ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS OF CESENA

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
Second Cycle Degree in Computer Science and Engineering

VISUAL PROGRAMMING PARADIGM FOR ORGANIZATIONS IN MULTI-AGENT SYSTEMS

Thesis in
PERVASIVE COMPUTING

Supervisor
Prof. ALESSANDRO RICCI

Co-supervisors
Prof. SIMON MAYER
Dott. SAMUELE BURATTINI

Presented by
ALESSANDRO MARCANTONI

Academic Year 2021 – 2022

KEYWORDS

Multi-Agent Systems
Multi-Agent Oriented Programming
Visual Programming
Organization

To the ones who have always believed in me

Index

Introduction

In the context of a Pan-Europen project focused on defining the next generation of IoT with a human-in-the-loop approach, this thesis aims at expanding the vision for an accessible Integrated Development Environment that mixes Multi-Agent Oriented Programming and Hypermedia in a seamless interface for both humans and machines.

The thesis work was carried out while being hosted by the University of St. Gallen at the Interaction- and Communication-based Systems research group that is currently exploring the field of engineering autonomous systems.

The idea comes from the need to keep up with the fast digitalization of business activities and to provide a user-friendly tool that can be used to create and configure complex systems with low or no code. Indeed, some efforts have already been made in this direction with the development of a block-based programming language for software agents.

However, the current state of the art does not provide a complete solution for the development of complex systems. As a matter of fact, dealing with interactions and coordination between agents directly within them does not represent a scalable approach as the design complexity exponentially increases.

All Multi-Agent Systems possess some form of organization, although it may be implicit and informal. With the increasing complexity of scenarios and the need to deal with a large number of agents, the need for an explicit specification of the organization at design time became more and more evident. Nowadays, the organization is considered one of the first-class abstractions in the design of Multi-Agent Systems.

Therefore, this thesis aims at providing a solution that allows users to easily impose an organization on top of the agents. Since ease of use and intuitiveness remain the key points for this project, users will be able to define organizations through the use of visual language and an intuitive development environment.

Chapter 1 goes more in-depth into the motivations that brought the definition of this research proposal while Chapter 2 provides a brief overview of the current state of the art for Multi-Agent Systems, Multi-Agent Oriented Programming, and Hypermedia Multi-Agent Systems.

In chapters 3, 4, and 5 the development process is presented, focusing on

the analysis of the problem and the reference technology, the design of the solution, and the implementation of the prototype.

Finally, Chapter 6 describes the evaluation of the prototype that consisted of a test carried out with a group of users to gain qualitative feedback on the usability of the tool and to identify possible future improvements.

Chapter 1

Context, Motivations and Research Proposal

This project was born thanks to the collaboration between the *Pervasive Software Lab*[1] of the *University of Bologna* and the *Interaction- and Communication-based Systems*[2] research group of the *University of St. Gallen*, in Switzerland.

Since both groups are interested and involved in similar research topics, such as the interactions among devices and people in ubiquitous computing environments, a highly enriching opportunity for an internship abroad arose.

Moreover, the research group in St. Gallen is contributing to a European project named *IntellIoT*[3], which stands for "Intelligent IoT" and whose aim is, together with its partners, to develop a reference architecture to enable IoT solutions for applications where autonomy, intelligence, and the Human-in-the-Loop strategy are key requirements. Hence, the concurring perspective of the two research groups and the St. Gallen group's participation in such a visionary initiative made it easy to shape the requirements of the thesis work.

The following sections will present the main objectives of the *IntellIoT* project to describe the context in which the thesis was conducted.

[1] https://apice.unibo.it/xwiki/bin/view/PSLab/
[2] https://ics.unisg.ch/chair-interactions-mayer/
[3] https://intelliot.eu/

1.1 The *IntellIoT* Project

IntellIoT is a Pan-European project that focuses on developing integrated, distributed, human-centered and trustworthy IoT frameworks, with particular attention to sectors like agriculture, healthcare, manufacturing, energy, construction, and smart cities.

To achieve the latter goals, *IntellIoT* explores and exploits new enabling technologies such as 5G connectivity, distributed technology, Augmented Reality, Artificial Intelligence, and tactile internet. Of course, this is possible thanks to the project's partners which are spread across ten countries and form a competitive ecosystem.

Among them, the *University of St. Gallen* is currently focusing on integrating physical things into the Web, increasing the autonomy of Web-enabled devices, and making interactions of connected devices intelligible for people using **Hypermedia Multi-Agent Systems**. Indeed, the primary objective of this thesis is to explore how humans can effectively define the organization of the software agents that control such systems.

1.1.1 Mission

Smart technologies play a significant role in our life and work. However, the traditional approach based on cloud technologies has limitations, such as unreliable connectivity, limited bandwidth, long reaction times, lack of autonomy, and trust concerns. Therefore, the goal of *IntellIoT* is to tackle these issues and create a framework enabling next-generation IoT solutions. Specifically, these issues are addressed by focusing on the following three pillars, which are the central research topics of the project.

Collaborative IoT

Various semi-autonomous entities should cooperate to achieve the system's overall goal. Hence, self-awareness and knowledge of the task to perform and the environment where they are located are vital abilities to seek. Entities can acquire knowledge either by interacting with the environment via sensors or by communicating with each other. However, since providing complete knowledge to entities in open and continuously changing environments is practically infeasible, Artificial Intelligence and Machine Learning algorithms are exploited.

Human-in-the-Loop

Since IoT applications cannot be completely autonomous in how they decide and act, humans need to be involved in controlling and optimizing the Ma-

chine Learning Systems the devices are endowed with. Indeed, the interaction between humans and intelligent systems can expand the latter's knowledge about the environment or the application by exploiting the former's experience. In fact, by applying Machine Learning algorithms, the devices can learn new features and information about the overall process so that they will have enough knowledge to react to similar scenarios in the future automatically.

Moreover, end users should be involved more in the design process of the system, as they are the ones who know the application domain best. Their contribution can be indeed crucial, moving from just defining the requirements of the application to the actual development and configuration. Therefore, a user-friendly and intuitive environment is needed to enable non-expert users to easily define the system's behavior.

Trustworthiness

As beneficial as IoT devices are, they present some major security concerns. The Mirai botnet exploiting embedded devices to perform DDoS attacks [2], possibly hackable cardiac devices, and Stuxnet sabotaging Iranian nuclear facilities [3] are only a few examples of critical breaches. Thus, security, privacy, and trust are vital for IoT systems and applications and their broader acceptance. Therefore, these concepts must be considered early in the design process and regard computation and communication infrastructure.

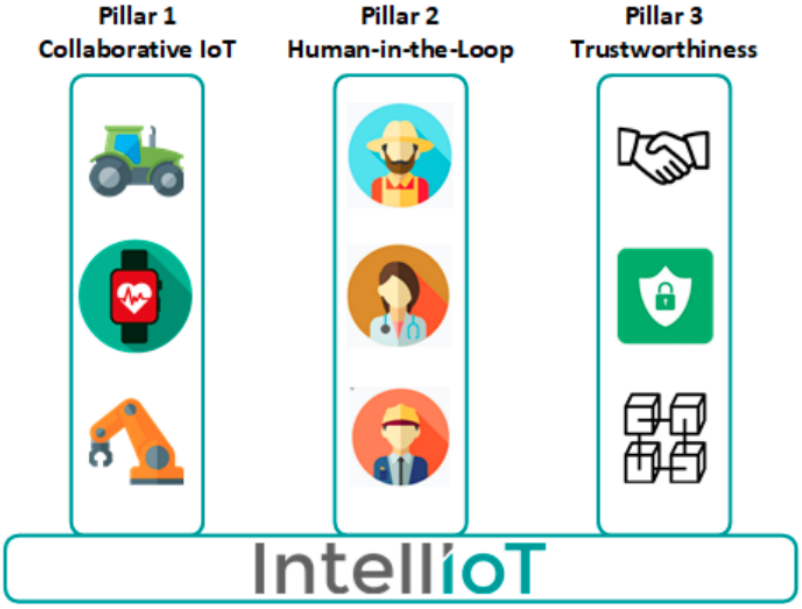


Figure 1.1: The three pillars of *IntellIoT*.

1.1.2 Use Cases

The above three key component areas are supported by *IntellIoT*'s dynamically managed network and computation infrastructure that, combined, provide resource and edge management, orchestration capabilities, and network choreography, exploiting cutting-edge technologies like 5G. Moreover, for the pillars to not remain only abstract concepts, various use cases that aim to address real-life problems in three core sectors were developed:

- **Agriculture**: the application of IoT in agriculture could be a life-changer for humanity as we now witness how extreme weather, deteriorating soil, dry lands, and collapsing ecosystems make food production more and more complicated and expensive, not to mention the population growth that increases the demand for resources.

  Although "smart farming" is already quite popular, *IntellIoT* aims to bring it to the next level thanks to autonomous devices such as self-driving tractors and drones endowed with sensors and actuators. However, even though machines perform potentially dangerous, tiring, and repetitive tasks for humans, the latter still play a crucial role in managing the farm. Indeed, they can remote control the devices in uncertain situations, refining the Artificial Intelligence models. Additionally, human operators are in charge of defining the goals of the farming system, leveraging their experience and knowledge about the domain.

- **Healthcare**: IoT is revolutionizing the healthcare industry, mainly due to remote patient monitoring. Indeed, being endowed with specific sensors, devices can track the latter's vital signs and other health metrics and send the collected data to healthcare providers. Some examples of such devices are *Continuous Glucose Monitoring* [14] and *Dissolvable Brain Swelling*[27] sensors or ordinary smartwatches. This process improves the patient's quality of life, which does not need to be limited to their home or the hospital and can thus carry on with their regular everyday activities.

  Moreover, continuous monitoring and real-time data sharing allow timely interventions when necessary, and automatic data collection can drastically decrease the time and effort required to retrieve and manage information about the patient. Not to mention the opportunities for data analysis and possible Machine Learning models that would support clinicians in being more efficient. However, this workflow potentially exposes patients' sensitive information; therefore, we find privacy, security, and trust assurance among the main focuses of *IntellIoT* regarding this use case.

- **Manufacturing**: IoT is one of the core driving forces behind Industry 4.0, which aims to digitalize and automate operational processes counting on as little support as possible from human operators, from the order submission to the delivery of the product. Leveraging AI and Machine Learning, robotics, and data analysis, IntellIoT envisions a future with shared manufacturing plants and multiple customers utilizing manufacturing as-a-service.

  According to the latter scenario, an intelligent IoT environment would derive a production plan from data received from a customer. Subsequently, the software agents that control the machines would organize according to their role, which is adopted taking into account the machine's capabilities, making it possible to complete the planned steps. However, whenever AI is not sufficiently confident about a step, a human-in-the-loop can take over control remotely, providing information that will be learned by the Machine Learning algorithm thanks to continual learning.

1.2 Domain-Expert Programming

As seen in the mission of the *IntellIoT* project, the crucial role of end-users in the definition of autonomous systems, the importance of their intervention in case of need, and their expertise are utterly unmatched by Artificial Intelligence algorithms and probably will be for a long time.

On top of that, the gap between programmers and end-users regarding domain comprehension is a well-acknowledged issue concerning software development. Indeed, developers' poor understanding of the domain often results in projects missing their schedules or exceeding their budget, poor-quality software, or even wrong functionalities [13]. To address this critical issue, several techniques have been developed. For instance, one of the core aspects of Domain Driven Design is *knowledge crunching*, which aims to extract domain knowledge from the experts to reflect it in the code during the subsequent development phases.

On the other hand, a different approach might be taken directly involving domain experts in the programming process. This kind of user can be defined as professionals in some domain different from computer science who need to use computers in their daily work and often have real needs to perform some programming activities that result in the creation or modification of software artifacts [12]. Given the latter definition and the premise suggesting the importance of domain expertise, providing domain experts with tools, such as domain-specific languages (DSL) or more user-friendly visual tools, that allow them to "code" or configure parts of complex systems feels

very natural. Therefore the need for an intuitive development environment in which users with no proper computer science background can naturally and seamlessly transform their domain knowledge into specifications with low-code (or possibly no-code).

1.3 Agent-Oriented Visual Programming

Multi-Agent Systems is one of the core enabling technologies of the infrastructure of *IntellIoT*. MAS fit IoT because they tackle the complexity and handle the decentralization of the IoT ecosystem by providing a framework for coordinating the actions of a large number of devices, allowing the latter to communicate and make decisions toward the achievement of a common goal. Another critical advantage of MAS is their ability to deal with partial knowledge, incomplete and imperfect information, and adapt and reason in real-time, which is crucial in dynamic, uncertain, open, and distributed networks.

However, the high-level expertise required to program agent-based systems hinders the large-scale adoption of MAS. Therefore, to facilitate the spread of this technology, efforts have been made to eliminate the entry barrier to MAS development. One example of such endeavor is the development of an agent-oriented programming paradigm [31][8], which enables individuals without coding experience, but with knowledge of specific target domains, to design and (re)configure autonomous software.

This proposal makes the development of software agents easier in two ways:

- **Use of human-oriented abstractions**: the application of the BDI (Belief-Desire-Intention) model, which makes use of concepts such as goals, plans, beliefs, and intentions, allows defining the agent's behavior more naturally, as this paradigm matches more closely people's everyday experience.

- **Use of visual programming techniques**: this project makes use of block-based programming, which is a visual programming paradigm that uses blocks to represent the program's instructions. The adoption of a no-code environment allows non-technical users to easily create and modify agents.

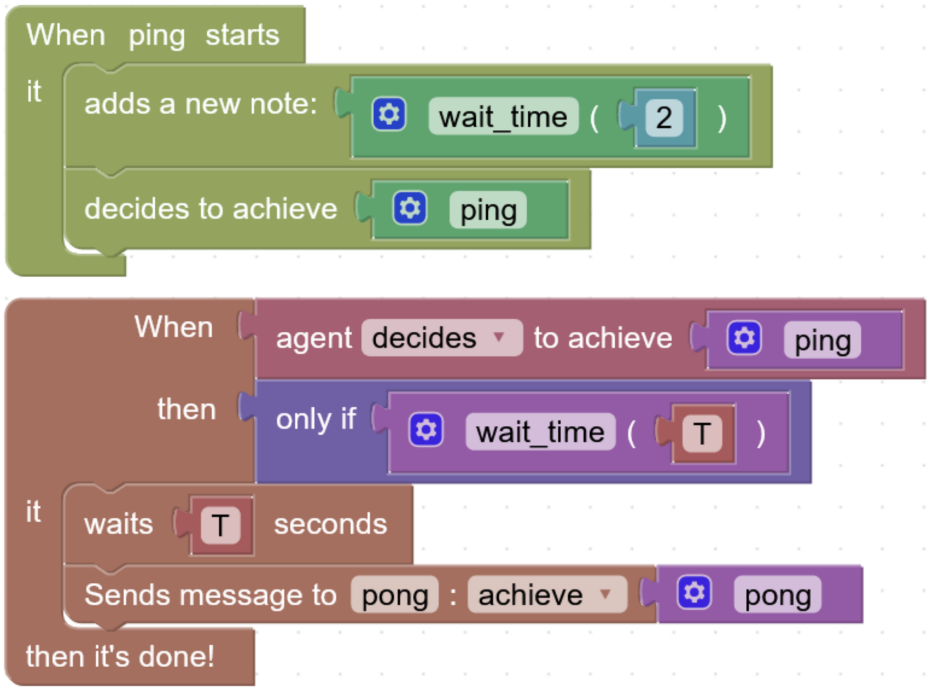


Figure 1.2: Example of a "ping" agent implemented with the block language. Adopted from [8].

An example of this work can be found in fig. 1.2, which shows a simple "ping" agent. When started, the agent has a belief that the `wait_time` is 2 seconds, and it has to achieve the goal `ping`. In addition, the agent is given instructions on how to achieve the goal through a plan. The latter states that when the agent decides to achieve the goal `ping` and knows what the `wait_time` is, it should first wait for the time specified in its belief, then send a message to the agent `pong`.

1.4 Proposing a Visual Programming Paradigm for Organizations

Although the above block-based visual development environment is suitable for defining single entities, a level of abstraction to handle the relations among and coordination of the latter is still missing.

When dealing with multiple agents in a MAS, the complexity of the system increases dramatically and the coordination of and interaction among the agents' becomes more and more challenging. Even though these aspects could be technically represented and managed directly in the mind of the agents,

the adoption of adequate abstractions makes the solution dramatically more straightforward and elegant. [4]

Indeed, multiple approaches to organization in MAS have been reported in the literature. However, all of them require significant coding skills and deep knowledge of the underlying concepts and technologies to be implemented. Thus, none of these approaches is easy enough to be used by non-technical users such as domain experts.

Therefore, the idea is to design and implement a visual language and a supportive development environment for MAS organizations, which is a novelty to our knowledge. The core thesis work will be based on studying the existing organization models, with particular attention to the $\mathcal{M}$OISE$^+$ model [23], and developing an appropriate representation that could provide a suitable tradeoff between the expressiveness of code-based specifications and the ease of use of graphic programming interfaces.

Such a tool would allow domain experts to easily define the organization of a MAS, without the need to write code. From the *IntellIoT* project's perspective, this would allow addressing more complex scenarios, therefore expanding the range of applications of the IoT framework it aims to create. Indeed, for the time being, the applications mainly concern the programming of single agents, which reduces the potential of the technology because of the limited interaction and coordination capabilities that this approach allows.

