational model to increase knowledge in a runtime robot. The model proposes a solution for developing a system able to increase its knowledge even under the assumption of limited resources and knowledge. The architecture shown in Chapter 3 implements these solutions for handling and organizing the knowledge.

Chapter 5

Anticipating Actions and Learning Plans

Chapter 3 shows a solution for realizing Human-Robot Teaming Interaction (HRTI). The theoretical model, discussed in Section 3.2, implements functionalities useful for establishing reliable interactions. As said before, the most important change lies in the deliberation process module of the cognitive architecture. In addition to classical modules for realizing learning, reasoning, and action selection, this module shows a novel block that lets an agent anticipate actions to be executed for pursuing an objective. This ability implements in some sense a kind on *imaginary* skill. The anticipation process returns information about the scenario's evolution regarding plans to schedule to be executed. This process involves the knowledge and the plan library that an agent owns for acting in an environment. To implement these abilities, a possible solution involves the creation of a simulating system [183]. The simulator was implemented using a graphics engine used to develop games, the Unreal Engine[1]. Besides this ability to anticipate situations [47], the module acts as an innovative interface that lets humans inspect the robot's mind through the interface. It is, in fact, possible to see the result of the decision-making process of the robot, making this process transparent, as well as navigating the knowledge of the robot, exploring corridors of the mind as in a video-game, so a human can see concepts in the robot's knowledge, instances, and their description. The simulator is also useful to test plans generated at runtime by the robot without an agent

[1]https://www.unrealengine.com/

programmer's supervision. Indeed, the robot works in a partially known and dynamic scenario. There may be the possibility that all its functionalities could not be completely sufficient for facing all challenges for the team. To let agents compose plans at runtime, an internal action for the agent was done. This internal action enriches, in terms of capabilities, the agent. The capability to compose plans at runtime is shown in this chapter. This approach is based on using a different type of logic, the non-axiomatic logic [202, 206]. It manages the plan library, handling all beliefs, plans, and goals of the agent by means of the Non-Axiomatic Reasoning System (NARS) [98][2].

5.1 Anticipating Actions: a Development Point of View

Humans and robots collaborate and cooperate in pursuing a shared objective need to rely on the other for carrying out an effective decision process and revising knowledge when necessary in a dynamic environment.

For doing that, a solution was realized through a cognitive architecture, discussed in Chapter 3. The cognitive architecture shows characteristics and functionalities thought to be used in a robotic platform that has to cooperate with a human. The requirements were analyzed by studying the human-human cooperation counterpart. The model, designed for implementing this type of interaction, sees the turning point in the deliberation process with respect to other architectures. More than usual modules for learning and selecting which intention is more reliable to be executed under a specific context, the cognitive architecture's deliberation process is based on the cognitive abilities to *imagine* the evolution of the scenario. This ability took inspiration from a previous approach described in Seidita et al. [184]. Thus, anticipating actions makes the difference in building autonomous robots to cooperate with a human in a dynamic and partially known context. The anticipation is an important module in the architecture in Figure 3.1.

This section aims to show how it was possible to develop the anticipation module in the proposed cognitive architecture.

[2]This work was done during the collaboration with the Temple University with Prof. Pei Wang and the colleague Dr. Patrick Hammer.

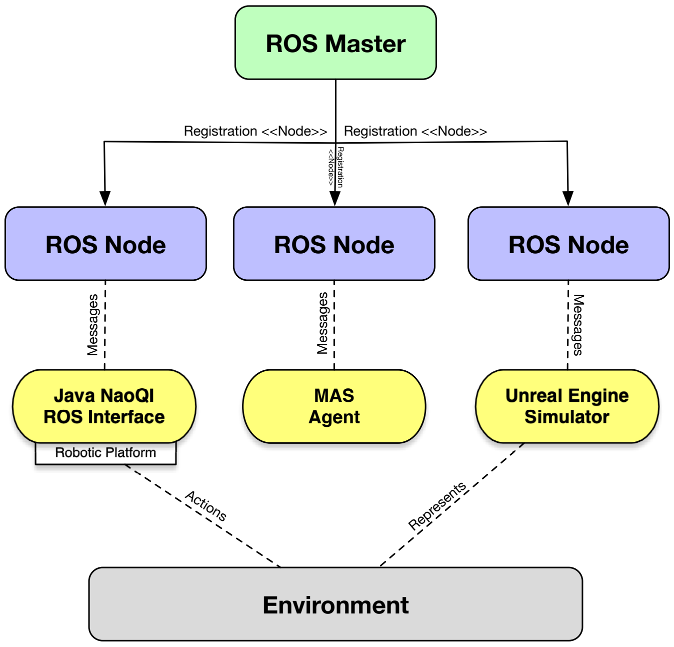


Figure 5.1: The ROS nodes configuration for letting communication among components.

5.1.1 The Simulator

The *current situation* represents the current state in which the agent is living and acting. It contains all information about the inner and the outer world that see the agent as an entity that acts upon it. The software module in charge of supporting the decision-making process in the sense of anticipating actions is the simulator.

Thus, the simulator realizes the *anticipation process* of the cognitive architecture. It was possible by linking the cognitive architecture with the simulator through the usage of the Robot Operating System (ROS) [164]. To recall, ROS is a *middleware* widely used in robotics. ROS is not a real operating system, but it lets programmers develop robotics applications employing hardware abstraction, device control, and common functionalities for implementing low-level and high-level capabilities, package management, and communication modules. The simulator was developed using Unreal Engine 4 [81] and the communication with other components of the system was done through a plugin for Unreal Engine, called ROSIntegration [140]. The cognitive architecture uses the jason_ros bridge[3] to integrate the Jason framework and ROS.

[3]https://github.com/jason-lang/jason_ros

Thus, the Figure 5.1 contains the diagram that shows the instantiated nodes in Robot Operating System (ROS) [164] for allowing communication among the simulator, the multi-agent system that contains all the agent for implementing the cognitive architecture, as shown in Figure 3.2 and the robotic platform for accessing on it through the ROS interface.

The simulator was built with modules developed in C++ and Blueprints. This last is a visual scripting system based on the concept of using an interface based on visual nodes for creating elements from within Unreal Engine Editor. The simulator handles the publishing and subscribing mechanism. Thus it exchanges information continuously through topics for representing and simulating activities into the virtual scenario.

The simulator, in the background, executes capabilities to realize the anticipation process. In this way, the robot tests itself in the current situation and tries to anticipate results.

Results of the simulation are returned to the multi-agent system, and so the system may be informed before the real execution if the current scenario reaches or not the fixed goal through the set of actions used in the virtual scenario.

To recap, the framework, summarized in Figure 5.1, was built considering:

1. one or more Java applications, named Talker, are responsible for communicating with the robot, using a set of libraries: ROSJava. It was developed by the ROS community and allowed to instantiate the ROS concept (nodes, services, and topics) inside the Java code. The Java applications use the NaoQI library, provided by Softbank Robotics, to interface with the Nao robot. NaoQI extrapolates information of interest[4] (such as robot temperature, robot joints position, battery level, and so on), and write them into ROS topics;

2. the ROSIntegration plugin, installed in the UE4 editor, creates a bridge between ROS and UE4, allowing UE4 to subscribe to the ROS topics created by the Java applications and recover the information from them;

3. on the UE4 side, one or more C++ classes, called Listeners are responsible for retrieving and elaborate robot data. UE4 functionalities were used to display such information in a friendly graphic manner.

[4]For the sake of completeness, the information could be acquired exploiting the ROS framework at all, for the system's compatibility and to facilitate the test, it was used the native library.

4. the multi-agent system was developed using Jason [27] and CArtAgO [171] as described in Chapter 3.

5.1.2 Generating the Anticipation: A Proof of Concept

Let consider a robot to pursue a simple objective: to reach a specific position in the room. The robot starts from a known initial position at the time t_0.

The mission is a navigation task for path-finding based on A* based algorithm [101]. The robot does not know the environment, and obstacles could spawn along the path established. The planner calculates the route to follow according to the robot's knowledge. The simulator results help the robot to re-plan when somethings wrong occurs.

The robot is designed for performing the mission and is equipped with essential knowledge for representing the working environment. As shown in Figure 5.2, the simulation interface allows the human to be aware of what the robot knows about the surrounding environment, at the beginning of the mission, and during its execution.

Figure 5.2 shows the system at work. The first row shows the robot deployed onto the real world, in which it is acting toward the objective. The second row shows the simulation interface in which the robot is acting in the simulated environment. Figure 5.3 shows another type of perspective about the robot's mind, the corridor navigable by a human to inspect the knowledge of the cognitive system, and so, of the robot. As can be seen, going from left to right:

1. at the beginning of the mission execution, the robot is aware of its position and the goal's position. An avatar appears in the simulation environment. The avatar represents a robot's mental extension, the image it has of itself in the environment. The avatar executes the designed path to reach the goal (a middle couple of figures). It represents the anticipation of the mission;

2. at the end of the simulation/anticipation the robot starts to execute the path;

3. during the execution of the task, the environment could suddenly change or may be different from the one designed in the initial ontology. Indeed, in this scenario, we put a second obstacle without inserting it into the ontology. When perceiving the new situation, the robot stops its navigation. The left ROS node (Figure 5.1) and the NaoQI interfaces communicate with the ROS master for handling the new

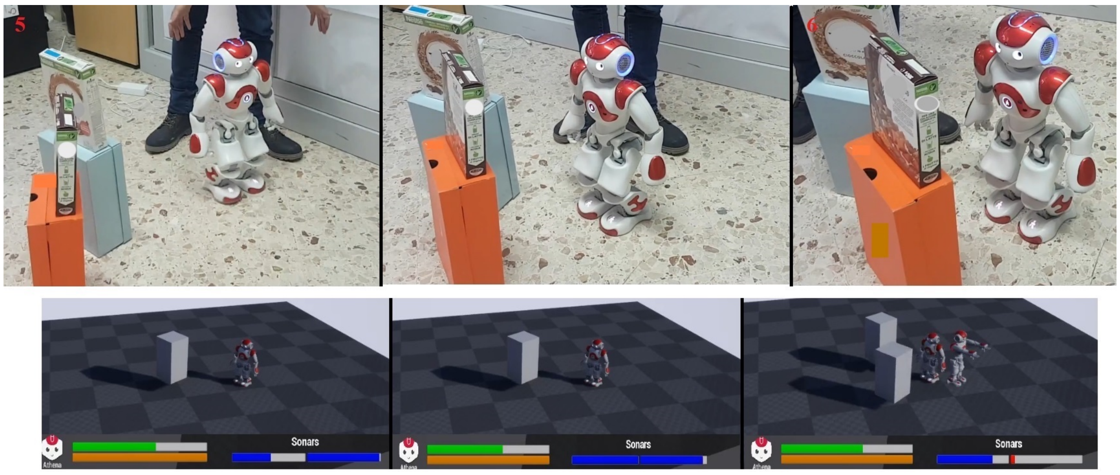


Figure 5.2: The robot's mission seen in the real and the simulated environment.

situation. The obstacle appears in the simulation environment through the right ROS node, and the robot recomputes the best path. Then, the new anticipation is showed by means of its avatar (right part of Figure 5.2);

4. the process is repeated until the robot reaches the goal or until there are no more available paths to follow.

The robot used for this proof is the NAO robot by Softbank Robotics. The choice of the robotic platform is not fundamental. The usage of ROS ensures the compatibility with a large set of platforms as a robotic framework. The environment was realized into the RoboticsLab's room, and generic objects are randomly located in the room.

Software usefulness is evident when the robot behaves differently from expectations, i.e., when the robot cannot reach the goal although there is an available path. The proof of concept was tested before in the virtual environment, spawning the robot's avatar to verify if the scheduled plan is reliable for the current context. Detected obstacles, the anticipation process informs the multi-agent system that the selected plan will fail during the execution phase. So, the reasoning system generates another plan, and the simulator will test it before acting, as done for the previous one. Only when the simulation gives back a valid result the agents move toward the goal, this anticipates the future situations letting the robot be able to see the results of its actions.

Finally, anticipation is achieved in the execution of operations before applying them in the real environment. The simulation process results are returned to the multi-agent

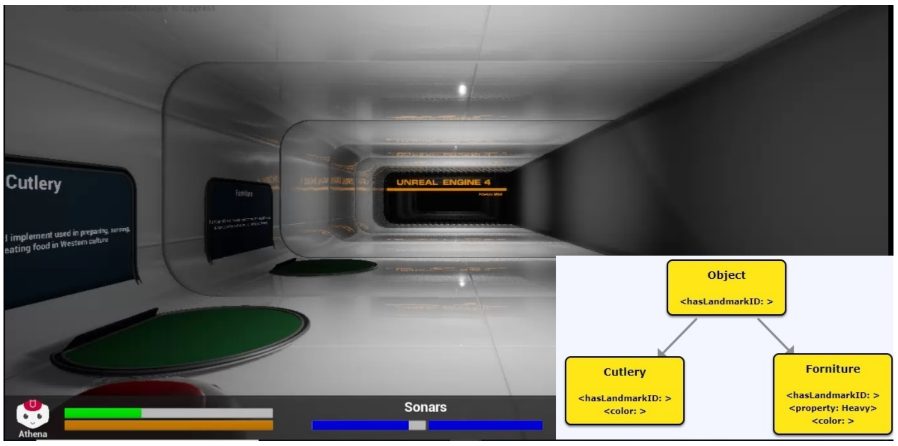


Figure 5.3: The simulator interface showing a view of the robot's mind and how it represents its knowledge.

system to make the agents think about the current situation and then guide the robot in its mission.

5.1.3 Making Transparent Robots' Knowledge and Reasoning

A practical aspect of the simulator is the possibility given to the human teammember to inspect the robot's work. So, this may be seen as an extra option for this simulator. In this case, the simulator is not used as a simulation framework for testing before launching the system in the real environment. However, it is a representation of reality in the robot's mind. The simulator lends itself to be a solution for implementing transparency and explanation in the robot's reasoning system. The simulator was endowed (Figure 5.3) with an extra virtual scenario, on which the human could navigate the robot's mind by going through the robot's mind corridors. The corridors contain all the knowledge owned and acquired. In conclusion, the simulator has revealed a good solution for implementing the anticipation process postulated in the theoretical model as discussed in Chapter 3 and is useful for enhancing the human-robot interaction through an interface able to show mechanisms behind the robot's reasoning and knowledge. Considering this last factor, it was possible to show results of the unsupervised incremental knowledge process, discussed in Chapter 4.

5.2 A Method for Learning Plans at Runtime

As many times mentioned in Chapter 1 and Chapter 3, in a scenario where humans and robots cooperate and collaborate to reach a common objective, one of the most challenging issues is to endow the robot with the ability to plan actions during the execution phase proactively.

Normally, the robot is programmed to pursue its objective by executing a plan composed of a set of actions. The robot acts based on its knowledge of the environment and of the actions to perform. Given a *static* environment, where everything is apriori known, the robot follows programmer prescriptions. In such a case, the robot is endowed with the ability to face and respond to all kinds of known inconveniences. Developers have all the means and information to face design time issues, so ideally, nothing unexpected will occur while the robot is working.

In the case of robots and humans cooperating in a partially, or entirely, unknown environment, a desirable correspondence between a goal and a plan cannot be assured. Above all, it cannot be decided at design time. Here, the need for a completely autonomous robot aware of its objective arises. The robot has to perceive elements from the environment and, consequently, select the right plan for pursuing that objective. If the robot has not been endowed with suitable plans, it should find new ones. This latter situation corresponds to human autonomy and self-adaptation abilities and is one of the most urgent challenges in human-robot interaction.

Composing plans at runtime requires intelligent methods that let the robot understand how to manipulate its knowledge and its abilities for creating new plans that are feasible and reliable to be applied by the robot in the context in which it operates. Besides, the uncertainty and the dynamism of the environment do not contribute positively to simplify the creation of this ability.

An efficient way to face the aforementioned problem is using the agent-oriented paradigm with the support of a method for composing a high-level plan. In this case, the BDI model is used for implementing these capabilities, employing the Jason [26, 27] framework. For implementing the capability to compose plans at runtime, the idea is to merge the Jason reasoning cycle with the Non-Axiomatic Reasoning System (NARS) [98, 207]. Both Jason and NARS can be utilized to handle the planning. Jason has a reasoning cycle implemented and coded inside an agent; it needs well specific and defined inputs from the outside to select a plan, and this is the main limitation of Jason for our objectives. NARS, instead, is a general-purpose reasoning engine that does not have this limitation. NARS often allows constructing plans even in the absence of specific knowledge [205, 208]. It is based on information gained from observed correlations, or structural similarities between existing plans, or both.

The two systems alone cannot completely solve the main problem, but the strengths of both systems can be combined. In the human-robot interaction domain, the interaction is at the same time a source of changes and a source of information useful for planning and for deciding on actions. So, the idea to merge both the systems, Jason and NARS, and to create a system where the core is composed of one (or more) Jason agent that, typically, performs its usual reasoning cycle may face the aforementioned challenges.

Merging Jason and NARS lets the agent, every time it cannot select a plan from the plan library, invoke aid by NARS. NARS generates a plan to be added to the agent's plan library. NARS mostly works as a portion of the brain of an intelligent system; it takes control only when all the other automated brain functions do not have success while listening to the beliefs in the background. NARS is an external reasoning system that helps the agent modify its behaviors defined at design time.

5.2.1 Composing High-Level Plans in BDI Agents

Jason and NARS alone are not able to handle runtime planning during human-robot interaction. This proposal consists of merging the two to create a multi-agent system where several Jason agents deliberate and act to let a robot reach the team's objective. Jason agents follow the usual reasoning cycle shown in Figure 2.11 and call NARS as often as they do not find executable plans in their library. This is possible thanks to a bridge developed for adapting Jason to NARS and vice-versa. The two systems work in different languages, so translating the main elements of communication is necessary. All the knowledge was considered useful for reasoning about the action to perform to pursue an objective: beliefs, goals, and plans (see Chapter 2 for details about them).

Algorithms and methods for composing high-level plans treated in [143, 142] were analyzed for designing, modeling, and implementing this system. The control cycle that regulates work among Jason and NARS is shown in Figure 5.4. In the next sections, the

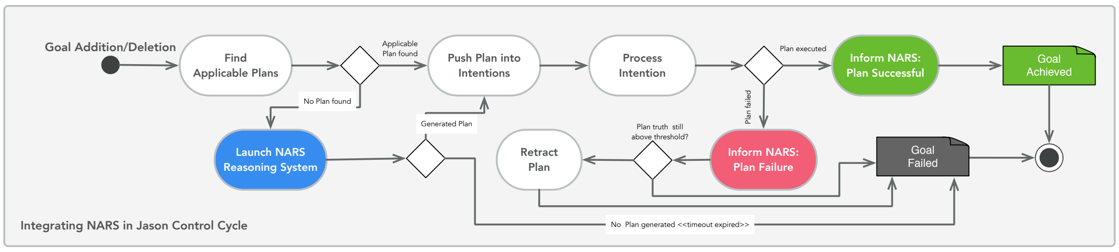

Goal Addition/Deletion
Find Applicable Plans
Applicable Plan found
Push Plan into Intentions
Process Intention
Plan executed
Inform NARS: Plan Successful
Goal Achieved
No Plan found
Generated Plan
Plan failed
Plan truth still above threshold?
Launch NARS Reasoning System
Retract Plan
Inform NARS: Plan Failure
Goal Failed
Integrating NARS in Jason Control Cycle
No Plan generated <<timeout expired>>

Figure 5.4: The revised AgentSpeak control cycle to support OpenNARS.

method that lets a robot (or an agent) to compose high-level plans at runtime merging Jason and NARS is illustrated as well as the algorithm and the development process.

