A SOFTWARE FRAMEWORK FOR SIMULATION STUDIES OF INTERACTION MODELS IN AGENT TEAMWORK

by

OMID ALEMI

B.Sc. Computer (Software) Engineering
Arak University, Arak, Iran.

THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
MATHEMATICAL, COMPUTER, AND PHYSICAL SCIENCES
(COMPUTER SCIENCE)

UNIVERSITY OF NORTHERN BRITISH COLUMBIA

2012

© Omid Alemi, 2012

Abstract

This thesis proposes a new software framework that facilitates the study of agent interaction models in early development stages from a designer's perspective. Its purpose is to help reduce the design decision space through simulation experiments that provide early feedback on comparative performance of alternative solutions. This is achieved through interactive concurrent simulation of multiple teams in a representative microworld context. The generic simulator's architecture accommodates an open class of different microworlds and permits multiple communication mechanisms. It also supports interoperability with other software tools, distributed simulation, and various extensions. The framework was validated in the context of two different research projects on helpful behavior in agent teams: the Mutual Assistance Protocol, based on rational criteria for help, and the Empathic Help Model, based on a concept of empathy for artificial agents. The results show that the framework meets its design objectives and provides the flexibility needed for research experimentation.

Contents

List of Figures

Acknowledgements

Above all, I would like to sincerely express my gratitude to my supervisor, Dr. Jernej Polajnar for his support during my study. This thesis would not have been possible without his experience, knowledge, advice, and patience.

I am thankful to my committee member, Dr. Desanka Polajnar for her invaluable advice on my thesis from a software engineering perspective.

Special thanks to Dr. Matthew Reid for his interest and time in serving in my supervisory committee.

I would like to thank my colleagues Narek Nalbandyan and Behrooz Dalvandi for their input and feedback on using the software developed as a part of this research.

Chapter 1

Introduction

One of the most important characteristics of the multiagent paradigm is the ability of agents to autonomously and intelligently *interact* with each other. Interactions enable agents to coordinate their activities, cooperate, collaborate in teams, and negotiate in a social environment. According to Wooldridge [2009], many researchers believe that, in the future, computation should be understood mainly as "a process of interactions". With the increasing dominance of networks and distributed systems, the focus of agent studies has shifted from single agents towards their social abilities and organizational structures, bringing agent interaction models into the forefront of agent research.

The development of agent interaction models is a challenging area in multiagent systems (MAS) research. The designer of an agent interaction model typically faces a large number of decisions with many possible outcomes, whose impact upon the system properties and performance is often difficult to predict. Simulation experiments have been a key method in the evaluation of possible designs since the earliest studies of the agent interaction models (e.g., the Contract Net Protocol introduced by Smith

[1980]). However, despite the growing research interest in agent interactions, universal simulation tools for such studies have not yet appeared.

In this thesis I propose a new software framework for building simulators that facilitate the design of a particular class of agent interaction models. The purpose of such a simulator is to help the designer of an agent interaction model to study the properties, and in particular the impact upon overall system performance, of the model in situations of practical interest. In general, an agent interaction model denotes a specific pattern of interaction that agents use in order to achieve a certain objective.

The primary application scope of the current framework version, as described in this thesis, is the study of interaction models for helpful behavior in teamwork of artificial agents. This topic is an active segment of the MAS research scene. Teamwork of artificial agents has become a mainstream research area in MAS [Aldewereld et al., 2004, Sycara and Sukthankar, 2006, Dunin-Keplicz and Verbrugge, 2011]. Following the research on human teamwork (e.g., [Lepine et al., 2000]) which suggests that the capability and willingness of team members to provide direct assistance to each other are important factors of team success, there has been an increasing interest in the study of helpful behavior in agent teamwork [Yen et al., 2004, Fan et al., 2005, Kamar et al., 2009, Polajnar et al., 2011, 2012]. The current framework is a design tool that is specifically tailored for agent interaction models for helpful behavior and facilitates their design and development.