Moreover, the adoption of a visual language would provide a valuable tool to monitor and debug the system, learn the concepts behind the technology, and hopefully bring new users to the field of MAS.

The thesis project will end with the implementation of a prototype that allows users to define the organization of a MAS by using the developed visual language. In particular, the aspects of the organization that will be covered, and thus the users will be able to specify, are:

- **Structure**: This part is responsible to define how agents should be organized in the system, i.e., how they should be grouped and how they should interact with each other;
- **Behavior**: This aspect regards the definition of the common goals that the agents have to achieve, and how agents should coordinate their actions to achieve them;
- **Norms**: This concept concerns the rules that regulate the agents' behavior. In particular, they represent what agents are and are not allowed to do, or what they are obliged to do.

To evaluate the developed tool, a qualitative evaluation is planned to receive feedback for the developed language and development environment and

to study how non-technical users solve problems by exploiting the visual language. Moreover, the visual language will be used and tested with a real-world scenario to evaluate the correctness of the resulting MAS. This twofold evaluation will allow assessing the user-friendliness of the developed tool on the one hand, and its expressivity and potential on the other.

Before proceeding with the technical description of the designed language and the implemented prototype, the following chapter provides an introduction to the background of the main technologies and research topics explored in the thesis project.

Chapter 2

Background

To better understand this thesis work, a brief excursus of the existing literature about the main concepts the project relies on is presented. The aim is to provide some basic knowledge of the topics to understand what is considered state-of-the-art and to introduce some of the technologies exploited to develop the project.

2.1 Multi-Agent Systems

Multi-Agent Systems are considered a promising engineering style for developing adaptive software systems able to handle the continuous increase in their complexity. Moreover, they allow the design and implementation of software systems using the same ideas and concepts that are the very founding of human societies and habits.

In this section, a brief overview of the main concepts and characteristics of MAS is provided, as well as the main approaches to their design.

2.1.1 What is an Agent?

The concept of a software agent can be traced back to the early days of research into Distributed AI in the 1970s when Carl Hewitt proposed the concurrent Actor model. In his paper, he introduced the concept of a self-contained, interactive, and concurrently-executing object which he termed "actor". The latter is a computational *agent* with a mail address and behavior. Actors can communicate by message-passing and carry out their actions concurrently. [19]

Software agents strongly rely on many of the concepts introduced by the Actor model, such as the idea of a self-contained entity endowed with its control flow that can interact with other entities and the use of message-passing for communication. On top of that, agents bring in autonomous and proactive

behavior as researchers were interested in the development of systems made up of entities with human-like skills such as reasoning, problem-solving, and decision-making.

The term "agent" quickly spread to heterogeneous research fields; therefore, there is no commonly agreed definition for it. However, a generally accepted description of what an agent is is the following by Wooldridge [25]:

> *An agent is a self-contained program capable of controlling its own decision-making and acting, based on its perception of its environment, in pursuit of one or more objectives.*

To sum up, a set of features that an agent should possess can be identified [25]:

- **Autonomy**: agents should be able to perform most of their tasks without the direct intervention of humans or other agents, and they should encapsulate control over their actions and internal state
- **Social ability**: agents should be able to interact with each other and possibly humans to complete their tasks.
- **Responsiveness** (situatedness): agents should perceive their environment and respond to changes in it.
- **Proactiveness**: agents should exhibit opportunistic, goal-directed behavior and take the initiative when appropriate.

Since the first years, the research concentrated on interaction and communication among agents, decomposition and distribution of tasks, and coordination and cooperation. The goal was to specify, analyze, design, and integrate systems containing multiple collaborative agents.

2.1.2 From the Individual to the Collective

Multi-Agent Systems (MAS) have been studied as a per se field since the 1980s and gained widespread recognition in the 1990s. Since then, international interest in the topic has grown enormously as agents are considered an appropriate paradigm to exploit the possibilities presented by massive open distributed systems. Moreover, MAS seem to be a natural metaphor for understanding and building a wide range of what might be called *artificial social systems*. [37]

According to the *Alan Turing Institute* [24]

> *A Multi-Agent System consists of multiple decision-making agents which interact in a shared environment to achieve common or conflicting goals.*

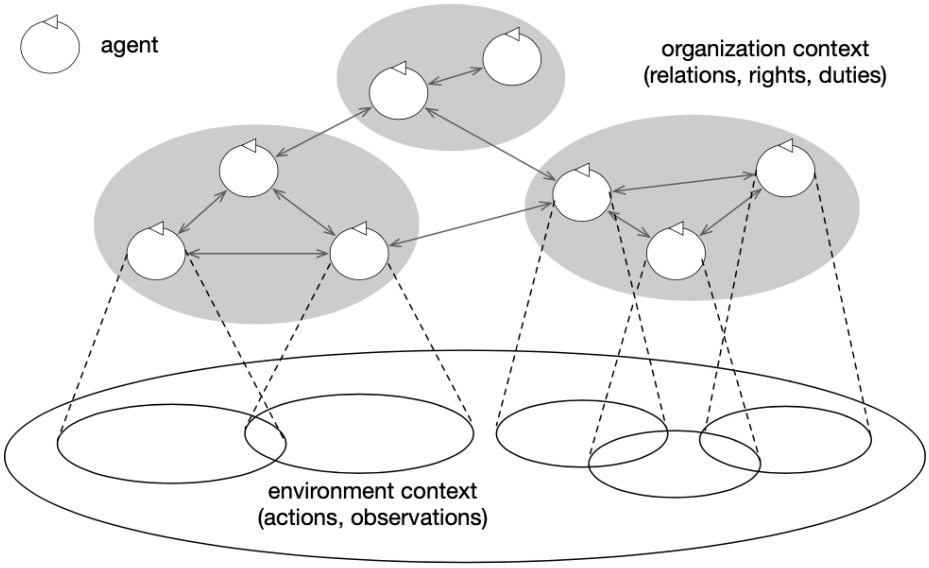


Figure 2.1: A representation of Multi-Agent Systems. [26]

Therefore, as it can also be noticed in fig. 2.1, MAS are composed of an environment and agents existing within it that are bonded by relations.

2.2 Multi-Agent Oriented Programming

In principle, there is no constraint on the programming technologies that can be used to implement MAS. However, the risk is to have an agent-centered interpretation of the system in which the environment and/or the organization contexts are represented and managed in the mind of the agents. Therefore, the adoption of programming languages and paradigms that directly support first-class abstractions for these contexts highly simplifies the task of designing and implementing MAS and makes it possible to keep the level of abstraction coherent from design time to development time and finally also at runtime.

Multi-Agent Oriented Programming (MAOP) is an approach to programming MAS that promotes the use of first-class programming abstractions that concern three main dimensions that characterize these systems: [4]

- **Agent dimension**: it concerns the concepts and programming abstractions for the definition of the agents that participate in the system. Specifically, it allows the definition of software entities with their logical thread of control, which can autonomously and proactively achieve their goals, and interact with the environment, other agents, and the organization that regulates the system.

- **Environment dimension**: it offers concepts and abstractions to define the distributed resources and connections to the world shared among the agents. Thus, the environment abstraction is what makes the agents situated and provides them with tools that help them achieve their goals.
- **Organization dimension**: it collects all the concepts regarding the definition of relations, shared tasks, and policies among agents (inter)acting in a shared environment. In open systems, the organization is fundamental for coordination and regulation among agents.

Since agents have already been discussed, the following sections provide a description of the latter two dimensions, with a particular focus on the one regarding the organization, as it is the core of this thesis work.

2.2.1 Environment in Multi-Agent Systems

The environment in MAS plays a dual role [36]:

- *The "external world"*: agents become aware of the context they are immersed in and its dynamics by perceiving the environment through sensors. Moreover, they pursue their goals through actions performed by actuators that aim at modifying the environment, eventually reaching the latter's desired state.
- *A medium for coordination*: agents exploit the environment to share information and coordinate their behavior. Each agent follows simple behavioral rules, resulting in a collective behavior that is more complex than the sum of the individual behavior; this pattern resembles stigmergic systems in which agents coordinate their behavior through the manipulation of marks.

The environment not only enables the agents to interact with the deployment context, which they can access through sensors and actuators but also provides them with external resources that they can exploit to achieve their goal.

The *Agents & Artifacts* (A&A) conceptual framework [30] argues that, just like in human society, MAS environments should contain different kinds of objects, tools, and artifacts in general that agents can use to support their activities. This vision also constitutes a revolution from an engineering perspective as it encourages system designers to model the environment as a set of *artifacts*, each of which encapsulates its intended purpose and exposes its observable state. Moreover, the A&A meta-model provides an effective abstraction level that shields low-level details of the deployment context, so that designers can focus on the agents' behavior.

2.2.2 Organization in Multi-Agent Systems

An agent organization can be defined as a social entity composed of a specific number of agents that accomplish several common tasks or goals and that are structured following some specific topology and communication interrelationships to achieve the main aim of the organization. All MAS possess some form of organization, although it may be implicit and informal.

Approaches to Organization in MAS

There are two approaches to organizing agents in a MAS. [1] The first one regards *agent-centered* MAS, in which the focus is given to individual agents. According to this viewpoint, the designer should concern about the local behavior of the agents and their interactions without worrying about the global structure and goal of the system as the latter should emerge as a result of the lower-level individual interactions in a bottom-up fashion. The key issues of this approach are unpredictability and uncertainty since it could lead to undesirable emergent behaviors. As Weyns [35] stated, giving the responsibility of system organization implicitly to individual agents is highly complex and not suitable for real-world large-scale scenarios.

The second approach is *organization-centered* MAS, in which the structure of the system is given greater attention. The developer designs the entire organization and coordination patterns on the one hand, and the agents' local behavior on the other. This can be seen as a top-down approach as the organization abstractions impose constraints on the agents and regulate their interactions, simplifying the design of complex and scalable systems and allowing more accurate modeling of the problems being tackled. On top of that, the organization-centered approach avoids the emergence of undesirable behaviors such as divergence. Indeed, larger MAS bring a higher risk of divergence, therefore the need for explicit organizational regulation.

Organizational Paradigms

Although no two organizational instances are likely to be identical, there are identifiable classes of organizations, that emerged from research and real-world applications, which share common characteristics. These classes are called *organizational paradigms* and cover particularly common, useful, or interesting structures that can be described in some general form. Here is provided a brief overview of the most common paradigms [20]:

- **Hierarchies**: agents are conceptually arranged in a tree-like structure, where agents higher in the tree have a more global view than those below

them. Interactions only take place between connected entities; data produced by lower-level agents in a hierarchy typically travels upwards to provide a broader view, while control flows downwards as the higher-level agents provide directions to those below.

- **Holarchies**: agents are organized in holons. The term *holon* comes from the Greek words *holos*, meaning "whole", and *on*, meaning "part". Therefore, a holon is a self-contained entity that can be considered both as a part of a larger entity and as a whole in itself, and that has a character derived but distinct from the entities that make it up and at the same time it contributes to the properties of a greater whole. Examples showed how holarchies can be used to effectively model the division of labor in MAS, where capabilities and tasks were imparted to holons instead of single agents. This results in a layer of abstraction that allows other entities to interact with a group as a single functional unit.

- **Coalitions**: they are formed by agents that share a common goal and that are willing to cooperate to achieve it and are generally short-lived as they are formed with a purpose in mind and dissolved when that need no longer exists. Although there may be a distinguished "leading agent", within a coalition the structure is typically flat. However, since once formed coalitions may be treated as a single entity, it is possible to form a hierarchical structure by nesting coalitions.

- **Teams**: they consist of several cooperative agents that agreed to work together towards a common goal. Unlike coalitions, teams attempt to maximize the performance of the group as a whole, rather than the performance of individual agents. This is usually achieved by assigning roles to the agents, which become responsible for specific tasks, and by providing the agents with representations of the shared goals, knowledge, and plans.

- **Congregations**: although similar to the latter two structures, they differentiate because they are assumed to be long-lived and are not necessarily formed with a specific goal in mind. Indeed, congregations are formed among agents with similar o complementary characteristics to facilitate the process of finding partners for collaborations.

- **Societies**: they are open systems where agents of different kinds may come and go at will while the society persists, acting as an environment through which the participants meet and interact. Societies impose on agents a set of constraints which are known as *social laws*, *norms*, or

conventions. These represent rules by which agents must act and provide a level of consistency in behavior that facilitates the coexistence of possibly heterogeneous agents.

- **Federations**: they are groups of agents which have ceded some amount of autonomy to a single delegate who represents the group. The members of the group interact only with the delegate, who accepts skills and needs descriptions from them, which are then used to match with requests from delegates representing other groups.

- **Markets**: buying agents may request or place bids for a common set of items, such as shared resources, tasks, services, or goods, or even supply items to the markets to be sold. On the other hand, sellers are responsible for processing bids and determining the winner.

- **Matrix**: they can be seen as a relaxation of the one-agent, one-manager restriction in hierarchical organizations, that permit many managers to influence the activities of an agent.

All of the above structures come with their benefits and drawbacks and it is generally agreed that there is no single type of organization that is suitable for all situations. Indeed, sometimes two or more organizational paradigms may be combined to form a compound organization, exploiting features of each of the component organizations.

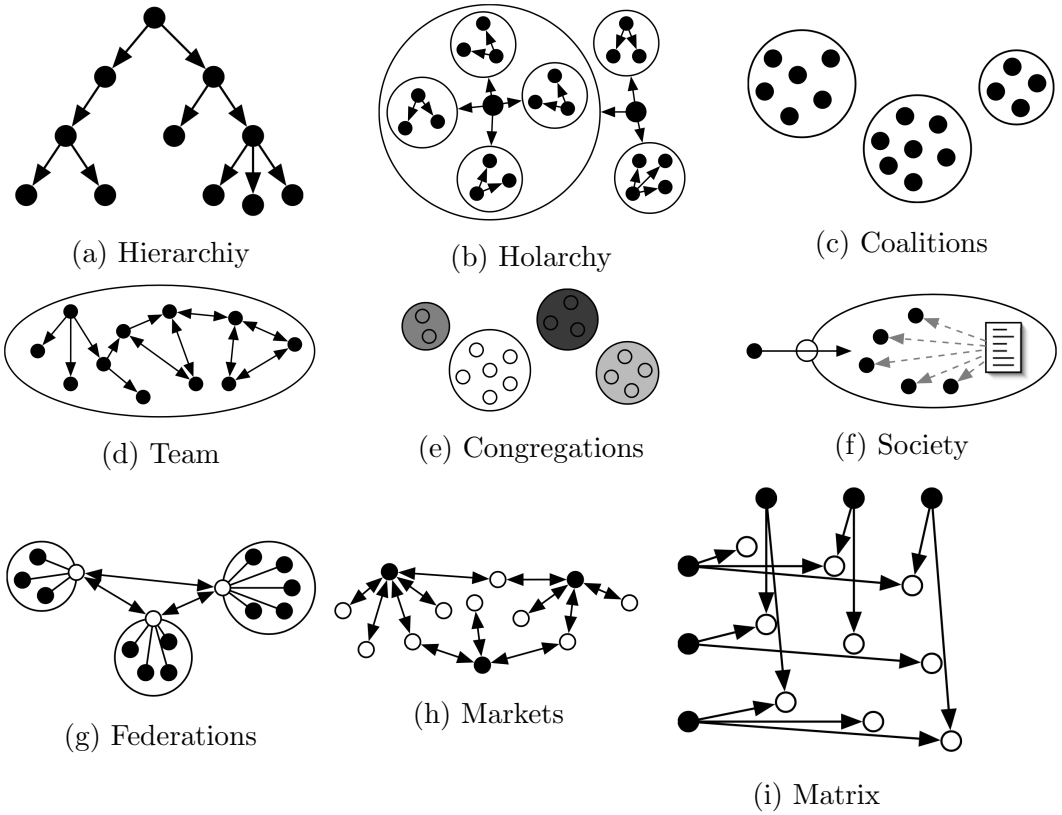
2.2.3 The JaCaMo Platform

Regarding the engineering and implementation of MAS, one of the reference technologies is the JaCaMo platform [5], which supports practical programming based on the first-class abstractions introduced before to develop organized agents situated in a shared environment. Therefore, the platform gives convenient tools to program agents, their environment, and the organizations they belong to. JaCaMo is built on top of three other existing platforms:

- ***Jason***: provides a programming language to code autonomous intelligent agents based on the BDI (Belief, Desire, Intention) architecture [6][7].

- `CArtAgO`: as the way to define the environment in which agents will be situated, following the *A&A* metamodel [29].

- $\mathcal{M}$OISE: based on the $\mathcal{M}$OISE$^+$ model [23], it allows the explicit definition and management of organizations within the systems [22].



(a) Hierarchiy (b) Holarchy (c) Coalitions

(d) Team (e) Congregations (f) Society

(g) Federations (h) Markets (i) Matrix

Figure 2.2: Visual representation of the organizational paradigms.

Since the JaCaMo platform was chosen as the enabling tool supporting the instantiation of the organizations built with the visual language developed, the latter takes inspiration from the $\mathcal{M}$OISE$^+$ model and its concepts, which are briefly described in the following section.