Endowing Jason Agent with High-Level Plans Capability

Methods for letting an agent in Jason pursue its objective even when preconditions for selecting a proper plan miss is an interesting challenge to face. Meneguzzi et al. [142] collected a series of methods and approaches for doing that, analyzing and highlighting them to show potentiality and weakness. As said before, the Jason control cycle Figure 2.11 alone does not support what it was mentioned above. It was necessary to endow Jason agent with a capability act to composing high-level plans at runtime also under the assumption of insufficient knowledge and resources [205, 202, 207].

To extend the control cycle in Figure 2.11 and to identify the new one in Figure 5.4, the model is inspired from work done by Meneguzzi et al. illustrated in [143] as regards the development of planning algorithms for BDI agent systems.

Let us suppose that a Jason agent owns partial knowledge of the environment and of the goals it has to achieve, its plans, actions, and so on. All the activities performed by the Jason agent are represented in Figure 5.4.

The Jason agent, using only its reasoning cycle, may face a couple of situations. The agent may find an applicable plan in its library or not; in the first case, the process goes on as the Jason reasoning cycle prescribes. In the second case, the agent autonomously invokes the NARS reasoning system to support making a decision. A *recovery plan* is triggered by the agent launching an *internal action*. It informs NARS about the plan's failure, and when no alternative plan is available, it lets NARS effectively take control until it finds a new solution. In the best case, NARS generates a plan that is then added to the agent's plan library and is successfully executed (*Push Plan Intentions* and *Process*

Intention activities), letting the Jason system take over again. In the latter case, the agent informs NARS that the generated plan was successful, so NARS revises the inner hypothesis that corresponds to the plan, increasing its truth value. The *truth value* is used to summarize the evident support for a plan; that's how its successes and failures are tracked. Indeed, NARS has a memory system where goals, plans, related parameters, and contexts are stored. Each time a newly generated plan is considered successful, NARS has a higher chance to keep it in its memory [5] to serve similar future situations.

In more detail, the Jason reasoning cycle has control when a plan in the library matches the current beliefs and the current goal. Simultaneously, NARS listens for *inform requests* while receiving the agent's beliefs. The new plan pushed into the *plan library*, is scheduled following the Jason system scheduler during this period. So the reasoning cycle converts the plan into an *intention* and processes it. After the execution of the *intention*, NARS is informed about the success or the failure of the added plan; it takes note of the result of the action to enhance the success estimate of the plan-related hypothesis. Suppose the new plan fails and it has been already converted into an *intention*. In that case, NARS is informed about that, and the system checks the truth value (*expectation* following [207]) to decide whether the plan should be retracted.

It is worth to note that the external reasoning cycle, based on NARS, uses an inferential (non-axiomatic logic [202, 206]) approach to make the decision. Here, the only extra operation the Jason agent performs is to inform NARS about the results of the plan application; moreover, the agent autonomously requests help from NARS.

Looking at the control-cycle in Figure 5.4, it is worth to note that the control cycle is not completely changed and, in standard condition, the agent continues to work as a general Jason agent, with the addition that if it is needed, it can change its behavior including an external reasoner for composing high-level plans at runtime. In this sense, the system's complexity is not increased, and the capability was implemented exploiting Jason's feature for handling cases on which the context of a plan is not valid, and so, no plan may be executed. This capability is intrinsic to Jason, since it is supported directly by AgentSpeak using *recovery plans* [27]. The method was implemented extending the Jason internal action, and this means that this ability is free to use in each agent within the multi-agent system.

Figure 5.5 shows the structural view of the proposed reasoning cycle of Figure 5.4

[5]NARS uses a fixed-sized memory, so forgetting is inevitable once its memory is at full capacity

The upper part of the figure (the dotted square) represents the Jason framework and its main elements. The rest of the figure represents the added classes to the framework and the relations with the existing ones. Allowing agents to create a new plan at runtime does not require great changes in the code's structure developed for the multi-agent application. The solution adopted for this method is simple to use. Programmers have to apply a few changes in their program, introducing only a *recovery plan*, which invokes the new capability to generate new plans at runtime. The extension is merely focused on the framework, whereby each agent defined in the multi-agent system could call the new internal action to invoke NARS.

The *NarsEngine* class shown in the UML diagram in Figure 5.5, involves different methods to let the system work as intended. The handling phase for plans, and the methods used within this phase, to translate the knowledge of the agent, will be discussed in sections 5.2.1 and 5.2.1.

Handling Plans in Jason using NARS

As mentioned before, the agent yields control to the external reasoning system using two processes that let NARS reason on beliefs and plans already present in the agent. The appropriate mechanism that lets the agent yield control to NARS is implemented using the Internal Action interface (see the UML diagram in Figure 5.5).

Algorithm 5: A pseudo-implementation of the invokeNARS capability.

```
$PL$ ← Agent.getPL();
$BB$ ← Agent.getBB();
while Agent.isAlive() do
    P ← PL.getPlan();
    B ← BB.getBeliefs(P);
    if (isIntentionValid(P, B) then
        executePlan(P, B);
    else
        invokeNARS(PL, BB);
end
```

The agent is endowed with a `PlanLibrary` attribute, a `BeliefBase` attribute, a `TransitionSystem` attribute and other attributes relevant for the agent.

Each Jason agent owns all the attributes to perform the reasoning cycle (in Figure 5.5 some of them are shown). The diamond in the figure represents a Map object. This object contains a String-key and an InternalAction-value. To extend the control cycle, as mentioned in Section 5.2.1, we realized to add a specific internal action that enables the agent to use the external reasoning cycle. The new *internal action class*, called *invokeNARS*, extends a class that involves NARS, in the figure it is called *NarsEngine*, and it implements the `internal action` interface.

The pseudo-code that implements the reasoning behind the usage of the invoke-NARS capability is represented in Algorithm 5. For each cycle, the agent checks if the intention associated with the plan is valid. In case it is valid, the agent executes the plan; otherwise, the agent launches the internal action that invokes NARS, passing the current list of plans and beliefs as an argument. The *NarsEngine* class is liable to perform the operation needed to develop the control cycle proposed in Figure 5.4.

Once, the agent executes the *invokeNARS* internal action the *execute function* is executed. This implements the handling process that translates the list of beliefs and the list of plans to the Narsese language (subsection 5.2.1). If no other plan applies, NARS takes control of the situation, with all knowledge owned by the agent.

Adding a new plan means that NARS succeeded in combining existing plans in the agent's Plan Library to create a new one. In general, if the plan exists, it shows a context that matches with the situation, so it may be used to pursue the goal.

NARS has to assemble plans based on new perceptions and existing plans. For instance, NARS can probe the environment to fill in existing knowledge gaps by learning patterns from the sensory-motor experience [96].

Once NARS gets its reasoning system started with the information translated from Jason to Narsese, it uses inference, such as the *deduction rule* (for the inference details[6] see [207]) for generating the new plan.

The non-axiomatic reasoning system uses events to start the reasoning process. So, when beliefs are transferred from Jason to NARS, observed ones and others are sent as events. NARS can also send predicted events as beliefs to Jason, but these are marked as inferred and come with additional information, encoded as p(x,t), where the additional information t represents the occurrence time of the prediction. These predicted events do not go back to NARS again but can be useful to the Jason agent.

[6]example of a *deduction rule*: $\{a \nRightarrow b, b \nRightarrow c\} \vdash a \nRightarrow c$ and $\{a \nRightarrow b, (b, c) \nRightarrow d\} \vdash (a, c) \nRightarrow d$.

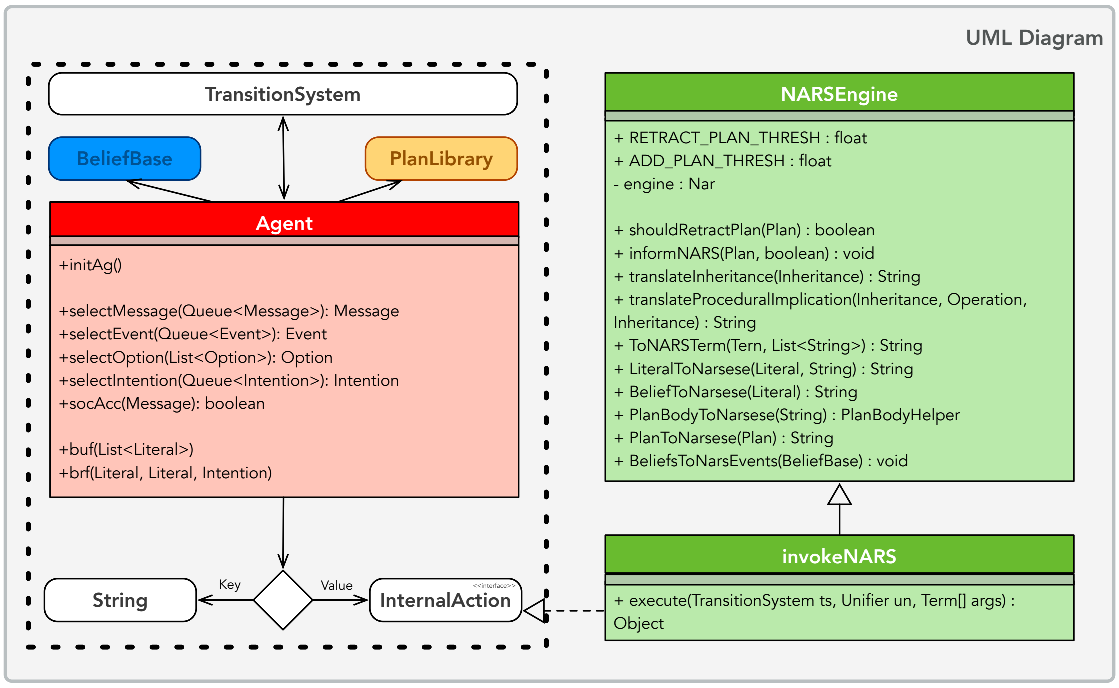

UML Diagram
TransitionSystem
BeliefBase
PlanLibrary
Agent
+initAg()
+selectMessage(Queue<Message>): Message
+selectEvent(Queue<Event>): Event
+selectOption(List<Option>): Option
+selectIntention(Queue<Intention>): Intention
+socAcc(Message): boolean
+buf(List<Literal>)
+brf(Literal, Literal, Intention)
String
Key
Value
<<interface>>
InternalAction
NARSEngine
+ RETRACT_PLAN_THRESH : float
+ ADD_PLAN_THRESH : float
- engine : Nar
+ shouldRetractPlan(Plan) : boolean
+ informNARS(Plan, boolean) : void
+ translateInheritance(Inheritance) : String
+ translateProceduralImplication(Inheritance, Operation, Inheritance) : String
+ ToNARSTerm(Tern, List<String>) : String
+ LiteralToNarsese(Literal, String) : String
+ BeliefToNarsese(Literal) : String
+ PlanBodyToNarsese(String) : PlanBodyHelper
+ PlanToNarsese(Plan) : String
+ BeliefsToNarsEvents(BeliefBase) : void
invokeNARS
+ execute(TransitionSystem ts, Unifier un, Term[] args) : Object

Figure 5.5: An UML Diagram for showing classes and methods used to merge Jason and NARS.

The model is implemented using the observer pattern inside the *NarsEngine* to avoid potential errors in the reasoning system. Each time a plan fails or succeeds, Jason informs NARS to allow for the NAL-based inferences to revise.

Translating beliefs and plans for NARS

The described functionality requires translating the knowledge expressed as plans and beliefs from AgentSpeak into Narsese, and back. The translation was done employing the non-axiomatic logic grammar described in [206]. Being the AgentSpeak language based on first-order logic and on a term logic, the translation process was easy to implement since even Narsese is a term logic. Besides, the non-axiomatic logic is endowed with all logical constructions to map AgentSpeak in Narsese and vice versa. As a first illustration for the conversion from BDI AgentSpeak representations to Narsese, the following example can be considered:

```

  // This line is a generic belief defined in AgentSpeak.
```

```
 on(switch0).
 // The same line but this time translated in NARS.
 <{switch0} --> [on]>. :|:

 // This line is a generic goal.
 !on(switch0) // The desires to put on the switch in AgentSpeak.
 <{switch0} --> [on]>! :|: // // The desires to put on the switch
in Narsese.


 /* This line is a generic plan.
 Plan: invoking the activate operation after
 being at the position of a switch turns it on. */
 +!on(switch0) : at(X) & X=switch0 <- .activate(). // writtent in
AgentSpeak.
 <(&/,<{switch0} --> [at]>,+5,(^activate,{SELF}),+5) =/> <{switch0
} --> [on]>>. // written in NARS.
```

Listing 5.1: Example of AgentSpeak/NARS translation.

Listing 5.1 represents the translation from AgentSpeak to Narsese. The listing contains three blocks, the first for representing a belief, the second block represents a goal, and the third block a plan. The codes are in the order, first AgentSpeak and later in Nars. This excerpt of code is simple lines for letting a robot activate a switch when it is in the correct position. The *NarsEngine* class is in charge to translate the goals, the plan library and the beliefs from AgentSpeak to NARS and vice versa.

5.2.2 Proof of Concepts

JaCaMo[7] v0.0.7b [25] and OpenNARS[8] 3.0.2 [98] are the frameworks used to develop the method for composing high-level plans described above.

OpenNARS [98] is the open-source version of NARS and JaCaMo [25] is a framework for multi-agent systems programming that combines Jason with other two important technologies, CArtAgO [171] and Moise [99].

Two scenarios are treated in this section.

[7]http://jacamo.sourceforge.net/
[8]https://github.com/opennars/opennars

The first proof shows the ability to retract a plan via a statistical approach. Generally, in the regular control cycle, developers have to write a recovery plan to handle unknown situations, and the operation to provide success/failure feedback to NARS is appended. The method takes into account failing plans and removes them if the truth expectation (c * (f - $\frac{1}{2}$) + $\frac{1}{2}$) (where c is confidence and f frequency, see [207]) of the used plan drops below a threshold. The feedback is given to the NARS engine via the *inform request* sent by Jason.

The second case study shows the ability to generate a new plan using the external reasoning system. As described in section 5.2.1, the process starts by translating the actual plans and beliefs defined in the PlanLibrary and the BeliefBase. Once NARS has received the translated knowledge, it starts the reasoning process to produce a new solution for the present request. This solution is converted again into Jason's beliefs and Jason's plan to be executed in the next agent reasoning cycle. So, in the beginning, the agent tries to find a solution using plans written at design time by developers, and if none is applicable, NARS (running in the background) is expected to find a plan by reasoning and motor babbling. Once a plan is found, the plan with the associated beliefs are pushed into the PlanLibrary and BeliefBase (because the plan exceeded a truth expectation threshold), and Jason takes over again.

Before proceeding with the use cases, another definition is necessary to understand how the proposed method works. The definition, extracted from page 3 and section 2.3 of [208], concerns the trust expectation (or truth value):

> The *truth value* of a statement measures its extent of evidential support, rather than that of agreement with a corresponding fact [208].

The examples below show the functionalities introduced by the new control cycle shown in Figure 5.4.

Plan retraction example In this case, a simple plan is added to the plan library that is set up in such a way that it most often does not succeed. After each failed attempt, the expectation value should sink due to the revision rule of OpenNARS. After it falls below a threshold, it is removed from the plan library. Note that successful executions increase the expectation. This example is fundamentally different from a plan that just detracts another plan cause it fails, in fact:

```
//Beliefs
position(kitchen).
//Desires:
!serve_user.
//Plans:
@critical
+!serve_user : position(kitchen)
             <- .print("I_am_in_the_kitchen").
-!serve_user <- .remove_plan(critical).
```

Listing 5.2: An AgentSpeak approach for performing an operation.

The listing 5.2 shows a classical method for retracting a plan without any support. The variant, shown in listing 5.3, relies on NARS's revision functionality to decide the plan retraction looks as follows:

```
@critical
+!serve_user : position(kitchen)
             <- .print("I_am_in_the_kitchen");
                jia.invokeNARS(critical, true).
-!serve_user <- jia.invokeNARS(critical, false).
```

Listing 5.3: The proposed approach for performing an operation with the support of an external reasoning in the case of missing plan for handling the goal.

Note the *true* flag on plan success and the *false* flag on failure.The experiment led to the retraction of the critical plan after multiple failures, making the system effectively reject plans that do not work anymore. This is a toy-example to show the difference between retracting a plan cause it does not respect precondition just one time and the version that uses the proposed method for retracting a plan only if its truth expectation value goes below a threshold to mean that this plan is irrelevant for the robot.

Plan addition example In cases where the system's goal cannot be fulfilled by an existing plan given the current beliefs, NARS is liable to take action. It is provided with a set of operations it is allowed to perform. As a consequence of the operations, new procedure knowledge will be acquired, as described in [96]. So a plan generated as procedure knowledge in NARS is automatically translated to AgentSpeak and added to the plan library when above a certain truth expectation threshold[9].

```
/* Beliefs */
goal_serve_user. // A helping belief to inform NARS about the plan
   that requires invokeNARS supervision.
current_position(bedroom).
target_position(kitchen).

/* Initial goals */
!start.

/* The plans */
+!start <- !serve_user; .wait(500); !start.
@critical +!serve_user:
    current_position(kitchen) <- .print("I_am_in_the_kitchen"); ...;
   jia.invokeNARS(critical, true).

-!serve_user <- jia.invokeNARS(critical, false).
```

Listing 5.4: A second example to show the plan addition.

Listing 5.4 shows an excerpt of the code used for testing the plan addition capability. The scenario represents a case in which the robot has to reach the kitchen for serving a user, and it misses plans. In listings 5.4, beliefs identify the current position of the robot and the target position, the goal *start*. The first plan `+!start` is the starting plan, and it is in charge to launch the reasoning system. The next plan `+!serve_user` is given to accomplish the goal, and it contains actions to execute if the context of the plan is verified. As you note, the plan's context is false cause the current position is not the kitchen. Cause of that, the *recovery plan* `-!serve_user` is triggered. The method described above is started, and all the knowledge of the agent, goals, and the current full plan library is translated and transported to NARS. NARS produces a plan, and this last is added to the library to be executed in the next iteration of the `+!start` plan. Besides, as you note, other data are given as a parameter to NARS, the plan's label that needs the support identified by the character '@' and followed by a label, and a true or false value as described in the case above, for informing failing or success of a plan.

Finally, the case study validated the method, even for adding a runtime plan, since a

[9]Note that plan addition can also happen during normal operation while Jason is in control, as NARS is always running in the background and reasons about newly incoming beliefs of the agent.

plan was generated and added to the plan library. This plan was combined with all the beliefs (that contains all knowledge acquired by the agent, including the inner state of the robot), plans, and goals of the agent.

