# Plan Acquisition Through Intentional Learning in BDI Multi-Agent Systems

A thesis submitted for the degree of

***Doctor of Philosophy***

at the

Department of Computer Science and Electronic Engineering

University of Essex

by

**Wulfrano Arturo Luna Ramírez**

**2018**

# Contents

# Acknowledgements

To **Perfecta and Wulfrano**, my beloved parents who are present at every time in my life, with eternal gratitude. I owe all to you, my two suns. The whispers of the wind, the sound of the trees, and the blinking stars, all comes from you and drive me where you are. This is a small memorial of your strong but sweet voices and wise words, many thanks for all your teachings. I am grateful to my siblings **María**, **Toño** and **Francisco**, who have provided me through inspiration and encouragement.

To **Esther** with love and gratitude, for the infinite hours of traversing across the sea, for your amazing smile and your loving heart. This step would be impossible without you. Let's see the rays, the hope and the miracles again. I cannot forgot about **Lugo** family, who gave me the shelter when I come back to my city. I am also grateful with all my family members and friends who have supported me along my life.

I wish to express with heartfelt sincerity my gratitude to my supervisor **Maria Fasli**, who has insightfully guided me all along this endeavour, I thank for her recommendations, suggestions and advices to go further in my PhD and to undertake research as a pleasant challenge. I learnt to love agents even more under her mentoring. I am specially thankful to **Carlos Jaimez**, who introduce me to my supervisor in Teotihuacan Mexico and for being so supportive during my studies.

A profound gratitude goes to **John Clark**, **Miguel Capllonch** and **Duncan Bounds**, who help me in reading the manuscript and making very relevant suggestions for the final version, and specially to **Emmanuel Ferreyra** who made an intercontinental printing possible!

I am very thankful to my sensei and my friends from **Akari Shotokan Karate-Do** and the **Osos Grises** from the **PDMU**, who gave me the support and made me strong enough to face the hurricanes of the unknown. **Federico**, **Franciso** and **Salvador**, **Jorge** and **Araceli**... I cannot mention them all but they are here together.

I would specially like to express my gratitude to my lecturers from my master degree in Universidad Veracruzana, who showed me the path and open the door to the marvelous world of Artificial Intelligence. I thank **Alejandro Guerra** for his teachings and for inspiring me to continue with my studies. I thank **Gustavo Ortiz** as well for his smart comments on my work.

With special mention to **Manuel Martínez** and **Mirna Valdés**, who enlighted me with the power of the word and offered me a earnest friendship filled with smartness and beauty. I also like to mention **Angélica García** and the omnipresent **José Negrete** for reawekening the seed of the curiosity, and **Christian Lemaître** for his sincere endorsement and wise recommendations, he gave me the opportunity to see through the window of the academia.

I am grateful with the staff from University of Essex, particularly **Edmundo Hernández**, **Claire Harvey**, **Rachel Gooch**, **Nicole Smither**, **Emma McClelland** and **Shaaba Lotun**. I also express my sincere gratitude to colleagues and staff from Universidad Autónoma Metropolitana-Cuajimalpa: **Héctor Jiménez**, **Alfredo Mateos**, **Carlos Rivero**, **Judith Llamas** and **Patricia Flores**. To all of them for their unfailing support and assistance.

I am also grateful to the people who accompanied and help me along the PhD journey: **Anasol**, **Paloma**, **Angeliki**, **Jabir**, **Norah**, **Thabat**, **Sameera**, **Asia**, the **Essex Writers Circle** and the **Essex Zapatista Solidarity Group**, I will miss you all.

I thank the o the project no. 48510384 of SEP-DSA/UAM-169, for its sponsorship to my PhD.

And finally, last but by no means least, also to my reviewers **Dr. Samuel Steel** and **Dr. Peter Barbrook-Johnson** for their insightful comments and clever suggestions to this thesis.

# Abstract

## Plan Acquisition Through Intentional Learning in BDI Multi-Agent Systems

by *Wulfrano Arturo Luna Ramírez*

Doctor of Philosophy in Computer Science

University of Essex, United Kingdom, 2018

*Professor MARIA FASLI, Supervisor*

Multi-Agent Systems (MAS), a technique emanating from Distributed Artificial Intelligence, is a suitable technique to study complex systems. They make it possible to represent and simulate both elements and interrelations of systems in a variety of domains. The most commonly used approach to develop the individual components (agents) within MAS is reactive agency. However, other architectures, like cognitive agents, enable richer behaviours and interactions to be captured and modelled. The well-known Belief-Desire-Intentions architecture (BDI) is a robust approach to develop cognitive agents and it can emulate aspects of autonomous behaviour and is thus a promising tool to simulate social systems.

Machine Learning has been applied to improve the behaviour of agents both individually or collectively. However, the original BDI model of agency, is lacking learning as part of its core functionalities. To cope with learning, the BDI agency has been extended by Intentional Learning (IL) operating at three levels: belief adjustment, plan selection, and plan acquisition. The latter makes it possible to increase the agent's catalogue of skills by generating new procedural knowledge to be used onwards.

The main contributions of this thesis are: a) the development of IL in a fully-fledged BDI framework at the plan acquisition level, b) extending IL from the single-agent case to the collective perspective; and c) a novel framework that melts reactive and BDI agents through integrating both MAS and Agent-Based Modelling approaches, it allows the configuration of diverse domains and environments. Learning is demonstrated in a test-bed environment to acquire a set of plans that drive the agent to exhibit behaviours such as target-searching and left-handed wall-following. Learning in both decision strata, single and collective, is tested in a more challenging and socially relevant environment: the Disaster-Rescue problem.

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Overview

Imagine the vast quantity of living beings present in a rain forest, from microorganisms to large mammals, from fungi to ceiba trees, all them playing specific roles and holding innumerable relations. Envisage an inextricable telecommunications network composed by thousands of phone lines, terminal nodes and links that make it possible to interconnect two people or organisations across the world. Recall the tangled grid of crossing of roads, avenues and streets used by hundreds or thousands of vehicles, cyclists and pedestrians struggling to reach their destinations. Considering all of these, reveals that these are instances of systems that have in common complexity and whose constituent components are a variety of entities with multiple interrelations which may be hard to analyse, reproduce and understand.

Complexity and complex systems have attracted the interest of many disciplines ranging from Natural Sciences to Humanities. Social and natural systems (living and non-living) as in the aforementioned, have inspired many artificial models driving the development of methods and theories devoted to their comprehension: Non-linear Dynamic Systems, Complex Adaptive Systems, Chaos Theory, Synergetic, and many others [32, 33, 41].

Computer Science and Artificial Intelligence, particularly in the field of Multi-Agent Systems (MAS), have become especially concerned with studying complex systems. Computational simulations make it possible to represent complex systems and come closer to their understanding. Furthermore, MAS offer a suitable approach for studying complexity as they make possible to represent a given system comprising its elements and relations, both components of complexity. In addition, MAS are complex systems on their own and even their development process exhibits complexity [32, 33].

An agent can be considered as an entity acting for the sake of a given purpose, interacting with the environment by means of perceptions and actions [61, 94]. Agents have been conceived under a range of architectures, reactive and cognitive being the most consolidated of them. Moreover, a MAS can be understood as an ensemble of agents that interact with each other sharing resources and goals, and are situated in the same environment. One notable feature present in MAS is the capability of agents to improve their behaviour through learning. Many algorithms and approaches have been developed at the agent level with different representations and learning techniques. Learning efforts in the collective level have been approached from single-agent to multi-agent perspectives that includes the joint action of multiple learners.

Additionally, the studies on complexity point out that it is associated with two key phenomena: Self-organisation (SO) and Emergence (EM) [33,44], both are concepts used in a wide range of disciplines including MAS, where they suppose an improvement in performance and capability of problem solving, carrying on a powerful design approach [33, 113], i.e. the system could be able to adapt and reorder its design purposes when facing environmental changes, and bring about outcomes useful in fulfilling its tasks like coordination and cooperation [36, 124].

SO can be viewed as the mechanism or process that allows a system to change its organisation (i.e. statistical complexity, structure, states, etc.) without any external explicit command during its execution time [47, 102]. On the other hand, EM can be conceptualized as the existence of a global behaviour that is novel with respect to the constituent parts of the system [80, 81, 113]. Both are dynamic and robust processes that occur in the lifetime of a system, they commonly appear together, but it could be possible to observe them in a separated way [32]. Besides, learning could be assumed as a useful method in the generation of behaviours based on the activities and relationships of the components of the MAS, namely the agents and the environment. EM is a process arising at the lower level of a system that can lead to different global behaviours (including Self-organisation), it is important to notice the relation that can be established between the generation of abilities and, consequently, behaviour at the agent level through a process of learning, and the impact on the global system, i.e. the behaviour of the system as a whole. On top of that, most of the works to include learning in MAS are mainly directed to reactive agents whereas there is a lack of approaches focused on cognitive agents particularly in the case of learning from a collective perspective [8,56,90,117]. In addition, cognitive agents based on mentalistic notions like the Belief-Desire-Intentions architecture, whose original definition does not comprise learning capabilities, offer an avenue to explore the learning possibilities from either single and collective perspectives exploiting their robustness and adaptability, as they offer great flexibility and a powerful deliberation cycle to react accordingly with the environmental changes.

## 1.2 Motivation

Multi-agent Systems (MAS) have been proposed as a suitable approach to the study of complex systems. Considering the relation of SO and EM with complexity, the development of MAS has moved towards their representation in the system, mainly based on [5, 33, 34, 102, 118]: Genetic Algorithms, Artificial Neural Networks, Swarm Intelligence, Stigmergy, or for cooperative purposes, through lists of predefined Non-Cooperative Situations to easy their identification by agents. Furthermore, the development cycle has been adapted to take into account both phenomena at design time. One common way to observe SO in MAS is through coordination, i.e. the joint action of agents. Despite some works have reached emergent coordination behaviour not necessarily by using learning, most approaches have used multi-agent learning, mainly based on the sound approach of Reinforcement Learning [8, 21, 78, 120], with the purpose of coordinating actions for the sake of reaching SO [21,52,53,64,122].

Even though cognitive agents have higher capabilities in problem solving and complex behaviour, like intentional architectures, most of the efforts to include SO and EM have been carried out with reactive agents. Only a few of them have focused on cognitive intentional agents, but still remain in a grid-world like environments performing test-bed tasks.

As a summary, the more important literature findings that motivate this thesis are listed as follows.

a) Global emergent behaviours have been generated within MAS composed of reactive agents (including or not learning mechanisms) [5,32,33];

b) Architectures of cognitive agents exhibit a richer set of behaviours and more capabilities than reactive agents [61];

c) Among cognitive agents, the intentional architecture known as BDI, which is based on the mentalistic notions of Beliefs, Desires and Intentions, is a well-established way to simulate complex behaviours [93];

d) Emergent behaviours can be obtained in a MAS composed by BDI-based cognitive agents. There are three types of emergence associated to the components of the BDI agency [80]:

   – Belief-related: knowledge distribution or grouping;

   – Goal-related: cooperation, task distribution;

   – Plan-related: reinforcement of best plans, improved task processing.

e) Extending the BDI notion of agency to include learning has been undertaken using diverse learning techniques, particularly reinforcement and inductive learning [11,90], being Intentional Learning (either inductive or abductive) the most suitable framework for this purpose [50,56,117].

f) Intentional Learning can be applied in the level of action selection, goal driving and plan generation [56, 90,117]. The latter is the only one closely related with the acquisition of new skills and consequently with the arising of emergent outcomes.

g) Intentional Learning focused on action selection and goal driving has been pursued at the MAS level [56, 90], although this is not the case of plan acquisition.

g) Acquire new plans of actions focused only on the single-agent level has been pursued in BDI-like architectures using test-bed environments [115–117].

Therefore, this work focuses upon: the development of a learning mechanism that allows intentional agents in a MAS to acquire new behaviours oriented towards problem solving, in a fully-fledged BDI environment (i.e. an environment that makes use of the BDI constructs like beliefs, plans and goals [16,20]), under a logic-based knowledge representation; executing a reasoning loop, where the agent behaviour is generated by plans executed accordingly with agent goals and beliefs, regarding world state of affairs alongside the agent's own internal states. Through the learning process, the BDI agents must acquire new skills (a new set of plans), driving the whole MAS to perform its global task.

## 1.3 Research Question and Objectives

On top of the above, the main question to answer with this research is the following:

**RQ.** How intentional learning, as plan acquisition, can be conceptualised and implemented in a fully-fledged BDI agent architecture and framework?

### 1.3.1 Objectives

To answer the Research Question above (1.3), the following objectives are set to point out some sub-problems to drive the research.

1. **Exploring the behaviour acquisition in the BDI agency**

   Learning in the intentional architecture focused on plan acquisition implies considering the type of behaviours that can be obtained within a full BDI framework. It is highly relevant to determine the cases where the learning is applicable and how the process should be fired, whilst keeping the reasoning cycle operating accordingly with the BDI theory. Furthermore, there is a need to determine how the learning mechanism should be controlled.

2. **Developing a BDI MAS considering the concepts of EM and SO in its design**

   The agents within the BDI MAS are intended to be capable of plan acquisition while the whole system fulfils its global task. Derived from the concepts of emergence and self-organisation, the design of a BDI MAS that accomplishes a global objective composed of a set of high-level tasks, and within a changing environment, must account for the following conditions:

   a) The agents that compose the MAS are heterogeneous, as they have different types of capabilities, alongside partial knowledge of both other agents and the environment.

   b) In the low-level (individual agents), there is no representation of the fulfilment of the global objective. Furthermore, when concerning the global objective or each higher-level task, the capabilities of each agent are not enough to perform it individually or fulfil it.

   c) The processing of each higher-level task is performed by subsets of agents.

   d) There is no centralized control mechanism (external or internal) pre-defined in the BDI MAS.

e) The global behaviour of the BDI MAS must fulfil the global objective.

3. **Designing a collective intentional learning mechanism**

   The learning mechanism is intended to manage the information of the BDI agents and the environment to generate new agent skills. The mechanism needs account for the heterogeneity of the agents and their representation of beliefs, goals and plans, as a basis to promote emergent outcomes. A set of parameters can be established based on the individual BDI outcomes, the influence between groups of agents and the analysis of the environmental information. The different kind of data generated during the operation of MAS (individual, collective, environmental) must be analysed to identify: a) the suitable representation of the information in the BDI architecture; b) the changes in the environment that result in failure behaviour; so they may c) establish the relationship between agents (their communications, which affects the stability of the system), and their impact on the structure/hierarchy of the MAS. Finally, those elements should be integrated to develop an intentional learning mechanism, that includes the influence of other agents in the generation of new skills in the BDI architecture.

## 1.4 Methodology

In order to answer the research question and fulfil the research objectives, this research is driven through the following stages:

1. The development of a BDI MAS (without learning) applied to the domain described below in Section 1.4.1.

2. Exploring the learning possibilities of plan generation of a fully-fledged BDI environment in a single-agent context.

3. Developing a learning mechanism at the plan generation level to achieve problem solving (whilst factoring in environment changes) in the BDI MAS considering the environmental changes. Plan updating and plan acquisition are two processes to explore in learning; they are focused on altering the agent behaviour to achieve problem solving capabilities.

4. The development of the extended intentional learning mechanism at the plan acquisition level, to account for the influence of other agents, and changes within the environment, when generating new plans.

5. Testing both BDI MAS (with single-agent and collective learning) in the domain of application, and analysing their results.

### 1.4.1 Domain of Application

Whether natural or artificial systems, agents and MAS have been used to simulate a variety of them, ranging from networks to human societies [67], even giving birth to a specialised sub-field known as Agent-Based Modelling (ABM) where the focus is on the agent's actions and relations between agents and the environment, and their impact on the global system. Despite researchers' preference for using test-bed domains to develop their experimentation, some real world simulated domains have garnered the attention of MAS and ABM communities. One example of that is the simulation of the Disaster-Rescue domain. It is rooted in the emergency management endeavours from government and rescue organisations, and is an active field with abundant research and publications including the well known RoboCup contest [66,89,104,112] simulations and applications of rescue staff and public agencies [40], due to the challenging dynamic of the simulated situations and its noteworthy social relevance. ABM simulation tools are helpful in evaluating, planning and predicting real-life situations. Since they provide an in silico experimental scenario for educational, training, understanding and decision support, they can be applied in emergence management, mainly in activities to improve response actions and define actual actions required to tackle an emergency, like fire extinguishing or street cleansing [55]. ABM simulations provide relevant contributions [55]:

- Planning: determining the impact of a disaster event identifying the ways to be prepared and how to respond.
- Vulnerability analysis: evaluation of the strategies of emergency response.
- Identification and detection: determining the occurrence possibility of a disaster.
- Training: training emergency services staff by simulation.

- Real-time response support: determining alternative response strategies given an emergency situation.

Finally, this domain of application is an excellent test-bed to MAS, where policies, behaviours and algorithms of coordination, task allocation, and coalition formation are continuously evaluated. One of the most known benchmarks for rescue teams' implementation can be found in the RoboCup Rescue Multi-Agent Simulation (RBCR), among others as can be seen in Appendix C.

## 1.5 Contributions

Learning to acquire new behaviours in BDI MAS has mainly been pursued at the level of plan applicability and goal adjustment, regarding the case of single-agent learners. Therefore, this work is concerned with collective learning in the context of cognitive agents and MAS, specifically using the BDI architecture. As a result of this research, the state of the art would be progressed in the following aspects:

1. Including Intentional Learning at the level of plan generation in a fully-fledged BDI environment, to acquire new skills in agents. Learning is tested in single-agent test-beds, aimed at achieving behaviours composed from a set of primitive actions.

2. Built upon the behaviour acquisition in the single-agent case, to then extend the BDI Intentional Learning settings from individual to collective. It implies the design of a BDI-learning method that includes a representation of the interactions and influence of other agents, to acquire new capabilities through the generation of new plans.

3. Developing a BDI MAS applied to the Rescue Domain. This represent a fourfold contribution:

    - Providing a socially relevant tool to perform realistic simulation of disaster situations that could be used by civilian security or rescue groups, allowing the inference of disaster management policies.
    - Progress the state of the art MAS in the Rescue Domain to include BDI agents (that represent some aspect of human decision making processes) enhanced with learning capabilities.
    - From the view of engineering BDI MAS, learning plans focused upon problem solving could ease the development of Agents/MAS by avoiding the need to design a detailed solution and

centralized control. Moreover, it helps to handle situations where it is not possible to anticipate the whole dynamics of the MAS and the environment.

- Bridging the gap between Agent-Based Modelling and Multi-Agent Systems, by combining the agent-centredness of the Multi-Agent approach to solve a problem (where the reasoning capabilities of the agent are emphasized as a central part of the solution), with the more system-centredness approach of Agent-Based Modellings, where the phenomena of Emergence and Self-organisation are commonly accounted for when profiling solutions. In this work, the framework of experimentation interconnects two major development environments in these fields. A BDI framework in charge of the deliberation processes, and a reactive-based MAS programming environment to represent the elements of the Disaster Rescue Domain.

## 1.6 Outline

This document is organised as follows. In Chapter 2, entitled Literature Review, a brief discussion (the features and importance) of the concepts of Agents and Multi-Agents Systems, Machine Learning, and Planning, is provided. In Chapter 3 a problem definition of intentional learning and its integration within the BDI reasoning cycle, are presented. The framework of experimentation is described, and a simulation developed within it, applied to the Disaster-Rescue domain, are introduced. Chapter 4, describes Intentional Learning in a fully-fledged BDI environment, and results of its implementation in the experimentation framework to acquire new skills in two domains of application. Chapter 5, describes the proposed extension to Intentional Learning, namely, Collective Intentional Learning, where experiments are presented and discussed. and Chapter 6, discusses the contributions and the main findings obtained in the present thesis.

# Chapter 2

# Literature Review

This Chapter is focused on providing the required background to understand the entire research. It also aims to progress towards the fulfilment of the Research Objective 1 “Exploring the behaviour acquisition in the BDI agency” by examining the possibilities of BDI agency to increase its repertoire of skills for the sake of behaviour acquisition and the associated techniques that have been reported in the literature. Additionally, the review advances the fulfilment of Research Objective 2 “Developing a BDI MAS considering the concepts of EM and SO in its design”, by presenting information about the development of Multi-Agent Systems and two key phenomena associated to complexity.

In the following, a review of the concepts of Multi-Agent Systems, Machine Learning, Planning and their relation with Self-organisation and Emergence is given as it has been published in literature from the field. This is sourced from numerous research groups, projects and conferences, as well as international work and seminars.

## 2.1 Agent and Multi-Agent Systems

Complexity is the constant when talking about software system development, mainly due to a) the nature of the problem to solve; b) the large number of its components; c) the architecture of its components; and, d) the intensive relations sustained by components with each other. Additionally, the designed and implemented system has to overcome its inclusion in a real world where there are many environmental pressures and changes. Therefore, many approaches of design and methodologies have been proposed, all of them focused on affording complexity through a set of tools of abstraction and their own techniques and formalisms [9, 28, 33, 62, 67, 94, 134].

A brief definition of Agent is the following [33, 38, 134]:

**Def. 2.1** A computer system that is situated in some environment, and that is capable of autonomous interaction with it, in order to meet its design objectives.

Derived from the Def. 2.1 some features can be noted, as the autonomy, or degree of self-control, the agents decisions and actions have; reactiveness, regarding the capability of perceiving environmental changes and to change their behaviour accordingly; and social ability, the communication capability commonly based in well-defined protocols.

Other additional features attributable to agents are [135]:

- Proactiveness. Concerned with goal-driven persistent behaviour.
- Veracity. From its own knowledge, to communicate true information only.
- Mobility. Being able to change its location, v.gr. move through a network.
- Benevolence. Avoiding conflicting goals to accomplish its sole purpose of design.

Furthermore, reactive agents (i.e. an agent fulfilling the aforementioned features of autonomy and social ability and focused mainly on reacting to its environment through interactions) are defined under the weak notion of agency [135]. Such an agent can be implemented as a set of action-condition rules, commonly called productions, leading to reflex behaviour [33, 94]. This kind of agent has been useful in several applications due to some of its advantages, namely being easy to design, predict and implement, and not using too many computing resources, among others.

In contrast, one approach for conceiving and designing agents has been developed which accounts for the intentional stance [135]. In brief, when a system is not known entirely beforehand, it is better to use the power of an abstraction tool to properly analyse and understand it [135]. Therefore, some researchers have attempted to work under a strong notion of agency imbued with a deeper sense of intentional stance, using mentalistic notions such as knowledge, beliefs, intentions and obligations to define the agent, but mainly to include the feature of rationality (to act for the fulfilment of its goals) [135].

With this in mind, the following advantages of cognitive agents typically listed in the literature are [61]:

- Declarative (high-level) communication.
- Flexibility in problem solving within an environment with partial control and partial observability.
- Context-dependent decision making due to the interactions with the environment.

A Multi-Agent System (MAS), can be defined as [33, 38, 134]:

**Def. 2.2** A group of agents (homogeneous or heterogeneous) which interact with each other (in a cooperative or competitive manner) within a particular environment and share resources, stimuli and perhaps a common objective.

Some considerations must be given based on Def. 2.2: The terms homogeneity/heterogeneity are concerned with a) the similarity or difference in the architecture of agents (reactive, cognitive, etc.); b) their skills or abilities; and c) final purpose or objectives. Interactions are fundamental to the behaviour of an agent. Consequently, in a complex system interactions that cannot be anticipated at the design time should be managed at run time. They can be direct (messaging) or indirect (through the environment). Agents can be looking for equal or different purposes, in a competitive or cooperative way (i.e. while pursuing different, even contrasting individual goals, they can reach the same objective as a whole system [5,36]). In addition, as highlighted previously, both the abstraction level and the agent features offered by MAS represent a feasible, robust way of studying and modelling complex systems.

Consequently, reactive and cognitive agents have been used in many applications including network resource management, adaptive routing, ubiquitous computing, and mobile applications, amongst others [28, 33, 47, 67].

### 2.1.1 The BDI Agency

To implement intentional agents, the well-known Belief-Desire-Intention (BDI) architecture [93] based on Beliefs (the current status of an agent in a certain time) Desires (its goals or states to achieve) and Intentions (the set of actions to perform in order to accomplish its goals) has been developed [38, 93, 134].

In addition, BDI agents store a queue of events coming from the environment or from their own reasoning operations. The reasoning cycle of an agent under this architecture starts with updating the list of events (internal or external). Then, event selection is performed and a set of plans are chosen accordingly, and subsequently one plan is selected to create an intention. Finally, an intention selection function determines the actual course of action to be carried out, i.e. the execution of the actions defined in the plan.

Some languages have evolved in order to implement such agents [12,19,28,67]. BDI architecture has been implemented following different formalisms and using several programming languages, including those based on object-oriented or list-processing paradigms [15, 28, 95]. However, these works commonly lack of a direct representation of some of the intentional attitudes of the BDI architecture like beliefs and desires. This situation prevents for pursuing a straightforward implementation, and makes it difficult to exploit some benefits of the BDI architecture like its abstractions tools.

On the other hand, the BDI has been formalized through a family of logics that describes the BDI operational semantics and the central elements of the architecture, namely beliefs, desires and intentions [28, 38, 134]. Consequently, some approaches have focused on the development of specific frameworks and languages to implement higher level constructs belonging to the BDI agency.

One of the earlier of such systems is the Procedural Reasoning System (PRS) [38, 134]. According to the PRS, an agent has at its disposal a library of manually programmed plans. Those plans are composed by these three elements [18]:

- Goal: the post-condition of the plan. It defines what the plan is for, i.e. what the plan is actually achieving.
- Context: the pre-condition of the plan. Determines the situation where the plan can be applied.
- Body: a sequence of actions to be executed. Each element of the plan's body can be either a primitive action or a goal, i.e. another plan can be triggered in the body of any plan.

Based on the PRS, in [93] AgentSpeak, a programming language especially created to the development of BDI agents was introduced. Furthermore, Jason [16–18], an AgentSpeak interpreter is one of the most solid development environments of BDI MAS. In Section 3.1 of Chapter 3 a more detailed explanation is provided.

A BDI agent in Jason drives its behaviour by selecting plans from its *PlanLibrary*, according to its intentions, and in response to a certain event or goal. A plan is the representation of the procedural knowledge and capabilities the agent has at its disposal. Several plans can react to the same event or goal. Actions and sub-goals are the building blocks of plans, and they directly affect the environment, possibly changing it. In brief, a Jason plan is defined as: `triggering_event : context <- body`. As can be seen, the plan's goal (post-condition) is stated by `triggering_event`, `context` expresses the applicability of the plan (pre-condition), and `body` is the recipe of actions and sub-goals to be carried out in the plan.

The research presented here is focused on learning in BDI agents as described above, taking Jason as the programming language because it strictly implements the principles of the BDI architecture.

## 2.2 Learning and Planning in Multi-Agent Systems

The environment, and the degree of difficulty of the task that the MAS must perform, increases the complexity of its organisation and the relationships between the agents involved. When dealing with complexity, it is important to note the relevance of learning as a method to improve behaviour. One of the main aspects in MAS organisation is the exchange of messages between agents, and the agreement that can be obtained when tackling the task. The different views/levels of information are: partial/individual information; partial/collective information; and environmental information. Moreover, learning has been used to reach adaptive behaviour [121, 128]. Additionally, within a MAS it is possible to observe unexpected behaviours due to the interaction between agents-agents/agents-environment [32, 33].

On the other hand, agents need to determine which is the next action to carry out based on different factors. Planning accounts for the process to organise actions in the interest of reaching goals and achieving a better performance. Planning in agents has been undertaken from several perspectives, from logic-based approaches, case-based reasoning, operator decomposition, and as a searching process in a space of possibilities [38]. Additionally, planning is restricted by a couple of knowledge definition problems: a) the need for an accurate definition of the problem; and b) the space of solutions can prevent the planning process to be applied in real-world problems, requiring the definition of a search control which can be as difficult as the definition of the planning task. Planning in the individual agent level has been approached including learning techniques, commonly based on the Reinforcement Learning (RL) approach [120], i.e. capturing the interaction between the agent and its environment through a weighting mechanism to account for the next action to be executed.

Beyond individual agents, planning in the MAS scope, avoiding to have a central plan or to perform a centralized plan construction, is focused on a distributed planning ending in distributed plans, where agents should help each other achieve their plans whenever it is required [37].

In the case of BDI planning, it has been endeavored mostly in the agent level [74] with a few works considering the collective scope [23]. So planning in BDI MAS it is still a field to be explored and it is one of the motivations of the work presented here.

Cognitive agents exhibit some advantages over reactive agents, namely [4,38,106,134]: a) a persistent and goal-directed behaviour; b) their noteworthy feasibility to represent more complex behaviours than the mere reactive ones (like planning and heterogeneous reasoning), that allows to have differences in the decision-making processes of similar agents according to their particular contexts; c) a higher level communications capabilities; and d) the close relation between their theoretical concepts and the common terminology to model reasoning, that allows a higher level explanation of the agent's reasoning processes and make it possible to be communicated to non-computer science specialists, facilitating the integration of knowledge from different disciplines. Due to the aforementioned features, this work is concerned with learning (for acquiring new abilities) in MAS composed by BDI agents.

## 2.3 From Individual to Multi-Agent Learning

Machine Learning can be understood under the classic definition provided by Mitchell [77] page 2, given below:

**Def. 2.3** A computer program is said to learn from experience $E$ with respect to a class of task $T$ and a performance measure $P$, if its performance at tasks in $T$, as measured by $P$, improves with experience $E$.

An agent is a sort of program with a set of objectives to fulfill through the execution of some tasks $T$ able to be evaluated by a performance measure $P$. After the experience $E$ is collected, through agent's observations of the environment, it is processed for improving the accomplishment of the agent's objectives. So ML is straightforwardly applicable to agents.

At the agent-level, the predominant ML approach has been based on RL [13, 58, 65, 117, 120] focused on measuring either the individual agent performances or the competitive and collaborative efforts in predefined collective tasks, including coordination. Other important approaches are based on Inductive Programming [48, 56, 82] to capture the dynamic of the environment to adapt the agent decision-making process. Finally, other approaches have relied on mixed techniques, from the relational and propositional representations (through inducing a set of Boolean formulae, decision list, decision trees, etc.) to artificial neural networks models and genetic algorithms (to drive the agent behaviour ecompassing the environmental changes) [39,69,70,127,130], including case-based reasoning (to represent a set of cases to be managed by the learning exploiting the agent's experience within its environment) [24, 26, 88, 119].

Furthermore, BDI agents collect observations from the environment and react to them according to both a set of pre-defined plans of actions and their internal state. Learning in such a case refers to the transformation of observations from the environment into new ways of decision-making. So, learning is close related to a method for using observations to tune their plan selection or to acquire new plans useful in coping with environmental changes.

Similarly to single agents, a MAS needs to adapt its behaviour to the environmental changes and keep working properly. Therefore, learning becomes a key issue to MAS autonomy and functionality. The integration of ML to MAS generated the field known as Multi-Agent Learning (MAL) [10, 126].

The MAL problem can be outlined as follows [8, 85, 114, 126]:

**Def. 2.4** A MAS situated within a changing environment can behave optimally, based on the work of their components (agents or group of agents), that are able to adjust their individual behaviour by learning, while there exists incomplete information, a large space of states, and influence from other learning agents and the environment.

Many ML algorithms and approaches have been developed from a single-agent perspective and then extended to meet MAS perspectives, including RL, Artificial Neural Networks and Genetic Algorithms as the more used, and even originating the specialized field of MAL focused upon RL (MARL) for the sake of coordination and cooperation [85, 126], including other techniques for adapting MAS to a changing environment like logic-based and social learning methods [8, 114], encompassing BDI MAS [82, 128]. Other approaches explore different techniques like Case-based Reasoning [26,119] and Intentional Learning [49,56].

A brief chronology of different aspects and approaches on MAL according to [21,126] is shown in Table 2.1.

| **Approaches** | **Techniques** |
|---|---|
| Startup period (1980-2000)<br>It consisted of a breadth-first like exploration of MAL, its concepts, mechanisms and possibilities. Mainly based on RL and bio-inspired adaptive parallel computation techniques. Artificial Life field was originated. Flocking or herding behaviour, Evolutionary computation, Social learning, Neural Networks, Interactive and imitation learning. Swarm Intelligence, RL, Temporal-Difference RL Evolutionary Game Theory. | - Single-agent RL<br>- Q-Learning, Team and Distribute Q-Learning, Frequency Maximum Q-Value, Q-functions<br>- Minimax-Q<br>- Replication Equations<br>- Joint Action Learning<br>- Optimal Adaptive Learning |
| Consolidation period (2000-today)<br>It consists of a deep-first like exploration. Focused on MARL and Game Theory, and foundations of Theory of MAL. MARL in Game Theory, Case-Based BDI Agents, Inductive Logic Programming (ILP)-based Learning. BDI-MAS Learning. | - Nash-Q Learning<br>- Nonstationary Converging Policies (NSCP)<br>- Gradient Ascent Algorithms<br>- Case-Based BDI Agents<br>- ILP in MAS and First Order Logic Decision Trees |

**Table 2.1:** *Approaches and techniques of Multi-Agent Systems Learning.*

Owing to the development of the MAL field, different classes of learning have been identified to group up the effort and advances. Learning in MAS can be classified taking into account some features, according to their relevance to the aims of this thesis: a) the type of task; b) the agent's awareness degree; and c) the agent's heterogeneity.

The type of task the system is focused on determines whether the agents can behave in a cooperative/-competitive way or both [21, 126]. So they could follow a shared rewarding method to fulfil their goals or behave in a self-interested way. Cooperative behaviour is the most relevant to this work because here the messaging between agents is exploited to learn and to solve the task of the whole MAS defined at design time.

Other classifications account for: degrees of awareness when analysing an agent's relationship to other agents within the MAS; the learning process, relative to the nature of their learning goals (its own behaviour stability or adaptation to other agents, etc.); and when concerning the cooperative or competitive focus of agents, the degree of independence they have. This degree ranges from fully aware to fully unaware [21,126].

Those three intentional levels comprise agents with an homogeneous set of beliefs, but a higher (social) level of learning can appear when agents with different beliefs interact to learn [21, 56]. Additionally, the homogeneity of agents can be determined by the set of capabilities the agent can exert. Furthermore, agents can refine a starting model (used for cooperative or competitive purposes) through interactions with the environment; or when lacking from such a model, attempt to build an optimal policy (to guide their actions in response to the perceived state of affairs in the environment) [21, 103].

Having heterogeneity based on the agent's capability is relevant for the research presented in this work as it allows to explore behaviour acquisition in different agents. In addition, message exchange enables agents to be aware of each other to work together. Furthermore, the application domain requires different agents to perform a variety of actions, relying in the direct communication of agents to fulfil the global task of the entire MAS.

## 2.4 Learning as Planning in BDI Agents

It should be noted that the original definition of BDI agents does not include the learning capability [117]. As such, some proposals to join Case-Based-Reasoning with BDI agents [24, 26, 119] or to apply swarm techniques to a BDI architecture [128] have been developed. On the other hand, some efforts have been carried out mainly under **Intentional Learning** (IL) which stands for learning recognised as a behaviour driven by the mental attitudes of agents, like goals, beliefs and intentions [117] (this is discussed with greater detail in Chapter 3). IL has been addressed in three main directions: a) adjust goals and beliefs to drive the agent behaviour [90]; b) modify the plan selection by modifying the contexts (i.e. the applicability of plans) [6, 50, 56, 109–111]; and c) modify the actual set of plans available in the agent's repertoire [115–117].

Several works have focused on IL just in the level of plan applicability using some machine learning strategies, mainly based on logical tree induction algorithms (either using external tools or including them as Jason libraries) [48,49,56,109–111], or learning goals [90] even implemented a pure RL approach in Jason [11].

On the other hand, plan acquisition in cognitive agents has been of interest in the agent and MAS community. Due to the procedural knowledge of the BDI agents is encoded in the form of plans, learning, commonly, is featured as plan updating and plan acquisition, by using learning methods. In [75], an integrated version of Jason and a STRIPS-like planner is proposed, allowing the agent to add new plans as a result of the planner. To couple the two systems, a bidirectional translation stage from AgentSpeak to the plan's representation in the planner is required, to update the *PlanLibrary* of the agent. More recently, in [73], a successful probabilistic method of learning using tracked interactions with the environment has been proposed. Despite using first-order logic to represent the agent knowledge, this work is further away from BDI agents. Consequently, it does not use AgentSpeak to encode the plans but the Relational Dynamic Influence Diagram Language (RDDL) [98].

Finally, it is worth noting that works on IL, whose focus is learning the context of plan applicability and goal adjustment without the consideration of acquiring new plans (i.e. no new skills are learned by the agent), are not tackling the generation of new plans. Additionally, there is still missing an approach aimed at including IL in cognitive agents at the plan acquisition level, within the same framework, without using external systems. This work aims to progress the state of the art in BDI MAS in that direction.

## 2.5 Planning Techniques in Agents

Planning is a deliberation process that has been pursued in Artificial Intelligence to organise actions, to reach better results, or anticipate them. Planning can be independent or domain specific, whether it uses a generic or specific representation and techniques. In any case, the purpose of planning is not a search for a computationally optimal plan, but one near-optimal enough to overcome a current problem (due to planning being complicated and time-consuming), and a trade-off between costs, the effort of planning, and the benefits of those plans produced [42].

*Domain independent planning* is also called *planning from first principles* and is normally slow and costly, whilst usually complementing the domain specific one [74]. It makes use of mental simulation of actions.

Independent planning deploys theoretical devices called Conceptual Models, to describe problems and their elements, clarify assumptions, and represent the knowledge used. For instance, 4-tuple representation of a State Transition System (STS):

$$\sum = \langle S, A, E, \gamma \rangle$$

Where, $S$ represents the set of states in the system, $A$ is the set of actions the agent can pursue, $E$ is the set of events sensed by the agent, and $\gamma$ is a state transition function over the set of states, i.e. $\gamma : S \times (A \cup E) \mapsto 2^S$. The action's applicability is defined as: $a \in A, \gamma(S, a) \neq \emptyset$. While applying an action $a$ in $S$ produces a new set $S' \in \gamma(S, a)$.

Furthermore, a *Dynamic Planning* process interleaves planning and execution, undertaking plan supervision, revision and re-planning stages. A dynamic plan can be understood as a closed loop between planning and controlling, holding an execution status. On the contrary, *Offline Planning* is regarded with the initial goals and states, without considering the dynamics of the agent or the current state.

Commonly, planning is conceptualized as a search problem defined at a certain abstraction level, by deciding which actions and states belong to the space of states, representing them as a directed graph using the applicability of actions over states to establish the links between states. One of the early endeavours to do planning within this state-space search is the classical heuristic-based planner STRIPS [68,74] which analyses what happens before and after the planning process. This procedure uses a heuristic function that encodes knowledge regarding to a specific problem.

**Deterministic planning** is concerned with a planning procedure where an action $act \in A$ only has two possible outcomes, success or failure:

$$\text{when succeeds} \left\langle \; st_1 \xrightarrow{act} st_2 \right. \qquad \text{when fails} \left\langle \; st_1 \xrightarrow{act} st_? \right.$$
$$st_? = \text{unknown/arbitrary state (undefined)}$$

**Probabilistic planning** is concerned with situations where an action $act \in A$ can transit to multiple states $st \in S$, with a probability $p$ associated to each of them: $st_1 \Rightarrow p_{st_1} \ldots st_n \Rightarrow p_{st_n}$

In [74], a comprehensive survey on planning systems and reasoning systems integrated with the BDI paradigm is presented, pointing out the lack of native planning methods in the BDI architecture. According to [74], the work on planning within the BDI architecture has been focused upon whether to include an external planner to produce new elements of the *PlanLibrary* (plans) from scratch, or to guide the plan selection whilst running, be it guided by learning or other techniques.

The planning endeavours closely related to BDI agents are [74] the Markovian Decision Process with full or partially observability, and Hierarchical Network Tasks. In addition to the aforementioned techniques, Hindsight Optimization is here briefly reviewed, as it is a relevant method for sampling the environment, to gather the required information for pursuing planning within a changing environment.

Probabilistic planning is commonly implemented as a Markov Decision Problem (MDP) based on the *Markovian Assumption* [120] , (i.e. the dynamics of the environment can be modelled as a Markov Chain of stochastic transitions between states). In other words, the probability of transitioning from one state $st_a$ to another $st_b$ depends, partially, on either the current state (not the history of states) and the agent's action:

$$P(st_a \rightarrow st_b) \text{ depends on} \begin{cases} \textit{current state} \\ \textit{past action} \end{cases}$$

Under this assumption, planners implicitly represent goals as a function $reward(st_i, act_x)$ that defines a numerical reward for each state $st_i$ reached by the agent given an action $act_x$. Commonly, this function maps the state-action pair to a real number, or it could be binary only [120]:

$$reward : S \times A \mapsto \{0, 1\} \mid \mathbb{R}$$

In this regard, a MDP can be defined as:

$$MDP \ = \sum \langle S, A, P, R \rangle$$

Where, $S$ is a finite set of states, $A$ is a finite set of actions, $Pr$ is the state transition probability, and $R$ is the reward function of state transitions $reward(s_i, a_j)$.

A Partially Observed MDP (POMDP) is an extension of a MDP that adds uncertainty to the current state of the world (i.e. when deciding on optimal actions, the agent has no direct ways of knowing what the current state is). The agent only perceives indirect observations, with certain probabilities being governed in each state of the world according to a conditional probability function. Adding $\Omega$, a set of observations from the environment, a POMDP can be defined as:

$$POMDP \ = \sum \langle S, A, P, \Omega, R \rangle$$

### 2.5.1 MDP/POMDP and BDI Agents

There are some parallelisms between MDP/POMDP and BDI agents that have been underlined by some researchers [74, 101, 105, 128]. For example, a MDP/POMDP agent acts accordingly with a *policy* (i.e. the description about which is the best action to take in a given state), while a BDI agent incorporates this action-selection mechanism in its set of plans stored in its *PlanLibrary*, with each plan having a scope of applicability determined by the state of affairs within the environment alongside its beliefs and goals. On this basis, the relationship between MDP based and BDI systems has been established in [74, 101, 105].

The BDI architecture is not natively based on and a priori stochastic environmental description. In other words, environments in BDI do not have an explicit representation of the state probabilities $p_{st_a}$ associated to a given action *act*. So, when executing *act*, there is no model to determine the next state $st_b$ .

Furthermore, a BDI agent itself does not reason about possible failures, the failure is treated a posteriori.

$$act \rightarrow plan \quad \left\langle \begin{array}{ll} succeeds & \rightarrow \text{go to next plan} \\ fail & \rightarrow \text{plan to deal a posteriori} \end{array} \right.$$

States in stochastic models are similar to BDI agent's states [101]. As environment in BDI is modelled as a finite set of boolean variables representing every possible proposition in the domain. An environmental state is a truth assignment to these variables within the domain. By comparing both architectures [101], the components of BDI model can derive MDP optimal planning agents, and a BDI agent is able to plan more efficiently to generate a solution for a MDP. So, a BDI agent is an approximation of the optimal solution to a stochastic planning problem. A chosen action in a BDI agent is not necessarily optimal, but it is based on domain knowledge (stated by the designer and encoded in the *PlanLibrary*). The author even proposes a way to translate from both types of agents, once they are fully engineered (i.e. the total set of plans are previously defined in the BDI agent). Converting a BDI agent in a MDP, or even a POMDP based one (to account for incomplete sensing), implies a trade-off between the optimality and the aforementioned usage of knowledge. The outcome of such a conversion is that the solution to the stochastic planning problem could be used by the BDI agent to optimally choose the best plans at every possible state.

The assignment of utility to states, and probabilities to state transitions, makes a MDP an ideal approach to implement agents. Furthermore, MDP agents can learn those values by interacting with the environment.

The principle of Maximum Expected Utility MEU (MEU) states that agents should carry out a mapping from every state, based on the probable outcomes of every possible action in that state. However, reaching a MEU-optimality [105] is intractable. This is the main problem with stochastic based agents, as they are not scalable enough when going from toy problems to real-world.

In [105] a comparison between MDP and BDI architectures is performed, summarized in Table 2.2.

| **MDP** | **BDI** |
|---|---|
| Description | |
| $MDP_{Description} = \langle S, A, T, R, P, \Pi \rangle$<br><br>Where,<br>$S$: space of states<br>$A$: space of actions<br>$T$: state transition function<br>$R$: reward function<br>$P$: probability distribution over S<br>$\Pi$: set of policies<br><br>$Ag_{MDP}$ = an agent that uses a $MDP_{Description}$ to act | $BDI_{Description} = \langle S, A, T, B; D; I; Del, M \rangle$<br><br>Where,<br>$S$: space of states<br>$A$: space of actions<br>$T$: state transition function<br>$B, D, I$: Beliefs, Desire, Intentions, respectively<br>$Del$: a deliberation component<br>$M$: a means-end reasoning component<br><br>$Ag_{BDI}$ = an agent that uses a $BDI_{Description}$ to act |
| Features | |
| • Overcome heuristic based approaches in small/tractable worlds<br>• Intractable in many environments | • Can scale to outperform MDP in treating problems when scale is beyond MDP |

**Table 2.2:** *Comparison between MDP and BDI architectures (based on [105]).*

Relying on the comparison outlined in Table 2.2, the following similarities can be observed [105] (the underscript refers to whether the element belongs to a MDP or BDI description):

- Straightforward equivalences: $S_{MDP} \equiv S_{BDI}$, $A_{MDP} \equiv A_{BDI}$, and $T_{MDP} \equiv T_{BDI}$
- Functional equivalence: $P_{MDP} \equiv B_{BDI}$, both identify $s_c \in S$: the agent's current state
- $\{R, \Pi\}_{MDP} \equiv \{D, I, Del, M\}_{BDI}$, where $\Pi_{MDP}$ and $I_{BDI}$ are the key relation to be determined

Regarding the last equivalence in [105] asserted, $I_{BDI}$ can be conceptualized as the state the agent has committed to bring about, and $R_{MDP}$ is a means to determine policies; then $D_{BDI}$ is a step on the path to determining $I_{BDI}$, and they can be ignored, since the main comparison is made between policies and intentions.

So, the equivalence $BDI \equiv MDP$ is based on the fact that both can be complementary and translated into each other. Some steps to translate MDP into BDI are [105]:

- Given a policy extract a BDI description to approximate the solution to the MDP.

- Given a complete BDI description obtain a policy that a MDP agent can use to control its action.

As stated before in Subsection 2.2 RL, as a learning technique to capture the interaction of agents with their environments, is the most studied technique in MAS context (called MARL). In [10] an extensive survey on MARL, including its approaches, strategies and challenges, is presented. Despite its feasibility when applied to single agent, this learning is not solid enough to afford MAS learning, because of the interactions with other agents and the environmental changes (originated by both its own dynamic nature and by those agent's interactions whit the environment) [10,126], impose convergence difficulties as tractability directly increases. As mentioned above, a MDP-based agent need to map every state considering the outcomes of every possible action for each state. Consequently, when talking about MAS, where the set of states are increased by the number of agents, the learning space is exponentially bigger [22, 125]. Additionally, specifying a learning goal is hard (learning is not stationary, due to the agents in the MAS are learning simultaneously, generating changing policies). Therefore, agents need a way to coordinate their actions [10].

Despite these shortcomings, some efforts has been focused on reducing the search, by implementing different techniques under a cooperative or competitive stance, in domains beyond test-bed applications (such as search and rescue simulations [22,66,125]). This knowledge can be beneficial, including MAL based on other strategies (like the intentional approach), as the interaction with the environment is one of the main sources of information to shape the policy of action, altogether with the weighing process employed in RL approaches.

To conclude this point, one attempt to escape from the inherent intractability of the MDP/POMDP models is to make use of purely online action-selection policies, which compute (for each state) a planning procedure to determine the action to be executed. Hindsight Optimization [60, 137] (HOP) is a fast, online heuristic method of action-selection for stochastic control (i.e. it provides a principled reduction from probabilistic planning to deterministic planning). The idea behind HOP is to approximate the value of a state by sampling a set of non-stationary, deterministic problems that originated in that state (known as horizons or futures), then solving those problems as MDP, and combining their values (for example, by averaging the results). On the contrary to other MDP/POMDP methods, the heuristic of HOP optimizes a plan, not a policy. HOP estimates the cost-to-go of the belief by assuming full observability will be obtained at the next time step, resulting in a system that never tries to gather information, but can efficiently plan within the deterministic sub-problems [60].

### 2.5.2 Hierarchical Network Task and BDI systems

BDI agent's behaviour is described in the *PlanLibrary* (i.e. a set of plans designed by the programmer, that are consulted by the reasoning loop whilst running). This represents a search process, to retrieve the most applicable plan for a given stimuli. This search for plans is closely related to planning based on Hierarchical Task Networks (HTN) [42,76,100,123], an automated planning technique for hierarchical decomposition that progresses from a task decomposing it into subtasks until reaching non-decomposable or primitive ones in order to solve a problem according to engineered information provided to the planner [23,79].

The relation between BDI agents and HTN planners is explored by [100], and their research outlines that HTN planners can be employed by an agent to decide which plans to instantiate in order to succeed, but does not allow the agent to create new plan structures. The addition of a planning component to a BDI agent model has recently been revisited by other researchers, especially by [100]and [129].The former describes a BDI programming language that incorporates HTN planning by exploring the similarities between these two formalisms, but agents in this approach are no more flexible than they would normally be, since they still rely on the same plans originally designed prior to deployment. The latter approach is based upon a specially adapted planner to support the agent, preventing the model from taking advantage of new planning algorithms.

A comparison between HTN and BDI systems [29,99,133] is shown in Table 2.3

Some efforts have been undertaken to add lookahead planning to BDI (or BDI-like agents, or online execution, to HTN [74,86,87]. Due to their similarities, some work has focused on including HTN into BDI systems [74], following a procedure similar to the following:

1. Transform event-goals in BDI into abstract operators.
2. Call a classical planner with $\langle currentinitialstate, requiredgoalstate, abstractoperators \rangle$.
3. Check the correctness of the resulting plan to ensure there is a viable decomposition.
4. Eliminate redundancies/refine plans.

| BDI | HTN |
|---|---|
| 1. Focus on execution of plans<br>2. Respond to goals and information<br>3. Take decisions based on a current state<br>4. Backtracking is not an option<br>5. Interleave execution and planning<br>6. Take care of other possibilities that arise when executing a plan<br>7. Action based on event/goals and plans/rules<br>8. Failure → a plan/potential plan is not suitable | 1. Concerned with generation of plans<br>2. Bring about goals<br>3. Take decision based on multiple potential states<br>4. Perform hypothetical reasoning about actions and their interactions<br>5. Are ahead of execution<br>6. A search process that results in a plan<br>7. Action relies on a net of primitive and compound task and methods<br>8. Failure → an active plan ought to be aborted |
| Which plan to execute? | Does a plan exist? |

**Table 2.3:** *Comparison between HTN and BDI systems*

The second step includes the usage of an external planning tool (i.e. going outside proper BDI architecture). Regarding the case of a set of agents working together, recent work aims to provide formalisms to integrate HTN within a MAS using different MAS platforms. In [35] the authors propose a formalism to include HTN planning into the IMPACT MAS development environment [1] introducing a new algorithm for agent planning and exploiting the inter-agent communication capabilities of the platform. On the other hand, based on the aforementioned work, [23] describes a formalism for domain and problem representation that integrates the JaCaMo MAS system development platform [15] with an HTN extension to MAS scope [23]. Being a step ahead in the MAS planning, it still lacks from translating the plans generated (by an external planner) into plans that can be added to the respective agents plan library.

## 2.6 Decision making in the collective scope

Early attempts to apply ML to MAS were from a *single-agent perspective* (i.e. how to improve the agent skills or knowledge), paying little attention to the interaction between other agents, rather focusing on the environment [8]. The result of these individual learners is propagated to the whole MAS to adjust its behaviour. Furthermore, a recent proposal extends the traditional reinforcement setup accounts not only for

[1]The IMPACT Project is leading by University of Maryland, it aims to facilitate the creation, deployment, interaction, and collaborative aspects of software agents in a heterogeneous, distributed environment `http://www.cs.umd.edu/projects/impact/`.

individual learning, but the collective influence of other agents [54].

Furthermore, by emphasizing differences between collective or individual approaches (single-agent and multi-agent learning), learning can be classified as follows [126]:

- **Multiplied learning**. Only individual learning exists, despite communication with other agents. Agents are generalists for they can act as an entire learning process.
- **Divided learning**. The learning mechanisms divide the task a priori, (i.e. the goal is shared). The individual learning results are joined. The inter agent communication concerns the input/output of the individual learning effort. The agents are specialists, responsible for just a fragment of the learning process.
- **Interactive learning**. The solution to the learning task is obtained through a shared understanding of the learning goal, which emerges from knowledge communicated in a flexible, dynamic way (consensus, argumentation, mutual explanation, etc.). Agents are neither generalists nor specialists, but a regulator to the learning process, and integrator to the different (possibly conflicting) agent perspectives.

However, the result of these approaches of MAL is a joint learning outcome (i.e. knowledge or plans), and agents are just a part of the process. This leave aside cases where learners interact to each other to improve its own learning process without being generalists nor specialist just in charge of part of the learning. In this regard, the work presented in this thesis is more closely related with a multiplied learning approach but for the sake of a global problem solution (i.e. the purpose of the entire MAS), in other words, the agents are expected to learn new skills through interacting with others. Consequently, despite the importance of these approaches, they are too rigid to meet the requirement aforementioned.

In addition, IL in the level of plan acquisition must determine the actions to be included in the new generated plans. For this purpose, the learning process needs to consider the state of the environment and the influence of other agents to construct plans. Some of the main approaches to model the decision-making process in both the agent and the MAS levels are within the MARL field. The relevant decision-making models to this thesis, are Collective vs Individual, Joint Intentions, and Social Learning, summarised as follows [25,53,54,78,114,125,126]:

- The Collective vs individual model (CvI), based on the HTN planning method, it adopts an approach that is neither purely individual nor purely collective, exploring an explicit separation between both decision strata whilst aiming to conciliate their reciprocal influence. It considers the individual choice of agents, and assumes that behaviour coordination happens by learning from a timely, prolonged relation exercised in collective and individual strata.

  The operation of the MAS is based on a *Cooperation task* that configures a hierarchical task organisation based on the framework of *options* [120]. It extends the MDP theory with *temporally abstracts actions* (i.e. tasks that obey termination conditions) whose execution use a subset of *primitive actions* or *one-step actions*. On the contrary, multi-step actions are used during a variable time duration. This configures a multi-hierarchy of which each option is an element. The policy of each option chooses amidst other lower-level options.

  The decision-making process relies in the interaction of both strata, individual and collective, described as follows:

  - *Individual stratum*. Composed by a set of agents with capabilities described as a hierarchy of options. When agents belong to a certain hierarchy (using the same set of options), they are considered homogeneous, while agent homogeneity is established by those agents belonging to a different hierarchy (having a diverse set of options).
  - *Collective stratum*. Defined by a single institutional agent that condenses the set of agents into one that represents the collective. But when acting, it is individual agents actually performing the actions.

  In the operation dynamic, agents compute the benefit of selecting any option from their own set of options, or by requesting the collective stratum for a decision. Consequently, the model avoids a centralised decision-making.

  However, this model does not capture the intentional stance of the agents (like the BDI architecture) towards team attitudes. Furthermore, the collective stratum is explicitly represented by only one agent. So, while allowing a decentralized decision-making, the entire set system depending on just one agent, decreasing the capability of failure tolerance of the MAS.

- Joint-intentions in the teamwork (JI). Instead of adopting the notion of *intention* as an internal commitment to perform an action (a timely, persistent goal, or proposition to be satisfied), it also considers intentions as a representation of linear plans, which agents have adopted when committed to reach a certain state. The JI model establishes a difference between *team* and *teamwork*. A team is a set of agents collectively committed to achieving a certain goal. Teamwork is a temporal notion consisting of a set of agents that agreed to conform future directed joint-intentions, keep those joint-intentions over time, and then jointly act. Additionally, they mutually believe they are acting to reach the same goal. Teamwork ends whenever all the agents believe that at least one member considers the goal finished, whatever the cause (the goal is achieved, impossible, irrelevant, etc.).

  This model is not built on a pure BDI architecture, but inspired by it. In spite of capturing the intentional stance, it lacks the MDP domain-independent support for sequential decision-making in stochastic environments.

- A combined approach (CvI-JI): Integrates both aforementioned approaches, by modifying (at the collective stratum) the option selection process following a RL typical options selection. Given a perceived state, a collective action is chosen based on its compatibility with the hierarchy of the agents involved, in a selection process premised upon $\epsilon - greedy$: choosing at random from available collective actions; or by choosing the highest valued collective action for a given state.

  A noteworthy concept of CvI-JI model is the *Teamwork Commitment* that allow agents to have asynchronous decisions. The agents keep joint-intentions over time, then jointly act. It does not imply immediate action to reach the same goal. At each cycle, an agent may establish a joint-action whilst still acting to satisfy another intention.

  To do so, this model carries out a teamwork formation process, allowing agents team reconsideration. Team formation can have two stages: a) the ongoing task continue, where an agent decides to become a team member by establishing a joint-intention, even if it is still executing another task or joint-intention; and b) the team option startup, where an agent in the team decides to start executing the team option. Also, in team reconsideration, an agent can withdraw from the team beforehand.

  The *Teamwork design component (tdc)* is a structure to manage the teamwork, composed by an ongoing set of states (those states where the agent is working to achieve the team goal), and the probability of an agent effectively committing to perform the team option. It implies that: a team option is always represented in more than one agent; an agent in the team specifies a tdc scenario for every team option it may be committed to; and each ongoing set of states is particularly defined by the agent's perception (i.e. it is specified by only taking the agents local view of the environment).

The model reduces the option space resulting in an improvement to the decision-making models it is based upon. Despite using intentional concepts, the approach is focused on a pure RL framework rather than in fully-BDI agents. Although, the usage of the joint intention and joint commitment altogether with the teamwork reconsideration are interesting ideas to be explored in a MAS composed by BDI agents.

- Social learning: This is based on a dynamic partnership (an agent repeatedly interacts with different and randomly chosen agents). Each agent learns its policy based on iterated, pairwise interaction with many peers. The set of agents in the MAS, each having different roles (capabilities), are divided by $T$ groups and teams. The diversity of roles is spread amongst the teams so that agents with the same role belong on different teams. The pairwise learning is modelled by means of a payoff matrix ruling the two-player interactions of peers in a cooperative game. There are three levels of observability to agent interactions: local/individual; collective; and global. These are regulated by a parameter $M$.

  - *Local interaction*. Between agents of the same team ($M = 1$), an agent can perceive the action-payoff of both itself and peers within its team.
  - *Collective interaction*. Between agents from a given number of different teams $M < T$).
  - *Global interaction*. Between agents form all different teams in the MAS ($M = T$).

  On this basis, the model defines two ways to implement social learning in a MAS:

  - *Individual action learning (IAL)*: An agent can only perceive the tuple action-payoff in a local interaction. It implements Q-learning through an optimistic updating of values (taking the highest reward for each action, episodically). Additionally, it uses a heuristic [64] to update the values, considering the frequency of each action to be highly rated in the current episode.

- *Joint action learning (JAL)*: Contrarily to learning the Q-values of individual actions only, agents learn by collective interactions, where the agent has access to the joint actions of peers from its own group and other $M$ groups. In addition, agents learn the Q-values of individual actions, and the relative performance of each individual action, with respect to the same action when taken by their peers in the current episode.

Some points about the state of the art in learning in Multi-Agent Systems are stated as follows:

1. The need to develop group learning (either team or concurrent) algorithms, and strategies to reach Multi-Agent learning.
2. The fact that learning continuously is the most suitable approach to MAS as a group learning effort.
3. Multi-Agent learning in the RL framework commonly reduce the individual and collective decision-making in one model, being either purely collective or centralised.
4. Hybrid MDP-BDI approaches exploits BDI plans to improve MDP tractability, while using MDP to improve plan selection, they omit automatic plan generation in BDI agents.
5. The focus upon more realistic environments and test-bed for MAL, different to the toy-problem that traditionally used for testing. A domain based on real-life would be an appropriate, inspiring way to tackle the theoretic and practical issues of MAL.