The $\mathcal{M}$OISE$^+$ Model

Just like the $\mathcal{M}$OISE model [18], which it extends, the $\mathcal{M}$OISE$^+$ model aims at providing a way to cope with both the agent-centered and organization-centered approaches to the design of MAS. This way, the ability to manage the complexity taken from the organization-centered approach, and the flexibility taken from the agent-centered approach, can be combined to face the constantly growing complexity of MAS applications. Indeed, the MAS designer can specify an *organization specification* (OS), which defines an "a priori" set of constraints and cooperation patterns imposed on the agents; on the other hand, agents themselves can reason about and modify the *organization entity* (OE), which is the actual instantiation of the organization on the agents.

In addition, the $\mathcal{M}$OISE$^+$ model proposes an organizational modeling language that explicitly decomposes the specification of organizations into *structural*, *functional*, and *deontic* dimensions [22][23].

The structural dimension specifies the *roles*, *groups*, and *links* of the organization. The definition of roles states that when agents decide to play a role, they are accepting some behavioral constraints related to the role.

The functional dimension specifies how the *global collective goals* should be achieved. Specifically, how these goals are decomposed in *plans*, grouped in coherent sets by *missions*, and how are distributed to the agents. The decomposition of global goals results in a goal tree, called *scheme*, where the leaf goals can be achieved directly by the agents.

Finally, the deontic dimension serves as a binding between the structural and functional dimensions, specifying the roles' *permissions* and *obligations* for missions.

The infrastructure adopted by JaCaMo for the $\mathcal{M}$OISE$^+$ model is called ORA4MAS (Organizational Artifacts for Multi-Agent Systems) [21] which conceives *organizational agents* that control, manage and adapt the organization operating on *organizational artifacts*, thus adhering to the *A&A* metamodel, such as OrgBoard, GroupBoard, SchemeBoard, and NormativeBoard.

The next section illustrates how the above artifacts can be used in the deployment of an organization.

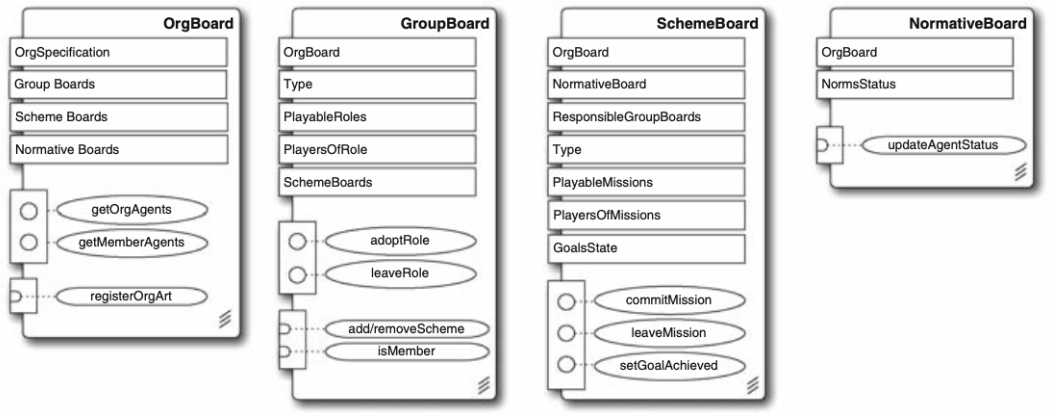


Figure 2.3: Organizational artifacts in ORA4MAS with their interface including operations, observable properties, and link interfaces. Adopted from [21].

ORA4MAS in Action

When a set of agents wants to coordinate their actions to achieve a common goal, they can do it, for instance, through the following steps:

1. One of the agents, which is also an *organizational agent*, creates an OrgBoard artifact based on a specification.

2. The organizational agent creates a GroupBoard artifact for each group of agents that will be part of the organization. Once the GroupBoard is created, the artifact registers itself in the OrgBoard exploiting the latter's link interface.

3. All the other agents get notified about the new artifacts and therefore may decide to adopt a role in one of the groups. They can do so using the adoptRole operation of the GroupBoard artifact.

4. The organizational agent can now create a SchemeBoard artifact to start the organization's collective goals. As for the GroupBoard, the SchemeBoard artifact registers itself in the OrgBoard. As every SchemeBoard has one NormativeBoard, the latter is created automatically and linked to the former.

5. Once the SchemeBoard is created, obligations and permissions are computed and verified by the NormativeBoard. Agents can now commit to their missions according to the NormativeBoard rules.

6. Once the scheme is well formed, the goals of the scheme can be achieved by the agents.

2.3 Hypermedia Multi-Agent Systems

The current technological landscape brings new challenges to the engineering of MAS such as the support of large-scale open systems, and the support of heterogeneous agents and humans in the loop.

2.3.1 The World Wide Web

The Web has had remarkable success as a worldwide and long-lived system of people, providing them with a *distributed hypermedia environment*, composed of interrelated Web pages, that they can navigate and use in pursuit of their goals.

No doubt REST (REpresentational State Transfer), the architectural style of the Web [15], is one of the enabling factors of the above characteristics. REST consists of a set of architectural constraints, such as *client-server* and *stateless* interaction, and *uniform interface* [16]. The latter principle is fundamental for RESTful systems, as it simplifies and decouples the Web architecture, and it is achieved through four constraints:

- *Identification of resources*: each Web-based concept is modeled as a resource identified by a URI.
- *Manipulation of resources through representations*: clients manipulate representations of resources. The same resource can be represented in different ways, e.g. as HTML, XML, or JSON. The key point is that the representation is a way to interact with a resource but it is not the resource itself.
- *Self-descriptive messages*: each message includes enough information to describe how to process the message.
- *Hypermedia as the engine of application state (HATEOAS)*: the representation of a resource includes links that the client can use to dynamically discover other resources, therefore enabling *hypermedia-driven interaction* [32].

According to HATEOAS, a typical Web application, which is composed of multiple hyperlinked Web pages, can be seen as a finite state machine where each page represents a state and hyperlinks between pages represent transitions between states. Indeed, given the URI of a page, a client can dereference the URI to retrieve an HTML representation of that page. This action triggers a transition to a new application state which provides the client with a new set of reachable states in the form of hyperlinks to other Web pages. Similarly,

the client can send a request to the server to update a resource, thus triggering a transition to a new state.

The key point is that both the next reachable states and the knowledge required to transition to those states are conveyed to the client through hypermedia. Therefore, a client should be able to discover new resources and how to use them at runtime, allowing components to be deployed independently from one another.

2.3.2 The Web of Things

The Web of Things (WoT) [33], first introduced in 2007 [17], is a set of W3C standards that aim to improve the interoperability and usability of the Internet of Things (IoT). The idea is to apply the architectural principles and standardized technologies of the Web to integrate the different technological stacks used by the current IoT *things*.

One of the fundamental building blocks of the WoT is the *Thing Description (TD)* [34] that acts as a machine-readable manual for the interaction with the *thing* it describes. The TD is based on the concept of *interaction affordances* which refer to the perceived and actual properties of the *thing* that determine how the latter can be used. The three types of affordances are:

1. **Properties**: they represent the state of the *thing* that can be read and/or written.

2. **Actions**: they allow the invocation of a function of the *thing* which manipulates its state or triggers a process.

3. **Events**: they describe an event source that asynchronously pushes event data to the observers.

The affordances of a *thing* are intended in the hypermedia perspective of presenting information and control, suggesting to the clients the possible choices for interaction and how to use them in the form of hyperlinks.

2.3.3 Web-based Multi-Agent Systems

There has been extensive research on using the Web as an infrastructure for distributed MAS. The early approaches usually fall into one of the following two categories:

- **The Web as a Transport Layer**: these systems use HTTP as a transport layer for the communication between agents; thus, they make limited use of the architectural properties of the Web.

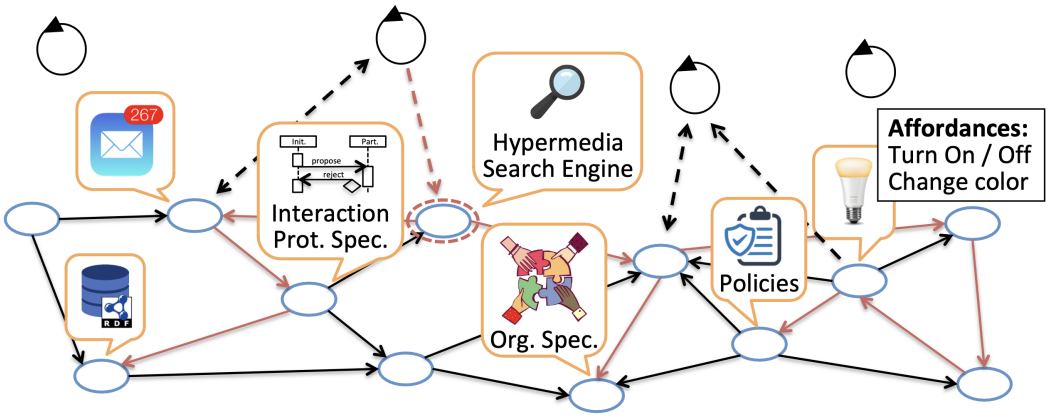


Figure 2.4: Hypermedia MAS. Adopted from [11].

- **The Web as a non-Hypermedia Application Layer**: agent services are translated into Web services, which expose REST-like interfaces. Even tho these systems typically respect the first three uniform interface constraints, they do not support HATEOAS and therefore hypermedia-driven interaction, making clients and servers tightly coupled to one another, which is an important limitation when engineering large-scale, open systems.

The premature development of these approaches, not completely adhering to the Web architectural style, and the lack of crucial initiatives such as the Web of Things, together with the shortage of real-world applications, have hindered their widespread acceptance.

2.3.4 The Bridge between the Web and Multi-Agent Systems

Integrating the environment in multi-agent systems with the Web architecture helps bridge the gap that previously existed between MAS and the Web [11]. Hypermedia MAS are systems composed of both people and autonomous agents situated in a shared *hypermedia environment* that is distributed across the Web and thus becomes an *hypermedia application* [10]. According to this approach, all the entities, both agents and artifacts, are Web resources and their representations can be related by hyperlinks.

In contrast to typical environments, a hypermedia environment uses hypermedia to drive interaction in the MAS: agents navigate and crawl the environment at runtime to discover other entities in the MAS, as well as the means to interact with them, in an analogous approach to the one used in the Web

of Things through the Thing Description. This reduces coupling and enhances the scalability and evolvability of the systems.

The three key design principles meant to ensure the proper use of hypermedia as a general mechanism for uniform interaction in MAS are the following [9]:

1. **Uniform resource space**: all entities in a hypermedia MAS and relations among them should be represented in the hypermedia environment in a uniform, resource-oriented manner. For instance, one agent could send a message to another by writing an RDF representation of the message in the hypermedia. To receive messages, an agent could observe a resource that represents its mailbox in the hypermedia. To turn on a light, an agent could manipulate the state of a resource that represents the light bulb in the hypermedia. Anyways, interactions between agents and resources in their hypermedia environment should conform to the REST constraints.

2. **Single entry point**: given a single entry point into the environment of a hypermedia MAS, an agent should be able to discover the knowledge required to participate in the system by navigating the hypermedia. The core idea is to minimize coupling by enabling system-wide discoverability as agents can crawl the hypermedia to discover what other agents, tools, or entities in the system can help them achieve their goals. Equally important, agents can also discover in the hypermedia how to interact with entities through their affordances.

3. **Observability**: in a hypermedia MAS, any resource in the hypermedia environment that could be of interest to agents should be observable. While the first two principles ensure the dynamic discovery of a hypermedia MAS via crawling, the latter promotes the use of mechanisms that allow agents to selectively observe entities of interest. This is important to improve the scalability and handle larger environments.

Engineers can choose to ignore one or more of these principles, but the MAS would most likely make limited use of the hypermedia and would not achieve uniform interaction, hindering the scalability and openness of the system.

After having introduced the main concepts this thesis work is based on, the next chapter goes into more detail about the $\mathcal{M}$OISE framework and its abstraction that will help shape the requirements for the proposed solution.

Chapter 3

Requirements

As stated in chapter 1, the formulated proposal was to first design a user-friendly visual programming paradigm that domain experts could use to specify MAS organizations and then implement a prototype web-based integrated development environment (IDE) that would allow users to exploit the proposed paradigm and enforce the resulting organizations on the agents.

When designing the requirements for the proposed platform, some assumptions were made and constraints were imposed.

The platform will be used by domain experts, who have no or very little knowledge of programming and software development. However, they are expected to have a deep understanding of the domain they are working in, which means that they can specify the requirements of the organization they want to implement.

The platform needs to be web-based to provide continuity and coherence and allow seamless integration with the already existing platform for the development of agents.

Finally, but most importantly, the platform needs to be compliant with the $\mathcal{M}$OISE$^+$ model, as the JaCaMo framework is currently exploited by some of the participants of the *IntellIoT* project, among them the *University of St. Gallen*. Since the $\mathcal{M}$OISE component of JaCaMo is of crucial importance for the project and to better understand this project's requirements, its features are hereby presented and explained in more detail.

3.1 $\mathcal{M}$oise Features

Organizing a MAS is a process that starts with a *definition* phase, carried out by the designer during the development of the system, followed by an *execution* phase, in which the agents behave under the constraints imposed by the organization. It is worth mentioning that this process may also include

iterative interleaving of the former phases undertaken by the agents through reasoning on their collective behavior, indeed producing a *self-organization* phase. However, considering the two main phases, $\mathcal{M}$OISE distinguishes between the *organization specification (OS)* and the *organization entity (OE)*.

The OS is a declarative description of the organization that answers the *what* questions, such as *what are the roles?*, *what are the collective goals?*, etc., and defines the expected behavior to be produced by the agents.

On the other hand, the OE corresponds to the enactment of the OS by the agents and describes the evolving state of their coordinated and regulated behavior.

The definition of an organization may cover several aspects of the collective activity of the MAS. Below is an explanation of the dimensions that can be specified in a $\mathcal{M}$OISE organization.

3.1.1 Structural Dimension

Here are presented the main structural abstractions that allow the definition of the structure of an organization.

A *role* represents the position that an agent can occupy in the organization and it is identified by a unique label. An inheritance relation is also supported, enabling the reuse of properties attached to the inherited role similar to what happens in object-oriented programming.

A *group* represents a possible community of agents. They are also identified with a unique label and can be nested to form a hierarchy of groups. Each group is composed of roles, links between those roles, possibly other subgroups, and a set of formation constraints. The group-formation constraints define expected properties such as:

- *role compatibility*: it is a directed relation that enables an agent to adopt the target role while already playing the source role.
- *role cardinality*: defines upper and lower bounds on the number of agents that can play a given role in the group.
- *group cardinality*: defines upper and lower bounds on the number of subgroups entities that can be instantiated from the subgroup defined within the group.

Finally, a *link* represents the type of interaction that can take place among agents in a group when playing a role. The current version of $\mathcal{M}$OISE supports *communication*, *authority*, and *acquaintance* links which, namely, state who can communicate with whom, who has authority over whom, and who can represent and access information from whom.

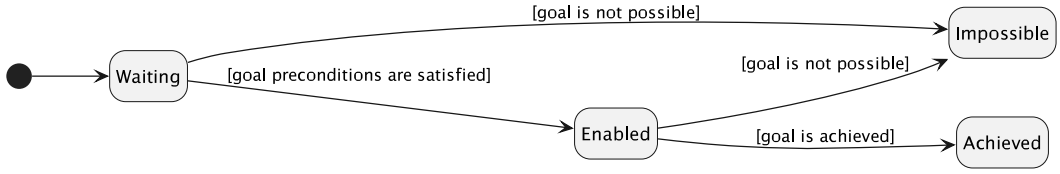


Figure 3.1: Goal life cycle.

3.1.2 Functional Dimension

As for the structural abstractions, here are presented the concepts that allow the definition of the behavior of the agents within an organization.

An *organizational goal* abstracts a state that has to be satisfied by one or several agents. Goals are denoted by an identifier and are arranged in a tree structure called *goal decomposition tree* where the root is a global goal and the leaves are goals that can be satisfied by the agents. During the execution phase, the goals can be in one of the following states:

- *waiting*: the goal cannot be pursued yet because it depends on the satisfaction of other goals;
- *enabled*: the goal can be pursued by the responsible agents;
- *achieved*: the agents responsible for the goal have been able to achieve it;
- *impossibile*: the agents responsible for the goal concluded that they will not be able to achieve it.

Each non-leaf goal is decomposed into subgoals by *plans* using three operators:

- *sequence*: the plan $g1 = g2, g3$ means that the goal $g1$ is satisfied if and only if $g2$ and subsequently $g3$ are satisfied;
- *choice*: the plan $g1 = g2|g3$ means that the goal $g1$ is satisfied if one and only one of $g2$ or $g3$ is satisfied;
- *parallel*: the plan $g1 = g2 \parallel g3$ means that the goal $g1$ is satisfied if both $g2$ and $g3$ are satisfied, and they can be pursued in parallel.

Therefore, a social plan denotes a structure of interrelated organizational goals to be satisfied by multiple agents that have to coordinate to handle dependencies between goals.

Goals can be gathered together in *missions*, meaning that they should be achieved under the responsibility of a single agent. When an agent participates in the organization, it commits to missions, meaning that it will try to achieve the goals contained in them.