The framework supports interactive experimentation that allows the user to interact with the simulation model and dynamically change its properties, as well as other parameters of the experiment. In order to make the interactive experimentation efficient, the framework's design includes mechanisms for early feedback that allow

the user to immediately access the results of an experiment in progress and observe how agents' behavior reacts to the dynamic changes. These mechanisms help the designer to eliminate the undesirable features of the agent interaction models, as well as unproductive experimental setups, in the early stages of the experimentation cycle.

An essential feature of the framework that facilitates early feedback is the *concurrent simulation of multiple teams* which employ different agent interaction models that the designer wishes to compare. This feature enables the experimenter to observe the behavior and performance of multiple teams at the same time. The designer can simulate multiple teams that use different agent interaction models but follow the same experiment scenario in identical task and environment configurations. The concurrency enables the designer to perform a comparative analysis during the simulation and draw conclusions as the experiment progresses. One can also simulate multiple instances of the same agent interaction model with slight differences in order to optimize the agent interaction model parameter settings. Another possibility is to compare a team that employs an agent interaction model to address a specific problem with teams that use substantially different mechanisms to address the same problem in identical circumstances.

The proposed framework uses a distributable architecture that allows the simulation to run on a network of computation nodes. One of the factors that increases the computational complexity of the simulation process is that each experiment requires a large number of runs in order to provide statistically significant results. The concurrent experimentation with multiple teams increases the computational requirements and the time needed to complete a simulation run. In order to overcome those limitations, the framework can spread the experiments over multiple computation nodes in order to generate the results faster.

Simulation of multiagent systems requires modeling of the environment in which agents are going to be situated in a way that is suitable for studying them. However, real-world problems have details that obscure the core elements that need to be studied. The world model must be free of such details in order to efficiently support simulation-based design-oriented comparisons between alternative agent interaction models. An approach that has proven to be useful in many areas of artificial intelligence is the construction of a suitable microworld, that only represents the essential elements of the problem and provides a highly simplified abstract model that serves as a vehicle for studying it. Examples of successful microworlds include the Blocks World, used in research on planning in classical AI [Russell and Norvig, 1995], and the Colored Trails [Gal et al., 2010], used in the study of human-agent decision making. In this thesis, we have developed a microworld that is inspired by Colored Trails but is designed to represent the concepts needed in the research of helpful behavior in agent teamwork.

The current implementation of the framework supports the study of agent interaction protocols (AIPs) for collaboration among the members of the same agent team. In general, AIPs define the legal sequences, and content types, of the messages that the agents are allowed to send and receive in prescribed scenarios [Paurobally and Cunningham, 2002, Dunn-Davies et al., 2005]. AIPs can be standardized and included in libraries that can be used in different MAS platforms. There have been a variety of AIPs developed such as the Contract Net Protocol, different auction protocols (English Auction, Dutch Auction), and negotiation protocols [Wooldridge, 2009]. Agent interaction protocols rely on a shared message passing infrastructure which allows them to send messages directly to each other. In this research, a message passing mechanism is developed and validated in the framework.

The framework's design strikes a balance between specialization and extendibility. On the one hand, in order to keep this research within the limits of a Master's thesis, its current version is restricted to mainly support the study of agent interaction protocols for helpful behavior in the context of teams consisting purely of artificial agents. On the other hand, the framework has an open architecture that allows its functionality and application areas to be extended in a number of different directions. By connecting the framework to external systems, such as MATLAB, various tasks can be automated and different functionalities can be added to the overall use of the framework without the need for modifying its core structure. In addition, the framework's open architecture supports implementing different communication mechanisms and MAS models which could be used for other types of MAS research. The current restrictions do not exclude the possibility that some of the solutions developed in this thesis may have a wider scope and be applicable, for instance, to selfish agents or to individual interactions without an immediate group context.