Finally, some proposals have included the use of learning techniques, based mainly on Reinforcement Learning and bio-inspired mechanisms [72] to promote SO in MAS, mainly based on reactive agents. Additionally, an important improvement to the development of MAS that includes SO and EM is lifting from reactive to BDI agents. Regarding the implementation of EM with BDI-based MAS, some proposals have reached emergent properties [80, 81] (the cognitive agents has not pre-engineered the intended ability). Therefore, EM in MAS refers to the resulting collective behaviour of agents extending beyond the individual behaviour of each [34, 91], making it relevant to include ML in the MAS for promoting new behaviours in individuals and in the global system, as a strategy to promote emergent outcomes when solving a given problem. In [81], a summary of agent capabilities and emergent properties is presented, particularly team formation based on knowledge, knowledge distribution, collaboration patterns, reinforcement of the best plans to use, and task processing improvement. A classification of EM in BDI agents is shown, namely: a) Belief-related EM; b) Goal-related EM; and c) Plan-related EM.

## 2.7 Summary

This Chapter progresses with the fulfilment of Research Objectives 1 and 2. A background, required to understand the research, is provided. Furthermore, a review of the literature regarding the possibilities of behaviour acquisition in the BDI agency and the associated techniques is reported.

The need to understand and afford complexity has been carried out by several fields. Multi-Agent Systems, being complex systems on their own, are a suitable technique to take complex systems as an object of study [32,33].

Self-organisation and Emergence, two phenomena occurring in complex systems, are also present in Multi-Agent Systems. Self-organisation allows a system to change its organisation without depending on an external explicit command whilst running. Emergence, particularly Computational Emergence, is the existence of a novel global behaviour (with respect to the constituent parts of the system).

The development of Multi-Agent Systems can benefit those phenomena accounted for, as they can better cope with complexity, easing both the design and understanding of the development process. More importantly, a better comprehension of the systems to be simulated can be acquired by using such an approach.

Cognitive agents, particularly BDI agents, are a robust approach to develop agents and Multi-Agent Systems useful for representing aspects of human behaviour. It is based on mentalistic notions of belief, desire, and intentions, to represent the knowledge of the agent, the design purpose, and its goals, which are intended to be reached by combining reactive and deliberative approaches. In order to act, a BDI agent has at its disposal a set of plans that encode the procedural knowledge and agents' capabilities. Due to this, it is relevant for studying complexity, and for the development in simulations of social and natural systems, for example, social simulations of disaster and rescue tasks in urban scenarios.

Machine Learning is a discipline that has been applied to Multi-Agent Systems for the sake of improving the behaviour of agents, whether in the individual level or in the collective stratum. Endeavours in this field can help with the design of systems that exhibit Self-organisation and Emergence. On the other hand, most of the work carried out in the field is focused on reactive agents. Despite Reinforcement Learning has been used with cognitive agents, it does not include BDI agents to generate new plans, nor planning in a BDI MAS.

Learning in BDI agents is related to the management of goals, plan selection or plan acquisition. In the last case, learning is essentially planning. Planning methods in agents are commonly based on statistical measures and Reinforcement Learning, such as Markovian Decision Process -fully or partially observable- (MDP/POMDP), and Hierarchical Task Networks. It is worth noting that none of those methods are committed to generate a plan (such those recipes used by BDI agents), but to decide the next action in a current state.

Regarding the interleave of Machine Learning and Planning, most of the proposals in the field lead to a centralised or purely individual approaches making it difficult to scale the solution beyond test-bed domains.

In BDI agents, planning has been implemented through Intentional Learning, a technique that can combine other approaches (such as Reinforcement Learning) to generate new plans that drive the agent behaviour. It is closely related with Hierarchical Task Networks, from similarities in the action selection of the BDI agency. However, they are different, BDI is regarded with the execution of plans, but Hierarchical Task Networks are concerned with plan generation. Multi-Agent Learning under intentional learning has been reached by adding ideas and techniques from other traditional ML techniques, like Reinforcement Learning, Inductive Logic Programming, and Manipulative Abduction, and it has been applied (under the agent-level point of view) to the scope of a set of agents for the sake of goal adjustment and plan selection, leaving out the plan generation. Consequently, there is still the need to fully couple the plan acquisition by learning in the BDI agency and even more, going beyond the individual strata of agents, to reach collective learning in intentional systems.

Furthermore, learning is closely related with Self-organisation and Emergence, particularly with the concept of Explanatory Gap (i.e. the emergence of outcomes which are unexplainable in terms of a given interpretation only, but through means of a new language created by the emergent products in a higher level of abstraction). This is related through having different properties than the original system. The last point is similar to the effects of Machine Learning: it applies a process (for instance, plan acquisition) to a stratum (the data), building up a hypothesis or explanation expressed in a different stratum. While comprising the data, the hypothesis demands a higher level of interpretation, so requires a different representation. Consequently, the change in the representation is based on a procedure that creates an emergent outcome: the explanation of the data. Hence, Machine Learning represents a way to produce Computational Emergence, purposely for problem solving, and Intentional Learning in the plan acquisition level represents a feasible way to implement learning in the BDI agency, to develop new skills for changing the behaviour of the agents. However, there is still the need to fully couple the plan acquisition by learning in the BDI agency, then going beyond the individual strata of agents to reach collective learning in intentional systems.

# Chapter 3

# Enhancing the BDI Agency With Learning

This Chapter presents the first step to answering the Research Question "How intentional learning, as plan acquisition, can be conceptualised and implemented in a fully-fledged BDI agent architecture and framework?", progressing to explain the learning problem in the BDI architecture, and introducing the experimental framework, to fulfil the Research Objectives: 1 "Exploring the behaviour acquisition in the BDI agency" and 2 "Developing a BDI MAS considering the concepts of EM and SO in its design".

The following sections report on a threefold topic: A) the description of the learning problem in BDI agents, in both the individual and the collective scopes; B) the description of the tools to set an experimentation framework, where the learning BDI MAS will be pursued; and C) the description of the Disaster-Rescue (DR) domain, and the first implementation of a BDI MAS in this Domain.

## 3.1 Learning in the BDI Agency

Recalling Def. 2.1, agents are computers systems situated in some environment. They are problem solving entities [38, 62], executing a three stepped loop [62, 134]: perception, deliberation, action. Then, agents are constantly interacting with the environment, whilst trying to autonomously achieve its goals in the fulfilment of their design objectives [38, 62]. The abilities and tools that both individual agents and MAS use to look for a solution to their intended problems, are defined in designing time. However, the environment can change whilst agents are still operating to carry out their tasks, and those changes frequently cannot be foreseen by the programmer. This difficulty is even higher in the case of MAS, and it is crucial when considering dynamic environments [8]. Consequently, including learning as a method to improve the behaviour of agents takes a great relevance to the field of MAS.

### 3.1.1 The BDI architecture

This research is focused on learning in cognitive agents, specifically BDI agents which original definition lacks of learning capability. Before to describe the learning problem it is helpful to review the main features of the BDI architecture.

BDI agents use three intentional attitudes, representing the information, motivational, and deliberative states of the agent [93]. These attitudes are modeled in the architecture as data structures: beliefs (i.e. knowledge resulting of perception or the internal operations of the agent during the reasoning loop); desires (i.e. the condition or state of the world the agent is motivated to achieve); and intentions (i.e. the selected course of actions the agent is engaged to execute). Additionally, BDI agents operate with a input queue of events. Furthermore, intentions represents a stack that hierarchically relate plans, and several stacks can run altogether. Agents execute a cycle of perception, deliberation and action depicted in Figure 3.1.

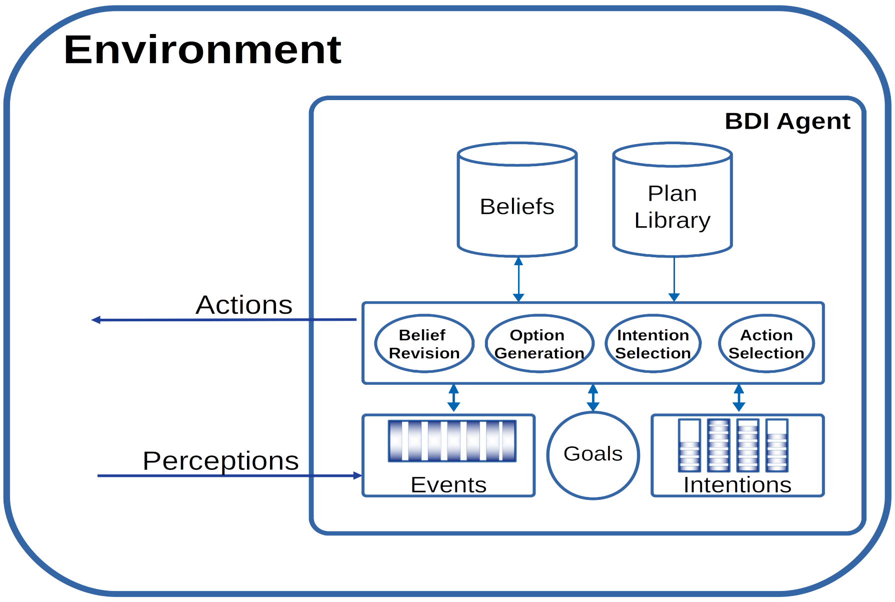


**Figure 3.1:** *The cycle of perception, deliberation and action, in the BDI architecture. Reasoning is performed through some BDI functions that operate over several elements, including events, beliefs, intentions, goals, and plans.*

In addition to beliefs, desires and intentions, such a cycle is performed using other components of the

BDI architecture [38, 134], namely:

- Belief revision function: Constantly updates the set of belief with information from two sources, the environment and the processing of previous agent's beliefs.
- Option generation function: Determines how the agent will reach its intentions (known as mean-ends reasoning or planning), it selects the agent's desires based on the beliefs and intentions the agent currently holds. It starts from abstract definitions of intentions to concrete actions to carry out.
- Intention-selection function: Acts as a filter, deciding what to do (the agent's deliberation process), it determines the current intentions An agent should adopt based on its beliefs, desires and previous intentions.
- Action selection function: Selects the actual action to be performed, in according to the current selected intentions.

The reasoning cycle of agents based on the BDI architecture is shown in Algorithm 1 (adapted from [93] and [38] (p. 53).

**Algorithm 1:** The general BDI reasoning cycle

```
Data: B: beliefs, I: intentions, O: options, E: event queue
Result: a: action
begin
    // Initialise Beliefs and Intentions
    initialise(B)
    initialise(I)
    while True do
        // Update beliefs with new events
        O ← perceive(E)
        B ← belief-revision(B,O)
        // Planning, decides how to achieve intentions (get desires)
        D ← option-generation(B,I)
        // Deliberation, decides what to do (get new intentions)
        I ← intention-selection(B,D,I)
        // Action selection, select an intention to be executed
        a ← action-selection(I)
        // Execute the action (defined in a plan belonging the selected intention)
        execute(a)
        // Update I and D if required
        drop-successful-attitudes(I,D)
        drop-impossible-attitudes(I,D)
    end
end
```

As can be seen, the main operation is outlined in lines 3 to 8. The reasoning cycle of a BDI agent continuously update the list of events whether internal or external, and a belief revision is performed (lines 3, 4). Then, the event selection function chooses a set of plans 5, hence one plan is selected and one intention is created (line 6). Thereafter the action selection function decides the actual course of action to be carried out (i.e. the plan to be executed accordingly, -line 7). Finally, the selected action is executed 8.

### 3.1.2 Intentional Learning

One approach that has been proposed to address the need for learning within the BDI architecture is the Intentional Learning framework, where learning exploits the intentional attitudes of the BDI to add learning as part of the agent's deliberation process in order to modify the generation of options, modifying the agent's behaviour accordingly [56,115].

On this basis, this kind of learning can be stated as follows, in definition 3.1:

**Def. 3.1 Intentional Learning** is a learning process incorporated as an agent's behaviour driven by its mental attitudes, such as goals, beliefs and intentions

In other words, IL requires the agent to have explicit mental attitudes that motivate and guide the learning.

Based on this definition, IL can be applied to BDI agents in three different levels, according to the outcomes of the learning:

1. Goals [90]. Learning is focused upon monitoring beliefs and adjusting them as needed in order to select the applicable plan to the current state of affairs.

2. Plans contexts and applicability [6,50,56,111]. Learning is centred around the adjustment of triggering conditions for plans when given certain contexts. Plans remain the same, but the agent the agent learns that they can be applied in different contexts.

3. Activities or Plans [115–117]. Learning is described in pre-specified plans, and directed by goals in a reactive-like manner. New skills are gained by adding new plans to the agent repertoire.

The research presented in this thesis is focused on the acquisition of new skills in BDI agents. Consequently, the most relevant approach of IL is learning at the activities level (i.e. the plan level), as it is the only one that includes the possibility of new behaviour generation.

### 3.1.3 Problem description

In the BDI agency, plans encode the procedural knowledge that determines the course of action to be carried out by agents. Plans are defined as recipes, in design time. This means that whilst operating, agents can face a situation that is not covered by any plan (or set of plans). Agents manage this cases by either reconsidering their intentions, or by raising a failure if not having any option to cope with the situation.

Consider an agent situated in a grid-like environment, whose main purpose is foraging, as shown in Figure 3.2. This foraging agent inhabits in a given cell at a given time. It can perceive the adjoining cells to its position. The agent wanders around the environment searching for food, by moving in to any adjacent free cell. Such agent keeps itself perceiving, strolling, and collecting food when found, in an everlasting loop.

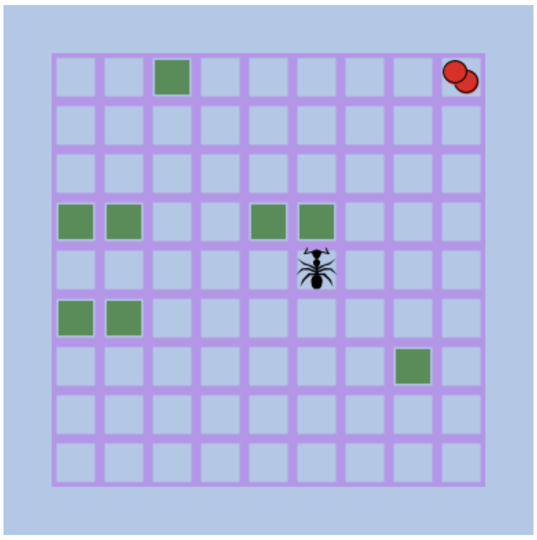


**Figure 3.2:** *A foraging agent situated in a grid-like environment.*

A plan library of the agent described so far should comprise the set of plans for moving and foraging. Recalling the Procedural Reasoning System (PRS) [38, 134], one of the earlier implementations of the BDI agency, a plan is composed of three elements, namely: goal, context, and body. Adopting a PRS-inspired plan structure, expressed in a non-standard representation (i.e. used here just for illustrative purposes), this plan library is shown in Figure 3.3:

Plans IDs are given just as a way to clearly identify each plan. Goals `move` and `collect`, are repeatedly adopted by the agent whilst operating, accordingly to the BDI reasoning cycle. So, the agent acts in accordance with the perceived state of the environment (i.e. the cells around its current position), and its internal beliefs,

```
ID:TLeft                     ID:MoveOn
goal:                    goal:
    move                     move
context:                 context:
    left-cell-free           cell-ahead-free
body:                    body:
     turn(-90)                step-forwards


ID:TRight                     ID:Collect
goal:                    goal:
    move                     collect
context:                 context:
    right-cell-free          food-on-current-cell
body:                    body:
     turn(90)                pickup-food
```

**Figure 3.3:** *Plans for moving and collecting food, from the plan library of a simple foraging agent.*

desires and intentions. Each plan is selected as an option by means of its goal. For example plans for moving, to be considered, requires the fulfilment of a condition: the corresponding cell (whether to the left, right or forefront) should be empty. Moreover, the goal `move` derives in several options, as many as the number of plans in the library related to it. Then, the actual actions to be executed are described in each plan's body: `turn(-90)` and `turn(90)` to turn ±90° left or right, respectively; `step-forwards` to advance; and `pickup-food` to collect food.

A hypothetical trace of the agent while foraging, assuming the execution of the following sequence of plans (denoted by their ID's): ⟨`TRight`, `MoveOn` (× 3), `TLeft`, `MoveOn` (× 3)⟩, is depicted in Figure 3.4,

Hence, the foraging agent seems as fulfilling its task. However, under a different sequence of executed plans, the situation could be very different. For example, assuming the following sequence: ⟨ `TRight`, `MoveOn` (× 5) ⟩, the agent could reach an unmanageable state. Given the set of plans it has at its disposal to act, the agent could get ensnared itself in the environment. This is because of it has no plan to cope with tunnel-like structures (i.e. none of the plans could fulfil its context's condition; therefore, no plan can be applied). This is illustrated in Figure 3.5.

To solve this situation, increasing the set of plans available to the foraging agent is required. Discarding the option of stop the agent's execution to add newly designed plans (this time considering the uncovered situation), the alternative is to include a way to cope with unexpected situations in an autonomous way. This

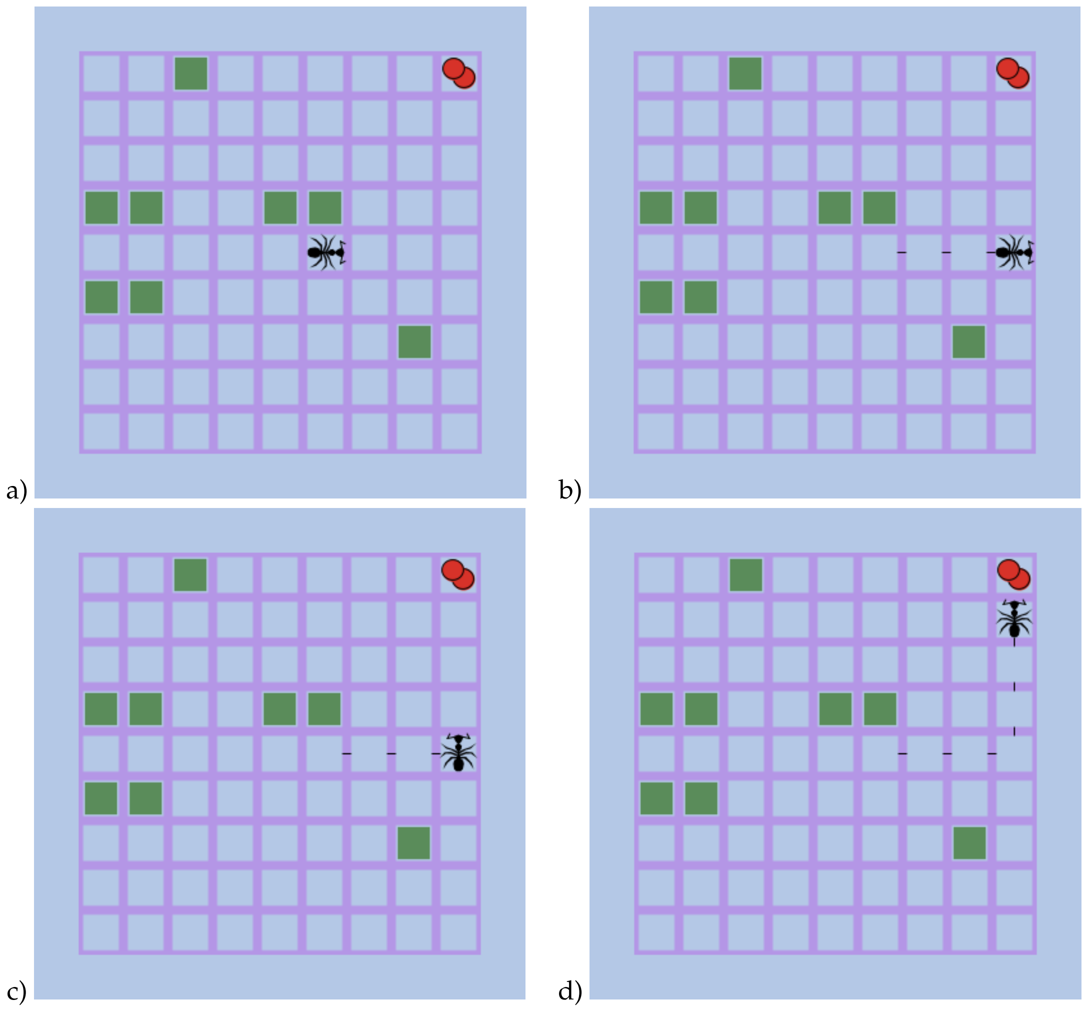


**Figure 3.4:** *A hypothetical trajectory followed by the foraging agent, resulting in finding food. a) agent starts searching for food, turning right. b) agent walk forwards till reaching a wall. c) turning left. d) walking to one step before finding food.*

is the purpose of IL at the level activities or plan acquisition.

Furthermore, from Def. 3.1, IL should be incorporated as a behaviour in the agent. Therefore, the agent needs to identify when to learn (i.e. learning is adopted as an intention). Consequently, IL needs to be fully incorporated in the BDI reasoning loop introduced in Algorithm 1.

Additionally, IL in the level of plan acquisition is concerned with the functions in charge of planning, deliberating and acting (lines 5 to 7), as follows: In both `option-generation` and `intention-selection`, the operation is not affected, but rather the set of options to be generated is increased (i.e. more plan -recipes- can be considered in the option generation to create intentions). Complementarily, the action selection could be adapted to ensure that a plan anew can be selected once it is added to the plan library of the agent.

On this basis, applying IL at the level of plan acquisition, to the problem of the foraging agent, will

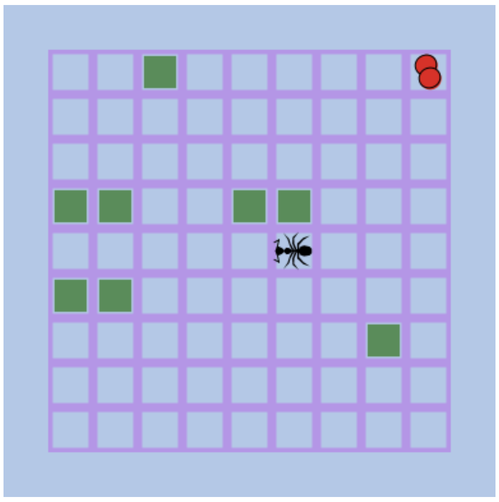
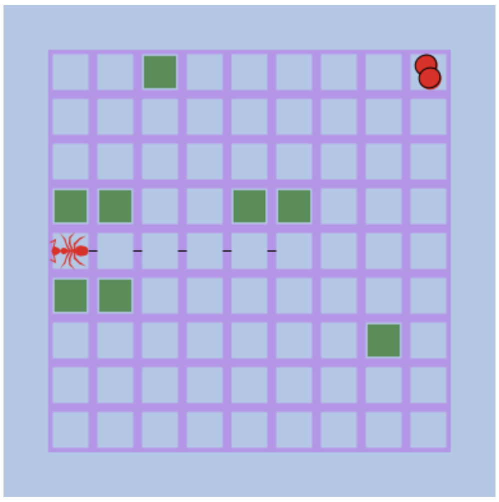
a)  b) 

**Figure 3.5:** *An alternative trajectory pursued by the foraging agent, resulting in being ensnared. a) agent starts searching for food, turning left. b) agent walk forwards till reaching a wall, but this time it is a tunnel-like structure. Then, the agent got trapped in the environment.*

result in adding a new plan or set of plans to the agent's plan library, aimed to cope with the tunnel-like structures. To continue with the example, assume that such a learning process produces one of the simpler solutions: to add a plan applicable when the cells to the left, right, and front, are not empty. This state could be expressed, for illustrative purposes, as a negation of the beliefs `right-cell-free`, `left-cell-free`, and `cell-ahead-free`, (i.e. they are not fulfilled). Finally, this condition is verified, thus turning left or right twice. A plan like this is shown in Figure 3.6.

```
ID:LearntPl
goal:
    move
context:
    not right-cell-free and not left-cell-free and not cell-ahead-free
body:
     turn(90)
     turn(90)
```

**Figure 3.6:** *A hypothetical plan after learning. It enables the agent to identify the tunnel-like structures and acting accordingly (i.e. turning twice).*

Using the newly added plan, the foraging agent can properly manage the tunnel-like structures. For instance, the agent can execute the following sequence: ⟨`LearntPl`, `MoveOn` (× 2)⟩. Therefore, the foraging agent carries out the following steps, as illustrated in Figure 3.7:

1. Triggers the learning when there is no plan to cope with the tunnel-like structure.

2. After learning, a new plan is added (`LearntPl`), and it is selected as the current option to be applied.

3. The plan is executed and the first turn from the plan's body is performed.

4. The second turn in the plan's body is executed.

5. The agent executes the movement accordingly with the only applicable option: `MoveOn`.

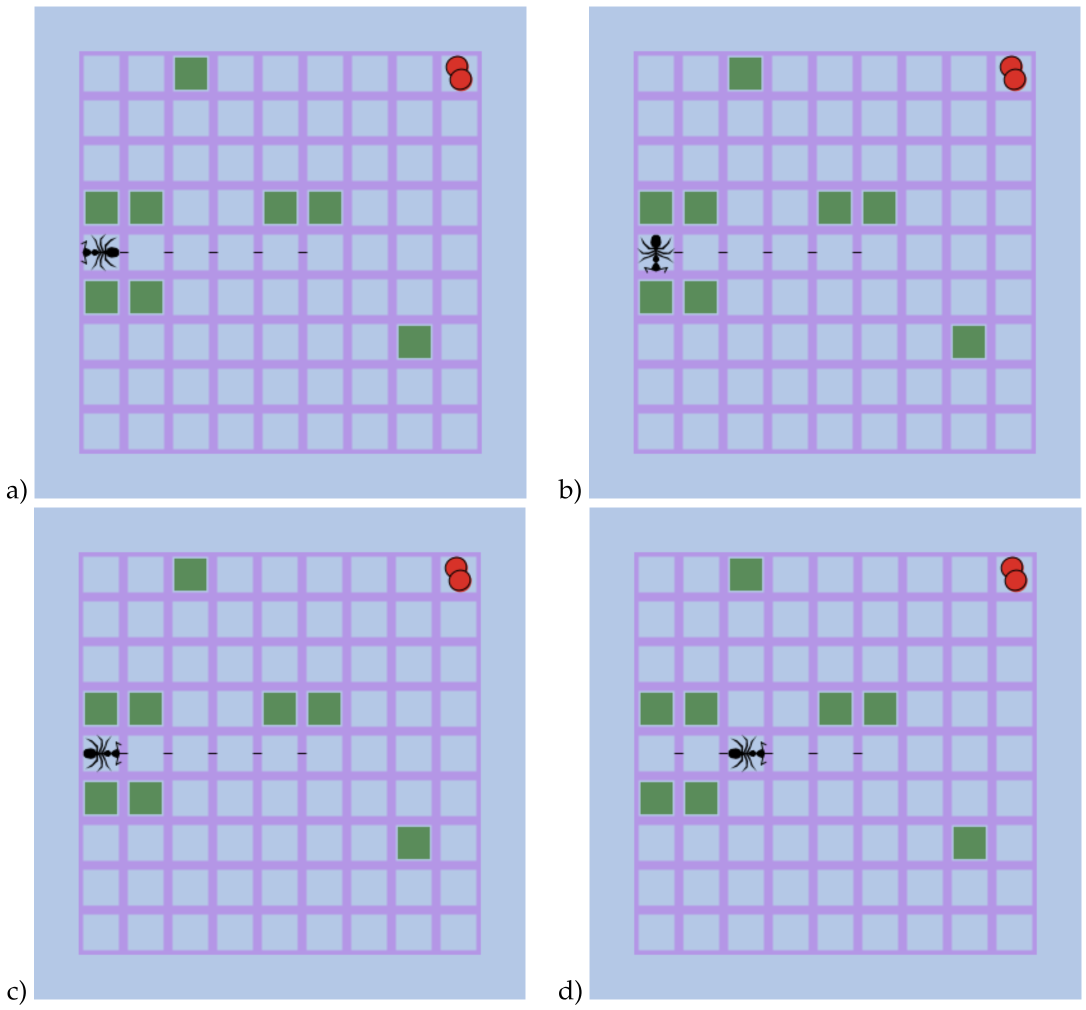


**Figure 3.7:** *After learning, the new plan enables the foraging agent to escape from tunnel-like structures. a) Learning is triggered. b) The new plan is selected as an option to be applied, then the first turn is performed. c) Then agent turns again. d) Agent move forwards because of taking the only applicable option:* `LearntPl`.

With this plan, the foraging agent can travel freely over the environment and it is able to increase its procedural knowledge by adding new plans to its plan library as needed.

Finally, IL in the level of plan acquisition, consists of having as a plan (or set of plans) that generates other plans. Then, the learning problem is double-sided:

- Determine when learning should be triggered, and
- Determine the learning method for acquiring new plans based on the agent intentional attitudes.

### 3.1.4 Intentional Learning in Multi-Agent Systems

Learning in general is applied to agents for behaviour improving, to cope with new situations by acquiring new skills either in the single agent case or in a MAS. In BDI agents, IL has been addressed as plan selection (analyzing the plans's context of applicability) in both agents and MAS. However, there is no any approach to include IL to acquire new plans in BDI agents at the MAS level.

As asserted in Def. 2.2, a MAS is a group of interacting agents. Agents can be homogeneous or heterogeneous, accordingly with: firstly, architectural heterogeneity, concerning the architecture they are based on (reactive or cognitive); secondly, related with the knowledge agents manage during the operation within the MAS, including variations in goals, and the values of their parameters and variables (quantitative heterogeneity); and thirdly, the set of abilities they are able to perform (defined by their procedural knowledge). The last two aspects can be related with the *role* an agent is committed to perform within the system. Another feature to be considered in MAS is if the agents interact in a cooperative or competitive way, which can be also affected by its homogeneity or heterogeneity.

Collective Intentional Level (CIL) accounts for IL from the individual to the collective scope, through interactions with other agents in the MAS. CIL is outlined in 3.2 given below:

**Def. 3.2 Collective Intentional Learning (CIL)**, is a learning process incorporated as an agent's behaviour driven by its mental attitudes like goals, beliefs and intentions, and the sharing of knowledge with other agents.

This can be illustrated by an example as follows. Considering a group of three homogeneous agents allocated in the same grid-like environment described in the previous section. The simpler case, where an entire plan is exchanged between heterogeneous agents is depicted in Figure 3.8.

This simple case proceeds as follows:

1. The learning agent would request to other agents to share their procedural knowledge. For illustrative purposes, the request is about having a plan to be applicable to a given event and context: $Pl(E, C)$?,

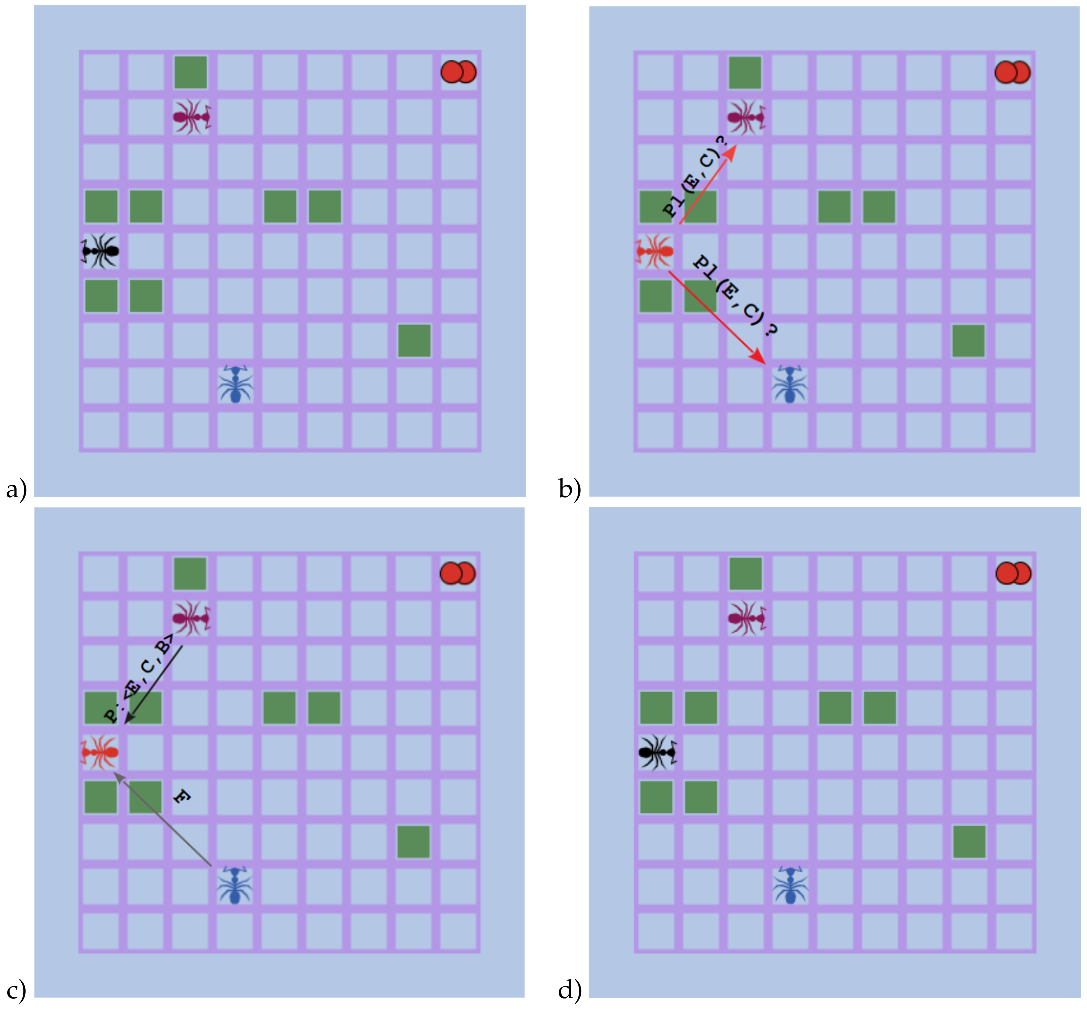


**Figure 3.8:** *Collective Intentional Learning in a MAS composed by three foraging agents. Plan sharing to escape from tunnel-like structures. a) Once the black colored agent gets into a tunnel-like structure, learning is triggered. b) Learning agent (red colored) ask others if they have an applicable plan to the current event and context. Pl(E, C)?, denotes a plan-like structure with no body* $P :< E, C, \_ >$*. c) Agents reply according with its own intentional attitudes, in this case: F for not having a plan, otherwise with the actual plan structure* $P :< E, C, B >$*. d) Learning agent (black colored again) adds the plan to its plan library. It can continue operating in the environment now.*

denoting a plan-like structure, but without a body component: $P :< E, C, \_ >?$.

2. Peer agents should answer to this requests if they are able to fulfill the request (i.e. they have a suitable plan to be shared). Any agent able to provide an applicable plan would answer accordingly (i.e. with a plan structure with the required body component: $P :< E, C, B >?$). Otherwise, a false ($F$)value is replied to the learning agent.

3. Then, once the learning agent receives a plan, it proceeds adding the newly acquired plan to its set of plans. Continuing its reasoning cycle, it takes the applicable option provided by the new plan: moving

forwards.

Then, similarly to the case of IL, the challenge in CIL is to define a learning method that accounts for:

1. Determine when learning should be triggered.
2. Determine the learning method for acquiring plans based on:
    - Agent's intentional attitudes (i.e. beliefs, desires, intentions).
    - Interaction with other agents for knowledge exchange.

The knowledge to be shared includes beliefs, particularly information related with the perceived environment, and entire plans.

## 3.2 Intentional Learning in a Fully-BDI environment

The original BDI architecture lacks of learning capabilities. IL is aimed for filling this gap. Several works have approached IL just at the level of plan applicability, using some machine learning strategies, mainly based on logical tree induction algorithms [48, 49, 56, 111], or learning goals [90], even implementing a pure reinforcement learning approach [11].

The IL framework on the plan level has been implemented, in two ways [117]:

- By developing a BDI layer upon NetLogo, and
- On a hybrid NetLogo-BDI architecture (using JAM as the BDI engine).

However, a straightforward implementation has not been pursued. In the first case, not fully BDI agents were used. In the second case, the BDI agent is embedded into a reactive agent (i.e. the reasoning cycle is entangled with that one from the reactive agent). This situation makes it difficult to exploit the BDI architecture's abstractions tools and its full reasoning cycle.

On the other hand, in [95, 96] a very basic BDI implementation in pure NetLogo programing language is reported, mainly focused on teaching the BDI MAS foundations. This approach includes communication capabilities and a straightforward way to implement intentions (as a list of commands). However, being a basic implementation, it lacks a proper definition of plans, and other facilities offered by the BDI oriented platforms [95]. Additionally, it was not conceptualised to include learning.

### 3.2.1 The Jason BDI Agency

The earlier work on IL, used BDI-like architectures in both the single agent and in the MAS scope. Consequently, IL and CIL in fully BDI environments are still missing, particularly in the level of plan acquisition (i.e. the generation of new plans guided by a learning process).

Jason [17] is a fully fledged multi-agent programming environment based on AgentSpeak, an extension of logic programming to implement the BDI agent-based architecture [16, 93]. As such, it offers a concrete framework for implementing BDI agents and Multi-Agent Systems (MAS), making direct use of mentalistic attitudes. In Jason, knowledge of the agents is encoded through beliefs expressed as first-order logic predicates [17], allowing for higher-level representation and a rich expressiveness. A detailed description of the language can be found in [16, 93]. In brief, the main elements of AgentSpeak language are listed as follows:

```
ag ::=  bs   ps                         // agent definition
bs ::=  at_1 . ... at_n.       ( n >= 0) // belief base, variables not allowed
at ::=  P( t_1 ,..., t_n )     ( n >= 0) // atomic formulas, P=predicates,
ps ::=  p_1...p_n              ( n >= 1) // plan library
p  ::=  te : ct <- h .                  // plan
te ::=  +at | -at | + g | - g           // triggering event
ct ::=  true | l_1 &...& l_n   ( n >= 1) // context
h  ::=  true | f_1 ;...; f_n   ( n >= 1) // plan body
l  ::=  at | not at                     // literal
f  ::=  A(t_1,...,t_n) | g | u ( n >= 0) // body formula, A = action symbol
g  ::=  ! at | ? at                     // goal of achieve or test
u  ::=  +at | - at                      // addition or deletion of beliefs
Where, t is a first-order logic term
```

Being a logic-based formalism, in AgentSpeak, all the elements of the language are Prolog-like first-order literals (functions and predicates) or terms (constant terms, like numbers or names) [16, 93].

Since its inception, Jason has been extended, for it offers many features that make it amenable for developing a wide range of systems and applications. For instance, Java classes are used to extend the agent capabilities (known as *internal actions*), and interfacing packages to link the BDI engine with other systems. This also includes even the potential of linking and reusing legacy code. Although agents and MAS can run purely within the Jason framework, there is the potential to develop complex environments (including user interfaces) in a different platform or programming language, and interface the BDI agents created in Jason with the environment.

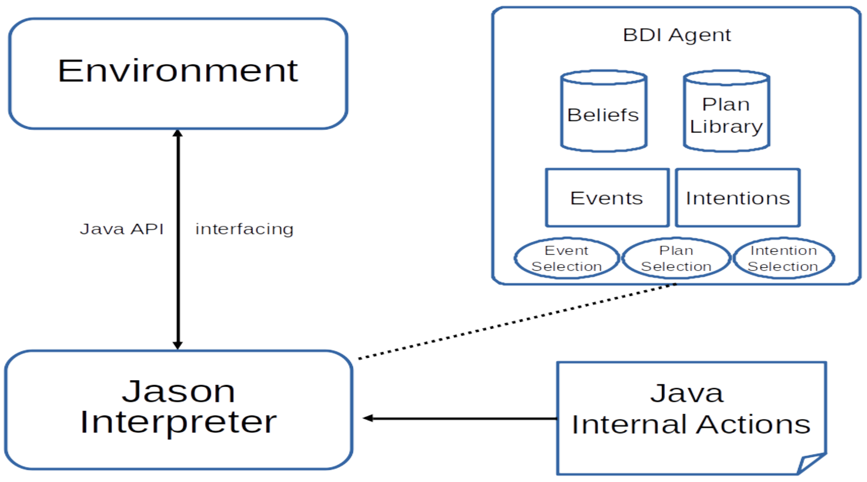


**Figure 3.9:** *The components of the Jason BDI framework.*

As illustrated in Figure 3.9, a BDI agent in Jason comprises of *Beliefs* (the agent's knowledge and perceptions), *Events* (the stimuli external or internal that the agent perceives), a *PlanLibrary* (the repertoire of possible courses of action that an agent is capable of,in response to events, expressed as *plans*), and *Intentions* (the commitment undertaken by the agent to act, in response to some event). The agent's reasoning cycle involves three functions:

- *Event selection*: an agent chooses a single event of those available through perceiving changes in the environment, or coming from the list of intentions due to beliefs being updated;
- *Option selection or applicable plan selection*: an agent chooses the course of action suitable for the chosen

event; and

- *Intention selection*: the agent chooses the next intention of a particular plan to be executed.

Then, an agent `ag` comprises a set of beliefs `bs` and a set of plans (the *PlanLibrary*) `ps`. Agents are executed in the Jason interpreter through this cycle of perception, deliberation and action. The reasoning cycle of an agent under this architecture starts with updating the list of events (internal or external). Then, event selection is performed, and a set of plans are chosen accordingly. Subsequently, one plan is selected to create an intention. Finally, the intention selection function determines the actual course of action to be undertaken (i.e. the execution of the actions defined in the plan).

As stated before, all the courses of action an agent has at its disposal are encoded in plans stored in the *PlanLibrary*.

Recalling the AgentSpeak grammar, a plan is defined as:

```
@label
triggering_event : context
            <-
               body
```

The `@label`, `triggering_event` and the `context` constitute the `head` of a plan. The literal `@label` is optional and identifies the plan in a unique way. The literal `triggering_event` defines the name of a plan and determines when it can be initiated. Hence, during the execution loop of the agent when an event (internal or external) matches this name, this plan is eligible for execution. There may be multiple plans that can have the same event as a trigger. Beliefs, and events are also expressed as first-order logic literals.

The `context`, formed by a conjunction of first-order literals, defines the applicability of the plan when given some condition or state of affairs within the environment, translated into agent perceptions or beliefs. Therefore, `context` specifies under what conditions a plan both applies and might be executed, and this may vary from plan to plan. As a result, an agent can have multiple plans with the same triggering event, which are differentiated by their `context`. Consequently, plans specify the means to achieving an intention, and the options the agent have to achieve this state of affairs [93].

The body of a plan is composed either of subgoals, which are calls to other plans (defined in the *Plan-Library*), or internal actions (defined in Java code). Goals, and sub-goals are preceded by a + symbol, and can be of two types: *test*, denoted by ?, used to consult the agent's beliefs; or *achieve*, denoted by !, used to actually be adopted as an intention, and once it is achieved, the plan resumes. Finally, Jason allows beliefs updating in the body of a plan, in other words, a plan can also delete (-) or add (-) new beliefs.

For instance, in Figure 3.10 a snippet of Jason code (*AgentSpeak*) is shown. This code defines some initial beliefs, and declares a set of plans that implements three of plans from the plan library introduced in Figure 3.3 of the previous section.

```
/* Initial Beliefs */
my_shape(ant).
my_color(black).
...
/* Plans           */
@TLeft                               // Plan's Label
+!move : left-cell-free              // Plan's head (triggering_event : context)
  <-                                 // Denotes the starting of the body
  .print("I'm going to turn left!"); // A Jason primitive to print messages
  !turn(90).                         // An achieve goal to be pursued

  ...

@Collect                             // Plan's Label
+!collect : food-on-current-cell     // Plan's head
  <-                                 // Plan's body starts
  +food-found;
  .print("Food found!");             // Message printing
  !pick-up.                          // An achieve goal to be pursued
```

**Figure 3.10:** *A snippet of code to define a BDI agent in Jason. Initial beliefs, and plans are shown.*

As defined in this code, each plan is triggered whenever an event !move or !move is fired. The applicability of the plans is defined by a condition which in this case is defined by the beliefs left-cell-free and food-on-current-cell. A plan context can be defined as being always fulfilled if denoted by true (i.e. such a plan can be triggered every time). The body of the plan TLeft includes these sub-goals: a) the printing of a message; and b) the execution of another plan (i.e. a sub-goal) whose triggering event is !turn(90) and has its own definition. The definition of this goal, can be implemented as another plan, or as an *internal action* (i.e. a Java code) to actually perform the action in the environment, in this case, turning 90°. The @ symbol is the prefix to denote a label, which can be used as an optional plan identifier. Finally, the last plan (@Collect) performs the addition of the belief`food-found`.

As it can be seen in this example, Jason allows for a straightforward implementation of agents, due to the adoption of AgentSpeak as the language to define agents, while rigorously observing the BDI theory.

## 3.3 An Experimentation Framework for Intentional Learning

This thesis is focused on developing a method of IL and CIL for plan acquisition, and its implementation on a fully BDI framework. As agents are located in a given environment, consequently, the domain of application imposes some restrictions and difficulties to the learning process. The more complex environment, the more restrictions to look after. Therefore, environments beyond a simple test-bed, can be exploited to analyse and test learning in agents.

This section introduce the experimentation framework, where IL and CIL will be pursued. Additionally, it contributes to fill the gap on the development of MAS based on cognitive agents, through a socially relevant application: a Disaster-Rescue simulation based on a BDI MAS.

### 3.3.1 Implementation Platforms and Tools

According to [12, 55, 67] JADE, Jadex, NetLogo, and Jason are the most noteworthy platforms for MAS development (including BDI agents), either for general purpose systems, or when applied in complex context, such as those typically addressed by simulations of social and natural systems, which comprises the Disaster-Rescue Domain[1].

Based on that, in this work, Jason and NetLogo are the platforms and tools used for the development of learning agents. Firstly, in the single agent case, a simple environment with several configurations is presented. Secondly, in the MAS scope, learning is studied in the Disaster-Rescue Domain (described in detail in Section 3.4.1.

Both are Java-based free software with active communities of support and a large number of applications in many fields:

[1]A deeper comparison of the platforms is shown in Appendix C

- **Jason** is a platform for BDI agent development: it offers robustness, stricter implementation of the BDI principles, and the facilities of AgentSpeak as a higher level agent programming language. Additionally, it allows the definition of environments, either from legacy code or belonging to other systems. Version 2.4.0 [17]

- **NetLogo** is the platform for simulating the environment, and acts as the main interface between the simulation and the final user. It offers a diverse range of possibilities for simulating different kinds of reactive agents (such as those needed by the rescue domain), and it is easy to set up, model, and implement environmental features. Version 5.2.0 [132]

In addition, it is important to note that NetLogo is built under the approach of Agent-Based Modelling (ABM), a sort of MAS focused on directly representing the elements of a system, including their relationships and interactions over time. ABM is becoming the *de facto* tool to develop simulations applied to a wide range of domains and to explore and validate social theories [67, 92, 97].

Integrating both approaches, ABM and MAS, boosts the resultant system by combining the modelling capabilities and more complex behaviours, including some aspects of human behaviour in the case of BDI agents. The main advantages of integrating Jason and Netlogo are outlined:

- NetLogo is empowered by the deliberation process of BDI Jason agents, that enhances the capabilities of the purely reactive NetLogo agents going beyond pure quantitative heterogeneity (variations in the values of their parameters and variables).

- Jason can be enriched in at least two aspects:
  - A powerful tool for rendering environments, and the capability to easily interact with them through the avatar-like Netlogo agents (known as *turtles*).
  - Inherits the skills and functionalities already present in the turtles, useful for defining lower-level tasks, for instance navigation and sensing.

- Both tools can be combined with other systems to manage more than one user-defined environment.

The capability to define reactive agents besides BDI agents working altogether in the same system makes it possible to combine features and abilities belonging to both kind of agents. In Figure 3.11 this composition is depicted. As can be seen, a Jason agent that is provided with BDI plans can capitalize on the reactive functions

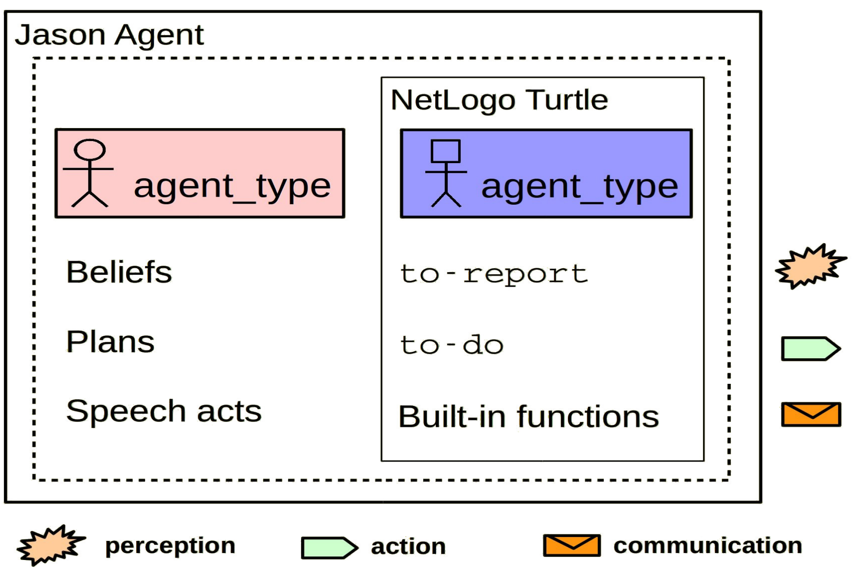


**Figure 3.11:** *Jason agents and NetLogo turtles enrich each other: executing BDI plans results in calling NetLogo functions to act, to perceive or to inter-communicate.*

implemented in its associated NetLogo turtle: `to-do` to act directly in the environment; and `to-report`, to perceive the current state-of-affairs, resulting in an update of its beliefs and possibly triggering other plans. Both agents can make the best use of the set of methods and functions defined in their corresponding framework, including their communication systems, where their higher- and lower-level representations (defined by speech acts in Jason and built-in functions in NetLogo) can be employed as required.

The integration of both type of agents conveys the flexibility to define and represent the elements and entities of a simulation. For instance, there is a need of having purely reactive agents to represent environmental elements, like fire. Simultaneously, agents with more robust deliberative processes are required to represent human individuals/organizations, like fire-fighters).

Table 3.1 summarizes the advantages of interconnecting Jason and NetLogo (J+N) contrasted with both tools taken in isolation such as interfacing external environments to Jason, or NetLogo's plot and statistics generation (any feature can be used separately or in a combined way in J+N).

At the implementation level, the connection of both platforms is granted by their API interfaces, and the facilities they offer. On the one hand, Jason allows the inclusion of Java-based environments to interact with the BDI agents. On the other hand, NetLogo allows the possibility to interact with its agents and environment

| System | Reactive agents | Cognitive agents | Plug external environments | Counterpart agent representation | Legacy code | Plotting and statistics |
|---|---|---|---|---|---|---|
| NetLogo | ✓ | | | | ✓ | ✓ |
| Jason | | ✓ | ✓ | ✓ | ✓ | |
| **J+N** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

**Table 3.1:** *Main advantages of integrating Jason and NetLogo.*

through its own GUI and from the outside.

Additionally, the Jason allows to create and execute agents into another system adding to their deliberation processes the reasoning BDI engine only. Nevertheless, the approach of this work accounts for maintaining the BDI agents (including its reasoning cycle) as they are, but enhancing its perceiving and acting possibilities through taking advantage of a NetLogo reactive agent allocated in the environment, as described below.

There are two possibilities to pursue this connection. In each case NetLogo is intended mainly as the environment:

1. **Jason-in-NetLogo**, includes Jason, using just the BDI engine in a NetLogo model. To be included into Netlogo, Jason allows to extend the agent architecture used to create the Jason agent and manage its reasoning cycle, as it is managed by the NetLogo turtles (i.e. turtles trigger the reasoning cycles in its Jason counterparts). The operation of this option is outlined in Algorithm 2. In brief, the following actions are performed:
   (a) From the NetLogo side, after executing the NetLogo GUI with the loaded model, the system creates the turles and other elements of the environment (line 2), embedding the Jason BDI reasoning engine in turtles as required (line 4).
   (b) In each reasoning cycle, each turtle behaves as usual (line 13), if having associated a Jason BDI engine (line 9), turtles send perceptions to it (line 8), receiving back an action to be performed in the environment (line 11) as a result of the BDI reasoning loop.
   (c) For sending perceptions and receiving actions from the environment to the Jason agent and *vice versa*, a process of translation from the NetLogo list-based representation to the *AgentSpeak* Prolog-like representation have to be done (lines 10 and 12).
   (d) The, turtles execute their intended actions based on the previous deliberation process (line 14).
2. **NetLogo-in-Jason**, includes Netlogo in Jason using the former as the environment. The inter operation

is based on extending the class in charge of manage environments in the Jason architecture, and the NetLogo facilities to control externally the agents defined in the model. Algorithm 3 outlines how this option works. This inter connection is described as follows:

(a) The Jason MAS starts and create a NetLogo model as its defined environment (line 1). Both platforms are running together, but the control is on the side of Jason.

(b) Whenever an agent requires perceiving or acting, it fires a new NetLogo cycle (line 9), turtles are expected to receive orders from their Jason counterparts to act (line 19), providing feedback about the state of the environment on request (line 11), taken as perceptions for the Jason agents.

(c) It is possible to ask all the elements of the environment for information or request action. But, the latter is restricted to keep the consistency in the relation between Jason agents and turtles.

(d) The MAS creates its agents as usual. If any agent will require a representation in NetLogo, then the turtle is created accordingly.

(e) In this option, to interact with NetLogo, a translation process between AgentSpeak and NetLogo commands is pursued in Jason mechanism in charge of managing the environment interaction with agents (lines 10, 12 and 18).

#### 3.3.1.1 *The Jason-in-NetLogo* settings.

This option of interconnection was tested to try both MAS platforms. The first step in this process was to use a model from the NetLogo distribution model library, namely, `Fire.nlogo` [132].

In brief, it implements a set of turtles situated in an environment formed by random shapes that simulates trees in a forest, as seen in Figure 3.12 (i). The NetLogo environment starts by creating a Jason agent which acts as the reasoning engine of the turtles. The main purpose of the Jason agent is to order its corresponding turtle in NetLogo to move. This processing could be avoided, because the turtles can move without receiving the direction of any external agent, but the purpose was to test how the Jason agent could guide the behaviour of the turtles. This very simple example was sufficient to test the possibilities of control between Jason agent and NetLogo turtles. I Figure 3.12 (ii) the state of the NetLogo system can be seen. The turtle starts its behaviour by clicking the *setup* button, sending perceptions to its associated Jason agent which in turn

**Algorithm 2:** Jason-into-NetLogo

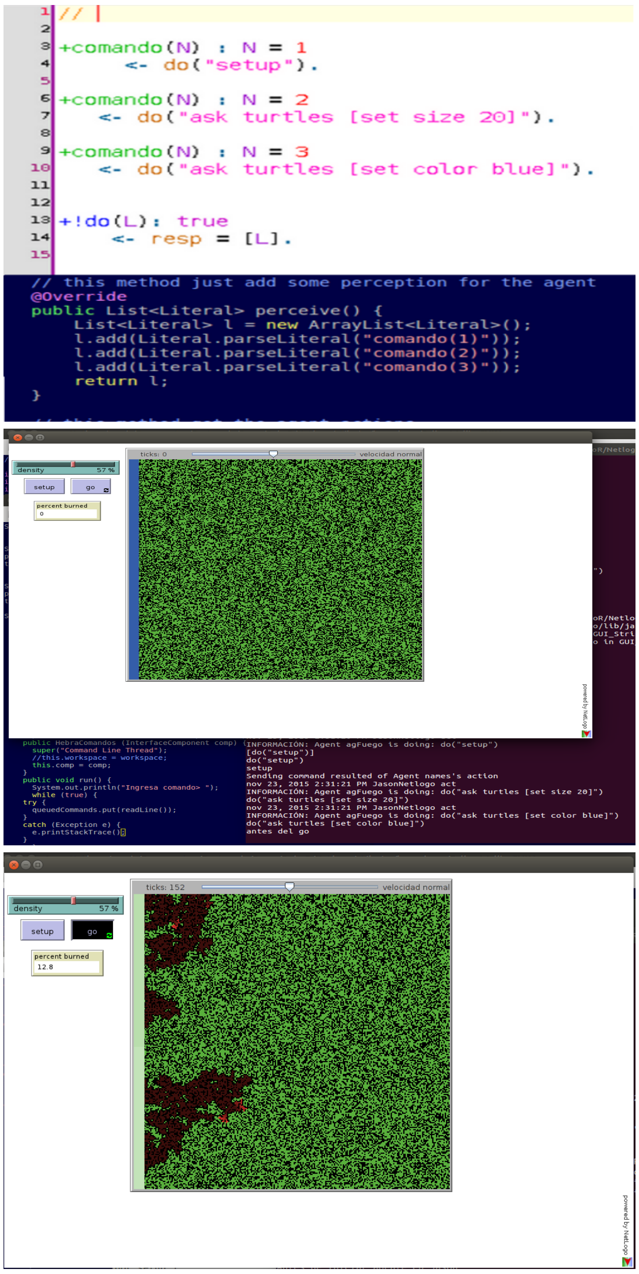
```
Data: NL: A NetLogo model; J: A Jason BDI engine;
NLBDI: The set of agents in the MAS that require use J to deliberate
P: The set of turtle perceptions; Pj: A set of turtle perceptions, encoded in AgentSpeak
a: A definition of an executable action; aj: A definition of an executable action, encoded in AgentSpeak
begin
       /* Start the NetLogo model, and render the environment */
1      NL.start()
       // Create all the turtles in the model
2      foreach $t \in T$ do
3          if $t \in NLBDI$ then
4              t.ja ← createJasonAgent(t.id)
           end
       end
5      while True do
           /* Starting a new cycle in the MAS */
6          NL.tick()
           // All turtles execute their reasoning cycles
7          foreach $t \in T$ do
               // Perception
8              P ← t.perceive()
               // Deliberation
9              if $t \in NLBDI$ then
                   // Turtle deliberate based on its associated BDI agent
10                 tp ← translate(P)
                   // Executing a BDI reasoning cycle with the perceptions so far
11                 aj ← t.ja.reasoningCycle(tp)
12                 a ← translate(aj)
               end
               else
                   // Turtle deliberate as usual
13                 a ← t.deliberate(t.P)
               end
               // Action
14             t.execute(a)
           end
       end
end
```

analyses the input, reasons about it, then orders its corresponding turtle to act accordingly. This behaviour is very simple; it does consist in changing the state of the trees (i.e. changing their colour from green, to red, then brown, representing a burning process), as shown in Figure 3.12 (iii).

This example represents a successful interaction between both platforms. By creating a model that can exploit the reasoning capabilities of both kinds of agents, from the reactive side (NetLogo) to the BDI deliberation (Jason), this interconnection becomes applicable to any problem involving MAS.

**Algorithm 3:** NetLogo-in-Jason

```
Data: NL: A NetLogo model; JMAS: A Jason BDI system;
JasonT: The set of agents in JMAS requiring a turtle counterpart
F: turtle features B: Beliefs; I: Intentions; D Desires; O: Options; E: Event queue p: A perceiving
request, encoded in AgentSpeak; pi: A perceiving request, encoded in NetLogo a: An executable
action; ta: A definition of an executable action, encoded in NetLogo
begin
       /* Start the Jason MAS                                                          */
       /* Create the NetLogo model, to render the environment                          */
1      MAS.NL.start()
       // Create all the Jason agents
2      foreach j ∈ JMAS do
3          if j ∈ JasonT then
4              j.t ← MAS.NL.createTurtle(j.id,F)
           end
5          j.initialise(B)
6          j.initialise(I)
       end
7      while True do
           /* Starting a new cycle in the Jason MAS                                    */
           // All BDI agents execute their reasoning cycles
8          foreach j ∈ JMAS do
               // Perception
               if j ∈ JasonT then
9                  MAS.NL.tick()
                   // Requiring to NL turtle to perceive and translate its resulting
                      perception
10                 j.tp ← translate(j.p)
11                 j.pi ←12j.t.perceive(tp);  j.E ← translate(j.pi)
                   // Executing a BDI reasoning cycle with the perceptions so far
13                 j.O ← j.t.perceive(E); // Belief Revision
14                 j.B ← j.beliefRevision(j.B,j.O)
                   // Planning
15                 j.D ← j.optionGeneration(j.B,j.I)
                   // Deliberation
16                 j.I ← j.intentionSelection(j.B,j.D,j.I)
                   // Action Selection
17                 j.a ← j.actionSelection(j.I)
                   // Translate the selected action and require to NL turtle to act
18                 j.ta ← translate(j.a)
19                 j.t.execute(ta)
               end
               else
                   // Agent deliberate as usual
20                 j.O ← j.perceive(e)
21                 ...
               end
           end
       end
end
```



```
// |

+comando(N) : N = 1
      <- do("setup").

+comando(N) : N = 2
    <- do("ask turtles [set size 20]").

+comando(N) : N = 3
    <- do("ask turtles [set color blue]").


+!do(L): true
     <- resp = [L].
```

```
// this method just add some perception for the agent
@Override
public List<Literal> perceive() {
    List<Literal> l = new ArrayList<Literal>();
    l.add(Literal.parseLiteral("comando(1)"));
    l.add(Literal.parseLiteral("comando(2)"));
    l.add(Literal.parseLiteral("comando(3)"));
    return l;
}
```

```
public HebraComandos (InterfaceComponent comp) {
  super("Command Line Thread");
  //this.workspace = workspace;
  this.comp = comp;
}
public void run() {
  System.out.println("Ingresa comando> ");
  while (true) {
try {
  queuedCommands.put(readLine());
}
catch (Exception e) {
  e.printStackTrace();
}
```

```
INFORMACIÓN: Agent agFuego is doing: do("setup")
[do("setup")]
do("setup")
setup
Sending command resulted of Agent names's action
nov 23, 2015 2:31:21 PM JasonNetlogo act
INFORMACIÓN: Agent agFuego is doing: do("ask turtles [set size 20]")
do("ask turtles [set size 20]")
nov 23, 2015 2:31:21 PM JasonNetlogo act
INFORMACIÓN: Agent agFuego is doing: do("ask turtles [set color blue]")
do("ask turtles [set color blue]")
antes del go
```

(i) Both pieces of code of the agents (AgentSpeak and Java) to receive, translate and act upon the perceptions from the NetLogo environment. The actions to perform are: setup the environment accordingly with the model, and a couple of commands for the turtles: set up the turtles shape to 20, and set their colour to blue.