All of the above concepts regarding functional abstractions make up a *social scheme*, which denotes the collective and coordinated behavior that is expected to be produced by a group of agents in the organization.

3.1.3 Normative Dimension

Whereas structural abstractions address the structuring of the agents in the system and functional abstractions target the coordination of their behavior, normative abstractions are concerned with the regulation of agents' behavior in the organization

The main concept is the *norm*, which defines the rights and duties of agents by connecting the structural and the normative dimensions with deontic modalities such as *obligation*, *permission*. Specifically, a norm refers to a role and a mission, thus obliging or permitting the agent playing the role to commit to the mission. Moreover, a norm can be associated with an *activation condition* that must be satisfied in order for the norm to be active, and a *time constraint* that defines a deadline by which the norm must be satisfied.

3.1.4 Organization Execution

As introduced in section 3.1, the organization entity is a representation of the state of the organization at runtime which is distributed into three types of entities. The concept is similar to the one of *classes* and *objects* in object-oriented programming.

The *group entity* is related to a group defined in the structural specification and its state contains:

- the owner of the group;
- the links to children and parent groups;
- the set of role-player agents with their links.

The *scheme entity* is related to a social scheme defined in the functional specification, including:

- the owner of the scheme;
- the groups responsible for the scheme;

- the commitments of the agents to the missions;
- the state of each goal.

Finally, the *normative entity* is related to group and social scheme entities. It is created every time a group entity becomes responsible for a scheme entity and it contains the status of a set of norms build from the abstract norms of the normative specification.

3.2 Functional Requirements

Since the platform needs to be compatible with $\mathcal{M}$OISE, its features should be available in the new platform as well.

Structure Users should be able to define the structure of the organization. In particular, the abstractions that allow its definition are the roles that agents can play, the groups that can contain roles and possibly nest other subgroups, and the links between roles. As far as the links are concerned, the core subset of $\mathcal{M}$OISE should be supported, i.e. extension and compatibility.

Behavior Users should be able to define the expected behavior of the agents. Specifically, they will be provided with a way to define the collective goals that agents should achieve and the dependencies among them. The type of dependencies supported is *finish-to-start*, meaning that the target goal cannot be pursued until the source goal is achieved.

Goal Assignment Users should be able to assign collective goals to roles, thus connecting the structure and the behavior of the organization. Either one, more, or no role can be assigned to a goal and the assignment may have a *obligation* or *permission* deontic modality.

Persistence Users should be able to save and load the organizations they create for future editing. Indeed, the platform will provide a way to edit organizations previously created since defining one usually requires multiple iterations and changes.

Deployment Users should be able to enforce the organization they create on running agents. In particular, they will choose the agents that will be part of the organization and the roles they will play.

3.3 Non-Functional Requirements

Given the target user of this platform, i.e. domain experts, the main non-functional requirement is the ease of use and user-friendliness. In particular, the interface should be as intuitive as possible, conveying the possibility of designing agent organizations in a way that typical users are expected to easily understand.

What is more, the system should be easily accessible. Therefore, the most suitable technologies for its development are web-based ones, which allow the user to access the platform from any device with an internet connection. Although, a mobile version is not required since it will mainly be used on wide screens.

The above functional and non-functional requirements, that arose from the *IntellIoT* project and the need to address a higher level of abstraction for the definition of organization-centered MAS by domain experts, will be therefore fulfilled through the development of a visual programming language and a web-based integrated development environment that exploits it.

Chapter 4

Design

The following chapter describes the details concerning the design process of the system.

The whole process was carried out with a step-by-step approach, starting from the design of the visual language and early prototypes of the development environment, proceeding with the storage backend, and finally concluding with the integration with the execution backend.

Specifically, the design of the visual language and the development of the web-based IDE prototypes were carried out iteratively, in order to adopt a PDCA (Plan-Do-Check-Act) approach. The latter allowed receiving realistic feedback straight away, thus intercepting potential problems early on and speeding up the entire process.

Due to the hard time constraint of the internship and to obtain ease of use, although sometimes trading off with expressivity, only a core subset of $\mathcal{M}$OISE features were implemented. This also implies that the system was not designed as a complete replacement of $\mathcal{M}$OISE, but rather as a tool to facilitate the use of this technology also by users with no programming skills.

Moreover, as some parts of the system were designed knowing that they might need a full project focused only on them, this project leaves space for future improvements and integrations, such as the implementation of the remaining features and the exploitation of different technologies.

4.1 A Visual Language for Organizations

The main challenge in this project is surely the design of a visual language that, on one hand, is expressive enough to represent the organizational structure and the goals of an organization, and, on the other hand, is easy to use and understand by non-technical users.

Since the visual abstractions are the main artifact of the system, the design of the language was the first step of the project and has gone through several iterations, possibly even evolving further in the future thanks to feedback from real users.

4.1.1 The Visual Paradigm

The choice of the most suitable visual paradigm was the first step of the design process since it is crucial to define the overall look and feel of the language. Indeed, it is of fundamental importance to choose a paradigm that is easy to use, allows the users to easily understand the meaning of the visual abstractions, is coherent with the concepts to represent, and can be easily translated into the actual specification.

Comparing the existing paradigms for visual programming, the most natural choice was the diagram-based paradigm. Other approaches, even though already proven to be effective in many fields, were not perfectly suitable for this purpose. For instance, flow-based programming better fits the modeling of the way data must flow during the execution of the program, while block-based programming is more suitable to describe instructions to be executed in an imperative programming style.

On the other hand, the diagram-based paradigm is appropriate to specify the characteristics of a system in a declarative programming fashion, therefore being highly convenient for the definition of an organizational specification.

4.1.2 Reference Language

To facilitate the design of the visual language, it was necessary to study and understand the organization specification language in order to identify the "building blocks" that could be used to represent the organizational structure and the goals of an organization. Since, as already mentioned, the system has $\mathcal{M}$OISE as a strong constraint and reference, as the organization specifications created should be compliant with it, the $\mathcal{M}$OISE language was chosen as a reference.

The analysis of the concepts and the syntax of the reference language, to be ported as visual elements, first involved looking at the $\mathcal{M}$OISE XML metamodel that defines the rules for the organization specification syntax. A few examples of the main constructs are shown in listing 4.1[1].

As can be observed, a role definition requires the specification of an attribute `id`, that is the name of the role, and it may specify some roles it

[1]The entire metamodel can be found at `https://github.com/moise-lang/moise/blob/master/src/main/resources/xml/os.xsd`

extends from through `extends` children elements. On the other hand, a goal definition requires the specification of the attributes `id`, that is the name of the goal, `ds`, that is the description, etc. and it may specify arguments, dependencies from other goals, and plans, with `argument`, `depends-on`, and `plan` children elements, respectively.

```
<xsd:complexType name="roleDefType">
    <xsd:sequence>
        <xsd:element maxOccurs="1" minOccurs="0" name="properties"
            type="moise:propertiesType"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
            name="extends">
            <xsd:complexType>
                <xsd:attribute name="role" type="xsd:string"/>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
</xsd:complexType>

 <xsd:complexType name="goalDefType">
    <xsd:sequence>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
            name="argument" type="moise:argumentType"/>
        <xsd:element maxOccurs="unbounded" minOccurs="0"
            name="depends-on" type="moise:dependOnType"/>
        <xsd:element maxOccurs="1" minOccurs="0" name="plan"
            type="moise:planType"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
    <xsd:attribute name="min" type="xsd:nonNegativeInteger"
        use="optional"/>
    <xsd:attribute name="ds" type="xsd:string" use="optional"/>
    <xsd:attribute name="type" type="moise:goalType"/>
    <xsd:attribute name="ttf" type="xsd:string" use="optional"/>
    <xsd:attribute name="location" type="xsd:string" use="optional"/>
</xsd:complexType>
```

Listing 4.1: $\mathcal{M}$OISE syntax rules for the XML specification of roles and goals.

The actual syntax used in the XML organization specification for roles and goals definition can be found in listing 4.2. As can be observed, there are two roles defined, `role1` and `role2`, with the latter extending from the former. As

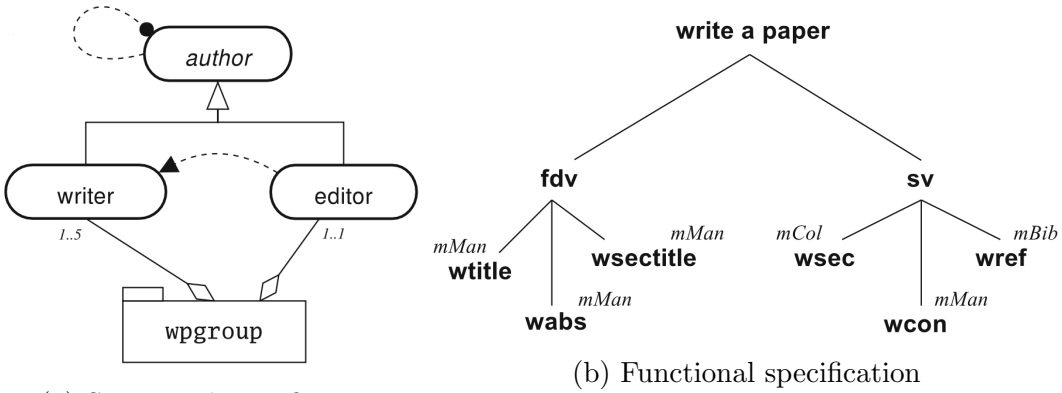
far as the goals are concerned, there are four of them in total; the goal `goal2` has a description, the argument `arg1`, depends on the goal `goal1` and has a plan that consists of the sequence of the two goals `goal3` and `goal4`, that is $g2 = g3, g4$.

```
<role id="role1" />
<role id="role2">
    <exends role="role1" />
</role>

<goal id="goal1" />
<goal id="goal2" ds="description for goal2">
    <argument id="arg1" />
    <depends-on goal="goal1" />
    <plan operator="sequence">
        <goal id="goal3" />
        <goal id="goal4" />
    </plan>
</goal>
```

Listing 4.2: $\mathcal{M}$OISE actual syntax for the XML specification of roles and goals.

Moreover, since a graphical representation of some of the $\mathcal{M}$OISE constructs is already available, it was possible to use them as a reference and starting point for the design of the visual language.



(a) Structural specification

(b) Functional specification

Figure 4.1: Visual representation of the structural and functional specification in $\mathcal{M}$OISE. Adopted from [21].

The scenario described in fig. 4.1 considers agents that aim at writing a paper and therefore have to collaborate.

As shown in fig. 4.1a, the structure of the organization has only one group, that is the `wpgroup` represented by a folder, with two roles defined, `writer` and `editor` represented by rounded rectangles, that extend, and therefore inherit the properties of, the `author` role. The relations and links among roles and groups are represented by arrows, with the pattern of the line and the different harrow heads indicating the type of relation. Indeed, a solid line with an empty diamond head indicates that the role is a member of a group, i.e. `writer` and `editor` are members of the `wpgroup` group. On the other hand, a solid line with an empty harrow head indicates that the source role extends the target role, i.e. `writer` and `editor` extend the `author` role. Moreover, a dashed line with a round head indicates that the source role can communicate with the target role, i.e. `author` can communicate with `author`, which in this scenario means that every agent can communicate with every other agent. Finally, a dashed line with a filled harrow head indicates that the source has authority on the target, i.e. `editor` has authority on `writer`.

As far as the functional specification shown in fig. 4.1b is concerned, the goals of the organization are represented in a tree-like structure, thus following the goal decomposition tree abstraction, with the root goal being the `write a paper` goal. The latter is decomposed into two subgoals, `fdv` (first draft version) and `sv` (submission version), through a sequence plan. What is more, the `fdv` and `sv` goals are decomposed through sequence plans into three subgoals each, that are `wtitle`, `wabs`, and `wsectitles` and `wsec`, `wcon`, and `wref`, respectively. Therefore, the plans can be formalized as:

$$\texttt{write a paper} = \texttt{fdv}, \texttt{sv}$$
$$\texttt{fdv} = \texttt{wtitle}, \texttt{wabs}, \texttt{wsectitles}$$
$$\texttt{sv} = \texttt{wsec}, \texttt{wcon}, \texttt{wref}$$

4.1.3 Focus Group

To better understand how people perceive an organization from a visual point of view, a focus group was organized. The latter was composed of 5 people, all members of the research group in St. Gallen and therefore with a background in computer science.

However, four of them were not familiar with the $\mathcal{M}$OISE tool since they had never used it, while one of them uses it on a nearly daily basis. This choice was thought to be the most appropriate since it allowed having most of the feedback from non-biased users and one from an experienced and knowledgable user of MAS organizations.

The members of the focus group were asked to perform a task that involved formalizing an organization for a smart-farming scenario that was presented to them and that will be described in detail in chapter 6.

In particular, they were asked to first identify the core concepts of the organization such as the roles, the groups, and the tasks that the scenario suggested. Then, they were asked to give a visual representation of the core concepts and the relations among them, such as the membership of a role in a group, the dependencies among tasks, the assignation of tasks to roles, etc.

The members carried out the task individually for multiple reasons:

- they could freely express their thoughts and ideas;
- they would not be influenced by the ideas of the other members;
- their mental process could be observed and analyzed.

Indeed, one of the main goals of the focus group was to understand how people reason when they have to formalize an organization so that the Web IDE could be designed accordingly to provide a natural workflow. Therefore, the members were asked to think aloud while they were performing the task so that their mental processes could be clearly understood.

After the experiment, the results were analyzed and the main common grounds were identified.

The totality of the members of the focus group managed to identify most of the core concepts of the organization, particularly excelling in the identification of the tasks.

As far as the roles are concerned, the members were able to identify them but with different strategies. Indeed, some of them identified the roles based on the function that they perform, while others identified them based on the characteristics of the agents that are supposed to play them.

Even though the identification of the groups was not as straightforward as expected, once done the members were able to correctly identify the relations among the roles and the groups.

Finally, what the members struggled the most with was the identification of the relations among the tasks. In particular, they had difficulties expressing the latter's dependencies because they were mostly focused on a sequential execution of the tasks, which is not always the case in the smart-farming scenario they were presented.

4.2 Visual Language Design

Once the main concepts were extrapolated both from a syntactical and semantic point of view, their visual representation was analyzed, and the results

from the focus group were reviewed, the design process of the visual language started.

One of the first crucial decisions was to keep the structure and the behavior of the organization separate, as it is done in $\mathcal{M}$OISE. This choice was made because it allows the user to focus on one aspect at a time, thus making them feel not overwhelmed by the complexity of the tool. Moreover, it allows reusing some visual building blocks in both the structural and the functional specification with a different meaning, thus reducing the number of elements that the user has to learn.

Thereafter, the main components of the core concepts of the visual language were defined. As already mentioned, their design followed an iterative design process that made them more and more refined thanks to the feedback from the user tests.

4.2.1 Structure of the Organization

In this section, the visual representation of the main concepts regarding the structure of the organization are described. As already mentioned, the structure and the behavior of the organization are modeled separately using two different diagrams, therefore the following visual elements will be used in the structural specification diagram.

Roles

The roles of the organization are represented by circles, with the name of the role written above them. As can be seen in fig. 4.2, the concrete roles are represented by a solid circle, while the abstract roles are represented by an empty circle.

(a) Representation of a concrete role. (b) Representation of an abstract role.

Figure 4.2: Roles representation in the visual language.

The introduction of the abstract roles was made to allow the user to define a role that is not directly played by an agent or part of a group, but that can be extended by other roles. This is useful and powerful when the user wants to define some common properties that are shared by more roles, thus avoiding

the repetition of the same properties in the different roles. For instance, such roles could be assigned goals, links, constraints, etc. that are then inherited by the roles that extend them, in a similar way to the abstract classes inheritance in object-oriented programming. Moreover, the visual diversification between concrete and abstract roles emphasizes the difference between the two concepts, thus making the language more intuitive and suggesting the user this feature.

Groups

The groups of the organization are represented by rounded rectangles, with the name of the group written above them as shown in fig. 4.3. This container-like shape was chosen to suggest to the user that elements such as roles and other groups can be placed inside it. Therefore, the visual semantics of an element placed inside a group is that of membership, i.e. the element is a member of the group.

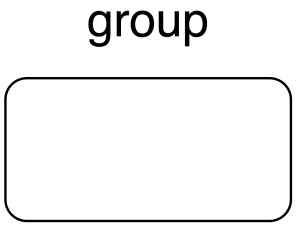


Figure 4.3: Representation of a group.

The elements that can be placed inside a group, and therefore be part of it, are the roles and other groups.

The choice to allow groups to be members of other groups was made to allow the user to define a hierarchy of groups, thus allowing them to define a more complex structure of the organization. This is particularly powerful when a group is composed of multiple instances with the same structure, as it allows the user to define the subgroup structure only once, using it as a template to check the well-formedness of the instances.

As far as the roles are concerned, only the concrete roles can be members of a group, since, as already mentioned, the abstract roles are not directly played by an agent and therefore not part of a group.

In fig. 4.4 it is possible to see an example of a simple structure that exploits group hierarchies and the role membership concept. In particular, the group `g1` is composed of the role `r1` and the group `g2`, which is composed of the role `r2`. This shows how exploiting only two visual elements, i.e. the groups and the roles, it is possible to define rather complex structures.