Interoperability is another important aspect of the framework. It can be achieved in different ways. First, external systems can create and run experiments and access the simulation results; this eliminates the need for the framework to include different functionalities that already exist in other systems. An instance of such interoperability has been demonstrated in our research projects by connecting a simulator built in our framework to the MATLAB's Global Optimization Toolbox. Such a connection allows the interaction model designers to use optimization algorithms in order to determine the optimal configuration for their model. Second, the framework allows its MAS model to employ external agent reasoning engines to provide complex reasoning capabilities for its agents.

The design of the framework remains open to a number of extensions to support

a wider scope of research on agent interaction models. First, in addition to message passing, two other communication mechanisms are also included as options in the general architecture model: indirect communication through environment and communication through shared storage. Second, different MAS models, that represent different sets of problems in the real world, can be incorporated into the framework. Those extensions do not require modification of the framework's architectural structure.

The insights concerning the absence of suitably flexible simulation tools for AIP design in the domain of agent teamwork, and the need to develop a new software framework for that purpose, have developed gradually in the course of our studies of interaction protocols for helpful behavior in agent teamwork at the University of Northern British Columbia (UNBC) [Polajnar et al., 2011, Nalbandyan, 2011, Dalvandi, 2012, Polajnar et al., 2012]. During my participation in the ongoing MAS research at UNBC, I have examined and evaluated the development of AIPs for agent teamwork, and have identified the requirements for developing a design tool to facilitate their design and study.

The design of the framework proposed in this thesis has been incrementally refined in interaction with the AIP research projects that employed its successive versions in their simulation experiments. One of the projects has been the study of the mutual assistance protocol (MAP) [Nalbandyan, 2011, Polajnar et al., 2012], which uses a bilateral approach for deciding whether an agent should perform an action to help a teammate. Another project has investigated how incorporating empathy into teamwork of artificial agents, as a mechanism for triggering help, can improve the teamwork performance [Polajnar et al., 2011, Dalvandi, 2012]. Two other agent interaction protocols that use unilateral help approaches have also been modeled for comparative

studies of the MAP. The ability of the framework to model different types of AIPs has thus been validated through application in ongoing AIP research. The interoperability of the framework with the MATLAB Global Optimization Toolbox has been successfully used to perform the optimization of the empathic help model. A dynamic teamwork environment has been modeled in a microworld and used for various experiments. Finally, the framework's distributable architecture has been tested and validated for its performance.

The rest of the thesis is organized as follows. Chapter 2 covers the necessary background and related work. Chapter 3 describes the research problem addressed in this thesis and specifies my motivation and objectives for designing a new framework for studies of agent interaction protocols. The next three chapters describe different aspects of the framework. First, I explain the architecture of a generic simulator that can be built using the framework in Chapter 4. In Chapter 5, I present the approach to the modeling of AIPs, and describe a common general world model for agent teamwork as well as two different specializations of the world model that lead to two separate simulators, each designed for a different class of helpful behavior AIPs. Finally, Chapter 6 describes the use of those simulators in conducting experiments with the two groups of AIPs. Chapter 7 presents an evaluation of the framework, and Chapter 8 the conclusions and future work.

Chapter 2

Background and Related Work

This chapter presents the necessary background information and an overview of the previous work in multiagent systems, agent interactions and interaction protocols, agent teamwork, engineering of agent interactions, and simulation of agent interactions.

2.1 Multiagent Systems

There is no widely accepted definition of an agent or a multiagent system. According to Wooldridge [2009], a multiagent system consists of multiple agents that interact with each other. From another perspective, Shoham and Leyton-Brown [2008] define multiagent systems as "...systems that include multiple autonomous entities with either diverging information or diverging interests or both".

Wooldridge and Jennings [1995] define an *agent* as a computer system that exhibits autonomous behavior, is situated in an environment, and pursues its objectives. Wooldridge [2009] also specifies the capabilities that an *intelligent* agent is expected to have: *reactivity*, the ability of the agent to perceive the environment that it is situated in and respond to perceived changes; *proactiveness*, the ability of the agent to perform goal-directed behavior by taking the initiative; and *social ability*, that lets the agent meaningfully interact with other agents and/or humans.