(ii) The NetLogo running environment after receive the commands from the Jason Agent.

(iii) The system behavior after pressing the button go, as defined by the model.

**Figure 3.12:** *The Fire model used under the Jason-into-NetLogo option*

### 3.3.1.2 *NetLogo-in-Jason* settings

This option, besides being novel in ABM/MAS development, it is the natural way to implement our system to include IL. Figure 3.13 depicts the interaction diagram in UML. As can be seen, both systems interact with each other through their corresponding APIs, allowing the Jason agents to act within the environment using the NetLogo turtles, whilst updating their percepts through requesting the execution of actions and updates concerning the current state of the environment.

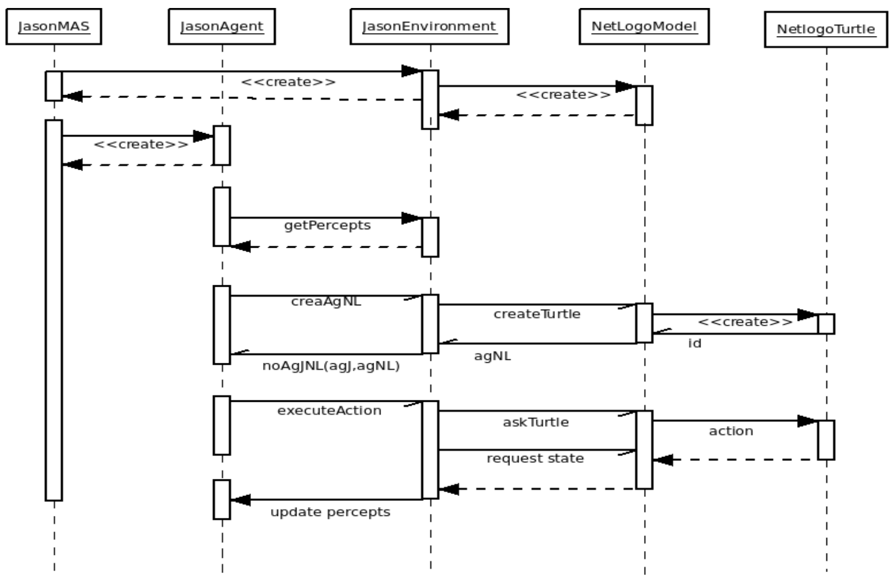


**Figure 3.13:** *Sequence Diagram in UML of the Netlogo-in-Jason interconnection option*

To send perceptions and receive actions from the environment to the Jason's agents and vice versa, a process of translation from the NetLogo list-based representation to the AgentSpeak Prolog-like representation have to be done in both of the integration cases aforementioned, as stated in Algorithms 2 (lines 10, 12) and 3 (line 10, 10, and 18). To achieve this, the following classes are required: `Agent`, `ActionExec`, `TransitionSystem`, `Literal` (contained in package `jason.asSemantics`). In Table 3.2 some examples of these translations are presented, relating to some sensing and acting commands in the case of a firefighter agent. The translation procedure is carried out by the Jason Java classes related to managing the environment. In brief, an AgentSpeak literal was introduced in the Jason side to denote when the agent is sending a command to the environment: `sendCmdNL/2` whose arguments are the agent ID and the intended command. The agent ID is a number identifying the turtle in NetLogo that acts on behalf of the Jason agent. The command is a string that identifies the action to be pursued, then is sent to the Jason Java class `Environment` and translated to the NetLogo final command.

The second option (*NetLogo-in-Jason*) allows to define NetLogo as the environment to allocate agents, both reactive and BDI. Furthermore, the control is on the Jason side, whilst those reactive agents not related to Jason agents (for instance, enviromental elements), stay in charge of NetLogo. This circumstance is favorable to this research in two ways: a) it leaves the BDI reasoning cycle with no change, helping in the study of IL in a diversity of domains; and b) it represents a straightforward way to simulate the disaster location to

| Description | Jason literals (AgentSpeak) | NetLogo commands |
|---|---|---|
| | Perceiving the environment | |
| *Jason-in-NetLogo*<br><br>Detect if a fire is found in the trajectory of a given agent | Sensing request literal:<br><br>`sendCmdNL(AgNL,"senseFire");` | Get the report from NetLogo<br><br>`[detect-one-fire-cone] of turtle AgNL` |
| *NetLogo-in-Jason*<br><br>Perceive the response from the environment about finding fires. | Translating the answer into literals:<br><br>`removePerceptsByUnif(agName, Literal.parseLiteral("fireFound(ID)"));`<br>and<br>`addPercept(agName,Literal.parseLiteral(" fireMarkedFnd(ID)"));` | |
| | Acting in the environment | |
| To walk in the environment, one step forward the agent's front | `sendCmdNL(AgNL,"fd 1");` | Order a turtle to walk<br>`ask turtle AgNL [fd 1]` |

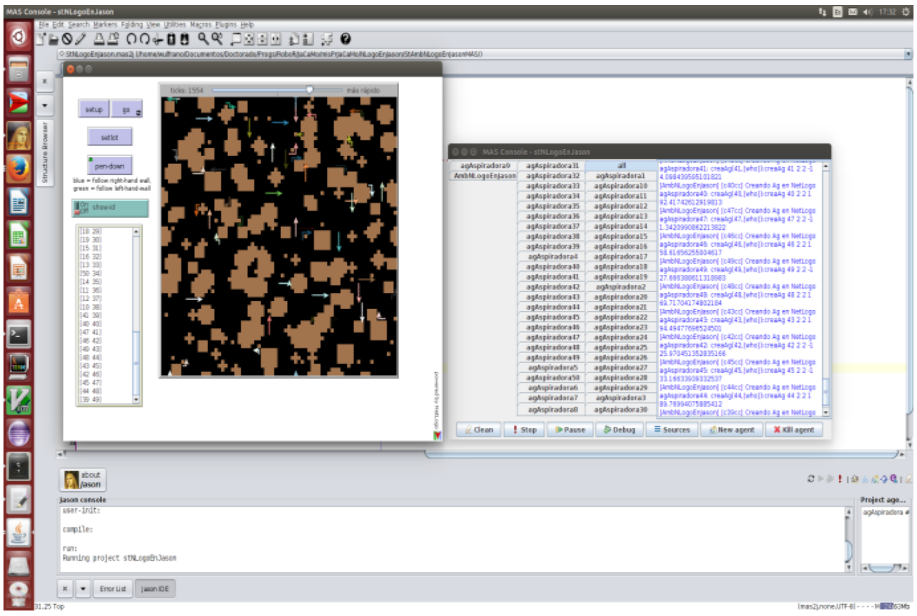
**Table 3.2:** *Examples of the translation of perceptions and commands from Jason to NetLogo and vice versa. NetLogo identifiers:* `AgNL`*: a turtle,* `ID`*: a fire, and* `agName`*: the corresponding Jason agent.*

develop a BDI MAS applied to the Disaster-Rescue Domain.

The first stage to test this interconnection option was to set a proof of concept. A couple of examples were developed by coupling both platforms. The first one aimed at controlling the turtles from Jason, using NetLogo as the environment, allocating the agents in a simple wall-following environment. The second one regards the implementation of IL upon the described framework taking a vacuum-cleaner environment.

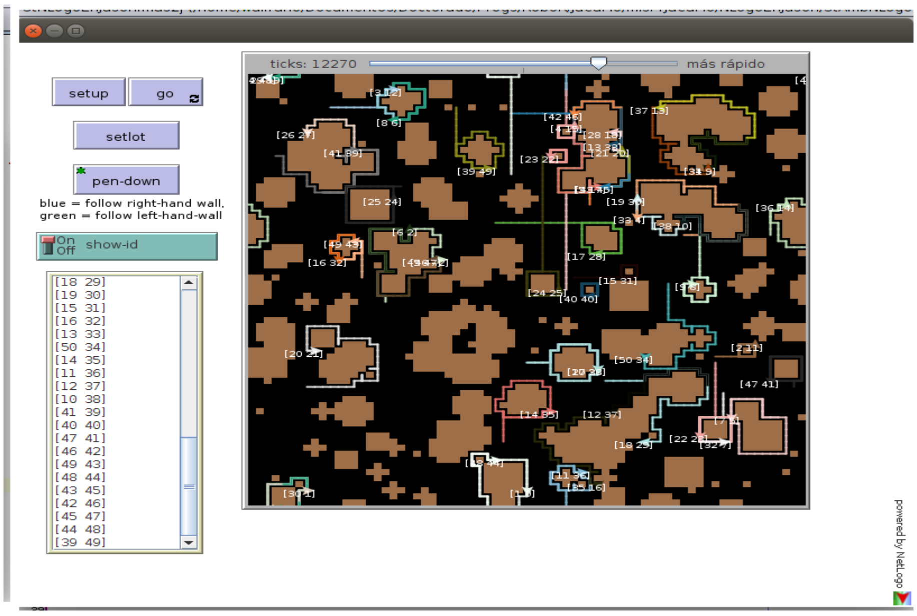
### 3.3.2 A Simple Model: Wall-following Agents

Left-handed wall-following was the first step to try the *NetLogo-in-Jason* option. Briefly, it was an experiment where each of the NetLogo agents (turtles) represents a kind of slave, or avatar, of the Jason BDI agents, using their capabilities to sense the environment, and inform about those perceptions when required. The environment chosen was the `Wall-following` example model from the NetLogo *Code Examples* library. Briefly, it implements a set of turtles situated in an environment formed by random shapes, depicted through arbitrary polygons with angles of 90°. The result of this step is presented in Figure 3.14: in a) the Jason GUI is shown starting the NetLogo Environment, and the turtles acting on behalf of the Jason Agents; in b) a screenshot of the execution with 50 agents is shown with ID labels.



```
user-init:

compile:

run:
Running project stNLogoEnJason
```

(a) 50 Jason Agents starting their behaviour in a NetLogo environment



```
[18 29]
[19 30]
[15 31]
[16 32]
[13 33]
[50 34]
[14 35]
[11 36]
[12 37]
[10 38]
[41 39]
[40 40]
[47 41]
[46 42]
[49 43]
[48 44]
[43 45]
[42 46]
[45 47]
[44 48]
[39 49]
```

(b) A labelling of the corresponding agents ids in Jason and in NetLogo

**Figure 3.14:** *The wall-following model used under the NetLogo-in-Jason option.*

### 3.3.3 A Vacuum Cleaner Test-Bed

Once the suitability of the interconnection between both frameworks was established under the **NetLogo-in-Jason** aforementioned option, the next step was to set the test-bed to implement IL in this framework.

In order to implement the *NetLogo-in-Jason* option, the first issue was to implement a basic NetLogo model to represent an environment, described as follows:

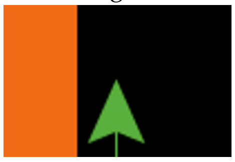
- Environment: A 11 x 11 grid world, partially accessible and static, populated with targets, obstacles and one vacuum cleaner agent.
- Description: the well-known vacuum world, where a situated agent either collects as many targets as possible whilst avoiding obstacles (the cleaning task); or performs a movement, following the walls in a left-handed direction.
- Agent: BDI agent with limited perception of the environment, with restricted movement (only via the 4-connected way, i.e. cells at front, back, left or right, if available).
  - Perceptions: the agent perceives the environment as an array of cells in front of them with a certain depth of sight.
  - Actions: collect targets whilst moving through the environment, using a set of three **primitives** to move forward, and turn left and right

The main perception is a vector such as: $[(o1, d1), (o2, d2), (o3, d3), (o4, d4), (o5, d5)]$ Where literals $oN$ and $dX$, represent the number of object situated in the position $N$, at the distance of $X$ cells.

To represent this perceptions in Jason, the arrangements shown in Figure 3.15 were performed.

```
 o2   | o3 |o4
--------------
 o1   | Ag |o5
```

In NetLogo:



Represented as Jason Literal:
`[o1(d1),o2(d3),o3(d3),o4(d4),o5(d5)]`
Where: $dx$ is a color in the environment
$x = 1..5$ for each $d1..d5$
Considered only $d = 1$:

No object: Background color

Background `color = 0` (black in NL)

Obstacle `color = 25` (orange in NL)

Target `color = 55` (green in NL )

**Figure 3.15:** *The Wall-Following Test-Bed.*

The implementation was carried out by using the design shown in Figure 3.16. As can be seen, the agent is situated in a NetLogo environment, using the *NetLogo-In-Jason* option described previously.

The list of perceptions, actions and scenarios is shown in Table 3.3.

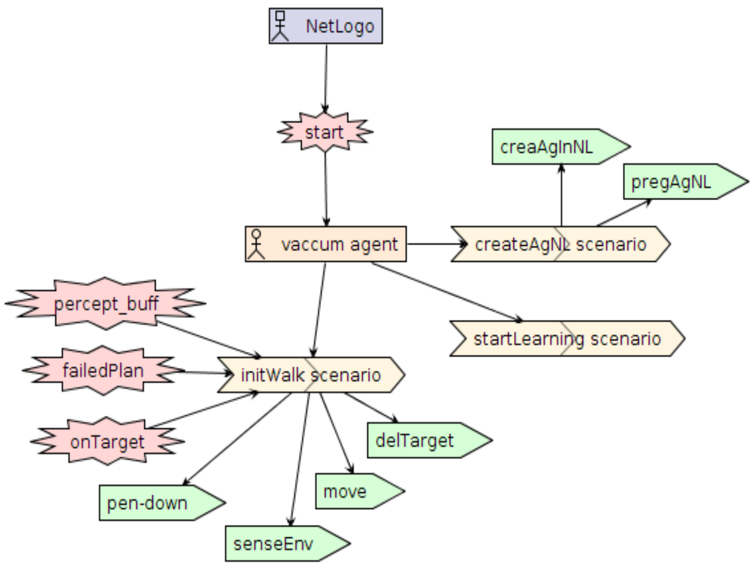


**Figure 3.16:** *The design of the learning agent situated in a NetLogo grid-like environment to display the vacuum-cleaning scenario.*

Once started, the environment sends the `start` perception to the Jason agent, that configures the command to create the turtle in the NetLogo environment, and begins the intended task (target-searching or wall-following). In other words, the agent is wandering throughout the environment using its plans to move, detect and collect targets, or follow walls to the left. The perception `failedPlan` triggers the scenario `starLearning`, the IL procedure.

For the wall-following behaviour this test-bed includes three different mazes, as illustrated in Figure 3.17 (Though, any other maze can be configured easily).

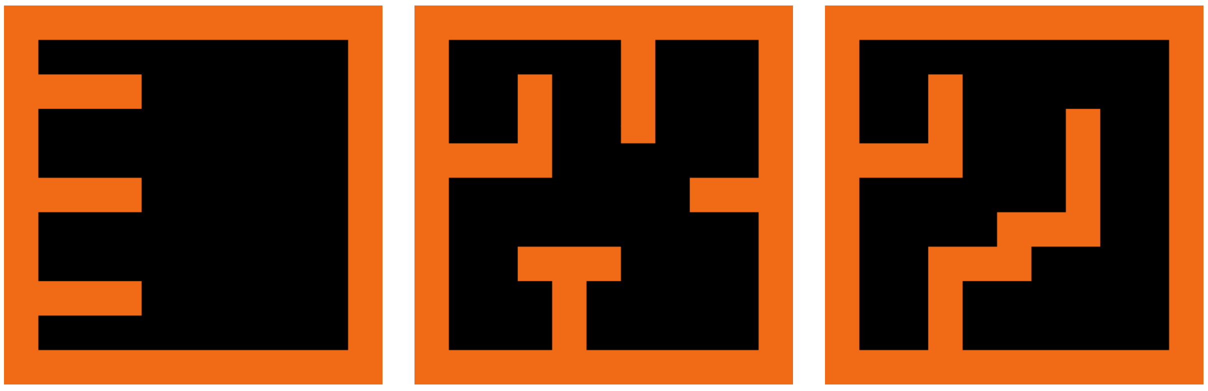


**Figure 3.17:** *The three mazes to test the single vacuum cleaner agent in the wall-following behavior.*

This test-bed provides another maze, to test the target-searching behaviour as depicted in Figure 3.18. Apart from barriers in the middle of the scenario, this environment has a *positive target* and a *negative target*

| **Perception** | **Description** |
|---|---|
| `start` | Initiates the agent in the environment triggering the `initWalk` scenario |
| `percept_buff` | Array of cells in front of them with a one cell ahead depth of sight |
| `failedPlan` | Detects when a plan has failed and triggers the the scenario `startLearning` |
| `onTarget` | Indicates whether the agent is currently on a target |
| **Action** | **Description** |
| `creaAgInNL` | Creates the NetLogo turtle which is acting in behalf of the Jason agent in the environment |
| pregAgNL | Asks for the id of the NetLogo turtle, and posit the belief `noAgJNL(NoAgJ,AgNL)`, where `NoAgJ` is the number of the Jason Agent, and the AgNL is the number of the NetLogo turtle |
| `pen-down` | Asks the turtle in NetLogo to draw a trace of its movements |
| `senseEnv` | Asks the turtle in NetLogo to retrieve the sensing of the environment according to the vector `sense_buff` |
| `move` | Command the agent to move in the environment using its primitives |
| `delTarget` | Asks the turtle in NetLogo to delete the turtle representing a target |
| **Scenario** | **Description** |
| `initWalk` | Allows the movement of the NetLogo agent in the environment by repeatedly firing the `move` command |
| `createAgNL` | Generates the configuration to create the avatar turtle in NetLogo. Set parameters like shape, color, position, and size by calling to `creaAgInNL` |
| `startLearning` | Java-based *Internal action* to initiate the IL procedure |

**Table 3.3:** *Listing of perceptions, actions, and scenarios performed by the agent in the environment.*

in order to teach the agent to discriminate between paths when it is intent upon the*positive target*.

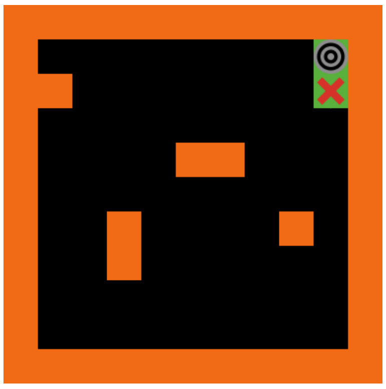


**Figure 3.18:** *The scenario to test the single vacuum cleaner agent in the target-searching behaviour.*

## 3.4 The Disaster-Rescue Domain

Previous points described the suitability of the interaction between NetLogo and Jason in the *NetLogo-in-Jason* option. This opened the door to set up a Disaster-Rescue (D-R) model in this framework, for developing the simulation. As previously explained before, NetLogo renders and manages the environment, and some of the agents are represented in Jason and, or as, NetLogo turtles.

The D-R domain of application is an excellent test-bed to MAS. The features of this domain facilitates the creation of simulations where policies, behaviours and algorithms of coordination, task allocation, and coalition formation can be continuously evaluated.

### 3.4.1 The Disaster-Rescue Problem

In [55] an emergence management cycle is presented, which is composed of four phases: Mitigation, Preparedness, Recovery and Response. Agent Based Simulation tools are helpful in evaluating, planning and predicting real-life situations. As they provide an in silico experimental scenario for educational, training, understanding and decision support they can be used mainly during the phases of Preparedness (activities to improve the response actions) and Response (the actual actions to tackle the emergence, like fire extinguishing), [55]. Concretely, Agent Based Simulation tools provide the following contributions on the aforementioned stages [55]:

- Planning: determining the impact of a disaster event identifying the ways to be prepared and how to respond.
- Vulnerability analysis: evaluation of the strategies of emergency response.
- Identification and detection: determining the occurrence possibility of a disaster.
- Training: training emergency services staff by simulation
- Real-time response support: determining alternative response strategies given an emergency situation

In order to develop such a simulation, in the literature has been proposed a *Disaster-Rescue MAS Problem* that can be described as follows:

> Given an environment that simulates a city or location under disaster conditions (earthquakes, fires, etc.) a population is represented as a set of agents, divided into victims or civilians, and rescue agents. The rescue agents help the victims to evacuate and avoid risks. The rescue group of agents have to coordinate their actions to minimize the damage, casualties and keep the civilians

in safe conditions, possibly by joining forces, under uncertain and changing conditions where the time is an important restriction.

This is a coordination problem that has been commonly treated as a Task Allocation (which includes some variants as Social Welfare Maximization, and Coalition or Team Formation). This problem includes the implementation of a strategy to organise rescue teams (namely rescue brigades, ambulances, or fire brigades). The main activities are:

1. Mobility in the disaster area looking for victims
2. Perform rescue tasks
   (a) Detection of people in danger, fires and street blockades
   (b) Temporary save people in danger (Release trapped people; provide medicine, water, or other supplies; helping people to evacuate from buildings, etc.)
   (c) Collecting and transporting people in danger to a safe place
   (d) Fire controlling and extinguishing
   (e) Obstacle clearing

General characteristics:

- Partially accessible and dynamic environment, with time-restrictions, able to be traversed in a restricted way
- Partial communication
- Continuous space of states
- Common/shared goals
- Heterogeneously skilled Reactive and Cognitive agents involved
- Hierarchical organisation needed: teams, coalitions, network, etc.
- Solution shaped as task allocation

- Low/large size population (up to 200 agents or more)

Macro-Micro behaviours:

- Micro/Lower level (individual agent) behaviour: keep itself safe, follow a protocol (plan) to keep safe and rescue people, and perform other rescue tasks, repulsion or attraction movements.
- Macro/Higher level (global MAS) behaviour: Decentralised control (coordinate the teams) to generate and perform efficient policies for emergency services, i.e. pursue the rescue tasks, while minimize the casualties/injuries (people/rescue teams members) within the time restrictions and using the available resources.

This domain of application is an excellent test-bed to MAS where policies, behaviours and algorithms of coordination, task allocation, and coalition formation are continuously evaluated. One of the most known benchmarks for rescue teams' implementation can be found in the RoboCup Rescue Multi-Agent Simulation (RBCR), among others as can be seen in Appendix C.

### 3.4.2 A Disaster-Rescue Model

The environment was configured inspired by the work presented in [3,95,96] making use of the `gis` extension included in the NetLogo distribution to render a map of Mexico City centre, particularly the main square known as "El Zócalo", but other maps could be used (as explained in the NetLogo section of Appendix C).

The elements of the environment are divided into breeds of turtles, namely ambulances, civilians, rescue units, fire brigades and police, and they represent the NetLogo agents acting on behalf of the Jason agents. Additionally, there are turtles representing fire, obstacles and a base (elements of a disaster environment) with behaviours not governed by Jason Agents. In Figure 3.19, a screenshot of the model with legends of the agents and elements is shown.

The visual representation of the map, uses a PGM format image, transformed into ASC format as required by the `gis` library. Both formats store the image as a matrix of ASCII characters with slightly different structures. Then, the map is pre-processed by the NetLogo model to segment the buildings and the streets, allowing the agents to only traverse the environment via streets. To easy usability, the map is superposed onto

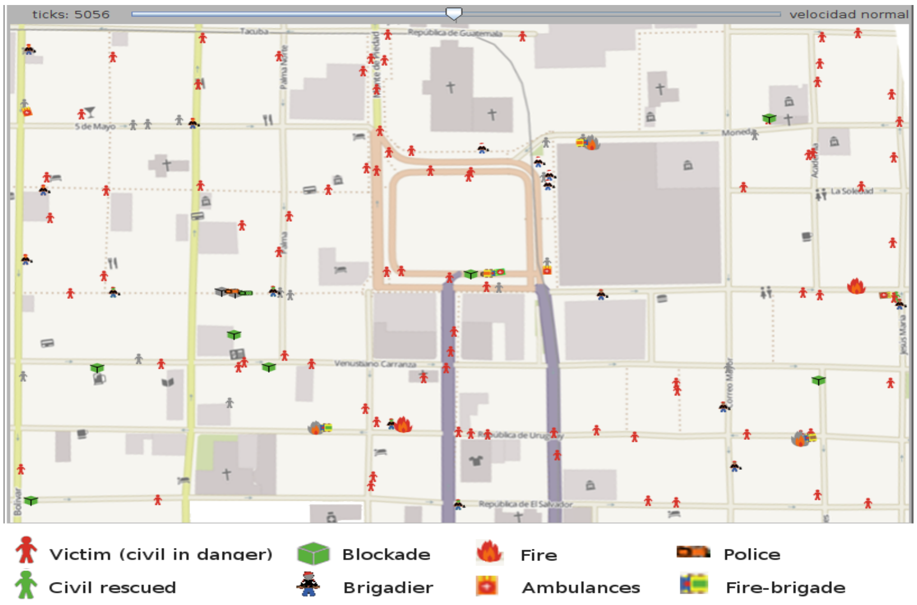


**Figure 3.19:** *The model representing the Mexico City main square and the elements of the Disaster-Rescue domain used to develop the simulation*

the pre-processed map. A screenshot showing the segmented image of the model is portrayed in Figure 3.20.

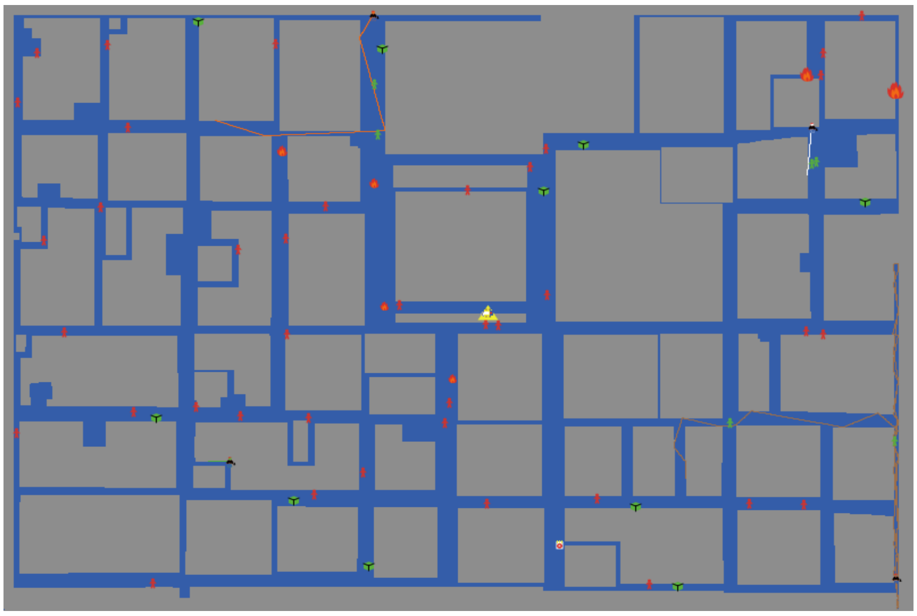
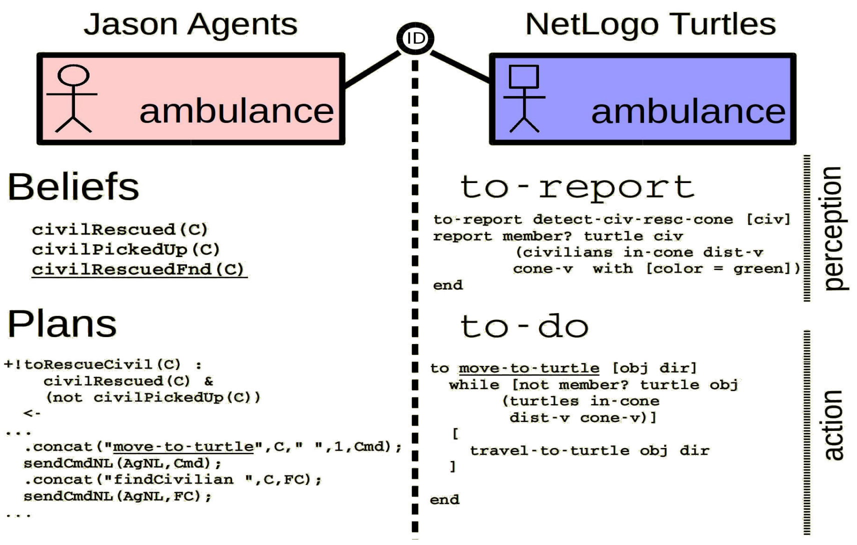


**Figure 3.20:** *The pre-processed map, it is actually used by the agents in the model to traverse the environment only through streets, represented by the blue zones.*

Furthermore, Figure 3.21 illustrates the integration *NetLogo-In-Jason* through the example of how to exploit the reactive and cognitive behaviours of turtles and Jason agents respectively. The plan `!toRescueCivil(C)` belonging to a Jason agent `ambulance`, takes advantage of the reactive capabilities of its corresponding turtle by calling the function `move-to-turtle`, to locate and head towards position of a detected victim denoted by *C* (i.e. the ID of the turtle represents the victim). The reactive function in its turn calls another function named `travel-to-turtle` that uses the native NetLogo functions available to move towards the position of a given turtle (*obj* takes the value of *C*). The result of `detect-civ-resc-cone` will cause a beliefs update, by adding `civilRescuedFnd(C)`.



```
civilRescued(C)
civilPickedUp(C)
civilRescuedFnd(C)
```

```
+!toRescueCivil(C) :
    civilRescued(C) &
    (not civilPickedUp(C))
  <-
...
  .concat("move-to-turtle",C," ",1,Cmd);
  sendCmdNL(AgNL,Cmd);
  .concat("findCivilian ",C,FC);
  sendCmdNL(AgNL,FC);
...
```

```
to-report detect-civ-resc-cone [civ]
report member? turtle civ
          (civilians in-cone dist-v
          cone-v  with [color = green])
end
```

```
to move-to-turtle [obj dir]
  while [not member? turtle obj
         (turtles in-cone
          dist-v cone-v)]
  [
    travel-to-turtle obj dir
  ]

end
```

**Figure 3.21:** *Jason agents act and perceive by means of their turtle counterpart in the environment. The BDI plan* `!+toRescueCivil(C)` *triggers the execution of function* `move-to-turtle`*, providing the required input (i.e. the ID of a victim stored in C). The NetLogo function* `detect-civ-resc-cone` *will result in adding the belief* `civilRescuedFnd(C)`*. Underlined text is just for an easy reading.*

With this basis, the next step was to perform the Jason-NetLogo connection under the *NetLogo-in-Jason* option, accounting for a basic design of the BDI MAS and the environment.

### 3.4.3 The Disaster-Rescue Simulation with a BDI MAS

This section endeavours to fulfil Research Objective 2 "Developing a BDI MAS based on the concepts of EM and SO in its design".

Given the Disaster-Rescue model, the interconnection between NetLogo and Jason platforms described before, and the framework introduced in Section 3.3, a basic version of a BDI MAS (without learning capabilities applied) applied to the Disaster-Rescue domain was developed.

The design of this basic version of the BDI MAS was performed using the editor Prometheus Development Tool (PDT) [84], as it has conveniently used in Jason development [15], outlined with greater detail in Appendix B.

The system implements a basic policy of task allocation, by dividing the map into four sectors where the agents can operate, and integrating the teams according to the number of agents. This means each sector contains, as much as possible, the same number of agents, as depicted in Algorithm 4.

These directions are integrated in the start plan of each agent as an initialization stage, just prior to the creation of the avatars in NetLogo.

**Algorithm 4:** Map partition in *N* sectors and team formation accordingly *N*.

```
Data: Take the dimensions of the map: as defined in NetLogo:
N: the number of sectors
AgTypes: list of type of agents in the MAS
Result: Sector assigned to each agent in the MAS
begin
      /* Assign the sector to each agent in the MAS */
1     foreach $type \in AgTypes$ do
2         $nAgent \leftarrow 0$
3         foreach $ag_{type} \in MAS$ do
4             $nAgent{+}{+}$
5             $nSector \leftarrow 1$
              if $nAgent > N$ then
                  $nSector = nAgent \bmod N$
              end
6             $ag.Sector = nSector$
          end
      end
end
```

The system behaves in a decentralized way, as no any agent leads the teams or the activities of the others, aside from noticing targets in charge of the rescue-units agents. In this manner, the higher level behaviour can be characterized by the performing of search and rescue stages.

For simplicity, the measure of the global behaviour of the system consists of the counting the rescue actions, and the number of actions performed in the environment. In this manner, to indicate the Performance (*P*), only a plot of the number of rescue actions (*RA*) respective to the total number of actions (*PA*) is provided in

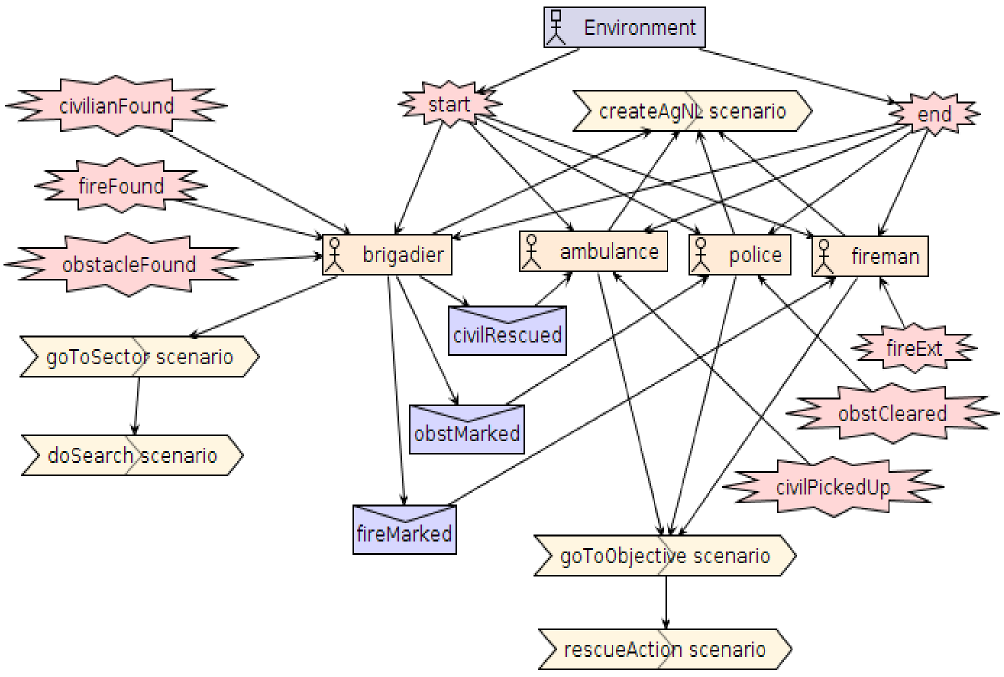


**Figure 3.22:** *Analysis Overview Diagram of the BDI MAS for the D-R Domain. For the sake of clarity, the actions are not depicted in this diagram*

the GUI, as described by Equation 3.1.

$$Pi = \frac{RAi}{PA}$$
$$Where,\ i = \{rescued \mid collected \mid extinguished \mid cleared\} \tag{3.1}$$

In Figure 3.22, the Analysis Overview Diagram (edited in PDT), which represents the global design of the DR BDI MAS, included the interaction with the environment, the agents belonging to the system, and the perceptions that arose from either the environment or message exchange between BDI agents. The environment fires the agents by sending the `start` perception. Then the agents initiate the `createAgNL` scenario. After that, each agent performs its own behaviour, differentiated by agent type, the environment and the messaging between other agents.

In this version, the system ends its function when all civilians are rescued and collected, all fires extinguished, and obstacles cleared, or when a threshold of number of actions is reached. A description of the scenarios is presented below.

As can be seen in Figure 3.22, the system comprises the following elements:

**BDI agents**

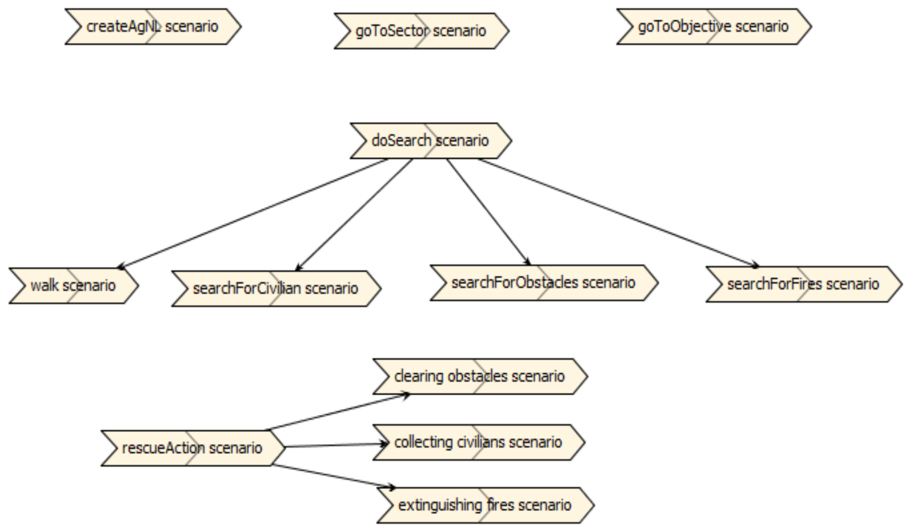

createAgNL scenario
goToSector scenario
goToObjective scenario
doSearch scenario
walk scenario
searchForCivilian scenario
searchForObstacles scenario
searchForFires scenario
clearing obstacles scenario
rescueAction scenario
collecting civilians scenario
extinguishing fires scenario

**Figure 3.23:** *A scenario diagram showing the main processes of the D-R BDI MAS.*

- brigadier: Representing the rescue unit agent, it is in charge of searching for civilians, rescuing them, advertising to firemen and police about street fires and obstacles.
- ambulance: Representing the ambulance agents, which simulates an ambulance team, responsible of collecting rescued civilians (once brigadier agents identify the location of civilians in need of rescuing).
- police: Representing the police unit agent, it is in charge of clearing obstacles (once brigadier agents report their location).
- fireman: Representing the fire brigade agent, it is in charge of extinguishing fires (once brigadier agents report their location).

**Main scenarios**

*Scenario* stands for an abstract description of a sequence of steps within the system. Figure 3.23 shows a diagram with the scenarios of the D-R BDI MAS edited in PDT.

With each process of sensing, or acting, in the environment, the Jason Agent acts via its turtle, which performs the actual action within the environment (such as rendering the image to illustrate its movements whilst traversing it). For brevity, this situation is not detailed in the following descriptions.

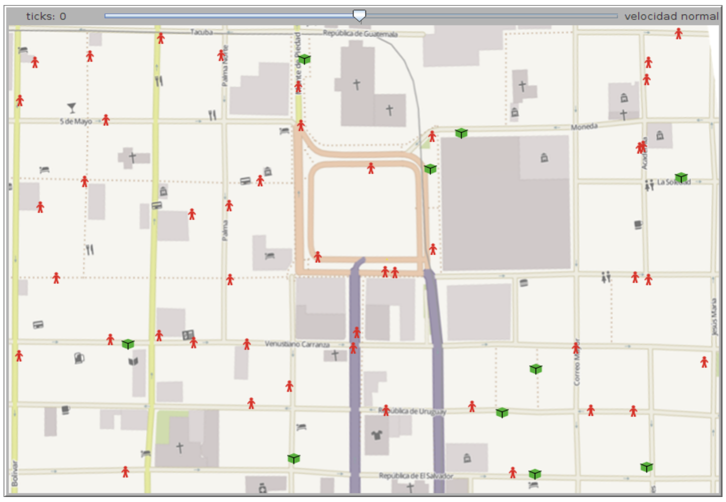
- createAgNL: Creates the NetLogo turtle, for use as an avatar by Jason agents in the NetLogo environment.

- goToSector: The brigadier agents go to their designated sector (1 of 4).
- doSearch: The brigadier agents perform a search for civilians, obstacles and fires in their designated sector. Once an objective is found it is added as a belief, and notifies a suitable agent about it. It is divided in the following scenarios:
  - walk: The brigadier agents move through their designated sector
  - searchForCivilian: The brigadier agents sense the environment looking for civilians
  - searchForObstacles: The brigadier agents sense the environment looking for obstacles
  - searchForFires: The brigadier agents sense the environment looking for fires
- goToObjective: The agents traverse the environment, looking for the objective (civilian, obstacle or fire) with the intent to perform rescue actions.
- rescueAction: Once the agents reaches their current objective (within it scope governed by distance and width of sight parameters in NetLogo) they perform, rescue, clear, pickup, or extinguish actions. It is divided in the following scenarios:
  - clearing obstacles: The agent acts, performing a clearing action on the target.
  - collecting civilian: The agent acts, performing a collect civilian action on the target.
  - extinguishing fires: the agent acts, performing a fire extinguishing action on the target.

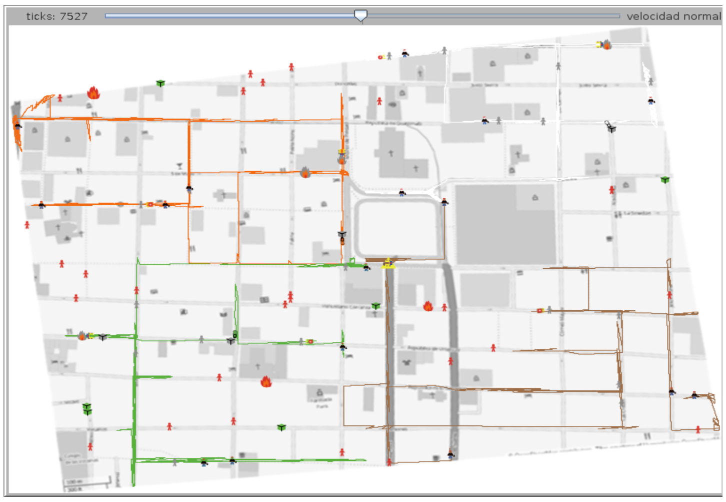
In Figure 3.24 a screenshot of the D-R BDI MAS is shown, running under the following configuration of agents:

| **Agent** | **Instances** |
|---|---|
| active | |
| brigadier | 16 |
| ambulance | 4 |
| police | 4 |
| fireman | 4 |
| passive | |
| fires | 10 |
| victims | 45 |
| obstacles | 10 |

For illustrative purposes, the traces of their movements are shown. Each rescue agent is assigned a distinct colour, making its trajectory more distinguishable.



(i) The BDI MAS running



(ii) A screenshot of the map, where can be seen the civilians, fires and the trace of the agents looking for targets

**Figure 3.24:** *The D-R BDI MAS, developed under the option* ***NetLogo-In-Jason*** *described in the previous section.*

### 3.4.4 Testing and Validation of the D-R Simulation

The testing is focused on the interaction between NetLogo and Jason, to illustrate the feasibility of the platform. To scale in the number of agents performing rescue actions, two set of tests were performed, varying the number of agents reported.

The first set was run on an DELL Optiplex 7010 Core i5 CPU with 8GB of RAM with Ubuntu 14.04.5 LTS. In Table 3.4, a summary of the system running with an assorted number of agents is presented. Rescue actions did not rise after increasing the number of brigadier agents, searching time decrease. Whereas, increasing the number of collecting agents (police, fire brigades and ambulances) reflects a positive change in the number of rescue tasks performed. The Performance, defined in Equation (3.1) glimpse of the number of pursued rescue actions, once the system reached the maximum of 50,000 actions per run. In the case of the run tagged as Av250, an increase in the number of rescue staff increases this ratio, for the number of expected actions is higher than other runs due to larger number of civilians. In the second set of experiments, the system ran on

**Table 3.4:** *Averages of runs up to 50000 actions with variations of the number of agents.* ***Av[16,200,500]:*** *⟨{A, P, F=4}, {B=16, 200, 500}, {K ,R=10},V=45⟩ 10 runs.* ***Av250:*** *⟨{A, P, F=16},{K ,R=10}, B=50, V=250⟩ 5 runs. Letters stand for: B=brigadier, A=ambulances, P=police, F=firebrigade, K=blockades, R=fires, V=victims.*

| | **Av16** | **Av200** | **Av500** | **Av250** |
|---|---|---|---|---|
| Time | 06:39 | 03:40 | 05:35 | 05:10 |
| Victims in danger | 9 | 13 | 29 | 67 |
| Saved | 36 | 32 | 16 | 182 |
| Collected | 36 | 32 | 15 | 175 |
| Cleared | 6 | 5 | 5 | 7 |
| Extinguished | 3 | 6 | 4 | 6 |
| Performance x 1000 | 1.62 | 1.5 | 0.8 | 7.4 |

a laptop DELL Vostro 3750 Core i7 CPU with 10GB of RAM with Ubuntu 14.04.5 LTS. Figure 3.25 shows a view of 5 runs, with up to 50, 000 actions, with the following configuration of agents: fire (10), blockades (10), victims (200), police (4), ambulances (4), and fire brigades (4). Only brigadier agents were varied: 16, 50, 100 and 200. The figure shows a plot of rescue actions fulfilled by the D-R BDI MAS (people in danger, saved, or collected; blockades cleared, and fires extinguished), where the Y axis defines the number of rescue actions performed by the system, and the X the number of runs. As can be seen, the system behaves in a similar way during each run, despite varying the number of agents (representing the rescue staff). Circle spots in the lines identify the number of actions reached by the system for each type of rescue action, completing within a cycle of 50, 000. For instance, in (a), the number of people remaining in danger varies from 63 to 75 victims, whilst people saved ranges from 128 to 136.

Averages of rescue actions from each 5 runs, per number of brigadier agents, is shown in Table 3.5.

| **BA** | **Danger** | **Saved** | **Collect** |
|---|---|---|---|
| 50 | 69 | 130 | 126 |
| 16 | 64.2 | 134.8 | 130.4 |
| 100 | 69 | 130 | 125.6 |
| 200 | 66.8 | 132.2 | 127 |

**Table 3.5:** *Averages of* 5 *runs of the BDI MAS up to* 50, 000 *actions. BA stands for Brigadier Agents.*

In the first test, summarized in Table 3.4, the observed correlations between the number of brigadier agents (from 16 to 500) and the rescue actions denotes a positive effect through having more agents looking for victims: 0.9808 for people in danger (Danger) and −0.9808 for saved victims (Saved), whilst collected

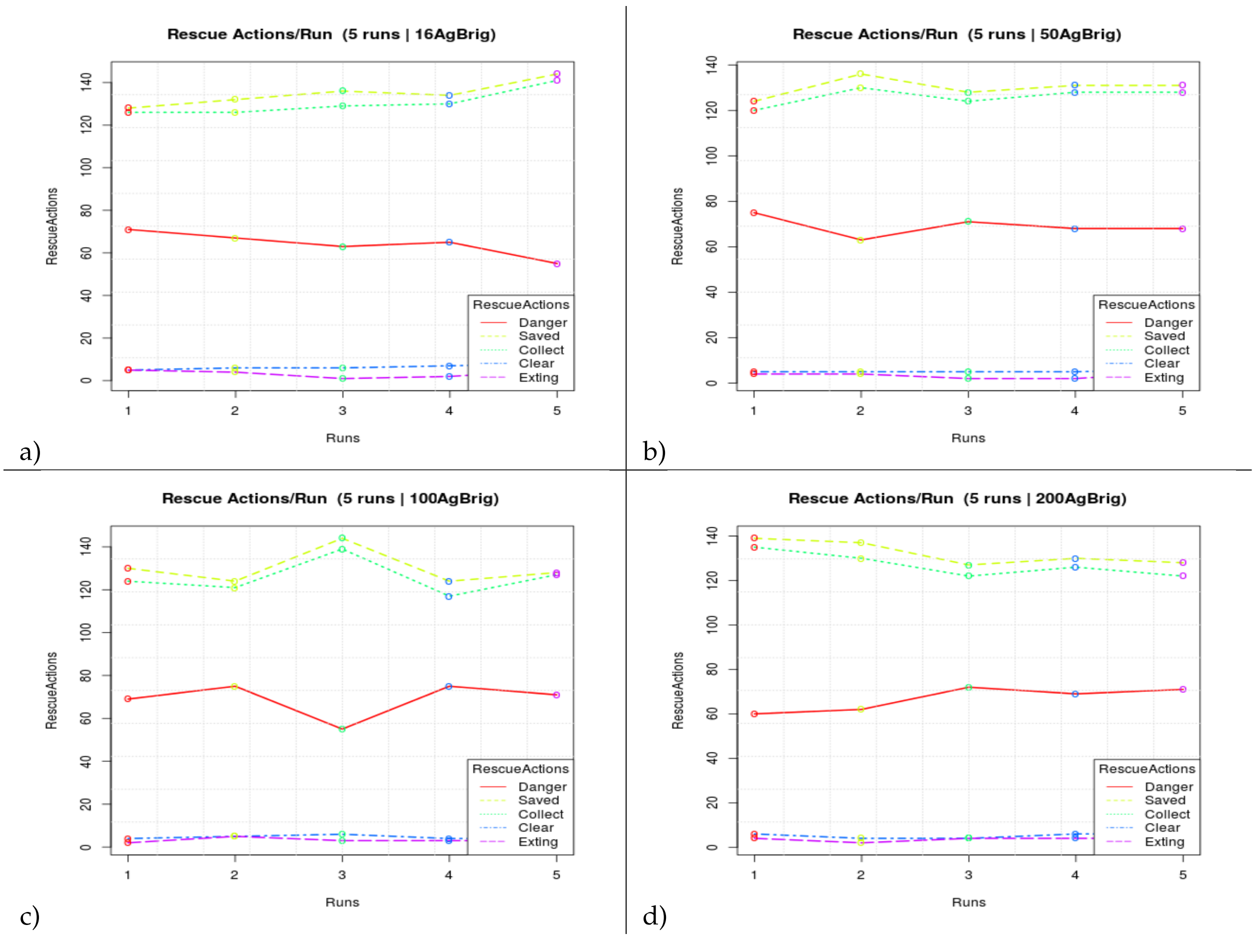


**Figure 3.25:** *The MAS BDI performance of 5 runs with different rescue staff: a) 16 brigadiers; b) 50 brigadiers; c) 100 brigadiers; and d) 200 brigadiers. The rest of agents remain constant: fire (10), blockades (10), victims (200), police (4), ambulances (4), and fire brigades (4).*

people (Collect) presented −0.9789. The cases of clearing blockades (Clear) and extinguished fires (Exting) reached −0.789318 and 0.1947211, respectively.

Taking the averages of the second test (Table 3.5) to calculate the correlation between the variables, there is a weak correlation between the increased number of brigadier agents (BA) and the number of rescue actions to save people, and a very low correlation with clearing obstacles and extinguishing fires. In the case of Saved it is of −0.2254 and 0.2254 for Danger; while Collect reported −0.4331, Clear −0.5272, and 0.5095 for Exting.

Saved and Danger have a correlation of −1 in both tests, as they inverse each other; while the correlations Collect/Saved were 0.9999 for the first test and 0.9756 for the second one. Collect/Danger shown −0.9999 and −0.9756 in the first and second experiments respectively. Finally, the calculated correlation between Exting and Clear was −0.7559 in the first experiment and 0.3244 for the last, which can be considered acceptable since no dependence is expected on those values.

The results of both set of tests correspond each other about the scalability of the integration of Jason and NetLogo, whilst working with variations in the number of agent type. This situation remains when there is a large population of victims, increasing the number of brigade agents is insufficient, and does not significantly increase rescue tasks execution.

In the next chapters the learning mechanisms are presented using the test-beds described above for the single agent scope to the collective perspective of learning in a D-R BDI MAS.

## 3.5 Summary

This chapter reports the work of the first step taken in answering the Research Question 1.3, by introducing the learning problem in BDI agents, and setting up the experimentation framework, necessary to fulfil the three Research Objectives.

To achieve this, the cognitive agency, based on the Belief-Desire-Intention (BDI) architecture is presented. Its constructs and its reasoning cycle, that goes from perceiving to acting, is presented. The BDI agency defines a deliberation process, carried out through three main functions, namely: belief revision, options generation and intention selection. Then Intentional Learning (IL) is introduced, some points about learning in BDI agents under this approach are underlined as follows:

- IL, at the level of plan acquisition, in fully BDI environment has not been reported in the field.
- IL in the collective scope has not been approached yet.
- The potential of BDI agents to represent complex systems is still scarcely exploited in domains more challenging than test-beds, like social simulations.

To cope with this, in this chapter a definition of Collective Intentional Learning is provided. Then, Jason, a framework for MAS development strictly lined up with the BDI architecture, is reviewed. Regarding the the last point above, this chapter presents work in two directions. Firstly, a description of the Disaster-Rescue problem, a challenging and socially relevant domain is presented. This domain portrays a city under disaster conditions, people in danger, and a group of rescue staff. The challenges are associated with the coordination, the task division and the organisation of the MAS, representing a good scenario to test learning agents.

Secondly, despite D-R simulations having been developed using mostly reactive agents, such endeavours can benefit from using cognitive agents achieve realistic simulations. Consequently, the first step is to configure a framework able to represent a D-R environment where agents can be allocated, sensing and acting within according to their perceptions, intercommunication, and deliberation processes.

Such a framework is aimed to serve as a platform for developing learning BDI agents under the IL approach, in both the single agent case and in the MAS scope. Regarding the latter, the interconnection of both MAS development frameworks is introduced, and a first implementation of a MAS is showed.

Furthermore, the framework presented here progresses in bridging the gap in combining Agent-Based Modelling (ABM) and Multi-Agent Systems (MAS). ABM is concerned with the study of either natural or social complex systems, by simulating their components and relations in a dynamic way. On the other hand, MAS aim to solve problems in a flexible and robust way, by assembling sets of agents interacting in cooperative or competitive ways to fulfill potentially common objectives. The framework combines both approaches by integrating two major platforms in the field of ABM and BDI MAS, namely, NetLogo and Jason.

The integration of both tools can be done in two ways: *Jason-in-NetLogo*, to include Jason into NetLogo (i.e. using just the Jason BDI engine); or *NetLogo-in-Jason*, to include NetLogo in Jason (i.e. using the former as the environment). Both options are explained through running examples, and the algorithms to carry out each of them are provided. The latter option is more suitable to develop the intended MAS D-R simulation, to exploit the rendering capabilities of NetLogo while maintaining the higher-level deliberation cycle of Jason agents. In this way, BDI Jason agents are enriched through using the reactive and lower-level capabilities of its avatar-like NetLogo turtles, and the latter are powered by the BDI reasoning loop of Jason agents. Executing BDI plans results in calling NetLogo functions to act, to perceive or to communicate with others.

The D-R BDI MAS uses a simple strategy of map partitioning and distribution of the rescue agents in the environment, with communication capabilities and a simple task allocation policy. Environmental elements are represented as NetLogo agents, while rescue agents are represented in both platforms, giving Jason agents higher level task of reasoning and decision making. In this way, NetLogo agents act in the environment on behalf of their counterpart agents in Jason.

The systems scales well, from a small to larger population of rescue agents, demonstrating the feasibility of integrating both platforms. Finally, to the best of my knowledge, there is no other implementation interconnecting NetLogo and Jason in the way presented here, and it does constitute a novel contribution in MAS development.

# Chapter 4

# Intentional Learning

This chapter is devoted to accomplishing the Research Objective 1 "Exploring the behaviour acquisition in the BDI agency ". The method to introduce IL in a fully-fledged BDI environment in the level of plan acquisition, in a single agent scope, is presented.

This chapter is organised as follows. First, the learning problem described in Subsection 3.1.3 of Chapter 3 is undertaken, and an IL method at the level of plan acquisitions (aimed to learn new skills), is presented. Second, the integration of IL in a fully-fledged BDI environment (Jason) is provided. Third, the experiments to acquire two behaviours through learning: wall-following and target-searching, within a simple vacuum-cleaning environment, are reported. Finally, the learning is tested in the Disaster-Rescue Domain.

## 4.1 Involving Intentional Attitudes in Learning

As stated in 3.1 in Chapter 3, IL is a behaviour driven by the mental attitudes that characterise BDI agents, namely, beliefs, goals and intentions. Consequently, IL can be applied to three different areas: a) Goal adjustment (focused on monitoring beliefs and adjusting them as needed); b) Context of plan applicability (centred on the adjustment of triggering conditions); and c) Activities or Plans (learning is pre-specified in plans, driven by goals is reactive way, adding new plans to the agent plan library). Consequently, IL to acquire new plans implies to have plans that generate new plans.

This research is aimed to incorporate IL at plan level in fully BDI agents, as this learning includes the possibility of acquire new skills due to the updating of the repertoire of plans.

Accordingly to the problem description 3.1.3 introduced in Chapter 3, such a learning involves answering two questions: *when* and *how* to learn (about *what* to learn, i.e. the content to be learned, it is determined by the domain).

### 4.1.1 Identifying *when* learning is needed

Within the BDI architecture, intentions constitute a stack where they are stored and retrieved from. During the reasoning cycle, agents determine how to achieve an intention generated by processing an event either external or internal. To do so, the option generation retrieves a set of applicable plans from the plan library. After such a selection, one option (i.e. a plan) is processed to determine if its body components are executable actions (if so, they are carried out) or sub-goals (they are stacked as new intentions to be pursued). The triggering condition of learning can be established taking care of two processes of the BDI reasoning loop, namely intention and action selection. This situation is outlined in Algorithm 5 (the IL related sentences are highlighted).

IL is interleaved as an intention, devoted to monitoring other intentions. The purpose of monitoring is to identify the cases of failure. In other words, those cases where an intention could not be pursued (line 6): a) it is unachievable (due to the changes in the state of the environment), or if there is not any option to be applied (due to a lack in the procedural knowledge -i.e. the agent does not have any plan to answer with); and b) it fails, despite being selected, due to environmental changes, some of the actions belonging to the intention cannot be executed (line 11). In any case, IL performs a plan generation procedure (lines 7, and 12).

There are two basic cases when learning is required, both are related with the failure of an intention:

- Despite being selected, the intended action cannot be applied.
- The agent has not the procedural knowledge to react an event.

Both are regarded with improvement in the agent behaviour. The former accounts for reviewing the conditions under an intentions fails, generating a new plan as required. Furthermore, the the latter can be used to generate new plans from the set of executable actions available to agents, leading to the acquisition of new skills.

**Algorithm 5:** BDI reasoning cycle with IL

```
Data: B: beliefs, I: intentions, O: options, E: event queue, PL: Plan Library
Result: a: action
begin
    // Initialise Beliefs and Intentions
1   initialise(B)
2   initialise(I)
    while True do
        // Update beliefs with new events
3       O ← perceive(E)
4       D ← option-generation(B,I)
5       I ← intention-selection(B,D,I)
        // Start plan generation
6       if $I = \emptyset$ then
7           PL ← il-plan-generation(B,I,PL)
8           restart-cycle(E);
        end
9       a ← action-selection(I)
10      execute(a)
11      if fail then
12          PL ← il-plan-generation(B,I,PL)
        end
        // Update I and D if required
13      drop-successful-attitudes(I,D)
14      drop-impossible-attitudes(I,D)
    end
end
```

### 4.1.2 Determining *how* to learn

After identifying the cases where learning is required, the next question is how to define a learning strategy to generate new plans.

Plans can be thought as special forms of beliefs, representing abstract specifications of both the means for achieving certain desires and the options available to the agent [93]. Consequently, learning can be introduced as a set of plans to generate new plans, triggered by the failure of other intentions, as stated before. In Algorithm 6 the learning process is shown.