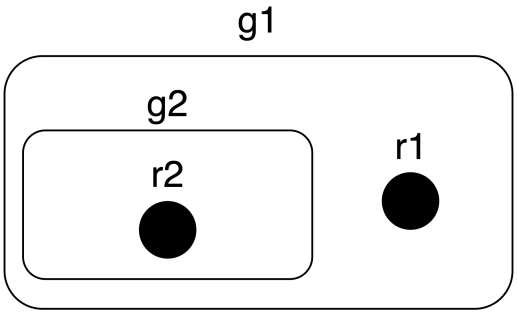


Figure 4.4: Example of a structure that exploits group hierarchies and the role membership concept.

Relations

The relations between the roles of the organization are represented by arrows, with the style of the arrow giving information about the type of relation. Only a subset of the relations defined in $\mathcal{M}$OISE is currently represented and supported by the visual language since the others are either not used in practice or there is no way to enforce them in the current version of the JaCaMo framework. Therefore, the core relations were identified and chosen, giving space to future work to extend the visual language to support the remaining relations.

(a) Extension relation among roles. (b) Compatibility relation among roles.

Figure 4.5: Visual representation of the implemented relations.

As shown in fig. 4.5, the subset of relations that are currently supported by the visual language are the extension and the compatibility relations.

The extension relation, visible in fig. 4.5a, is represented by an arrow with a dashed line, an empty harrow head, a slightly dimmed color, and the `extends` label on it. In particular, the source role, i.e. `r1`, extends the target role, i.e. `r2`, meaning that the source role is a specialization of the target role.

On the other hand, the compatibility relation, depicted in fig. 4.5b, is represented by an arrow with a solid line, a filled diamond on the target end,

a dark color, and the `compatibility` label on it. In this scenario, the source role, i.e. `r1`, is compatible with the target role, i.e. `r2`, meaning that the target role can be played by an agent that is already playing the source role.

4.2.2 Behavior of the Organization

After describing the visual components of the organization structure, the next step is to present the visual abstractions that allow the user to define the behavior of the organization. The following elements will be used in the functional specification diagram, therefore separated from the one regarding the structure.

Goals

The goals of the organization are represented by rectangles with rounded corners, with the name of the goal written inside them as shown in fig. 4.6.

Figure 4.6: Representation of a goal.

Unlike the groups, the choice to place the label inside the rectangle was made to suggest to the user that the goal can be a complete element by itself.

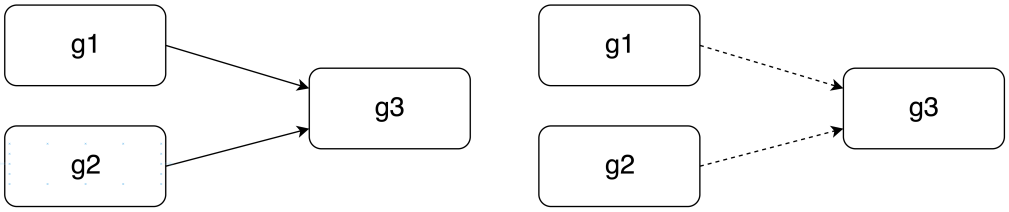
Even though according to the $\mathcal{M}$OISE model a goal can be composed of multiple subgoals, representing them inside the supergoal rectangle would make the diagram too complex and hard to read, not to talk about the scalability issues that would arise. Indeed, when defining complex scenarios, several levels of decomposition can be reached, and the multiple nesting of subgoals would make the diagram not understandable at first glance.

To address this issue, the choice was made to also represent the subgoals of a goal as top-level elements. In this way, the user can still easily identify the subgoals of a goal, while the diagram stays readable and easy to understand.

Therefore, the decision was to diverge from the $\mathcal{M}$OISE model and represent the goals not as a goal decomposition tree but as a *dependency graph*, where the nodes are the goals and the edges are the dependencies between them.

Dependencies

The dependencies between the goals of the organization are represented by arrows that connect goals, with the style of the arrow giving information about the type of dependency, as shown in fig. 4.7.



(a) Dependency between goals with `AND` semantics.

(b) Dependency between goals with `OR` semantics.

Figure 4.7: Visual representation of the dependencies among goals.

It is possible to define two types of dependencies between goals with different semantics.

The first type of dependency is the *AND* dependency, visible in fig. 4.7a, which is represented by an arrow with a solid line and a harrow head on the target end. In this scenario, the target goal `g3` depends on the source goals `g1` and `g2`. The `AND` semantics means that the target goal can be achieved only if all the source goals are achieved, with similar behavior to the plan with `parallel` operator in $\mathcal{M}$OISE.

$$g3 = g1 \wedge g2$$

On the other hand, the second type of dependency is the *OR* dependency, depicted in fig. 4.7b, which is represented by an arrow with a dashed line and a harrow head on the target end. In this scenario, the target goal `g3` depends on the source goals `g1` and `g2`. The `OR` semantics means that the target goal can be achieved if at least one of the source goals is achieved, with similar behavior to the plan with `choice` operator in $\mathcal{M}$OISE.

$$g3 = g1 \vee g2$$

Finally, the `sequence` operator is achieved through the concept of dependency itself, since dependencies have a *finish-to-start* relationship, meaning that the target goal can be achieved only after the source goals are achieved.

The dependency graph closely resembles the concept of a *Program Evaluation and Review Technique (PERT) diagram* [28], which is a statistical tool

used in project management designed to analyze and represent the tasks involved in completing a given project. However, unlike the latter which is usually made up of a single connected component, the dependency graph here described explicitly allows the user to define multiple connected components to represent groups of goals that can be achieved independently.

Goals Allocation

Although goals allocation in $\mathcal{M}$OISE is handled through norms in a separate dimension from the structural and functional one, the visual language allows the user to define the allocation of goals to roles directly in the functional specification diagram. Indeed, having a third diagram to accomplish this task would make the process of defining the organization unnecessarily complex. Therefore, being the drawbacks in terms of customization negligible, the choice was made to exploit the functional specification diagram.

This choice also abstracts from the concept of *mission* that is no longer needed to define the allocation of goals to roles. While in $\mathcal{M}$OISE goals can be grouped to form missions, which are then assigned to roles, in the visual language the goals are directly assigned to roles without the need for an intermediate step. Again, the ease of use and the simplicity of the diagram were preferred over the possibility of customization, which is acceptable in most actual use cases.

The role responsible for achieving a goal is represented by a circle with the initials of the role's name written inside it. To represent that a role is responsible for achieving a goal, its circle is placed inside the rectangle representing the goal, as shown in fig. 4.8.

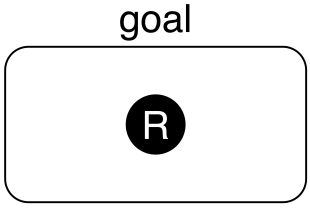


Figure 4.8: Representation of a goal allocation.

The visual representation of the rectangle corresponding to the goal slightly changes when a role is responsible for achieving it. Indeed, the goal name is moved from inside the rectangle to the top of it to make room for the roles.

There are two ways in which a goal can be assigned to a role and they come with different semantics, corresponding to the *deontic modalities* of $\mathcal{M}$OISE's normative dimension:

- **Obligation**: the goal is assigned to the role with the `obligation` modality, meaning that the agent playing that role is obliged to achieve the goal.

- **Permission**: the goal is assigned to the role with the `permission` modality, meaning that the agent playing that role is allowed to achieve the goal if it wants to.

As far as the cardinality of goal allocation is concerned, multiple scenarios are possible. A goal can be assigned to one, multiple, or no roles.

When the goal is assigned to no roles, it means that no specific agent will actively achieve it, but the goal will be considered as achieved as soon as the goals it depends on are achieved. This is particularly useful when representing dummy goals that are used to represent the completion of a task. Indeed, they improve the readability and understandability of the diagram but do not affect the actual behavior of the organization.

Moreover, the type of role that the goal is assigned to slightly changes the semantics. If the goal is assigned to a *concrete* role, then the goal is assigned to that specific role. On the other hand, if the goal is assigned to an *abstract* role, then the goal can be assigned to all the roles that are instances of, i.e. extend from, that abstract role.

4.3 Main Components and Architecture

During the design process, it was also necessary to identify all the software components that would be needed to address the requirements of the system. The choice of the components was made trying to optimize the separation of responsibilities and to minimize the coupling between the different components, thus obtaining a clean and expandable architecture.

4.3.1 Web-based IDE

This component represents the front end and it is the main interface through which the user interacts with the system.

As already mentioned, the Web-based IDE allows the users to create and edit the organization specifications through a user-friendly and intuitive graphical interface. Specifically, since the designed visual language includes two diagrams, the IDE provides the user with two different views to edit the structural and functional diagrams, respectively.

What is more, the IDE is also responsible for the translation of the organization specifications from the visual language to a format that can be

understood by the JaCaMo platform. In particular, the translation happens from and to the XML format, which is the format used to represent the organization specification in $\mathcal{M}$OISE. This allows the user to easily import and export the specifications.

Finally, the IDE provides the user with the possibility to run the organization created and enforce it on the agents currently running in the runtime environment. Therefore, the component includes an additional view that allows the user to directly interact with the agents.

4.3.2 Storage & Backend

Since the user should be able to save the organization specifications and load them later, the system needs a component that is responsible for the persistence of the data. This component should therefore provide basic CRUD functionalities for the organization specifications.

Since the data to be stored is not very complex and structured, but rather consists of strings representing the XML files, the choice was made to use a NoSQL database. In particular, the choice fell on a document-oriented database because of its simplicity and the fact that it is possible to define flexible schemas that allow for the data model to evolve as applications need change.

Together with the storage component, the system also includes a *backend*. The latter has a twofold purpose:

- It serves as a proxy between the Web-based IDE and the storage component, thus hiding the details of the storage mechanism from the IDE.
- It provides an URL to the runtime environment through which the latter can retrieve the organizations' files.

To achieve the above goals, the backend component exposes an HTTP API. This approach allows the Web-based IDE to perform operations on the stored data in a REST-like fashion and the runtime environment to retrieve organization specifications treated as resources, therefore adhering to the Web architecture.

4.3.3 Runtime Environment

This component is responsible for the execution of a MAS and therefore hosts the agents that are currently running. Thus, it will be also in charge of keeping track of the runtime information about the organizations and making the agents aware of them. In order for the organization to be enforced

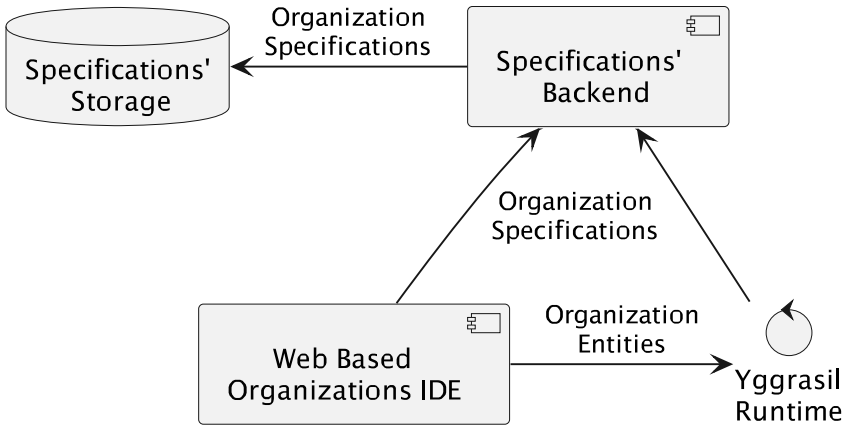


Figure 4.9: Overall architecture of the system.

on the agents, the Web-based IDE needs to communicate with the runtime environment and send the organization entities to it.

What is more, the Web-based IDE also needs to be able to retrieve information about the running agents so that users can directly assign roles to them when specifying the organization entities.

Finally, the runtime environment should communicate with the backend component in order to retrieve the organization specifications files using URLs. URLs are provided by the Web-based IDE when the latter sends an organization entity to the runtime environment.

Chapter 5

Development

In the following chapter, the development process that followed the design phase and brought to the creation of the prototype of the whole system is described.

Some technological constraints were imposed from the requirements and the design phase. In particular, the need for a lightweight Web application that didn't have to be installed on the user's machine and, from the MAS infrastructure point of view, the need to make the solution compatible with the JaCaMo platform, therefore to produce XML $\mathcal{M}$OISE specifications.

The development process was divided into three main parts. First, the Web IDE was implemented to get continuous and fast feedback about the visual language. Second, the backend and the specifications storage were developed to provide persistence to the organizations created by users. Finally, the Web IDE was integrated with the runtime environment to allow the user to access information about the running agents and deploy the organizations.

The important details and choices about the implementation of the main component of the systems are described in the following sections.

5.1 Web-based IDE

Given that the main goal of the project is to provide non-technical users with an easy and intuitive tool to create and run organizations, the Web IDE is probably the most important component of the system together with the visual language. When users are confronted with the UI of the IDE they should be able to clearly understand what each component of the interface means and how to use it without any detailed explanation.

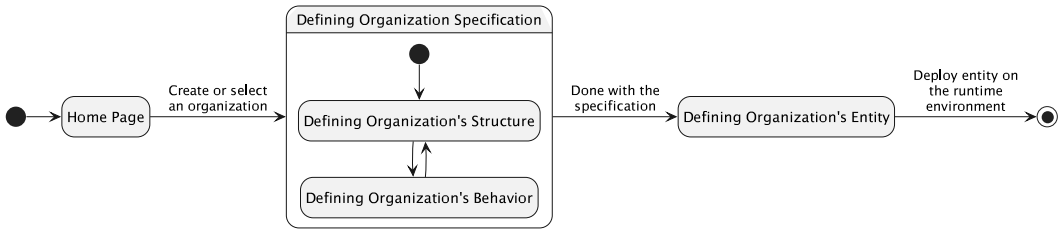


Figure 5.1: Typical workflow of a user utilizing the Web-based IDE.

5.1.1 Web User Interface

As a starting point for the design of the UI, the typical workflow of a user that wants to create an organization is considered.

As shown in fig. 5.1, the user starts by either creating a new organization or selecting an existing one.

Then, the user can define or edit the organization specification by using the designed visual language. In particular, the user specifies the structure and the expected behavior of the organization in an iterative way. Indeed, it is important to avoid sequentiality in the process as the user should feel free to easily navigate through the two separate diagrams and continuously modify them as the organization evolves.

Once the specification is ready, the user can define the organization entity by assigning the roles to the running agents. Finally, the organization can be deployed on the runtime environment and enforced on the agents.

Therefore, the final UI of the Web IDE consists of four main pages:

- **Home Page** where the user can specify the name of the organization and create it or select an existing one.
- **Structural Diagram** where the user can define the structure of the organization, thus creating a structural specification.
- **Functional Diagram** where the user can define the expected behavior of the organization that includes the functional and normative specifications.
- **Entity Definition** where the user can assign the roles to the running agents and deploy the organization.

In fig. 5.2 the UI of the Structural Diagram is shown. As can be seen, the left panel contains the available elements that can be used to create the organization structure. Indeed, the user can create new roles choosing whether they are concrete or abstract, and new groups. Once a component has been

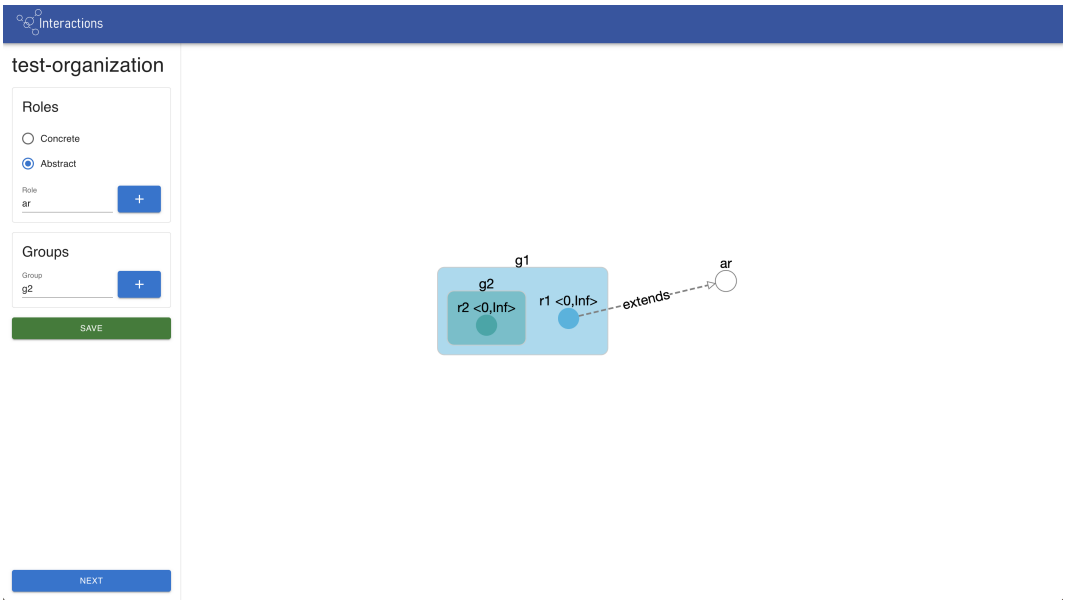


Figure 5.2: Structural Diagram of the Web IDE.

created, it can be moved around the diagram and modified by clicking on it. The click on a component opens a side menu that allows the user to edit the properties of the selected component. In particular, fig. 5.3 shows the side menu for the role `r1`. Here the user can specify which role, if any, the selected role extends from, which group it belongs to, and the cardinality of the role in the group, i.e. the minimum and maximum number of agents that can be assigned to the role.