Applications of MAS vary from space applications [Sierhuis et al., 2003] and manufacturing [Monostori et al., 2006] to electronic commerce [Luck et al., 2003] and social sciences [Sun, 2006].

2.2 Agent Interactions and Interaction Protocols

Agent interaction is one of the central aspects of multiagent systems and agent-oriented design. Agents can interact in different ways to achieve complex tasks by coordinating their activities and behavior [Weiss, 2000]. The nature of such interactions varies from being competitive to being cooperative. Furthermore, agent interactions can be implemented using different communication mechanisms. Three different communication mechanisms that are discussed in this thesis are: message passing, shared-storage communication, and implicit communication through the environment.

Message passing is the most commonly used communication mechanism in MAS. In message passing, a sender agent sends a message to the receiver directly by knowing its address [Uhrmacher and Weyns, 2009]. In order to achieve complex tasks

using message passing, often a number of messages need to be sent back and forth between agents in some meaningful sequence. Although each message participates in the interaction, the final outcome of the interaction is the result of all messages being exchanged together. To ensure the successful outcome of such interactions, certain constraints and rules need to be used to manage them efficiently. These constrains and rules that are imposed to the messages are defined by an agent interaction protocol (AIP) [Chen and Sadaoui, 2003, Paurobally and Cunningham, 2002, Dunn-Davies et al., 2005]. AIPs define the legal sequences, and content types, of the messages that the agents are allowed to send and receive in prescribed scenarios. There have been a variety of AIPs developed, such as the well-known Contract Net Protocol [Smith, 1980], different auction protocols (English Auction, Dutch Auction), negotiation protocols, and protocols for helpful behavior.

In a shared-storage communication, agents interact through a shared memory to store and retrieve information [Fortino and Russo, 2005] in order to solve a given problem. An important class of artificial intelligence (AI) systems for distributed problem solving that rely on shared-storage communication are the blackboard systems.

The blackboard concept is best described by Corkill [1991] as an approach similar to a group of human experts working on a problem:

> Imagine a group of human specialists seated next to a large blackboard. The specialists are working cooperatively to solve a problem, using the blackboard as the workplace for developing the solution.
>
> Problem solving begins when the problem and initial data are written onto the blackboard. The specialists watch the blackboard, looking for an opportunity to apply their expertise to the developing solution. When a

> specialist finds sufficient information to make a contribution, she records the contribution on the blackboard, hopefully enabling other specialists to apply their expertise. This process of adding contributions to the blackboard continues until the problem has been solved.

Blackboard systems generally consist of three main components: knowledge sources (agents), a shared storage, and a control component. The knowledge sources are software agents that contain the knowledge and expertise needed to solve a specific sub-problem. The agents in this model do not necessarily need to be aware of other agents and their special expertise in the system and are responsible to contribute to solving the main problem whenever they can solve the sub-problems regarding their specialty. Therefore, each agent can have its own internal architecture, programming paradigm, and knowledge representation which suit its own expertise. The blackboard is the global memory that may contain different data structures such as input data, partial solutions, and other data needed in different stages of the problem solving as well as providing a medium for communication and interaction between agents. The control component is responsible for execution of the system and the problem solving by notifying each agent whenever they can contribute to solve the problem. The main structure and the role of the control component differ in each system.

Implicit communication through environment is a form of indirect communication that is based on concepts taken from biology and ethology where animals perform collective behavior by using signals left in their environment as a means of communication [Uhrmacher and Weyns, 2009]. Keil and Goldin [2006] define indirect interaction as an interaction through making changes in a persistent environment so that the recipient agent can observe the changes. The environment needs to be persistent in the sense that it has a memory of interactions. As indirect interaction is a form of low-overhead

interaction which can be used by agents without sophisticated computational power [Holland, 1996], one of its main application areas is the kind of multiagent systems in which there could be no explicit task assignment or reasoning capabilities. In this class of MAS, agents could be simple entities without much computational power and they simply react to the signals they perceive in their environment in order to coordinate their activities.