The learning algorithm receives the belief base *B* the failed intention *FI*, and the plan library *PL*. The triggering event *FT* and the context *FCntx* are extracted from the intention structure (lines 1-2). Plans *CP* are collected through matching their triggering events with *FT*, or searching in the entire plan library in the absence of matching plans (line 3). A set of candidate actions *CA* is collected from *CP*, considering *FT* and the current context (*FCntx*) (line 4). The set *CA* is refined based on their applicability, and following a ranking

**Algorithm 6:** il-plan-generation

```
Data: $B$: belief base; $FI$: The current (failed) intention; $PL$: Plan Library;
Result: Pl: A new plan to be added into the agent Plan Library
begin
1   FT ← getTrigger(FI)
2   FCntx ← getContext(FI)
    // Collecting Plans
3   CP ← collectPlans(PL,FT)
4   CA ← collectPrimitiveAction(CP,FT,FCntx)
    // Selecting and Ranking actions
5   SCA ← selectRankCandidateActions(CA, $\lambda$)
6   Body ← setBody(SCA,$\psi$)
7   Cntx ← setContext(FCntx,SCA,B)
    // Build the new plan
8   Pl ← setPlan(FI,Cntx,Body)
end
```

criteria $\lambda$ (line 5). Then, to integrate the body *Body* of the new plan, choosing actions from *SCA* (line 6) according to an ordering strategy $\psi$. The context of the new plan is set in line 7, it is determined based on the current context ($fCntx$), the selected actions (*SCA*) and the beliefs of the agent ($B$). Finally, in line 8 the new plan is set with *FI* as the triggering event, *Cntx* as the context, and the actions defined in *Body*.

The central activities are to determine *CA* (by collecting actions), *SCA* (by selecting and ranking), and *Body* (the set of actions included in the new plan). Then, defining the ranking criteria $\lambda$ and the plan building strategy ($\psi$) is the core of IL, it constitutes a planning activity embedded in the BDI architecture.

### 4.1.3 Collecting and Selecting Actions

The method introduced in this work is based on the learning and planning techniques exposed in Section 2.5 of Chapter 2, namely Reinforcement Learning and Partially Observed Markov Decision Processes to drive the action collection and selection.

Additionally, as the selected actions have to be sorted in plans, selecting actions goes beyond the solely ranking of collected actions. Then, it is required to define an arrangement strategy for the new actions (i.e. the referred ordering $\psi$).

In [117], an heuristic method of BDI-like plan building is introduced: Manipulative Abduction (MA). Rooted in Logics [71], MA is a deliberation method based on the observation of possible causes to explain an effect, in other words, actions are evaluated a posteriori, when their consequences can be observed [117]. As

MA can be stated as plan templates that represent the behavioural response to finding patterns or regularities in the environment [117]. Taking this approach, here the ordering strategy $\psi$ is implemented through MA, represented as plan templates to define how to build new plans from other plans in the agent's repertoire.

While plan templates determine the plan construction, the $\lambda$ criterion to select the actions to be taken from other plans is defined based on a Reinforcement Learning technique, namely Q-Learning, a well-established approach of learning, based on the usefulness of the interaction of the agent and its environment for the sake of performing an intended task [120].

## 4.2 The learning method to acquire new plans

To integrate IL in the plan acquisition level, the learning controls the deliberation process and the execution of intentions to monitor the actions looking for failures to change the *PlanLibrary* at run time for the sake of behaviour improvement and new skills generation.

In the following the $\lambda$ criterion, and the set $\psi$ of plan templates are presented.

### 4.2.1 The Criterion for Ranking Actions

The generation of new plans, need to determine the components of the plan to be constructed. Adopting the AgentSpeak formalism stated in Section 3.2.1 of Chapter 3, a plan comprises the following elements: `@label head` $\leftarrow$ `body`, where `head = te : cntx`, and `body = {sub-goals | actions}`. Furthermore, `sub-goals` can be: achieving literals (`!`), or belief testing (`?`).

Differing from `sub-goals`, `actions` are non-decomposable elements (i.e. they are defined as executable actions), then in terms of planning, they are considered *primitive actions* and are the main input for learning to generate new plans.

The $\lambda$ criterion is expressed as two RL-inspired concepts[1], namely reward and utility. Recalling from Chapter 2, Section 2.5, a reward is defined as a mapping of each perceived state or state-action pair of the

[1]Despite this criterion can be expressed using the constructs of any machine learning technique, in this work it is lined up with the RL techniques due to its correspondence with the planning problem in agents.

environment to a single number [120], as follows:

$$reward : S \times A \mapsto \{0, 1\} \mid \mathbb{R}$$

Then, a reward indicates the desirability of an action-state, in an immediate sense [120]. On the other hand, to estimate how good an action is, when performing a task, a utility measure is associated to it, namely $q_r$.

Plans have associate a utility measure also, denoted by $Q_p$, computed based on the utilities of its body components. The more beneficial the plan, the more likely it is it will be used, and consequently its utility will increase. $Q_p$ takes into account just the current state of the agent when that plan is fired. Then the utility is computed by means of a RL-inspired formulae (Q-Learning) [120]:

$$Q_p = \sum_{i}^{n} q_{r_i} \tag{4.1}$$

Where $q_{r_i}$ are each of the values of the actions belonging to a plan, when executed in a given state. In the case of plans with only one action, $q_{r_i}$ is the past value $Q_p$, otherwise it is the summation of all the primitives belonging to the plan executed in similar contexts.

For instance, the utility of a plan `P1` defined as follows: `P1 : <context> <- a; b; c.`

could be calculated given the following $q_{r_i}$ values for each of the actions of `P1`:

$q_{r_a} = 0.15$, $q_{r_b} = 0.05$, $q_{r_c} = 0.011$

Then, the $Q_{P1}$ value is $Q_{P1} = q_{r_a} + q_{r_b} + q_{r_c}$:

$Q_{P1} = 0.15 + 0.05 + 0.011$

$Q_{P1} = 0.211$

So, the utility of `P1` should be 0.211.

On the other hand, $q_r$ values are computed according to the Q-Learning utility updating:

$$q_r = q_{r-1} + \delta * (R + \gamma * Q_p - q_{r-1}) \tag{4.2}$$

Where $\delta$ is the learning rate set to 0.1; $\gamma$ is the discount factor set to 0.95; and $q_c$, as defined in Equation 4.1, is the plan utility. This keeps the balance between past and current utilities, allowing the updating to take into account the new observations from the environment.

On this basis, the $\lambda$ criterion establishes a priority order in the plan selection, determining the behaviour of the agent whilst running, and conferring the capability of keep learning (as required by environmental changes) when pursuing its intended task.

Furthermore, one of the main points in planning is to determine the pre-conditions and post-conditions of the new plans. The latter is achieved by the context of applicability of `sub-goals` and `actions`, while the latter requires the agent to perform a tracking to analyse the consequences of actions, that cannot be foreseen. To collect such observations, in addition to the beliefs for storing Qr-related values, other beliefs are needed, namely the set of *tracking beliefs*, denoted by ($\theta$), which includes:

- `st/?` (`st(...)`): Stores a representation of the state in a given domain, then its arity is domain-dependent.
- `tAB/4` (`tAB(st1,action,st2,frequency)`): Stores the transition of states before and after an action is executed.
- `q` (`q(plan,st,utility)`): Stores the computed utility $Q_p$ of a plan once it has been triggered in a given state `st`.
- `qr/4` (`q(plan,action,st,utility)`) : Stores the utility of a primitive action belonging to a certain plan $q_r$, in a given state `st`.
- `rewardAcSt/3` (`rewardAcSt(st,action,reward)`): Stores the reward of an action in a given state.
- `planUtility/3` (`planUtility(plan,utility,frequency)`): Stores the plan utility and its frequency of execution.

### 4.2.2 The Exploration/Exploitation Tradeoff

A characteristic issue in learning is the exploitation and exploration dilemma. This regards the option to keep using the learned plans versus continuing to add new ones. The agent possesses the following elements to help it decide:

- An exploration parameter $e$;

- A discount factor $\nu$ to penalise the less beneficial plans;
- Re-learning stage.

The first case introduces variations in the eligibility of actions, by transferring the Reinforcement Learning *e-greedy* strategy [120] from the action selection to the learning process. In other words, instead of applying that strategy every time the agent acts, it is restricted to only the learning case, keeping the action selection mechanism based on plan utilities. As a result, low ranked actions (which suggest lower order of preference) are enabled to form part of a new plan. *e* is set to 0.01 as in [120].

In the second case, whenever a plan is triggered without successfully reaching the intended behaviour, its utility is decreased. It speeds up learning, for it promotes the utility updating of plans that are not helpful enough when fired many times.

The last case is related with the fulfilment of a higher level task. It represents a measure of the achievement of an intended behaviour. It is domain-dependent, and it gives a hint towards progress that achieves the target behaviour. On this basis, the re-learning stages are implemented as a failure, arising in plans where a number of actions has exceeded a threshold (designed to calculate less-beneficial plans). This test can be expressed as follows:

$$adequacy(Ad, F) \quad \begin{cases} F & Ad > n \\ F & Ad = n, \text{ and } F \leq t \\ T & otherwise \end{cases}$$

Each time a plan has failed in achieving the intended behaviour, $A$ and $F$ values are increased. The first case prevents the agent of launching re-learning stages up to a number of $n$ plan executions. Whenever a plan seems unbeneficial to reaching the intended behaviour, the value of $t$ is increased. Then, once the first condition is accomplished, the agent will fire the learning plans each $t$ times. The minimum of $n$ actions allows the agent to explore the environment, collecting the observations required in the learning process. Once the re-learning is performed, $t$ is restated to $t + 1$, given that the PlanLibrary has increased by one plan.

### 4.2.3 Templates for Plan Generation

To fulfil the purpose of acquiring new plans through IL, to increase the range of actions of learning agents, the learning uses a building strategy $\psi$ (based on Manipulative Abduction), expressed as plan templates to define the structure of new plans (jointly with the aforementioned process) to select the candidate actions to be part of the learned plans.

In the following these templates are presented starting with the most simple template to include just one action in the new plans and then upgrading the learning beyond simple plans by including repetitions and sequences of actions.

#### 4.2.3.1 New Plans With One Action

The first template generates a plan with only **one action** in its body. The actions considered as candidates to be added in the new plan belong to the set of actions included in the other plans' bodies. The selection criteria are the applicability of the candidate actions to the current context, for instance the context that gave rise to the failure, and the utility of the action for the current state of affairs of the environment.

The template is depicted in Figure 4.1.

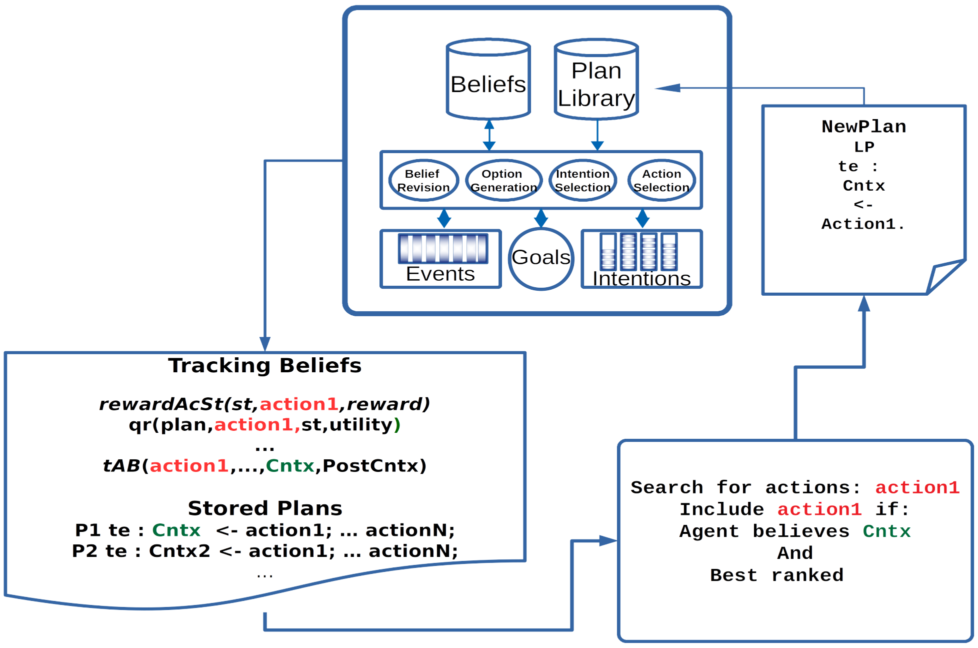
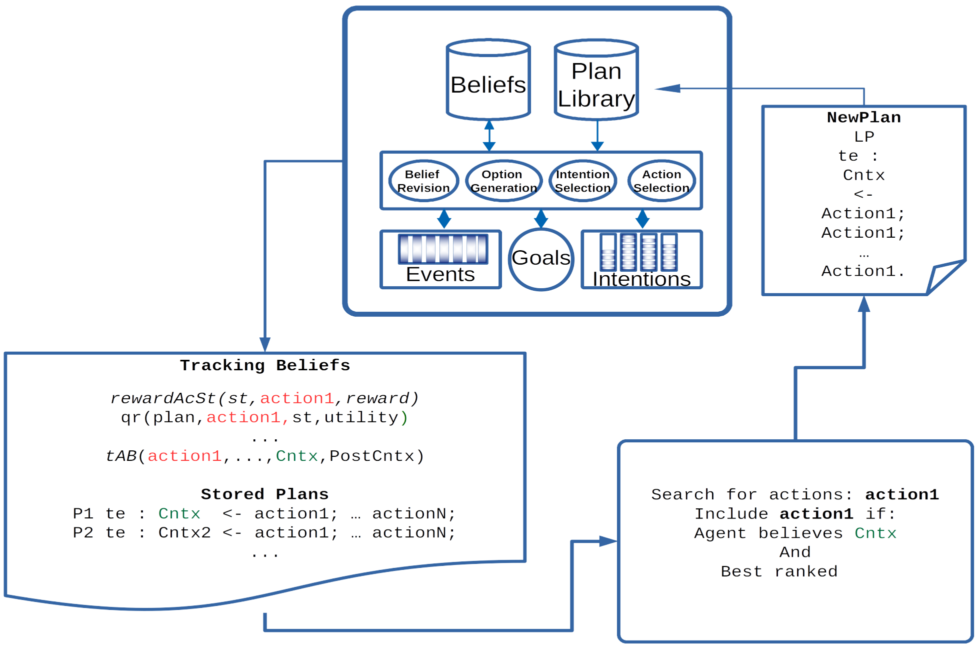


**Figure 4.1:** *The plan template to generate a new plan with only one action in its body.*

#### 4.2.3.2 New Plans With Repetitions of Actions

The second template concerns the generation of more complex plans (i.e. plans with repetitions of actions), that can potentially give rise to more useful behaviours, due to their interaction with the environment and the new set of actions included in them. The number of actions is given by a parameter $N$, set up at design time. Analogous to the previously described learning template, the purpose of learning repetitions is to acquire a new plan accordingly with the current primitive actions and their applicability. Then agents can continue operating with a new plan, recovering from a possible failure or improving the fulfilment of a given task.

The template is depicted in Figure 4.2.



```
NewPlan
LP
te :
Cntx
<-
Action1;
Action1;
…
Action1.
```

```
Tracking Beliefs

rewardAcSt(st,action1,reward)
qr(plan,action1,st,utility)
...
tAB(action1,...,Cntx,PostCntx)

Stored Plans
P1 te : Cntx  <- action1; … actionN;
P2 te : Cntx2 <- action1; … actionN;
...
```

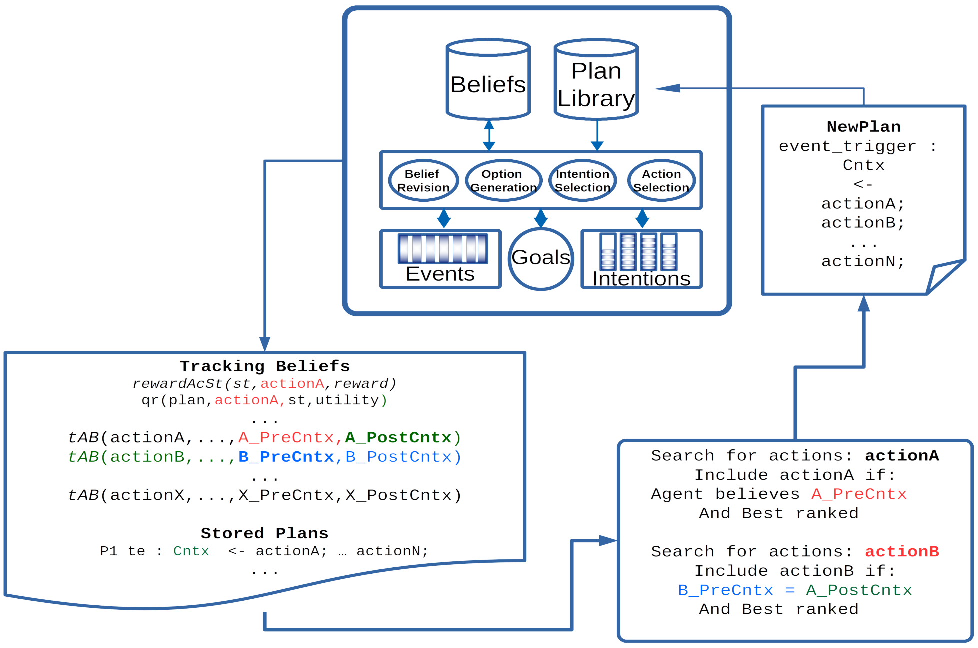
```
Search for actions: action1
Include action1 if:
Agent believes Cntx
And
Best ranked
```

**Figure 4.2:** *The plan template to generate a new plan with repetitions of actions in its body.*

#### 4.2.3.3 New Plans with Sequences of Actions

The last template for implementing IL produces a new plan including an alternate sequence of actions. This is different from [117], where plans and actions are stored in a list to avoid repetitions. The selection of actions is based on their applicability, according to the current context, and the utility of actions in the history of executions (i.e. tracking the trace of executed plan/action so far). In this case, the length of the sequence is also defined by the parameter $N$. In comparison with the previous templates, this one rely more in the tracking beliefs. For instance, to observe the sequence of fired actions while the agent is interacting with the environment before a failure occurs, and to track sequences of valid actions according to their demonstrated past applicability and utility. Thus, there is a need to track the times an action is triggered, and the pre and post conditions when it does so (i.e. the transitions between states and the actions to perform those changes). This is accomplished by setting tracking beliefs on the actions undertaken by the agent, then computing its utility. The belief `tAB`, used to collect the transitions of the context once a primitive plan is fired plays a key role in this template.

The template to add sequences of actions to a new plan is depicted in Figure 4.3.



```
NewPlan
event_trigger :
Cntx
<-
actionA;
actionB;
...
actionN;
```

```
Tracking Beliefs
rewardAcSt(st,actionA,reward)
qr(plan,actionA,st,utility)
...
tAB(actionA,...,A_PreCntx,A_PostCntx)
tAB(actionB,...,B_PreCntx,B_PostCntx)
...
tAB(actionX,...,X_PreCntx,X_PostCntx)

Stored Plans
P1 te : Cntx  <- actionA; … actionN;
...
```

```
Search for actions: actionA
Include actionA if:
Agent believes A_PreCntx
And Best ranked

Search for actions: actionB
Include actionB if:
B_PreCntx = A_PostCntx
And Best ranked
```

**Figure 4.3:** *The plan template to generate a new plan, adding sequences of actions in its body.*

### 4.2.4 Plan revision

The learning templates, presented previously, generate three kind of plans based on partial information (i.e. the perceptions of the agent up to a given moment). Sequences of actions are more sensitive to the history of environment observations: when applied early can result in loops of turns left and right, keeping the agent in the tunnel-like structure. In some cases, the sub-goals of a plan could not complete their execution.

Consequently, a revision of the plan is needed. Plan revision tuning mechanism for learning to improve the agent's behaviour, once the learned plans' results are not good enough to fulfil an intended task or to reach certain behaviour. There are two cases where plan revision is needed:

- In the basic case, to cope with the inapplicability of sub goals; and

- When a plan seems no longer helpful for fulfilling a task, given its negative utility, and it execution has surpassed a fixed threshold. Then, plan revision aims to increase the utilities of a plan again.

In both cases, the target plan is reviewed, and some of the actions are substituted by new ones. In the first case, an action is eligible to be deleted from a plan whenever it fails. On the other hand, actions in the plan are rearranged based on its utility and their context of applicability, giving priority to the most beneficial sequences. The candidate actions are selected based on the tracking beliefs and their matching context of application.

## 4.3 Implementation in a fully-fledged BDI framework

From its establishment [117] the IL was implemented in two ways: a) as a BDI layer upon NetLogo, i.e. a list-based implementation of the BDI architecture directly coded in NetLogo's programming language; and b) interconnecting the BDI framework Jack and NetLogo.

On the other hand, in this thesis, a method to include IL in a fully-fledged BDI framework (Jason), is introduced. Additionally, the interaction between Jason and NetLogo is carried out, configuring a framework to test IL in both the single agent case and the MAS scope.

In [117] some concerns regarding the development of IL are presented. They are centred on the inclusion of intentions for controlling and monitoring other intentions, and monitoring actions, identifying the failure cases. According to [117], the last feature can be reached implementing meta-level plans, that allow the control deliberation, execution commitment and explicitly drive the learning process through cycles of abduction on hypothesis testing. On the other hand, one of the reasons discussed in [117], was that BDI frameworks were immature to support the IL requirements aforementioned. However, due to the Jason current development, it is now able to fulfil those requirements.

Table 4.1 presents the main issues when considering IL, and the current state of Jason development, that makes this inclusion possible.

In addition to the methods for suspending or dropping intentional attitudes (`.current_intention( Intention)`, `.suspend`, `.wait`, `.resume`, among others), and manipulating the intentions and plan structures, Jason defines some constructs that can be used to implement the proposed algorithms and ranking functions (the $\lambda$

| **Intentional Learning issues** | **Jason constructs** |
|---|---|
| Capture the intention structure at run time | It is possible to get the current intention and the intended means, given the following methods and internal intentions: `.current_intention(Intention)` `.desire`, `.suspend`, among others |
| Change the plan library at run time | Adding new action to a plan and even a full new plan to the Plan Library is possible with these methods and internal actions: `p.getBody().add(0,newAction);` `ag.getPL().add(p);` and `.add_plan();` |
| Control the execution of an intention (by means of another intention) | This can be done by using the *fail plan* to capture failures (expressed as a negative achieve goal: `-!`, like `-!planName(X)`), and *meta plans*, even with *internal actions* to start some processing on the current intentions and its plan. Additionally, *meta-events* are useful. |
| Monitoring actions through meta-level plans for checking failure cases before including actions into a plan body. | This is the main issue and the most difficult one to implement the IL. The use of tracking beliefs and *meta-events* (to unify even with the name of the plans), *annotation* on plan and beliefs, the *fail plan* to manage the failure cases, and even more, the *meta-level* plans allows to implement this central feature of IL. |

**Table 4.1:** *Comparison between the issues in the implementation of Intentional Learning [117] and the current state of Jason development*

criterion) of IL, described as follows [18]:

- Annotations: lists of first-order terms (constant values, not variables), providing details that are strongly associated with one particular belief. Annotations are enclosed in square brackets immediately following a literal: `p(X)[ann1]`, where `ann1` is the annotation of the literal `p(X)`. This configures a useful mechanism to add more information to the beliefs, and to identify some aspects of it, such as: the source of some incoming message (i.e. the agent that sent a message); the perceptual information; and, the most relevant to this work, to make mental notes.

- Plan labels: are literals to identify a plan in a unique way, is its possible to add annotation to a plan label, for instance:

  `@pl +!g : cntx <- a1; a2.`,`@pl +!g : cntx <- a1; a3.`,and `@pl +!g : cntx <- a3,a2.`

  they are three different plans, answering to the same event `!g` with diverse actions in their bodies, clearly identified through their labels.

- Meta-variables: are higher-order variables, they allow to manipulate different structures, like predicates or formulas, or even to manipulate any event. For instance, some expression or formula defined as a belief can be retrieved in a plan through a higher-level variable: given this belief `someAction(inCorner,.print(‘‘In Corner!’’).` stored in the belief base, and the following plan `+!@p1 S[source(baddy)] : someAction(S,A) <- A.`, if the agent perceives the event `!inCorner`, the plan `@p1` will be triggered, executing the printing action.

- Meta-events: Denoted by the prefix ˆ, are events to monitoring goal state changes, they can be started, finished (the goal has been achieved), failed, suspended, and resumed. When combining with meta-events, they can constitute meta-level plans to monitoring the execution of other plans, including those belonging to a failed intention.
- Fail plan: Whenever a plan trigger by `!+g` fails, or there is no applicable plan for handling it. The fail event, denoted by `-!g`, substitutes `!+g` (and the plan associated with it), and pushed to the top of the intention. Then, the execution of the failed plan is replaced by that one associated with `-!g`. Therefore, it suffices to include a plan: `@pf -!g : cntx <- body`, to handle the failure.
- Internal actions: user-defined actions running internally into the agent. Implemented as Java code available to the agent. They can appear in the context and in the body of a plan. They are not aimed to change the environment, but the internal state of affairs within the agent itself.

In Jason changes in the Plan Library at run are not included directly in the agent code file, but just in the memory of the running agent. So, to keep the updated plans, a saving process of the code is required.

Jason provides an action selection mechanism implemented by its Java *plan selection* function that can be extended in order to adjust the way the plans are selected to be executed in case they are applicable and they share the same name. The default selection criterion is to take the first plan defined in the Jason code (i.e. the first plan defined, the first plan selected). Nevertheless, a different priority order for plan selection can be defined. Consequently, it is possible to drive the behaviour of the agent by modifying this function. Hence, exploiting this Jason facility, each plan is labeled with an annotation about its utility while performing a task. Such a measure is computed based on the primitives and their performance during the learning process.

On this basis, annotations are used to store the utility of a plan according to the $\lambda$ criterion, whilst the intention selection, and options selection functions can be modified to take into account the plan annotations, and establish a priority when choosing a given structure.

The following snippet of plans illustrate how it can be implemented :

```
@moveF[utility(0.1),speed(1)]

+!move: o3(25) <-   !moveBack.


@moveTL[utility(0.15),speed(1)]

+!move: o3(20) <-   !goLeft.


@moveFree1[utility(0.2),speed(1)]

+!move: o3(30) <-   !goAhead.


@moveFree2[utility(0.3),speed(1)]

+!move: o3(30) <-   !stop.
```

As can be seen, the plans shares the same triggering event (`!move`), but some of them does not have the same applicability, as their contexts differs. However, in the case of those plans labeled as `moveFree`, and `moveFree2`, their contexts are the same. They have two annotations, one with the same information, namely `speed(1)`, but the other, related with its ranking information, is different: `utility(0.2)` and `utility(0.3)`. Despite both plans have the same triggering event, and context, the precedence is determined by its utilities. Consequently, the first plan to be selected when the event `!move` is fired, is `@moveFree2` due to its higher annotated utility.

### 4.3.1 A BDI Learning Agent

The architecture of IL in the framework of experimentation that interconnects NetLogo and Jason is depicted in Figure 4.4, including the internal action that implements IL, by extending the Jason agent architecture.

The *PlanLibrary* of the learning agents includes *primitive plans* to define the basic actions that an agent can perform in the environment. Non-primitive plans call primitives when an action is required to be performed in the environment. In any case, each plan is triggered as a consequence of its applicability to the current state of the environment and the current mental state of the agent.

Learning can be started in two cases: a) as a *failure handling* procedure related to errors raised by the absence of a plan to respond to a perceived event, i.e. the agent does not have a defined plan able to handle

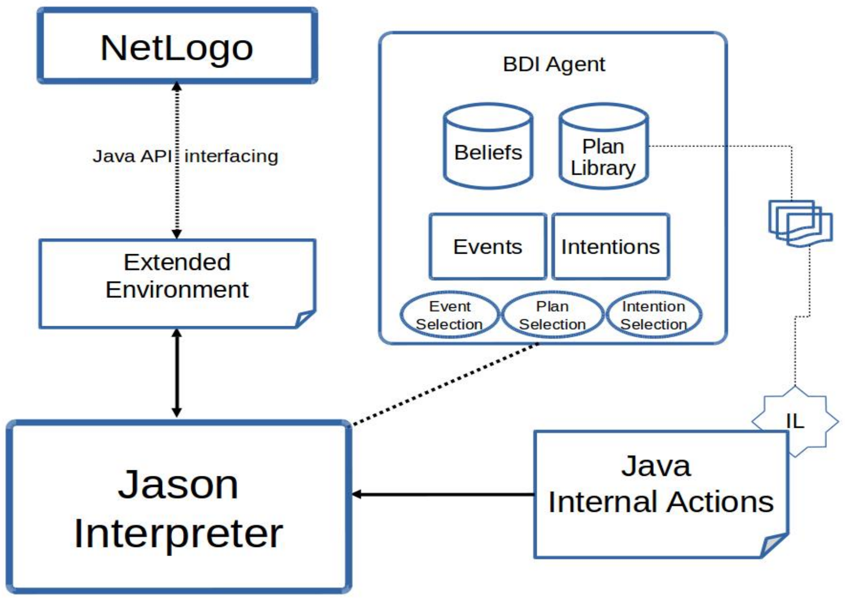


**Figure 4.4:** *The extended architecture of the framework to develop IL in Jason & Netlogo. Learning is implemented as a Java Internal Action that produce new plans added to the PlanLibrary.*

the state of affairs of the environment; or b) as a process of behaviour improvement related with a low performance of the executed plans. The first case is addressed as plan acquisition that drives the agent to the generation of new skills shaped by the second case, regarded as an adequacy process of the learned plans.

Once started, the Jason agent commands its NetLogo turtle avatar to begin the intended task. Whenever a *failure* is raised it is captured by a *meta-plan* that monitors the execution of plans, starting the learning action whose purpose is to add a new plan to the agent *PlanLibrary* according to the aforementioned templates, using the $\lambda$ criterion of counting and testing the applicability of actions.

The general IL process to add a new plan is shown in Algorithm 7, according to the terminology and constructs of Jason.

Basically, the process computes a set of $N$ applicable primitive action (`candidateActions`) 1, from either those plans that match the failed trigger `tFL`, or actions coming from other plans in the *PlanLibrary*. Then, `candidateActions` are ranked (line 2), and those actions matching the current beliefs of the agent (line 7) are added to the list of actions to be included in the plan (line 8). Actions applicability is determined by its consistency in the agent belief base using one of two tests: a) `ag.Believes()` determines if the current internal state of the agent entails the action context `cntx`; or b) `actionApplicable(newAction,cntx,`$\theta$`)`, to

**Algorithm 7:** ILGeneralPlanGeneration

```
Data: ag: the learning agent; tFL: the failed trigger, ex. !g;
θ: tracking beliefs
fpCntx: the current failed plan's context; N: the number of actions allowed for the new plan;
candidateAct: candidate actions to add in a new plan; rankedActions: ranked candidate actions;
Result: A new plan newPl added to the a's PlanLibrary
begin
     // Collect actions from PlanLibrary
1    candidateAct ← getActions(ag.PlanLibrary,tFL,fpCntx)
     // Rank and sort the actions according to their utility so far (λ criterion)
2    rankedActions ← rankActions(candidateAct)
     // Choose the N most convenient actions to include in the new plan
3    b ← false
4    for i ← 1 to N do
5        newAction ← rankedActions.get(i)
6        cntx ← newAction.getContext()
         // The list of actions starts with the first action entailed by the belief base
            or the tracking beliefs
7        if (ag.Believes(cntx) OR actionApplicable(newAction,cntx,θ) ) OR (b = true) then
8            bActions.add(newAction)
9            b ← true
         end
10       i++
     end
     // Build the the new plan:  @label te :  cntx <- body
11   newCntx ← selectCntx(fpCntx,bActions.first())
12   newLabel ← setLabel()
13   newBody ← setBody(bActions)
     // Set up the new plan in agent's PlanLibrary
14   ag.PlanLibrary.addPlan(newLabel,TFL,newCntx,newBody)
end
```

verify if the action appears in the tracking beliefs $\theta$ with the given context. The label of the plan is set adding the annotation `utility(U)`, where `U` is the utility associated to the plan, and the annotation `learn(1)` to identify the plan as a new learned one. Finally, the plan is built (line 14) and added to the *PlanLibrary*.

## 4.3.2 Learning Plans With One Action

The first template, described in 4.2.3.1, is outlined in Algorithm 8. It is a simplification of the general IL algorithm presented above, as it is required to select just one action. Basically, a set of applicable actions is collected (line 5), the top ranked is chosen as the action to be part of the new plan (line 7).

**Function** getActions(*Plans*,*tFL*,*cntx*): Get actions from plans

```
Data: Plans: a set of plans; tFL: a triggering event ; cntx: a plan context
RelPlans: a set of plans matching with the trigger tFL
Result: A list of actions from plans in RelPlans
begin
      // Get actions from plans matching the (failed) triggering event
1     RelPlans ← relevantPlans(tFL,Plans)
2     if RelPlans = ∅ then
          // Otherwise, from the entire set of plans
          RelPlans ← Plans
      end
3     foreach plan in RelPlans do
          // Update context and take actions from current plan's body
4         pActions ← plan.getBody()
          // Get the candidate actions from the current plan body
5         foreach pAct in pActions do
              if isAction(pAct) then
6                 cActions.Add(pAct)
              end
          end
      end
end
```

## 4.3.3 Learning Plans With Repetitions of Actions

The template to generate a plan with *N* repetitions of the same action in its body (shown in 4.2.3.2), is outlined in Algorithm 9. Similarly to learning plans with one action, learning repetitions applies for failure-handling or improving the plans when pursuing a given task. In the former case, default plans cannot be applied (for instance, when a tunnel-like structure appears in the route of the agent as in the example shown above). Learning is able to acquire a new plan with repetitions of action in its body 8. Then agent can continue operating with a new plan, recovering from a possible failure.

## 4.3.4 Learning Plan With Sequences of Actions

The plan template for learning alternate sequences of actions (4.2.3.3), is showed in Algorithm 10.

The operation of this learning template benefits from tracking beliefs to collect the interactions with the environment, i.e. the fired actions, before a failure occurs to track the sequences of valid actions based on their past applicability. It is briefly explained as follows.

Once the set of plans (principally, those who match the failed trigger `tFL`) are collected (line 1), they are ranked (line 2).

**Algorithm 8:** ILOneActionPlanGeneration.

```
Data: ag: the learning agent; tFL: the failed trigger, ex. !g; fpCntx: the current failed plan's context;
      candidateAct: candidate actions to add in a new plan;
Result: A new plan newPl added to the a's PlanLibrary
begin
    // Collect and Rank actions from PlanLibrary
1   candidateAct ← getActions(ag.PlanLibrary,tFL,fpCntx)
2   rankedActions ← rankActions(candidateAct)
    // Choose the applicable actions
3   foreach cAction in candidateActions do
4       cntx ← cAction.getContext()
5       if ag.Believes(cntx) then
6           bActions.add(pAct)
        end
    end
    // Choose the most convenient action to include in the new plan
7   act ← getMaximum(bActions)
    // Build the the new plan:  @label te :  cntx <- body
8   newCntx ← selectCntx(fpCntx,act)
9   newLabel ← setLabel()
10  newBody ← setBody(act)
    // Set up the new plan in agent's PlanLibrary
11  ag.PlanLibrary.addPlan(newLabel,TFL,newCntx,newBody)
end
```

Context `pcntx` is initialised with the value of `cntx` (line 4) in order to query the belief `tAB` looking for an applicable action `cAct`. From the tracked beliefs, the learning performs a sequence, by matching the contexts of the past fired actions (`while` loop in line 6) storing in the list `bActions` (line 8). In addition, the the context of the first literal must be a logical consequence of the agent's beliefs at the current moment, i.e. the current perceptions of the environment (line 7). Contexts and applicable actions are updated accordingly to the belief's content (lines 9 and 10). The process continues until it collects at most $N$ actions.

## 4.4 Experiments and Results

As discussed in the literature review IL is not a set of algorithms, but a framework of learning [117] in the sense that it provides the general framework to accomplish learning (like plans updating/generation) within intentional systems, namely, BDI agents.

The experimentation is mainly concerned with demonstrating how IL is integrated in the BDI architecture to generate new plans to be added into the *PlanLibrary* of the learning agent in order to increase its range of

**Algorithm 9:** ILRepetitionOfActionsPlanGeneration.

```
Data: ag: the learning agent; tFL: the failed trigger, ex. !g;
fpCntx: the current failed plan's context; N: the number of actions allowed for the new plan;
candidateAct: candidate actions to add in a new plan; bRActions: a list of actions;
Result: A new plan newPl added to the a's PlanLibrary
begin
        // Collect and Rank actions from PlanLibrary
1       candidateAct ← getActions(ag.PlanLibrary,tFL,fpCntx)
2       rankedActions ← rankActions(candidateAct)
        // Choose the applicable actions
3       foreach cAction in candidateActions do
4           cntx ← cAction.getContext()
5           if ag.Believes(cntx) then
6               bActions.add(pAct)
            end
        end
        // Choose the most convenient action to include in the new plan
7       act ← getMaximum(bActions)
        // Adding the repetitions the new plan
8       for i ← 1 to N do
9           bRActions.add(act)
10          i++
        end
        // Build the the new plan:  @label te :  cntx <- body
11      newCntx ← selectCntx(fpCntx,act)
12      newLabel ← setLabel()
13      newBody ← setBody(bRActions)
        // Set up the new plan in agent's PlanLibrary
14      ag.PlanLibrary.addPlan(newLabel,TFL,newCntx,newBody)
end
```

skills and reaction to the environment.

There are two main scenarios when learning is applicable.

- *Failure handling and plan generation* : The case when the agent has no plan to cope with the current state-of-affairs in the environment raises a failure. In other words, in a given state the agent fails to perform an action, and has no applicable plan to respond to the environment. However, this failure can be handled by means of learning.

  Learning begins when no applicable plan is found in the *PlanLibrary* for coping with the current state. The purpose of learning is to generate a plan to handle this failure, providing an answer to cope with the current state of the environment. The learning process is just required to find a matching action with the current perceptions of the environment to produce a new plan. This is reached by observing the utility of each plan (See equations 4.1 and 4.2, Section 4.2.1 above). With this, the agent can select

**Algorithm 10:** ILSequencesOfActionsPlanGeneration.

```
Data: ag: the learning agent; tFL: the failed trigger, ex. !g; θ: tracking beliefs
fpCntx: the current failed plan's context; N: the number of actions allowed for the new plan;
candidateAct: candidate actions to add in a new plan; rankedActions: ranked candidate actions;
Result: A new plan newPl added to the a's PlanLibrary
begin
    // Collect and Rank actions from PlanLibrary
1   candidateAct ← getActions(ag.PlanLibrary,fFL,fpCntx)
2   rankedActions ← rankActions(candidateAct)
3   act ← getMaximum(rankdeActions)
    // Choose the N most convenient actions as sequences
4   pcntx ← fpCntx
5   first = false
    // Get a sequence based on tracking belief tAB(preContext,action,postContext,...)
6   while (action = cAct in θ.tAb with preContext=pcntx) and (|bActions| < N) do
7       if (ag.Believes(pcntx)) OR (first = true) then
8           bActions.add(cAct)
9           pcntx ← postContext in θ.tAB with action = cAct and preContext = pcntx
10          act ← action in θ.tAB with preContext=pcntx
11          first = true
        end
    end
    // Build the the new plan:  @label te :  fcntx <- body
12  newCntx ← selectCntx(fpCntx,bActions.first())
13  newLabel ← setLabel()
14  newBody ← setBody(bActions)
    // Set up the new plan in agent's PlanLibrary
15  ag.PlanLibrary.addPlan(newLabel,TFL,newCntx,newBody)
end
```

the most convenient action matching the state-of-affairs at the time learning began.

- *Acquiring new skills* The purpose of IL aims to go beyond the error-handling, pursuing the acquisition of new skills. In other words, through increasing the *PlanLibrary* (the procedural knowledge of the agent) can derive in the behaviour of the agent, as it has new options to react in the environment. Furthermore, going beyond the acquisition of simple behaviours, requires to use more challenging cases for learning.

  Additionally, apart from templates to build plans with only one action, the generation of plans with more than one action in their body can potentially give rise to more useful behaviours. This is due to the interaction with the environment, and the new set of actions included in them. So, is also worth to considering also other learning templates, such as those with repetitions and sequences of actions.

IL is demonstrated in the next section based on the two cases of learning mentioned above. Firstly, In subsection 4.4.2, a failure is handled by reaching a very simple behaviour acting as a proof of concept: manage tunnel-like structures. Secondly, in subsections 4.4.3 and 4.4.4, two behaviours are acquired only from a set of primitive plans: target-searching and wall-following. Thirdly, learning is tested in the D-R domain, where the target is to acquire ambulance behaviours.

In all these three examples, new behaviour are acquired virtually from scratch (working with primitives only) as they are not defined by the original set of movement plans.

### 4.4.1 Description of the experiments

The purpose of the learning is to acquire a new plan accordingly with the current primitive actions and their applicability in the context where the default plans cannot be applied. It is focused on the learning of a plan within the vacuum cleaner environment, defined in Section 3.3.3 of Chapter 3.

Learning makes use of the three algorithms defining the plan templates to deal with the generation of new plans for updating the *PlanLibrary*: One action, repetitions, and sequences of actions. Monitoring meta-plans start the learning by calling a Jason Java-based internal action that implements these algorithms.

In Figure 4.5, a snippet of the meta-plans that trigger the IL is showed.

As stated in Section 3.3.3 within the vacuum cleaner environment the agent has two types of plans at its disposal: *movement* plans and *primitives*. Movement plans are triggered by the agent's main loop of reasoning, whenever a movement is required in the environment. Once any of these plans are triggered (i.e. once applicable, due to the environment and the current mental state of the agent), the plan fires a primitive. Being a very simple scenario, the repertoire of primitive actions is narrow, just three actions: `moveForward`, `turnLeft`, and `turnRight`.

Figure 4.6 shows the snippets of Jason code belonging to the non-primitive plans that allow the agent to move in the environment.

To perform IL, the following beliefs are defined :

- Beliefs to record both the times an action has fired, and the times a plan to move has been triggered: `planUtilities/3`, `planUtilities(PlanLabel,Utility,Frequency)`.
- Plan utilities are stored as plan annotations: `utility/1`, `utility(U)`.
- Collect the *sequence of executed actions* whilst the agent interacts with the environment before a failure

```
// Learning plan
// The context catches the current perceived state
+!planLearning(AnyPlan,TrigFL,ErrorId,FPlan,UFP)
    : o1(O1) & o2(O2) & o3(O3) & o4(O4) & o5(O5)
    <-
        ...
    // Call the internal action in file intLearning.startLearning.java
    Cntx = [o1(O1), o2(O2), o3(O3), o4(O4), o5(O5)];
    intLearning.startLearning(TrigFL,ErrorId,Cntx,moveL,Template,NL,UFP);
    ...
    // Save the agent definition after adding a new plan
    .my_name(AgName);
    .concat(AgName,"New",NL,".asl",NAgName);
    .save_agent(NAgName);
    ...
    .
// Meta-level plan to detect falure in plans with move as triggering event
-!P[error(ErrorId), error_msg(Msg), code(CodeBody)]
    : P =.. [AnyPlan,Ts,As] & (AnyPlan = move)
    <-
        ...
    // Test the Belief base to get the last plan fired
    ?lbLastMove(FPlan);
    // Retrieve the utilities of the last plan fired
    ?planUtils(FPlan,UFP,C);
    // Fire the IL plan
    !planLearning(AnyPlan,CodeBody,ErrorId,FPlan,UFP);
    // Retry P
    !P.
```

**Figure 4.5:** *Meta-plan in Jason code for monitoring failures and trigger IL.*

occurs. It takes into account the pre- and post-buffer of perceptions regarding the visibility of the environment sensed by the agent, i.e. the array of cells in its line of sight <`o1..o5`>.

The sequence of fired actions is tracked by the agent through the Java based file `AgFile.java` modifying its action-selection function and meta-plans implemented in Jason. The tracking beliefs defined to collect the sequences are:

- `tAB/13`: Collect the transitions of the context when a primitive `PlanPrim` is executed. For instance: `tAB(PlanName,PlaLab,PlanPrim,PrimCntx P1..P5,CurrCntx <O1...O5>)`.

  Where: `PCntx` and `CurrCntx` are variables representing the perceived cells in the agents vision when the plan `PlanName` was executed, i.e. the array of cells in its line of sight: `[o1(X),o2(X),o3(X),o4(X),o5(X)]`.

- `lastMove/13` collects the past execution of a plan for moving (the last previous movement action). For instance: `lastMove(PLab,Plan,LbAn,PastCntx,CurrCntx)`. Where: `Plan` is the name of the last

```
/*
    Perceptions of the environment for agent Ag:
    o2 |     o3   |    o4
 --------------------------
    o1 |     Ag   |    o5

Represented as: [o1(d1),o2(d3),o3(d3),o4(d4),o5(d5)]

    Background color = 0  (black in NL)
    Obstacle color   = 25 (orange in NL)
    ...
*/
@moveF[utility(0.1),learn(0)]
+!move: o3(O) & not O = 25                            // the cell ahead is free
        <-      !moveForward.

@moveTL[utility(0.1),learn(0)]
+!move: (o3(25) | o3(0)) & (o1(O) & not O = 25)       // cell to the left is free

        <-      !turnLeft.

@moveTR[utility(0.1),learn(0)]
+!move: ((o3(25) | o3(0)) & (o5(O) & not ((O = 25)))) // cell to the right is free
        <-      !turnRight.
```

**Figure 4.6:** *Snippets of Jason code belonging to the plans to traverse the environment.*

executed plan of movement, `PastCntx` is the context in the previous step before it is executed, and `CurrCntx` the context when it was actually triggered. `PastCntx` and `CurrCntx` are similarly defined as the previous belief.

Briefly, the method allows different criteria (the plan templates) to generate plans with *N* actions applicable under a given *cntx*, validated in its applicability through the tracking beliefs. The expected result looks like:

```
@planLX[learn(1),utility(N)]

+!event_trigger  : cntx   <-  Body
```

Where, *Body* = {*A*1...*AN*}, is a set of actions inserted accordingly to any learning template.

## 4.4.2 Error-handling Through Intentional Learning

This experiment is aimed to show how plan acquisition by IL can be used for error-handling. The failure is managed by adding a new plan with only one action in its body. The agent is allocated in the vacuum cleaner environment as depicted in part (A) of Figure 4.7 below. Being a simple problem, it illustrates the generation of new abilities in a BDI agent by extending its *PlanLibrary* through IL.

Briefly, learning proceeds as follows:

1. After starting, the agent explores the environment, and a failure is raised due to not any applicable plan being found (when it reaches a tunnel in the environment). This is depicted in part (B) of Figure 4.7.

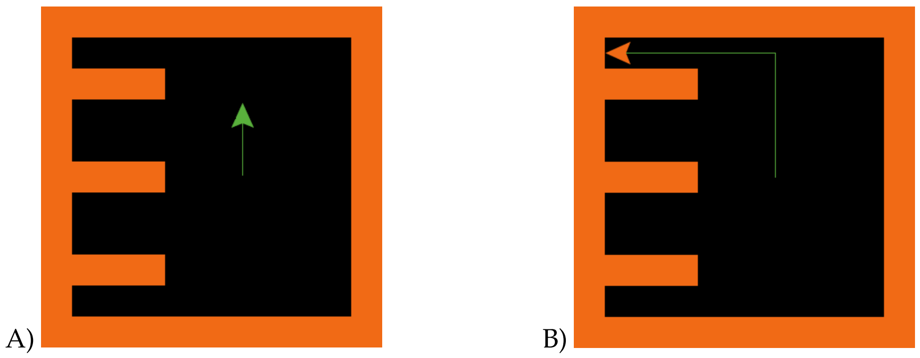


**Figure 4.7:** *In (A) the agent start exploring the environment using its predefined set of plans. In (B) the agent reaches a tunnel-like structure, as the plans in its PlanLibrary cannot cope with this situation, the learning procedure is started using some template.*

2. This failure is managed by the learning process to generate a new plan in AgentSpeak grammar:

   `@label failed_trigger : current_context <- action`

   Learning is started using some template (passed as a parameter to the learning function in Java).

   In the case of plan template with *one action* (4.3.2), learning analyses the matched plans (with the failed trigger) collecting candidate instructions. In this case due to it has been triggered more times than others, `!moveForward` could be the best candidate, according to the ranking so far. However the context of the primitive does not fit the current agent state (this is precisely the failed one!). Then the rest of the actions should be evaluated (i.e. `!turnLeft` and `!turnRight`). The procedure prefers the action with the higher score of utility to be inserted in the new plan, in the example it is `!turnLeft` (it has been used four times, while the other actions has not been used yet). Consequently, `!turnLeft` instruction is selected to be included in the new plan:

```
@moveL1[learn(1), utility(0.1)]
+!move : (o1(25) & (o2(25) & (o3(25) & (o4(25) & o5(25)))))
  <-
    !turnLeft.
```

   Similarly, using the plan template to generate *repetitions of actions* (4.3.3), `!turnLeft` is the selected. The plan is generated allowing $N$ actions into a plan, where $N = 3$. The result is shown in the following plan:

```
@moveL1[learn(1),utility (0.1)]
+!move : (o1(25)&(o2(25)&(o3(25)&(o4(25)&o5(25)))))
  <- !turnLeft; !turnLeft; !turnLeft.
```

Finally, two examples of plans generated with *sequences of actions* (4.3.4) are showed as follows.

```
@moveL1[learn(1),utility(0.1)]
+!move(C) : (o1(25)&(o2(25)&(o3(25)&(o4(25)&o5(25)))))
  <-  !turnLeft;  !turnRight; !turnLeft.


@moveL2[learn(1),utility(0.1)]
+!move  : (o1(25)&(o2(25)&(o3(25)&(o4(25)&o5(25)))))
  <-  !turnLeft; !turnLeft;
      !moveForward; !moveForward; !moveForward.
```

3. The new plan added to the *PlanLibrary*, and the agent's code is saved to hard disc to keep the changes.

4. Once the agent finishes its learning process, it uses the new plan, and now it can manage tunnel structures in the environment as can be seen in part (B) of Figure 4.8.

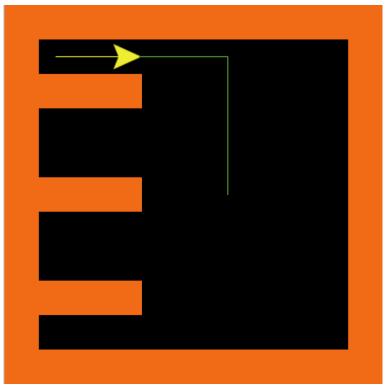


**Figure 4.8:** *After the learning procedure is finished, a new plan is added to the PlanLibrary, and the agent can now cope with tunnel-like structures.*

As can be seen, those plans are enough to accomplish the task of escaping from tunnel-like structures in a given scenario. In the case of one action template, the generated plans, being so simple, fulfil the task in a

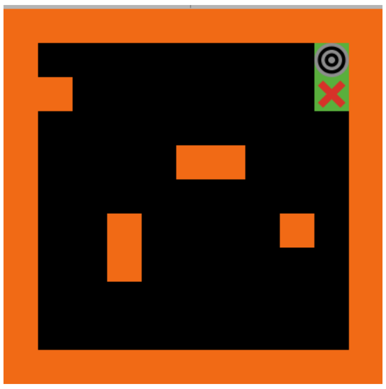


**Figure 4.9:** *Default target-searching maze used to test the acquisition of this behaviour.*

more efficient way (i.e. the repetitions and sequences introduce more actions than those strictly required). As a consequence, for the mentioned task, plans with more than one action should be reviewed, to avoid overuse of actions. Furthermore, plan `@moveL2`, may be reviewed as the agent could potentially reach a state-of-affairs when any of the `moveForward` instructions are non-applicable, for the agent is facing a boundary instead of empty space, violating their applicability. In such a case, it will raise another failure, and its body would be modified through plan revision. Plan revision is particularly important when pursuing more complex tasks.

### 4.4.3 Target-searching

This subsection is focused on showing how the target-searching behaviour can be acquired by means of IL. In Figure 4.9, the default maze of the experimentation framework for testing the target-searching behaviour is shown.

Due to the influence of RL techniques (particularly Q-Learning) in the method to include IL in fully-fledged BDI agents for plan acquisition, is interesting to highlight that it is not a mere implementation of Q-Learning using Jason and NetLogo. Q-Learning is an approach of learning based on evaluating the interaction of the agent and its environment, when performing an intended task. On the other hand, IL is based on monitoring the agent intentions identifying when learning is needed for the purpose of performance improvement. Learning is focused on a plan acquisition strategy addressing the cases of plans composed of one action, sequences or a repetition of actions that allow an agent to improve its behaviour at run-time. This is done at the pure BDI agent level.

On this basis, in this subsection a couple of implementations of agents that acquire the target-searching behaviour are presented. The former is based on a pure implementation of RL, while the latter is the application of IL to acquire the target-searching behaviour in single intentional agents.

A relatively recent work includes in a direct way the RL mechanism for action selection and indirectly undertakes IL at the level of goal adjustment. Basically, it is an implementation of Temporal Difference Learning in Jason, that attempts to bridge the gap between RL and BDI systems [11]. Despite the purpose of the experiment being to include new plans in the BDI framework, it is important to contrast the result between both systems, so I implemented a version of the work reported in [11]. In this implementation the environment has two possible final states: the correct one (the target), and the wrong one. These states convey a reward of 1 or $-1$, accordingly, as the RL mechanism needing to establish an optimal path from the starting point (randomly selected) to the correct final state. The agent needs to compute the difference in the cumulated utility of the states, from the starting state to the ending one (i.e. the route the agent needs to take from a starting point to the target). Figure 4.10 displays a couple of snapshots of the agent running up to 2000 and 3000 trials (a trial ends when the agent is in a final state). The first one, (a), is ending in the wrong cell after 2000 trials, whilst (b) ends in the right one after 3000. Coloured traces of the agent are shown (a different colour per trial).

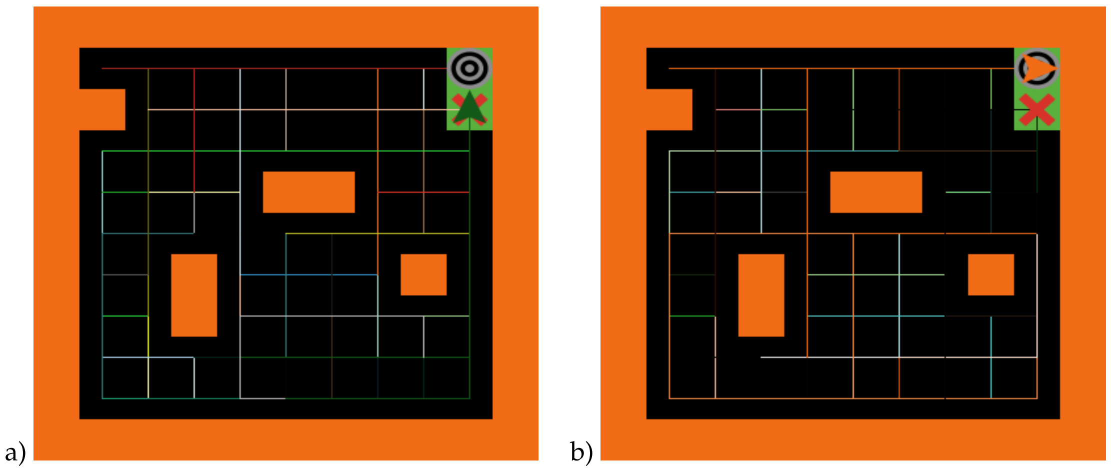


**Figure 4.10:** *The agent running up to (a) 2000 trials ending in the wrong cell, and (b) 3000 trials ending in the right one.*

Additionally to the work reported in [11], I performed a brief analysis of the performance of the agent after training in the default maze of Figure 4.9. The number of actions per trial is plotted in Figure 4.11. The first 2,500 trials correspond to the training, while the rest of the plot shows the testing stage. The number of actions increased during the training, due to the comparison the agent needs to do whilst examining surrounding cells, in order to take the most promising path at each state. But accordance with the $e - greedy$ strategy, it takes random cells in $1 - \epsilon$ of the times. Whilst testing, the agent just takes the most promising cell each time.

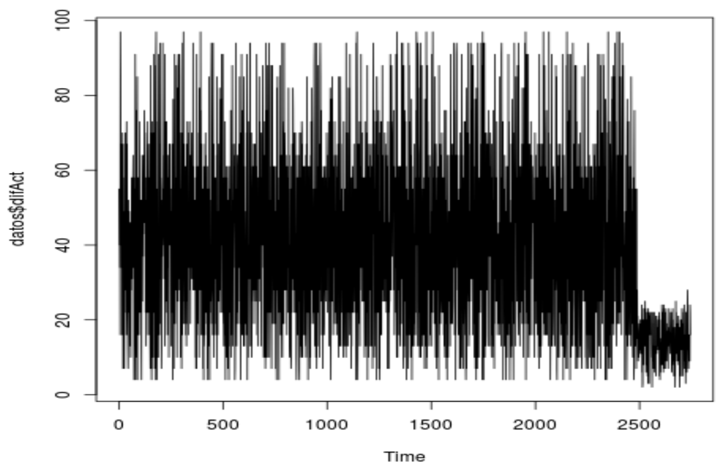


**Figure 4.11:** *Trials and actions while the pure RL-based agent is performing the target-searching behaviour while training (up to 2,500 trials) and during testing (the rest of the plot) in the default scenario.*

On the other hand, the results of the IL based agent show how the agent starts, from almost randomly exploring the space to a set of plans that are focused upon searching for the target in the designated cell, as in the case of the pure RL based agent. The difference in the maze is that it does not include a wrong state, as the IL agent's purpose is not to reach an optimal path (i.e. differently to the pure RL-based agent). The IL-based agent's environment representation does not comprise of mapping of the entire environment, but the generation of suitable plans to execute a target searching behaviour. However, the wrong final state can be added to the maze. It is a matter of including a representation of the wrong target to easily prevent the agent from proceeding further, so I decided to skip this state. In addition, the agent was run in different mazes. In Figure 4.12 snapshot of the agent is shown running in different mazes. The trace of the agent while traversing the scenario is shown as well to identify the capability of the agent to explore the environment.