5.1.2 Technologies

The following section describes the main technologies used to implement the Web IDE.

Since the component is based on Web technologies, the main two languages available for the implementation of the client-side logic are *JavaScript* [1] and *TypeScript* [2]. The latter was created to address problems coming with developing large-scale applications, introducing static typing and object-oriented features, thus extending the object-based features already present in JavaScript. Typescript's type-checking makes it easier to find and fix bugs, therefore, it was chosen as the language for the implementation of the Web IDE.

Moreover, a client-side framework was used to implement the Web-based

[1] `https://developer.mozilla.org/en-US/docs/Web/javascript`
[2] `https://www.typescriptlang.org/`

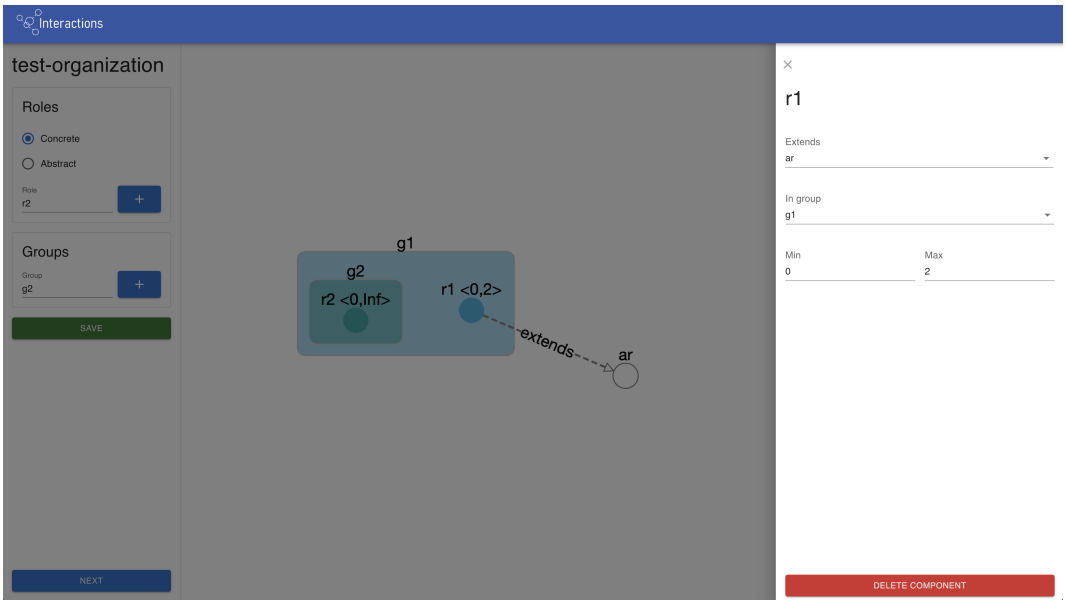


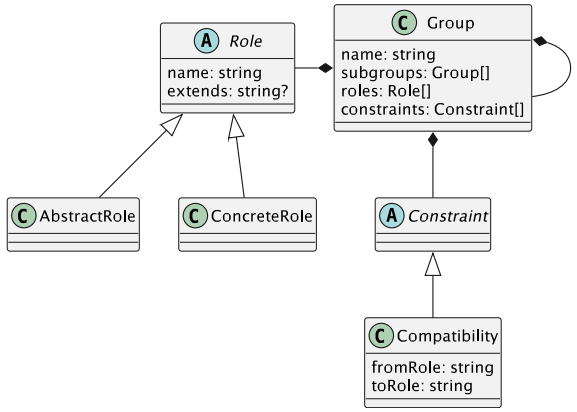
Figure 5.3: Detail showing the side menu in the Structural Diagram of the Web IDE.

IDE. The main reason for this choice is that frameworks provide a set of tools that simplify the development of interactive Web applications. They allow the developer to create reusable components that encapsulate the logic and the UI of a specific part of the application, therefore helping to divide the latter into smaller and more manageable parts and keeping the code clean and organized. The one chosen for this project is *React* [3]. Although there is no strong motivation for why it was preferred over others, React is probably the most popular, therefore, possible new contributors to the project will be more likely to be familiar with it.

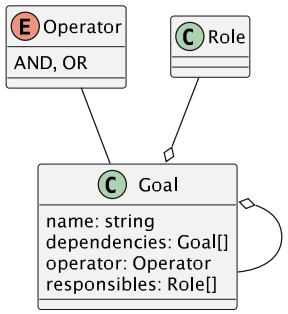
To further facilitate the development of the UI, the *Material-UI* [4] library was used. It implements Google's *Material Design* and it provides a set of tested prebuilt components that are ready to be used in the application.

Finally, to make the implementation of the diagrams easier, the *Cytoscape* [5] library was used. It is a modular graph library that provides a set of tools to create and manipulate graphs and diagrams. Using the library, together with a few extensions, made it possible to provide a simple and intuitive way to create and edit the diagrams.

[3] `https://reactjs.org/`
[4] `https://mui.com/`
[5] `https://js.cytoscape.org/`



(a) Structural Diagram Core Domain.



(b) Functional Diagram Core Domain.

Figure 5.4: UML class diagrams of the core domain.

5.1.3 Code Generation

Since the specifications created by the user with the Web IDE have to be compatible with the JaCaMo platform, and in particular, with $\mathcal{M}$OISE, the diagrams have to be translated into XML files. Equally important, actual $\mathcal{M}$OISE XML specification files have to be correctly parsed and translated into the corresponding visual representations.

Once the design phase of the different components of the visual language was in an advanced stage, the actual implementation of the code generation started.

The decision was to first implement the components that are present in both the Structural and Functional Diagrams. This allowed for having a core domain that is unlikely to change in the future and that could be used as a starting point for both the translation into the visual components and XML tags. In particular, the core domain consists of the components shown in fig. 5.4.

Specifically, the system keeps track of the components of the diagrams and the relations between them in a list. The latter is then taken as input by a `serialize` function that transforms the list into an XML string. On the other hand, an analogous process is performed when loading an XML specification. The `deserialize` function takes the XML string as input and transforms it into a list of components. In listing 5.1 and listing 5.2 are shown the functions for the code generation and parsing of the roles inside a group, respectively.

```
export const roles: (roles: Set<ConcreteRole>) => string = roles =>
    "<roles>\n" +
        Array.from(roles).map(r =>
            `<role id="role_${r.name}" ${r.min === 0 ? "" :
                `min="${r.min}"`} ${
                r.max === Number.MAX_VALUE ? "" : `max="${r.max}"`
            }/>`).join("\n") +
    "</roles>"
```

Listing 5.1: Code generation for the roles inside a group.

```
const role: (element: XMLElement) => Role = element =>
    new ConcreteRole(
        element.attributes["id"],
        roleTopography
            .find(rt => rt.name === element.attributes["id"])
            .map((rt: Role) => rt.extends).getOrElse(undefined),
        option(parseInt(element.attributes["min"])).getOrElse(0),
        option(parseInt(element.attributes["max"]))
            .getOrElse(Number.MAX_VALUE)
    )
```

Listing 5.2: Parsing of the XML specification of a role inside a group

As far as the goal dependencies are concerned, a one-to-one mapping was not possible since the visual language's semantics differs from the one used by MOISE. Specifically, the visual language does not allow for the creation of plans and subgoals that are not associated with a specific goal with a plan operator. In order to overcome this incompatibility, the *depends-on* concept of MOISE was used. Although it is usually not directly used in the XML specifications, *depends-on* is a fundamental concept of MOISE which plans and subgoals are built upon.

Therefore, this concept is used as a mapping between the visual language and the XML syntax. In particular, when expressing the *and* dependency, a simple *depends-on* tag is created between the two goals. On the other hand, when expressing the *or* dependency, an intermediate goal is created with a *choice* plan.

For instance, in listing 5.3 is shown the XML translation of an *and* dependency between the goals *g1*, *g2* and *g3* of the type:

$$g3 = g1 \wedge g2$$

```
<goal id="g1" />
<goal id="g2" />
<goal id="g3">
    <depends-on goal="g1" />
    <depends-on goal="g2" />
</goal>
```

Listing 5.3: XML translation of an *and* dependency.

```
<goal id="g1" />
<goal id="g2" />
<goal id="or_g3">
    <plan operator="choice">
        <goal id="g1" />
        <goal id="g2" />
    </plan>
</goal>
<goal id="g3">
    <depends-on goal="or_g3" />
</goal>
```

Listing 5.4: XML translation of an *or* dependency.

whereas in listing 5.4 is shown the XML translation of an *or* dependency between the goals *g1*, *g2* and *g3* of the type:

$$g3 = g1 \vee g2$$

5.2 Storage & Backend

Organizations can be saved persistently to be later recovered and edited. This feature is fundamental for the Web IDE since writing a specification is commonly an iterative process that may require the domain expert to edit some parts of an organization either for bug-fixing or to add new features.

5.2.1 Storage

Since the visual programming environment is based on Web technologies, centralized server-side storage was the most natural choice. Looking at the

system architecture in fig. 4.9, the **Specifications' Storage** component is responsible for this functionality.

The storage uses a *MongoDB* [6] database to store the generated XML specifications. For the time being, the database does not store the information about the diagrams, such as the position of the components that would make it easier to load the organization in the same state as it was saved.

Moreover, there is no separation between the specifications created by different users, therefore, in a production environment some form of authentication would be necessary.

Finally, a further improvement of the storage component would be to also allow the user to save organization entities, keeping track of which specific configurations of the organization in different scenarios.

5.2.2 Backend

Since, as already mentioned, the Web IDE is based on Web technologies, a backend seemed the most natural choice to make the frontend and the storage communicate.

The backend was implemented using the *Kotlin* [7] programming language and the *Vert.x* [8] framework. The latter is a reactive library that allows the developer to write asynchronous code and build Web servers quickly. It was chosen because the research group in St. Gallen already had some experience with it, therefore, it would be easier to maintain for other developers. In particular, the projects built by the team use *Java* [9] as the programming language, therefore, the choice of *Kotlin* was a fair tradeoff that allowed using a language that is similar to *Java* but with a more modern and concise syntax.

The backend exposes an HTTP API that allows the frontend to communicate with the storage. The API that is currently implemented is here described:

- `GET /organizations/:name`: returns the XML specification of the organization with the given name.
- `POST /organizations/:name`: saves the XML specification present in the body to the storage.
- `PUT /organizations/:name`: updates the organization with the given name, substituting it with the XML specification present in the body.

[6] `https://mongodb.com/`
[7] `https://kotlinlang.org/`
[8] `https://vertx.io/`
[9] `https://java.com/`

Since the `GET` route returns the XML specification, the runtime environment can also use it as a way to retrieve the specification in a resource-oriented fashion.

5.3 Running Organization Entities

In order for users to be able to run the organizations they create, the Web IDE has to be able to communicate with the runtime environment. Specifically, it needs information about the running agents to assign them to the correct roles and, subsequently, it needs to enforce the organization on said agents.

5.3.1 Runtime Environment

As already mentioned, the runtime environment, which is based on the JaCaMo platform, contains all the running agents and the deployed artifacts.

This component is currently being developed by the research group in St. Gallen. The name of the platform is *Yggdrasil* [10], which comes from the world tree in the Norse mythology, and it aims at providing uniform interaction among heterogeneous agents thanks to Hypermedia MAS. The idea is to model an environment based on the Agents & Artifacts metamodel through hypermedia and Web technologies, achieving scalability and uniform access to resources.

Yggdrasil exposes a REST-like API that allows users and agents to navigate the environment and the existing workspaces where artifacts and their operations are described with the Thing Description standard since the model fits the Agents & Artifacts metamodel. Moreover, the platform's API also allows the user to run agents and create and deploy artifacts.

The format used by Yggdrasil to describe the resources in the environment is the *Turtle* [11] format. The latter is a textual syntax for RDF that allows a knowledge graph to be represented in a compact and natural text form.

5.3.2 Running Agents

Agents in Yggdrasil, just like artifacts, are contained in workspaces. Therefore, to get the list of agents that are currently running, the Web IDE has to operate on a workspace and retrieve the list of resources that are contained in it.

[10] `https://github.com/Interactions-HSG/yggdrasil`
[11] `https://w3.org/TR/turtle/`

```
<workspaces/102> rdf:type eve:WorkspaceArtifact ,
                          <Workspace> ,
                          td:Thing ;
                 <directlyContains> <hypermedia_body_1> ,
                                    <hypermedia_body_2> ;
                 td:hasActionAffordance [
                                        ...
                 ];
                 ...
```

Listing 5.5: Detail of the Turtle representation of a workspace.

By performing a `GET` request to the `/workspaces` endpoint, specifying the workspace name in the URL, the Web IDE can retrieve a Turtle representation of the workspace; a short example of the response is shown in listing 5.5. As can be seen, the workspace, which is the subject of all the triples, *directlyContains* `hypermedia_body_1` and `hypermedia_body_2` which represent two agents currently running.

Therefore, the Web IDE can parse the Turtle representation and extract the list of agents that are currently running, filtering them from all the other resources that are contained in the workspace. Since this list is composed of URIs, `GET` requests to them can be performed to retrieve the Turtle representation of the agents. The response of the `GET` request to the agents provide useful information for the user such as the name of the agents that can be used to recognize them and assign them to the correct roles.

5.3.3 Artifacts Creation

Artifacts in Yggdrasil can be created by performing a `POST` request to the `workspace/:workspaceName/artifacts/` endpoint, specifying all the necessary information in the body of the request. For instance, the body should contain the type of artifact that is being created, the name of the artifact, and possible initial parameters that the artifact may need.

As far as the organizational artifacts are concerned, the types of artifacts available are the same as the ones described in ORA4MAS, therefore `OrgBoard`, `GroupBoard`, `SchemeBoard`, and `NormativeBoard`. Once the `OrgBoard` artifact is created, the most correct way to generate the other artifacts is through an action exposed by it. In particular, the `OrgBoard` artifact exposes the `createGroup` and `createScheme` actions that can be used to create the other artifacts. A `NormativeBoard`, on the other hand, is automatically generated when a `SchemeBoard` is linked to a `GroupBoard`.

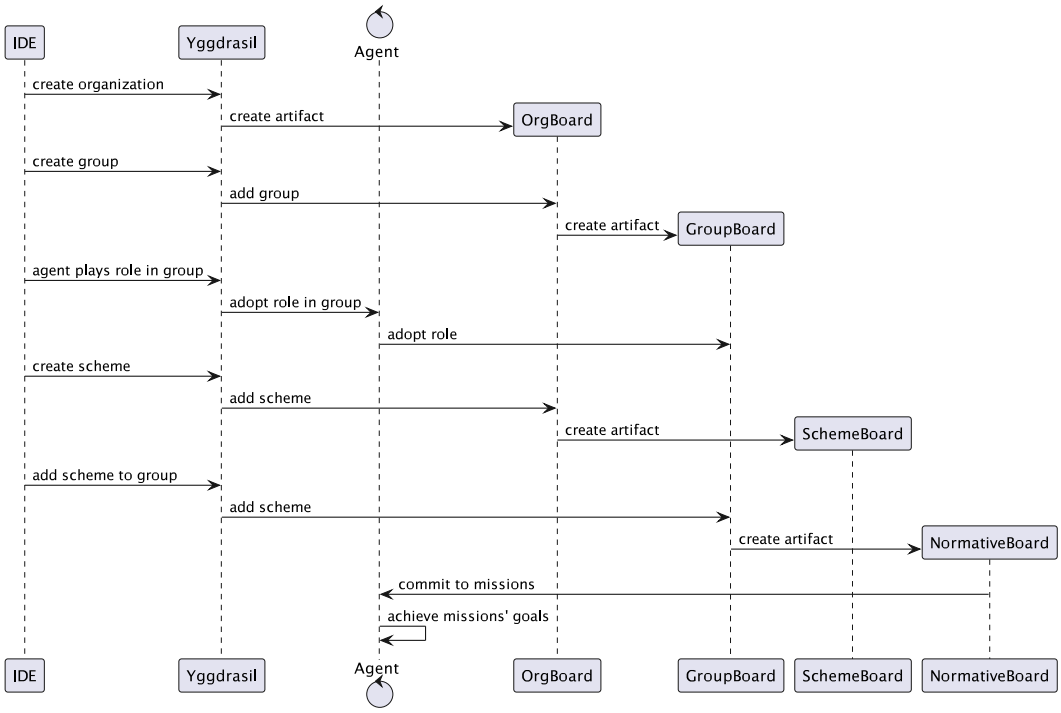


Figure 5.5: Sequence UML diagram of the deployment of an organization. The responses are omitted to improve the readability.

Therefore, to correctly instantiate the organizational artifacts and have them refer to the same organization, the Web IDE has to perform a `POST` request to the `workspace/:workspaceName/artifacts/` endpoint to create the `OrgBoard` artifact, and then, once the latter is created, perform a `POST` request to the `workspace/:workspaceName/artifacts/:orgName/create*` endpoints to create the other artifacts.