Uhrmacher and Weyns [2009] specify two forms of signals for MAS: marks and fields. Marks are signs that agents drop on their way on the environment (which could be in the form of pheromones, tracks, objects, etc) so that other agents, by perceiving these signs, can interpret their meaning or purpose. Fields are signals that are spread in the environment and their intensity can reflect the distance between a source and a location in the environment. They are mostly useful for avoiding obstacles or finding desirable objects in the environment.

2.3 Agent Teamwork

Teamwork is the collaboration of agents in order to achieve a common task. According to Cohen and Levesque [1991], teamwork is more than just the collection of simultaneous and coordinated tasks being done by a group. What mainly distinguishes teamwork from other group activities is that team members share a mutual mental state. For example, agents typically have some common beliefs and joint goals. This mutual mental state affects, and is affected by, the mental states of team members. The collective activity is performed by individuals that share this mental state.

The motivation for the research on agent teamwork comes from the fact that in

most real-world applications, the agents are situated in an uncertain, highly dynamic environment. Such environments are constantly changing. Therefore, any attempt to build the team based on a fixed, predefined algorithm for coordination among team members will result in failures in the system [Tambe, 1997]. The research on teamwork in multiagent systems can be divided into two groups: teams of pure agents and teams of human-agent. In my research I will be mainly concerned with the teamwork among artificial agents.

A team, in order to act coherently and address the problems raised by uncertainties of the environment, must have the following characteristics: provide flexible communication among the agents, enable agents to monitor their teammates' progress, and allow reorganization and reallocation of resources to all the team members [Tambe, 1997].

In order to increase the teamwork performance, agents can perform helpful behavior by assisting their teammates through performing tasks, providing relevant information, or giving away their resources. Helpful behavior is becoming an active area of research [Yen et al., 2004, Fan et al., 2005, Kamar et al., 2009, Polajnar et al., 2011, 2012].

A certain class of agent teams, called expert teams, are those in which each member may have a unique set of skills and knowledge that distinguishes it from other team members. This set of skills and knowledge defines the member's expertise which is not easily transferable to other members. Different research approaches have been taken on this class of agent teams in MAS [Singh, 1991, Polajnar et al., 2011]. The autonomy and distinct expertise of team members influence the design of the interaction mechanisms for such systems.

Teamwork models have shown their effectiveness in real-world applications in which agents work together to jointly accomplish a particular task [Nair et al., 2003]. Examples include robotic soccer [Kitano et al., 1997, Palamara et al., 2009], simulations of urban search and rescue [Kitano et al., 1999, Kruijff et al., 2012], battlefield simulations [Tambe, 1997, Li et al., 2010], and personal assistant agents [Tambe et al., 2002, Yorke-Smith et al., 2012].

An observation from reviewing the work on agent teamwork and helpful behavior indicates that agent interaction plays a central role in effective teamwork. In order to successfully implement and incorporate teamwork and helpful behavior models into real-world multiagent systems, one needs to design and employ sophisticated and flexible interactions.

2.4 Engineering of Agent Interaction Protocols

2.4.1 Formal Methods

Formal approaches can be used to develop and verify agent interaction protocols. These approaches are often used to specify protocols and verify and validate their properties and are usually extensions to the methods used to develop protocols in distributed systems. In MAS literature, in order to capture, represent, and specify AIPs, different formal approaches have been introduced. These approaches are mostly based on Extended Finite State Machines [Lind, 2002], Extended UML [Lind, 2002], and Petri Nets [Cost et al., 1999]. In the following, a short overview of some of these works is presented.

In [Odell et al., 2001], the authors argue that while the current Unified Modeling Language (UML) framework provides a number of different properties that can be applied to AIPs, there are some extensions specific to MAS. In particular, they propose Agent UML, an extension of UML that is adopted for multiagent systems. AUML uses a layered approach for modeling AIPs in which different AUML notations, including statecharts, are used to represent different aspects of AIPs.