5.3 Summary

This chapter focuses on developing two modules for cognitive architecture: a module for letting the system simulate its decisions before selecting for applying it the real world through a simulator, and an algorithm for letting agent compose high-level plan at runtime based on a non-axiomatic reasoning system.

While the first module enables the cognitive architecture to anticipate the result of an action, the second is in charge of building new plans for providing a solution that could satisfy a goal. The system uses the BDI paradigm [168], and the architecture developed employing the Jason framework [27].

Another result of the work discussed in this chapter is a concrete enhancement for the multi-agent programming paradigm, it is now endowed with the ability to composing high-level plans at runtime, necessary for handling unforeseen and unpredictable situations.

Even the instrument for anticipating and testing the results of these plans, shown in Section 5.1, before the execution in the real environment may be useful for solving conflicts and making robot's decision-making processes transparent. The simulator grants two factors in decision making, anticipating actions, and so results, and testing these actions before applying them in the real world. Modules are complementary in this sense because *composed runtime plans* could be tested in this virtual environment to see if the *generated plan* fits with the current situation. Composing plans at runtime is not allowed by the default implementation of a multi-agent system. Several works [142] proposed and collected solutions and approaches, and in [143] a tentative solution is based on STRIPS [70]. The main difference between approaches in the literature and the one in this dissertation lies in the ability to produce novel plans even under the assumption of insufficient knowledge and resource [205]. This ability was possible thanks to an artificial general intelligence system called NARS [98]. NARS' efficiency to learn procedure knowledge was proven using the mechanisms described in [96]. NARS provides a robust reasoning cycle when robots have to produce standalone actions and when it is crucial to find alternative solutions to the ones determined at design time.

The novelty introduced involves the system's ability to be autonomous in the sense that the system can find a plan (not known apriori) based on its knowledge.

The ability to modify itself from within is one of the most critical challenges for autonomous systems. However, NARS's ability to hold generated plans in the memory is dependent on the size of the memory bag. This fact affects the system performance in terms of wasting knowledge and the ability to infer new plans.

To conclude, the first module integrates cognitive skills for letting agents imagine results. The second module produces novel plans at runtime autonomously increasing the level of self-adaptivity of a single agent. Both are in charge of making the architecture efficient for human-robot teaming interaction since robots with these abilities may interact with humans even in partially and dynamic conditions.

Altogether, the proposed method can highly increase systems' autonomy, letting them better operate in scenarios where unexpected circumstances let existing plans fail or where new plans to deal with novel situations are necessary.

Chapter 6

Human-Robot Teaming Interaction in a Real Scenario

Monitoring patients through robotics telehealth systems is an interesting testbed cause patients' conditions, and their environment, are dynamic and unknown variables. Moreover, in the case of the robot that works as a caregiver, it should be able to interact with a physician, even remotely.

This chapter shows how to build an intelligent robotic platform using the cognitive architecture proposed in Chapter 3, thus a validation, based on the usage of this architecture to be used in a complex domain is given. The robot is in charge to serve a patient in its home, or hospice, and to interact with the physician for supporting him by remote [125]. The interaction between the robot and the physician was implemented through a desktop application on which the physician has the access to the patient's data. In this case, the interaction with the physician is not direct but it comes out through a series of messages exchanged during its execution, in any case, the robot has to interact with the patient in a smart way for supporting the physician in his tasks. The aim of the human-robot teaming interaction, in this case, is supporting physicians in changing therapies at runtime in a dynamic environment. Besides, to be sure that the architecture used was appropriate to the requirements defined by the problem, then Software Architecture Analysis Method (SAAM) [114] was used for validation.

6.1 Agents and Robots for Collaborating with Physicians in Healthcare Scenarios

Today there are many clinical scenarios in which a physician cannot rely solely on himself, his knowledge or experience, or his presence, to solve a patient's problems.

Current scenarios are made more complicated by the increase in the average life expectancy of citizens, especially in Western countries, which leads to an ever-increasing demand for healthcare systems. It is also remarkable today that societies are committed to ensuring access to care and well-being to all citizens [218]. There are also cases in which patients live in poor or not easily accessible places or in general cases in which the physician must provide, or receive if he is not on-site, fast and reliable diagnoses to be able to establish a therapy or otherwise solve a problem. This last is the case, for instance, of the emergency due to the COVID-19 pandemic. The problem often faced by emergency room physicians was not having the means for early identification of infected cases. This fact caused a lot of people infected and then dead also among doctors and nurses. Another case is the lack of professionals or, increasingly challenging, the presence of changing contexts. For instance, cases in which patients with the same disease but placed in different family or social contexts have different characteristics and needs. Probably in these cases, a unique protocol cannot be applied, but doctors have to be able to decide on a case by case basis.

Supporting physicians and patients in *complex* clinical contexts requires intelligent systems endowed with the ability to solve problems during interactions. During the execution, the system interacts with users and the environment that often change continuously as a result of the interaction.

This scenario fits perfectly into the context of this thesis, in fact, these challenges may be solved by multi-agent systems able to self-adapt to changing situations and deliberate also in the total or partial absence of input data from physicians or patients.

Moreover, aspects cannot be faced and solved at the design time. Developers cannot identify and implement all the possible situations where a high level of autonomy is required. At best, they can identify several conditional statements and allow for a set of possible alternatives in the system behavior.

The more dynamism or uncertainty in clinical contexts exist, the more physicians may need to be supported by an intelligent system. In this situation, developers have to implement mechanisms that let the system autonomously monitor patients, retrieve important information, reason, and suggest actions to physicians if necessary. All this may be done employing robots and agents working on the basis of the cognitive architecture (Chapter 3).

6.2 A Multi-Agent System for Healthcare

This section shows the pilot scenario, designed to model and project the healthcare application with the cognitive system.

Pilot Scenario. Alice is a nice elder woman with chronic bronchitis and she has to stay at home. She is assisted by the MyRob robot. MyRob is in charge of monitoring Alice and her environment to collect her health-data. Dr. Haus prescribes some medicines. MyRob daily collects and sends data to let Dr. Haus check Alice's health status or revise prescribed therapies. The data is stored in a data repository where Dr. Haus, through an application, analyzes them and decides new therapies or confirms the previous one. If Dr. Haus, on the basis of the analyzed data, finds the therapy good nothing changes and MyRob continues its tasks without interruptions otherwise MyRob may send feedback or recommendations to Dr. Haus. So far, social robots used as a companion or virtual caregiver [201, 137] satisfactorily face these kinds of duties.

> *What about if prescribed therapies are not adequate, because something unforeseen occurred in Alice's status, and Dr. Haus needs to revise them collaborating remotely with MyRob?*

MyRob checks for alternative actions to apply to the context and asks Dr. Haus's authorization for proceeding, otherwise it contacts him to inform about the situation. In the latter case, Dr. Haus decides to change the therapy and through his terminal, he proceeds to send the new therapy to MyRob. So that, MyRob is allowed to change dynamically the therapies. For instance, let us suppose that MyRob has been charged to open the window each time the quality of air lowers below a threshold and to administer timely previous medicines to Alice. MyRob is going to open the window but perceives an increase in Alice's body temperature. MyRob recognizes the right conditions for opening the window now lack. MyRob contacts Dr. Haus, informs him about the new

Alice status, and requires his consent for not opening the windows and administering paracetamol. If MyRob does not know this alternative action, it alerts Dr. Haus for receiving commands. Dr. Haus may suggest administering paracetamol.

6.2.1 The System Overview: Architectural Aspects

The objective is developing a robot able to change its behaviors at runtime according to the physician's prescriptions and the patient's status.

The robot is developed and programmed and then it behaves based on the theoretical architecture in Figure 6.1. It resembles the architecture in Figure 3.2 (the pivotal point of this thesis) that has been redrawn to better fit the healthcare problem domain. The healthcare problem domain requires the following modules:

- **Environment Management** – contains all elements for interfacing physicians, for monitoring the environment in which patients live, and for acquiring data;
- **Knowledge Management** – is the module for storing and managing data from the environment and patient;
- **Reasoning** – is the module devoted to compute data stored into the knowledge module and to produce a series of actions to accomplish a task;
- **Acting** – given a set of actions, it extracts the proper action for a specific situation.

The Knowledge Management module depends on the Environment Management module, indeed only data resulting from monitoring form the knowledge of the system. Data stored in the Knowledge Management module are used by the system for the reasoning process and, at the same time, all the results of the reasoning process update knowledge. The Reasoning module produces a set of actions, useful to reach one or more system goals.

This set of actions is passed as input to the Acting module to perform operations in the environment, producing a change in the state of the environment. As said, the idea is to use the BDI agents paradigm for providing intelligence support to the physician work, also allowing him to manage the robot remotely. Beliefs, rules, desires, and intentions for each agent are defined at design time by developers and users (physicians

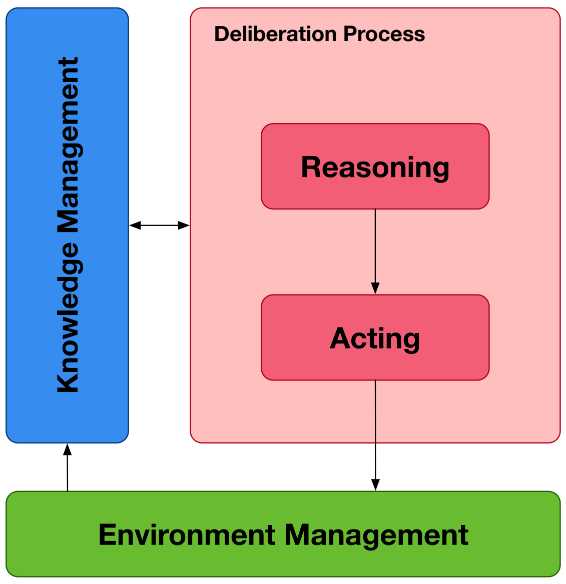


Figure 6.1: The architecture for implementing observing, orienting, deciding and acting cycle. Taken from [125].

and patients in the clinical domain). Robot's activities are easy to code thanks to the BDI paradigm underlying the cognitive model.

The next section discusses how it is possible to build a healthcare application for HRTI employing the cognitive architecture discussed in this dissertation.

6.2.2 Modeling a Healthcare Application for Human-Robot Teaming Interaction

The scenario involves two environments, the patient's residence, and the physician's office.

The two environments are connected through a multi-agent system that handles the real world, the patient, the doctor, and their interaction and the interaction with the environment. So, each environment owns one or more agents or better one or more agents are deployed in each environment.

The multi-agent platform serves as a bridge between the patient and the physician. It is worth to note that the middle layer of this representation constitutes the intelligent part, or better the cognitive architecture, to be added to a simple teleoperated system.

The physician uses his terminal to access the patient's environment and retrieves

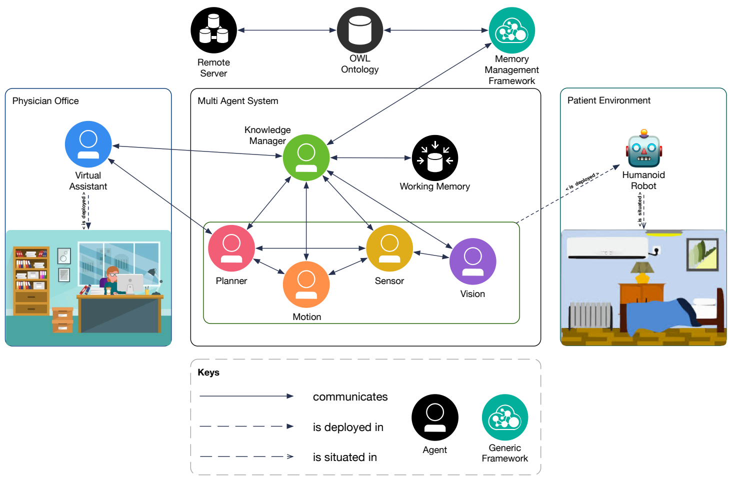


Figure 6.2: The multi-agent system for a robot in a healthcare scenario. The system shows circles for software modules such as agents and other used frameworks and three kinds of relations (arrows). The continuous arrow indicates communication between modules, dashed arrows are activities that identify on which nodes agents and robots work. The relation: (*i*) `<is deployed>` is used for software parts; (*ii*) `<is situated>` for cyber-physical system.Taken from [125].

data about the patient's status and the surrounding context. The interaction between the physician and the robot is handled through a software interface, instead, the patient communicates directly with the robot and vice-versa.

The cognitive model, followed by agents in the architecture figured out in Figure 6.1, starts its reasoning cycle from the Environment Management module. It is responsible for acquiring information from the environment. Collected data are organized and stored to be saved in the remote server. The Knowledge Management module organizes data to be synced with the system and updates them with the last perceived one for the diagnostic purpose by physicians. The same data are handled by the agents in the deliberation process module. This module is composed of two sub-systems, the first for reasoning and the second for acting.

Multi-agent systems are distributed systems over the network. Each agent could be relocated to other computers that host a node or a set of nodes of the system infrastructure. The multi-agent system can be distributed over remote sites connected via the internet. The system uses internet connectivity to share information, data, and alerts between the physician's office and the patient's environment.

The physician's terminal is connected to the multi-agent system through the *Virtual Assistant* agent. It implements all necessary functionalities to be into agents' network and it receives data from the patient's environment via the internet.

Data are stored using an OWL ontology, they are accessible from the physician by a terminal for diagnostic purpose. A memory manager holder organizes acquired perceptions and collected information by the system in an OWL ontology [11]. The knowledge is handled as described in [46] and in Chapter 4 and the agent which was delegated to manage knowledge is deployed into the system with the goal to keep all data synced and updated. The knowledge of the system is organized to be always synced, updated, and shared on the entire system and to avoid possible conflicts given the nature of beliefs. Indeed, beliefs may change during the interaction with the environment or with the patient so it is necessary to guarantee the right knowledge synchronization among all the modules.

The *Virtual Assistant* agent is deployed into the physician's personal computer as a computer application. It works as a remote controller application and it endows the physician for accessing the patient's environment.

The *Virtual Assistant* agent assists physicians in remotely collaborating with the robot located in the patient's environment. Admitted operations let physicians manage the robot, such as supervising the robot's behaviors, stopping or restarting it if necessary.

Nowadays, there exist several teleoperated robots that let the physician operate into the patient's environment, no one of them works in autonomy during the interaction with the patient. Autonomy, in this case, means to be able to recognize or retrieve the best action to perform in a particular situation, propose the action to the physician and wait for his command. In this sense, the robot is autonomous and it can suggest some actions to the physician. Indeed, in this cognitive system physicians can teleoperate in the sense of changing previous therapies or adding new ones to enhance the quality of life of the patient. The robotic caregiver should not be seen as a professional robotic avatar but as a valid instrument useful for monitoring and assisting patients in the place of clinicians.

Virtual Assistant agent endows physician for checking, revising, updating a therapy or a set of therapies initially prescribed and provided to the robot for taking care of the health status of the patient. Therapies are plans in the sense of the agent's plan that the *Planner* agent selects when the context is verified.

The *Planner* agent is responsible to realize the intelligence of the system. It communicates with the *Knowledge Manager* agent to obtain data synced, updated, and stored into the ontology for selecting the best plan that fits the current situation. To satisfy the system goal, the *Planner* agent tries to find the right solution using the context of a logical formula; once the context has verified a combination of actions and other plans let the system move toward the goal.

Logical formulas (as illustrated in Chapter 2), in multi-agent systems, are also called plans, and each plan is generally pushed into a repository of plans called Plan Library. Plans use perceptions and information acquired through the *Vision* and *Sensor* agents and are put in action through the *Motion* agent.

Some instances of *Vision* and *Sensor* agents are deployed into the robot, other instances that perform heavy computational processes are distributed on the same node of the *Knowledge Manager* agent or in other nodes if necessary, thus giving a high level of scalability to the system.

Table 6.1 summarizes the description of the role of the agents that are into the system.

6.2.3 How the Robot is Employed in the Clinical Scenario

The robot uses algorithms for implementing obstacle avoidance and motion. The agent may navigate in unknown indoor places using an adapted version of a state-of-the-art algorithm for robot SLAM (Simultaneous Localization And Mapping) [60].

The *Motion* agent is strictly connected to the *Vision* agent. While the former deals with the motion of the robot, employing an algorithm for navigating an environment [60], the latter deals with computer vision tasks. The *Vision* agent, continually, senses the environment using an RGB camera to discover and to recognize objects located in the environment. The object recognition module works using the YOLO deep neural network[1] [170]. YOLO is a state-of-the-art system for implementing real-time object detection and recognition. Recognizing objects is useful for motion's tasks, such as obstacle avoiding or to enhance the localization algorithm using some objects as landmarks, fixed-points, or landmarks (NAO-mark and QR-code) [69]. The *Vision* agent is also able to detect the status of the patient, such as understanding when the patient is

[1]YOLO v3 website: https://pjreddie.com/darknet/yolo

Agent	Role
Knowledge Manager	It is a part of the multi-agent system used for managing the system's knowledge. It uses a double kind of knowledge model, an ontology for storing information and gathering them as concepts and relations, and a knowledge base where beliefs are stored for acting in the working domain.
Planner	It is a part of the multi-agent system used for planning operations. This agent is delegated for selecting which plan is more adapt to pursuing the goal. Generally, the system goal is defined at design time but it can be handled at runtime. This agent listens for information gathered and it deliberates the set of actions and plans that each agent has to execute using the agent's communication module. This agent communicates directly with the virtual assistant agent. This last is responsible for upgrading the planner's plan library to add plans and enabling it for revising plans at runtime; this means that the multi-agent systems can act dynamically.
Vision	It is a part of the multi-agent system used for vision task operations.
Sensor	It is a part of the multi-agent system used for collecting data from sensors. These data contribute to the decision-making phase, to select the plan that best fit with the current situation.
Motion	It is a part of the multi-agent system used for handling robot motion. This agent implements modules for letting robot move and executing into the environment.
Virtual Assistant	It is a part of the multi-agent system used for letting physician be part of the system. This agent operates as a dashboard console on which the physicians retrieve data monitored by the robot and collected from the system. The agent lets physicians communicate new therapies to the robot that will be translated into plans and added into the plan library of the planner agent. Therapy will be executed during the next agent's reasoning cycle if pre-conditions signed by the physicians will happen. If this does not happen, an alert will be sent to the dashboard console.

Table 6.1: The table summarizes the role of the agents into the proposed multi-agent system, figured out in Figure 6.2. Taken from [125].

sitting or lying down or the patient is standing and recognize him though face detection and recognition [200, 2]. *Vision* agent takes also care of other tasks; a deep description of this module is out of the scope of the paper.

Other agents are deployed into the multi-agent platform to sense the environment and to enhance the quality of perceptions. An agent, delegated for handling information acquired by sensors, continually senses the environment with the aim to keep updated telemetries about the surrounding context. Data acquired are stored and shared between all agents.

Moreover, several vision tasks and heavy computational processes used by the robot are distributed over other nodes in the architecture for balancing the robot's workload.

Beliefs shared into the platform by agents are used to select the best plan that fits with the goal of the system, accordingly with the reasoning cycle. Methods are implemented as internal actions or operations for the agent. Perceptions are acquired through sensors and cameras. Sensors acquire information from the surrounding environment and each perception is translated into the corresponding belief and added to a set of beliefs handled by the knowledge manager as described.