5.3.4 Organizations' Deployment

Deploying an organization is a process that includes the creation of the correct organizational artifacts and the adhesion of the agents to the organization, i.e. playing the established roles in the prescribed groups.

In fig. 5.5 the typical deployment process of a simple organization is shown.

1. The Web IDE first sends a request to Yggdrasil to create the `OrgBoard` artifact, which is the artifact that represents the organization.

2. Once the artifact is ready, the Web IDE creates the groups specified in the organization entity. For each group, a `createGroup` request is sent

to the `OrgBoard` artifact, which creates the `GroupBoard` artifact that represents the group.

3. The next step involves telling the agents to play the roles specified in the organization entity inside the groups. To do so, the Web IDE sends a message to the agents, which know how to send a request to the `GroupBoard` artifact to play the role.

4. Subsequently, the Web IDE sends a request to create the scheme of the organization. This request is sent to the `OrgBoard` artifact, which creates the `SchemeBoard` artifact that represents the scheme.

5. Finally, the Web IDE sends a request to add the scheme to the groups. This request creates a link between the `SchemeBoard` artifact and the `GroupBoard` artifacts and a `NormativeBoard` is automatically generated.

6. Since the agents are listening to events generated by the `GroupBoard` artifacts, they will receive a notification telling them to achieve the goals specified in the scheme.

7. If the agents are obliged to achieve the goals, they will proceed to do so. On the other hand, if the agents are permitted to achieve the goals they will do so only if they have an interest in it.

Chapter 6

Evaluation

After developing a working prototype of the whole system, the next step was to prove that the solution satisfies the requirements.

This chapter describes the evaluation of the developed system. In particular, the evaluation is divided into two parts:

1. the first part involves the use of the developed system to solve a real-world problem;

2. the second part is a qualitative evaluation of the developed visual language and development environment by users;

As already mentioned, this twofold evaluation approach allows for assessing the user-friendliness of the development environment and the effectiveness of the visual language on one hand, and the expressivity and correctness of the latter on the other.

6.1 Case Study

In order for users to have a real-world problem to solve, both during the design phase with the focus group and during the evaluation phase with the end users, a use-case scenario was developed.

6.1.1 Smart-Farming Scenario

Since the *IntellIoT* project already defines three sectors of application, namely *Agriculture*, *Healthcare* and *Manufacturing*, the most natural choice was to use one of them to frame the case study. The *Agriculture* sector was chosen because it involves concepts that are easy to understand even for non-experts and it is possibly the most diverse.

Therefore, the case study was developed around the *Smart Farming* scenario where the aim is to define an organization for a typical farm that adopts IoT technologies. Here is reported the text of the scenario:

> Thanks to his investments, the farmer obtained the most cutting-edge technology machines to make his life easier. After obtaining this technology, the farmer wants to organize the management of the daily chores of his "smart farm" by exploiting the machinery he possesses, which includes a self-driving tractor, multiple drones, an automatic irrigation system, and devices to take care of the animals.
>
> With the acquired machinery, the farmer must irrigate the fields, which is highly demanding. Therefore, the farmer purchased drones to improve the process of checking the soil and gathering information such as temperature and humidity, which the irrigator can use to calculate the amount of water needed. The drones not employed for the former tasks will eliminate the moths and bugs that haunt the cultivation.
>
> In addition, the farm always has a field that is currently not used to grow any crop. However, the tractor must still plow the soil in that field. To perform this function, it will need a set of waypoints. The tractor can either have them computed by a drone flying over the field or as direct input from the farmer.
>
> Finally, the farmer wants to harvest mature fruit and vegetables (we can assume that the tractor knows how to harvest). After the harvest, the tractor can spray the field with pesticides to protect the crop.
>
> As far as the animals are concerned, the farmer wants to feed them and have a daily health check-up for every animal. Moreover, he wants to collect the eggs from the hens and milk the cows and the goats. It is worth mentioning that, during years of experience, the farmer noticed that feeding the animals before their health check-up makes them quieter, and the cows and the goats calmer when they get milked.

6.1.2 Use-Case Analysis

Although not only one solution is possible for this example, here a reference one is described. The approach to the problem, analogous to the one explicitly suggested to the focus group and to end users, consists of the following steps:

- identify the roles of the organization;
- identify the groups of the organization;
- identify the tasks to be performed and possible relations between them;
- identify which roles are responsible for the tasks.

In particular, the approach used to solve the case study follows some simple guidelines. As far as task decomposition is concerned, there could be different reasons why a task may be split into subtasks. For instance, a task may be too complex to be performed by a single agent since it needs the capabilities of multiple agents. Another reason may be that a task takes too much time to be performed atomically and it is better to split it into smaller tasks. Moreover, a task may be decomposed into subtasks to use the achievement of a subtask as a precondition for the achievement of another task. Finally, a task may be split into subtasks to allow for parallel execution of the subtasks by multiple agents.

On the other hand, when it comes to the definition of the roles, the approach mainly relies on the identification of a set of capabilities that the agent playing that role should provide in order to perform the tasks assigned to it. For instance, referring to the "write paper" example in fig. 4.1, the role of the `writer` is defined by the capability of writing the sections of a paper, therefore agents playing that role can successfully perform the tasks regarding the writing of the section of the paper.

Roles

The approach to the problem starts by identifying the roles of the organization. The ones defined for the case study are:

- **Tractor Pilot** which will be played by agents that can drive a tractor. This role is abstract and can be specialized in:
  - **Soil Plower**: played by an agent capable of plowing the soil;
  - **Harvester**: played by an agent that can harvest the crops;
- **Drone Pilot** which, in an analogous way to the tractor pilot, will be played by agents that can control a drone. This role is abstract and can be specialized in:
  - **Temperature Checker**: played by an agent that can measure the temperature;

  - **Humidity Checker**: played by an agent that can measure the humidity;
  - **Bugs Eliminator**: played by an agent that can kill bugs;
- **Irrigation System**: played by an agent that can irrigate the fields;
- **Animal Feeder**: played by an agent that can feed the animals;
- **Vet**: played by an agent that can perform a health check-up on the animals;
- **Product Collector**: played by agents that can collect the products from the animals. It is an abstract role that can be specialized in:
  - **Egg Collector**: played by an agent that can collect the eggs from the hens;
  - **Milk Collector**: played by an agent that can collect the milk from the cows and the goats;

Groups

Once the roles are identified, the next step is to identify the groups of the organization. In particular, the groups are defined as follows:

- **Farm Group**: it is a group of agents that can perform the tasks related to the farm. In this scenario, there are no roles that directly belong to this group. However, it contains the following subgroups:
  - **Field Group**: a group of agents that can perform the tasks related to the fields. The roles that belong to this group are: *Soil Plower*, *Harvester*, *Temperature Checker*, *Humidity Checker*, *Bugs Eliminator*, and *Irrigation System*;
  - **Animal Group**: a group of agents that can perform the tasks related to the animals. The roles that belong to this group are *Animal Feeder*, *Vet*, *Egg Collector*, and *Milk Collector*.

Tasks/Goals

The next step involves identifying the tasks, or goals, to be performed and the possible relations between them.

The first goal encountered is *Irrigate Field*. However, the latter needs some intermediate steps to be achieved and, therefore, it has dependencies on other goals. In particular, the *Calculate Water* is needed so that the irrigator knows

how much water to use. In turn, the *Calculate Water* goal needs the *Measure Temperature* **and** *Measure Humidity* goals to be achieved.

Proceeding with the scenario, the next goal is *Eliminate Bugs*. The *Eliminate Moths* goal could be also identified, but, for simplicity, it is not considered.

The next goal is *Plough Field* that needs either the *Compute Waypoints* or the *Input Waypoints* goals to be achieved, therefore it depends on them with an **or** relation.

Next, the *Harvest* goal is identified, which also enables the *Spray Pesticides* goal.

Finally, the goals concerning the animals are defined. In particular, the *Feed Animals*, *Health Check-Up*, and *Collect Products* goals are identified. The latter can be further specialized in *Collect Eggs* and *Collect Milk*. In turn, *Collect Milk* needs the *Milk Cows* and *Milk Goats* goals to be achieved. Moreover, the *Feed Animals* goal enables all the other goals related to the animals.

As far as the assignation of the goals to the roles is concerned, table 6.1 shows the result of the analysis:

Goal	**Responsible Role**
Irrigate Field	Irrigation System
Calculate Water	Irrigation System
Measure Temperature	Temperature Checker
Measure Humidity	Humidity Checker
Eliminate Bugs	Bugs Eliminator
Plough Field	Soil Plower
Compute Waypoints	Drone Pilot
Harvest	Harvester
Spray Pesticides	Tractor Pilot
Feed Animals	Animal Feeder
Health Check-Up	Vet
Collect Products	Product Collector
Collect Eggs	Egg Collector
Collect Milk	Milk Collector
Milk Cows	Milk Collector
Milk Goats	Milk Collector

Table 6.1: Assignation of the goals to the responsible roles.

6.2 Solution with the Visual Language

In this section, a solution for the above scenario using the developed visual language is presented.

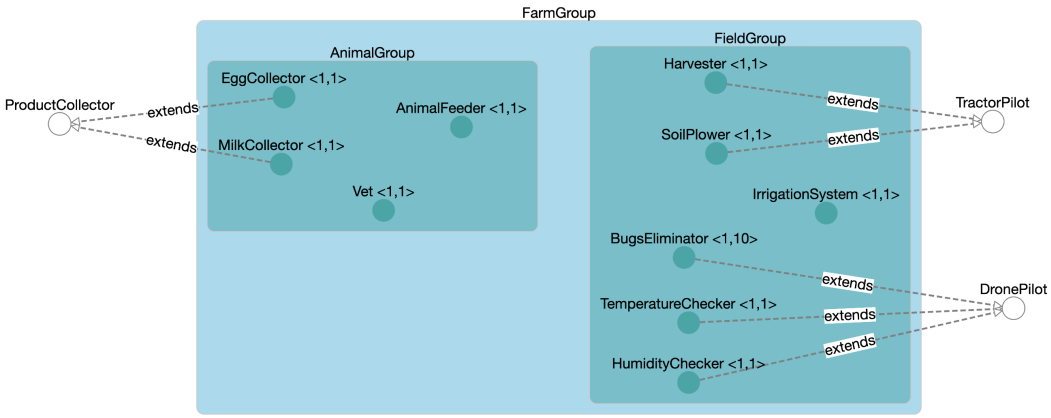


Figure 6.1: Solution for the structure of the organization.

In fig. 6.1 the solution regarding the structure of the organization is presented. The outermost rectangle represents the `Farm Group` that contains two subgroups, `Animal Group` and `Field Group`, depicted by the two inner rectangles. The three circles that are not placed inside any group represent the abstract roles `Product Collector`, `Tractor Pilot`, and `Drone Pilot`. As for the other roles, they are concrete and therefore placed inside their corresponding group. Some of them extend the abstract roles, as is the case of `Egg Collector` and `Milk Collector` that extend `Product Collector`. Finally, the cardinality of the roles inside the groups is represented in the form <`min,max`>. For instance, the `Bugs Eliminator` role has to be played by at least 1 agent and at most 10 agents while all the other roles have to be played by exactly 1 agent.

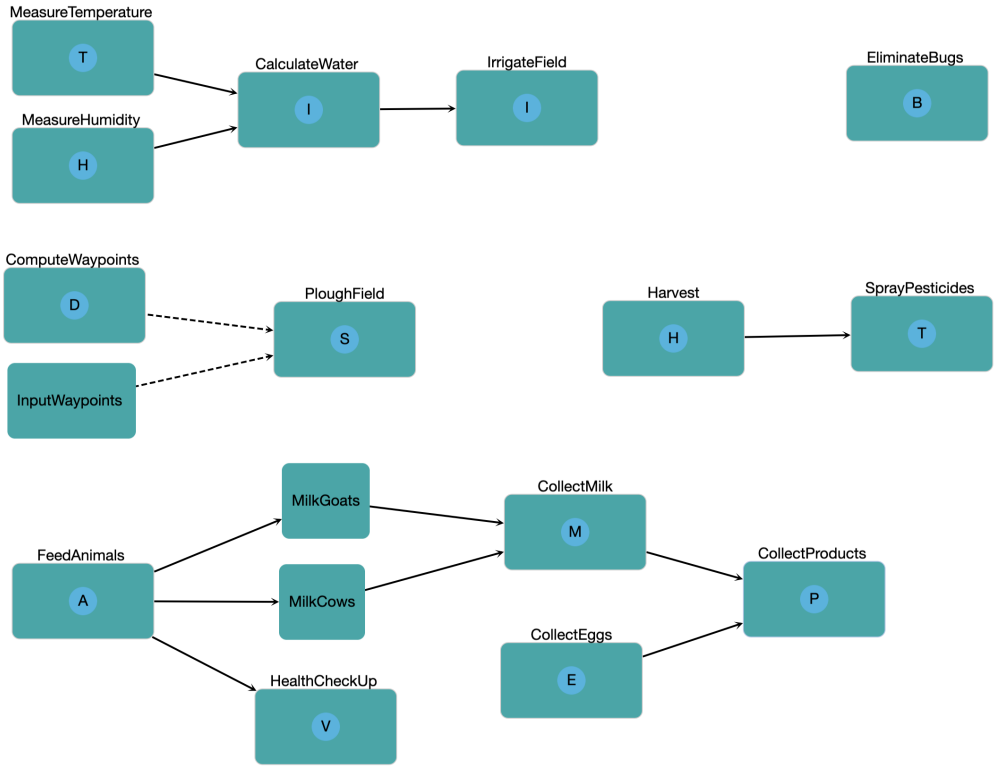


Figure 6.2: Solution for the behavior of the organization.

As far as the behavior of the organization is concerned, fig. 6.2 shows its solution with the visual language. For instance, the `Collect Products` goal requires the `Collect Milk` and `Collect Eggs` goals to be achieved, therefore, it depends on them with an **and** relation. The latter is represented by the solid arrows that connect the enabler goals to the enabled goal. Indeed, the use of dashed arrows would indicate an **or** relation, like the one between the `Compute Waypoints` and `Input Waypoints` goals and the `Plough Field` goal. In turn, the `Collect Milk` goal requires the `Milk Cows` and `Milk Goats` goals to be achieved. Again, the former depends on the latter goals with an **and** relation. Moreover, the `Milk Cows` and `Milk Goats` goals need the `Feed Animals` goals to be achieved in order for them to be pursued. Finally, the latter goal also enables the `Health Check-Up` goal.

To conclude, the assignation of the goals to the roles is represented by the presence of the circle corresponding to the role inside the rectangle representing the goal. For instance, the `Vet` is responsible for the `Health Check-Up` goal.

6.3 Users' Test

Due to the shortage of time, it was nearly impossible to gather a large number of users to test the developed visual language. Moreover, even though the target users for the system are domain experts, finding them and having them test the system would not have been feasible.

Thus, the users' test was performed on a small number of people from the research group. Since the users have a background in computer science and some of them are familiar with the domain of the scenario, the result was expected to be slightly optimistic but still useful to evaluate the usability of the system.

6.3.1 Test Description

The test consisted in asking the users to provide a solution for the smart-farming scenario using the Web IDE and therefore exploiting the designed visual language. The vast majority of the users were already familiar with the scenario since they participated in the focus group sessions that helped the design of the visual language itself. They were also asked to think aloud while they were building the solution, so that possible doubts could be clarified, mainly about the meaning of the concepts of the organization model such as roles, groups, and goals.

After the users had provided their solutions, they were asked to provide feedback through a short questionnaire. The latter was composed of a few questions about every step of the process, from the identification of the core concepts of the organizations to the use of the Web IDE to build the solution.

The questions were designed to evaluate the usability of the system and the visual language on one hand, and the effectiveness of the organization model on the other, that is the ability of the model to represent the organization in a way that is natural and understandable for the users. Specifically, the questions were modeled as statements such as *It was easy for me to identify what roles are needed in the organization* and *It was easy for me to create the roles using the IDE.*

The participants were asked to rate the statements on a scale from 1 to 5, where 1 means that they strongly disagreed with the statement and 5 means that they strongly agreed with the statement.

6.3.2 Results

Even though some of the users needed some hints to get started, all of them were able to provide a solution for the scenario in an acceptable amount

of time, with the average time being around 25 minutes.

All the solutions provided by the users produce a syntactically correct organization specification, which means that no errors will be encountered when the system tries to deploy the organization. However, the solutions provided by the users are not all necessarily semantically correct, that is, they might not represent the organization in a way that completely matches and satisfies the requirements of the scenario.

What the users most struggled with was the identification of the roles and the groups. Specifically, they had difficulties in distinguishing the concepts of agents and the roles played by them. Moreover, some of them had difficulty understanding that roles have to be inserted in groups. On the other hand, the users easily understood the concept of goals and were able to identify them in the scenario together with the relations between them.

However, once the users were explained the meaning of the concepts and some fundamentals of the organization model, they were able to provide a solution for the scenario in a short amount of time.

As far as the use of the Web IDE is concerned, all the users had no issues in creating the visual components and interacting with them. What is more, the navigation through the different pages of the IDE was straightforward for all the users.