In [Mazouzi et al., 2002], the authors propose a generic formal approach for protocol engineering that translates semi-formal specification using Agent UML into Colored Petri Nets and introduces the Recursive Colored Petri Nets formalism.

Chen and Sadaoui [2003] introduce a generic formal framework to develop and verify AIPs based on a formal specification language called Lotos which is widely used in distributed systems. Their approach handles concurrency and synchronization, provides the correctness of AIPs in terms of safety, liveness, and fairness and uses a number of different tools to formally analyze and verify AIP specifications.

Mokhati et al. [2007] propose a formal framework that can be used to formally specify the behavior of MAS interactions and verify and validate them.

Dunn-Davies et al. [2005] introduce the propositional statechart formalism to represent AIPs. Their approach is based on the statechart formalism, a popular method included in the UML standard, and supports protocol verification and validation.

2.4.2 Multiagent Languages and Platforms

The growing interest in research on multiagent systems has resulted in the development of different programming languages and tools that could be used to implement such systems. Using agent-oriented languages rather than conventional programming languages enables the programmers to model problems from a MAS perspective and implement them in terms of cognitive and social concepts of MAS such as beliefs, goals, plans, roles, and norms. Agent-oriented programming languages that are currently in the multiagent systems literature vary from being completely declarative, to being completely imperative. There are also several hybrid approaches as well. Our discussion of agent-oriented languages below is based on a survey by Bordini et al. [2006].

For most of the agent programming languages, there are platforms that implement their semantics (e.g., Jason platform implements the AgentSpeak(L) semantics). There are also agent platforms that are not based on any specific programming language. These platforms instead focus on the underlying infrastructures for agents to coexist with each other and be able to find each other and communicate (e.g., Jade).

Most of the cognitive aspects of the agents are declarative by nature and thus, there have been more declarative languages proposed. Such languages often follow a strong formal logic-based approach. Examples are FLUX [Thielscher, 2005], MINERVA [Leite et al., 2002], KABUL and EVOLP [Alferes et al., 2002], DALI [Costantini and Tocchio, 2002], and ReSpecT [Omicini and Denti, 2001].

There are a few purely imperative agent-oriented programming languages as it is often not convenient to implement agent-oriented abstractions using an imperative programming language. One of the examples of such a programming language is the

commercial JACK Agent Language (JAL) [Evertsz et al., 2004] which extends the Java programming language instead of using a logic-based approach.

Many of the well-known agent languages provide both declarative and imperative features. While one can model agent's cognitive aspects in a declarative manner, these languages allow the use of some imperative code implemented in an external language through some special constructs. Examples of such hybrid languages are 3APL (An Abstract Agent Programming Language "triple-a-p-l") [Hindriks et al., 1999], AgentSpeak(L) [Rao, 1996], IMPACT [Subrahmanian, 2000], GO! [Clark and McCabe, 2004], and AF-APL (Agent Factory Agent Programming Language) [Collier, 2002].

Among different agent platforms and frameworks, it is worth to mention TuCSoN (Tuple Centre Spread over the Network), a framework for multiagent coordination [Omicini and Zambonelli, 1999]; JADE (Java Agent DEvelopment framework) [Bellifemine et al., 2005], a Java framework for the development of distributed multiagent applications; Jadex [Pokahr et al., 2005], a framework for the creation of belief-desire-intention (BDI) agents; and Jason [Bordini et al., 2008], an interpreter and framework for implementing agents using AgentSpeak(L).

2.5 Simulation of Agent Interactions

Simulation is an experimental computational method for designing, testing, and studying theories or real systems [Uhrmacher and Weyns, 2009]. It is mostly used in situations where conducting experiments with a real-world system is either impossible or expensive. Furthermore, often real systems are not fully controllable and therefore

it is not easy to design the desired experimental settings [Smith, 1980]. According to Shannon [1975], simulation can be defined as:

> The process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system and/or of evaluating various strategies (within the limits imposed by a criterion or a set of criteria) for the operation of the system.