The *Planner* agent communicates with the *Knowledge Manager* and *Virtual Assistant* agents for teleoperating activities and with the *Motion* agent to move into the environment for performing operations. The *Planner* agent's *plan library* can be manipulated and it is partially revised for changing behaviors at runtime. Ideally, this agent is divided into two parts: (*i*) plans for robot acting, (*ii*) plans for administering therapies. The former is configured for permitting to operate and this section of plans are written at design time by *agent programmer*, the latter is the section of plans that the system activates to administer therapies to the patient to serve the physician and taking care of the patient.

In addition, the former is also in charge to handle interactions among patients and physicians. The cognitive model uses a series of plans for handling the interaction. In the next section, we describe how to revise therapies remotely starting from diagnosis and we introduce the mapping process between therapies and agents' plans.

6.2.4 Handling Diagnosis and Therapy using Plans

Diagnosis represents the status of the patient in a particular moment of his life and the triggering condition for administering a therapy. Therapies or medical treatments are prescriptions aiming to heal a person with health problems. Each therapy contains a list of indications and contraindications for the patient and it can be effective or not. There are several types of therapies, in this approach, it was considered only the following type:

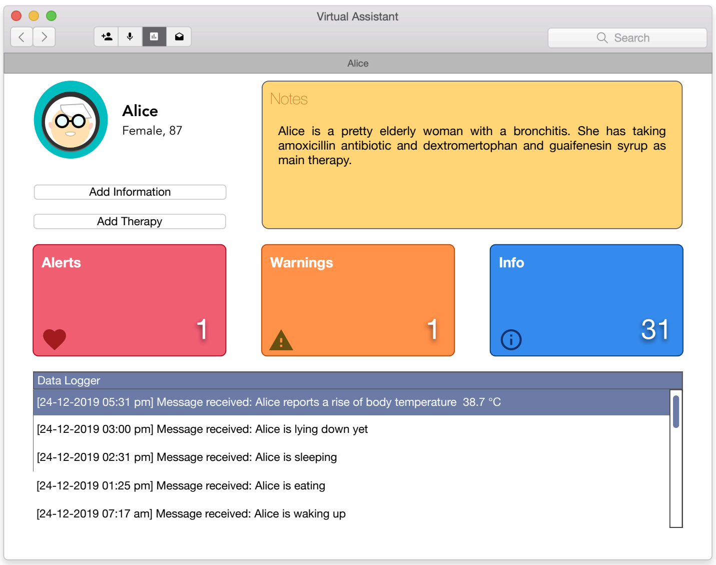
> ***Procedure and Human Interaction*** *– They consist of procedures and practices administered by humans. In this type of falls therapies for counseling, family, education but also psychotherapy, cognitive behavioral therapy, rehabilitation, physical therapy such as vision therapy, massage therapy. Lifestyle modification and life-coaching fall also in this category.*

Each therapy is supported by a set of conditions to apply it. Conditions are made by individuating data acquired from the physician, observing the patient's behaviors and activities. Consider the pilot scenario.

Alice is affected by bronchitis, so Dr. Haus inserts his diagnosis and the related therapy by using his remote controller application. The mock-up in Figure 6.3 represents the interface of the remote controller application. Handling this interface is part of the work of the *Virtual Assistant* agent. From the point of view of the agent, the diagnosis is translated into a belief or a set of beliefs useful for selecting a plan. Instead, therapy is translated into one plan. The system is programmed in a way that lets MyRob constantly monitor Alice, administer therapy, and update the physician through the *Knowledge Manager* agent.

Suppose that something new happens, Alice worsens and a sudden fever appears. MyRob detects the fever by interacting with her and it reports the *Virtual Assistant* agent notices the event and an alert is shown into the Dr. Haus terminal.

Even if Dr. Haus has just deployed some plans for handling fever and MyRob could select the best plan to face this new situation, it alerts him and waits for consent for administering the alternative therapy. Dr. Haus can confirm previous therapy and lets MyRob administer the recovery plan or inspect data for changing therapies. If Dr. Haus decides to consent to previous therapy, the *Virtual Assistant* agent sends an ack to the



Alice is a pretty elderly woman with a bronchitis. She has taking amoxicillin antibiotic and dextromertophan and guaifenesin syrup as main therapy.

```
[24-12-2019 05:31 pm] Message received: Alice reports a rise of body temperature  38.7 °C
[24-12-2019 03:00 pm] Message received: Alice is lying down yet
[24-12-2019 02:31 pm] Message received: Alice is sleeping
[24-12-2019 01:25 pm] Message received: Alice is eating
[24-12-2019 07:17 am] Message received: Alice is waking up
```

Figure 6.3: A mock-up of the Virtual Assistant agent deployed into the physician's terminal. The tab contains details about the patient. Alerts, warnings and info are shown into the namesake pills, the data-logger contains exchanged messages. Taken from [125].

Planner agent and MyRob continues working applying it, otherwise, Dr. Haus has to produce a new therapy. The multi-agent system allows Dr. Haus to insert all the needed therapies for facing the disease. He does this by evaluating information acquired by MyRob through its components.

Dr. Haus compiles the new therapy following some basic rules to be compliant with the system. Rules are necessary for translating therapies in the "AgentSpeak Language". This task is also in charge of the *Virtual Assistant* agent.

The translation process follows some basic rules, resumed in Table 6.2:

- a therapy owns a triggering event, conditions for detecting it and a list of procedures to act when conditions appear and they are verified;
- the triggering event has to catch the status that the system has to attention for handling the situation. It is defined with a label that represents the *triggering event name*;
- conditions have to be identified within the mind a list of symptoms or observations that the architecture can retrieve from the environment. Symptoms are perceptions

Therapy	**Plan**
Diagnosis Identifier	Triggering Event
Condition to administer therapy	Context Pre-conditions
Indications and Treatments	Actions, Sub-plans, Messages

Table 6.2: Mapping table between therapy and plan. The table contains the one-to-one correspondence that we identified for letting remote deployment of therapies in robotic caregivers that use the proposed architecture.

or beliefs and they are used in the unification process and logical inference for validating a context;

- the procedures list has to contain actions, behaviors or other therapies (plans) that try to resolve the issue acting or interacting with the agent;
- a procedure can be a simple action or a more complex structure that involve the usage of other therapies;
- an alternative therapy can be added into the plan library keeping the same *triggering event name* and changing conditions for validating it. If conditions of the default therapy fail, alternative therapy can be executed.

For adding therapies, Dr. Haus has to choose a triggering event name used to add the therapy into the cognitive system. To do this, Dr. Haus uses the interface (Figure 6.3) shown by the *Virtual Assistant* agent. Before, he has to add information useful for applying the therapy that works as conditions for administering the therapy.

In this case, Dr. Haus evaluates that one pill of paracetamol is a good therapy to cure Alice of the fever. Dr. Haus clicks on `Add Information` button and he writes:

```
  alice; current_diagnosis; fever;
  fever; paracetamol; 1;
```

Listing 6.1: Data entered into the system from the remote platform.

Once the aforementioned information has been added to the knowledge base, new beliefs are registered. The new beliefs are necessary for the new treatment. Now a new therapy can be added by clicking the namesake button in the interface. Dr. Haus choices a *triggering event name* that represent the health problem. Dr. Haus clicks on the `Add Therapy` button and assigns the name for this therapy through a textbox. He decides to call it `to_handle_fever`. A new screen is launched and Dr. Haus has to select which conditions (beliefs) must be satisfied to perform the therapy, next to Dr. Haus has to compose the list of indications to specify which behavior MyRob has to show to perform the therapy. The list of behaviors is shown and Dr. Haus indicates the order of action execution. The list of indications for the new therapy is translated in plan and the ordered list is made by Dr. Haus. It looks like:

```
   to_handle_fever : doctor(D) & D=='Dr. Haus' &
                     patient(A) & A=='Alice' &
                     current_diagnosis(F) & F=='fever' &
                     fever(P,Q)
                   <- .check_bodytemperature(A);
                      .search_medicine(P);
                      .prepare_medicine(P,Q);
                      .take_drug_to(A);
                      .check_therapy_taken_by(A);
                      .signal_to_physician(D).
```

Listing 6.2: An example of plan generated by the system from doctor's evaluation.

The physician can select actions that have been previously programmed for a specific robot.

Summarizing, Dr. Haus interacts with *Virtual Assistant* agent for inserting new therapies for Alice (Figure 6.3), this agent translates therapies in plans and sends plans to the *Planner* agent. The *Planner* agent pushes new plans into its own Plan Library. The *Motion* agent makes sure MyRob can operate for administering the therapies.

6.3 Discussing the Architecture Quality and Validation

In this section, the architecture was evaluated employing one of the mainly assessed methods for analyzing and validating software architectures: the SAAM, Software Architecture Analysis Method [114].

The main goal of the evaluation process depends on how much software systems are capable of fulfilling its quality requirements and identifying risks [127, 56, 18]. So, what makes an architecture good is how much it fits the goals and needs of the organization that is going to use it.

A recent survey [156] presents a comparative analysis of software architectures evaluation methods, their result is a taxonomy of evaluation methods. From this taxonomy, it arises that comparing and validating software architectures is a hard task due to the different languages and notations used for defining architectures. The chosen method provides a notation for describing the architecture, mainly highlighting its structural perspective. Relevant qualities taken in charge of evaluating the architecture are maintainability, portability, modularity, reusability, and robustness.

Activities for validating the architecture using SAAM [114] are:

1. *Characterize a canonical functional partitioning for the domain.*
2. *Map the functional partitioning onto the architecture's structural decomposition.*
3. *Choose a set of quality attributes with which to assess the architecture.*
4. *Choose a set of concrete tasks which test the desired quality attributes.*
5. *Evaluate the degree to which each architecture provides support for each task.*

The first activity identifies roles and responsibilities to assign to agents in the architecture in Figure 6.1. Then, Figure 6.4 figured out the result of converting the architecture using SAAM notation. In so doing, computational entities are highlighted as well as data and control connections among them. SAAM then prescribes to choose some quality attributes. The cognitive architecture presents maintainability, portability, modularity, and reusability, as intrinsic qualities given the nature of the architecture, the multi-agent system. The agent paradigm allows designing a software system with a high degree of modularity and a low level of coupling among components that guarantee these quality factors. It was considered *robustness* for validating the architecture using also FURPS [58, 107] requirements category. *Robustness* [34] refers to *the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions*.

The need is having software able to adapt to changes and developing an architecture supporting such a kind of software. To validate the cognitive architecture, some tasks have been identified. Specifically, tasks are: (*i*) recognizing new patient's state, (*ii*) adding new suitable plans and (*iii*) reusing previous successful plans.

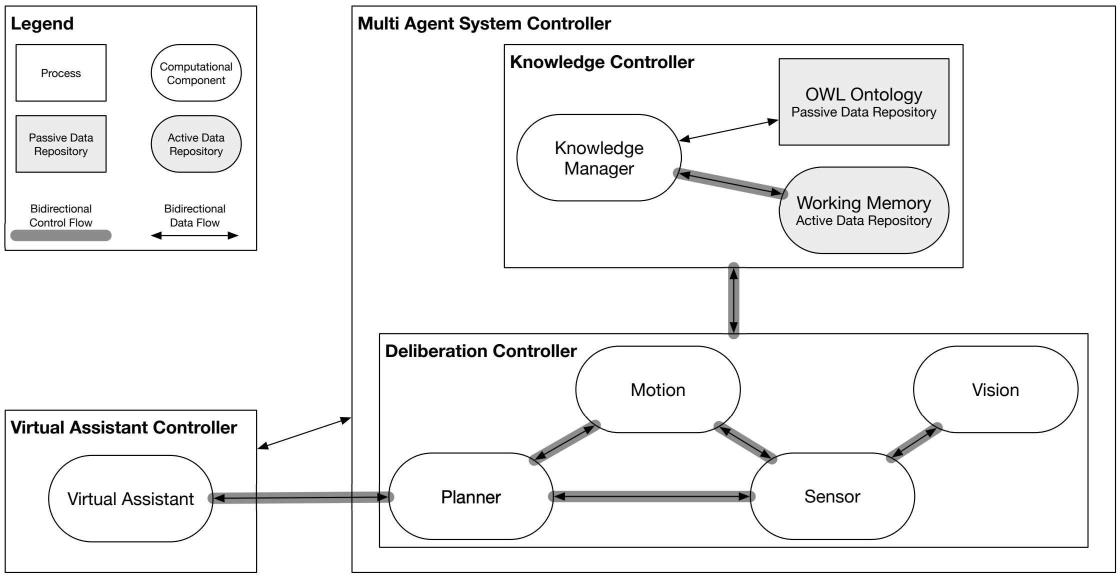


Figure 6.4: A SAAM based representation of the architecture in Figure 6.1. Taken from [125].

For validating the architecture in supporting these tasks, the first step is representing in the SAAM notation (Figure 6.4) the architecture. The SAAM notation has been created for separating the control flow from the data flow. As can be seen, three basic processes were identified whose thread of control is assured by the control flow and the data flow among the computational components, the active and passive data.

Recognizing new patient's state is a scenario in which guarantying the quality of robustness is fundamental. This scenario involves only two processes and one thread involving at the same time one control flow and one data flow. Inside the Deliberation Controller process, the responsibilities of computation are divided among four computational components. Each component is in charge of one specific monitoring function that is directly transferred to the System Controller process. Computational components may communicate with each other; communications are direct and not mediated by other components and processes. This point assures that a change in the environment is immediately caught and transferred to the related computational entity and then directly communicated to the knowledge controller. Each computational entity is atomic and quite reactive, so robustness and the ability to intercept changes is comparable to a portion of object-oriented code devoted to catching an event.

Adding new suitable plans, this task is supported by a single computational component inside the Virtual Assistant Controller computational component. Only one control flow and one data flow are necessary. The communication with the Planner computational component, devoted to managing the new plan, is direct. In this case, since the Virtual Assistant provides the right interface for inserting a new plan, an error is highlighted during the insertion process and immediately communicated, avoiding the risk of failure or unexpected input propagation.

Reusing previous successful plans is ensured by the system. These plans are stored in the Working Memory and are available to be selected when the patient's conditions fit with the plan's preconditions. Each time that the patient's state changes, the Multi-Agent System Controller computational module takes control for actuating a plan, interacting with the user if needed through the Virtual Assistant Controller. Later on, it sends data and control to the Deliberation Controller process. In this task, all computational components are involved, but mainly only two are relevant, the Multi-Agent System and the Deliberation Controllers. Supporting this task is more demanding since there are two data and one control flow exchange. The process involved in the thread of control is of two different levels where one comprises the second. Dependencies between processes could create an architectural coupling that may slow the thread of control for supporting that task. For instance, the scheduled plan might depend on other processes' results and the overall outcome is afflicted by the computational time needed. In so doing the reaction to the change may not be responsive and other sudden events may invalidate the chosen plan, affecting the system's robustness. Hence, this situation may make all the system asynchronous towards changes.

From the validation process, it was realized that the proposed agent architecture fits all the needs related to the functional requirements to project the healthcare system. As said, architecture is not good or bad in general but in concerning some specific goals. SAAM revealed that the architecture is modular and decoupled enough for support in reacting to changes at runtime.

Modules were tested onto the robot Pepper[2], simulating hypothetical input perceived from the environment.

[2]https://www.softbankrobotics.com/emea/en/pepper

6.4 Summary

This chapter shows an application of human-robot teaming interaction, based on the cognitive architecture described in Chapter 3, on which a robot and a human have to cooperate for assisting a patient in his home.

The usage of the proposed cognitive architecture is subordinate to the requirement that the scenario claims.

All modules were adapted to be used within the context domain. The objective is to implement within the architecture only the functionalities that are useful for the scenario.

The intent is testing the cognitive architecture in a specific scenario exploiting it in its main parts. The module of learning plans at runtime was not directly used, because forbidden by the domain, and used for suggesting strategies when necessary. The suggestion may be considered as the step before autonomous action and our way is in line with the concept of interaction with humans.

The robot is endowed with the ability to handle unforeseen situations and to communicate and collaborate with the physician, thus providing him with intelligent support.

The exploitation of the proposed architecture lets realizes all the system's functionalities may be split in the space. In this way, the overall system may be easily scalable and adaptable to any context. The agents, to be implemented based on the architecture, have been conceived at a higher level than the implementation one. Hence, thinking to their macro-level functionalities. It is for this reason that the combination of architecture plus agent system can be adapted and extended to other and more complicated clinical scenarios.

The validation made with the SAAM technique revealed the goodness of the cognitive system and it shows its potentiality. Finally, this proof of concept shows the feasibility of using this architecture for building an autonomous robot, able to adapt to the cooperative context.

Chapter 7

Theoretical Models for Trustworthy Interactions

Most of our lives depend on trust: our ability to manage it, granting or denying it whenever appropriate, is critical for our safety and well-being during our everyday lives. Misplaced trust can have catastrophic effects on our physical, emotional, or economic welfare. Being such an important factor for humans, it is natural to think that it would also benefit for the social robots we are hoping to involve in our future relationships. In particular, regarding shared goal scenarios where a human and a robot need to interact and collaborate to achieve a common objective, it is important that both the involved agents are able to estimate the trustworthiness of each other so as to adopt the best decisions that would ensure the successful completion of the task. Castelfranchi and Falcone [38] argue about the importance of trust in the context of society: we, as humans, use it to define our behavior, to guide our decisions, and in general to build successful relationships. The use of trust dynamics doesn't have to be limited to human interactions but can be extended, with all its virtues, to relationships with artificial agents. Because of this, robots could improve their performance by being able to perform decisions based on trust.

Two different perspectives are discussed in this chapter. The first model exhibits an implementation of the trust theory developed by Castelfranchi and Falcone [38]. The main contribution of the first model [39] is the ability to justify decisions when something wrong occurs. This process studied in the theory of trust contributes to making artificial intelligent systems trustworthy and explainable. The model, developed using the Jason framework, implements an adoption/delegation model for trustworthy interaction.

The second model [126] introduces a theoretical architecture based on global workspace theory [15] on which expert systems compete *to catch the spotlight of the conscious* to determine a level of trust to the person with whom the system interacts. It merges two kinds of artificial intelligence approaches, the Bayesian and the deep learning ones. This model is currently theoretical.

In the following sections, models are introduced and discussed for explaining theories that drove the research.

7.1 Endowing Robots with Self-Modeling Abilities for Trustful Human-Robot Interactions

Robots involved in collaborative and cooperative tasks with humans cannot be programmed in all their functions. They should be autonomous entities acting in a dynamic and often partially known environment.

Every robot applications present some kind of interaction with humans through explicit or implicit communications. In the case of autonomous robots operating as a teammate towards humans, humans provide the goal and the robot has to be able to maintain knowledge about the environment and the tasks to perform in order to decide whether to adopt or delegating a task or an action.

Autonomy, proactivity, and adaptivity are the features to decide, at each moment, which activity has to be fruitfully performed for efficiently pursuing objectives.