As it was stated in Subsection 2.5.1 of Chapter 2, it is important to note that according to the literature [101, 105] there is a relation between both systems. Specifically, a BDI agent can be translated to an agent based on a Markovian Decision Process (MDP) or Partially Observable MDP, and vice versa. Even in [105], algorithms for bidirectional conversion between a BDI agent and an MDP are provided, demonstrating that they can be

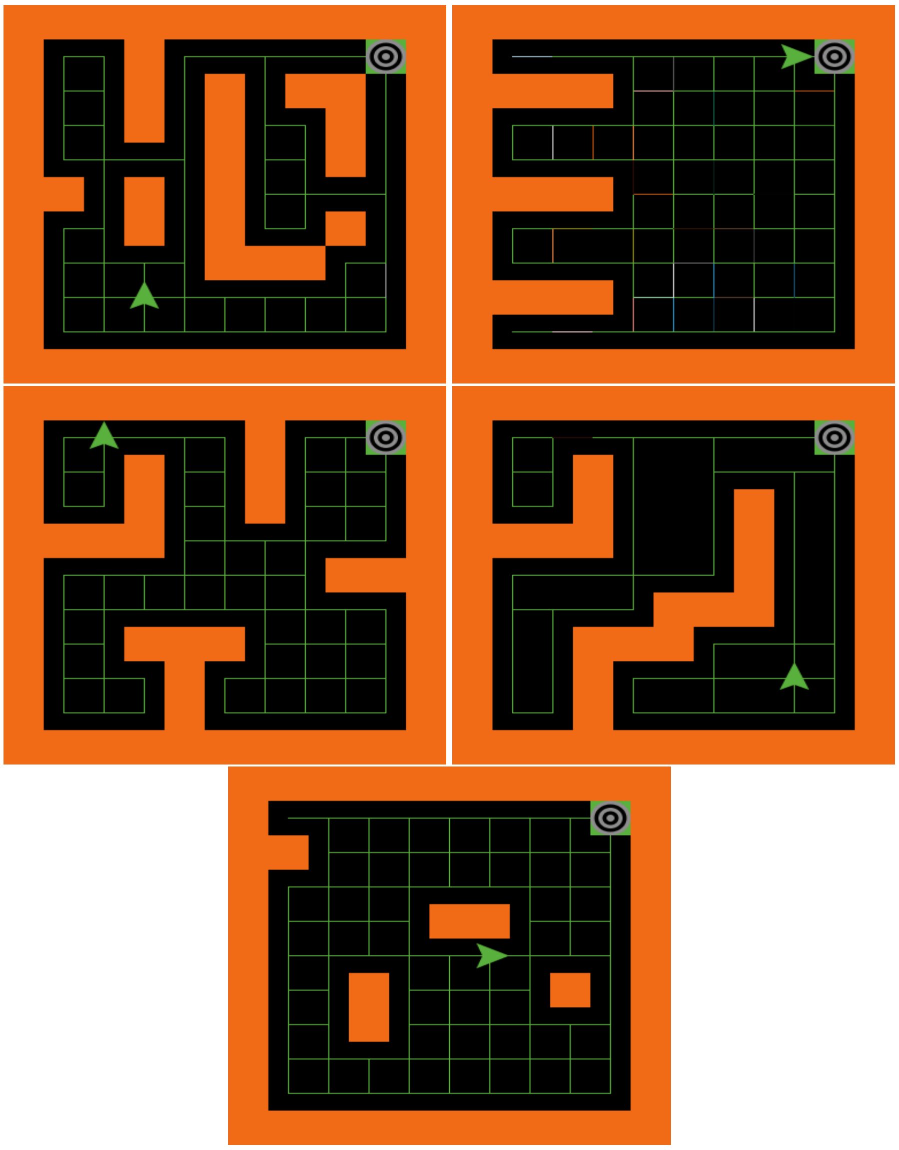


**Figure 4.12:** *The Intentional Learning-based agent while training on the target searching in different scenarios up to* 10, 000 *actions.*

equivalently modelled, as long as the transition function for actions in the environment is known (it is barely the case in BDI agents). The main drawback then is that at least one of them must be already solved, or modelled to be translated into the other. In the case of modelling a BDI agent from an already solved policy, it is not applicable in practice (why not simply use it, if you have it?). Regarding the inverse case, it seems useful when the complexity of the problem makes it intractable to use an MDP [105]. In conclusion, while a BDI agent can manage scenarios with higher degree of difficulty, the MDP-based agents overcome the BDI agency optimally.

The trade-off is that any BDI agent can be used when the complexity of the environment surpasses the MDP capabilities, whilst reaching a sub-optimal performance. Bearing this in mind, what is expected on the side of the BDI agent is to acquire a set of plans that undertakes the task with a lower performance than the MDP-based agent even if it is implemented in Jason.

Figure 4.13 compares the plots of both agents running after the training up to 290 trials. As usual, each trial ends whenever a collection of targets takes place. The X axis is the number of trials, while the Y axis shows the number of actions per trial.

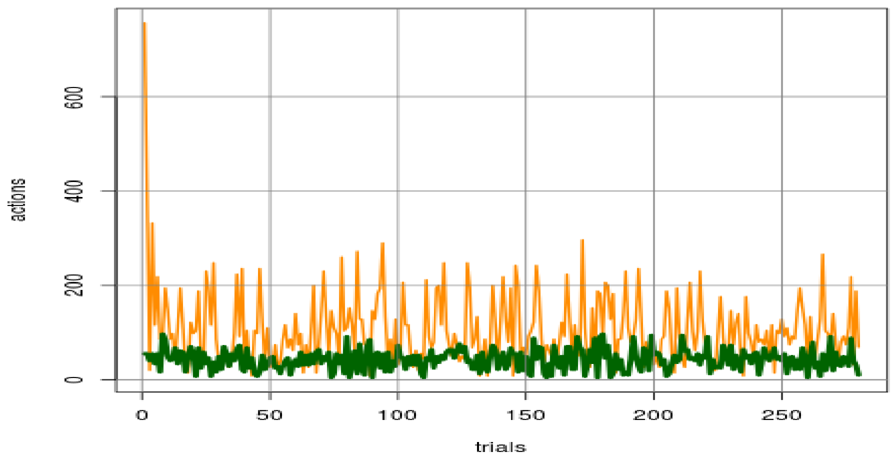


**Figure 4.13:** *Comparison between the Reinforcement and Intentional Learning-based agents performing the target searching behaviour after training, in* ***green*** *and* ***orange****, respectively.*

Additionally, it is important to state how learning in intentional systems can benefit from RL in the acquisition of new behaviours, based on the interaction between the agent and the environment (i.e. a RL inspired method, to rank the actions to be part of a plan, and drive the agent during the plan acquisition by selecting the best-rated actions whilst interacting in the environment, and tracking the action's contributions to the intended behaviour).

Finally, with this comparison its worth noting the difference between a mere implementation of RL methods in Jason, which is based on rating the states working in a simple environment (like the vacuum-cleaner). In other words, it pursues a mapping of states and utilities in the scenario in order to discover the optimal paths, but making it difficult to extend this setting to greater environments. Additionally, the result is not a plan at all, but a selection criteria that determines the next action based on the beliefs and the ranked states (i.e. the plan to do it is fixed, it does not change). On the other hand, the IL method using the specified framework (a fully-fledged BDI system, interconnected with NetLogo as the rendering tool for the environment where the agents are situated) can reach the target-searching behaviour at the cost of efficiency (i.e. by executing more actions to reach the target in comparison with the pure RL based agent), but resulting with a set of plans for performing the intended behaviour.

### 4.4.4 Wall-following

This task has as a purpose: that the agent follows walls only on the left side. The behaviour is reached after hundreds of interactions (sensing-acting) with the environment, the result of generating new plans and utility tuning as the agent traverses the environment. It is important to note that the agent has restricted scope of sight, making it impossible to sense when it has left or reached a wall through a corner, increasing the task's degree difficulty.

The wall-following behaviour is depicted in Figure 4.14 using three different mazes. Yellow lines mark when the agent touches a wall on the left side.

In Figure 4.15 the accumulated times a cell has been visited in each of the mazes, using the template with one action in the plan body, is shown. The darker cell, the more visited one. As can be seen, the boundaries of the environment are the most visited, illustrating wall-following behaviour.

Table 4.2 includes a summary of the plans after 5 runs, showing the number of useful actions (i.e. the times when the agent touches a wall with the left side), their reviews, and the liveness of the plans (their execution frequency), denoting how a plan can be re-used during the run due its applicability and its adequacy. Plans with more than one action require more reviews than others. This is a consequence of the narrow scope

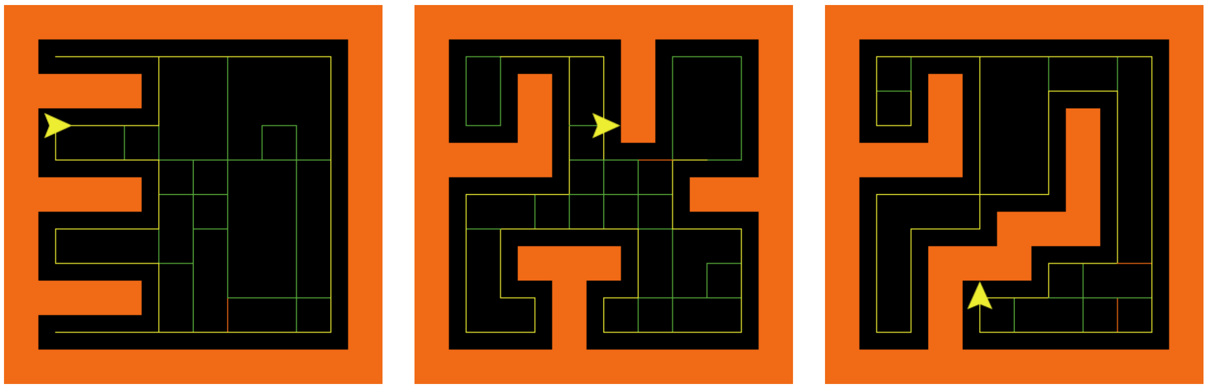


**Figure 4.14:** *The agent achieving the left-hand wall-following behaviour in different mazes.*

of applicability in plans with more than one action in their body (i.e. the plan context is determined by the circumstance where an error arose or a revision was performed). The applicability or benefit of the actions can be affected, as the state-of-affairs in learning could be different to those exhibited when the plan is fired. Consequently, longer plans receive less feedback from the environment, so may need to be constantly reviewed. Furthermore, the learning template with one action in their body populates the agent with a greater number of plans than the other templates.

| M | ILT | LrPl | PRev | PLiv | UsAc |
|---|---|---|---|---|---|
| 1 | O | 21 | 827 | 406 | 548 |
| | S | 19 | 445 | 85 | 135 |
| 2 | O | 27 | 867 | 370 | 501 |
| | S | 24 | 814 | 535 | 431 |
| 3 | O | 26 | 806 | 408 | 538 |
| | S | 19 | 816 | 119 | 139 |

**Table 4.2:** *Summary of the wall-following behaviour tests on diverse mazes, running up to 3500 movements, and testing up to 1750 actions. Letters stand for Maze (M), Template (ILT), Learned Plans (LrPl), Revisions (PRev), Plan Liveness (PLiv), Useful Actions (UsAc), where (O) denotes one action and (S) refers to sequences of actions.*

Finally, in Figure 4.16, A) shows the plan utilities of an agent running learning using different plan templates. As explained before, in addition to the plan applicability, the action-selection function considers the plan utility, to establish a precedence in plan execution. Accordingly, plans with better rewards and utility will be selected most frequently. In B) a plot of the plans generated by the three templates is shown while running up to 3500 actions. As can be seen, the strategy that includes more plans is the one action template. This is explained because longer plans include more actions, so take longer to be evaluated, thus delaying the addition of new plans. It has an impact upon training, as more cycles are required for the repetition and sequences.

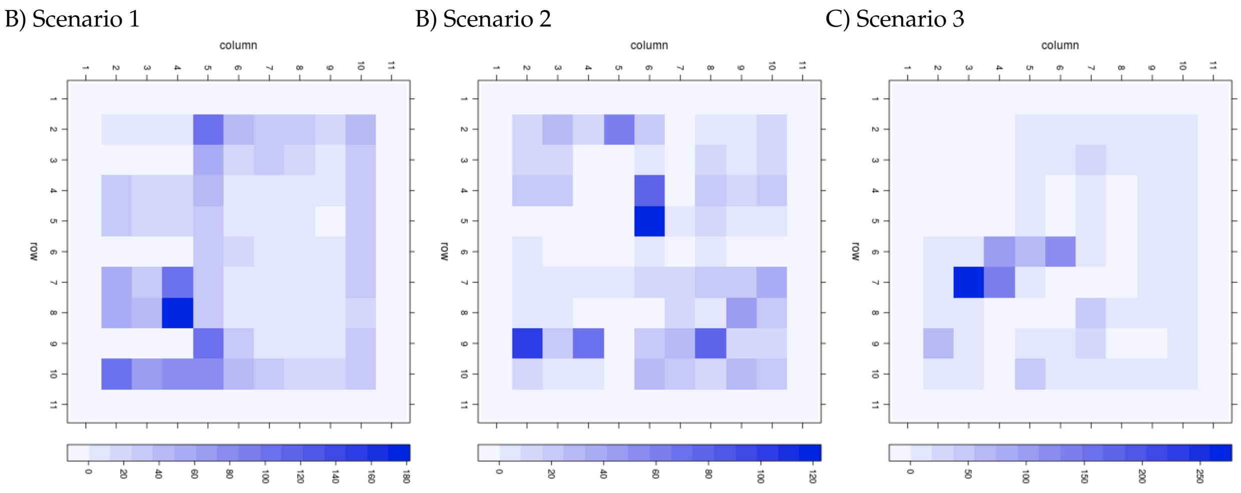


**Figure 4.15:** *Cumulated cell visits in the mazes, using the template to add one action in the plan.*

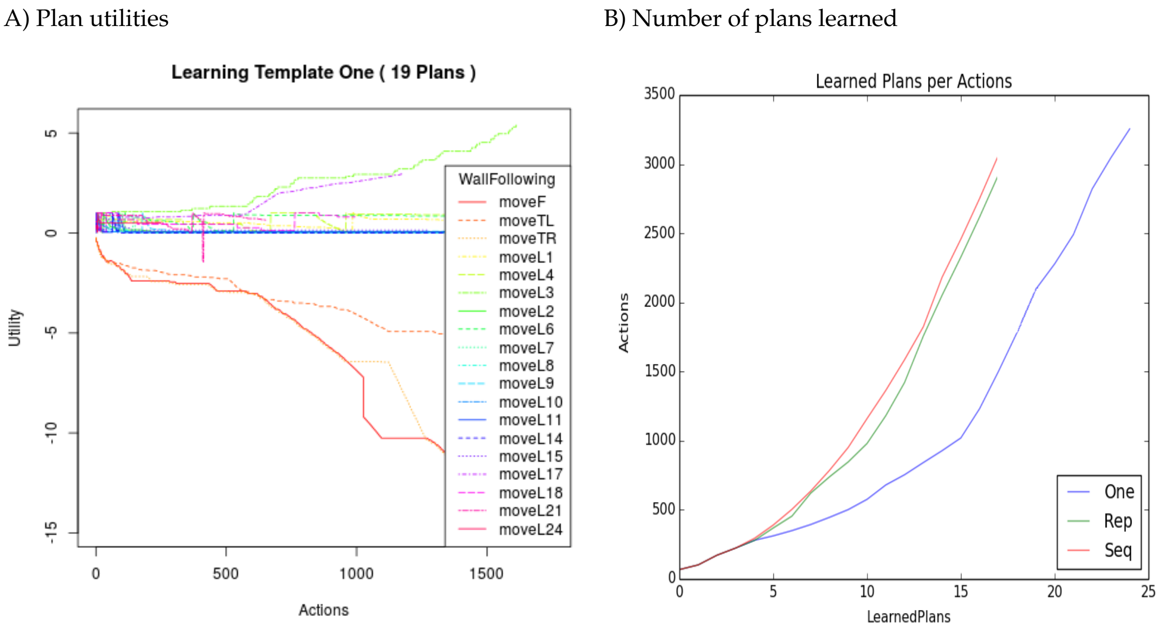


**Figure 4.16:** *A) Plan utilities when the agent is running, using the one action plan template B) Plot of the plans generated by the three learning strategies in the same scenario.*

The agent starts exploring the environment, and learning fires as the utilities of the plans decrease. The behaviour is reached jointly, with the addition of new plans and plan revision.

### 4.4.5 Disaster Rescue Domain

In the MAS context learning can be deployed either at the individual level or at the collective. In this section, IL in the individual scope within a set of Jason BDI agents, applied to the D-R domain, are presented, plus the results.

#### 4.4.5.1 Intentional Learning in the D-R Domain

Jason BDI agents are located in a NetLogo environment that simulates a city under disaster conditions, as described in Chapter 3, Section 3.4. This is the scenario where the agents perform actions, and from where they gather their perceptions. A *city map* is rendered on top, as a NetLogo model, in a grid-like way: each pixel of the map is depicted as a cell in the model. *Buildings* and *streets* are segmented to allow the agents to move only through the latter. **Environmental elements**, namely objects, are configured through NetLogo agents (*turtles*): one *base*, and several instances of *obstacle*, *fire*, *blockade* and *victim* to represent a civilian in danger. *Rescue staff* are also represented as NetLogo agents, to sense the environment and perform actions on behalf of Jason agents: *ambulance*, *rescue-unit*, *fire-brigade* and *police*. As *victim* is the most static agent, its behaviour can be considered as a pure reactive one, so those agents have behaviours not governed by Jason, but which are handled purely by NetLogo.

On the other hand, agents representing the rescue staff keep the deliberation process on the Jason side. NetLogo turtles perform actions on their behalf executing the commands and directions given by Jason agents through the plans included in their *PlanLibrary*, and collect perceptions as required. Lower-level functions (such as navigation) are managed directly by turtles. Both NetLogo and Jason agents are linked by an identifier, and whenever sensing and acting is needed, a translation step from Jason code to NetLogo and vice versa is performed through the interfacing Java classes.

A basic policy of task allocation is implemented, based on dividing the map in *N* sectors (experimentally set to 4). Where the teams of agents are assigned, each team has a similar number of agents. The system as a whole behaves in a decentralized way, for no agent leads the teams or the activities. *brigadier* agents message others only when a target is found.

Rescue agents have a narrow sight of view, just 10 pixels ahead in an angle of 90°. They have the following set of skills coded as sets of plans in their *PlanLibrary*, representing the main scenarios of their behaviour[2]:

- *doSearch* [Brigadier]: perform the search of civilians, obstacles and fires in their designated sector. Once

[2]In the Prometheus methodology scenario stands for an abstract description of a particular sequence of steps within the system.

an objective is found, it is added as a belief, and a suitable agent is notified.

- *goToObjective* [Ambulances, Fire brigades, Police]: traverse the environment looking for civilians, obstacles or fires to perform rescue actions, and
- *rescueAction* [All]: once an objective is reached/detected, agents perform actions like rescue, clear, pickup, or extinguish, as applicable.

Jason agents are in charge over the deliberation process, requesting, sensing, and ordering action execution in the environment to their corresponding turtles, such as movements to traverse the streets, perform rescue actions, and many others.

Learning behaviour is implemented only in *ambulances* as an additional internal action using an implementation that extends the Java classes of the Jason *Agent* architecture.

The purpose is to reach the ambulance's behaviour almost from scratch, based on the primitives, whose unique function is to compound a NetLogo command to be sent to the environment (the actual NetLogo model) to then be executed by turtles. The rescue behaviour is triggered as a request coming from *brigadier* agents. *ambulances* decide who to rescue following the received requests in a first-input/first-output basis, but they lack from an explicitly defined plan to perform the **rescue behaviour** (*ARB*) briefly described as:

1. Go towards rescued victims
2. Check its loading capacity
3. Collect victims, and
4. Carry victims towards the base of the city

In other words, with the starting set of plans, *ambulances* cannot perform *ARB*, such set of plans is the objective of the learning process.

All the procedural knowledge the agent has at its disposal is encoded as plans in the *PlanLibrary*. Consequently, learning at the level of activities is regarded as plan acquisition.

Hence, IL in the D-R domain is focused on the following **learning task** for an *ambulance* agent *ag*:

**Learning Task.** Given a set of rescue actions (primitives) $R_a \subset PlanLibrary$ available to $ag$, determine the relevant primitives $r_a \in R_a$ to generate a set of plans $ARB_{LP}$ that allows $ag$ to perform $ARB$ when executing $ARB_{LP} \cup PlanLibrary$.

The set $R_a$ is composed by the set of primitives listed in Table 4.3:

| Primitive | Description |
|---|---|
| `!goToTarget(C)` | Move $A$ to a *victim* `C` if agent $A$ have enough room to collect another *victim* |
| `!detCivilian(C)` | Sense the environment to detect if a *victim* `C` is reachable from its current position |
| `!pickupCivilian(C)` | Load a *victim* `C` if reachable |
| `!goToBase` | Move $A$ from its current position to the *base* as long as it is out from it and its capacity has run out |
| `!unload` | Leave the collected *victims* into the *base* |

**Table 4.3:** *Primitive actions $R_a$ to execute the rescue behaviour ARB*

The beliefs $RB$ of the agent related to learning are:

- The observed state-of-affairs in the environment `st/6`:

  `st(posAmb, civilRescued, distToAmbulance, distToBase, capacity, sector)`

- `cSt/6` and `lSt/6` for the current and last observed states, respectively: `cSt(X)`, `lSt(Y)`, where `X`,`Y` are states `st/7`

- Tracked action-states transitions `tAB/4`:

  `tAB(st1,action,st2,frequency)`

- Plan utilities (`planUtility/3`):

  `planUtility(plan,utility,frequency)`

- Q-values of plans in a given state `q/3`:

  `q(plan,st,utility)`

- Q-values of primitives in a given state `qr/4`:

  `qr(plan,action,st,utility)`

- Rewards `rewardAcSt/3`:

  `rewardAcSt(st,action,reward)`

Additionally, the weighting mechanism described in Equations 4.1 and 4.2 are in the case of the intended behaviour needing to verify the plan adequacy. To do so, a process of checking the achievement of a goal is performed by reviewing the existence of a relevant belief in the agent's belief base (denoted by $Belief?$)in order to modify the value of the utility (stored in `planUtility/3`) accordingly to the function ($\Delta(Bel?)$), defined in Formula 4.3.

$$\Delta(Bel) : Bel? \left\langle \begin{array}{l} \text{T:} \ +\nu \\ \text{F:} \ -\nu \end{array} \right. \tag{4.3}$$

Where $\Delta(Bel)$ is a relevant belief for a given plan, and $\nu$ is a value to modify the utility: $utility = utility \pm \nu$. Then, `planUtility/3` is updated to take into account the episodic nature of the task to keep the plan in a state considered adequate as its utility is affected by the achievement of an associated goal. In other words, `planUtility(plan,utility,frequency)` changes the value of `utility` accordingly with $\Delta(Bel)$. Experimentally, $\nu$ is set to 0.001.

On the other hand, the ideal set of plans (hand-coded) to implement *ARB* is depicted in the following snippet of Jason code.

```
@rescueCivil1
+!rescueCivil(C) : true
  <-    !goToTarget(C);   !detCivilian(C);   !collectCivil(C).


@collectCivil5
+!collectCivil(C): civilRescuedFnd(C) & belowCapacity
  <-    !pickupCivilian(C).


@collectCivil0
+!collectCivil(C): noMoreCapacity
  <- ...
    !goToBase.
```

Plan `rescueCivil(C)` labelled as `@rescueCivil1` performs the first stages of *ARB*: Going towards rescued victims is implemented by the primitive `!goToTarget(C)`; Check its loading capacity and Collect victims are undertaken by the primitive `!detCivilian(C)` and the sub-goal `!collectCivil(C)`. The latter is implemented by two plans labelled `@collectCivil5` and `@collectCivil0`. The former is required by its applicability context for the agent having enough capacity to collect victims. A victim is collected by means of the primitive `!pickupCivilian(C)`. On the other hand, if the agent does not have the capacity to collect more victims, then the second plan is triggered and the primitive `!goToBase` is called to accomplish the last part of the intended behaviour.

As in the previous Section, the process of IL is supported by means of Manipulative Abduction implemented as learning templates. For learning the *ARB* behaviour, the plan template used is *sequences of actions*, as depicted in Algorithm 10.

The set of experiments were carried out on a BDI MAS implementing IL with the configuration of agents shown in Table 4.4. The team formation strategy is accordingly with Algorithm 4 of Chapter 3.

| Reactive AGENTS | |
|---|---|
| Fires | 9 |
| Blockades | 9 |
| Civilians | 100 |
| BDI AGENTS | |
| Brigadiers | 16 |
| Police | 4 |
| Ambulances | {1...4} |
| FireBrigades | 4 |

**Table 4.4:** *The settings of reactive and BDI agents representing passive (environmental elements, and active (people) in the D-R BDI MAS.*

Consequently, as learning agents are just those representing ambulances they vary their numbers from 1 to 4 for each subdivision of the map, while the number of members of the rest of the agents remains in 4.

In Figure 4.17 the running D-R BDI MAS is presented with coloured traces. Coloured traces are just a reference about the ambulances' activity on the ground; each ambulance has its own colour depending on its designated subdivision (brown, orange, yellow and green).

In Figure 4.18 the summary of the D-R BDI MAS's global performances whilst running up to (a) $50,000$, and (b) $35,000$ actions. The plots belong to the number of rescue actions performed, with respect to the total of actions carried out in a simulation, running with four *ambulances* agents.

As can be seen, both runs are very similar in the performance, with small variations because of each runs have different dynamics provided by the actions of the agents in the MAS. The performance of two sets of 4 *ambulance* agents is shown in Figure 4.19. As in the global case, the X axis represents the number of rescue

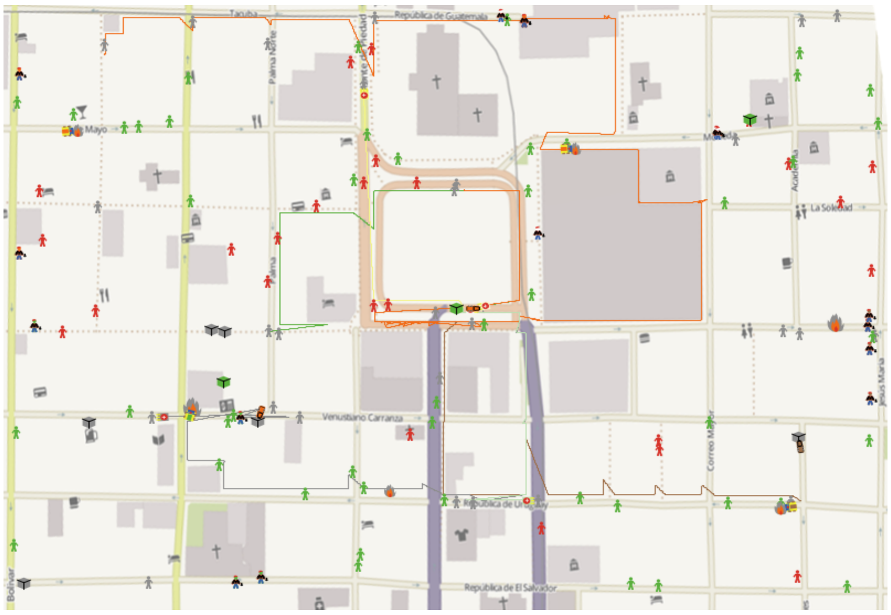
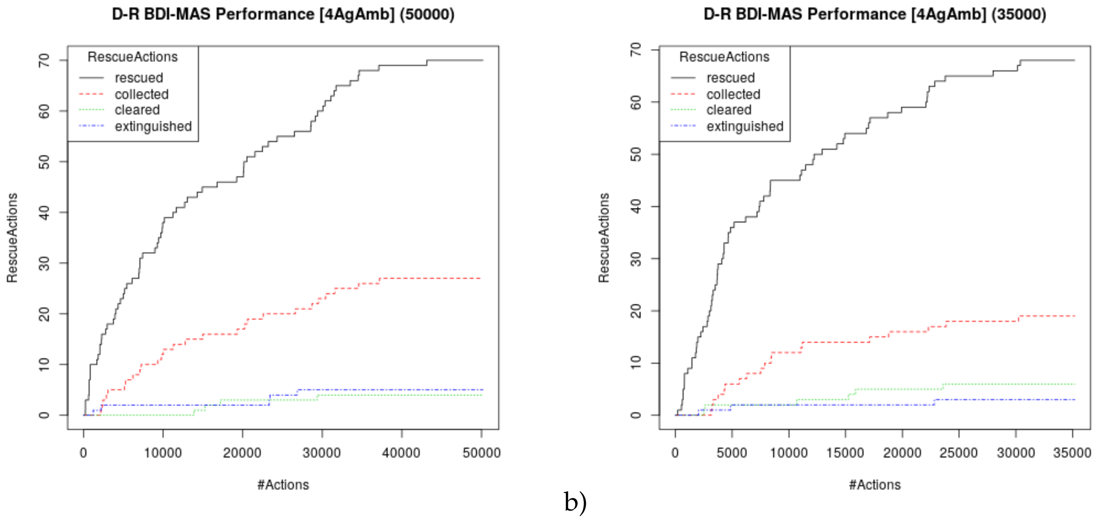


**Figure 4.17:** *The D-R BDI MAS running on the framework with coloured traces depending on their designated subdivision on the map.*

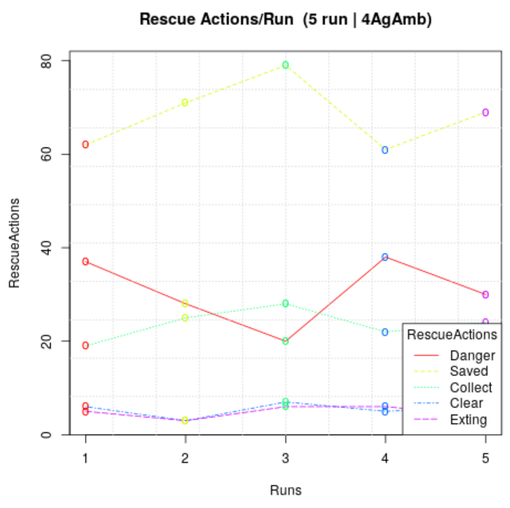


a) b)

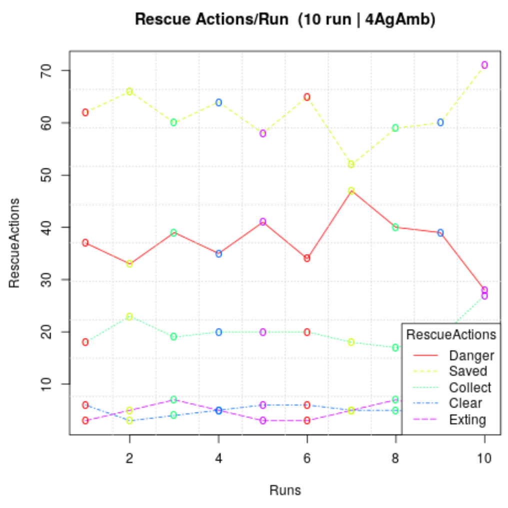
**Figure 4.18:** *Global performance of the D-R BDI MAS running with 4 ambulance agents. a) Running up to* 50, 000 *actions, and b) up to* 35, 000.

actions pursued for the D-R BDI MAS.

In the case of just one *ambulance*, this agent is in charge of respond to all the petitions of *brigadier* agents. In this regard, the number of collected *victims* depends on the number of notifications communicated from *brigadier* to *ambulances*, then, the more notifications, the more collected *victims*. Furthermore, there is a small variation in the number of collected *victims* between each run. This can be better observed in the 10 runs, where the performances are very similar across at least four runs.



a)



b)

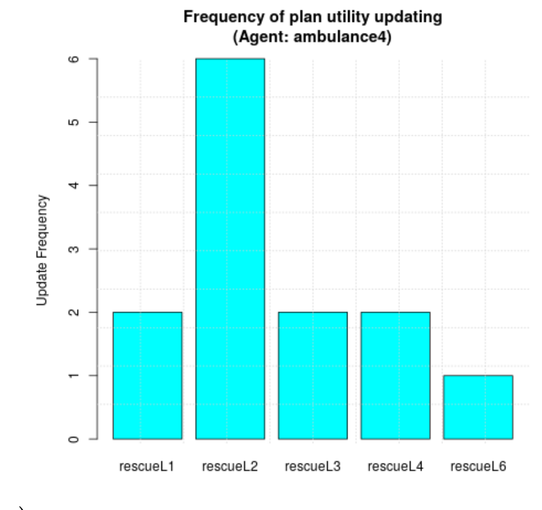
**Figure 4.19:** *Performance of the DR BDI MAS through (a)* 5 *runs* (50, 000 *actions), and (b)* 10 (35, 000 *actions), runs with* 4 *ambulance agents after acquiring the ARB behaviour, running up to* 35, 000 *actions.*

The set of learned plans ($ARB_{LP}$) for the case of one *ambulance* are shown in the following snippets.

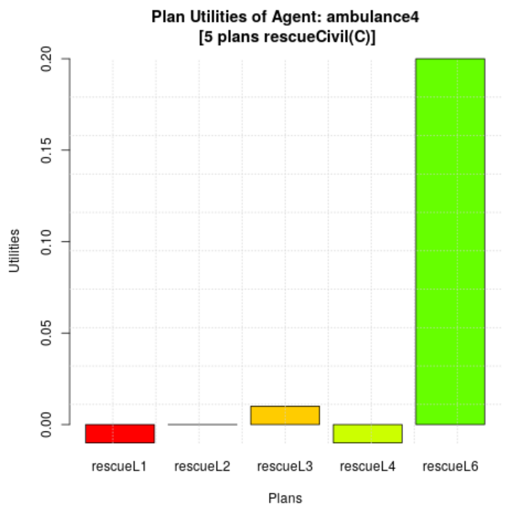
```
@rescueL1[atomic,learn(1),utility(-0.01)]

+!rescueCivil(C) : belowCapacity

<-  !goToTarget(C); !detCivilian(C); .


@rescueL2[atomic,learn(1),utility(0)]

+!rescueCivil(C) : belowCapacity

<-  !goToTarget(C); !detCivilian(C); !pickupCivilian(C); .


@rescueL3[atomic,learn(1),utility(0.01)]

+!rescueCivil(C) : ((posAmb(outBase) | posAmb(onTarget(_))) & noMoreCapacity)

<-  !goToBase; .


@rescueL4[atomic,learn(1),utility(-0.01)]

+!rescueCivil(C) : (noMoreCapacity & posAmb(onBase))

<-  !unload; .
```

```
@rescueL6[atomic,learn(4),utility(0.2)]
+!rescueCivil(C) : belowCapacity
<-  !goToTarget(C); !detCivilian(C); !pickupCivilian(C); !detCivilian(C); .
```

The annotation `utility/1` in the set of learned plans alongside with the context of each one, define an order in the applicability, namely (identified by plan labels): ⟨`rescueL6 (0.2)`, `rescueL3 (0.01)`, `rescueL2 (0)`, `rescueL1 (-0.01)`, `rescueL4 (-0.01)`⟩. The set of learned plans that fulfil with the leaning task (i.e. that define the *ARB* behaviour, i.e. the intended $ARB_{PL}$ is comprised by the plans with the following labels: ⟨`rescueL3, rescueL4, rescueL6`⟩, alternatively, `rescueL2` could be applied instead of `rescueL6`, but the applicability of the latter is higher. This is illustrated in Figure 4.20, where in (a) the frequency of updating, and in (b) the utility of each learned plan are depicted. Plan `rescueL2` is the most updated plan, while plan `rescueL6` (acquired later), is not updated so often, but its utility is the top ranked.



a)



b)

**Figure 4.20:** *Updating of utilities of learned plans, for an ambulance in DR BDI MAS whilst acquiring the ARB behaviour. (a) plan utility updating, and (b) plan utility when finishing.*

This situations is explained due to the plan generation, based on the ranking criterion $\lambda$ and the tracking beliefs $\theta$, but also because the process of adequacy drive the agent to plan revisions, that modify the content of a plan, resulting in plans that insert or change some actions in accordance with the current dynamic of the agent whilst operating in the environment, in this case, resulting in a plan with an extra action, but better ranked so far.

## 4.5 Summary

In fulfilment of the Research Objective 1 "Exploring the behaviour acquisition in the BDI agency ", this chapter reports the mechanism of IL at the level of plan acquisition on a full BDI declarative environment. Even more, in a fully logic-based agent oriented language using the framework composed by NetLogo and Jason platforms.

Learning in intentional systems has been addressed from different perspectives, including traditional Machine Learning algorithms that are called by the agents in certain circumstances, or closer proposals to the paradigm of intentional agents. The last includes IL, which has been developed in different approaches: learning referring to goals, plan selection, and plan acquisition. Here, IL at the level of plan acquisition direction is approached as part of the BDI reasoning cycle.

The plan acquisition method presented in this chapter, comprises three central activities: collecting actions, selecting and ranking, and determining the set of actions to be included in new plans. Then, the intention monitoring, the ranking criteria $\lambda$, and the plan building strategy $\psi$ are the core of IL. As such, the learning strategy proposed in this chapter constitutes a planning activity embedded in the BDI architecture.

Intention monitoring is based on BDI constructs, as meta-plans to detect two type of cases when learning is required: failure-handling, and lack in the procedural knowledge of agents. The former applies when a plan is no longer applicable, due to changes in the environment whilst the agent is operating. The latter regards to the acquisition of new skills from the set of primitive actions in the plan library of agents. The $\lambda$ criterion for selecting candidate actions to be part of a new plan, is based on evaluating the applicability and the utility of those actions. This is performed through a set of tracking beliefs ($\theta$), weighted alike in Reinforcement Learning techniques (Q-Learning), to register and evaluate the consequences of agent's actions. Finally, the $\psi$ strategy, for defining the structure of new plans, is guided by Manipulative Abduction implemented as three plan templates: a) one action; b) repetitions of actions; and c) sequences of actions.

Two cases of single-agent learning are successfully reached within a declarative BDI system (Jason), in a test-bed enviroment, namely: target-searching, and left-handed wall-following . The learning outcome are new plans encoding a target behaviour.

Then, IL was applied in a more challenging environment, the Disaster-Rescue domain, where the agents generate a set of plans to perform a rescue activity, in this case to describe the *ambulance* behaviour. Agents learnt how to collect victims (once saved by *brigadiers*) and take them to the base. This set of plans is generated by a group of primitive actions (arranged using the sequence of actions plan template), a set of weighted tracking beliefs. The performance of the agents is verified by reviewing the plan's adequacy for getting rescue people after the execution of the learnt plans, while the whole MAS performance is indicated by the rate of the rescue actions, namely, rescue, collect, clear, and extinguish, per the total number of performed actions. The global performance is not represented in the agents accordingly with the requested features of the MAS design, to avoid the definition of an explicit control that would drive the MAS to have a centralised control.

Finally, the contribution of this chapter is the first step towards collective learning in MAS: when considering the agent's communicative capability, it starts by learning at agent level, then opens the door to extend the IL to the MAS level in a pure BDI platform.

# Chapter 5

# Collective Intentional Learning

A changing environment represents a context where learning is a highly desirable feature to allow an agent to perform its design tasks. Despite BDI architecture not included learning in its original inception, this has been overcome by Intentional Learning (IL) presented in Chapters 2 and 4. IL is a learning framework introduced to exploit the BDI agent's reasoning features, allowing the agent to be capable of online learning. Under this approach learning is part of the agent's behaviour as its intentions are monitored, identifying when learning is needed for the purpose of performance improvement and acquiring new skills.

IL has been tested in single-agent scenarios, but in fulfilment of the Research Objective 3 "Designing a collective intentional learning mechanism", this chapter presents Collective Intentional Learning (CIL), an extension to IL that goes from the individual to the collective scope.

Developed in a full-fledged platform that strictly implements the BDI principles (Jason), CIL aims to progress the learning in MAS, and particularly in BDI MAS, by including collective learning capabilities in agents. Agents generate a set of plans based on the interaction with the environment and other agents in the system. As IL, the new set of plans account for improving agents behaviour and generate new skills. Plans are derived from primitive actions and a pairwise sequence of interactions with other agents.

This chapter is organised as follows. First a background is provided to understand the CIL method. Then the learning method is defined followed by the explanation of its functioning. The chapter concludes with the implementation of CIL and the results in the Disaster-Rescue domain.

## 5.1 Background

The environment, and the degree of difficulty of the tasks performed by a MAS, increases the complexity of their organisation and the relationships between the agents involved. When dealing with complexity, it is important to note the relevance of learning for improving behaviour. One of the main aspects in MAS organisation is exchange of messages between agents, and the agreement that can be obtained in order to tackle the task or purpose of the MAS.

At the agent-level, Machine Learning (ML) approaches have been mainly based on Reinforcement Learning (RL), Inductive Programming and mixed techniques, from the relational and propositional representations (Boolean formulae, decision list, decision trees, etc.) to artificial neural networks models and genetic algorithms [48, 82, 114, 126]. Many learning algorithms and approaches has been developed from a single-agent perspective and have been extended to meet the MAS perspectives, including Reinforcement Learning, Game Theory, Artificial Neural Networks and Genetic Algorithms as the more used [8, 85, 114, 126].

Within a group of interacting agents it is common that jointly action are performed, particularly with tasks that depends on cooperation (explicitly configured or generated by emergence) to be fulfilled. It is interesting to underline a relevant notion arising when agents work together, *joint action*, and other particularly important to the intentional approach of agency, *joint intention*.

It is noteworthy, from the point of view of the relation between the individual and collective stratum in the decision-making process, that some models explore the relationship with actions and intentions whilst jointly pursued by a set of agents taking notions from the HTN approach and BDI architecture, and the agent interactions under a Cooperative Games-based model with RL quantitative techniques [25, 53, 54, 78, 114, 125, 126]:

- Multi-agent Markov Decision Process (MMDP): this approach is purely collective and centralised. Each agent needs to be aware of the updating of state-action utilities within all other agents in the MAS.

- Multi-agent Semi-Markov Decision Process (MSMDP): it is decentralised but requires each agent to represent the whole space (states-actions) that may become very large, slow, and highly dependent on up-to-date information about the decisions of all agents in the MAS.

- Hybrid MDP-BDI: working in the single agent or MAS settings, it makes a correspondence between BDI plan and the policy of MDP, computing their performance. It exploits BDI plans to improve MDP tractability, and uses MDP to improve plan construction.

- Other Hybrid Approaches (CvI-JI): Explores the MAS MDP-BDI hybrid approach but not considering the relation between plan and policies, but instead the relationship with BDI and MDP temporally abstract actions. It considers intention as linear plans adopted by agents in the interest of reaching a given state. Joint-intentions are intentions defined in a pair of agents, joint-actions are the executing of actions to pursue such an intention. It establishes a 2-strata decision making process, to achieve a neither purely individual nor purely collective decision mechanism. Collective stratum is represented by an institutional agent acting as a coordinator of individual agents. Agents calculate the benefit to choose an option from its own set, or requests the collective stratum for a decision.

- Game theoretic approaches: each agent need to compute the utility of all combinations of actions executed by all other agents with a pay-off matrix. It search of Nash equilibria or a Pareto optimal solution. When general equilibrium exist, agents may adhere to a purely individual decision-making, not influenced by the collective perspective.

- Social learning: policy learning is performed in a pairwise (fixed or alternated) interaction between the agents in the MAS. It configures different levels of perception (local, collective or global), depending on the access to information of peers from other groups of agents comprising the MAS. It relies on a payoff matrix and a Q-Learning technique, to rate the benefit and reach policy acquisition.

In accordance with the CvI decision-making model, the collective stratum is represented by a coordinator agent that manages the individual agents, and consequently, the execution of individual actions. In this model, **collective options** are a set of options ending by a termination condition, i.e. a composition of shared individual options. *Purely individual tasks* are options that comprise the set of capabilities of a given type of agent (i.e. they belong to a given type of agent only). Similarly, *cooperative tasks* are options available in both the individual and the collective stratum. In other words, in the CvI model, the collective and cooperative options (multi-step options) are defined at design time as part of the definition of the single agent, that represents the entire group of agents or collective stratum.

On the other hand, in the JI model *team goals* are team decisions defined in terms of options available to the members of the team. Through the hybrid model, the collective options space is reduced by the use of joint-intentions, which acts as a filter, leaving out options that are not compatible with all the members of the team.

Social learning is a strategy to reach cooperation in MAS by means of pairwise agent interactions either local or collective. The purpose is to gather information from different sources by observing the individual and/or joint actions taken by peers belonging to the same or different teams. Joint actions are also defined at design time, for instance using a payoff matrix, where the benefit of taking those actions are specified [54].

The present work introduces CIL, an extension to IL that undertakes learning at the level of plan acquisition beyond the individual scope, to insert learning into a built up BDI MAS, based on the previously introduced concepts of social learning and the interplay between individual and collective decision-making strata.

## 5.2 Intentional Learning in a Collective Scope

As previously stated in Chapters 2, and 4, Intentional Learning (IL) has been addressed on the individual scope at the goal, plan applicability and plan acquisition levels. The IL framework presented in Chapter 4 introduces learning in single BDI agents. This learning method is profiled as a method of plan acquisition, enhancing agents with the capability to generate a new set of plans, driving its behaviour towards the acquisition of new skills. It was demonstrated using the framework of experimentation, a fully-fledged BDI platform enriched by a powerful Agent-Based Modelling tool to model its environment. CIL extends IL performing plan acquisition, based on agent messaging, and perceiving the environmental changes and the utility of new plans. Despite the main interaction being performed by agents belonging to the same type, learning agents can interact with both heterogeneous and homogeneous partners in the MAS.

In addition, Manipulative Abduction (i.e. the set $\Psi$ of plan templates to compound new plans), and plan revision stages, include the point of view of more than one agent for acquiring learning. The information that can be exchanged and the media to do so comprises:

- Direct agent messaging (centralised communication is avoided in this research as it prevents the system to be self-organised, according to the literature) or Indirect communication (like stigmergy or other means of communication).

- The sharing of different kinds of knowledge, ranging from environmental samples and statistical values (Q-values, rewards, policies and even episodes), beliefs, and plans.
- Reward definitions and a weighting mechanism to establish the utility of the performed actions, at individual and collective levels.

Therefore, in CIL the agent can learn from its own experience, yet also account for the interactions with other agents in a pairwise basis, using a direct communication based on the higher-level messaging of the framework. Depending on the type of the interacting partners, the shared experience could come either from a homogeneous or a heterogeneous agent. The learner applies a weighting criterion for the received knowledge, in order to use it appropriately.

Perceptions, plans and rewards are good candidates to be shared, so long as they are relevant and provide enough for learning [53,122]. Additionally, plan templates should determine the interchanged information and the utilities reported whilst the interchange is performed. In the case of beliefs, and values of the states, one important issue is the capability of the receiving agent appropriately to decode the information, which can be difficult with heterogeneous agents. An extreme case is the policy interchange, as it is only useful in the case of homogeneous agents.

## 5.3 Learning method in Collective Intentional Learning

To reach learning at the activities level in BDI agents, CIL extends the IL framework described in Chapter 4 from the single agent to a group scope. Like IL, it deals with some aspects of learning inherited from the individual case, adding those imposed by the collective one. As most of the learning components for the single agent case is already introduced, in the following brief comments on the aforementioned aspects of learning are presented, before exposing the CIL learning method.

- **The Exploration/Exploitation tradeoff**. This regards either keeping the learned plans in use, or continuing to add new ones to the *PlanLibrary*. Agents solve this problem by: a) An exploration parameter $e$, during the selection of the primitive actions, to form part of new plans. As in IL, the classical $e - greedy$

strategy is translated from actions selection to the plan acquisition process; b) A discount parameter $\delta$ to penalise the less beneficial plans (in the individual scope); c) an acceptability parameter $\phi$ to evaluate how convenient is to modify its own knowledge by accepting pieces of information from peers; and d) re-learning stages to comply with the global task, using a parameter to evaluate the acquisition of new behaviours.

- **Utility updating**. Plan selection is based on two criteria, plan applicability and plan utility. To update the plan utilities, the set $\Theta$ of tracking beliefs is used. Analogously to the set $\theta$ in IL, `qr/8` to store primitive action utility, `q/7` for plan utilities, and other beliefs to track the consequences of the execution of actions. Additionally, in the set $\Theta$, `qs/7` is added for estimating the utilities of shared knowledge.

In a more formal way, the CIL strategy comprises the following elements:

- The set of agent types $T$, defines a set of $M$ roles, or profiles, with a certain knowledge attached:
  $T = \{t_1, t_2, \ldots, t_M\}$.
- A Multi-Agent System composed by $N$ agents (either reactive of BDI based):
  $MAS = \{ag_1, ag_2, \ldots, ag_N\}$.
- Each agent belonging to a one of the different types:
  $MAS = \{ag_1^x, ag_2^y, \ldots, ag_N^M\}$ , $\exists\, ag_1^x, ag_2^y \mid (ag_1, ag_2 \in MAS,\ ag_1 \neq\ ag_2,\ x, y \in T,\ x = y \vee x \neq y)$
- The set $K$ of knowledge attached to each agent type:
  $K^t = \{plans, primitives, beliefs\} \mid t \in T$.

Each agent performs actions and manages its set of knowledge $K$ based on two scopes of decision:

- Local scope (act on its own)
  - Firing plans: sensing and acting
  - Triggering learning: plan acquisition and revision
  - Local updating of Q-values and rewards
- Collective scope

– Determine the acceptability of shared knowledge $s$:

$$\omega(s) = \begin{cases} 0 & \text{if} \quad s \notin K \\ 1 & \text{if} \quad \exists k \in K \,|\, k = s\ ,\ u_s > u_k \end{cases}$$

Where $u_x$, is the utility of a given piece of knowledge $x$.

Function $\omega$ leads an agent to accept knowledge from partners (when applicable). Otherwise, accept it only if better than its own knowledge. The knowledge $s$ to be shared can be:

- Perceptions and beliefs (states, transitions, Q-values, rewards, etc.).
- Plans and plans contexts.

An important step in CIL is to determine the actions to be part of a new plan. As in IL, in the collective scope, the agent applies the $\Lambda$ criterion, a Q-learning inspired weighting mechanism, to cope with this problem. The purpose of including the point of view of other agents in the MAS for plan acquisition is to pursue act jointly with a partner's capability (i.e. to reach collective decision). To do so, the weighting mechanism should take into account the interactions with other agents. Then, the collective updating of Q-values for an agent $ag \in MAS$ can be defined as in Equation (5.1):

$$Q_t^{ag}(s,a) = Q_{t-1}^{i}ag(s,a) + \alpha[R * \phi(s,a) - Q_{t-1}^{i}ag(s,a)] \tag{5.1}$$

Where $Q_t^i$, represents each of the values of the utility of an action $a$ when fired by agent $ag$ on a given context $s$. $\phi(s,a)$ is the average frequency of $a$ when taking by interacting partners, defined in equation (5.2):

$$\phi(a,s) = \frac{1}{n}\sum_{j}^{n} freq(ag_i, ag_j),\ ag_j \in \{MAS \setminus ag_i\} \tag{5.2}$$

Based on the aforementioned value updating, the learning strategy in CIL is described by Algorithm 11.

During its running time, each agent in the $MAS$ performs its intended behaviour until a failure is raised, due to either the lack of an applicable plan (lines 9-10) or because of a given plan does not seem to be beneficial to the intended task (lines (lines 14-15). In both cases a learning process is started, resulting in the update of

**Algorithm 11:** The general operation of CIL in a MAS composed by BDI agents.

```
   Input: T: Task to be performed; MAS: set of rescue agents; M: max number of actions
   Output: MAS with a set of new plans to perform T
   begin
1      initialize environment
2      foreach ag in MAS do
3          initialize ag
       end
       // Start the MAS activities
4      while $Ac < M$ do
5          teamFormation(MAS)
6          foreach ag in MAS do
7              ag.eventSel()
8              ag.planSel()
9              if Failure then
10                 ag.CIL(NewPlan)
               end
11             ag.intentionSel()
12             ag.executePlan()
13             ag.updateExperience()
14             if NoAdequacy then
15                 ag.CIL(PlanRev)
               end
16             update(Ac)
           end
       end
   end
```

the *PlanLibrary* of the given agent, accordingly with the learning method described earlier.

## 5.4 Implementing CIL in Jason

The implementation of CIL presented here is related to direct agent messaging taking advantage from the Jason higher level communication facilities (i.e. a speech-act based inter-agent communication). It works jointly with belief annotations, to know about information *sources* (other agents). Jason provides a `.send` internal action, which offer the following parameters [17]: a receiver (a name, or a list of names, of the agents with whom communication is established); the illocutionary force of the message (`tell`, `achieve`, `askOne`, `askAll`, or `askHow`), the message (a literal representing the content of the message). This internal action can also receive an optional parameter: an answer (the answer for performatives `askX`) and a timeout defined in milliseconds, for awaiting an expected reply to a question.

A message that performs of a `tell`-type produces a belief addition event in the receiving gent, or list of receiving agents. Through this, the agent can exchange `beliefs` (i.e. literals representing different pieces of information as complex as necessary, to exchange the required knowledge.

Furthermore, homogeneous agents can use the `askHow` performative in order to exchange an entire plan. This allow to establish a minimal protocol of communication and knowledge sharing to be exploited in the collective plan acquisition. For instance, to share plans between agents when lacking this knowledge:

1. $ag_i$ start learning
2. $ag_i$ request knowledge exchange to other agents
3. When some $ag_j$ answers the request, $ag_i$ and $ag_j$ establish an exchange of knowledge
4. Knowledge exchange cases:
   (a) Plan. $ag_j$ have any plan to share
   (b) Q-values. $ag_j$ have any of the relevant tracking beliefs in $\Theta$ to learn (`q, qr, qs`)
5. $ag_i$ determines if the exchanged knowledge is *acceptable*
6. $ag_i$ generates a new plan accordingly with its learning templates and the *accepted* knowledge

In Figure 5.1 the interaction diagram of this process is depicted.

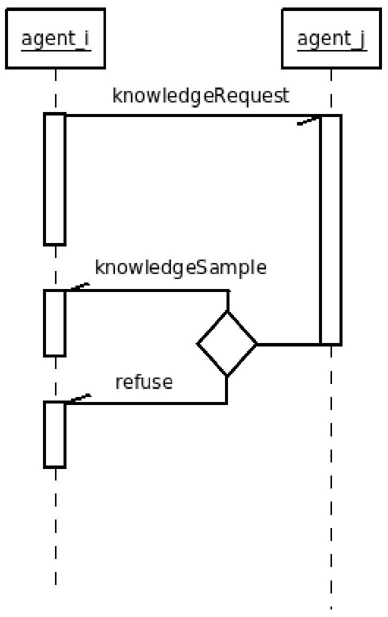


**Figure 5.1:** *Pairwise learning interaction in CIL.*

## 5.5 Experiments and Results

In previous chapter, learning under the intentional framework has been tested in single-agent scenarios. This section reports on a BDI MAS with learning in the collective scope, the results of the extension to IL that take learning from the single-agent level to a set of learning agents (namely Collective Intentional Learning), are shown.

### 5.5.1 CIL in the DR Domain

Learning in the collective scope within the D-R domain is built atop of the implementation of the IL in this domain. Similarly to the MAS in IL described in the previous section, the MAS with CIL is comprised of BDI agents and reactive agents. The set of agent types are the rescue staff, people in danger and the environmental agents, where the rescue staff split in four types. More formally, the MAS is composed by $N$ agents belonging to $T$, where $T = \{rescueStaff_{BDI}, civilians, envElements\}$. In its turn, $rescueStaff_{BDI} = \{brigadier, ambulance, fireBrigade, police\}$ represented as Jason agents with their counterpart as *turtles* in the NetLogo environment, while *civilians* and *envElements* are defined just as *turtles*.

Agents in the MAS have the same set of skills defined in Chapter 4, Subsection 4.4.5.1 encoded as plans in their *PlanLibrary* accordingly with the main defined scenarios in the DR domain:

- *doSearch* for *brigadier* agents.
- *goToObjective* for *ambulances*, $fireBrigades$ and *police* agents.
- *rescueAction* for all agents.

Analogous to the single agent setting, the purpose of the collective learning is to achieve the **rescue behaviour** (*ARB*) described in brief as: 1) Go towards identified victims; 2) Check its loading capacity; 3) Collect victims, and 4) Carry victims towards the base of the city.

Atop that, the **learning task** in CIL, applied to the DR domain, is defined for *ambulance* agents $A$ as follows
:

**Learning Task.** Given a set of rescue actions (primitives) $R_a \subset PlanLibrary$ available to $A$, determine *collectively* the relevant primitives $r_a \in R_a$ to generate a set of plans $ARB_{LP}$ that allows $A$ to perform $ARB$ when executing $ARB_{LP} \cup PlanLibrary$.

The set $R_a$ of primitives needed to pursue the $ARB_{LP}$ behaviour were previously listed in Table 4.3 of Chapter 4, namely:

- `!goToTarget(C)`. Move $A$ to a *victim* `C` if agent $A$ have enough room to collect another *victim*.
- `!detCivilian(C)`. Sense the environment to detect if a *victim* `C` is reachable from its current position.
- `!pickupCivilian(C)`. Load a *victim* `C` if reachable.
- `!goToBase`. Move $A$ Form its current position to the *base* as long as it is out from it and its capacity has run out.
- `!unload`. Leave the collected *civilians* into the *base*.

CIL uses the same direct inter-agent communication speech-act based upon messaging of Jason. Whenever `tell` performative message is sent, a belief is added in the receiver (or list of receiving agents).

The communication between agents is performed by a *Rescue Communication Protocol*, as exemplified in Figure 5.2, which depicts the interaction diagram between *brigadier* and *ambulance* agents.

In the interest of knowledge exchange, homogeneous agents use different performatives: `tell`, and the `askHow` performative to exchange full plans. The communication protocol and knowledge exchange for the sake of collective acquisition of plans is defined as follows.

When an *ambulance* agent $A$ is lacking the rescue plan `!rescueCivil(C)`, or it requires to be reviewed, it needs to interchange knowledge with other in a pairwise way:

1. $A_i$ start learning
2. $A_i$ request knowledge interchange to other *ambulance* agents
3. When some $A_j$ answer the request, $A_i$ and $A_j$ establish an knowledge exchange
4. Knowledge exchange cases:

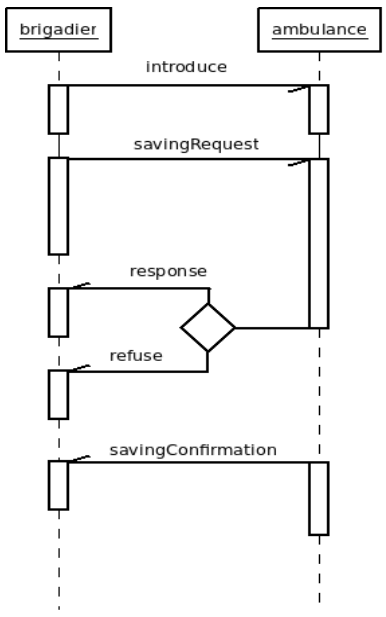


**Figure 5.2:** *Rescue Communication Protocol in CIL between brigadiers and ambulances.*

(a) Q-values. $A_j$ have any of the tracking beliefs in $\Theta$: `rewardAcSt, qr, tAB...` to share

(b) Plan. $A_j$ have the `!rescueCivil(C)` plan

5. $A_i$ determines if the interchanged knowledge is *acceptable*

6. $A_i$ generate a new plan accordingly with its plan templates and the accepted knowledge

This interaction is depicted in a more graphical way in Figure 5.3

The set of experiments where performed in a HP Pavillion Laptop AMD Ryzen 5, running Ubuntu Version 4.15, with 10GB in RAM. The sets of experiments comprises: a) plan exchange; b) `rewardAcSt`; `qr` exchange; and d) `tAB` exchange. On each set the number of ambulances vary from 2 to 6. The template used in the experiments is sequences of actions. It is important to note, that for fulfilling the learning task 5.5.1, repetitions is not the most useful due to the task needs the application of different sequences of actions rather than the repetition of one of them, in such a case a high number of plan revisions is expected. Similarly, in the case of the template of one action, the learning is delayed due to the amount of plans generated by the agent in response to the event `rescueCivil(C)`, each of them executing just a primitive action, making it difficult to complete the list of actions to perform *ARB*, and demanding a big number of reviews.

Additionally, when the number of *ambulances* goes beyond 4, some agents remain waiting for a rescue request from *brigadier* agents. This is explained because the Rescue Communication Protocol that define the

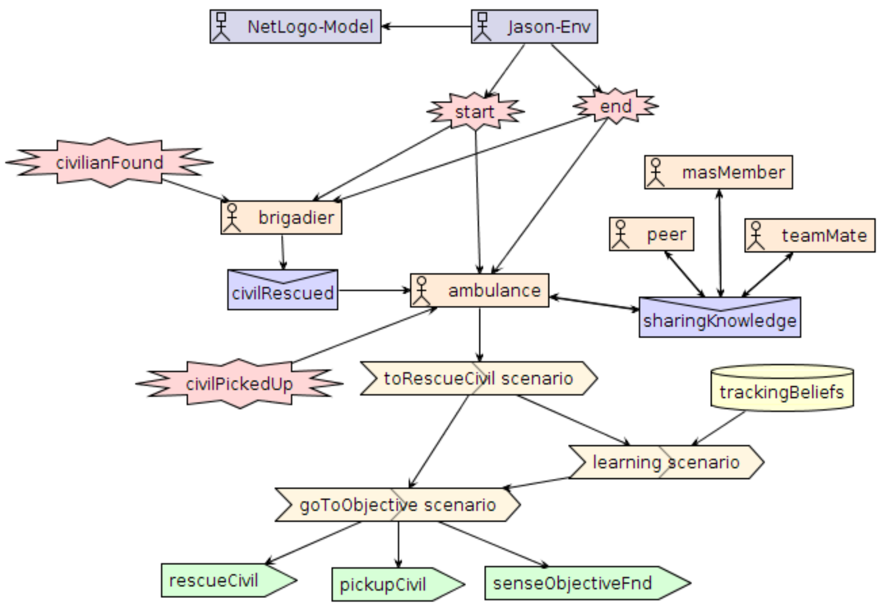


**Figure 5.3:** *Learning related pairwise interaction among brigadiers, ambulances, peers, team mates and other members of the MAS.*

interaction agreements between agents, operates in a first-input first-output mode. Consequently the first *ambulance* that agreed the interchange, keeps serving the incoming requests.