Therefore, these results show that the visual language is intuitive and that the Web IDE is easy to use. As far as the organization model is concerned, it appears to be effective in representing the organization powerfully and flexibly, even though some of the users do not inherently reason about organizations in terms of the model's concepts. Nevertheless, it turned out to be simple enough to be easily understood by the users when they are briefly introduced to the main concepts.

In fig. 6.3 and fig. 6.4 the ratings of the statements are shown. The first figure shows the results for the statements related to the organization model, that is the identification of the core concepts of the organization. On the other hand, the second figure shows the results for the statements related to the use of the Web IDE. Each color represents a step of the process of the test, that is, the identification and creation of the roles, groups, goals, dependencies among goals, and the assignation of goals to roles, respectively.

According to the feedback provided by the users, the result of the test is extremely positive. As already stated, the results were expected to be slightly optimistic since the users are familiar with the domain and the organization model. However, the results show that the organization model is effective in representing the organization and that the visual language is intuitive and easy to use.

Finally, it is worth noting that the user that provided the most complete

solution for the scenario was a computer science student who had never heard of MAS organizations before, proving that there is no need for a specific background and previous experience to define one.

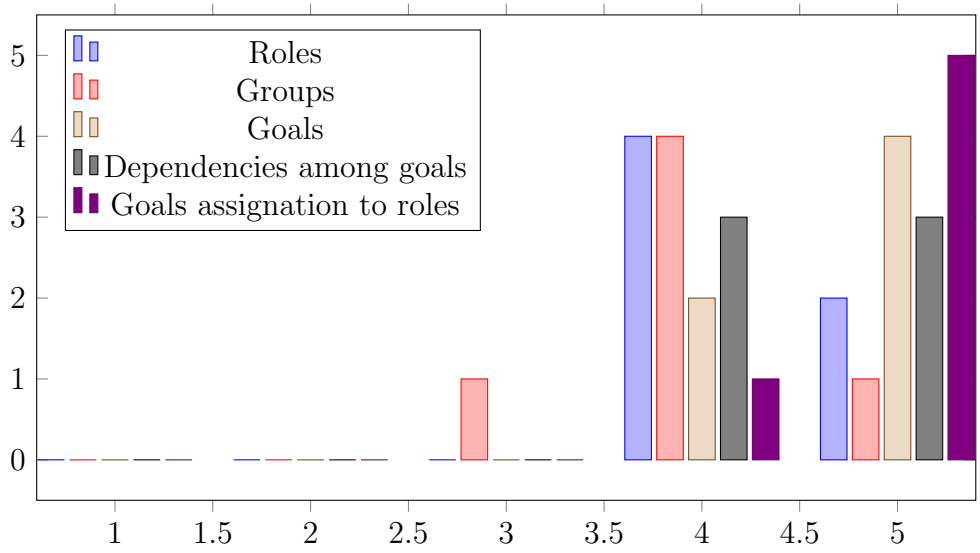


Figure 6.3: Average rating of the statements concerning the identification of the main concepts of the organization model.

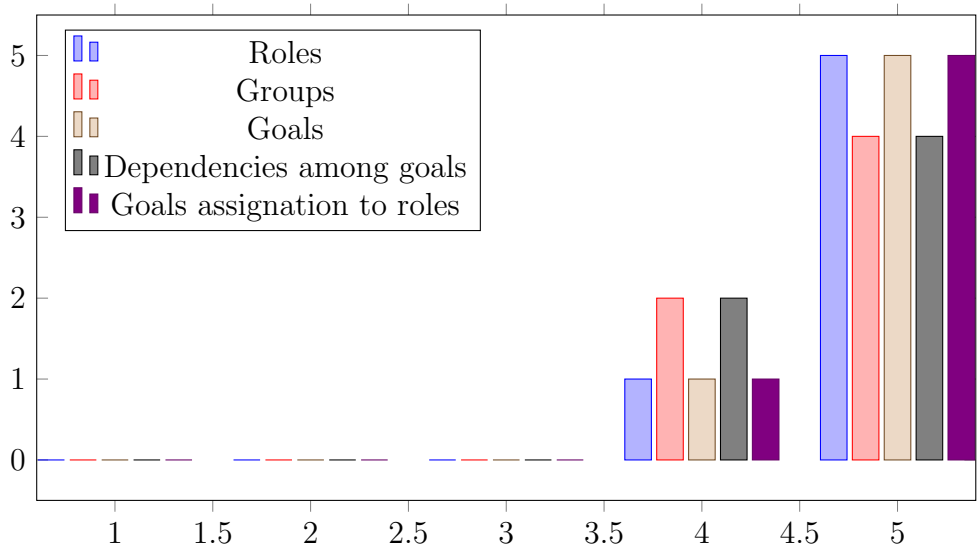


Figure 6.4: Average rating of the statements concerning the ease of use of the Web IDE when defining the concepts.

Conclusions

Main Contribution

As stated multiple times in this document, the main contribution of the thesis work is the application of visual programming techniques to the Multi-Agent Oriented Programming (MAOP) paradigm aiming at overcoming the challenges that come from the coordination of single entities in a way that is accessible by non-technical domain experts.

This project represents the natural evolution of the previous work from [8] that was also carried out at the University of St. Gallen and that was focused on the development of a visual programming paradigm for software agents. Indeed, it moves a step forward toward the vision for an accessible Integrated Development Environment (IDE) that mixes Multi-Agent Oriented Programming and Hypermedia in a seamless interface for both humans and software agents.

The work culminated in the development of a prototype of the system that was tested and evaluated on a sample of users and gave promising results in terms of usability, user-friendliness, and correctness. However, a long path was taken to reach this point.

The development steps were first focused on the analysis of the requirements coming from the *IntellIoT* project, the research interest of the hosting group, and the already implemented agent's IDE. Then the analysis of the tools and technologies used by the research team brought to the identification of $\mathcal{M}$OISE and the JaCaMo meta-model as a starting point to implement the solution.

The next step involved the analysis of the $\mathcal{M}$OISE specifications, its core concepts, and the gathering of a focus group to identify the building blocks of the visual programming language in a way that would be as understandable as possible for the target users.

The implementation of an interface supporting the usage of the visual language followed, alongside the development of a storage and backend to provide the persistence of the organization specifications created.

Finally, the Web IDE was integrated with the existing runtime environment to allow the execution of the organizations created by the users, therefore allowing the full cycle of development and deployment of the organizations.

The prototype subsequently underwent an evaluation study, first trying to address a rather complex real-world scenario, and then testing the usability of the system with a small sample of users in contact with the tool for the first time and with little to no explanation of its functionalities.

Overall, this thesis brought to the realization of a usable tool that can be used by non-technical users to create and deploy organizations in a way that is accessible and understandable for them. Most important, this opened a lot of potential directions for future work, as the system is still in its early stages on one hand, and it enables new research threads on the other.

Open Challenges and Future Work

Working on a project that touches multiple and diverse research fields, that relies on currently developing technologies, and in an environment that encourages the exploration of new ideas and the interaction with colleagues from different backgrounds and working on different research topics, can only result in a continuous flow of new questions and new directions to explore. Of course, the thesis work cannot follow all the possible paths and therefore a lot can be done to improve the existing solution and expand it.

First of all, the development of the visual language is an ongoing process that requires more iteration than the ones it was possible to carry out to be able to refine the visual building blocks and make them more intuitive and accessible for the users. Indeed, a lot of work is still needed to find the right visual abstractions to represent the core concepts that might require the exploration of different visual paradigms and the gathering of more feedback from the users.

Moreover, some concepts of the $\mathcal{M}$OISE specifications were purposefully left out of the visual language, both for time constraints and for the need to keep the language as simple as possible. However, more advanced users might find it useful to have access to these concepts. Therefore, a way to extend the visual language to include them should be explored.

As far as the Web IDE is concerned, the current implementation only supports the creation of organizations and their deployment on the runtime environment. However, it would be extremely interesting to add the possibility to monitor the state and development of the organizations and to provide a way to interact with them. This way, the user could be involved not only at design time but also at runtime, promoting the human-in-the-loop approach

the *IntellIoT* project is based on.

Regarding the runtime environment, for the time being, the user has to manually choose the agents that will participate in the organization and assign them the roles they should play. However, it would be interesting to explore the possibility of automatically assigning roles to the agents based on their capabilities. Indeed, some members of the research group are currently working on the development of a framework based on costs and rewards that are assigned to the agents when they adopt a certain role and achieve the goals the role is responsible for. This is a promising direction toward self-organization and re-organization in Multi-Agent Systems that will be for sure explored in the future.

As far as the technologies used are concerned, lately, a lighter version of $\mathcal{M}$OISE has been developed, called $\mathcal{M}$OISE *simple* [1], that, as the name suggests, should be easier to use for domain experts. Therefore, it could be fruitful to compare the two approaches and analyze the tradeoffs in terms of user-friendliness and expressiveness.

Hopefully, given the interest in the project from both the research group in St. Gallen and the one in Cesena, some of these directions can be explored in future research projects.

[1] `https://github.com/moise-lang/moise/tree/master/src/main/java/ora4mas/simple`

Acknowledgments

This thesis and my academic years would not have been the same without the many inspiring people that I have met along the way. Therefore, I would like to express my gratitude to my mentors, colleagues, and friends that have supported me during this journey.

Firstly, I would like to thank my supervisor, Prof. Alessandro Ricci for transmitting to me his passion for his research topics through his enthusiasm when teaching, and for giving me the opportunity to live such a unique experience by connecting me with some of the brightest people in the field of Multi-Agent Systems.

Then I would like to thank Samuele Burattini for always being there during the development of this thesis, for your reassurance and support, and for your precious advice. I wish you all the best for a successful and rewarding academic career.

I would also like to thank everyone I had the pleasure to work with in St. Gallen while working on this thesis, especially Prof. Simon Mayer, Prof. Andrei Ciortea, and Jérémy Lemée for their continuous support and for introducing and guiding me into the world of research. From the first day I arrived in St. Gallen, you made me feel at home and part of your group, and I am truly grateful for that. Thank you for all the interesting discussions, the lovely lunch chats, and for sharing your knowledge and experience with me.

Speaking of St. Gallen, I would also like to thank my flatmate Sebastian and my friends Lukas and Jennifer for making my stay there so enjoyable, letting me have some fun after long days of work, and supporting me when work was overwhelming and through sad and difficult times. In such a short time, we have been able to share so much and you have become such an important part of my life.

I want to thank my family for their unconditional everyday support and their help. Thank you for allowing me to keep studying and following my passions. I know that sometimes I am not the most open and outgoing person, but I truly appreciate all the love and care you give me.

Even though probably words are not enough to express my gratitude, I still want to try and thank the members of what I call my second family. First of

all, Simone whom I met during my first year of university and who had the burden to put up with me since. Hopefully, we can yell at each other some more when working together on other projects in the future. Equally, I would like to thank Marta and Martina. We have shared quite possibly all the joy and sorrow of our master's degree. You have always made me feel understood and appreciated, and I am infinitely grateful for that.

Last but not least, I would like to thank all the individuals whom I have met during this journey and who contributed to making my university experience valuable and unique.

Bibliography

[1] Hosny Ahmed Abbas. Organization of multi-agent systems: An overview. *International Journal of Intelligent Information Systems*, 4(3):46, 2015.

[2] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *26th USENIX security symposium (USENIX Security 17)*, pages 1093–1110, 2017.

[3] Marie Baezner and Patrice Robin. Stuxnet. Technical report, ETH Zurich, 2017.

[4] Olivier Boissier, Rafael H Bordini, Jomi Hubner, and Alessandro Ricci. *Multi-agent oriented programming: programming multi-agent systems using JaCaMo*. MIT Press, 2020.

[5] Olivier Boissier, Jomi F. Hübner, and Alessandro Ricci. *The JaCaMo Framework*, pages 125–151. Springer International Publishing, Cham, 2016.

[6] Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.

[7] Michael Bratman. *Intention, Plans, and Practical Reason*. Cambridge: Cambridge, MA: Harvard University Press, 1987.

[8] Samuele Burattini, Alessandro Ricci, Simon Mayer, Danai Vachtsevanou, Jérémy Lemee, Andrei Ciortea, and Angelo Croatti. Agent-oriented visual programming for the web of things. 2022.

[9] Andrei Ciortea, Olivier Boissier, and Alessandro Ricci. Engineering world-wide multi-agent systems with hypermedia. In *Engineering Multi-Agent Systems*, pages 285–301, Cham, 2019. Springer International Publishing.

[10] Andrei Ciortea, Olivier Boissier, Antoine Zimmermann, and Adina Magda Florea. Give agents some rest: Hypermedia-driven agent environments. In Amal El Fallah-Seghrouchni, Alessandro Ricci, and Tran Cao Son, editors, *Engineering Multi-Agent Systems*, pages 125–141, Cham, 2018. Springer International Publishing.

[11] Andrei Ciortea, Simon Mayer, Fabien Gandon, Olivier Boissier, Alessandro Ricci, and Antoine Zimmermann. A decade in hindsight: The missing bridge between multi-agent systems and the world wide web. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*. May 2019.

[12] Maria-Francesca Costabile, Daniela Fogli, Catherine Letondal, Piero Mussio, and Antonio Piccinno. Domain-expert users and their needs of software development. In *HCI 2003 End User Development Session*, 2003.

[13] Salem Ben Dhaou Dakhli and Mouna Ben Chouikha. The knowledge-gap reduction in software engineering. In *2009 Third International Conference on Research Challenges in Information Science*, pages 287–294, 2009.

[14] Andrea Facchinetti, Giovanni Sparacino, Stefania Guerra, Yoeri M Luijf, J Hans DeVries, Julia K Mader, Martin Ellmerer, Carsten Benesch, Lutz Heinemann, Daniela Bruttomesso, et al. Real-time improvement of continuous glucose monitoring accuracy: the smart sensor concept. *Diabetes care*, 36(4):793–800, 2013.

[15] Roy T. Fielding. *Architectural styles and the design of network -based software architectures*. PhD thesis, 2000. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2022-10-29.

[16] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, may 2002.

[17] Dominique Guinard. *A web of things application architecture: Integrating the real-world into the web*. PhD thesis, ETH Zurich, 2011.

[18] Mahdi Hannoun, Olivier Boissier, Jaime S. Sichman, and Claudette Sayettat. Moise: An organizational model for multi-agent systems. In Maria Carolina Monard and Jaime Simão Sichman, editors, *Advances in Artificial Intelligence*, pages 156–165, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[19] Carl Hewitt. Viewing control structures as patterns of passing messages. *Artificial intelligence*, 8(3):323–364, 1977.

[20] BRYAN HORLING and VICTOR LESSER. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(4):281–316, 2004.

[21] Jomi F. Hübner, Olivier Boissier, Rosine Kitio, and Alessandro Ricci. Instrumenting multi-agent organisations with organisational artifacts and agents. *Autonomous Agents and Multi-Agent Systems*, 20(3):369–400, 2010.

[22] Jomi F. Hubner, Jaime S. Sichman, and Olivier Boissier. Developing organised multiagent systems using the moise+ model: programming issues at the system and agent levels. *International Journal of Agent-Oriented Software Engineering*, 1(3-4):370–395, 2007.

[23] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Moise+: Towards a structural, functional, and deontic model for mas organization. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1*, AAMAS '02, page 501–502, New York, NY, USA, 2002. Association for Computing Machinery.

[24] Alan Turing Institute. Multi-agent systems. Accessed January 11, 2023 `https://www.turing.ac.uk/research/interest-groups/multi-agent-systems`.

[25] N. Jennings and M. Wooldridge. Software agents. *IEE Review*, 42(1):17–20, 1996.

[26] Nicholas R Jennings. On agent-based software engineering. *Artificial intelligence*, 117(2):277–296, 2000.

[27] Seung-Kyun Kang, Rory KJ Murphy, Suk-Won Hwang, Seung Min Lee, Daniel V Harburg, Neil A Krueger, Jiho Shin, Paul Gamble, Huanyu Cheng, Sooyoun Yu, et al. Bioresorbable silicon electronic sensors for the brain. *Nature*, 530(7588):71–76, 2016.

[28] Donald J. D. Mulkerne. Pert! what? why? how? *The Journal of Business Education*, 48(3):111–114, 1972.

[29] Alessandro Ricci, Michele Piunti, Mirko Viroli, and Andrea Omicini. *Environment Programming in CArtAgO*, pages 259–288. Springer US, Boston, MA, 2009.

[30] Alessandro Ricci, Mirko Viroli, and Andrea Omicini. Give agents their artifacts: The a&a approach for engineering working environments in mas.

In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, New York, NY, USA, 2007. Association for Computing Machinery.

[31] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[32] Balaji Varanasi and Sudha Belida. *HATEOAS*, pages 165–174. Apress, Berkeley, CA, 2015.

[33] World Wide Web Consortium (W3C). Web of things. https://www.w3.org/WoT/.

[34] World Wide Web Consortium (W3C). Web of things thing description. https://www.w3.org/TR/wot-thing-description10/.

[35] Danny Weyns, Robrecht Haesevoets, and Alexander Helleboogh. The macodo organization model for context-driven dynamic agent organizations. *ACM Trans. Auton. Adapt. Syst.*, 5(4), nov 2010.

[36] Danny Weyns, Andrea Omicini, and James Odell. Environment as a first class abstraction in multiagent systems. *Autonomous agents and multi-agent systems*, 14(1):5–30, 2007.

[37] Michael Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2009.