In the case of a team composed of only humans, the interaction with a teammate is based on the level of knowledge owned on the environment and on the “other”. Especially, knowledge about the capabilities of the other, about the interpretation of the actions of the other concerning the shared goals, and therefore also about the level of trust that is created towards the other. Trustworthiness is a parameter to be used for letting an entity decide which action to *adopt* or which to *delegate*.

To make compliant with the rest of the system, this self-modeling ability was done employing the BDI agent paradigm [28] and developed using the Jason framework [27].

Another objective of this research work is to implement interactions in teams of humans and robots, so that collaboration is as efficient and reliable as possible [43]. To do this, both entities involved in the interaction need to have a certain level of confidence in each other, that for facilitating the nomenclature we will call the *trust level*.

Measuring trust in the other is made easier if he has full knowledge of his capabilities, or if he can understand his own limitations. The more one of the two entities is aware of its limitations and abilities, the more the other entity can establish a level of confidence and create a productive and fruitful interaction. Implementing a justification capability may create a productive and fruitful interaction.

The computational model, developed for endowing the robot with this capability, was tested in a delegation and adoption task described by Castelfranchi and Falcone [41, 40, 63].

The idea is to exploit practical reasoning in conjunction with the well-known model of trust [38, 65] to let the robot create a model of its actions and capabilities, hence some kind of self-modeling abilities.

Starting from the BDI practical reasoning cycle, the deliberation process and the belief base representation were extended in a way that allows the robot to decompose a plan in a set of actions strictly associated with the knowledge useful for performing each action. In this way, the robot creates and maintains a model of the "self" and can justify the results of its actions.

Justification is an essential result of self-modeling abilities application and at the same time is a useful means for improving trustful interactions.

7.1.1 Self-Modeling using BDI Agents

How to design and implement a team of robots that possess a model of themselves, of their actions, behaviors, and abilities? And more, how to allow robots reason about themselves and infer information about their activities, such as why action has failed?

The trust theory [65, 38] was implemented through the multi-agent paradigm and the BDI model [28]. The trust theory model was integrated into the traditional BDI working cycle [27].

For employing the model of trust, it was considered the robot as the *trustee* and the human as the *trustor*. Assuming that the human delegates a part of his goals to the robot, the level of trust the human has in the robot may be derived from the robot's ability to justify the outcome of its actions, especially in the case of failure. Indeed, self-modeling is the ability to create a model of several features realizing the self. Among them the knowledge of owns capabilities, in the sense that the agent is aware of what it is able to do, and the knowledge on which actions may be performed on every part of the environment. Justifying action is the result of reasoning about actions, it is a real implementation of the self-modeling ability of an agent (human or robot). For doing this, it was proposed to represent the robot's knowledge through actions and beliefs on those actions.

In particular, it was developed a method for implementing the justification of an action, or of behavior, which should comprise components allowing to reason about the portion of knowledge useful for performing that action.

This has to be made for each action of a complete plan. If an action is coupled with all the concepts it needs for being completed then the performer may know at each moment whether and why an action is going wrong and then it may motivate all eventual faults. This scenario is the result of the implementation of self-ability and contributes to improving the trustful interaction. In the sense that trust, and then the attitude to adopt or delegate, may change accordingly.

In Section 2.5, in the trust function, the mental state of the trust is achieved through actions, agent beliefs are implicit and do not appear as direct variables in the trust function. For implementing it, it was necessary to associate to the plan (or to the action) the belief or the set of beliefs required to execute it. This choice allowed to map the theory of trust with the theory modeling the BDI cycle and to regularly report the new BDI cycle to the implementation part including Jason.

It was necessary to introduce a new representation in the model of τ from [38].

$$TRUST(X\ Y\ C\ \tau\ g_X) \quad \text{where} \quad \tau = (\alpha, p) \quad \text{and} \quad g_X \equiv p;$$

By combining the trust theory model and the self-modeling approach, τ is a couple of a set of plans π_i and the related results p_i. Indeed, now the trust model may implement the BDI paradigm breaking down actions and results into a combination of various

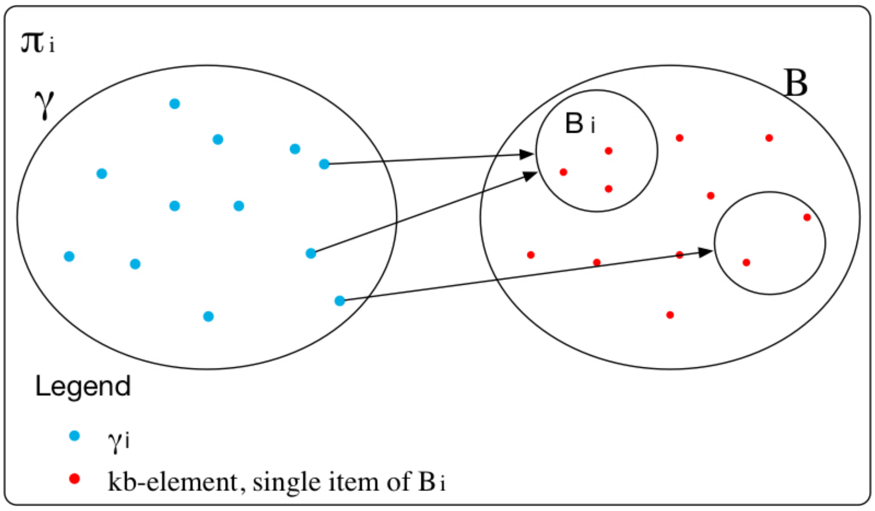


Figure 7.1: Mapping actions onto beliefs.

arrangements of plans and sub-results.

The model of τ is formalized as:

$$\tau = (\alpha, p) \quad \text{where} \quad \alpha = \bigcup_{i=1}^{n} \pi_i \quad \text{and} \quad p = \bigcup_{i=1}^{n} p_i$$

Moreover, each atomic plan π_i is the composition of action γ_i and the portion of belief base B_i for pursuing it; formalized as:

$$\pi_i = \gamma_i \circ B_i \Rightarrow \alpha = \bigcup_{i=1}^{n} (\gamma_i \circ B_i)$$

B_i is a portion of the initial belief base of the overall BDI system. The $\circ$ *operator* represents the composition between each action of a plan with a subset of the belief base (Figure 7.1).

This theoretical framework was implemented in the NAO robot exploiting Jason [27] and CArtAgO [171] for representing the BDI agents and the virtual environment. The environment model is created through the implementation of a perception module using NAO. Actions, into the real world, are performed using CArtAgO *Artifact* through `@Operation` functions.

What happens while executing actions can be explained by referring to the BDI reasoning cycle. Once the robotic system has been, at a first stance, analyzed, designed, and put in execution all the agents involved in the system acquire knowledge. They explore the belief base and all the initial goals they are responsible for (points 1. 2. 3. 4. - Figure 2.10). Then, the module implementing the deliberation and means-and-reasoning (points 5. 6. 7. - Figure 2.10) is enriched with a new function. Commonly at this point, while executing the BDI cycle, the tail of actions for each plan is elaborated to let the agent decide which action to perform. Since the interest is toward the tail of actions and in all the knowledge useful for each action, a new function was added:

$$A_c \leftarrow \textit{action}(B_{\alpha_i}, Cap) \tag{7.1}$$

where B_{α_i} and Cap are respectively the portions of belief base related to the action α_i and the set of agent's capability for that action.

Agents execution and monitoring imply points 8. 9. 10. 11. 12 of the BDI cycle. The novelty stands in enriching the BDI model with a new portion of the algorithm able to identify the *impossible* (I, B) and $\neg$ *succeeded*(I, B) (ref. point 9 of Figure 2.10).

In this step the effective trust interaction takes place, here we may assume that the robot is endowed with the ability to re-planning, justifying, and requesting supplementary information from the human being. Thus making the robot fully and trustfully autonomous and adaptive to each kind of situation it might face or learn depending on its capabilities and knowledge. The newly added functions, only for the case of *justification*, are shown in the following algorithm:

Algorithm 6: The process for evaluating and justifying the α_i element.

```
foreach $\alpha_i$ do
    evaluate($\alpha_i$);
    $J \leftarrow$ justify($\alpha_i$,$B_{\alpha_i}$);
end
```

Figure 7.2 details all the elements and the mapping process among beliefs, actions, and plans. Summarizing, τ is the goal that a trustor decides to assign to a trustee; it means that a BDI agent is assigned the responsibility to perform all the actions γ_i included in τ.

In the following section, the model was validated by developing an experimental scenario in which humans and robots interact. The experiment was performed employ-

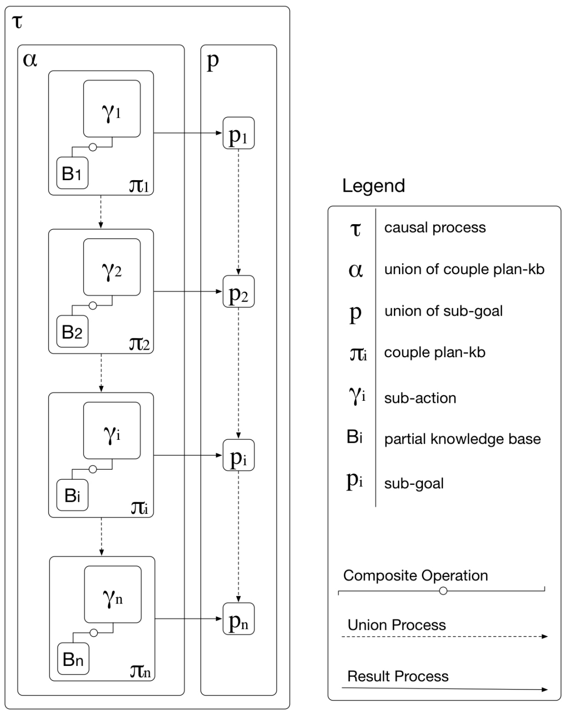


Figure 7.2: A block diagram representation of the causal process τ.

ing the NAO robot and one human.

7.1.2 The Robot in Action using Jason and CArtAgO

This case study focuses on a human-robot team whose goal is to carry a certain number of objects from a position to another in the room. This setup what thought to implement the first type of delegation shown in section Section 2.5: the *literal help*. The environment is composed of a set of objects marked with the landmarks useful for the NAO to work, the set of capabilities is made up basing on the NAO features, for instance, to be able to grasp a little box. The NAO is endowed with the capability of discriminating the dimensions of the box, and so on.

Let us suppose to decompose the main goal (as shown in Figure 7.3) *BoxInTheRigth-Position* in three sub-goals, namely *FoundBox*, *BoxGrasped ReachedPosition*. Let consider the sub-goal *ReachedPosition*, two of the actions that allow pursuing this goal are: *goAhead* and *holdBox*. The NAO has to go ahead towards the objective and contemporarily hold the box. The beliefs associated with these actions refer to the concepts of

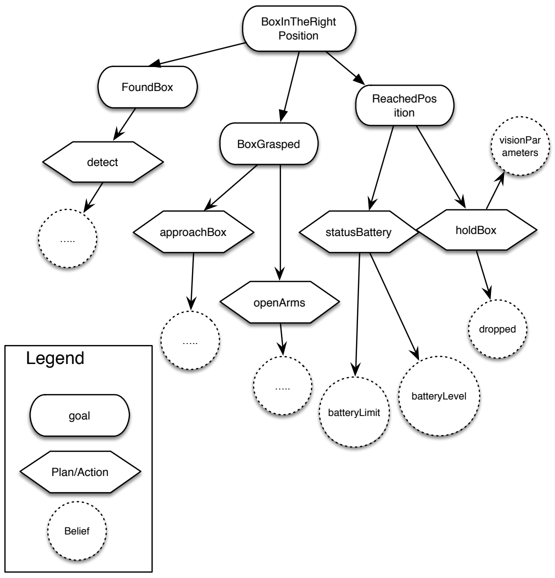


Figure 7.3: A portion of the assignment tree for the case study. For space concerns are shown only an excerpt of the whole Assignment Tree diagram, so only few explanatory belies for each action are reported.

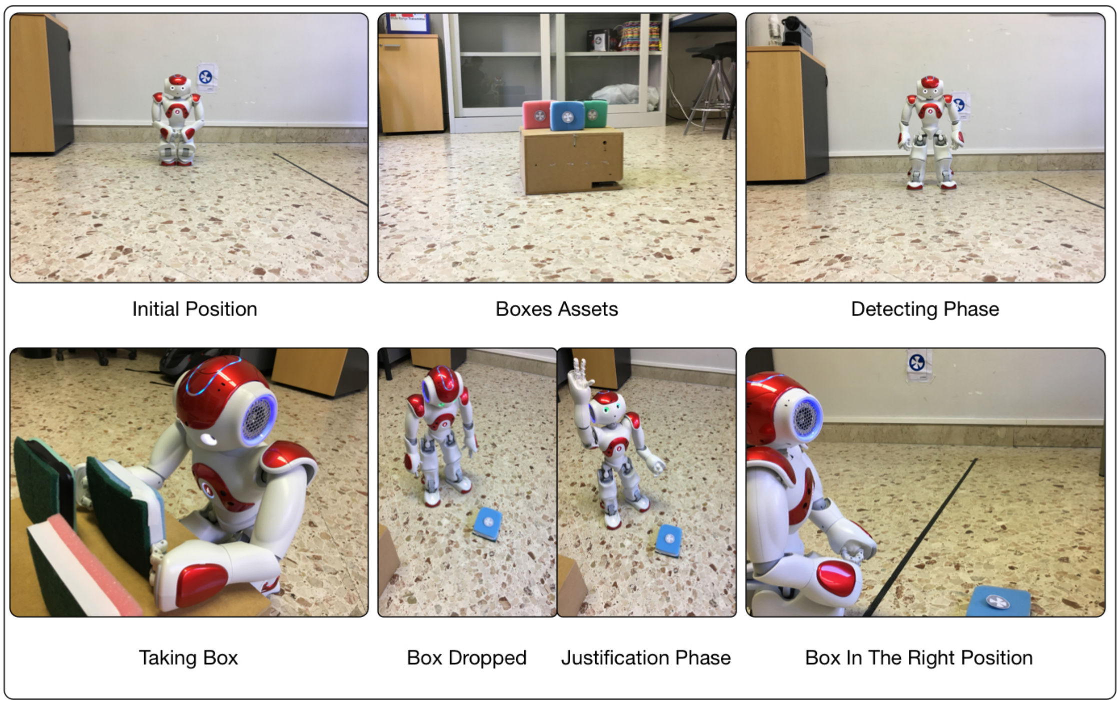


Figure 7.4: The NAO working on the *BoxInTheRightPosition* goal and the justification.

the knowledge base these actions affect. In this case, one of the concepts in the box, it has attributes like its dimension, its color, its weight, its initial position, and so on. The approach used for describing the environment results in a model containing all the actions that can be made on a box, for instance, *holdBox*, and a set of predicates representing the beliefs for each object, for instance, *hasVisionParameters* or *isDropped*. They lead to the beliefs *visionParameter* and *dropped* that are associated with the action *holdBox* through the relation number (7.1.1).

In the following, a portion of the code related to this part of the example is shown.

Algorithm 7: The portion of code that implements the τ decomposition.

```
+!ReachedPosition: true ← goAhead; holdBox. [$\tau$];
+!goAhead: batteryLimit(X) & batteryLevel(Y) & $Y < X$ ← say(“My battery
  is exhaust. Please let me charge."). [$\gamma_1^+$];
+!goAhead: batteryLimit(X) & batteryLevel(Y) & $Y \geq X$ ← execActions.
  [$\gamma_1^-$];
$B_1$: batteryLimit, batteryLevel ;
+!holdBox: dropped(X) & visionParameters(Y) & $X == false$ ←
  execAct(Y). [$\gamma_2^+$];
+!holdBox: dropped(X) & visionParameters(Y) & $X == true$ ← say(“The
  box is dropped."). [$\gamma_2^-$];
$B_2$: dropped, visionParameters ;
```

In Figure 7.4 some pictures showing the execution of the case study with the NAO robot.

In conclusion, the justification model here described refers to the trust theory model. The aim is to endow agents (or robots) with the ability for letting them explain. The tropos-like diagram in Figure 7.3 shows the plan decomposition treated in the revised model of the theory of trust, the decomposition makes possible the transformation process for identifying the action guilty of failure. The diagram in Figure 7.3 was enriched with a dotted circle to represent beliefs that normally are not figured in a tropos diagram. It is good to underline, that the showed model is developed for letting robot implementing justification and it is related to the Chapter 5, and specifically, with the simulator. The justification process makes the robot, and therefore its decision-making process, transparent to the eyes of its human teammate. This facilitates interaction and helps make interaction reliable and trustworthy.

7.2 A Global Workspace Theory Model for Trust Estimation in Human-Robot Interaction

In this section, the focus is on the trust that is assigned from a robot to a human, which means that the former will be the trustor and the latter will be the trustee.

The theoretical model takes into account an existing computational model of trust and Theory of Mind (ToM) proposed by Vinanzi et al. [199]. The proposed extension that refines the trust estimation procedure, takes into account social cues, like emotions, facial action units, and gaze direction. The objective is to overcome the two main limitations of this current model, which are: the unimodality of the perceptual information used and its sole reliance on personal experience. The proposed model acts as a framework and it uses machine learning techniques to gather the previously mentioned social features from an HRI scenario and uses the Global Workspace Theory (GWT) principles to determine whether the partner should be trusted or not during the joint action engagement.

7.2.1 The Global Workspace Model

The Global Workspace Theory (GWT) model depends on the interaction between several specialized nodes of a network, which are the expert systems that compete for the spotlight of the artificial consciousness. Figure 7.5 shows the proposed architecture based on GWT. The cognitive model hosts two expert systems: the Experience Expert System (ExES) and the Engagement Expert System (EgES), which are described in detail in the following Sections and are both capable of providing trust estimations. The GW component is in charge of deciding whether to assign the spotlight to one system or the other. This architecture is modular, meaning that the expert systems can be changed in typology and number based on specific needs.

Experience Expert System The ExES [199] is a developmental robotics cognitive architecture that enables a robot to perform trust evaluations based on personal experience. Its core component is a Bayesian Belief Network (BBN) that unifies trust and ToM considerations for the sticker finding game, as described in Figure 7.6. Following the original ToM experiment [195], it performs two main functions:

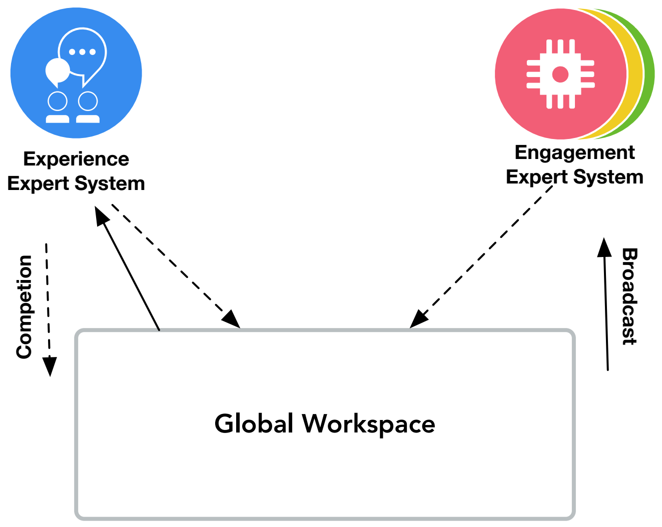


Figure 7.5: The GWT model to predict trust in HRI. It is composed of two expert systems, both of which use different features to estimate the informant's trustworthiness. The Experience Expert System (ExES) uses previous experiences of interactions, while the Engagement Expert System (EgES) makes use of human-like features detected on the partner during the interaction. The final decision of the system will be moderated by the Global Workspace (GW).