The configuration of agents used in these experiments are shown in Table 5.1 below:

| Reactive AGENTS | |
|---|---|
| Fires | 10 |
| Blockades | 10 |
| Civilians | 100 |
| BDI AGENTS | |
| Brigadiers | 16 |
| Police | 4 |
| Ambulances | {2, 4, 6, 8} |
| FireBrigades | 4 |

**Table 5.1:** *The settings of reactive and BDI agents representing passive (environmental elements, and active (people) in the D-R BDI MAS.*

The results are presented according to the following order:

- Summaries of the D-R BDI MAS performance per run
- For brevity, the results regarding to the 10th run are presented:
  - The plot of the DR BDI MAS' performance
  - The bar plots of the amount of collected civilians per ambulance

- The plan updating frequencies
- Some learned plan examples, and
- The view of the system before finishing taking the cases of 2 and 8 *ambulances*

### 5.5.2 Exchange of Plans

Plan exchange is based on the sharing of entire pieces of procedural knowledge in order to fill a gap in the *PlanLibrary* of a given agent through requesting its peers, consequently, the exchange of knowledge is only possible on homogeneous agents.

In Figure 5.4 the plots of summarized performances of 10 runs with 2, 4, 6 and 8 *ambulances* up to 450, 000 actions are depicted. In Table 5.2, the values of all the summarised runs are presented besides its mean, the maximum and minimum values, and the median.

| | **Danger** | **Saved** | **Collect** | **Clear** | **Extinguised** | **Danger** | **Saved** | **Collect** | **Clear** | **Extinguised** |
|---|---|---|---|---|---|---|---|---|---|---|
| | 2 *ambulance* agents | | | | | 4 *ambulance* agents | | | | |
| m | 25.00 | 55.00 | 20.00 | 3.00 | 4.00 | 27.00 | 54.00 | 18.00 | 4.00 | 3.0 |
| n | 36.50 | 62.50 | 25.50 | 5.00 | 5.00 | 35.50 | 63.50 | 21.00 | 5.00 | 4.5 |
| $\mu$ | 35.40 | 63.60 | 26.20 | 4.90 | 5.10 | 35.70 | 63.30 | 22.00 | 4.90 | 4.6 |
| M | 44.00 | 74.00 | 33.00 | 6.00 | 6.00 | 45.00 | 72.00 | 28.00 | 7.00 | 7.0 |
| | 6 *ambulance* agents | | | | | 8 *ambulance* agents | | | | |
| m | 32.0 | 57.0 | 17.00 | 3.00 | 2.0 | 23.00 | 59.00 | 20.00 | 3.0 | 4.0 |
| n | 36.0 | 63.0 | 21.50 | 5.00 | 4.5 | 34.00 | 65.00 | 22.00 | 5.0 | 5.0 |
| $\mu$ | 36.4 | 62.6 | 21.00 | 5.00 | 4.5 | 33.00 | 66.00 | 22.70 | 4.5 | 5.1 |
| M | 42.0 | 67.0 | 24.00 | 6.00 | 7.0 | 40.00 | 76.00 | 28.00 | 5.0 | 6.0 |
| **Avg** | 35.12 | 63.88 | 22.98 | | | | | | | |

**Table 5.2:** *The minimum (m), the median (n), the mean $\mu$, and the maximum (M) of the summarized values for the 10 runs of the D-R BDI MAS, varying the number of ambulance agents (2-8). Avg is the means of means for Danger, Saved and Collected.*

From these summaries can be seen that *brigadier* agents detect victims (identified with the column *Saved*) between 55 to 76 of the total amount of victims (100), with an average of 63.88 for all the 10 runs. The averages for *Danger* and *Collected* for all the runs are 35.12 and 22.98, respectively.

People in danger decreases inverse proportionally (with a correlation of $-1$). As expected, there is a relation between found victims (Saved) and those who have been drove to the base (Collected) to keep in safe, as presented in Table 5.3

In the case of the runs with 2, 4, and 8 *ambulance* agents the correlation is high. Though *ARB* is still performed, the runs with 6 *ambulances* exhibit a lower correlation, showing less variations during the 10 runs.

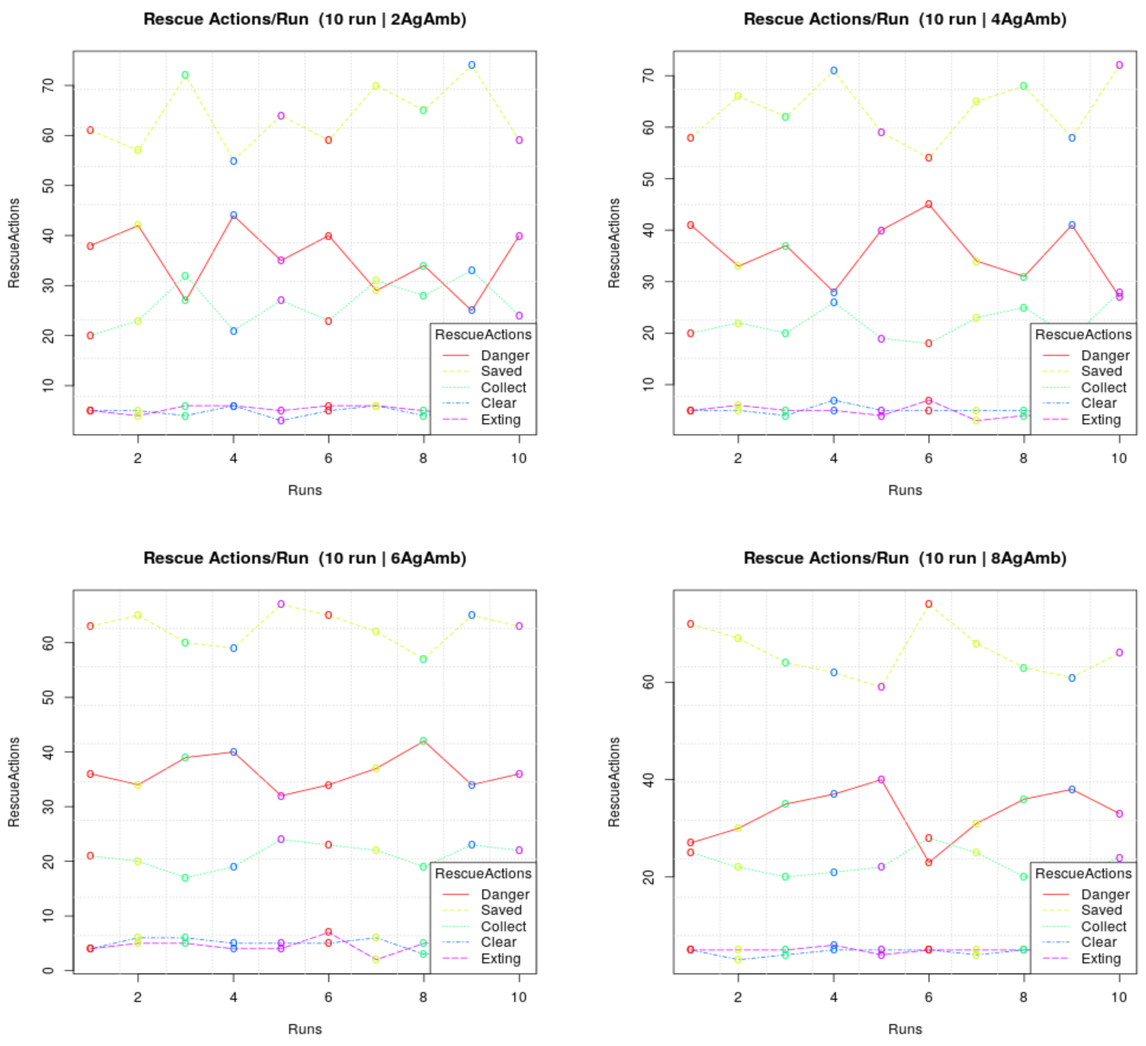


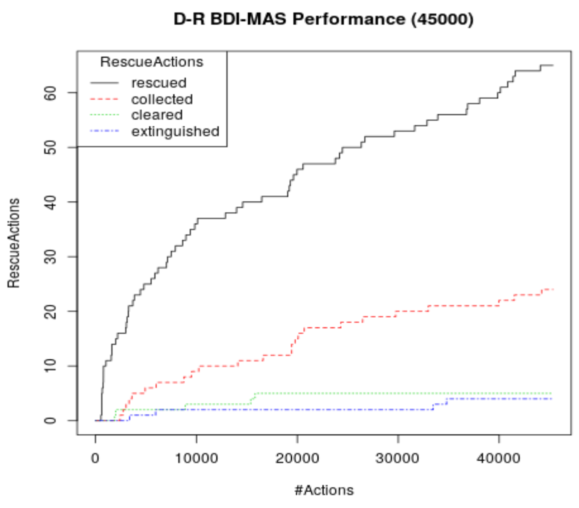
**Figure 5.4:** *Summary of the performance of each of the 10 runs of the BDI MAS-DR with Collective Intentional Learning, varying the number of ambulances form 2 to 8.*

| Ambulances | | | |
|---|---|---|---|
| 2 | 4 | 6 | 8 |
| 0.937 | 0.960 | 0.785 | 0.836 |

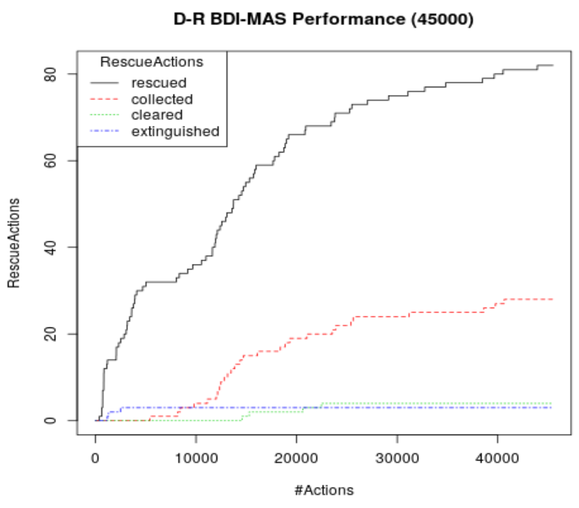
**Table 5.3:** *The Correlation (C) between Danger and Collected values for the 10 runs of the D-R BDI MAS, varying the number of ambulance agents (2-8)*

To give an overview of how the entire D-R BID MAS is operating (underlining how *ambulance* agents are working), in Figure 5.5, the performances from the 10th run of the D-R BDI MAS varying the number of *ambulance* agents from 2 to 8 are shown.

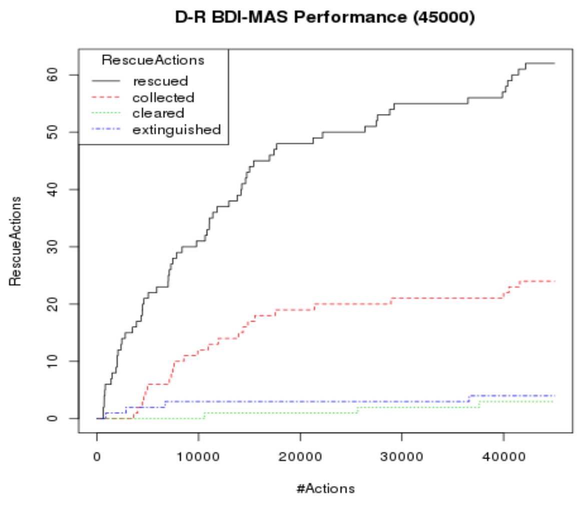
To observe the performance of the *ambulances* in the experiments in a more particular way, in Figure 5.6 the number of collected civilians per ambulance during each of the runs is presented.



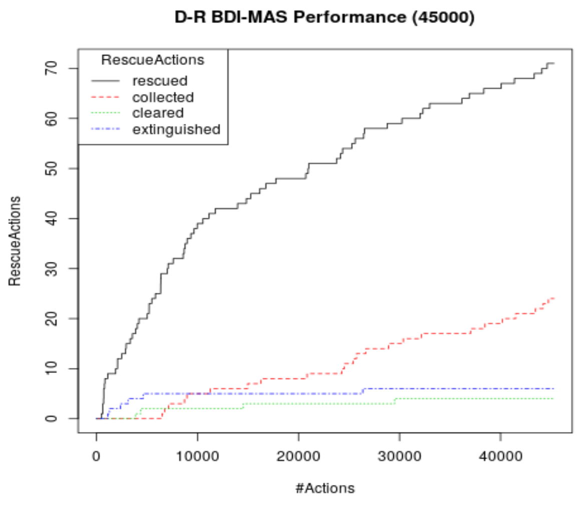
a)



b)



d)



d)

**Figure 5.5:** *The performances of the BDI MAS-DR in the 10th run varying the number of ambulance agents: (a) 2, (b) 4, (c) 6, and (d) 8.*

The number of collected civilians per *ambulance* shows that some of them do not participate with *brigadier* agents. As explained before, it is due to the current implementation the agreement is established in a first-input/first-output way. Consequently, it means that the cooperation agreement once established with a teammate agent during the interactions, leaves out the participation of those agents not forming part of the agreement. On this way, with 2 and 4 *ambulance* agents, all of them participate in the rescue actions, whilst in the case of having 6 and 8 *ambulances*, some of them do not collect civilians (as the cooperation agreement is not established during the run). In addition, in all the cases, there is an *ambulance* agent that collects more civilians than its peers. It is explained because of the difficulties of the ground imposes uneven conditions for traversing through it; besides, the random allocation of the civilians in danger make it possible that certain

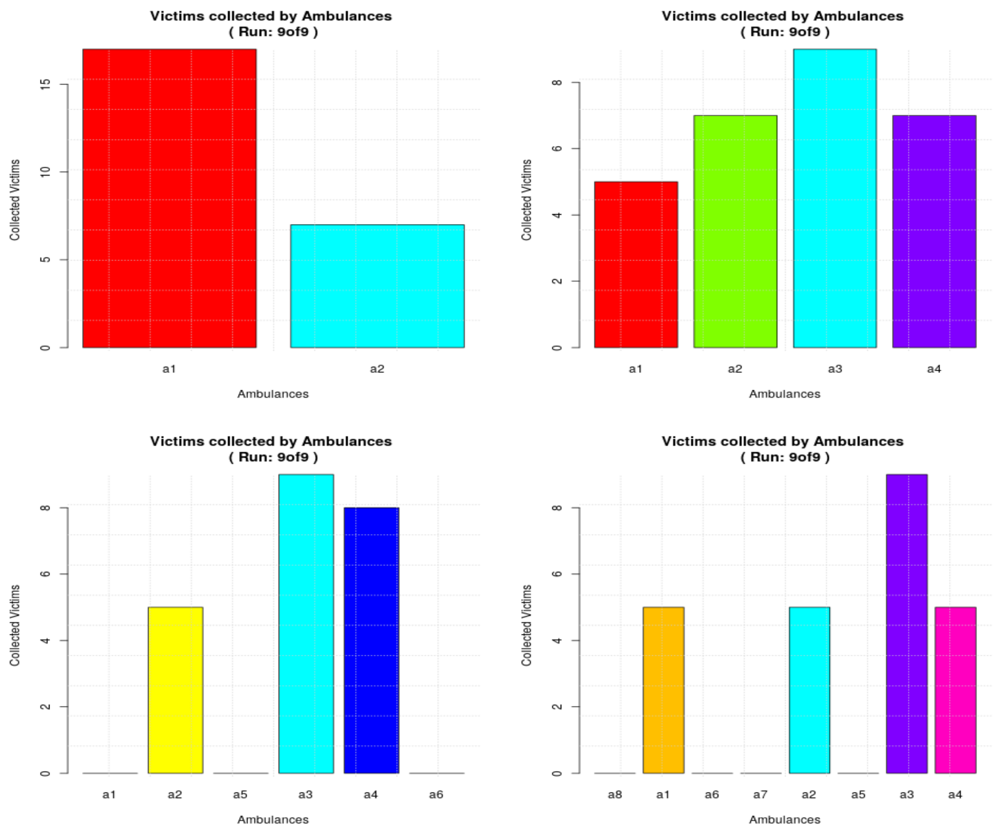


**Figure 5.6:** *The number of collection actions per ambulance within the BDI MAS-DR within each run with different numbers of ambulances: (a) 2, (b) 4, (c) 6, and (d) 8.*

*ambulances* look after more victims in such a place (populated with more victims than other parts of the city). Therefore, this explains the similarity between the performances of the different variations of *ambulances* presented before.

Furthermore, in Table 5.7, 5.8, 5.9, 5.10 the updating frequency of plans belonging to each *ambulance* agent in the 10th run for all the variation of agents is presented.

Figure 5.7 shows both *ambulance* agents updating one plan more times than the others, in the case of *ambulance* 1, there are three plans, being a plan labelled `@pResca1L1` the most frequently updated. In the case of *ambulance* 2, there are two plans, with a plan labelled `@pResca2L1` updated in 7 times.

Figure 5.8 shows *ambulance* 1 with seven plans split in almost two halves (according to their updating frequencies), which are updated not so often (reaching a maximum of 4 times). The set of plans includes those coming from other agents, such as `pResca3L1` and `@pResca4L1`, where 3 indicates that *ambulance* 3 is the original owner of this plan. Similarly, 4 indicates that this plan is coming from *ambulance* 4), being the most

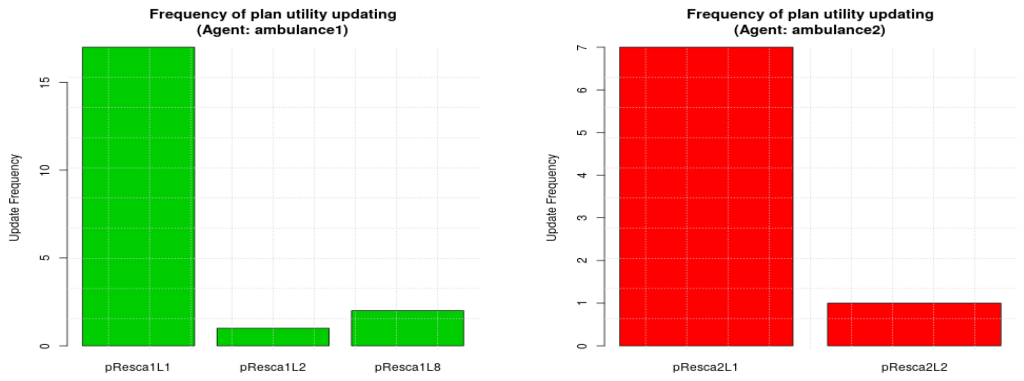


**Figure 5.7:** *The updating frequency of plan per ambulance within the BDI MAS-DR, from the 10th run with 2 ambulances.*

frequently updated (four times). This is an example of how shared plans are modified within the receiver agent as any other plan in its *PlanLibrary*.

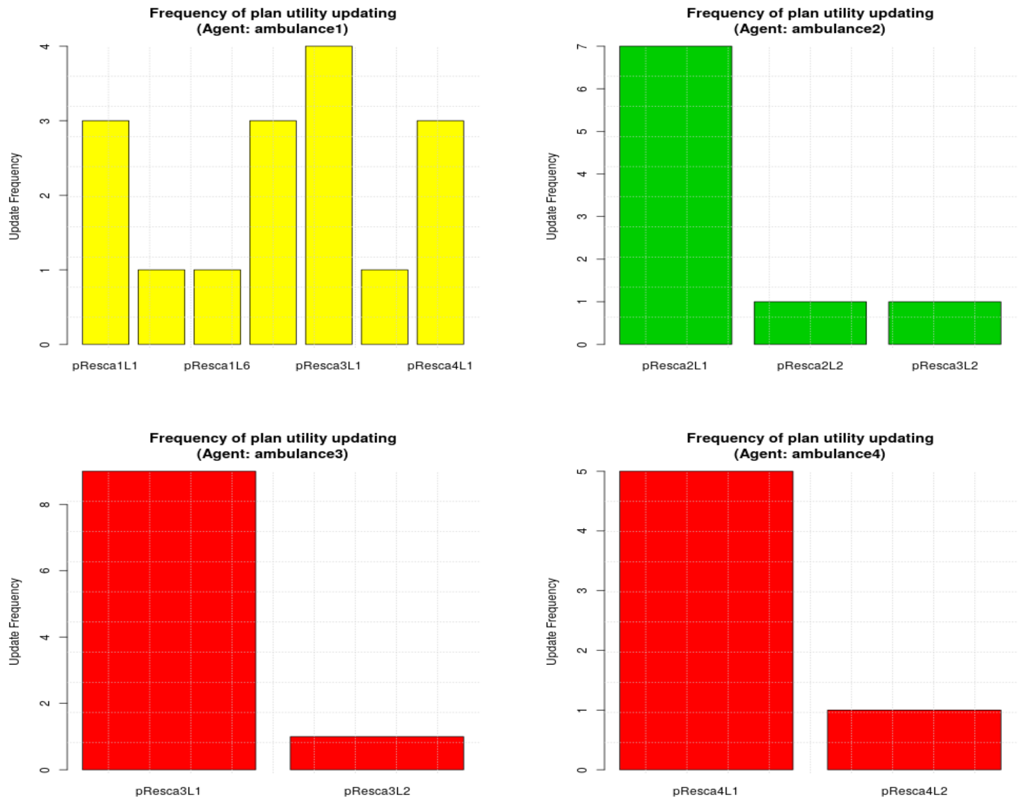


**Figure 5.8:** *The updating frequency of plan per ambulance within the BDI MAS-DR, from the 10th run with 4 ambulances.*

In Figure 5.9, there are also plans coming from peers (in *ambulance* 2), but those most frequently updated are generated in the agents themselves. In the case of *ambulance* 1, the most frequently update plan reaches

20 updates, contrarily to the less updated plans reaching only one. *ambulances* 3 and 4, show a similar pattern of updating, due to the similarities of their learned plans.

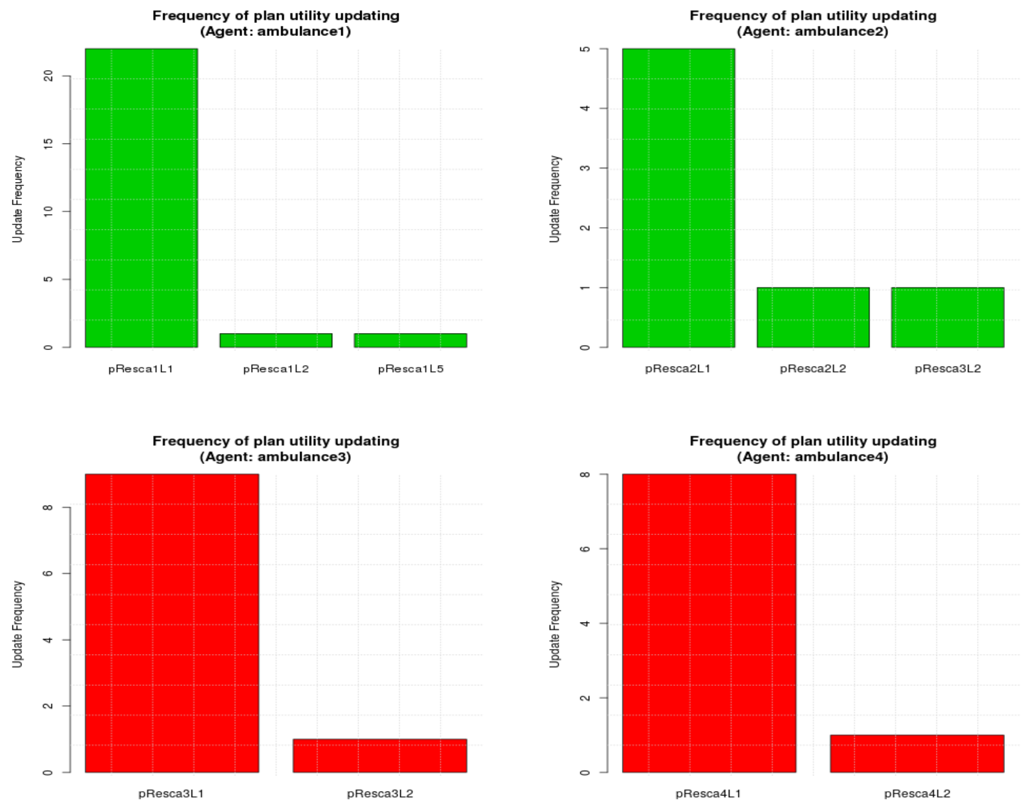


**Figure 5.9:** *The updating frequency of plan per ambulance within the BDI MAS-DR, from the 10th run with 6 ambulances.*

Lastly, in Figure 5.10 the plan updating of *ambulances* 1 and 2 reveals that the changes are focused on two plans, generated by themselves. The updates in both cases reaches a maximum of 5 times and a minimum of 1. Whilst, *ambulances* 4 and 6 show sets of three updated plans. In the case of *ambulance* 4, some shared plans are showed but the most updated one belongs originally to itself, similarly, *ambulance* 4 update three plans but all of them where originated in the same agent.

To illustrate how the learning is achieving the set of plans to perform *ARB*, some examples of the generated plans, from *ambulances* agents, are listed in the following.

```
// ambulance 1
@pResca1L1[atomic,learn(3),utility(0.01)]
+!rescueCivil(C) : belowCapacity
```

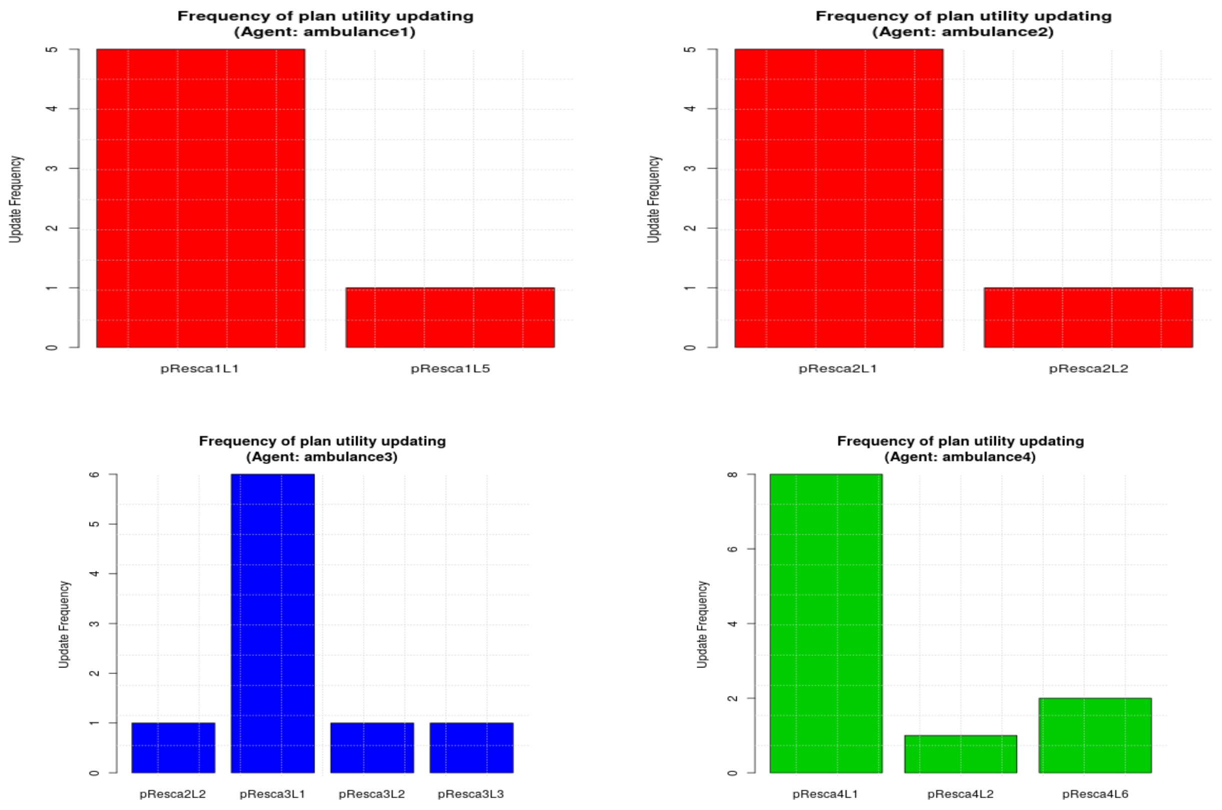


**Figure 5.10:** *The updating frequency of plan per ambulance within the BDI MAS-DR, from the 10th run with 8 ambulances.*

```
    <-  !goToTarget(C); !detCivilian(C); !goToTarget(C);
        !goToTarget(C); !pickupCivilian(C).
@pResca1L2[atomic,learn(2),utility(-1)]
+!rescueCivil(C) : (noMoreCapacity & posAmb(onBase))
    <- !goToTarget(C).
@pResca1L5[atomic,learn(3),utility(-1)]
+!rescueCivil(C) : belowCapacity
    <- !goToTarget(C); !goToTarget(C); !goToBase.
@pResca1L6[atomic,learn(2),utility(-1)]
+!rescueCivil(C) : ((posAmb(outBase) | posAmb(onTarget(_))) & noMoreCapacity)
    <- !goToBase; !unload; !goToTarget(C).
@pResca1L7[atomic,learn(2),utility(-1)]
+!rescueCivil(C) : (noMoreCapacity & posAmb(onBase))
```

```
    <- !unload; !goToTarget(C); !detCivilian(C).
@pResca1L20[atomic,learn(1),utility(0.01)]
+!rescueCivil(C) : ((posAmb(outBase) | posAmb(onTarget(_))) & noMoreCapacity)
    <- !goToBase; !unload; !pickupCivilian(C).
```

The learned plans are applied according to: a) their contexts (when matching the current beliefs and perceptions of the agent), and b) their annotation `utility(U)`, where positive values of `U` are applied in the first place. Consequently, in the case of *ambulance* 1 from the snippet, the plans `@pResca1L1` and `@pResca1L20` have more possibilities to be selected if their context match the current state-of-affairs of the agent, otherwise, the rest of the plans are selected accordingly to the agent's beliefs and their position in the agent file (Jason selects those plan listed in the first case). This listing presents a variety of plans, with different body sizes and distinct contexts of applicability, which utilities are differently valued also. The plans listed show how the plan acquisition and revision is performed shaping the agent's behaviour accordingly with its interaction with the environment and other agents. Four cases can be observed:

- Plans with positive values of `U` contribute the most to the fulfilment of *ARB* by having bodies that executed such behaviour.
- Despite having redundant instructions like the sequence $\langle$ `goToTarget(C)`,`goToTarget(C)` $\rangle$ in plan `@pResca1L1`, positively-valued plans still contribute to perform *ARB*.
- Even some negatively-valued plans can contribute in performing *ARB*, like `@pResca1L6` that goes to the base if the agent is out of it and the `noMoreCapacity` condition is fulfilled, allowing the right application of the primitive `unload`, making it possible for the agent to continue rescuing reported victims.
- On the other hand, some negatively-valued plans need to be refined due to they are not contributing to perform *ARB*. For example, the plan `@pResca1L2`, which context allows the execution of the primitive `!goToTarget(C)`, despite being applicable, this execution does not help in unloading the *ambulance* agent as should be expected because of its context's condition `noMoreCapacity`.

Finally, two snapshots of the 10th run are depicted in Figure 5.11 from 2 and 8 *ambulances*. Coloured traces show the trajectories of the agents once they touched the base to unload the rescued civilians, i.e. once an agent has achieved a set of plans to perform the *ARB*. The coloured traces show how the agents cover the ground in context within the environment alongside with the rest of the agents and environmental elements. In both cases, the ambulances cover the four sectors of the city, collecting reported victims.

### 5.5.3 Exchange of Rewards

This experiment reports the CIL with agents exchanging `rewardAcSt` values, related to the benefit of an action in a given state. This value is used to compute the `qr` values of the primitives to be part of new plans. Similarly to the plan exchange test, the results report the performances, the amount of victims collected, the plan update frequencies, the generated plans, and a view of the system as a whole while running. *ambulance* agents are varied from 2 to 4.

In Figure 5.12 the plots of summarized performances of 10 runs with 2 and 4 *ambulances* up to 450, 000 actions are depicted.

According to the summaries, there is not too much variation between each of the runs, nor between both sets of runs with 2 and 4 *ambulance* agents. In Table 5.4 the statistical values of minimum, maximum, the median, of the summarized performances; and the correlation between *Rescued* and *Collected* lines are shown.

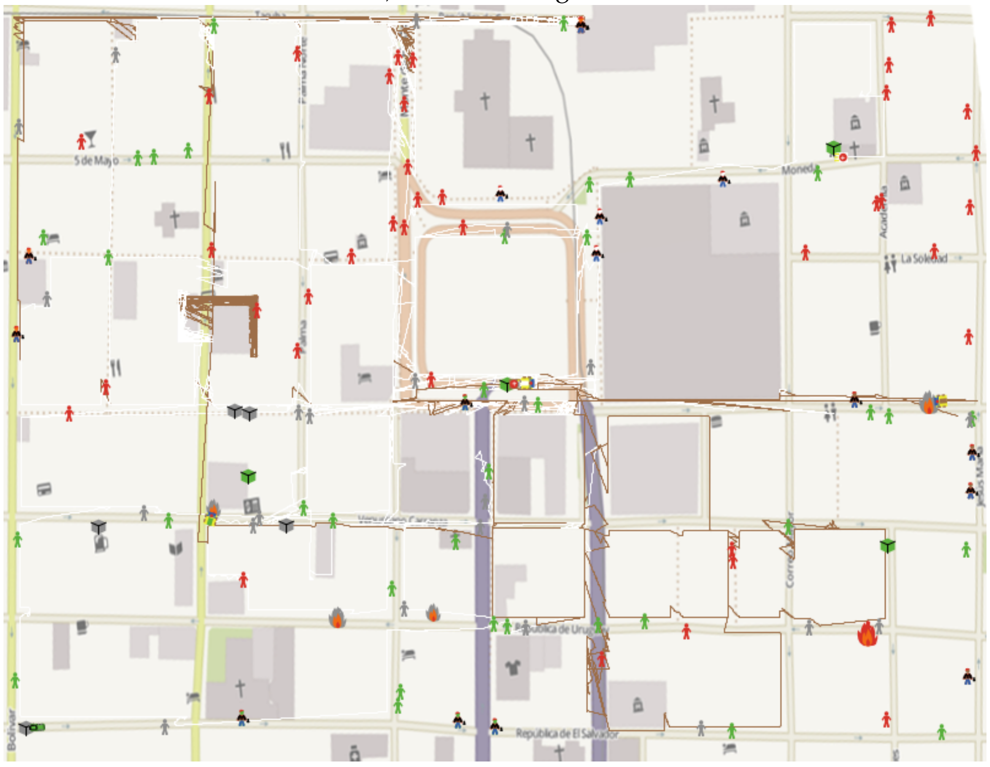
| | **Danger** | **Saved** | **Collect** | **Clear** | **Extinguised** |
|---|---|---|---|---|---|
| | 2 *ambulance* agents | | | | |
| **m** | 23.00 | 58.00 | 24.00 | 3.00 | 3.0 |
| n | 35.00 | 64.00 | 27.50 | 5.00 | 5.0 |
| $\mu$ | 33.90 | 65.10 | 27.60 | 5.00 | 4.8 |
| **M** | 41.00 | 76.00 | 34.00 | 7.00 | 7.0 |
| **C** | **0.9627** | | | | |
| | 4 *ambulance* agents | | | | |
| **m** | 31.00 | 57.00 | 21.00 | 3.0 | 4.0 |
| n | 36.00 | 63.00 | 23.50 | 5.0 | 5.5 |
| $\mu$ | 35.50 | 63.50 | 23.20 | 4.6 | 5.5 |
| **M** | 42.00 | 68.00 | 25.00 | 6.0 | 7.0 |
| **C** | **0.9286** | | | | |

**Table 5.4:** *The minimum (m), the median (n), the mean $\mu$, the maximum (M) of the summarized values for the 10 runs of the D-R BDI MAS, varying the number of ambulance agents, and the Correlation (C) between Danger and Collected values*

Both set of runs, with 2 and 4 *ambulances* exhibit a high correlation (0.96 and 0.92, respectively) regarding the *Danger* and *Collected* number of victims.

An overview of how the entire D-R BDI MAS works is presented in Figure 5.13, where the performances from the 10th run of the D-R BDI MAS varying the number of *ambulance* agents from 2 to 4 are depicted.

a) 2 *ambulance* agents



b) 8 *ambulance* agents

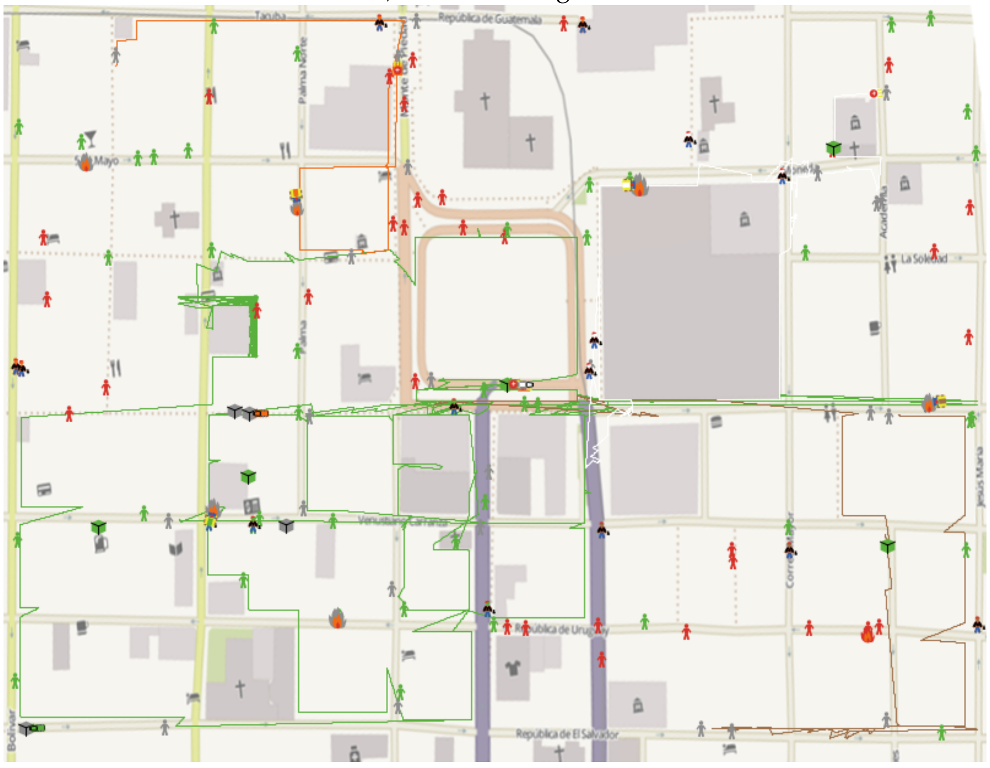


**Figure 5.11:** *The final state of the BDI MAS-DR: (a) the run with 2 ambulances, and the run with 8 ambulances.*

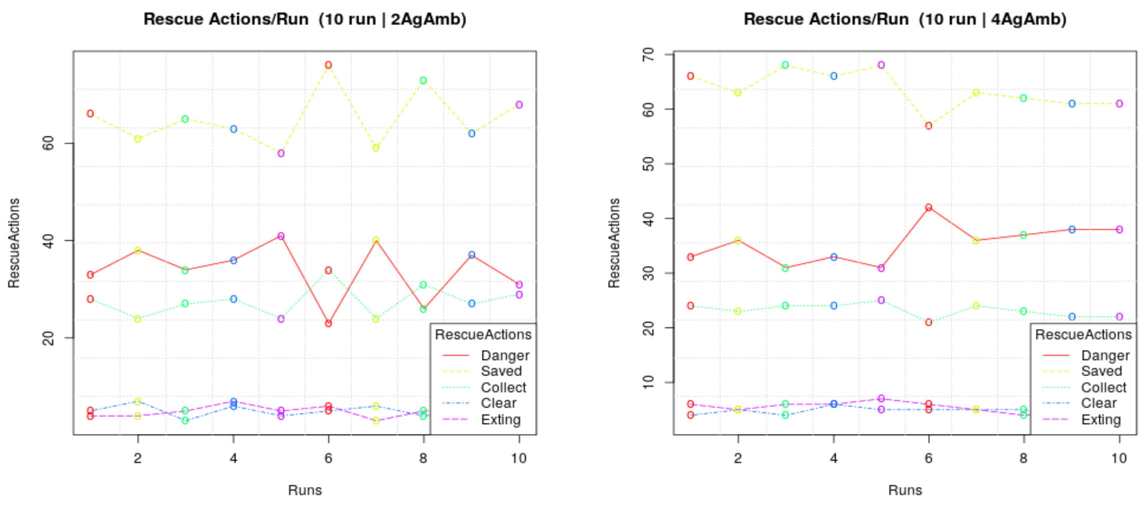


**Figure 5.12:** *Summary of the performance of each of the 10 runs of the BDI MAS-DR with Collective Intentional Learning, varying the number of ambulances form 2 and 4, interchanging rewards beliefs:* `rewardAcSt`*.*

a) 2 *ambulances*    b) 4 *ambulances*

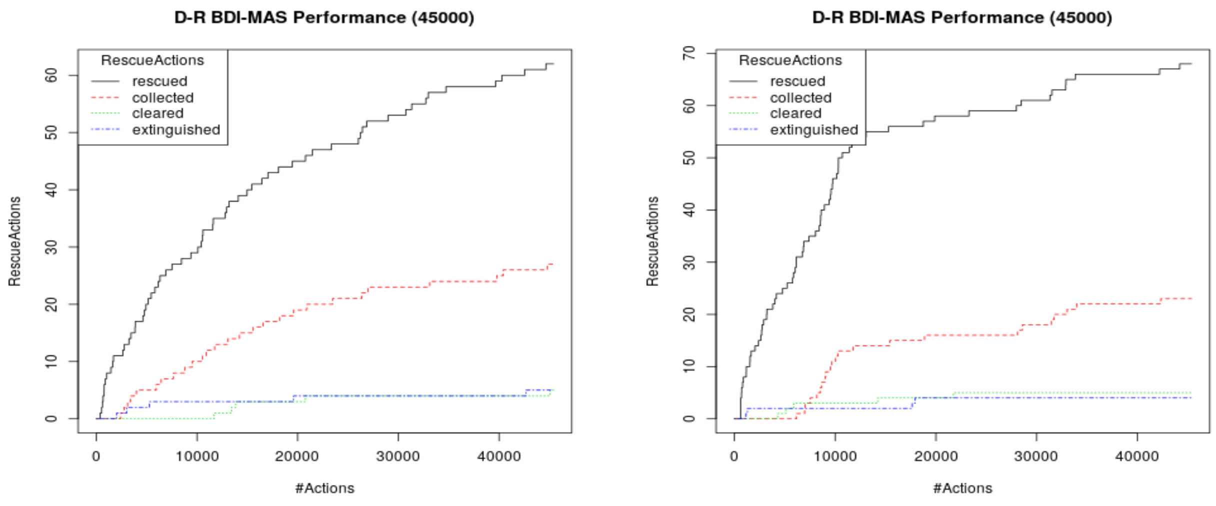


**Figure 5.13:** *The performances of the BDI MAS-DR in the 10th run, with exchanges of* `rewardAcSt` *beliefs, varying the number of ambulance agents: 2 and 4.*

The performance of the *ambulances* in the experiments can be observed in a more detailed way in Figure 5.14, where the number of collected civilians per ambulance during each of the runs is shown.

In the first case, one *ambulance* collected five times more victims than the other one. Whilst in the 4 *ambulances* settings, the load is more balanced, resulting in almost the same amount of victims collected by each agent.

In addition, in Table 5.15 and 5.16, the plan updating frequency of each *ambulance* in the 10th run for the

a) 2 *ambulances* a) 4 *ambulances*

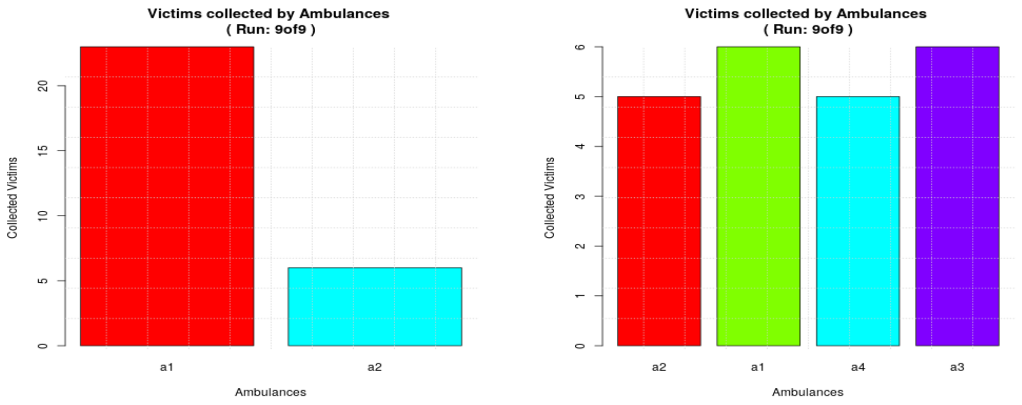


**Figure 5.14:** *The amount of collect actions per ambulance from the BDI MAS-DR of each run with 2 and 4 ambulances.*

two variations of agents is shown.

*ambulance* 1 *ambulance* 2

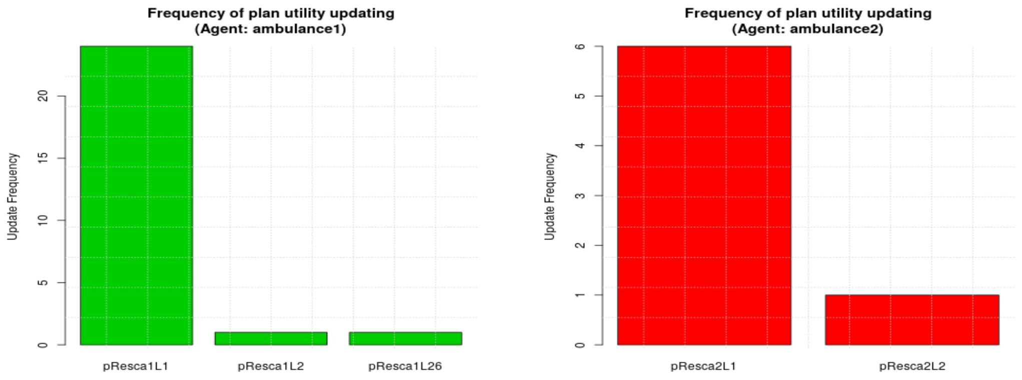


**Figure 5.15:** *The plan updating frequency per each ambulance within the D-R BDI MAS, from the 10th run with 2 ambulances.*

Agent *ambulance* 1 updated its plans more times than the other agent. Furthermore, *ambulance* 1 updated its first generated plan (denoted by the suffix `L1`) more frequently than the others. *ambulance* 2 updated one of its plans in six times, whilst the other plan was updated just once.

In the case of 4 *ambulance* agents, the amount of plan updating of agents 1 and 2 almost coincides: both have two updated plans and one of them was more frequently updated than the other one, making a big difference between them. Similarly, in the cases of *ambulance* 3 and 4, they updated all the plans but one of them was more frequently updated than the rest of plans.

To illustrate how learning is achieving the set of plans to perform *ARB*, some examples of the generated plans in ambulances, are listed as follows.

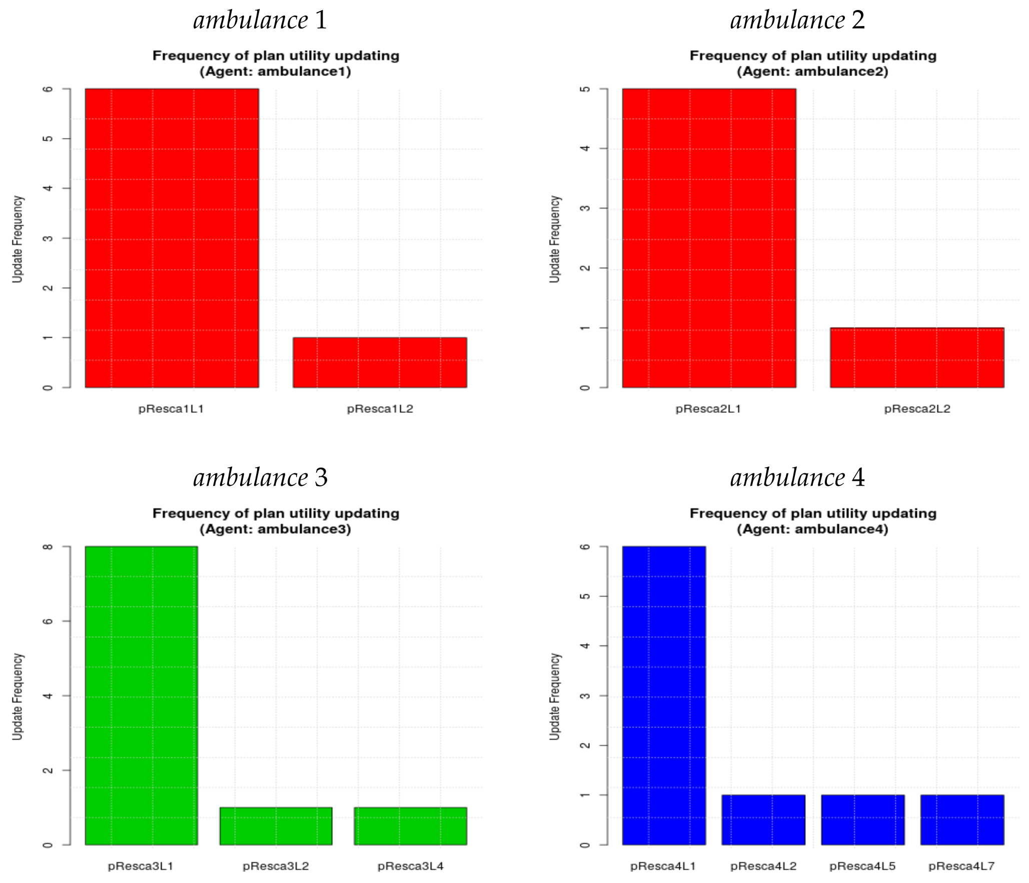


**Figure 5.16:** *The plan updating frequency of ambulance within the D-R BDI MAS, from the 10th run with 4 ambulances.*

```
 @pResca1L1[learn(5),utility(-1)]
+!rescueCivil(C) : belowCapacity
<- !goToTarget(C); !goToTarget(C); !goToTarget(C); !goToTarget(C); !goToTarget(C).
@pResca1L2[learn(2),utility(-1)]
+!rescueCivil(C) : (noMoreCapacity & posAmb(onBase))
<- !goToTarget(C).
@pResca1L6[learn(2),utility(-1)]
+!rescueCivil(C) : ((posAmb(outBase) | posAmb(onTarget(_))) & noMoreCapacity)
<- !goToBase; !unload; !goToTarget(C).
@pResca1L7[learn(2),utility(0.01)]
+!rescueCivil(C) : (noMoreCapacity & posAmb(onBase))
```

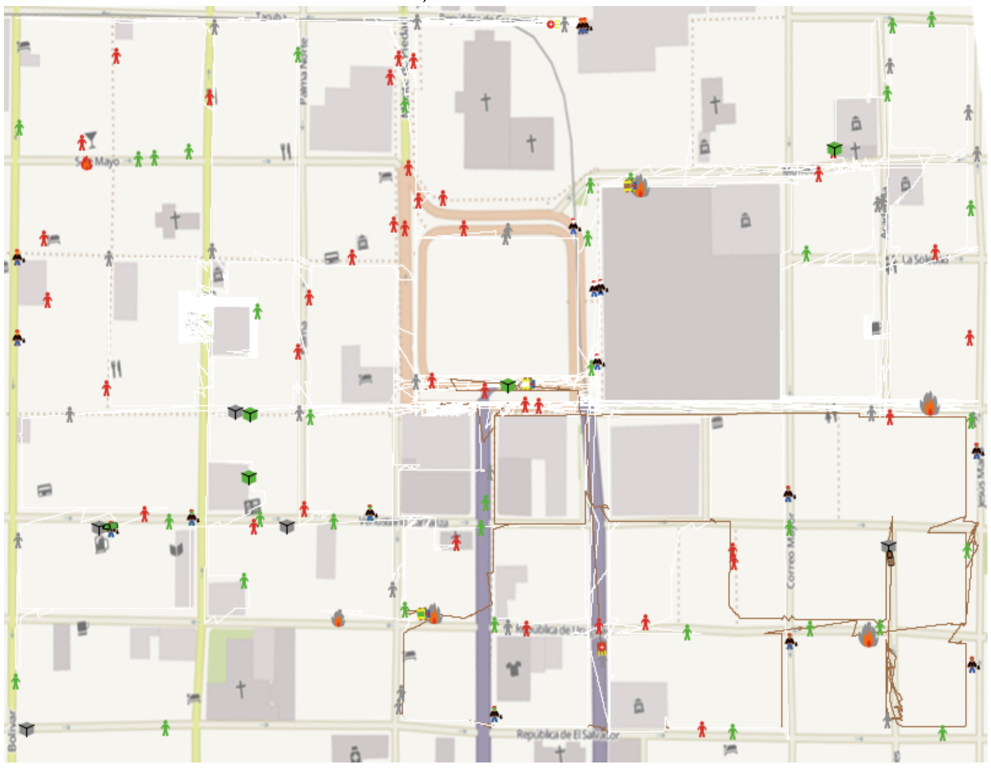
```
<- !unload; !goToTarget(C); !detCivilian(C).
@pResca1L14[learn(2),utility(-1)]
+!rescueCivil(C) : ((posAmb(outBase) | posAmb(onTarget(_))) & noMoreCapacity)
<- !goToBase; !unload; !goToTarget(C).
@pResca1L15[learn(1),utility(0.01)]
+!rescueCivil(C) : (belowCapacity & civilRescuedFnd(C))
<- !pickupCivilian(C); !goToBase; !pickupCivilian(C).
@pResca1L16[learn(2),utility(-1)]
+!rescueCivil(C) : ((posAmb(outBase) | posAmb(onTarget(_))) & noMoreCapacity)
<- !goToBase; !unload; !goToTarget(C).
@pResca1L26[learn(2),utility(-1)]
+!rescueCivil(C) : ((posAmb(outBase) | posAmb(onTarget(_))) & noMoreCapacity)
<- !goToBase; !goToTarget(C); !unload.
@pResca1L27[learn(2),utility(-1)]
+!rescueCivil(C) : (noMoreCapacity & posAmb(onBase))
<- !unload; !goToTarget(C); !detCivilian(C).
```

The set of plans generated based on the sharing of rewards (`rewardAcSt`) reached the *ARB*. The four cases observed in Section 5.5.2 are present here also: plan with positive values of utilities are the most useful to perform *ARB*, but they include redundant primitives (can be avoided and the agent still could perform *ARB*); on the other hand, negative values could contribute in pursuing *ARB*, but some of them require a lot of improvements to be useful. The last point can be illustrated from the plans listed above: the first plan `@pResca1L1` which has been modified in five times, contains a repetition of the primitive `!goToTarget(C)`, despite it is true that this primitive could be useful within other context, the repetition does not improve the behaviour, representing a wasting of resources (it also points out why the template with repetitions could be not so useful in domains like the current one).

Finally, in Figure 5.17, two snapshots of the 10th run with 2 and 4 *ambulances* are depicted. Coloured traces show the trajectories of the agents once they touched the base to unload the rescued civilians, i.e. once an agent has achieved a set of plans to perform the *ARB*.

The snapshot (a) contains white traces, recalling Figure 5.14 (a), where the *ambulance* 1 is pointed as the

a) 2 *ambulances*



b) 4 *ambulances*

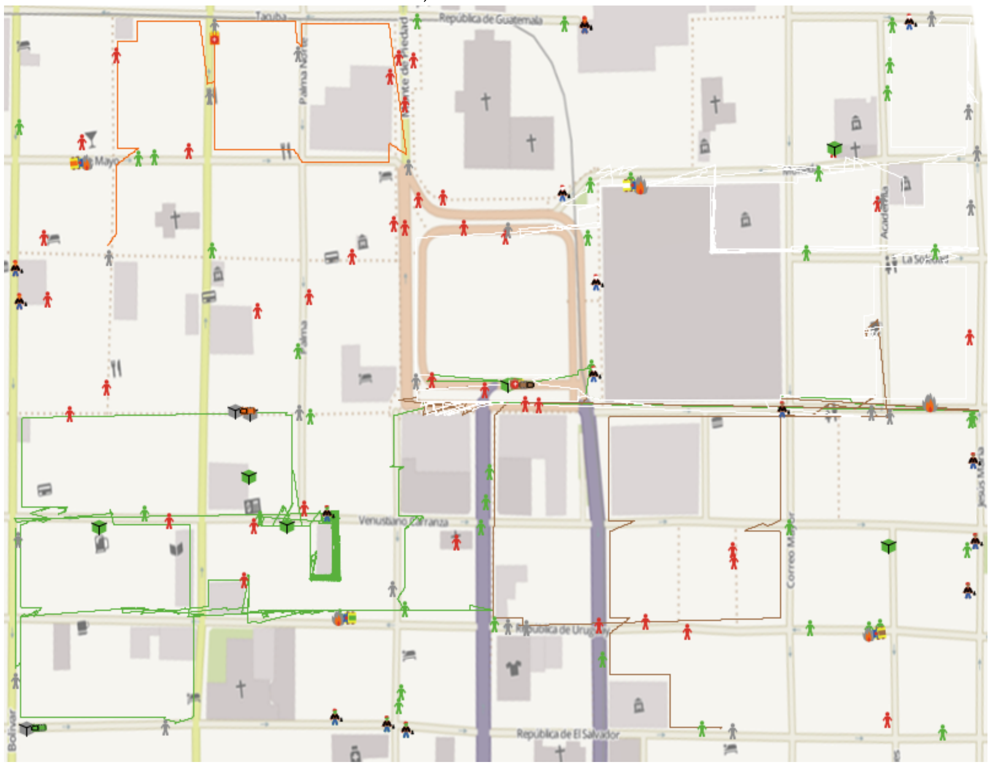


**Figure 5.17:** *The final state of the BDI MAS-DR: a) the run with 2 ambulances, and b) the run with 4 ambulances*

most active agent to collect victims, consequently, it traverse more territory in the city map. In snapshot (b), the traces reveals a more balanced load for *ambulances*, as reported in Figure 5.14 (b) as well.

### 5.5.4 Exchange of Q-Values

In this experiment, the agents interchange `qr` values, and generate the `qs` beliefs to compare the benefit reported by a primitive, accordingly with a given context and the shared `qr` values. Similarly to the reward exchange test, the results report the performances, the amount of victims collected, the plan update frequencies, the generated plans, and a view of the system as a whole while running. *ambulance* agents are varied from 2 to 4.

Firstly, to illustrate the exchange of `qr` values, in Figure 5.18 this interaction is represented as directed graphs from different stages and runs.

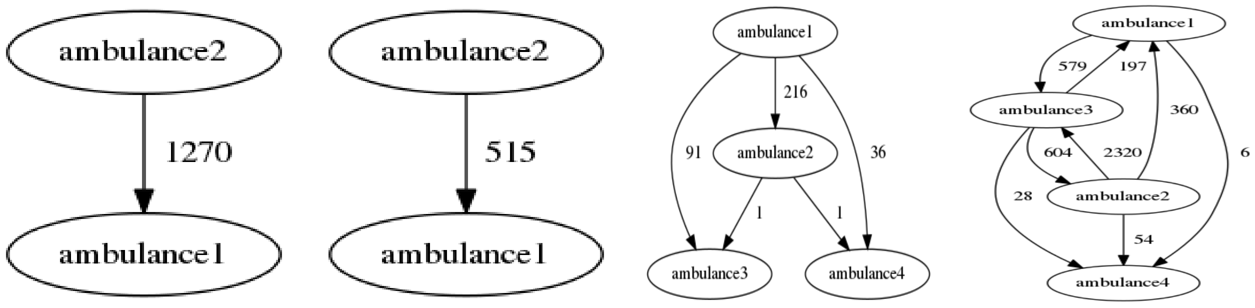


**Figure 5.18:** *The `qr` exchange between ambulance within the BDI MAS-DR, from the 10th run with 2 and 4 ambulances, represented as a directed graph.*

In Figure 5.19 the plots of summarized performances of 10 runs with 2 and 4 *ambulances* up to $450,000$ actions are depicted.