1. **Familiarisation**: the robot experiences some interactions with the informants, observes the outcome of their suggestions learning who should be considered a helper or a tricker, and trains a BBN for each of them;

2. **Decision making**: the robot will use probabilistic inference on the BBNs to decide whether to follow or reject the suggestion for each particular informant. See [199] for more information and a visual demonstration applied to the Vanderbilt experiment [195];

In addition to this, and following the psychological principles of trust development defined by Erikson [61], the robot can use its Episodic Memory to collect all of its past experiences and decide how to act towards a novel informant it has never familiarised with. The history of interactions shapes the robot's personal character development, which will determine its attitude to instinctively trust or distrust a stranger.

Engagement Expert System The EgES is a cognitive model whose purpose is to estimate the trustworthiness of the informant using social cues. According to Ekman

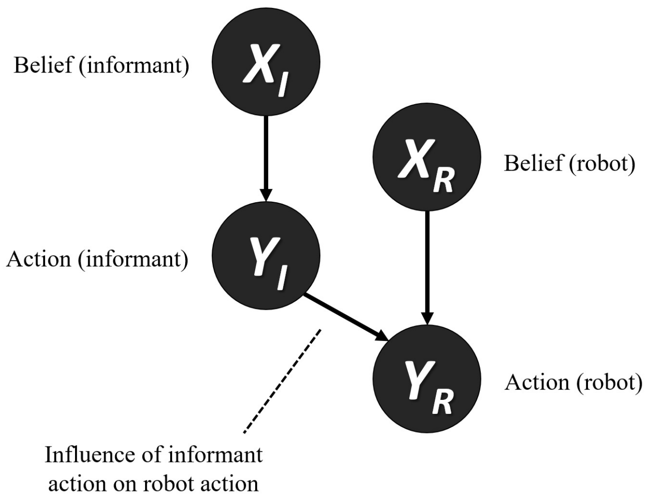


Figure 7.6: The BBN that models the relation between the robot and an informant within the ExES. The agent generates a separate network for each informant, with the same structure but different probability distribution. Redrawn from [154].

[59], emotions and micro-expressions are used as subtle hints to understand whether someone wants to trick us.

Using this principle, EgES feeds the robot's RGB camera images and auditory perceptions to a stack of machine learning algorithms that classify the current perception and extract a set of features for an Artificial Neural Network (ANN). The latter is used to predict the appropriate level of trust to assign to the informant under analysis. Figure 7.7 shows a schematic representation of this process. The computed features are the following:

- **Emotions**: the emotional state of the informant. This system classifies emotions as sad, angry, happy, or neutral;
- **Vocal Emotions**: the emotion expresses by the informant's voice signal, without considering its content;
- **Facial Action Units**: facial movements and micro-expressions;
- **Gaze**: the direction of the informant's gaze, to determine if the informant is looking directly at the robot or not;

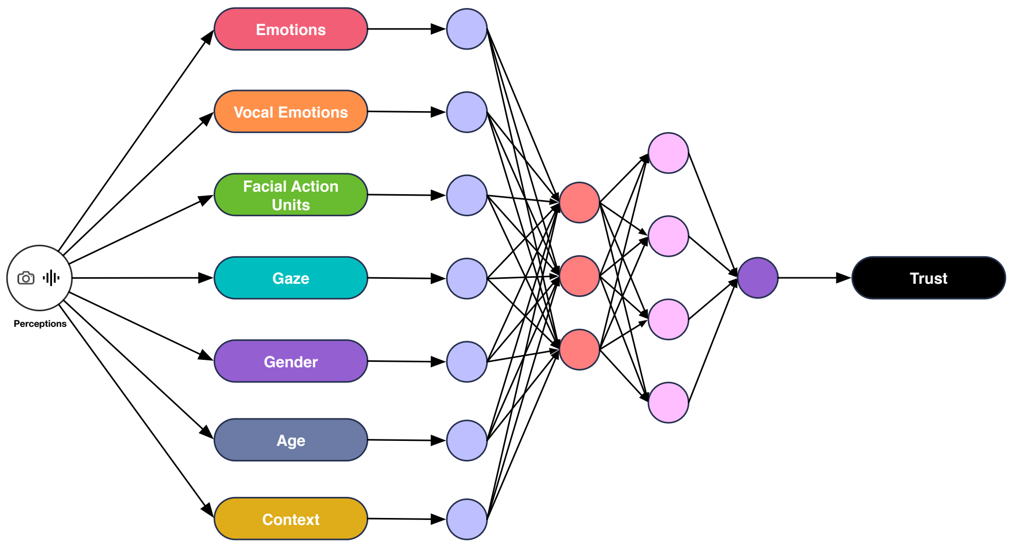


Figure 7.7: The EgES architecture. Perceptions are processed by a stack of machine learning algorithms which will compute a set of features. The latter are processed by an ANN to determine the final level of trust.

- **Gender**: different genders show differences in their social cues, this feature takes this into account;
- **Age**: as above, social cues differ with age;
- **Context**: the state of the environment, such as calm or agitated.

The proposed GWT model is built upon two expert systems that are able to independently estimate the trustworthiness of another agent engaged in joint action. These two systems will compete to earn the spotlight of selective attention by the GW, which represents the artificial consciousness of the system.

7.2.2 Implementation Details

This model is currently an ongoing work at the second phase of the experimentation, where humans are involved with the aim to collect data for training the Engagement Expert System.

The implementation involves the ROS framework for orchestrating singles social cues nodes meanwhile humans interact with the robotic platform. Data are collected

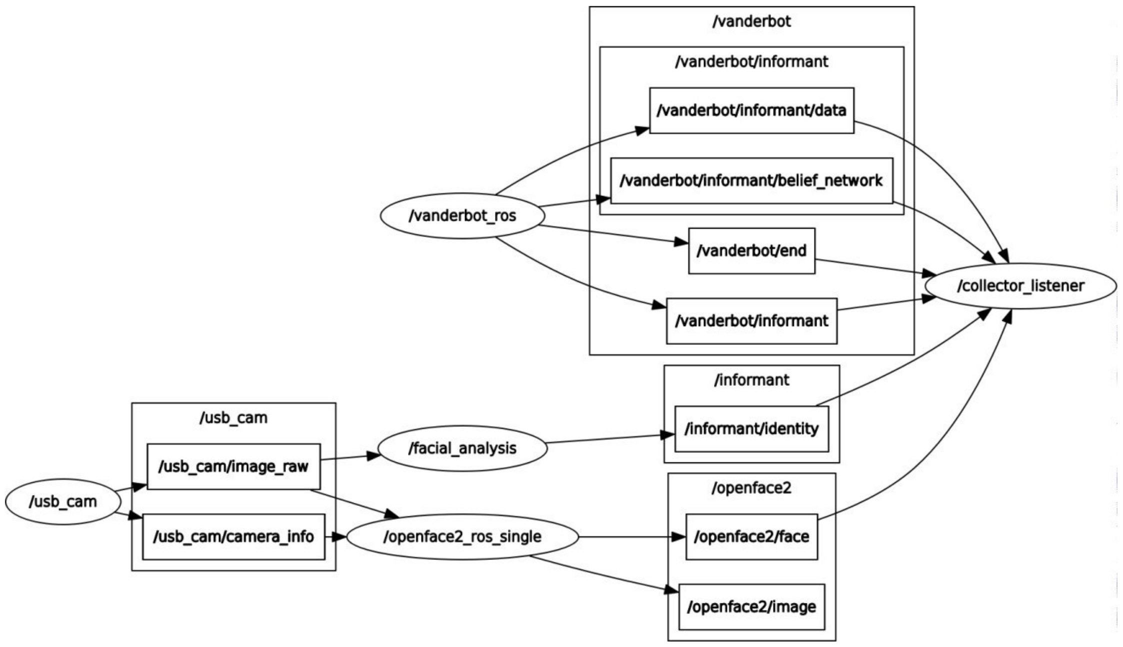


Figure 7.8: An overview of ROS architecture for implementing the EgES system with the addition of the ExES rewrote to be compliant with the ROS framework.

analyzing images with a 90 fps camera and feature are extracted using state-of-the-art algorithm for age and gender estimation [175, 176], the OpenFace framework for face feature analysis [22], such as the facial landmark detection and tracking [21, 221], eye gaze tracking [213], facial action unit detection [20]. Figure 7.8 shows the developed ROS package used to collect data in the first instance.

7.3 Summary

There is a big body of research that suggests the importance of trust in any kind of relationship, especially between team members [93].

In the previous sections, theoretical models for implementing trustworthy interactions among humans and robots were presented. The objective is building machines able to understand and make own the concept of trust in all its form [38, 66, 66, 67].

The first section shows an implementation of the trust theory model proposed by Falcone and Castelfranchi [38]. The main goal of this work is to equip the robot with the self-modeling ability that allows it to be aware of its skills and failures to implement a justification process within the cognitive architecture of Chapter 3.

The trust model has been integrated with a BDI-based part of the deliberation process to include self-modeling, exploiting the plan library of an agent. The model lets the cognitive architecture implement justification. The justification is also used within the simulator described in Chapter 5.

The outcomes in the various phases are not binding but inspired by a tropos-like [33] goals-oriented diagram shown in Figure 7.3. However, it is possible to use whatever methodological approach giving a view of goals and their decomposition, and decomposition into plans in a way useful to match with the related knowledge base.

The second section shows a theoretical model for trust estimation. The aim to estimate, and so, to measure trust is important for letting agents and robots decide to take into account a kind of social feature used by humans during team cooperation, or in general, to instance reliable social relationships. At the moment, the trust cognitive model by Vinanzi et al. [199] can perform trust evaluations based solely on past and present experiences of the involved agent, but the implementation of both the EgES and the GW seem to be able to enhance this decision making process by taking into account other factors that would influence human beings engaged in the same task, thus freeing this cognitive model from the boundaries of personal experience. This last model is shown but it is currently a theoretical model that aims to enhance the quality of this type of cognitive capability.

In conclusion, these models aim to contribute to the creation of transparent and explainable AI systems. These qualities are necessary to build the next generation of intelligent systems, ethical AI systems [193]. Namely, systems able to contribute to the improvement of the quality of life, respecting ethical standards, and requisitions typical of our world.

Chapter 8

Conclusions

Endowing robots with abilities for acting as humans during daily life tasks is the ambition that moves the research activity whose results are shown in this thesis.

In conclusion, I want to recall the main question that drove this dissertation:

> *How does a human being act in a team whose goals are known and shared in a changing environment? And, how can this condition be managed in a team made by humans and robots?*

The answer to these questions is in the proposed cognitive architecture, in the way the knowledge can be managed and incremented, novel plans can be composed at run-time, and in how it is possible to anticipate action results before applying them in the real world. Considering the cognitive architectures discussed in Chapter 2, the proposed cognitive architecture was thought to be used in an HRTI domain, this entails a series of trade-offs to accept to apply it as a general cognitive architecture. The proposed cognitive architecture does not want to be a solution for every possible problem in the field of cognitive systems but it was thought to be used in robotics cooperative scenarios. One-size-fit-all software architectures are very far from being feasible and reliable, also considering the complex domain in which I am investigating.

Thus, this research aims at proposing a novel solution for the field of Human-Robot Teaming Interaction (HRTI). Based on a deep analysis of existing cognitive architectures, it was realized a novel cognitive architecture thought for the robotics fields. This architecture takes into consideration aspects of cognition, such as self-modeling, for better meeting human-robot interaction in collaborative and cooperative scenarios.

The architecture employs the multi-agent paradigm and theories of rational agents for building a multi-agent system in which the decision-making process reproduces the human counter-part.

The choice to use a multi-agent system is justified by the fact that a human-robot teaming interaction system can be considered a complex system. For designing and building this type of system, the agent-paradigm, and the multi-agent oriented programming, well supports the application domain.

Besides, a multi-agent system helps in seeing the problem of building a cognitive architecture from another standpoint, the one that concerns the vision of cognitive architecture as a complex system. In this regard, the architecture has been modeled and developed, considering its parts that cooperate and collaborate to achieve the common goal of the cognitive/complex system.

The state-of-the-art agent-based cognitive architectures use a single agent that performs a reasoning cycle for deciding which action to perform.

To handle robots, maybe is not sufficient for representing or handling a robot with a single agent. Even because, with a single-agent view, the reasoning cycle becomes hard to adapt to the different approaches that may help programmers to overcome some issues during the interaction. For instance, the proposed cognitive architecture can scale its modules through a set of agents, even handled at runtime, and to implement on them symbolic and sub-symbolic reasoning techniques without constraints and restrictions. The solution overcomes these issues letting the programmer distribute the system over several computational nodes, such as High Performance Computing (HPC). This implies that the cognitive architecture may be extended without limitations of approaches and techniques.

To follow, a summary of the results achieved in this doctoral thesis, with particular attention to the novelties achieved.

A Cognitive Architecture for Human-Robot Teaming Interaction The inner life of a robot is composed of several cognitive abilities that move its decision-making process. Although cognitive processes include capabilities for thinking, knowing, remembering, judging, and problem-solving and are extremely useful in several contexts, these are not easy to implement in an artificial being. It is possible to define cognitive processes such as the brain's higher-level functions and embrace different intelligent activities such as *language, imagination, perception*, and *planning*. The Chapter 3 introduces the cognitive architecture implemented for robots that have to work as a member of a team. Specifically, I am referring to the novel designing solution to separate the inner robot's beliefs from the *motivations* that are strictly connected to the team and the *anticipation* module. Due to this, it was defined as an extended reasoning cycle for the BDI agent. The decision-making process uses both beliefs and motivation for pursuing an objective, in accord with the task assigned to the robot. The decision making is handled through a situation handler on which priorities, constraints, and requirements are taken into account for deliberating which intention has to be scheduled to perform. The deliberation process, exploiting the anticipation, can anticipate results and pursuing an objective, accordingly with the current situation; this means if something goes wrong, anticipating results robot does not perform the action, and so, it starts to find an alternative solution through other mechanisms discussed in previous chapters.

Knowledge Acquisition and Management As mentioned before, knowledge is one of the most important parts of our system. In a generic cognitive system, where all is based on a single agent, the agent performs a single reasoning cycle in a loop. Everything is just ready to use, but in a multi-agent system, all agents are distributed, and each of them works autonomously. Besides, agents employed for developing this architecture use the BDI paradigm on which beliefs, by definition, could be expired or wrong. This means that it was necessary to endow the architecture with a mechanism that regulates the acquisition and the distribution, as well as the organization of the knowledge into the system. A solution was proposed, implemented, and unveiled to be useful for knowledge management. As well as the novel algorithm uses semantic and syntactic similarities inherited by the natural language processing domain; the aim is facilitating the autonomous knowledge increment. Results showed that this method, besides being fast and accurate in organizing concepts in the knowledge base. The model generalizes and creates the best semantic match even in critical conditions, such as when the robot is working under the assumption of insufficient knowledge and resources.

Anticipating Actions and Learning Plans Another important intelligent activity concern the property to model itself, the surrounding context, and the environment through sensors or other capabilities owned by the robot. Besides, proactiveness is another cognitive ability for letting robots (or agents) operate intelligently in such domains. The cognitive architecture contains modules for representing the robot's extrospection, introspection, and proprioception, and an algorithm for letting agents (and so robots) compose high-level plans at runtime. In Chapter 5 these modules were presented and detailed. In the first case, the architecture's anticipation process uses the simulator to represent the robot, its capabilities, and its knowledge. The simulator helps the cognitive architecture to estimate and evaluate plans before applying them in the real world. The simulator has two functionalities, the first, to reproduce in a simulated environment the reasoning of the system and the decisions made by it, to test it before, implementing a kind of imagination; and the second function is to help the teammate to look inside the robot's mind, to clarify any possible situations as well as inspect the robot's knowledge. The proactiveness was developed employing an Artificial General Intelligence (AGI) model. The usage of NARS helps the BDI agent in trying to find a solution. The model is in charge of applying the NAL logic to the BDI agent through a conversion of the agent's knowledge. The model is detailed in Section 5.2. The implementation was made using the internal action of a generic Jason agent and the NARS framework.

Endowing agents with an ability that lets them continue to operate when no possible plans are available is an important step toward autonomy and self-adaptation. Results on the usage of this model promise to support complex scenarios even when proper conditions lack or the robot is not completely equipped with relevant plans necessary to face situations. The major limitation of this model is the library of plans given at design time by the *agent programmer*. To be fruitful, the model needs a library of plans on which complex plans are composed by atomic plans (or simple plans), easy to shuffle for composing novel valid high-level plans. Moreover, this method, joint with the simulator, can test and verify generated plans to use them only if they are applicable to the context. Following these guidelines, this method and the overall architecture open the way to new scenarios on which the robot may support team members in different contexts and environments, even in critical conditions.

Human-Robot Teaming Interaction in a Real Scenario A case study was presented in Chapter 6. In this chapter, an application was designed and developed exploiting the architecture. The scenario aims to face an important problem for telemedicine systems. The team made by a robot and a physician has to collaborate by remote. The application uses modules required for administering therapies by remote and monitoring the patient. The SAAM validation certified the goodness of the system and shown the degree of coupling between the parts.

The *agent programmer* has to code only the part of the system delegated to operate with the patient. This moves the programmer's attention to the HRI part of the system, letting him produce an efficient application for the telemedicine scenario. The fact that the system can adapt autonomously in the environment also reduced a series of problems related to the usage of robots in an unknown environment.

Theoretical Models for Trustworthy Interactions Since social interactions depend on social features, it is important to endow the system with abilities that implement these capabilities. From a theoretical point of view, two models were designed and developed. The first model is in charge of reproducing all the mechanisms that implement a kind of trust between two entities, the trustor and the trustee. By introducing trust theories in the cognitive architecture, it was possible to establish some relevant features for letting a teammate justify its decision or choosing the level of contribution that this last has in the team. As in any regular team, each member can choose if he wants to adopt a task or delegate one. The model, based on the trust theories of Castelfranchi and Falcone, decomposes a goal in a set of sub-tasks and lets the robot find a justification when the system is not able to perform an operation or fulfill an order. The second model is more focused on trust estimation. For a correct teaming interaction, it's important to endow a robot with the ability to estimate a level of trust in its informant. This trust-level helps the robot to understand if the informant is reliable or not. Besides, trust estimation also helps decision-making since some situations need a strong trust in the informant. This last is an ongoing work, and a first theoretical model is given and discussed in Chapter 7. The aim is to endow robots with some vision skills that let them trust someone simply looking for it, involving past experiences and vision capabilities.

Finally, the cognitive architecture aims to build autonomous machines able to work with humans side by side. Future works concern the development of artificial intelligence systems that aim to face challenges in AI, employing new skills, and with the respect of guidelines and codes of conduct to realize ethical robots. For instance, the main interest is to develop models able to learn activities, employing a new learning paradigm based on educational theories, instead of more common learning approaches, such as imitation learning or reinforcement learning.

Acronyms

AC Attention Codelet.

AGI Artificial General Intelligence.

AI Artificial Intelligence.

ANN Artificial Neural Network.

AOP Agent-Oriented Programming.

AOSE Agent-Oriented Software Engineering.

BB Belief Base.

BBN Bayesian Belief Network.

BDI Beliefs-Desires-Intentions.

CSM Current Situational Model.

EgES Engagement Expert System.

ExES Experience Expert System.

GW Global Workspace.

GWT Global Workspace Theory.

HPC High Performance Computing.

HRI Human-Robot Interaction.

HRTI Human-Robot Teaming Interaction.

KB Knowledge Base.

KBS Knowledge-Based System.

MAOP Multi-Agent Oriented Paradigm.

MAPE Monitor, Analyzing, Planning and Executing.

MAS Multi-Agent System.

NAL Non-Axiomatic Logic.

NARS Non-Axiomatic Reasoning System.

PRS Practical Reasoning System.

ROS Robot Operating System.

SAAM Software Architecture Analysis Method.

ToM Theory of Mind.

Bibliography

[1] Sam Adams, Itmar Arel, Joscha Bach, Robert Coop, Rod Furlan, Ben Goertzel, J Storrs Hall, Alexei Samsonovich, Matthias Scheutz, Matthew Schlesinger, et al. Mapping the landscape of human-level artificial general intelligence. *AI magazine*, 33(1):25–42, 2012.

[2] Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen. Face recognition with local binary patterns. In *European conference on computer vision*, pages 469–481. Springer, 2004.

[3] Ramirez-Noriega Alan, Juarez-Ramirez Reyes, Martinez-Ramirez Yobani, Jimenez Samantha, and Inzunza Sergio. Using bayesian networks for knowledge representation and evaluation in intelligent tutoring systems. In *New Advances in Information Systems and Technologies*, pages 169–178. Springer, 2016.

[4] James S Albus, George A Bekey, John H Holland, Nancy G Kanwisher, Jeffrey L Krichmar, Mortimer Mishkin, Dharmendra S Modha, Marcus E Raichle, Gordon M Shepherd, and Giulio Tononi. A proposal for a decade of the mind initiative. *Science*, 317(5843):1321–1321, 2007.

[5] Igor Aleksander. Artificial neuroconsciousness an update. In *International Workshop on Artificial Neural Networks*, pages 566–583. Springer, 1995.

[6] John R Anderson. *The architecture of cognition*. Psychology Press, 2013.

[7] John R Anderson. *Language, memory, and thought*. Psychology Press, 2013.

[8] John R Anderson, Michael Matessa, and Christian Lebiere. Act-r: A theory of higher level cognition and its relation to visual attention. *Human–Computer Interaction*, 12(4):439–462, 1997.

[9] John Robert Anderson. *The architecture of cognition*, volume 5. Psychology Press, 1996.

[10] Jesper Andersson, Luciano Baresi, Nelly Bencomo, Rogério de Lemos, Alessandra Gorla, Paola Inverardi, and Thomas Vogel. Software engineering processes for self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II*, pages 51–75. Springer, 2013.

[11] Grigoris Antoniou and Frank Van Harmelen. Web ontology language: OWL. In *Handbook on ontologies*, pages 67–92. Springer, 2004.

[12] Bernard J Baars. *A cognitive theory of consciousness*. Cambridge University Press, 1993.

[13] Bernard J Baars. In the theatre of consciousness. global workspace theory, a rigorous scientific theory of consciousness. *Journal of Consciousness Studies*, 4(4):292–309, 1997.

[14] Bernard J Baars. The conscious access hypothesis: origins and recent evidence. *Trends in cognitive sciences*, 6(1):47–52, 2002.

[15] Bernard J Baars. Global workspace theory of consciousness: toward a cognitive neuroscience of human experience. *Progress in brain research*, 150:45–53, 2005.

[16] Bernard J Baars. The global workspace theory of consciousness. *The Blackwell companion to consciousness*, pages 236–246, 2007.

[17] Bernard J Baars and Stan Franklin. Consciousness is computational: The lida model of global workspace theory. *International Journal of Machine Consciousness*, 1(01):23–32, 2009.

[18] Muhammad Ali Babar, Liming Zhu, and Ross Jeffery. A framework for classifying and comparing software architecture evaluation methods. In *2004 Australian Software Engineering Conference. Proceedings.*, pages 309–318. IEEE, 2004.

[19] Timea Bagosi, Joachim de Greeff, Koen V Hindriks, and Mark A Neerincx. Designing a knowledge representation interface for cognitive agents. In *Interna-*

tional Workshop on Engineering Multi-Agent Systems, pages 33–50. Springer, 2015.

[20] Tadas Baltrušaitis, Marwa Mahmoud, and Peter Robinson. Cross-dataset learning and person-specific normalisation for automatic action unit detection. In *2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, volume 6, pages 1–6. IEEE, 2015.

[21] Tadas Baltrusaitis, Peter Robinson, and Louis-Philippe Morency. Constrained local neural fields for robust facial landmark detection in the wild. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 354–361, 2013.

[22] Tadas Baltrusaitis, Amir Zadeh, Yao Chong Lim, and Louis-Philippe Morency. Openface 2.0: Facial behavior analysis toolkit. In *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, pages 59–66. IEEE, 2018.

[23] Paul Baxter and Greg Trafton. Cognitive architectures for human-robot interaction. In *2014 9th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 504–505. IEEE, 2014.

[24] Sean Bechhofer. Owl: Web ontology language. In *Encyclopedia of database systems*, pages 2008–2009. Springer, 2009.

[25] Olivier Boissier, Rafael H Bordini, Jomi F Hübner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, 78(6):747–761, 2013.

[26] Rafael H Bordini and Jomi F Hübner. BDI agent programming in agentspeak using jason. In *International Workshop on Computational Logic in Multi-Agent Systems*, pages 143–164. Springer, 2005.

[27] Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. John Wiley &amp; Sons, 2007.

[28] Michael Bratman et al. *Intention, plans, and practical reason*, volume 10. Harvard University Press Cambridge, MA, 1987.

[29] Michael E Bratman. What is intention. *Intentions in communication*, pages 15–32, 1990.

[30] Michael E Bratman. Shared cooperative activity. *The philosophical review*, 101(2):327–341, 1992.

[31] Michael E Bratman, David J Israel, and Martha E Pollack. Plans and resource-bounded practical reasoning. *Computational intelligence*, 4(3):349–355, 1988.

[32] Cynthia Breazeal, Kerstin Dautenhahn, and Takayuki Kanda. Social robotics. In *Springer handbook of robotics*, pages 1935–1972. Springer, 2016.

[33] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[34] Bernd Bruegge and Allen H Dutoit. Object-oriented software engineering: Using UML, Patterns and Java, 2003.

[35] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems*, pages 48–70. Springer, 2009.

[36] Stuartk Card, THOMASP MORAN, and Allen Newell. The model human processor- an engineering model of human performance. *Handbook of perception and human performance.*, 2(45–1), 1986.

[37] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *In AAAI*, 2010.

[38] Christiano Castelfranchi and Rino Falcone. *Trust theory: A socio-cognitive and computational model*, volume 18. John Wiley & Sons, 2010.

[39] Cristiano Castelfranchi, Antonio Chella, Rino Falcone, Francesco Lanza, and Valeria Seidita. Endowing robots with self-modeling abilities for trustful human-robot interactions, 2019.

[40] Cristiano Castelfranchi and Rino Falcone. Delegation conflicts. *Multi-agent rationality*, pages 234–254, 1997.

[41] Cristiano Castelfranchi and Rino Falcone. Towards a theory of delegation for agent-based systems. *Robotics and Autonomous Systems*, 24(3-4):141–157, 1998.

[42] Antonio Chella, Francesco Lanza, Arianna Pipitone, and Valeria Seidita. Knowledge acquisition through introspection in Human-Robot Cooperation. *Biologically Inspired Cognitive Architectures*, 25:1–7, 2018.

[43] Antonio Chella, Francesco Lanza, Arianna Pipitone, and Valeria Seidita. Human-robot teaming: Perspective on analysis and implementation issues. In *CEUR Workshop Proceedings*, volume 2352. CEUR-WS, 2019.

[44] Antonio Chella, Francesco Lanza, and Valeria Seidita. A cognitive architecture for human-robot teaming interaction. In *Proceedings of the 6th International Workshop on Artificial Intelligence and Cognition*, Palermo, July 2-4 2018.

[45] Antonio Chella, Francesco Lanza, and Valeria Seidita. Human-agent interaction, the system level using jason. In *Proceedings of the 6th International Workshop on Engineering Multi-Agent Systems (EMAS 2018)*, Stockholm, July, 10-15 2018.

[46] Antonio Chella, Francesco Lanza, and Valeria Seidita. Representing and developing knowledge using Jason, CArtAgO and OWL. In *CEUR Workshop Proceedings*, volume 2215, pages 147–152, 2018.

[47] Antonio Chella, Francesco Lanza, and Valeria Seidita. Decision Process in Human-Agent Interaction: Extending Jason Reasoning Cycle. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11375 LNAI, pages 320–339. Springer, Cham, 2019.

[48] Antonio Chella and Riccardo Manzotti. Machine consciousness: a manifesto for robotics. *International Journal of Machine Consciousness*, 1(01):33–51, 2009.

[49] Kai Chen, Fangkai Yang, and Xiaoping Chen. Planning with task-oriented knowledge acquisition for a service robot. In *IJCAI*, 2016.

[50] Trevor Cohn, Phil Blunsom, and Sharon Goldwater. Inducing tree-substitution grammars. *Journal of Machine Learning Research*, 11:3053–3096, 2010.

[51] Marios Daoutis. Knowledge based perceptual anchoring - grounding percepts to concepts in cognitive robots. *KI*, 27(2):179–182, 2013.

[52] Marios Daoutis, Silvia Coradeschi, and Amy Loutfi. Cooperative knowledge based perceptual anchoring. *International Journal of Artificial Intelligence Tools*, 21:1250012, 06 2012.

[53] Randall Davis, Howard Shrobe, and Peter Szolovits. What is a knowledge representation? *AI magazine*, 14(1):17–17, 1993.

[54] Rogério De Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer, 2013.

[55] Daniel Clement Dennett. *The intentional stance*. MIT press, 1989.

[56] Liliana Dobrica and Eila Niemela. A survey on software architecture analysis methods. *IEEE Transactions on software Engineering*, 28(7):638–653, 2002.

[57] Wlodzislaw Duch, Richard Jayadi Oentaryo, and Michel Pasquier. Cognitive architectures: Where do we go from here? In *Agi*, volume 171, pages 122–136, 2008.

[58] Peter Eeles. Capturing architectural requirements. *IBM Rational developer works*, 2005.

[59] Paul Ekman. *Telling lies: Clues to deceit in the marketplace, politics, and marriage (revised edition)*. WW Norton & Company, 2009.

[60] Felix Endres, Jürgen Hess, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. 3-d mapping with an rgb-d camera. *IEEE transactions on robotics*, 30(1):177–187, 2013.

[61] Erik H Erikson. *Childhood and Society*. W. W. Norton & Company, 1993.

[62] Ivan Ermolov. Industrial robotics review. In *Robotics: Industry 4.0 Issues & New Intelligent Control Paradigms*, pages 195–204. Springer, 2020.

[63] Rino Falcone and Cristiano Castelfranchi. Levels of delegation and levels of adoption as the basis for adjustable autonomy. In *Congress of the Italian Association for Artificial Intelligence*, pages 273–284. Springer, 1999.

[64] Rino Falcone and Cristiano Castelfranchi. The human in the loop of a delegated agent: The theory of adjustable social autonomy. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 31(5):406–418, 2001.

[65] Rino Falcone and Cristiano Castelfranchi. Social trust: A cognitive approach. In *Trust and deception in virtual societies*, pages 55–90. Springer, 2001.

[66] Rino Falcone and Cristiano Castelfranchi. Trust dynamics: How trust is influenced by direct experiences and by trust itself. In *Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004. Proceedings of the Third International Joint Conference on*, pages 740–747. IEEE, 2004.

[67] Rino Falcone and Cristiano Castelfranchi. Socio-cognitive model of trust. In *Human Computer Interaction: Concepts, Methodologies, Tools, and Applications*, pages 2316–2323. IGI Global, 2009.

[68] C. Ferri-Ramírez, J. Hernández-Orallo, and M.J. Ramírez-Quintana. Incremental learning of functional logic programs. In Herbert Kuchen and Kazunori Ueda, editors, *Functional and Logic Programming*, pages 233–247, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[69] Jan Figat and Włodzimierz Kasprzak. Nao-mark vs qr-code recognition by nao robot vision. In *Progress in Automation, Robotics and Measuring Techniques*, pages 55–64. Springer, 2015.

[70] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[71] Sandro Rama Fiorini and Mara Abel. A review on knowledge-based computer vision. Technical report, Technical Report, UFRGS, Porto Alegre, 2010.

[72] Stan Franklin. A conscious artifact? *Journal of Consciousness Studies*, 10(4-5):47–66, 2003.

[73] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 21–35. Springer, 1996.

[74] Stan Franklin, Arpad Kelemen, and Lee McCauley. Ida: A cognitive agent architecture. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)*, volume 3, pages 2646–2651. IEEE, 1998.

[75] Stan Franklin, Tamas Madl, Sidney D'Mello, and Javier Snaider. Lida: A systems-level architecture for cognition, emotion, and learning. *IEEE Transactions on Autonomous Mental Development*, 6(1):19–41, 2014.

[76] Stan Franklin, Tamas Madl, Steve Strain, Usef Faghihi, Daqi Dong, Sean Kugele, Javier Snaider, Pulin Agrawal, and Sheng Chen. A lida cognitive model tutorial. *Biologically Inspired Cognitive Architectures*, 16:105–130, 2016.

[77] Artur Freitas, Rafael H Bordini, Felipe Meneguzzi, and Renata Vieira. Towards integrating ontologies in multi-agent programming platforms. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2015 IEEE/WIC/ACM International Conference on*, volume 3, pages 225–226. IEEE, 2015.

[78] French. Pseudo-recurrent connectionist networks: An approach to the 'sensitivity-stability' dilemma. *Connection Science*, 9(4):353–380, 1997.

[79] Joaquın M Fuster. Upper processing stages of the perception–action cycle. *Trends in cognitive sciences*, 8(4):143–145, 2004.

[80] Diego Gambetta. Can we trust trust? trust: Making and breaking cooperative relations, department of sociology, university of oxford, 2000.

[81] Epic Games.

[82] J. Gao and C. Zhou. A reasoning system about knowledge acquisition in bi-agent interaction. In *2016 Eighth International Conference on Advanced Computational Intelligence (ICACI)*, pages 412–417, Feb 2016.

[83] Peter Gärdenfors. *Conceptual spaces: The geometry of thought*. MIT press, 2004.

[84] M Georgeff and A Rao. Rational software agents: from theory to practice. In *Agent technology*, pages 139–160. Springer, 1998.

[85] Yolanda Gil and Varun Ratnakar. A comparison of (semantic) markup languages. In *FLAIRS Conference*, pages 413–418, 2002.

[86] Laura Giordano and Francesca Toni. *Knowledge Representation and Nonmonotonic Reasoning*, pages 87–111. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[87] Asunción Gómez-Pérez and Oscar Corcho. Ontology languages for the semantic web. *IEEE Intelligent systems*, 17(1):54–60, 2002.

[88] Melvyn A Goodale, A David Milner, et al. Separate visual pathways for perception and action. In *Human Cognitive Neuropsychology: A Textbook With Readings*. Psychology Press, 1993.

[89] Michael A Goodrich and Alan C Schultz. *Human-robot interaction: a survey*. Now Publishers Inc, 2008.

[90] Anna Gorbenko, Vladimir Popov, and Andrey Sheka. Robot self-awareness: Exploration of internal states. *Applied Mathematical Sciences*, 6(14):675–688, 2012.

[91] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. Owl 2: The next step for owl. *Journal of Web Semantics*, 6(4):309–322, 2008.

[92] Scott A Green, Mark Billinghurst, XiaoQi Chen, and J Geoffrey Chase. Human-robot collaboration: A literature review and augmented reality approach in design. *International journal of advanced robotic systems*, 5(1):1, 2008.

[93] Victoria Groom and Clifford Nass. Can robots be teammates?: Benchmarks in human–robot teams. *Interaction Studies*, 8(3):483–500, 2007.

[94] Sumit Gulwani, José Hernández-Orallo, Emanuel Kitzelmann, Stephen H. Muggleton, Ute Schmid, and Benjamin Zorn. Inductive programming meets the real world. *Commun. ACM*, 58(11):90–99, October 2015.

[95] Patrick Hammer. Adaptive neuro-symbolic network agent. In *International Conference on Artificial General Intelligence*, pages 80–90. Springer, 2019.

[96] Patrick Hammer and Tony Lofthouse. Goal-directed procedure learning. In *International Conference on Artificial General Intelligence*, pages 77–86. Springer, 2018.

[97] Patrick Hammer and Tony Lofthouse. 'opennars for applications': Architecture and control. In *International Conference on Artificial General Intelligence*, pages 193–204. Springer, 2020.

[98] Patrick Hammer, Tony Lofthouse, and Pei Wang. The opennars implementation of the non-axiomatic reasoning system. In *Artificial General Intelligence*, pages 160–170. Springer, 2016.

[99] Mahdi Hannoun, Olivier Boissier, Jaime S Sichman, and Claudette Sayettat. Moise: An organizational model for multi-agent systems. In *Advances in Artificial Intelligence*, pages 156–165. Springer, 2000.

[100] Elliotte Rusty Harold. *Effective XML: 50 specific ways to improve Your XML.* Addison-Wesley Professional, 2004.

[101] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[102] Md Hasanuzzaman, Vuthichai Ampornaramveth, Tao Zhang, MA Bhuiyan, Yoshiaki Shirai, and Haruki Ueno. Real-time vision-based gesture recognition for human robot interaction. In *2004 IEEE International Conference on Robotics and Biomimetics*, pages 413–418. IEEE, 2004.

[103] Benjamin Hirsch, Thomas Konnerth, and Axel Heßler. Merging agents and services—the jiac agent platform. In *Multi-agent programming*, pages 159–185. Springer, 2009.

[104] Hal Hodson. Deepmind and google: the battle to control artificial intelligence, 2019. https://www.economist.com/1843/2019/03/01/deepmind-and-google-the-battle-to-control-artificial-intelligence . Last access: 8th November, 2020.

[105] Guy Hoffman and Cynthia Breazeal. Collaboration in human-robot teams. In *AIAA 1st Intelligent Systems Technical Conference*, page 6434, 2004.

[106] John R Hollenbeck, D Scott DeRue, and Rick Guzzo. Bridging the gap between i/o research and hr practice: Improving team composition, team training, and team task design. *Human Resource Management: Published in Cooperation with the School of Business Administration, The University of Michigan and in alliance with the Society of Human Resources Management*, 43(4):353–366, 2004.

[107] Renata Janošcová. Evaluation of software quality. *IMEA 2012*, page 24, 2012.