From the summaries, some statistical values can be identified. In Table 5.5, the minimum and maximum, and the median of the summarized performances; jointly with the correlation between Rescued and Collected lines are presented. The mean of the Saved value in the runs of 2 *ambulances* is 65.9, whilst the mean of Collected victims is 23.3, both values show a correlation of 0.7723. On the other hand, 4 *ambulances* shows a correlation of 76.81 between Saved and Collected victims, with a maximum of 36 and a minimum of 25 for values of Collected. The trade-off in this case, is the larger communication between agents, as they interchange a higher amount of information, including requests and refusing signaling (as can be seen in

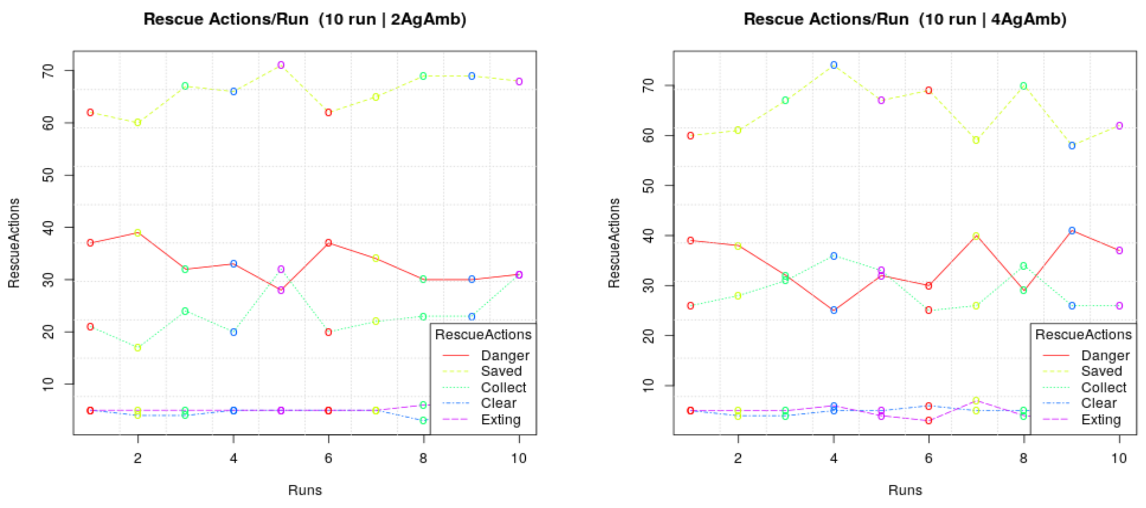


**Figure 5.19:** *Summary of the performance of each of the 10 runs of the BDI MAS-DR with Collective Intentional Learning, varying the number of ambulances from 2 to 4.*

Figure 5.18 above).

| | **Danger** | **Saved** | **Collect** | **Clear** | **Extinguiseduised** |
|---|---|---|---|---|---|
| | | | 2 *ambulance* agents | | |
| **m** | 28.00 | 60.00 | 17.00 | 3.00 | 5.00 |
| **n** | 32.50 | 66.50 | 22.50 | 5.00 | 5.00 |
| $\mu$ | 33.10 | 65.90 | 23.30 | 4.70 | 5.30 |
| **M** | 39.00 | 71.00 | 32.00 | 6.00 | 6.00 |
| **C** | | | **0.7723** | | |
| | | | 4 *ambulance* agents | | |
| **m** | 25.00 | 58.00 | 25.00 | 3.00 | 3.00 |
| **n** | 34.50 | 64.5 | 27.0 | 5.00 | 5.00 |
| $\mu$ | 34.30 | 64.70 | 29.10 | 4.70 | 4.80 |
| **M** | 41.00 | 74.00 | 36.00 | 6.00 | 7.00 |
| **C** | | | **0.7681** | | |

**Table 5.5:** *The minimum (m), the median (n), the mean $\mu$, the maximum (M) of the summarized values for the 10 runs of the D-R BDI MAS, varying the number of ambulance agents, and the Correlation (C) between Danger and Collected values*

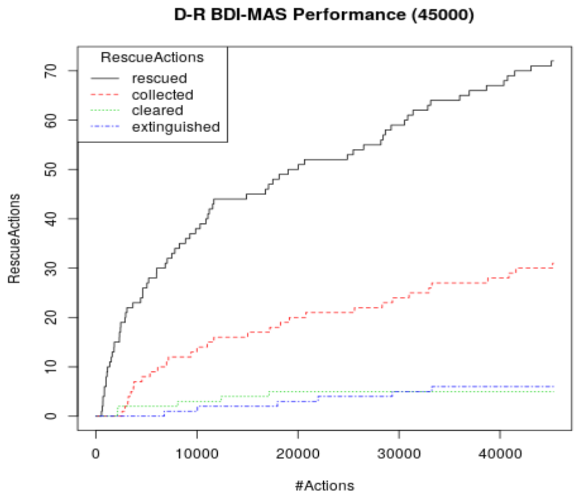
Figure 5.20 shows a brief overview of how the D-R BDI MAS is working (highlighting the *ambulance* agents), with the plots of the performances belonging to the 10th run of the system.

For identifying more specifically the performance of the *ambulances* in the experiments, in Figure 5.21 the number of collected civilians per ambulance during each of the runs are presented.

In Table 5.22, the updating frequency of plan belonging to each *ambulance* agent in the 10th run are depicted, including just the 2 agents setting.

To show how the learning achieves the set of plans to perform *ARB*, a snippet of the plan building and

a) 2 *ambulances*



b) 4 *ambulances*

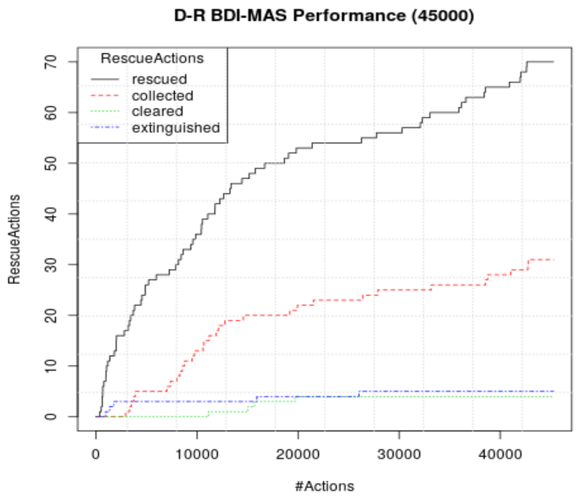


**Figure 5.20:** *Performances of the BDI MAS-DR in some runs varying the number of ambulance agents from 2 to 4, sharing* `qr` *values.*

the generated plans, belonging to one of the ambulances from the reported experiments, namely *ambulance* 1 of the 2 agents settings, are listed as follows.

```
// Learning process of plan acquisition
[ambulance1] >>>>>>>>-->-->--> Error in plan rescueCivil(42)
[ambulance1] Going to learning internal action: with template List l and state
st(posAmb(onBase),civilRescued(42),distCBase(42,364.0467),distCAmb(42,363.2987),
capacity(5),miSector(1))
```

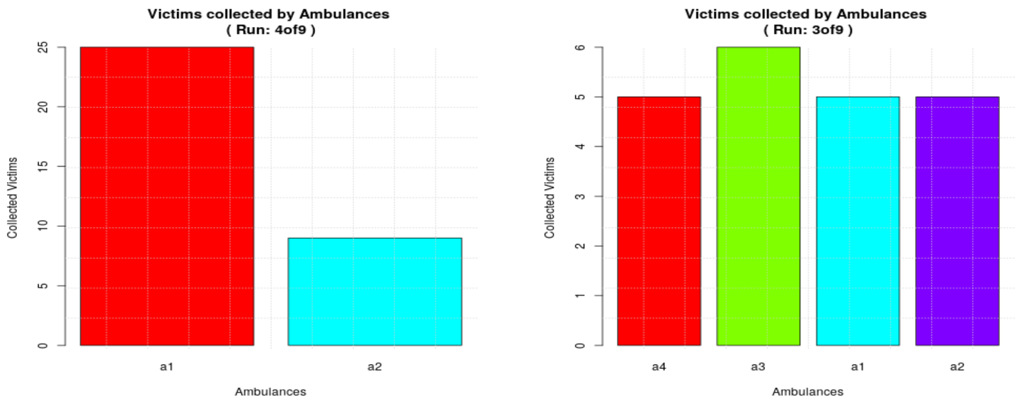


**Figure 5.21:** *The collection actions per ambulance within the BDI MAS-DR in different runs with 2 and 4 ambulances while sharing* `qr` *values.*

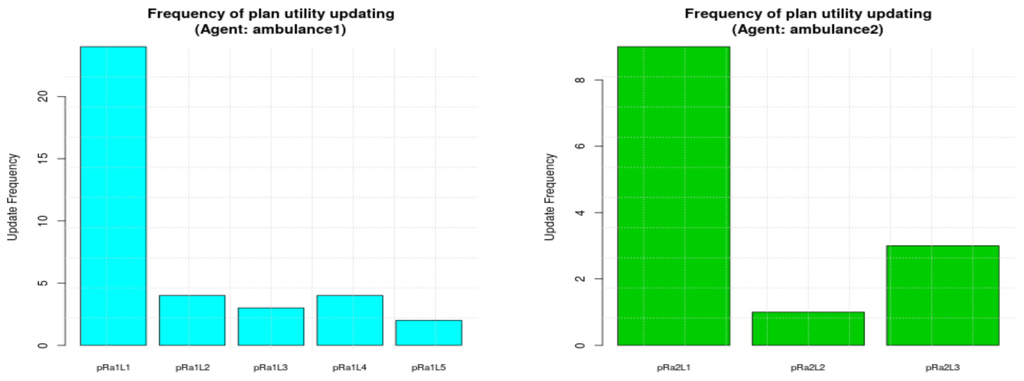


**Figure 5.22:** *The updating frequency of plan per ambulance within the BDI MAS-DR, from the 10th run with 2 ambulances.*

```
[ambulance1]    >>>>>>Arg[0] :rescueCivil(42)[source(self)]
[ambulance1]    >>>>>>Arg[1] :no_relevant
[ambulance1]    >>>>>>Arg[2] :[st(posAmb(onBase),civilRescued(42),
distCBase(42,364.0467) ,distCAmb(42,363.2987), capacity(5),miSector(1))]
[ambulance1]    >>>>>>Arg[3] :"pResca1L"
[ambulance1]    >>>>>>Arg[4] :1
[ambulance1]    >>>>>>Arg[5] :1
[ambulance1]    >>>>>>Arg[6] :0
...
[ambulance1][cil-plan-generation]: 3. Collecting plans: taken from PlanLibrary
[ambulance1][cil-plan-generation]: 4. Collecting Primitive Actions from PlanLibrary
```

```
[ambulance1][getPrimActionsPL] [-+-]Primitive: goToTarget(C)

[ambulance1][getPrimActionsPL] [-+-]Primitive: detCivilian(C)

[ambulance1][getPrimActionsPL] [-+-]Primitive: pickupCivilian(C)

[ambulance1][getPrimActionsPL] [-+-]Primitive: goToBase

[ambulance1][getPrimActionsPL] [-+-]Primitive: unload

[ambulance1][cil-plan-generation]: 5. Ranking Primitive Actions

[ambulance1][SelectRankCandActions] [-+-]Current primitive goToTarget(C)

[ambulance1][SelectRankCandActions] [-+-]Searching in Qr

[ambulance1] [primRevQr]    [-i-r]Querying goToTarget(C)

[ambulance1] [primRevQr]    [-i-r]  xNOT FOUND in qr()

[ambulance1] [primRevQr]        [-i-]      Utility set to -Infinity

[ambulance1][SelectRankCandActions] [-+-]Searching in rewardAcSt

[ambulance1] [primRew]  [-i-r]Querying goToTarget(C)

[ambulance1] [primRew]  [-i-r]  FOUND

[ambulance1] [primRew]  [-i-r]Utility reported by action [goToTarget(C)]:

[ambulance1] [primRew]  [-i-] rewardAcSt's utility: 1.0

[ambulance1][SelectRankCandActions] [-+-]Current primitive detCivilian(C)

[ambulance1][SelectRankCandActions] [-+-]Searching in Qr

[ambulance1] [primRew]  [-i-r]Utility reported by action [unload]:

[ambulance1] [primRew]  [-i-] rewardAcSt's utility: 0.0

...

[ambulance1][SelectRankCandActions] [-+-]Returning: 5 primtives

[ambulance1][cil-plan-generation]: Ranking of Primitives:

[ambulance1][cil-plan-generation]: goToTarget(C), 1.0

[ambulance1][cil-plan-generation]: detCivilian(C), 0.5

[ambulance1][cil-plan-generation]: unload, 0.0

[ambulance1][cil-plan-generation]: goToBase, 0.0

[ambulance1][cil-plan-generation]: pickupCivilian(C), 0.0

[ambulance1][cil-plan-generation]: 6. Set Body with Primitive Actions and TEMPLATE 1
```

```
...
```

Then, the outcome of the learning process is a new set of plans, as those listed as follows.

```
// ambulance 1
@pResca1L1[atomic,learn(3),utility(22.9309)]
+!rescueCivil(C) : belowCapacity
    <-  !goToTarget(C); !detCivilian(C); !goToTarget(C);
        !goToTarget(C); !pickupCivilian(C).
@pResca1L2[atomic,learn(8),utility(-0.2034)]
+!rescueCivil(C) : ((posAmb(outBase) | posAmb(onTarget(_))) & noMoreCapacity)
    <- !goToBase; !goToTarget(C); !goToTarget(C).
@pResca1L3[atomic,learn(1),utility(-0.0745)]
+!rescueCivil(C) : (noMoreCapacity & posAmb(onBase))
    <- !unload; !goToTarget(C); !goToTarget(C) .
@pResca1L4[atomic,learn(5),utility(-0.2605)]
+!rescueCivil(C) : ((posAmb(outBase) | posAmb(onTarget(_))) & noMoreCapacity)
    <- !pickupCivilian(C).
@pResca1L5[atomic,learn(2),utility(0.01001)]
+!rescueCivil(C) : ((posAmb(outBase) | posAmb(onTarget(_))) & noMoreCapacity)
    <- !goToBase; !goToTarget(C); !goToTarget(C).
```

The plan labelled `@pResca1L1`, has a very high utility, recalling that it is used for plan's eligibility, jointly with the plan's context, which is tested the first. This plan together with the others configure a set of plans to perform *ARB*, even if some of them have a negative utility due to their applicability context. In Figure 5.23 the plan utilities at the end of the 10th run (Time, in the X axis represents the number of updates rather a timing measure), and the plots of the plans during the run are illustrated.

Finally, a snapshots of the 10th run are depicted in Figure 5.24 from 2 *ambulances*. As usual, coloured traces show the trajectories of the agents once they touched the base to unload the rescued civilians, i.e. once an agent has achieved a set of plans to perform the *ARB*.

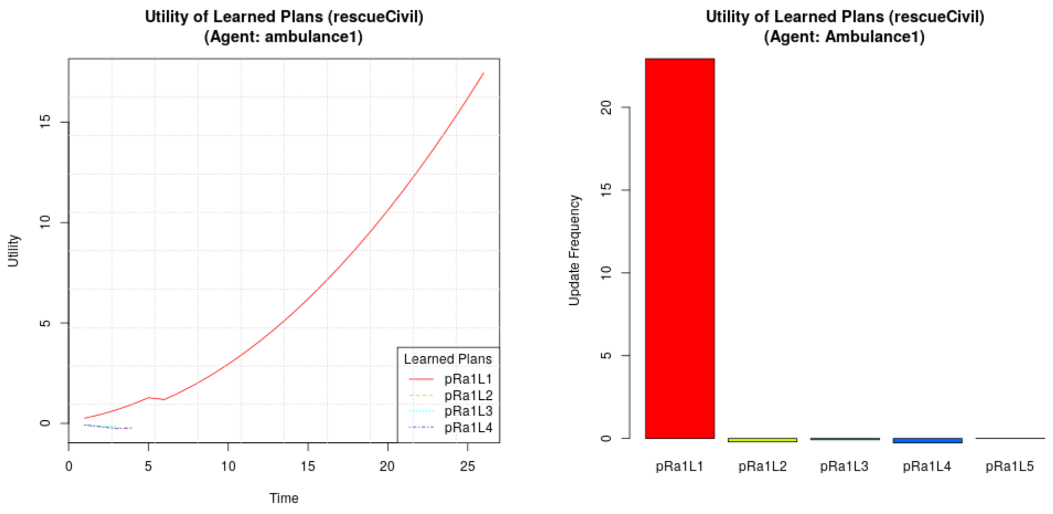


**Figure 5.23:** *The plan utilities of ambulance 1 in the 10th run of the D-R BDI MAS. a) plot of the utilities while running; b) the final utilities.*

a) 2 *ambulances*

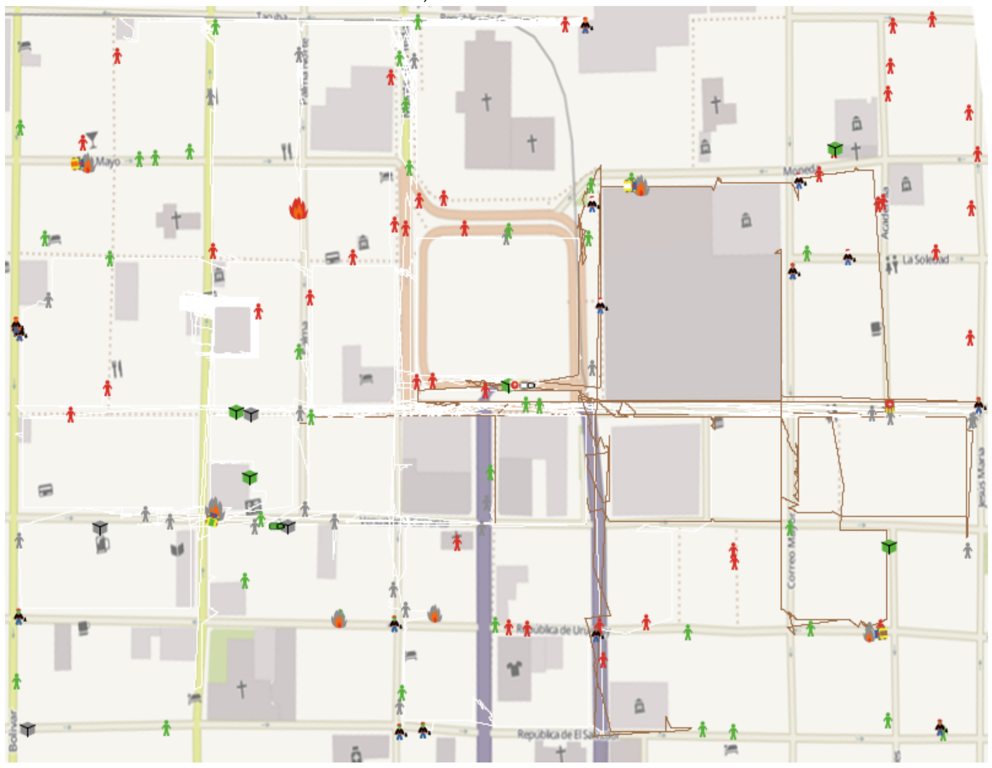


**Figure 5.24:** *The final state of the BDI MAS-DR: the run with 2 ambulances.*

## 5.6 Summary

This chapter presents the work to fulfil the Research Objective 3.

The reported method is one of the main contributions of this thesis: an extension of intentional learning

(IL) that goes from the individual to the collective scope: Collective Intentional Learning (CIL). Learning agents are tested in the Disaster-Rescue domain, a challenging and socially relevant one, where BDI agents are a promising approach for developing realistic simulations.

CIL builds up the IL framework discussed in Chapter 4. It is focused on the level of plan acquisition as a technique to generate a set of plans, based on the interaction with the environment and other agents in the system. The target of the learning is the *ambulance*-related behaviour. Briefly speaking, this is the set of plans that allow an *ambulance* agent to collect civilians in danger (that have been reported), and rescue them, in order to relocate them to a safe place.

As in IL, a set of tracking beliefs ($\Theta$) are used to register the state-of-affairs of the environment, and the result of executing plans in a given state. Additionally, agents need to store the shared knowledge, and a list of team-mates to interact with, plus the weights and utilities associated to them (defined by the $\Lambda$ criterion: action and plans utilities, either shared by a partner or collected by its own). Learning aspects, such as the Exploration and Exploitation Dilemma, are performed in similar way that in IL.

Regarding the utility, updating are tackled as usual in IL, but integrating more information coming from other agents. Consequently, each agent has two strata of decision, the local or individual and the collective. The former allows the agent to manage the firing of plans, including triggering the learning processes, and the updating of the utilities for each executed plan. The latter refers to determining the acceptability of the knowledge that is shared by other agents, when requested.

Under CIL, agents exchange relevant information to help each other in the acquisition of new plans, for improving their behaviour and generate new skills. The plan acquisition is oriented to the learning of an agent behaviour almost from scratch. The plans are derived from primitive actions and a pairwise sequence of interactions with other agents, in accordance with the social learning approach.

Despite the main interaction is performed by agents of the same type, learning agents can interact with heterogeneous agents as well by means of a Communication Protocol. This protocol establishes a contract of collaboration between agents in order to interchange information with members of the same team or from other teams in the MAS. Agents request either plans (in the case of homogeneous agents) or environmental samples (in the case of heterogeneous agents) to perform a given task.

Finally, the experiments presented here in this chapter are focused on showing how the CIL works, and the interaction between agents. Besides, the performance of the agent and the fulfilment of the target rescue-related behaviour defined in the Learning Task 5.5.1, namely the Ambulance Rescue Behaviour (*ARB*). The results include the following:

- Summaries of the D-R BDI MAS performance in 10 runs under different variation of agents, and exchanged knowledge.
- For brevity, only the results from the 10th run are presented:
  - The plot of the DR BDI MAS performance
  - The bar plots of the amount of collected civilians per *ambulance*
  - The plan update frequencies and shared knowledge
  - Examples of the learned plans
  - The view of the system before finishing with variations of *ambulances*

# Chapter 6

# Conclusions

## 6.1 Concluding Remarks

Multi-Agent Systems (MAS) are a feasible approach for studying complex systems. They can represent either natural or artificial, living and non living systems, including their elements and relationships. Being complex systems on their own, the development of MAS is a source of complexity itself.

MAS have mainly been developed using reactive agents, in a wide range of application domains [4,43,55, 92], including simulations of groups and social phenomena such as crowds, traffic control, ethnocentricity, emergency-response teams, emergency evacuation, between others; and natural systems, for instance bacteria simulations, physic processes and chemical reactions, ecological phenomena and epidemiology like deceases spreading. This is due to the relatively simple control loop needed and because they are easy to implement [4, 135]. Furthermore, in such domains it is common to use a higher level of abstraction that does not require detailed decision-making processes in agents allowing to consider them as homogeneous individuals and to represent several individuals as just one single agent [4,43].

Cognitive agents, particularly BDI agents, are an interesting avenue for exploring a richer and more powerful architecture, representing elements with a higher degree of complexity, which could help to represent some aspects of human behaviour in a more realistic way [4,83,107,108].

It is true that BDI architecture lacks from learning capabilities in its original definition. However, this has been tackled by using different Machine Learning techniques, Intentional Learning (IL) being one of the most adequate, because its definition is fully integrated with the BDI architecture (in the reasoning loop, and the definition of agents, as established by the BDI theory). IL has been focused on goal adjustment, plan selection, and plan generation. This framework of learning has been approached in BDI agents, both in the single-agent settings and in the MAS perspective, but only for goal adjustment and plan applicability.

The review of the literature of MAS unveils that there are some drawbacks due to in the earlier stage of development of MAS the focus was on the agents more than in their collective abilities. For instance, Planning and Machine Learning were growing up as independent disciplines applicable to a wider range of computing problems more than only as a single-agent related capability.

The work presented here is motivated by the following drawbacks found in the literature of MAS: a) Plan acquisition has been introduced in BDI-like architectures, but only in the single-agent case. b) Planning in fully BDI architectures has been approached in the single-agent case, but using external planners and translating the results back to the BDI formalism. c) IL at the level of plan acquisition has not been endeavoured in MAS composed of BDI agents. d) Realistic simulations can be achieved when using BDI agents, due to their capabilities to emulate some aspects of human behaviour like planning, norm representation, and heterogeneous decision making allowing inter-agent differences and practical reasoning. Additionally, BDI agents manage higher level communication and symbolic representation of their reasoning based on a theory of mind able to be explained and designed in terms of folk psychology understandable for non-computer science specialists that allows the interdisciplinary collaborations to set a simulation with knowledge from different fields [4,107,108]. However, they are not extensively used to perform such simulations. e) MAS can benefit from adopting real life domains, they configure challenging environments (such as the Disaster-Rescue problem), where different aspects of agents (e.g. learning) can be tested. This is due to real life domains generally implies a partially accessible and dynamic environment where agents need to constantly evaluate their perceptions and the results of their actions. Moreover, they have partial communication conditions, implying that agents need to deal with interruptions and incorrect information, and common/shared goals are managed by the group of agents in order to fulfill the global task. Even more, D-R domain involves sets of heterogeneously skilled Reactive and Cognitive agents, where hierarchical organisation is needed [55,66,108]: forming teams, coalitions, networks, etc. So, such environments represent more challenges to the agent's design and operation, giving the opportunity to test several algorithms to cope with the aforementioned situations.

The main contributions of this research to the state of the art in MAS, are:

- A method to introduce IL at the level of plan acquisition, in a fully-fledged BDI framework (Jason), generating plans that drive individual agents towards new behaviours. This contributes to fill the gap in having purely BDI agents enhanced with learning.
- The planning method generates plans directly represented in Jason's code (AgentSpeak). As a result, the use of external tools (like traditional planners) is avoided. Moreover, agents built new plans solely based on their primitive actions and their interactions with the environment. This aims to progress the state of the art in Jason agent development.
- IL is extended from the single-agent to the collective scope, through the method named Collective Intentional Learning (implemented in Jason), progressing the state of the art in BDI MAS.
- A framework of experimentation, where IL can be included in both the single-agent and the MAS scope is proposed. This framework progresses the state of the art in MAS development, bridging the gap between Agent-Based Modelling and MAS approaches. It melds the strengths of reactive and BDI agencies, and is based on the interoperability of both main development tools: Jason and NetLogo.
- A socially relevant simulation is developed, composed by heterogeneous (BDI and reactive) agents, and applied to the Disaster-Rescue domain. This addresses seldom use of cognitive agents in simulations.

### 6.1.1 Intentional Learning, Self-Organisation and Emergence

Self-organisation (SO) and Emergence (EM) are two phenomena observed in complex systems either together or in a separate way [32]. MAS are a sort of complex systems, so both SO and EM are also present in them [33].

SO refers to the capability of a system to change its organisation as a result of external stimuli for the sake of continuing operating in the environment and still fulfilling its task. EM is not about optimization but the rising of unexplainable outcomes in terms of the individual parts. In this concern, EM is said to happen when there is an observable outcome that is not reducible to the individual components and their capabilities, it is observable as a physical change (in the shape or organisation of its parts) and/or qualitative one (exhibiting new abilities or behaviour). There can be SO as a result of EM, or predefined in design time. EM could drive the MAS to a different outcome than SO. Even more, the emergent outcome could be not helpful to the fulfilment of the MAS global objective.

In this thesis, SO can be observed as a result of learning while the system operates in the environment: ambulances start with no procedural knowledge to fulfill their task (the Ambulance Rescue Behaviour or *ARB*), and at some point they build a sort of plans to behave as ambulances performing the *ARB*. In this sense, SO is reached by means of EM.

To understand how this is possible, let's observe that there is a relation between ML and a key concept of EM, the Explanatory Gap (EG) [44, 45]: when emergent outcomes (patterns, structures or behaviours) are observed in a system, they cannot be interpreted based solely on the level of explanation of the system's components (substrate). There is a gap in the explanation of the outcomes that requires to have a different interpretation or to make a redefinition of the phenomena using a higher level of abstraction that encompasses the component's lower level of explanation.

In [45] three conditions to understand the EG are proposed. They could be used as clues for understanding EM in computing systems and how it is related with ML in the present work:

1. Imagine the traversing of the transformation from substrate (expressed in a language $D$) to outcomes (expressed in a language $D'$). This transformation is performed by ML, which purpose is to transform a given input to an output by executing a sequence of operations.

2. Imagine how this transformation goes from origin to terminus ($D \rightarrow D'$) keeping in mind the EG along the way. It does refer to those constraints and conditions to change the agent's definition when acquiring a new plan: the assorted primitive plans ($D$) to be part of new plans to perform the *ARB* ($D'$) when added to the agent's *PlanLibrary*.

3. Imagine how this can be accomplished by natural means (not supra natural ones). It is about the EM's mechanism that obeys the natural laws of the substrate, where emergent outcomes are produced accordingly. This is defined by the learning strategy of IL/CIL for plan acquisition explained in Chapters 4 and 5, how to combine different set of primitives guided by observations and the criteria for selecting

those to be part of a new plan.

Then, EG occurs when we observe that primitive plans could drive to a different set of plans, steering the agent to possible behaviours other than *ARB*, i.e. there is a right sequence of primitives to achieve *ARB*, but this behaviour is not directly interpretable from the original set of primitives on their own.

To summarize, the relation between ML and the EG, could represent a kind of strategy to understand SO and EM in MAS. In this work, the addition of ML in a MAS contributes to the promotion of EM by building from a base language of representation (the set of primitive actions) towards a new language, with a higher level of expressiveness for problem solving (the set of plans encoding the intended behaviour –*ARB*). SO is reached by the arrangement of the agents whilst interacting with each other, following their own behaviours with no central agent driving the others, nor centralized plans or control.

## 6.2 Research Achievements

Chapter 2 of this work provides a review of the field of MAS and their relationship with Machine Learning (ML), Planning in agents, and the relevance of Reinforcement Learning (RL) based techniques. Additionally, there is a study of the relationship between MAS and ML with two central concepts in complexity: Self-organisation and Emergence (particularly Computational Emergence) and the relevance for taking those concepts into account when developing MAS. This study reports the main findings in literature, showing endeavours performed by the wider community, and the similarities and differences in these findings. It emphasises the drawbacks that ultimately inspired this work, which are addressed by the inception of IL in a full-fledged BDI environment, and a proposal for lifting this learning framework form the single-agent to a collective perspective, via a pairwise-based social learning approach.

In Chapter 3, the background of BDI agency, a description of the learning problems in the single-agent and MAS scopes, and the framework of experimentation needed to develop learning in fully-fledged BDI architecture, are presented. IL in fully BDI agents, at the level of plan acquisition, have not been reported in literature. Previous work has focused on BDI-like architectures [116,117]. Then, this chapter shows how IL, as a planning process, is integrated in a full BDI architecture, providing a problem description, a definition of the learning, and the cases where it is applicable. Similarly, a description of the problem of IL in the collective scope is provided.

This chapter introduces a framework focused on coupling two approaches: ABM and MAS, emphasising the direct representation of elements within a system, and including their relationships and interactions in the system's life time. The integration melds BDI and reactive agents. The BDI deliberation enriches the decision-making process of reactive agents with a higher level reasoning. The result of the integrating both approaches boosted the resultant system, by exploiting their modelling capabilities and complex behaviours reached with BDI agents. This represents a novel contribution to the MAS development tools, given that it integrates two of the major MAS programming environments: NetLogo and Jason. NetLogo offers a powerful rendering tool for representing rich environments and reactive-based elements. Jason provides a robust decision-making process driven by the BDI agents reasoning cycle. Two options of control are explored: reactive agents can use the BDI reasoning cycle as required; or BDI agents can control the deliberation, maintaining a sharp separation between decision-making and action in the environment (utilizing reactive agents as an extension to perceive or react within the environment). Finally, a BDI MAS applied to the Disaster-Rescue domain (DR) is developed using the framework.

In Chapter 4, IL, the method for learning in intentional systems is presented and tested in the experimentation framework. The method reported in this chapter, extends the Jason BDI agency, through providing a method to acquire plans based on the primitive actions stored in the *PlanLibrary* of agents. IL is integrated in a full-fledged BDI environment, namely Jason. IL is implemented using three components: a set of tracking beliefs ($\theta$), a ranking criteria ($\lambda$), and a plan building strategy ($\psi$). The purpose of the learning is the generation of new plans in the agent *PlanLibrary*. $\psi$ is implemented through three plan templates: add one action, repetitions and sequences. Learning is also a result of the interaction between an agent and its environment, so the set ($\theta$) is used for registering the environmental changes, arising from the execution of plans, controlled by the plan templates and the weighting mechanism $\lambda$, inspired by RL methods. The domain for demonstrating learning is the single-agent vacuum cleaning environment (making use of the NetLogo as the environment where the agent is located). The learning agent starts by having no definition of the intended behaviour to be acquired, but a set of primitive actions that would comprise the new plans accordingly with the IL process. Two behaviours are acquired: target-searching and left-handed wall-following. Furthermore, IL is applied to the D-R domain to acquire an ambulance-related behaviour: to collect victims and drive them to a safe place (i.e. the base of the MAS). The template used allowed the agent to learn plans with sequences of actions.

Additionally, the method proposed here enables planning in Jason, generating new plans directly in AgentSpeak, the Jason's programming language. Then, differently from other approaches [6,74], the reported method avoids the use of external planners by integrating the planning generation in the same set of plans available to agents. Consequently, there is no need of translation processes between different plan representations. This constitutes also a contribution in the Jason BDI MAS development.

After introducing IL in a pure BDI agent oriented framework in the level of plan acquisition, and demonstrating learning in the single-agent case, the next step is presented in Chapter 5: an extension of IL, namely the Collective IL method, focused on the generation of plans through two scopes of decision, the local and the collective ones. The CIL method extends IL by accounting for the viewpoint of other agents in the MAS, via direct agent messaging and knowledge sharing. Agents interacts with others under a social learning approach, based on a pairwise interchange of knowledge and criteria of knowledge of acceptability defined by a parameter $\Phi$. This knowledge sharing can be performed between homogeneous or heterogeneous agents (i.e. interacting with a peer or a team mate).

As it builds on IL, the CIL method relies upon an extended version of $\Lambda$, the RL-inspired weighting mechanism used in IL, for integrating the shared knowledge in the plan generation; a set of tracking beliefs $\Theta$, and $\Psi$, the plan templates to define the structure of new plans. CIL results in new plans for performing the intended behaviour. CIL is tested in the D-R domain with the template to add sequences of actions.

IL, in the collective scope, has not been approached in the literature. Most of the works in the field of learning in MAS, are focused on other agent architectures different from BDI [8,73,128]. A closer endeavour to the work presented in this thesis, is the proposal to introduce planning in a BDI MAS, recently reported in [23], although it relies in an external planner. Consequently, CIL represents a novel contribution to learning in MAS, and particularly, to learning in BDI MAS.

The relevant features of the work presented here can be summarized as:

- The procedural knowledge of the agent is reused either for failure handling or to improve the agent behaviour. By plan acquisition the agent is capable of performing tasks not specified in design time.

- The use of RL techniques for action selection confers the agent the capability of learning through interacting with the environment, on this way, the agent can approximate the model of the world by encoding this knowledge in the set of plans and the capability of refine this procedural knowledge from new interactions. Despite the work presented here relies in basic Reinforcement Learning (RL) techniques for the selection criteria, other ranking methods can be used as long as they allow to establish a way of determining when an action is beneficial for an intended task.

- The learning mechanism is fully integrated in the BDI framework without the use of external planning tools, progressing in enhancing agents with plan acquisition (IL in the level of activities).

- The set of plan templates (representing the Manipulative Abduction component) can be adapted to the domain, through changing the organization of actions in a structurally different way, and incorporating domain knowledge when pursuing a different task.

- CIL aims to progress the learning in MAS by lifting IL in the level of plan acquisition from the single-agent to the collective scope. CIL takes into account in the learning process not only the learner agent but also other agents. Interactions of agents can be between homogeneous and heterogeneous partners. The communication method is based on direct inter-agent messaging, but other methods can be used.

- As defined by the framework of experimentation, the system exploits NetLogo as the tool to represent the environment where Jason agents are located, maintaining a clear delimitation between decision making and action in the environment, with decision-making driven by the BDI agent's reasoning cycle. This represents a novel contribution, aiming to bridge the gap between the integration of Agent-Based Modelling and the Multi-Agent Systems.

## 6.3 Application in Other Areas and Future Work

### 6.3.1 Other Possible Domains

The framework introduced in this work combines MAS and ABM approaches, adding up them with BDI agents enhanced with learning. It was tested in the vacuum-cleaner and D-R domains, nevertheless, it is feasible to be applied to other domains and areas where planning or learning is required. Those areas where the interaction with the environment is a key element could be considered as domains where this framework could be applied. Here are some examples of those areas of application:

- Computer game agents. These kind of domains impose the need of planning and learning in a dynamic environment frequently working with partners (human gamers or other agents) with time restrictions. Different planning systems have been adapted to games, for instance in [79] the HTN planning method (similar to the BDI reasoning) was used in a fighting game with good results. Additionally, in some games, agents are required to be more complex than the traditional reactive agents, so BDI based game agents are ideal to cope with such requirements. The learning frameworks IL/CIL presented here could be interconnected to games, adapting or interfacing the game environment as the external environment of the Jason BDI agents. A possible difficulty to this purpose is that the game should be compatible in terms of programming language and intercommunication, however, this is as a technical issue more than a theoretical one as the BDI reasoning/learning loop still would be in charge of the deliberation and decision-making, then gaming agents should behave accordingly.
- Supply Chain Management (SCM). Companies carry out several activities in a dynamic way: negotiating with suppliers to obtain materials for production, handling client orders, controlling inventories, managing manufacturing schedules and the delivering of finished products, providing services to the client, among others. All that while keeping the balance between requirements and the sparseness of resources. This is the purpose of the SCM [59], that have been approached as a planning problem where BDI agents has been introduced as a promising technology. SCM is a domain that can benefit from the work presented in this thesis due to the learning capability of BDI agents and the possibilities of simulation and representation of dynamic environments showed in previous chapters.

### 6.3.2 Limitations and Future Work

Some limitations of the framework presented on this thesis can be pointed out:

- As the applicability of plans generated by the learning templates with more than one action is affected (by the narrow space of feedback that the agent gets when the learning occurs), these learnt plans demand further refinements steps. In this regard, the current implementation is focused on demonstrating the plan acquisition possibilities but nothing prevents the integration with IL at other levels, namely learning in the level of goals and plan applicability [49,51,90]. Furthermore, a simple version of plan revision, based on the reported utility and the applicability of actions when the learning process is triggered was implemented, although a more robust analysis of the interactions with the environment could be incorporated. For instance, by adopting a more flexible action selection that consider other contexts apart from the current one. Additionally, in [30] the concept of "ideal plan" embodied as hybrid-plans (composed by primitive and other higher level plans) is formalised, it could be useful to improve the plan revision strategy presented in this thesis.

- To scale the approach for a more difficult environment or task, both the relevant set of plans and a set of applicable rewards to the aimed task must be provided, as they are specific for an intended behaviour. In the problems presented here, pre- and post-conditions of the actions where captured by tracking the state perceived by the agent before and after executing an action. Consequently, jointly with the use of learning templates, an important aspect for scaling (to reach behaviours with higher level of complexity) is the configuration of the primitives and their precedence that should be expressed accordingly with the intended task. In [31] an interesting approach to formalise the notion of pre-condition is presented, despite it lays on traditional planning, it could be considered as a future work for the analysis of pre- and post- conditions.
- Additionally, the re-learning stages are driven by measuring the adequacy of the plans when pursuing a task; in the left-handed wall-following behaviour, the negative utility indicates when a new learning process is needed. When changing the task, this criteria should be adjusted as required to fit the new conditions.

The current version of the BDI MAS in the D-R domain includes some key assumptions in the implementation that impose potential constraints to the system's functionality, described as follows:

- For the sake of realism, in this work agents can traverse the environment using the streets only instead of using a graph based representation of the routes as usual in D-R simulations. The agent's navigation strategy is based on a simple algorithm to avoid buildings based on the sensations from their frontal sight-line of 10 pixels in an angle of 90°. Then, sometimes agents cannot go around corners.

- Plan acquisition seems to be sensitive to the history of perceptions and the order of sharing knowledge. Accordingly to the weighting mechanism, observations gathered earlier get higher values. Then, sometimes newly acquired plans need more revision steps to fulfill their tasks.
- The task allocation method split the city map in four quadrants to be assigned to the same number of agent teams. Each team should be formed by heterogeneous agents from the MAS as they reply the calling messages to be part of it. Incoming requests are attended by those agents that reply the first, then some agents could remain without being called and may take not part in the rescue actions.
- Sharing of information is based on a simple contract net protocol: calling for team formation considers early agent's replies in the first place. Observations are requested to teammates that have replied at that time. Consequently, some agent could be requested more frequently than others preventing to have a wider opinion from others agents of the team. This could be improved by implementing a more complex method of team formation and intercommunication.
- Observations are stored as agent's beliefs, including the state of the environment as perceived by them, as first-order logic statements. Both the querying of observations and the sharing requests between agents could be performed based on logic unification of variables, or grounded atoms, or a mix of them, involving different amount of computing resources depending on their structure.

Finally, some avenues can be explored to improve the integration of NetLogo and Jason:

- Integrate a set of task allocation algorithms to be selected by the user and executed by the agents. As stated before, the current task allocations algorithm follows a fixed policy, distributing the agents accordingly with a predefined map partition adequate to the used map. Then, a more complex algorithm could be used for this purpose, for instance to be automated and be applied independently of the used map.
- Represent a large set of features for the environmental elements, such as fire propagation and strength, time alive for the victims, fuel decrease in the police and ambulance vehicles, among others. Elements, represented through reactive agents, should exhibit an independent behaviour, governed by their own

circumstances. Additionally, civilians could be represented as BDI agents, including a higher-level of realism to the simulation.

- Include a more interesting partition algorithm to divide the scenario, accounting for different aspects, namely, topography, demography and infrastructure information of the city.
- Define a set of metrics to measure the system performance as a whole and in the agent level. The adoption of rescue policies and emergency management protocols, could be included to the model to analyse their effectiveness in the simulated ground.

## 6.4 List of Publications

2018

1. **Bridging the Gap between ABM and MAS: A Disaster-Rescue Simulation Using Jason and NetLogo**. Luna Ramirez W.A., Fasli M. Computers 2018 7(2). Accepted: 2018/04/07. Published: 2018/04/11. `http://www.mdpi.com/2073-431X/7/2/24/html`

2017

2. **Plan Acquisition in a BDI Agent Framework Through Intentional Learning**.
   Luna Ramirez W.A., Fasli M. (2017) Plan Acquisition in a BDI Agent Framework Through Intentional Learning. In: Berndt J., Petta P., Unland R. (eds) Multiagent System Technologies. MATES 2017. Lecture Notes in Computer Science, vol 10413. Springer, Cham
   `https://rd.springer.com/chapter/10.1007/978-3-319-64798-2_11`
3. **Integrating NetLogo and Jason: a Disaster-Rescue Simulation**. Luna Ramirez W.A., Fasli M., Proceedings of the 9th Computer Science & Electronic Engineering Conference. Colchester. 2017/09/27. IEEE Explore, United Kingdom.
   `https://ieeexplore.ieee.org/document/8101627/`

2015

4. **Auto Organización, Emergencia y Sistemas MultiAgente**,

   Wulfrano Arturo Luna Ramírez, Komputer Sapiens. Pag 25-30. Vol 1, Ao 7, enero-abril 2015. ISSN 2007-0691. (Spanish)

   `http://www.komputersapiens.smia.mx/`

## 6.5 Code Availability

To conclude, there is a brief description of the implementations and extra directions on it in Appendix D. The code of the D-R Simulation without learning can be downloaded from:

`http://corban.cua.uam.mx/~wluna/JNLBDI-DR/index-en.php`

Additionally, the rest of the code is available on request by writing to: `wluna@correo.cua.uam.mx`

Personal page (Spanish):

`http://hermes.cua.uam.mx/informacion_academicos.php?academico=WulfranoArturoLunaRamirez`

# Bibliography

[1] FIPA. FIPAwphttp://fipa.org/.

[2] NetLogoBDI. NetLogoBDIwphttp://users.uom.gr/~iliass/projects/NetLogo/.

[3] Traffic Model. TrafficModelNLwphttp://modelingcommons.org/browse/one_model/3851#model_tabs_browse_info.

[4] Carole Adam and Benoit Gaudou. BDI agents in social simulations: a survey. *Knowledge Eng. Review*, 31(3):207–238, 2016.

[5] J. G. Adam. *Designing emergence : automatic extraction of stigmergic algorithms from lattice structures.* PhD thesis, Dept. of Computer Science and Electronic Engineering University of Essex, 2006.

[6] Stéphane Airiau, Lin Padgham, Sebastian Sardiña, and Sandip Sen. Enhancing the Adaptation of BDI Agents Using Learning Techniques. *IJATS*, 1(2):1–18, 2009.

[7] Ebrahim Al-Hashel, Bala M Balachandran, and Dharmendra Sharma. A comparison of three agent-oriented software development methodologies: ROADMAP, prometheus, and MaSE. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 909–916. Springer, 2007.

[8] Eduardo Alonso, Mark D'inverno, Daniel Kudenko, Michael Luck, and Jason Noble. Learning in multi-agent systems. *The Knowledge Engineering Review*, 16:277–284, 9 2001.

[9] Sangeeta Arora, CP Agrawal, P Sasikala, and Arun Sharma. Developmental approaches for Agent Oriented system: A critical review. In *Software Engineering (CONSEG), 2012 CSI Sixth International Conference on*, pages 1–5. IEEE, 2012.

[10] R. Babuška, L. Buşoniu, and B. De Schutter. Reinforcement learning for multi-agent systems. Technical Report 06-041, Delft Center for Systems and Control, Delft University of Technology, July 2006. Paper for a keynote presentation at the *11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2006),* Prague, Czech Republic, Sept. 2006.

[11] A. Badica, C. Badica, M. Ivanovic, and D. Mitrovic. An Approach of Temporal Difference Learning Using Agent-Oriented Programming. In *2015 20th International Conference on Control Systems and Computer Science*, pages 735–742, May 2015.

[12] Tina Balke and Nigel Gilbert. How Do Agents Make Decisions? A Survey. *Journal of Artificial Societies and Social Simulation*, 17(4):13, 2014.

[13] Levi Barton and Vicki H Allan. Adapting to changing resource requirements for coalition formation in self-Organized social networks. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, volume 2, pages 282–285. IEEE, 2008.

[14] Bernhard Bauer and Jörg P Müller. Using uml in the context of agent-oriented software engineering: State of the art. In *International Workshop on Agent-Oriented Software Engineering*, pages 1–24. Springer, 2003.

[15] Olivier Boissier, Rafael H. Bordini, Jomi F. Hubner, Alessandro Ricci, and Andrea Santi. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, 78(6):747 – 761, 2013.

[16] 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.

[17] Rafael H. Bordini and Jomi F. Hübner. Jason. Java-based AgentSpeak interpreter used with SACI for multi-agent distribution over the net. `http://jason.sourceforge.net/wp/description/`, 2005.

[18] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology)*. John Wiley &#38; Sons, Inc., USA, 2007.

[19] RafaelH. Bordini and AlvaroF. Moreira. Proving BDI Properties of Agent-Oriented Programming Languages: The asymmetry thesis principles in AgentSpeak(L). *Annals of Mathematics and Artificial Intelligence*, 42(1-3):197–226, 2004.

[20] Michael E. Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, November 1987.

[21] L. Buşoniu, R. Babuška, and B. De Schutter. A comprehensive survey of multi-agent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38(2):156–172, March 2008.

[22] L. Buşoniu, R. Babuška, and B. De Schutter. Multi-agent reinforcement learning: An overview. In D. Srinivasan and L.C. Jain, editors, *Innovations in Multi-Agent Systems and Applications – 1*, volume 310 of *Studies in Computational Intelligence*, chapter 7, pages 183–221. Springer, Berlin, Germany, 2010.

[23] Rafael CARDOSO and Rafael BORDINI. A multi-agent extension of a hierarchical task network planning formalism. *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, 6(2), 2017.

[24] Olivia Cindy, Chee-Fon Chang, Carlos F. Enguix, and Aditya K. Ghose. Case-Based BDI agents: An Effective Approach to Intelligent Search on the WWW. In *Intelligent Agents in Cyberspace, AAAI Spring Symposium*. Stanford University, 1999.

[25] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998:746–752, 1998.

[26] Juan M. Corchado, Juan Pavon, Emilio S. Corchado, and Luis F. Castillo. Development of CBR-BDI Agents: A Tourist Guide Application. In Peter Funk and Pedro A. Gonzalez Calero, editors, *Advances in Case-Based Reasoning*, volume 3155 of *Lecture Notes in Computer Science*, pages 547–559. Springer Berlin Heidelberg, 2004.

[27] Khanh Hoa Dam and Michael Winikoff. Comparing Agent-Oriented Methodologies. In Paolo Giorgini, Brian Henderson-Sellers, and Michael Winikoff, editors, *Agent-Oriented Information Systems*, volume 3030 of *Lecture Notes in Computer Science*, pages 78–93. Springer Berlin Heidelberg, 2004.

[28] Mehdi Dastani. A Survey of Multi-agent Programming Languages and Frameworks. In Onn Shehory and Arnon Sturm, editors, *Agent-Oriented Software Engineering*, pages 213–233. Springer Berlin Heidelberg, 2014.

[29] Lavindra de Silva and Lin Padgham. A comparison of bdi based real-time reasoning and htn based planning. In Geoffrey I. Webb and Xinghuo Yu, editors, *AI 2004: Advances in Artificial Intelligence*, pages 1167–1173, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[30] Lavindra de Silva, Lin Padgham, and Sebastian Sardina. Htn-like solutions for classical planning problems: An application to bdi agent systems. *Theoretical Computer Science*, 763:12 – 37, 2019.

[31] Lavindra de Silva, Sebastian Sardiña, and Lin Padgham. Addendum to: Summary information for reasoning about hierarchical plans. *CoRR*, abs/1708.03019, 2017.

[32] Tom De Wolf and Tom Holvoet. Emergence versus self-organisation: Different concepts but promising when combined. In Sven A. Brueckner, Giovanna Di Marzo Serugendo, Anthony Karageorgos, and Radhika Nagpal, editors, *Engineering Self-Organising Systems*, volume 3464 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2005.

[33] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos. Self-organising systems. In Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos, editors, *Self-organising Software*, Natural Computing Series, pages 7–32. Springer Berlin Heidelberg, 2011.

[34] Marie-Pierre Gleizes Di Marzo Serugendo, Giovanna and Anthony Karageorgos. Self-organisation and emergence in mas: An overview. *Informatica*, 30(1):45–54, 2006.

[35] Jürgen Dix, Héctor Muñoz-Avila, Dana S. Nau, and Lingling Zhang. Impacting shop: Putting an ai planner into a multi-agent environment. *Annals of Mathematics and Artificial Intelligence*, 37(4):381–407, Apr 2003.

[36] Jim E Doran, SRJN Franklin, Nicholas R Jennings, and Timothy J Norman. On cooperation in multi-agent systems. *The Knowledge Engineering Review*, 12(3):309–314, 1997.

[37] Edmund H. Durfee. *Distributed Problem Solving and Planning*, pages 118–149. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[38] Maria Fasli. *Agent technology for e-commerce*. John Wiley & Sons Chichester, 2007.

[39] Carolina Felicíssimo, Caroline Chopinaud, Jean-Pierre Briot, Amal El Fallah Seghrouchni, and Carlos Lucena. Contextualizing normative open multi-agent systems. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 52–59. ACM, 2008.

[40] EricR. Frykberg. Disaster and Mass Casualty Management. In L.D. Britt, DonaldD. Trunkey, and DavidV. Feliciano, editors, *Acute Care Surgery*, pages 229–248. Springer New York, 2007.

[41] Carlos Gershenson. A General Methodology for Designing Self-Organizing Systems. *CoRR*, abs/n-lin/0505009, 2005.

[42] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann, Burlington, 2004.

[43] Nigel Gilbert. *Agent-Based Models (Quantitative Applications in the Social Sciences)*. SAGE Publications, Inc, annotated edition edition, September 2007.

[44] Jeffrey Goldstein. Complexity and philosophy: Re-imagining emergence: Part 1. *Emergence: Complexity and Organization*, 15(2):78, 2013.

[45] Jeffrey A Goldstein. Reimagining emergence, Part 3: Uncomputability, transformation, and self-transcending constructions. *Emergence: Complexity and Organization*, 16(2):116, 2014.

[46] Alma M Gómez-Rodríguez and Juan C González-Moreno. Comparing agile processes for agent oriented software engineering. In M. Vierimaa M. Ali Babar and M. Oivo, editors, *International Conference on Product Focused Software Process Improvement*, volume 6156 of *Lecture Notes in Computer Science*, pages 206–219. Springer Berlin Heidelberg, 2010.

[47] V. I. Gorodetskii. Self-organization and multiagent systems: I. models of multiagent self-organization. *Journal of Computer and Systems Sciences International*, 51(2):256–281, Apr 2012.

[48] Alejandro Guerra-Hernandez, Amal El Fallah-Seghrouchni, and Henry Soldano. Learning in BDI Multi-agent Systems. In Jurgen Dix and Joao Leite, editors, *Computational Logic in Multi-Agent Systems*, volume 3259 of *Lecture Notes in Computer Science*, pages 218–233. Springer Berlin Heidelberg, 2005.

[49] Alejandro Guerra-Hernández, Carlos Alberto González-Alarcón, and Amal El Fallah-Seghrouchni. Jason Induction of Logical Decision Trees: A Learning Library and Its Application to Commitment. In Grigori Sidorov, Arturo Hern?ndez Aguirre, and CarlosAlberto Reyes García, editors, *Advances in Artificial Intelligence*, volume 6437 of *Lecture Notes in Computer Science*, pages 374–385. Springer Berlin Heidelberg, 2010.

[50] Alejandro Guerra-Hernandez and Gustavo Ortiz-Hernandez. Toward BDI Sapient Agents: Learning Intentionally. In ReneV. Mayorga and LeonidI. Perlovsky, editors, *Toward Artificial Sapience*, pages 77–91. Springer London, 2008.

[51] Alejandro Guerra-Hernández, Gustavo Ortiz-Hernández, and Wulfrano Arturo Luna-Ramírez. Jason Smiles: Incremental BDI MAS Learning. In *2007 Sixth Mexican International Conference on Artificial Intelligence, Special Session (MICAI)*, pages 61–70, Nov 2007.

[52] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. Coordinated Reinforcement Learning. In *In Proceedings of the ICML-2002 The Nineteenth International Conference on Machine Learning*, pages 227–234, 2002.

[53] Jianye Hao, Dongping Huang, Yi Cai, and Ho fung Leung. The dynamics of reinforcement social learning in networked cooperative multiagent systems. *Engineering Applications of Artificial Intelligence*, 58:111 – 122, 2017.

[54] Jianye Hao, Ho-Fung Leung, and Zhong Ming. Multiagent reinforcement social learning toward coordination in cooperative multiagent systems. *ACM Trans. Auton. Adapt. Syst.*, 9(4):20:1–20:20, December 2014.

[55] Glenn I Hawe, Graham Coates, Duncan T Wilson, and Roger S Crouch. Agent-based simulation for large-scale emergency response: A survey of usage and implementation. *ACM Computing Surveys (CSUR)*, 45(1):8, 2012.

[56] Alejandro Guerra Hernandez. *Learning in intentional BDI Multiagent Systems*. PhD thesis, Université de Paris 13. Institut Galilée. LIPN – Laboratoire d'Informatique de Paris Nord, Paris Villetaneuse, France, 2003.

[57] Carlos Iglesias, Mercedes Garijo, and José Centeno-González. A survey of agent-oriented methodologies. In *Intelligent Agents V: Agents Theories, Architectures, and Languages*, volume 98, pages 317–330. Springer, 1998.

[58] Toru Ishida, Les Gasser, and Makoto Yokoo. Organization self-design of distributed production systems. *IEEE Transactions on Knowledge and Data Engineering*, 4(2):123–134, 1992.

[59] C. R. Jaimez-Gonzlez and W. A. Luna-Ramrez. An agent-based architecture for supply chain management. In *2013 IEEE Eleventh International Symposium on Autonomous Decentralized Systems (ISADS)*, pages 1–6, March 2013.

[60] S. Javdani, S. S. Srinivasa, and J. A. Bagnell. Shared autonomy via hindsight optimization. *ArXiv e-prints*, mar 2015.

[61] Nicholas R. Jennings and Michael Wooldridge. Agent-oriented software engineering. volume 117, pages 277–296. 2000.

[62] NicholasR. Jennings. Agent-oriented software engineering. In Ibrahim Imam, Yves Kodratoff, Ayman El-Dessouki, and Moonis Ali, editors, *Multiple Approaches to Intelligent Systems*, volume 1611 of *Lecture Notes in Computer Science*, pages 4–10. Springer Berlin Heidelberg, 1999.

[63] Yubo Jia, Chengwei Huang, and Hao Cai. A comparison of three agent-oriented software development methodologies: Mase, gaia, and tropos. In *Information, Computing and Telecommunication, 2009. YC-ICT'09. IEEE Youth Conference on*, pages 106–109. IEEE, 2009.

[64] S Kapetanakis and D Kudenko. Reinforcement learning of coordination in cooperative multi-agent systems. pages 326–331, 2002.

[65] Igor Kiselev and Reda Alhajj. An Adaptive Multi-agent System for Continuous Learning of Streaming Data. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '08, pages 148–153, Washington, DC, USA, 2008. IEEE Computer Society.

[66] Hiroaki Kitano, Satoshi Tadokoro, Itsuki Noda, Hitoshi Matsubara, Tomoichi Takahashi, Atsuhi Shinjou, and Susumu Shimada. Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS*, pages 739–746. IEEE Computer Society, 1999.

[67] Kalliopi Kravari and Nick Bassiliades. A Survey of Agent Platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11, 2015.

[68] Marián Lekavý and Pavol Návrat. Expressivity of strips-like and htn-like planning. In Ngoc Thanh Nguyen, Adam Grzech, Robert J. Howlett, and Lakhmi C. Jain, editors, *Agent and Multi-Agent Systems: Technologies and Applications*, pages 121–130, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[69] Sylvain Lemouzy, Valérie Camps, and Pierre Glize. Towards a self-organising mechanism for learning adaptive decision-making rules. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and International Conference on Intelligent Agent Technology - Workshops, 9-12 December 2008, Sydney, NSW, Australia*, pages 616–620. IEEE Computer Society, 2008.

[70] Oscar Javier Romero López. Self-organized and evolvable cognitive architecture for intelligent agents and multi-agent systems. In *European Conference on the Applications of Evolutionary Computation*, pages 392–401. Springer, 2010.

[71] Lorenzo Magnani. *Theoretical and Manipulative Abduction*, pages 1–61. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[72] Jean-Pierre Mano, Christine Bourjot, Gabriel Lopardo, and Pierre Glize. Bio-inspired mechanisms for artificial self-organised systems. *Informatica*, 30(1):55–62, 2006.

[73] David Martínez, Guillem Alenyà, Carme Torras, Tony Ribeiro, and Katsumi Inoue. Learning Relational Dynamics of Stochastic Domains for Planning. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016.*, pages 235–243, 2016.

[74] Felipe Meneguzzi and Lavindra De Silva. Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning. *The Knowledge Engineering Review*, 30(1):1–44, 2015.

[75] Felipe Meneguzzi and Michael Luck. Declarative Planning in Procedural Agent Architectures. *Expert Syst. Appl.*, 40(16):6508–6520, November 2013.

[76] Felipe Meneguzzi and Michael Luck. Declarative planning in procedural agent architectures. *Expert Systems with Applications*, 40(16):6508–6520, 2013.

[77] Thomas Mitchell. *Machine Learning (Mcgraw-Hill International Edit)*. McGraw Hill Higher Education, 1st edition, October 1997.

[78] Goncalo Neto. *From Single-Agent to Multi-Agent Reinforcement Learning: Foundational Concepts and Methods*, 2005.

[79] X. Neufeld, S. Mostaghim, and D. Perez-Liebana. HTN fighter: Planning in a highly-dynamic game. In *2017 9th Computer Science and Electronic Engineering (CEEC)*, pages 189–194, Sept 2017.

[80] Andrei Olaru and Adina Magda Florea. Emergence in cognitive multi-agent systems. In *CSCS17, the 17th International Conference on Control Systems and Computer Science, MASTS Workshop*, volume 2, pages 515–522, 2009.

[81] Andrei Olaru, Cristian Gratie, and AdinaMagda Florea. Emergent Properties for Data Distribution in a Cognitive MAS. In GeorgeAngelos Papadopoulos and Costin Badica, editors, *Intelligent Distributed Computing III*, volume 237 of *Studies in Computational Intelligence*, pages 151–159. Springer Berlin Heidelberg, 2009.

[82] Gustavo Ortiz-Hernandez. Aprendizaje Incremental en Sistemas Multi-Agente BDI. Master's thesis, Departamento de Inteligencia Artificial, Fac. de Fisica e Inteligencia Artificial, Universidad Veracruzana, Mexico, Sep 2007.

[83] L. Padgham, D. Singh, and F. Zambetta. Social simulation for analysis, interaction, training and community awareness. In *2015 Winter Simulation Conference (WSC)*, pages 3130–3131, Dec 2015.

[84] Lin Padgham and John Thangarajah. The Prometheus Design Tool (PDT). `https://sites.google.com/site/rmitagents/software/prometheusPDT`, 2005. Is a tool that supports the Prometheus methodology for building agent based systems. PDT is developed and supported by the RMIT Intelligent Agents Group.

[85] Liviu Panait and Sean Luke. Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[86] Massimo Paolucci, Onn Shehory, Katia Sycara, Dirk Kalp, and Anandeep Pannu. A planning component for retsina agents. In Nicholas R. Jennings and Yves Lespérance, editors, *Intelligent Agents VI. Agent Theories, Architectures, and Languages*, pages 147–161, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[87] Massimo Paolucci, Onn Shehory, and Katia P. Sycara. Interleaving planning and execution in a multiagent team planning environment. *Electron. Trans. Artif. Intell.*, 4(A):23–43, 2000.

[88] MA Pellicer and JM Corchado. Development of CBR-BDI agents. *International Journal of Computer Science and Applications*, 2(1):25–32, 2005.

[89] André H Pereira, Luis G Nardin, Anarosa AF Brandao, and Jaime S Sichman. LTI agent rescue team: a BDI-based approach for Robocup Rescue. *Proceedings of RoboCup*, 2011.

[90] Toan Phung, Michael Winikoff, and Lin Padgham. Learning Within the BDI Framework: An Empirical Analysis. In Rajiv Khosla, RobertJ. Howlett, and LakhmiC. Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 3683 of *Lecture Notes in Computer Science*, pages 282–288. Springer Berlin Heidelberg, 2005.

[91] J. Quinqueton. Emergence in problem solving, classification and machine learning. In *2006 Eighth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 5–92, Sept 2006.

[92] Steven F. Railsback and Volker Grimm. *Agent-Based and Individual-Based Modeling: A Practical Introduction*. Princeton University Press, 2012.

[93] Anand S. Rao and Michael P. Georgeff. BDI Agents: From Theory to Practice. In *In Proceedings of The First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 312–319. AAAI, 1995.

[94] Stuart J Russell and Peter Norvig. *Inteligencia Artificial: un enfoque moderno*. Colección de Inteligencia Artificial de Prentice Hall. Prentice Hall Hispanoamericana, 1996.

[95] Ilias Sakellariou, Petros Kefalas, and Ioanna Stamatopoulou. Enhancing NetLogo to Simulate BDI Communicating Agents. In John Darzentas, GeorgeA. Vouros, Spyros Vosinakis, and Argyris Arnellos, editors, *Artificial Intelligence: Theories, Models and Applications*, volume 5138 of *Lecture Notes in Computer Science*, pages 263–275. Springer Berlin Heidelberg, 2008.

[96] Ilias Sakellariou, Petros Kefalas, and Ioanna Stamatopoulou. Teaching Intelligent Agents using NetLogo. In *Proceedings of the ACM-IFIP Informatics Education Europe III Conference, IEEIII Venice, Italy*, 2008.

[97] Mauricio Salgado and Nigel Gilbert. Agent Based Modelling. In Timothy Teo, editor, *Handbook of Quantitative Methods for Educational Research*, pages 247–265. SensePublishers, 2013.

[98] Scott Sanner. Relational dynamic influence diagram language (rddl): Language description. *Unpublished ms. Australian National University*, 2010.

[99] Sebastian Sardina, Lavindra de Silva, and Lin Padgham. Hierarchical planning in bdi agent programming languages: A formal approach. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 1001–1008, New York, NY, USA, 2006. ACM.

[100] Sebastian Sardina and Lin Padgham. A BDI agent programming language with failure handling, declarative goals, and planning. *Autonomous Agents and Multi-Agent Systems*, 23(1):18–70, 2011.

[101] Martijn Schut, Michael Wooldridge, and Simon Parsons. On Partially Observable MDPs and BDI Models. In Mark d'Inverno, Michael Luck, Michael Fisher, and Chris Preist, editors, *Foundations and Applications of Multi-Agent Systems*, pages 243–259, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[102] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Anthony Karageorgos. Self-organization in multi-agent systems. *Knowledge Engineering Review*, 20:165–189, 2005.

[103] Y. Shoham, R. Powers, and T. Grenager. If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365–377, 2007.

[104] Harith Siddhartha, Rahul Sarika, and Kamal Karlapalem. Retrospective analysis of RoboCup rescue simulation agent teams. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 1365–1366. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[105] Gerardo I. Simari and Simon Parsons. On the Relationship Between MDPs and the BDI Architecture. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, pages 1041–1048, New York, NY, USA, 2006. ACM.

[106] Dhirendra Singh and Lin Padgham. Community evacuation planning for bushfires using agent-based simulation: Demonstration. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, pages 1903–1904, 2015.

[107] Dhirendra Singh and Lin Padgham. Emergency evacuation simulator (ees) - a tool for planning community evacuations in australia. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 5249–5251, 2017.

[108] Dhirendra Singh, Lin Padgham, and Brian Logan. Integrating bdi agents with agent-based simulation platforms. *Autonomous Agents and Multi-Agent Systems*, 30(6):1050–1071, Nov 2016.

[109] Dhirendra Singh, Sebastian Sardina, and Lin Padgham. Extending bdi plan selection to incorporate learning from experience. *Robotics and Autonomous Systems*, 58(9):1067 – 1075, 2010. Hybrid Control for Autonomous Systems.

[110] Dhirendra Singh, Sebastian Sardina, Lin Padgham, and Stéphane Airiau. Learning context conditions for BDI plan selection. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 325–332. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[111] Dhirendra Singh, Sebastian Sardina, Lin Padgham, and Geoff James. Integrating Learning into a BDI Agent for Environments with Changing Dynamics. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, pages 2525–2530. AAAI Press, 2011.

[112] Cameron Skinner and Sarvapali Ramchurn. The robocup rescue simulation platform. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 1647–1648. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[113] Jolanda van Steenbergen. Emergence in Multi-Agent Systems. Master's thesis, Faculcty of Science, utrecht Unversity, 2012.

[114] Peter Stone and Manuela Veloso. Multiagent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[115] Budhitama Subagdja, Iyad Rahwan, and Liz Sonenberg. Learning as Abductive Deliberations. In Qiang Yang and Geoff Webb, editors, *PRICAI 2006: Trends in Artificial Intelligence*, volume 4099 of *Lecture Notes in Computer Science*, pages 11–20. Springer Berlin Heidelberg, 2006.

[116] Budhitama Subagdja and Liz Sonenberg. Learning Plans with Patterns of Actions in Bounded-Rational Agents. In Rajiv Khosla, RobertJ. Howlett, and LakhmiC. Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems*, volume 3683 of *Lecture Notes in Computer Science*, pages 30–36. Springer Berlin Heidelberg, 2005.

[117] Budhitama Subagdja, Liz Sonenberg, and Iyad Rahwan. Intentional learning agent architecture. *Autonomous Agents and Multi-Agent Systems*, 18(3):417–470, 2009.

[118] Budhitama Subagdja and Ah-Hwee Tan. iFALCON: A neural architecture for hierarchical planning. *Neurocomputing*, 86:124–139, 2012.

[119] Gundeep Tanwar Naveen Alhawat Surjeet Dalal. Designing CBR-BDI Agent for implementing Supply Chain system, January-February 2013.

[120] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[121] Ah-Hwee Tan and Dan Xiao. Self-organizing cognitive agents and reinforcement learning in multi-agent environment. In *Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, pages 351–357, Sept 2005.

[122] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *In Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337. Morgan Kaufmann, 1993.

[123] Yuqing Tang, Felipe Meneguzzi, Katia Sycara, and Simon Parsons. Planning over MDPs through HTNs. 2011.

[124] Huaglory Tianfield and Rainer Unland. Towards self-organization in multi-agent systems and grid computing. *Multiagent and Grid Systems*, 1(2):89–95, 2005.

[125] Paulo Trigo and Helder Coelho. Decision making with hybrid models: the case of collective and individual motivations. *IJRIS*, 2(1):60–72, 2010.

[126] Karl Tuyls and Gerhard Weiss. Multiagent Learning: Basics, Challenges, and Prospects. *AI Magazine*, 33(3):41–52, 2012.

[127] M. Vallee. Towards Analysing and Controlling Self-Organising Systems with Socio-economic Principles. In *2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshop*, pages 67–71, Sept 2010.

[128] H Van Dyke Parunak, Paul Nielsen, Sven Brueckner, and Rafael Alonso. Hybrid multi-agent systems: Integrating swarming and BDI agents. *Engineering Self-Organising Systems*, pages 1–14, 2007.

[129] Andrzej Walczak, Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. Augmenting bdi agents with deliberative planning techniques. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal

El Fallah Seghrouchni, editors, *Programming Multi-Agent Systems*, pages 113–127, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[130] Wenwen Wang, Budhitama Subagdja, Ah-Hwee Tan, and Yuan-Sin Tan. A self-organizing multi-memory system for autonomous agents. In *The 2012 International Joint Conference on Neural Networks (IJCNN), Brisbane, Australia, June 10-15, 2012*, pages 1–8. IEEE, 2012.

[131] Stefan Warwas. The Bochica Framework for Model-Driven Agent-Oriented Software Engineering. In Joaquim Filipe and Ana Fred, editors, *Agents and Artificial Intelligence*, volume 358 of *Communications in Computer and Information Science*, pages 158–172. Springer Berlin Heidelberg, 2013.

[132] U. Wilensky. NetLogo, 1999. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

[133] Michael Winikoff and Stephen Cranefield. On the testability of bdi agent systems. *J. Artif. Int. Res.*, 51(1):71–131, September 2014.

[134] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[135] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 6 1995.

[136] Michael Wooldridge and NicholasR. Jennings. Agent theories, architectures, and languages: A survey. In MichaelJ. Wooldridge and NicholasR. Jennings, editors, *Intelligent Agents*, volume 890 of *Lecture Notes in Computer Science*, pages 1–39. Springer Berlin Heidelberg, 1995.

[137] Sungwook Yoon, Alan Fern, Robert Givan, and Subbarao Kambhampati. *Probabilistic planning via determinization in hindsight*, volume 2, pages 1010–1016. 2008.

# Appendix A

# Methodologies of Multi-Agent Systems Development

Traditional software engineering techniques are insufficient either for the Multi-Agent Systems (MAS) development and the development of MAS taking into account the concepts of Self-organisation (SO) & Emergence (EM). For example, the structured paradigm and Object Oriented (OO) paradigm does not suffices, the reason is it only makes it possible to define a global behaviour when it is a function of the behaviour of the various parts [9, 33, 136], due the diversity and complexity of relations an agent is more than an active object, and agent can decide if an incoming request can be attended or not, and the mental states cannot be specified properly with the OO tools and abstractions, nor the Component Oriented Paradigm and its independents objects [9]. So, a different way to "divide and conquer"must be thought and more powerful abstraction tools must be developed to cope with specific issues of agents like social abilities (interaction/communication protocols and policies), complex behaviours (weak and strong notion of agent) and concurrency/distribution (a central point in MAS). Additionally, some concerns must be addressed when engineering EM and SO in MAS: the so called traditional design of MAS (based in coordination, service description or ontologies) relies on pre-programmed interactions, so the process of SO/EM cannot freely exit as they can adapt to unexpected environmental changes.

On the other hand, currently design of MAS directly address SO and includes the design of distributed

algorithms like those bio/socially-inspired, and some methodologies has been proposed, additionally, middleware technology non naturally-inspired has given good results. However, a lack of robust verification mechanism and whole engineering methods persists yet
citeDiMarzo2011SOS. There is under research the definition of methods to control the emergence window (mechanisms to control it and avoid as possible or minimize the undesired emergents) i.e. to conform a space of desired EM, to close the space of possibilities for EM occurs. Furthermore, it is needed to find methods to predict the convergence of systems to a desired state in order to guide the design of systems. Environment must be considered as a coordination mechanism and a source of changes and adaptation but without to fall in a direct and/or centralized control. In conclusion, it can be said that the research effort still is addressed to develop principles, theories, models, mechanisms, and methodologies to develop MAS with exhibit SO, EM or both of them called neo-computation with many applications such as autonomic, pervasive and ubiquitous computing [33]. In the following point is presented a brief review of some methodologies that has performed successfully in the development of MAS that can exhibit SO/EM.

Taking this into account, in order to develop a MAS for modelling the system described in the Domain of Application, some methodologies of MAS development has been reviewed as showed in the next point.

## A.1 Software Engineering Methodologies of MAS Development

Methodologies used to the development of MAS can be arranged in two classes. On the first hand, the reviewed methodologies belong to the, lets say, general application approach, since they are focused in the development cycle of MAS according to Software Engineering.

On the other hand, there are just a few methodologies that include the SO and/or EM concepts into their cycle of development, furthermore, the methods to control such emergence are still under research and development.

Both of them are reviewed in the following.

### A.1.1 Developing MAS

It is important to notice, as has been established before, that the capability of problem solving of an individual agent is limited by its capability of information processing, i.e. perceptions (processing external/environmental information) and its own scope of abilities or actions (changes or responses to the environment). Such capability is limited and it is not enough to cope with complex problems, moreover, when the distributed nature of the problem requires a distributed response, a simple agent is not a feasible solution. In this manner, a community of agents (a MAS), can be profiled as a more suitable solution to complex problems and many applications has been developed under this approach.

As a result of the multiplicity of applications of MAS in a several fields, a lot of software development processes that introduce the agent perspective have been developed, and many methodologies and approaches has been proposed and they emphasize different aspects like functions or goals orientation and use a variety of notations and graphical representations like the Agent Oriented extension to the Unified Modelling Language (AUML), but they are not enough mature to large scale commercial software development and is still missing a standard agent architectures and agent programming languages, as a consequence, there is not such as standard code generation from models [7, 27, 46, 57]. There are two essential components of Agent Oriented software development methodologies and processes: the available frameworks (the tools and development environment which could imply the automatic code generation given a specification of agents) and the methodologies and development processes by means of which the system construction is guided [46]. It is worth pointing out the difference between software development process and methodology. The former can be seen as the set the following components: roles or workers and their related activities; work products derived from the activities; and work definitions, that is the particularities of the works to carry on. On the other hand, a methodology defines the models to build, the concepts, and the used notation to describe them [46].

Some of the developed methodologies are INGENIAS, MESSAGE, TROPOS, MASE, PROMETHEUS, PASSI, ROADMAP, MaSE, ADELFE, among others [7, 27, 33, 46, 57]. Given that diversity of methodologies and approaches, some studies has been carried out to analyse their similarities, strongnesses and weaknesses, although some criteria has been proposed in order to compare them stressing different aspects of them, but there is not an general agree to perform comparisons. The methodologies can be divided in three categories [7, 57]:

1. Extensions or adaptations of OO methodologies: it has the main advantage of similarities between Agent Oriented and OO paradigms, the more used OO tools and methodologies within the development community, and the used OO languages in the MAS development. The MAS can be specified through thee perspectives: static view (objects structure and structural relationships); dynamic view (objects interactions); and functional (data flow of objects methods). Agile methods like OpenUp and RUP are commonly the most used to adapt the AO software process development. Finally, despite the lack to specify decision taking (execution of task and solving contradictory goals), the graphic representations such as uses cases and classes responsibilities collaborations cards can be used to specify MAS.

2. Adaptation of knowledge engineering models or other techniques: Knowledge engineering. Used to develop knowledge based systems that could be employed to model agent knowledge as knowledge acquisition process, including modelling aspects and reuse. Unfortunately this is the only aspect addressed. Moreover, despite the existent tools can be used, they are not enough extendable and conceive the solution based in a centralised system ridding of the social aspects and the autonomy and goal oriented behaviour of agents. So, this extensions are not suitable to MAS engineering with SO and EM. Formal methods. Allow the complete specification of the systems and have different levels of abstractions, but lack of tools to specify agent interactions, and are not easily scalable in practice.

3. Agent-oriented methodologies (AO): includes agent concepts (mental states, tasks, communication/interaction and group modelling) into the software development process, extending the available artefacts, models and tools in software engineering community and proposing new ones. Some of them are good to cope with the analysis & design stages and support the entire specification of MAS, validation against user requirements and evolutions stages. Most of them have been developed thinking in a domain of application, they are not general methodologies, though they can be applied in a variety of problems when MAS are the most suitable solution. Moreover, some of them can be applied no matter the agent theory, agent architecture or agent language used.

A brief summary of some methodologies is showed in Table A.1, according to the information derived from case of studies and comparison between them as presented as the most representatives AO in [7,14,15, 27,33,46,57,63,131].

| **Methodology & Process Description** | **Tools/Frameworks & Agent concept** |
|---|---|
| Agent Modelling Technique for Systems of BDI Agents The MAS is conceived from tow point of view: 1) external (agent decomposition of the system, complex objects are characterized in terms of purpose, responsibilities, services, information and interactions); and internal (agent architecture, beliefs, goals and plans). Interactions are characterized as speech acts and in terms of the information they carry on and are collected in the agent instance model. | The methodology comprises several models: Agent class and instance, interaction, belief, goal and plan. They are based in UML diagrams. State Chart Diagrams are used to model plans. Agent concept: BDI agent. Application domain suggested: Intelligent and BDI MAS. |

| | |
|---|---|
| GAIA It affords the macro and micro levels of MAS. It ranges from analysis of requirements to a detailed design. It has two stages with some steps within: 1) Analysis (finding the system roles and their attributes: responsibilities, permissions, activities and protocols; modelling the interaction between roles); 2) Design (mapping roles into agents types; determining the services model needed to fulfil a role in agents; creating the acquaintance model for the representation of agent communications). Unfortunately, does not approach the requirements gathering phase neither and integrated model of execution environment. It is not an iterative methodology, and lacks of an explicit modelling of social tasks/laws and goals. Finally, system description is flat, so it cannot afford complexity, it has not the typical hierarchy of MAS. It is recommended to create new software reengineering and designing systems with reuse components. | It has formal operators and templates in order to represent roles and their attributes; additionally it has schemas used to represent interactions. The implementations phase is not well supported Agent concept: heterogeneous, social agents. Application domain suggested: small and closed systems. Coarse-grained computational systems. |

| | |
|---|---|
| Role oriented Analysis and Design for Multi-Agent Programming (ROADMAP) Is extension of Gaia methodology, includes specification in the analysis phase. It improves Gaia adding: formal models of knowledge and environment, role hierarchies, explicit representation of social structures and relationships and incorporates dynamics changes. Each role (described by means of schemas) has a knowledge component to cope with environmental changes. It has a limited but easy and clear notation; it is easy of use and it is adaptable scalable and traceable. While allows abstract and high-level the architectural design, it has not a detailed design phase. | It has use case models; environment and knowledge models; goal, role and revised role models and a social model. The environment model derives from use cases and has a hierarchy of zones with these attributes: static objects, objects (entities) and sources of uncertainty. It has an activity protocol implemented from the assigned role for services of an agent. Roadmap Editor Built for Easy Development (REBEL) is its main analysis stages tool. Agent concept: knowledge schema based agents with medium level of autonomy. Application domain suggested: coarse-grained computational, complex and open systems. |

| | |
|---|---|
| Multi-Agent Systems Engineering (MaSE) It has a complete life-cycle development methodology mainly focused in analysis and design phases. System specification has a medium degree of detail whereas analysis and architectural design are good detailed. Its notation is clear and easy of learning and use, and it is adaptable but not easily scalable. The interaction is limited to one agent to other one. Its process includes seven steps within two phases: a) Analysis (1. Capturing goals that produces a goal hierarchy; 2. Applying use cases and build sequence diagrams; 3. Role model and concurrent task model construction); b) Design (4. Creating agent classes that produces the agent class diagrams which describe the entire MAS; 5. Constructing conversations with communications class diagrams; 6. Assembling agent classes; and 7. System design which produces a Deployment diagram). It has good support of concurrency. | Some used notations are: case uses and roles; agent classes, interaction and deployment diagrams (where the final result of the analysis and design stages is presented), finite machine automata. Its graphic interactive engineering tool agentTool allows the translation of models into design artefacts. Agent concept: heterogeneous agents. Application domain suggested: heterogeneous MAS. |

| | |
|---|---|
| PROMETHEUS It has a process for specifying, designing and implementing MAS. Define its process through several deliverables, its emphasis is on the interfaces compatibility and plans consistency. It is based in three phases with several activities: a) System Specification (determining the environment; determining goals and functionality); b) Architectural Design (Defining agent types; designing the system structure; defining the interactions between agents); and c) Detailed Design (defining capabilities, internal events and plans; determining the detailed agent data structure). The perceptions, actions and external data are the base to define the environment. Functionality is identified by means of goals and sub-goals grouped up according to data coupling. It has a clear notation, easily learned and used; it is adaptable and traceable methodology but not easily scalable. Weak support of concurrency. | The system specification is based in stakeholders and scenarios diagram; the analysis phase is supported by goal overview, role and data coupling diagrams; the architectural design generate agent acquaintance, system overview, and agent descriptors (describe the lifecycle of each agent type, its functionalities, data used and produced, its goals and events related, its actions and other agents which can interact with it) and protocols; finally, the detailed design is based on process, agent overview diagrams, and capacity capability overview, event, data and plan deliverables. It has two tools: Prometheus Design Tool (PDT) which allows cross-checking, and edition of diagrams; and the JACK Development Environment (JDE) which generates JACK code. System specification is not supported by these tools. Agent concept: highly autonomous BDI agents. Application domain suggested: intelligent and BDI MAS. |

| | |
|---|---|
| TROPOS It is strong focused on early requirements analysis to capture the reason to develop the MAS. The software development process consists of five phases: 1) Early Requirements (functional and non-functional requirements are identified as hard and soft goals respectively; stakeholders are actors; both are represented in actor and goals diagrams; goal and plans are analysed based on means-end analysis, AND/OR decomposition); 2) Late Requirements (extending the systems models to the environment and their interdependencies); 3) Architectural Design (new actors are included and described by means to an extended actor diagram; identifying the capabilities; grouping them to form agent types accordingly); 4) Detailed Design (defining the low level agent specification abilities, plans and interactions); 5) Implementation (implementing the specification in a BDI platform: JACK). It does not support the verification and testing phases. | Uses JACK as BDI platform which provides five main language constructs: 1) agents; 2) capabilities; 3) database relations; 4) events; and 5) plans. Does not provide an automatic way to convert Tropos' concepts to BDI concepts, but a guideline to do it is offered in the methodology. UML and AUML diagrams to represent the MAS elements are used like activity diagrams (detailed capabilities and plans) and agent interactions diagrams. It only has a diagram editor as a development tool. Good support of concurrency. Does not provide strong support for protocols or for modelling the systems dynamics. Agent concept: BDI agents Application domain suggested: intelligent and BDI MAS. |

| | |
|---|---|
| INGENIAS It covers analysis and design of MAS with a lifecycle defined based on Rational Unified Process and more recently it is based in OpenUp. Accordingly, five points of view describe the MAS: agent, interactions, organization, environment and goals/tasks. Recently, a Scrum process (a hyper-productive development software development process) has been adapted to this methodology. Scrum is an Agile project management framework based in development experience. Its development process consist in several iterations of five steps called sprints each one lasting for 2-4 weeks performed in two phases that comprises some tasks: 1) Preparation phase (initiate product backlog; plan release; and preparation tasks); and 2) Sprint phases (where each sprint goal is defined by the product owner accordingly to priorities and team capabilities. At the end of a sprint a product increment potentially releasable is generated. The evaluation of the product produces a backlog update before next sprint). Code generation facilities are explained in INGENIAS Agent Framework (IAF), particularly, Technique (an algorithm to create a work product) and Guideline (a set of rules and recommendations). | The main concept is the meta-model language. Five kinds of meta-models define the MAS: organization (defines the MAS architecture, structure and functionality, and the goals and workflow), environment (identify the sources of agent perception, systems resources, applications and agents, then the interactions are identified), task/goals (describes the dynamics of agent mental states), agent (its primitives, mental states and entities) and interaction between agents related to organizational goals. IAF allows the reuse of previous developments (in JADE) that requires the INGENIAS Development Kit (a graphical editor to work with the specification model). It is based on the automatic generation of code approach upon the established specification during the sprints. Code generated could be incomplete. Agent concept: heterogeneous agent. Application domain suggested: general purpose, e-commerce MAS. |

| | |
|---|---|
| Process for Agent Societies Specification and Implementation (PASSI and Agile PASSI) PASSI. It is a process which gathers requirements by means of iterative incremental design stages. It comprises the building of five models through several phases: 1) System requirement (domain description; agent identification; role identification; task specification); 2) Agent society which identify agent tasks, interactions and coupling (ontology description; role description); 3) Agent implementation (agents structure definitions; agents behaviour description); 4) Code (based on FIPA standard to reuse and automatic code generation); and 5) Deployment (a representation of code and distribution of parts of the system). Agents are identified based in their functionality not in its roles. Agile PASSI. It is a code oriented methodology that uses some abstractions: 1) domain requirements description; 2) Agent identification; 3) domain ontology description; 4) Code reuse; 5) Testing. Its process comprises five steps: 1) Requirements; 2) Agent society; 3) Test plan; 4) Code; 5) Testing. Both of them are not iterative in coding and deployment phases. | It uses with use case diagrams, sequence diagrams, domain ontology description, class diagrams, communication ontology description diagrams (Class diagrams + ML Schema for textual representation), extended deployment diagrams to define the agent deployment specifying the behaviour of mobile agents particularly. Uses a CASE tool (Rational Rose plug-in) to support the designing work and code production that lies in code patterns and code reuse. Agent concept: heterogeneous, based in robotics concepts. Application domain suggested: MAS driven for experiments in robotics. |

| | |
|---|---|
| Methodology for Engineering Systems of Software Agents (MESSAGE) It consists of: a) Applicability guidelines; b) A extended UML modelling notation to support agent related concepts; c) An analysis and design process based on Rational Unified Process. The main focus is the analysis phase. It has five analysis models: 1) organization (overall structure from social and individual views, behaviours, responsibilities, processes, information and resources); 2) goal/task (a graph of decomposition of high-level goals into subgoals and task to being performed to achieve them); 3) agent/role (is the internal point of view, it represents agent specific information, goals, interactions and services); 4) domain/information (relevant information about the problem domain; content message descriptions); 5) interaction (captures the information exchanging between roles/agents). | It has two organization diagrams: structural relationships and acquaintance relationships. And many other diagrams to support the five phases of the methodology: UML Activity diagrams. Goal implication diagrams, i.e. state transition diagrams (extended UML elements composing models) and workflow diagrams. Agent diagrams and delegation structure diagrams. Class diagrams. UML interaction protocols. Agent concept: heterogeneous agents. Application domain suggested: general purpose. |

| | |
|---|---|
| BOCHICA It is a model-driven platform independent framework to develop MAS, it is based in a Domain Specific Language (DSL) to cover the main concepts of MAS: agents, organizations, interaction protocols, goals, behaviours, deployment aspects, etc. It can be extended through interfaces for custom applications domains and execution environments. It poses an iterative adaptation process to incorporate conceptual extensions step by step, additionally modelling agents had been enriched virtual worlds. It comprises an iterative process based on five stages: 1) modelling; 2) code generation; 3) refinement (adding business logic and fix code mismatches); 4) evaluation (collecting bottom-up requirements; 5) extension. | The DSL Framework includes concepts related to methods, theory, application domain, and agent architecture. It uses graphical model languages to view the relations of model artefacts of sub-aspects of the systems, like Ecore, a meta-model that has a graphical modelling support of UML class diagram style, capability of import from UML, XML schema and de/serialization and existing Java code. Agent concept: BDI agents. Application domain suggested: general purpose. |

**Table A.1:** *Agent Oriented Methodologies of Software Engineering*

The commented AO methodologies exhibit different level of adequacy to agent concepts and different approaches to build software systems i.e. MAS. Furthermore, only some of them are focused in testing and verification issues, which is an important requirement to develop large scale software.

Finally, the AO methodologies of software engineering must adapt its abstractions and tools to include the concepts of SO + EM in their development processes: building a hierarchy and a micro/macro-level dynamics where new behaviour or self-organization process can emerge within a controlled degree, as has been said in section 2. Consequently, as has been pointed out in [DiMarzoL2011], three challenges must be afforded in methodologies of engineering MAS including SO+EM:

- Methods to generate a window of desired emergence (controlled emergence).
- Tools, models, and frameworks to develop MAS with SO+EM.
- Mechanism of validation and verification.

## A.2 Including SO and EM in the Development Process

... Pendient Methodologies that includes the SO and/or EM concepts into their cycle of development are showed in Table A.2 where a summary between AO methodologies that support SO and EM is presented [DiMarzoL2011, Gershenson2007thesis, Hu2014OrgAOP, Hannebauer2002Aut2002, Iglesias1998SurveyAOMet, Khanh2004AgOrInfSys, Ebrahim2007Comparison].

| **Methodology & Process Description** | **Tools/Frameworks & Agent concept** |
|---|---|
| Customised Unified Process (CUP) It is an iterative process based on Unified Process (UP) which emphasis is on engineering macroscopic behaviour of SO systems. It has four phases: 1) Requirement analysis (identification of functional/non-functional and macroscopic requirements); 2) Design (focused on microscopic issues by means of architectural and detailed design); 3) Implementation (focused on agent behaviour microscopic level); 4) Testing and verification as an empirical iterative feedback process on the macroscopic level, i.e. system behaviour, in order to test, verify and re-design: testing (early iteration architectural level coarse-grained decisions; later iteration parameter tuning); verification of the system dynamics at the macro-level (agent-based simulations and numerical analysis algorithms). | Use cases, feature lists, domain model. Information flows, locality concepts depicted as UML activity diagram, design patterns defining decentralised coordination mechanisms. Applications: autonomous guided vehicles and document clustering. Type of agents: reactive/cognitive agents. |

| | |
|---|---|
| Metaself It integrates an architecture and a development of manageable self-* systems process. Components and functional/non-functional features are described through system and environment metadata. The system behaviour is controlled by rules for self-organization and policies of control, dependability and self-management. The development process is divided in 4 stages: 1) Requirement and analysis (determining functionality; identifying when self-* requirements are needed; determining the required quality of service); 2) Design (D1: determining architectural patterns and self-* mechanisms; D2: designing of agents and artefacts, metadata, rules and policies); 3) Implementation (produces the runt-time infrastructure: agents, artefacts, metadata and executable policies); 4) Verification (verifying the MAS desired functionality; identifying potential faults and their consequences, including corrective measures). | System architecture comprises autonomous components, repositories, rules and policies, and reasoning services. They can be accessed and modified when the system is running. Applications: dynamically resilient Web Services and SO industrial assembly systems. Type of agents: reactive/cognitive agents. |

| | |
|---|---|
| General Methodology It is a domain independent methodology that emphasize the vocabulary to describe SO systems as a solution to complex problems, focusing on the design and control aspects that provides five iterative phases of system development,: 1) Representation (identified constraints and requirements are translated to an appropriate vocabulary abstraction levels, granularity, variables and interactions- and modular division of the system); 2) Modelling (defining an internal and distributed control mechanism based in reduce friction and promote synergy through a mediator with not absolute command over the entire system); 3) Simulation (implementation of the system modules; testing of the system in different scenarios and with different mediator strategies); 4) Application (developing and testing the system in a real environment); 5) Evaluation (the real system behaviour is tested, measuring its performance and contrasted with the previous one). | Applications: traffic light control, SO bureaucracies and SO artefacts.<br>Type of agents: reactive/cognitive agents. |

| | |
|---|---|
| Simulation Driven Approach Is an incomplete methodology that integrates some activities like other methodologies like Gaia. It is located in the analysis and design phase, named an early design phase with many feedback loops. Comprising the following iterative stages: 1) Modelling (formulating strategies to define the system behaviour explicitly); 2) Simulation (running the system specification through stochastic simulation tools and tracing the results to generate feedback derived from statistical analysis); 3) Tuning (the model is changed according to its respective feedback iterating to the first stage). | It uses an Agent & Artifact meta-model that considers agents as autonomous entities driven by goals and tasks. Environmental agents are introduced and are differentiated from user agents as they exploit artefact of passive nature and services. All of them ban be used by agents to achieve individual and social goals. Applications: collectively sorting and plan diffusion. Type of agents: reactive/cognitive agents. |
| Tropos4AS Is an extension of the Tropos methodology that presents a framework to develop self-adaptive systems which features are gathered during the design phase. It comprises different modelling steps: a) conceptual (related to self-adaptive extending the goal model of Tropos with goal types and relationships; b) failure (adding undesired states and recovery actions); and c) alternatives (through design patterns to adequate its configuration and constraints). | A combination of concepts, guidelines and modelling steps from the Adelfe methodology to reach a bottom-up approach for engineering collaborative MAS with emergent behaviour and deal to adapt its goals to unknown and changing environments. Applications: self-adaptive systems. Type of agents: cognitive agents. |

| | |
|---|---|
| ADELFE It is based in the AMAS theory to exploit self-organising by cooperation. It comprises a complete software development process (from requirements to deployment), notations and tools to support it and help to the developer. It is based on RUP and establishes six WorkingDefinitions (WD) or phases with several activities or steps: 1) Preliminary requirements (define user requirements; validate requirements; define consensual requirements; establish keywords set; extract limits and constraints); 2) Final requirements (characterise the environment; determine use cases and Non Cooperative Situation; elaborate/validate user interface prototypes); 3) Analysis (analyse the domain; verify AMAS adequacy; identify agents; study interactions); 4) Design (architecture and multi-agent model; interactions language; agent design; fast prototyping; complete design diagrams); 5) Implementation and 6) Test (micro-architecture generation; code generation; behavioural implementation). | It uses UML and AUML diagrams like protocol diagrams, agent structural diagrams (detailed design for cooperative agents). Additionally, it has several tools to support the development and describes the process and help to the application of the AMAS theory, like the AMAS-Modelling Language to generate an abstract agent architecture, micro Modelling Language to edit the specification of micro-components, AMAS adequacy tool, the graphical modelling tool and the model-driven engineering tool that automatically translates models into Java code. The full implementation kit is based on Eclipse Modelling Framework and the Atlas Transformation Language (a model-driven tool) Applications: self-organising by cooperation MAS.<br>Type of agents: cognitive agents. |
| AuReCon Collaborative problem solving process, in which intelligent Ag determine on their own the structure of the organisation on the individual level. This is done by two local operations (melting/splitting) in intelligent Ag that belong to the same social real (team, organisation, etc.), they act as a sort of redistributing problem solving knowledge, competence, goals, and skills. | Type of agents: BDI, other cognitive agents. |

| | |
|---|---|
| Oragent Is an Organisation-based Agent-oriented programming approach. Introduce software engineering concepts (encapsulation, decomposition/composition, information hiding, modularity, and reusability) and organisational concepts (agent, group, role as behaviour rules, organisation, and role class) as first-class entities. The dynamics of MAS is supported by means of its mechanisms of role enactment and role-based interaction. A program at design-time is a set of organisations with a hierarchical structure (self-contained recursive structure). At run-time, a program is a set of groups (including a set or roles and dynamic aggregation/disaggregation of agents), agents (enacting at least one role when belonging to a group), and interactions intra-groups. Groups are first-class execution entities. Support dynamism and flexibility and facility to development, control and management MAS. | Provides three levels: 1) programming model, 2) programming language (syntax/semantics), 3) execution platform (Jade-based)<br>Type of agents: autonomous, reactive (need test for cognitive agents) |

**Table A.2:** *Agent Oriented Methodologies that includes the SO and/or EM concepts into their cycle of development.*

## Appendix B

# Emergence in MAS, Platforms and Applications

A wide range of definitions on Emergence appears in the literature, as those given in Chapter 2. In [113] a more detailed discussion of the definitions is provided. Therefore, to keep consistency with Computational Emergence notions presented in the aforementioned Chapter, Table B.1 shows those definitions besides the type of agent that can be addressed, the suitable frameworks to develop MAS and the kind of applications used or suggested (See Apendix A for a description of the frameworks and toolkits).

| EM types | Agent-MAS relationship | Agent Arch | Toolkits | Applications |
|---|---|---|---|---|
| **I Deducible** with local feedback | Agent with local knowledge/information (acquired by their own perceptions). Applicable to problems where cooperation is required or it facilitates the solution. There is no change in the Agent behaviours. | Reactive Agent | Prestige, NetLogo | Networking, RoboCup challenges, Robot coordination, Task allocation, Distributed Problem Solving, Nest building |
| **II Deducible** with local and global feedback | Agent influence the MAS and the MAS influence agents, frequently through of awards/fees (punishments), by means of what agents derive a partial knowledge of the MAS and so they can react/response to it modifying their actions. Agents with learning are required. Parallel action is required | Cognitive, Deliberative | Adelfe General Methodology | Flocking simulation, Networking Task allocation, Distributed Problem Solving |
| **III Non deducible** with local feedback and multiple global feedback cycles | Composite adaptive and agent awareness (by other agents or groups/agent coalitions) of EM derived from their interactions. Agent change their behaviour as a consequence of the macro-level and their own observations (in the micro-level) even in a full manner: adopting another set of rules/plans of action and modifying the behaviour of their federated agent. Agent with learning are required. Parallel action is required | Cognitive (BDI), Deliberative and Learning Agents | Oragent, Jadex | Transference of data |

**Table B.1:** *Types of EM, agent types, platform of development and applications*

# Appendix C

# Disaster-Rescue Frameworks: a comparison

Disaster Rescue Simulation is a very active Domain, with many research and technical conferences, either from the pure computational purposes or as in silico tool helping to determine feasible policies of actions in emergency management in the real life [55]. Particularly Agent Based simulations are becoming the de facto tool with a wide range of implementations under different approaches. In the following subsections a review of the main platforms involved as reported in literature is shown.

Regarding to the MAS toolkits and platforms, in [55] several simulation tools are analysed, being Jade and Repast claimed as the more remarkable. On the other hand, in [67] a more extensive review focused on the type of agents/platforms and application domain is performed, this is summarised in Table C.1

According to purpose of this thesis, the system has the following general features:

- It is composed by BDI Agents extended with learning capabilities, under the Collective Intentional Learning setup.
- It is applied to the Disaster Rescue Domain, a challenging and partially observable environment

From the discussion presented in [55, 67], and accordingly with [12] the suitable toolkits and platforms for BDI MAS developing in contexts similar to the Disaster Rescue Domain (Artificial life and behavioural ob-

| **Domain** | **Toolkits & Platforms** |
|---|---|
| General purpose distributed simulations | JADE, Jadex, **Jason**, EMERALD, MaDKit, CybelePro, JIAC, AgentScape, AnyLogic, GAMA |
| General purpose agent based simulations | JAS, AgentBuilder, Agent Factory (AFSE), Swarm, MASON, INGENIAS Development Kit, Repast, MaDKit, AgentScape, SeSAm, AnyLogic, **NetLogo**, GAMA, JAMES II |
| Scientific simulations | Swarm, MASON, Cormas, Repast (Social Sciences), MaDKit, CybelePro, SeSAm, AnyLogic, GAMA, JAMES II |
| Dynamic and complex environments | JACK, Cougaar, CybelePro, AnyLogic |
| Real world GIS | AGLOBE, Cougaar (integrated with OpenMap), Repast, CybelePro, SeSAm, AnyLogic, GAMA |
| Large scale simulations | Cougaar, CybelePro, JIAC, AgentScape, GAMA, JAMES II |
| Scheduling & planning | Jadex, CybelePro, SeSAm, AnyLogic, **NetLogo**, GAMA |
| Mobile computing | JADE, Jadex, Agent Factory |
| Multiple domains | JADE, Jadex, MaDKit, CybelePro, Cougaar, JIAC, AgentScape, Agent Factory, Repast, SeSAm, AnyLogic, GAMA, JAMES II |
| Artificial life and behavioural observation | JADE, Jadex, **Jason**, SeSAm, EMERALD, JACK, Cougaar, CybelePro, AnyLogic, **NetLogo**, GAMA |
| **Biological & social studies** | JADE, SeSAm, Jadex, **Jason**, EMERALD, JACK, Cougaar, CybelePro, MaDKit, Repast, AnyLogic, **NetLogo**, GAMA, JAMES II |
| Economics/eCommerce | JADE, EMERALD, JACK, Cougaar, CybelePro, MaDKit, AnyLogic |
| **Natural resources & environment** | Cormas, Swarm, SeSAm, AnyLogic, NetLogo, GAMA, JAMES II |

**Table C.1:** *Platforms and application domains overview. Boldfaced names are the most relevant ones to BDI MAS and the Disaster Rescue Domain.*

servation, Biological and social studies, Natural resources and Environment, Agent Based Social Simulation, and General purpose) are JADE, Jadex, NetLogo, Jason and GAMA, between the most mentioned.

In [55] the prestigious RoboCup Rescue platform has claimed as having a normative decision theory perspective, i.e. that the main purpose is to maximize the objective function. Consequently, the better responses of agents behaviour are obtained by the design and implementation of new action/selection method mainly for rescue agents, which do not necessarily represents the real-life behaviour.

Furthermore, in [117] Jam and a NetLogo extended version (to include a minimal BDI layer) are claimed as preferred over other popular BDI development environments like Jason because of their accessibility to the agents intentions into the reasoning cycle. A rich environment to develop urban based environments is GAMA, but it needs to be adapted for supporting BDI agents. Particularly, NetLogo is highlighted because of its availability of rich environments. Finally, Jason has included diverse of features that easy the implementation of learning under the intentional architecture. Consequently, the usage of both frameworks, Jason and NetLogo are preferred to the development of the BDI MAS with Collective Intentional Learning.

## C.1 NetLogo: more than an environment rendering tool

NetLogo is highly relevant in the Disaster Rescue Domain due its flexibility to represent environments, as already mentioned, but also because of its capabilities as a full Agent-Based Modelling environment that allows to represent reactive agents with many features and abilities of perception and action.

NetLogo is defined by its creators as a MAS programmable modelling environment [132]. It offers a graphical environment to easily show and test the results of the programs, and customize the input/output to the MAS system, so the projects (known as models) can be developed under a rapid prototyping approach [43, 92, 96]. Models are coded in its own list-based programming language that exhibits a great expressiveness and offer a large set of primitives, maths operations, and a set of plotting and monitoring graphical tools.

The models presents a grid-like environment populated by two main agent entities: `turtles` (acting as agents with mobility among many other capabilities) and patches, the tiles that compose the entire environment. Other type of agent are the observer, acting as the interface with the outside and allow the reception of commands to the system, and links i.e. agents that connect two turtles. The agents can also be grouped by temporary sets known as `agentsets` or fixed sets known as breeds that in addition to the global set of variables and command, they can share its own set of them.

The extension of NetLogo presented in [95, 96] implements a limited BDI layer coded directly into the NetLogo programming language (the model code can be downloaded from [2]). This work is mainly focused on teaching purposes to be used in courses about reactive and BDI-based MAS. They implemented the Beliefs, Desires and Intentions in a very simple and effective list-based way excluding the explicit representation of plans, reaching a basic BDI behaviour making easy it use for the programmer. The main features of this extension are the following:

- The set of intentions representing the goals of agents are pushed into a stack that is specified as a turtle or breed-own variable.
- Each intention is a list composed by its name and a Boolean condition of fulfilment.
- A FIPA-ACL-like [1] message passing is also implemented, which allow the messaging between the agents in the environment using a list with the elements (performative, sender, receiver, content)

assuming the same ontology for all the agents.

On top of that, they proposed scenarios to integrate practical assignments for teaching purposes [95,96]. The relevant scenarios to this research are the rescue units and its extended version:

- Rescue units scenario: in which agents operate in a disaster area searching for victims to rescue. Composed by reactive gents with no message exchange capabilities.
  - Rescue procedure: a rescue unit locates the victim and provides medical supplies, so that they can be sustained in life until transportation arrives.
  - Elements:
    - Rescue team: consists of rescue-unit autonomous agents that can move around the disaster area, detect victims and temporarily rescue them.
    - Base station: provides refuelling and renewal of medical supplies services.
    - Agents sensors: a very limited set of sensors, civilians and obstacle detectors, sensors detecting, low fuel levels, etc.
    - Agents abilities: move around in the disaster area, provide immediate attention to the victims, move towards the base, etc.
- Extended Rescue units scenario. Additionally to the above configuration, all agents exchange explicit symbolic messages and follow BDI or hybrid architectures. Ambulance agents that are able to collect the rescued civilians and transport them to the base are added, with a main restriction: an ambulance can save a victim only if it has been discovered by a rescue unit.

In Figure C.1 a screenshot is presented where can be seen the environment populated by all available type of agents. However, despite of being an interesting proposal, as the purpose of this scenarios is merely instructive, lacks of realism to simulate environments under disaster conditions and more development is needed to be used as a simulation tool. On the other hand, for the sake of realism, it is important to notice that NetLogo includes an extension to support Geographic Information Systems (GIS) [132]. It provides the ability to load vector GIS data (points, lines, and polygons), and raster GIS data (grids) into the models. The

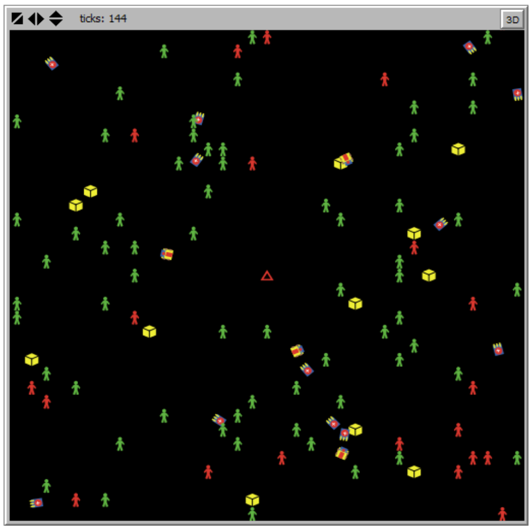
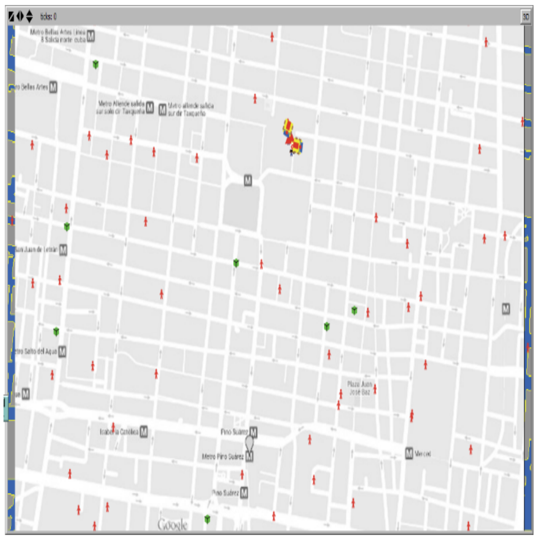


**Figure C.1:** *Screenshot of extended rescue unit scenario where the base, civilian, rescue unit, police and obstacle agents can be seen.*

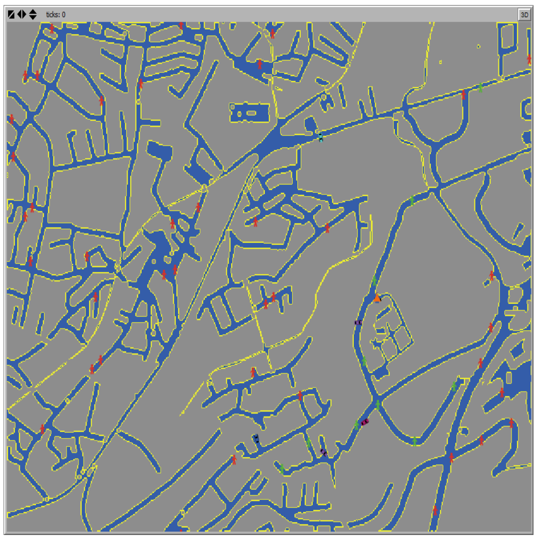
extension supports raster data in the form of `ESRI ASCII` Grid files (`.asc` or `.grd`) used as an interchange format by most GIS platforms [132].

The GIS facility is used to render a `.png` or `.jpg` image belonging to a map in the NetLogo community model *Traffic model on google map* proposed in [3]. In order to be loaded in the model the `.png` file of the image is transformed into an `.asc` file by an auxiliary program (not provided by the model). Using this approach, jointly with the rescue unit scenario, a more interesting environment can be reached, for instance, when taking an image of a true city map. A couple of examples of this settings are shown in Figure C.2 using maps of the centre of Mexico City, and the Cuajimalpa district located in its boundaries.

Finally, this mapping capability of NetLogo represents a convenient option for MAS development according to the particular design purposes and requirements of the domain. The BDI MAS focused on DR simulation was developed exploiting this rendering facilities as described in Chapter 3.



i) Map of the centre of Mexico City with the maps image superposed in the `.asc` file.



ii) Map for rescue simulation representing a neighbourhood in the boundaries of Mexico city. With no map superposed image.

**Figure C.2:** *Rescue simulation scenarios generated from real maps of Mexico City.*

Appendix D

# Notes to the Implementation and Code Availability

## D.1 Integrating ABM and BDI MAS: Jason and NetLogo

According to the literature, JADE, Jadex, NetLogo, and Jason are the most noteworthy platforms to MAS development (including BDI agents) either for general purpose systems or to be applied in complex context such as those typically addressed by ABM, which comprises the Disaster-Rescue Domain.

Jason is a full-fledged platform that enables the implementation of BDI agents with many user-customisable features including Java-based environments and legacy code. Jason is a very convenient framework given its robustness, more strict implementation of the BDI principles and the facilities of *AgentSpeak* as a higher level agent programming language.

NetLogo is an MAS programmable modeling platform for simulating social and natural phenomena given the facilities to represent different kinds of reactive agents and its power and easiness in defining and rendering rich environments with different features.

In this work, the interaction between both platforms is demonstrated through a framework working in two domains: the Vacuum-cleaner and the Disaster-Rescue one. In the first place, a single agent is placed in a simple grid-like environment. In the second one, a BDI MAS applied to the Disaster-Rescue domain was implemented exploiting the Java-based environments in Jason that allows the interaction with the BDI agents with other systems, and the possibility of NetLogo to receive external commands to drive its functioning. The interoperability of both platforms is warranted, given the facilities they offer through their API interfaces. There are two possibilities to pursue the connection Jason-NetLogo: either to include Jason into NetLogo, using just the Jason BDI engine or including NetLogo in Jason, using the former as the environment.

In any case, the components are:

1. **A NetLogo model**: the description of the environment, their elements and the functions to define new turtle agents and their capabilities. Example, an excerpt of the model `FollowWalls.nlogo`:

```
  to setup      ;; Set the environment
    clear-all
    set running 1
    ;; make some random walls for the turtles to follow.
    ;; the details aren’t important.
    ask patches [ if random-float 1.0 < 0.04
                    [ set pcolor brown ] ]
    ask patches with [pcolor = brown]
            [ ask patches in-radius random-float 3
                [ set pcolor brown ] ]
    ask patches with [count neighbors4 with
            [pcolor = brown] = 4] [ set pcolor brown ]
    reset-ticks
  end
...
  to createAg       ;; Function to define new agents
    sprout 1 [
      set size 2              ;; bigger turtles are easier to see
      set pen-size 2          ;; thicker lines are easier to see
      face one-of neighbors4  ;; face north, south, east, or west
      ifelse random 2 = 0
        [ set direction 1     ;; follow right hand wall
          set color blue ]
        [ set direction -1    ;; follow left hand wall
          set color green ]
      ]
  end
...
```

2. **A Jason definition**: the definition of the MAS project (even in the case of a single agent) and the definition of the Jason agent's code. Example, the definition of the MAS project `NLogoEnJason.mas2j`:

```
MAS nlogoInJason {
environment: AmbNLogoInJason
agents:
        agVacuumCleaner;
classpath:
"/usr/local/lib/netlogo5.2/NetLogo.jar";
}
```

And, part of the definition of the `agVacuumCleaner` agent.

```
 +start(ID) : true <-
        .print("Hi, the Jason Ag, agVacuumCleaner (me), is starting");
        // Actions that goes directly to NL to create the turtle
            // 1 Order the setup of the new turtle
            S = "setup";
            sendCmdNL(S);
            // 2 Configurate the new agent (size, shape, color, etc.)
            +creaConfNL(C);
            sendCmdNL(C);
            // 3 Put the pen down to show the agent's traces
            sendCmdNL("ask turtles [ pen-down ]");
            // 4 Start working to follow walls
            !continue(ID).
...
```

3. **Environment definition**: includes the opening of the NetLogo model used by the system as an environment, and the functions to receive/send information to/from the NetLogo model.

```
public class AmbNLogoInJason extends Environment {
    private Logger logger = Logger.getLogger("nlogoInJason.mas2j."
                            +AmbNLogoInJason.class.getName());
    final String dirmodelo = "NLogoInJason/";
    final String modelo = "FollowWalls.nlogo";  // The NetLogo model
    final javax.swing.JFrame frame = new javax.swing.JFrame();
    final InterfaceComponent comp = new InterfaceComponent(frame);
...
 /** Called before the MAS execution with the args informed in .mas2j */
  @Override
  public void init(String[] args) {
    super.init(args);
    try {
        java.awt.EventQueue.invokeAndWait(
            new Runnable() {
                public void run() { // Window features
```

```
                        frame.setSize(800,650);
                        frame.add(comp);
                        frame.setVisible(true);
                        try {           // Open the model
                            comp.open(dirmodelo+"/"+modelo);
                        }
                        catch (Exception ex){
                            ex.printStackTrace();
                        }
                }});
...
/* Overrides the executeAction function to perform actions in the environment */
  @Override
  public boolean executeAction(String agName, Structure action) {
    String cmd = "";           // To build the strings representing the commands
    logger.info("[INICIO]<"+agName+"> Executing: "+action.toString());
    //Identifying and translating incoming requests from agent to environment
    //sendCmdNL: an AgentSpeak formula for requesting turles to execute actions
    if (action.getFunctor().equals("sendCmdNL")) {
        cmd = action.getTerm(0).toString().replace("\"","");
        if(cmd.equals("setup")){
            sendCmd(cmd,this.comp);
        logger.info("[*] Adding perception: "+cmd+"");
            addPercept(Literal.parseLiteral(cmd));
        }
        if(cmd.contains("createAg")){
        logger.info("[*] Adding perception: createAg("+contAg+")");
            addPercept(Literal.parseLiteral("createAg("+contAg+")"));
```

```
                sendCmd(cmd,this.comp);
            }
            if(cmd.equals("ask turtles [ pen-down ]")){
                    sendCmd(cmd,this.comp);
            }
    ...
```

Both interconnection options and the developed systems are briefly described as follows.

### D.1.1 Jason-in-NL

Only use the Jason BDI engine. The following packages are used in doing so:

- Extend the Jason class AgArch (package `jason.architecture`): create a Jason agent and manage its reasoning cycle. An additional package is also required: `RunCentralisedMAS` belonging to the package `jason. infra. centralised`.
- From the NetLogo side: execute the NetLogo GUI with the model, `org`.`nlogo`.[`lite`,`headless`,`api`] packages and `InterfaceComponent`, `HeadlessWorkspace` and `CompilerException` classes are required.

The provided example use the `Fire` model from the NetLogo library that represents a burning field, where the fire are agents moving up changing the status of the patches in the background, simulating the burning effect in the screen.

### D.1.2 NL-in-Jason

NetLogo is included as an Environment in Jason. To do so, it is required to use the following facilities:

- Perceive the environmentby extending the class `Environment` of the Jason architecture package `jason.environm` to transform the perceptions taken from NetLogo into readable literals for Jasons agents according to the *AgentSpeak* semantic definition, in order to be inserted in the agents beliefs base.

- Act in NetLogo. The `InterfaceComponent` of the `nlogo.lite.InterfaceComponent` package allows for sending commands to the elements of the model (turtles and patches).
- Load the NetLogo model as an environment in Jason using a `JFrame`. Complementarily, `java.awt.EventQueue. invokeAndWait` and `java.io.[InputStreamReader, BufferedReader, IOException]` are used to perform the interface and keep both platforms running together.

Examples:

- Single agent. The example use a wall following example from the NetLogo example repository with a single agent: a turtle in an typical NetLogo environment for following walls, but the decisions are taken on the side of Jason, leaving the execution and perception on the NetLogo turtle's side. So, the turtle use the BDI-Jason engine only, to decide what to do from the perceptions translated from NetLogo language to Jason First-Order-Logic *AgentSpeak* language. After a cycle of perceiving-reasoning-acting, the agent changes its colour.
- MAS. The example shows the same environment, but using several turtles keeping the decision-making on the side of Jason, and the execution and perception on the NetLogo turtle's side. After a cycle of perceiving-reasoning-acting, agents change their colour.

## D.2 Vacuum-cleaner

A single agent is allocated in a simple Vacuum-cleaner environment. There are three implementations:

- Intentional Learning:
  - The agent has the purpose of performing the left-handed Wall-Following behaviour through Intentional Learning.
  - The agent has the purpose of performing the Target-Searching behaviour through Intentional Learning.
- Pure Reinforcement Learning. This example implements a pure Reinforcement Learning strategy to reach a target-searching behaviour, i.e. assigning Q-values to each of the states in the grid-like

environment and selecting that one with the higher value to move into.

## D.3 D-R Simulation

This implementation take advantage from the interaction of both platforms, where NetLogo is highlighted by the deliberation process of BDI Jason agents that enhance the capabilities of the pure reactive NetLogo turtles going beyond the pure quantitative heterogeneity (variations in the values of their parameters and variables). Jason can be enriched in at least two aspects:

- By including a powerful environments rendering tool and the capability to easily interact with it through the avatar-like Netlogo turtles.
- By inheriting the skills and functionalities already present in the turtles useful to define lower level tasks, for instance navigation and sensing.

Both tools can be combined with other systems to manage more than one user-defined environment.

In these implementations, a BDI MAS using NetLogo and Jason in the Disaster-Rescue domain is developed. In this example agents lack from learning capabilities, but they have a complete set of plans to perform all the required behaviours in the domain.

Finally, a BDI MAS is implemented using NetLogo and Jason in the Disaster-Rescue domain with the capability of learning in the single (IL) and in the collective cases (CIL).

## D.4 Code Availability

The code of the D-R Simulation without learning can be downloaded from:

`http://corban.cua.uam.mx/~wluna/JNLBDI-DR/index-en.php`

Additionally, the rest of the code is available on request in: `wluna@correo.cua.uam.mx`

Personal page (Spanish):

`http://hermes.cua.uam.mx/informacion_academicos.php?academico=WulfranoArturoLunaRamirez`

### D.4.1 Directions to run the simulation

It is thought to be run on Linux, but other systems can be used, or running from JEdit directly by opening the file `rescueBDI_vN.mas2j`:

```
1. Once opened and unzipped, the directory rescueBDI_vN will be created

2. Go to the directory ~/rescueBDI_vN/

3. Execute the script correPruebas.sh:

    utilerias/correPruebas.sh RUNS MAXACTS NUMAGS

 Where:

    RUNS = number of run per execution

    MAXACTS = max number of actions (ticks) (ex. 35000)

    NUMAGS = number of agents ambulance for each run (suggested: 4)


    Ex. utilerias/correPruebas.sh 2 35000 4

    ----- It will run the system twice up to 35000 actions for each one,

            with 4 Ambulance agents


    The default configuration of agents is:

        Fires:10

        Blockades:10

        Civilians:100

        AgBrigadiers:16

        AgFireBrigades:4

        AgPolice:4

        CityMap:MxCity


    It can be modified by the parameters:

        AgAmbulance:4
        MaxTime:35000


    LOG:

    ~/rescueBDI_vN/salidasExp/AAA-MM-DDlogRuns.txt


4. A directory in salidasExp will be created to store the results of each run:

    -images from the last view of the BDI-MAS

    -files of performances (values: .csv, & image: .png)

    -summary of the performance of each run (.txt)

    -log of the runs (.txt)


NB: to store the current run in a log, rename the file:

            _logging.properties -> logging.properties
```