[108] Apache Jena. Apache jena. *jena. apache. org [Online]. Available: http://jena. apache. org [Accessed: Mar. 20, 2014]*, page 14, 2013.

[109] Nicholas R Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1(1):7–38, 1998.

[110] Matthew Johnson, Jeffrey M Bradshaw, Paul J Feltovich, Catholijn M Jonker, M Birna Van Riemsdijk, and Maarten Sierhuis. Coactive design: Designing support for interdependence in joint activity. *Journal of Human-Robot Interaction*, 3(1):43–69, 2014.

[111] Dean Jones. Developing shared ontologies in multi-agent systems. In *ECAI'98 Workshop on Intelligent Information Integration*. Citeseer, 1998.

[112] Ivan J Jureta, Alexander Borgida, Neil A Ernst, and John Mylopoulos. The requirements problem for adaptive systems. *ACM Transactions on Management Information Systems (TMIS)*, 5(3):17, 2015.

[113] Daniel S Katz and Raphael R Some. Nasa advances robotic space exploration. *Computer*, 36(1):52–61, 2003.

[114] Rick Kazman, Len Bass, Gregory Abowd, and Mike Webb. Saam: A method for analyzing the properties of software architectures. In *Proceedings of 16th International Conference on Software Engineering*, pages 81–90. IEEE, 1994.

[115] Julian Kerr and Paul Compton. Toward generic model-based object recognition by knowledge acquisition and machine learning. In *In Proceedings of the Workshop on MixedInitiative Intelligent Systems, IJCAI*, pages 107–113. Morgan Kaufmann, 2003.

[116] Thomas Klapiscak and Rafael H Bordini. Jasdl: A practical programming approach combining agent and semantic web technologies. In *International Workshop on Declarative Agent Languages and Technologies*, pages 91–110. Springer, 2008.

[117] Graham Klyne. Resource description framework (rdf): Concepts and abstract syntax. *http://www. w3. org/TR/2004/REC-rdf-concepts-20040210/*, 2004.

[118] Daphne Koller, Nir Friedman, and Francis Bach. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[119] Iuliia Kotseruba and John K Tsotsos. 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artificial Intelligence Review*, 53(1):17–94, 2020.

[120] John E Laird. Extending the soar cognitive architecture. *Frontiers in Artificial Intelligence and Applications*, 171:224, 2008.

[121] John E Laird and Clare Bates Congdon. The soar user's manual version 9.5.0. *The Regents of the University of Michigan*, 2015.

[122] John E Laird, Christian Lebiere, and Paul S Rosenbloom. A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics. *Ai Magazine*, 38(4):13–26, 2017.

[123] John E Laird, Allen Newell, and Paul S Rosenbloom. Soar: An architecture for general intelligence. *Artificial intelligence*, 33(1):1–64, 1987.

[124] John Edwin Laird, Keegan R Kinkade, Shiwali Mohan, and Joseph Z Xu. Cognitive robotics using the soar cognitive architecture. In *CogRob@ AAAI*. Citeseer, 2012.

[125] Francesco Lanza, Valeria Seidita, and Antonio Chella. Agents and robots for collaborating and supporting physicians in healthcare scenarios. *Journal of Biomedical Informatics*, 108:103483, 2020.

[126] Francesco Lanza, Samuele Vinanzi, Valeria Seidita, Angelo Cangelosi, and Antonio Chella. A global workspace theory model for trust estimation in human-robot interaction. In *CEUR Workshop Proceedings*, volume 2483, pages 104–112. CEUR-WS, 2019.

[127] Nico Lassing, Daan Rijsenbrij, and Hv Vliet. The goal of software architecture analysis: Confidence building or risk assessment. In *Proceedings of First BeNeLux conference on software architecture*, pages 47–57, 1999.

[128] Yann LeCun, Fu Jie Huang, and Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'04, pages 97–104, Washington, DC, USA, 2004. IEEE Computer Society.

[129] M.K. Lee and M. Makatchev. How do people talk with a robot? an analysis of human-robot dialogues in the real world. *Conference on Human Factors in Computing Systems - Proceedings*, pages 3769–3774, 2009.

[130] Séverin Lemaignan, Raquel Ros, Lorenz Mösenlechner, Rachid Alami, and Michael Beetz. Oro, a knowledge management platform for cognitive architectures in robotics. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3548–3553. IEEE, 2010.

[131] Douglas B. Lenat. Cyc: A large-scale investment in knowledge infrastructure. *Commun. ACM*, 38(11):33–38, November 1995.

[132] Antonio Lieto, Mehul Bhatt, Alessandro Oltramari, and David Vernon. The role of cognitive architectures in general artificial intelligence, 2018.

[133] Antonio Lieto, Antonio Chella, and Marcello Frixione. Conceptual spaces for cognitive architectures: A lingua franca for different levels of representation. *Biologically inspired cognitive architectures*, 19:1–9, 2017.

[134] Antonio Lieto, Christian Lebiere, and Alessandro Oltramari. The knowledge level in cognitive architectures: Current limitations and possible developments. *Cognitive System Research*, 48:39–55, 2018.

[135] Gi Hyun Lim, Il Hong Suh, and Hyowon Suh. Ontology-based unified robot knowledge for service robots in indoor environments. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 41:492–509, 2011.

[136] Tony Lofthouse. Alann: An event driven control mechanism for a non-axiomatic reasoning system (nars), 2019.

[137] Matteo Luperto, Marta Romeo, Francesca Lunardini, Nicola Basilico, Ray Jones, Angelo Cangelosi, Simona Ferrante, and N Alberto Borghese. Digitalized cognitive assessment mediated by a virtual caregiver. *IJCAI International Joint Conference on Artificial Intelligence*, pages 5841–5843, 2018.

[138] Lanssie Mingyue Ma, Terrence Fong, Mark J Micire, Yun Kyung Kim, and Karen Feigh. Human-robot teaming: Concepts and components for design. In *Field and service robotics*, pages 649–663. Springer, 2018.

[139] Pattie Maes. The Dynamics of Action Selection. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'89, page 991–997, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[140] Patrick Mania and Michael Beetz. A framework for self-training perceptual agents in simulated photorealistic environments. In *International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, 2019.

[141] John McCarthy. Making robots conscious of their mental states, 1995.

[142] Felipe Meneguzzi and Lavindra de Silva. Planning in bdi agents: a survey of the integration of planning algorithms and agent reasoning. *Knowledge Eng. Review*, 30:1–44, 2015.

[143] Felipe Meneguzzi and Michael Luck. Composing high-level plans for declarative agent programming. In *International Workshop on Declarative Agent Languages and Technologies*, pages 69–85. Springer, 2007.

[144] George A Miller. *WordNet: An electronic lexical database*. MIT press, 1998.

[145] George Mois and Jenay M Beer. The role of healthcare robotics in providing support to older adults: a socio-ecological perspective. *Current Geriatrics Reports*, pages 1–8, 2020.

[146] Stephen Muggleton. Scientific knowledge discovery using inductive logic programming. *Commun. ACM*, 42:42–46, 1999.

[147] Kamran Munir and M Sheraz Anjum. The use of ontologies for effective knowledge modelling and information retrieval. *Applied Computing and Informatics*, 14(2):116–126, 2018.

[148] Antonio Natali, Andrea Omicini, and Francesco Zanichelli. Exploiting logic programming in robot applications. In *8th Italian Conference on Logic Programming, GULP'93, Gizzeria, Italy, June 15-18, 1993*, pages 535–548, 1993.

[149] Allen Newell. *Unified theories of cognition*. Harvard University Press, 1994.

[150] Allen Newell, Herbert Alexander Simon, et al. *Human problem solving*, volume 104. Prentice-Hall Englewood Cliffs, NJ, 1972.

[151] Marek Obitko and Vladimír Mařík. Adding owl semantics to ontologies used in multi-agent systems for manufacturing. In *International Conference on Industrial Applications of Holonic and Multi-Agent Systems*, pages 189–200. Springer, 2003.

[152] J. I. Olszewska, M. Barreto, J. Bermejo-Alonso, J. Carbonera, A. Chibani, S. Fiorini, P. Goncalves, M. Habib, A. Khamis, A. Olivares, E. P. de Freitas, E. Prestes, S. V. Ragavan, S. Redfield, R. Sanz, B. Spencer, and H. Li. Ontology for autonomous robotics. In *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 189–194, 2017.

[153] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the a&a meta-model for multi-agent systems. *Autonomous agents and multi-agent systems*, 17(3):432–456, 2008.

[154] Massimiliano Patacchiola and Angelo Cangelosi. A developmental bayesian model of trust in artificial cognitive systems. In *2016 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*, pages 117–123. IEEE, 2016.

[155] Peter F Patel-Schneider. Owl web ontology language semantics and abstract syntax, w3c recommendation. *http://www. w3. org/TR/2004/REC-owl-semantics-20040210/*, 2004.

[156] Anil Patidar and Ugrasen Suman. A survey on software architecture evaluation methods. In *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 967–972. IEEE, 2015.

[157] Arianna Pipitone, Giuseppe Tirone, and Roberto Pirrone. Quasit: A cognitive inspired approach to question answering for the italian language. In *Proceedings of the XV International Conference of the Italian Association for Artificial Intelligence on Advances in Artificial Intelligence - Volume 10037*, AI*IA 2016, pages 464–476, Berlin, Heidelberg, 2016. Springer-Verlag.

[158] J. Pitman and M. Yor. The two-parameter Poisson-Dirichlet distribution derived from a stable subordinator. *Annals of Probability*, 25(2):855–900, 1997.

[159] Gabriella Pizzuto and Angelo Cangelosi. Exploring deep models for comprehension of deictic gesture-word combinations in cognitive robotics. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2019.

[160] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. Jadex: A bdi reasoning engine. In *Multi-agent programming*, pages 149–174. Springer, 2005.

[161] David Poole. Logic programming for robot control. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, pages 150–157, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[162] Edson Prestes, Joel Luis Carbonera, Sandro Rama Fiorini, Vitor AM Jorge, Mara Abel, Raj Madhavan, Angela Locoro, Paulo Goncalves, Marcos E Barreto, Maki Habib, et al. Towards a core ontology for robotics and automation. *Robotics and Autonomous Systems*, 61(11):1193–1204, 2013.

[163] Jorge Peña Queralta, Jussi Taipalmaa, Bilge Can Pullinen, Victor Kathan Sarker, Tuan Nguyen Gia, Hannu Tenhunen, Moncef Gabbouj, Jenni Raitoharju, and Tomi Westerlund. Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision. *IEEE Access*, 8:191617–191643, 2020.

[164] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[165] Nauman A Qureshi, Anna Perini, Neil A Ernst, and John Mylopoulos. Towards a continuous requirements engineering framework for self-adaptive systems. In *2010 First International Workshop on Requirements@ Run. Time*, pages 9–16. IEEE, 2010.

[166] Uma Ramamurthy, Bernard J. Baars, D'Mello S. K., and Stan Franklin. Lida: A working model of cognition, 2006.

[167] Anand S Rao. Agentspeak (l): BDI agents speak out in a logical computable language. In *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 42–55. Springer, 1996.

[168] Anand S Rao, Michael P Georgeff, et al. BDI agents: from theory to practice. In *ICMAS*, volume 95, pages 312–319, 1995.

[169] Raj Reddy. Robotics and intelligent systems in support of society. *IEEE Intelligent Systems*, 21(3):24–31, 2006.

[170] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[171] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Cartago: A framework for prototyping artifact-based environments in mas. In *International Workshop on Environments for Multi-Agent Systems*, pages 67–86. Springer, 2006.

[172] Laurel D Riek. Healthcare robotics. *Communications of the ACM*, 60(11):68–78, 2017.

[173] Frank E Ritter, Farnaz Tehranchi, and Jacob D Oury. Act-r: A cognitive architecture for modeling cognition. *Wiley Interdisciplinary Reviews: Cognitive Science*, 10(3):e1488, 2019.

[174] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7:123–146, 1995.

[175] Rasmus Rothe, Radu Timofte, and Luc Van Gool. Dex: Deep expectation of apparent age from a single image. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 10–15, 2015.

[176] Rasmus Rothe, Radu Timofte, and Luc Van Gool. Deep expectation of real and apparent age from a single image without facial landmarks. *International Journal of Computer Vision*, 126(2-4):144–157, 2018.

[177] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009.

[178] Paul Ruvolo and Eric Eaton. ELLA: An efficient lifelong learning algorithm. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 507–515, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[179] Luca Sabatucci, Valeria Seidita, and Massimo Cossentino. The four types of self-adaptive systems: A metamodel. In *International Conference on Intelligent Interactive Multimedia Systems and Services*, pages 440–450. Springer, 2017.

[180] Eduardo Salas, Nancy J Cooke, and Michael A Rosen. On teams, teamwork, and team performance: Discoveries and developments. *Human factors*, 50(3):540–547, 2008.

[181] Alexei V Samsonovich. Toward a unified catalog of implemented cognitive architectures. *BICA*, 221(2010):195–244, 2010.

[182] Brian Scassellati. Theory of mind for a humanoid robot. *Autonomous Robots*, 12(1):13–24, 2002.

[183] V Seidita, C Diliberto, P Zanardi, A Chella, and F Lanza. Inside the robot's mind during human-robot interaction. In *7th International Workshop on Artificial Intelligence and Cognition, AIC 2019*, volume 2483, pages 54–67. CEUR-WS, 2019.

[184] Valeria Seidita and Massimo Cossentino. From modeling to implementing the perception loop in self-conscious systems. *International Journal of Machine Consciousness*, 2(02):289–306, 2010.

[185] Yoav Shoham. Agent-oriented programming. *Artificial intelligence*, 60(1):51–92, 1993.

[186] Javier Snaider, Ryan McCall, and Stan Franklin. The lida framework as a general tool for agi. In *International Conference on Artificial General Intelligence*, pages 133–142. Springer, 2011.

[187] Rudi Studer, V Richard Benjamins, and Dieter Fensel. Knowledge engineering: principles and methods. *Data & knowledge engineering*, 25(1-2):161–197, 1998.

[188] Budhitama Subagdja, Han Yi Tay, and Ah-Hwee Tan. Who am i?: Towards social self-awareness for intelligent agents. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, 2020.

[189] Ron Sun. The importance of cognitive architectures: An analysis based on clarion. *Journal of Experimental &amp; Theoretical Artificial Intelligence*, 19(2):159–193, 2007.

[190] Moritz Tenorth and Michael Beetz. KnowRob – A Knowledge Processing Infrastructure for Cognition-enabled Robots. *Int. Journal of Robotics Research*, 32(5):566 – 590, April 2013.

[191] Grace Teo, Ryan Wohleber, Jinchao Lin, and Lauren Reinerman-Jones. The relevance of theory to human-robot teaming research and development. In *Advances in Human Factors in Robots and Unmanned Systems*, pages 175–185. Springer, 2017.

[192] Kristinn R Thórisson. Gandalf: An embodied humanoid capable of real-time multimodal dialogue with people. In *Agents*, pages 536–537, 1997.

[193] Jim Torresen. A review of future and ethical perspectives of robotics and ai. *Frontiers in Robotics and AI*, 4:75, 2018.

[194] Endel Tulving et al. Episodic and semantic memory. *Organization of memory*, 1:381–403, 1972.

[195] Kimberly E Vanderbilt, David Liu, and Gail D Heyman. The development of distrust. *Child development*, 82(5):1372–1380, 2011.

[196] David Vernon. *Artificial cognitive systems: A primer.* MIT Press, 2014.

[197] David Vernon, Giorgio Metta, and Giulio Sandini. A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. *IEEE transactions on evolutionary computation*, 11(2):151–180, 2007.

[198] Federico Vicentini. Collaborative robotics: a survey. *Journal of Mechanical Design*, pages 1–29, 2020.

[199] Samuele Vinanzi, Massimiliano Patacchiola, Antonio Chella, and Angelo Cangelosi. Would a robot trust you? developmental robotics model of trust and theory of mind. *Philosophical Transactions of the Royal Society B*, 374(1771):20180032, 2019.

[200] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[201] Alessandro Vuono, Matteo Luperto, Jacopo Banfi, Nicola Basilico, Nunzio A Borghese, Michael Sioutis, Jennifer Renoux, and Amy Loufti. Seeking prevention of cognitive decline in elders via activity suggestion by a virtual caregiver. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1835–1837. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[202] Pei Wang. *Non-axiomatic reasoning system: Exploring the essence of intelligence*. Citeseer, 1995.

[203] Pei Wang. *Rigid flexibility: The logic of intelligence*, volume 34. Springer Science & Business Media, 2006.

[204] Pei Wang. The logic of intelligence. In *Artificial general intelligence*, pages 31–62. Springer, 2007.

[205] Pei Wang. Insufficient knowledge and resources-a biological constraint and its functional implications. In *AAAI Fall Symposium: Biologically Inspired Cognitive Architectures*. Citeseer, 2009.

[206] Pei Wang. Non-axiomatic logic (nal) specification, 09 2010.

[207] Pei Wang. *Non-axiomatic logic: A model of intelligent reasoning*. World Scientific, 2013.

[208] Pei Wang, Xiang Li, and Patrick Hammer. Self in nars, an agi system. *Frontiers in Robotics and AI*, 5:20, 2018.

[209] Changyun Wei and Koen V Hindriks. An agent-based cognitive robot architecture. In *International Workshop on Programming Multi-Agent Systems*, pages 54–71. Springer, 2012.

[210] Bob J Wielinga, A Th Schreiber, and Jost A Breuker. Kads: A modelling approach to knowledge engineering. *Knowledge acquisition*, 4(1):5–53, 1992.

[211] Michael Winikoff. Jack™ intelligent agents: an industrial strength platform. In *Multi-Agent Programming*, pages 175–193. Springer, 2005.

[212] William E. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Bureau of the Census, 1999.

[213] Erroll Wood, Tadas Baltrusaitis, Xucong Zhang, Yusuke Sugano, Peter Robinson, and Andreas Bulling. Rendering of eyes for eye-shape registration and gaze estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3756–3764, 2015.

[214] M. Wooldridge and P. Ciancarini. Agent-Oriented Software Engineering: The State of the Art. *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000, Limerick, Ireland, June 10, 2000: Revised Papers*, 2001.

[215] Michael Wooldridge. *Reasoning about rational agents*. MIT press, 2003.

[216] Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2009.

[217] Michael J Wooldridge and Nicholas R Jennings. Intelligent agents: Theory and practice. *The knowledge engineering review*, 10(2):115–152, 1995.

[218] World Health Organization. Healthy, prosperous lives for all: the European Health Equity Status Report, 2019. https://www.euro.who.int/en/publications/abstracts/health-equity-status-report-2019 . Last Accessed: 2020, 13 November.

[219] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32Nd Annual Meeting on Association for Computational Lin-*

guistics, ACL ’94, pages 133–138, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics.

[220] Holly A Yanco and Jill Drury. Classifying human-robot interaction: an updated taxonomy. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, volume 3, pages 2841–2846. IEEE, 2004.

[221] Amir Zadeh, Yao Chong Lim, Tadas Baltrusaitis, and Louis-Philippe Morency. Convolutional experts constrained local model for 3d facial landmark detection. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 2519–2528, 2017.