Based on Uhrmacher and Weyns's [2009] point of view, the relationship of MAS and simulation is twofold: from one perspective, simulation techniques can be used to design, study, and run a MAS; from another perspective, MAS can be used as a modeling paradigm to study and understand real-world complex systems that are composed of many interacting entities. Different MAS platforms that are developed to support building multiagent systems such as JADE [Bellifemine et al., 2005] and Jason [Bordini et al., 2008] belong to the first category. The second category includes a variety of tools that are developed for modeling various application areas such as studying the behavior of agents in the stock market, network security, and understanding the consumers purchasing behavior. More examples of such tools can be found in [Nikolai and Madey, 2009]. In this thesis I am concerned about the first category.

Simulation experiments have been a key method in the evaluation of possible designs since the earliest studies of the agent interaction models. The Contract Net Protocol introduced by Smith [1980] is developed and evaluated through a specially designed simulator called CNET that simulated a real-world environment of distributed sensors and processors. Using that simulator, Smith was able to conduct experiments,

evaluate his protocol, and introduce the necessary refinements, which may not have been possible which may not have been possible without the simulator.

In research on agent interactions, and in particular AIPs, researchers often develop their own simulators because of the lack of a general simulation environment. As such simulators are built to specifically support certain interaction models, they often lack the flexibility that is needed in order to allow other researchers to experiment with other classes of interaction models. Examples of such work can be found in [Findler and Elder, 1995, Wanyama and Homayoun Far, 2007]. In each case, the authors have developed their own simulator which is specifically built to support their own work. These type of simulators often remain specific and are not used by other researchers or for other classes of interaction models.

In order to conduct simulation experiments with an agent interaction model, it can be implemented within multiagent platforms (as described in Section 2.4) along with a MAS which can be used for experiments. These platforms often provide embedded reasoning engines, communication infrastructure and management services, and support for ontologies. Some of the research projects which require simulation of agent interaction models have been done using these MAS platforms. For example, [Pasquier et al., 2011] uses 3APL or [Cheng et al., 2010] uses Jadex.

For the purpose of using simulation experiments in designing agent interaction models, MAS platforms have some limitations. First, a MAS platform does not provide the user with mechanisms for experimentation and analysis of the results. Second, a MAS platform usually does not provide the means by which the experimenter could isolate the impact of agent interactions from the impact of other aspects of agent behavior.

As an example of a wide-scope simulator, Kotenko [2009] introduces a domain-independent simulator for agent teams' collaboration and competition. His proposed simulation framework is based on three main components: models of agent teams; models of team interactions; and interaction environment model. It supports domain specific ontologies through a subject domain library.

Simulation of multiagent systems requires modeling the environment which agents are going to be situated in. Often even a simple real-world problem can result in a relatively complex simulation model. Such a complex model is full of details that can limit the study of the effects of different factors on the model's behavior and performance. A successful approach that has been proven to be useful in artificial intelligence is to model the world using the microworld approach. In a microworld model, only the essential elements of a system that reflect the essence of the system are included in the model. Thus, the result is a very simplified model which can be used as a means to study the real system without the need to deal with unnecessary details. The best known example of a microworld approach is the Blocks World [Russell and Norvig, 1995] used in research on planning in classical AI. It basically consists of a set of solid blocks that are on a tabletop, and a robot arm that is able to pick one of them at a time and place it on top of another block or on the table. Given an initial state, the goal is to build one or more block stacks according the the specified planning structure. This simple yet effective model allows researchers to investigate different planning mechanisms.

In the realm of agent interactions, the Colored Trails (CT) game [Gal et al., 2010] introduces another well-known microworld that abstracts multiagent task domains in which players negotiate and exchange resources in order to achieve their individual or group goals. It is mainly designed to study and investigate different decision-making

models in open mixed networks. CT has been used in MAS research and specifically in helpful behavior [Kamar et al., 2009]. However, with respect to its possible use in the design of the interaction models, CT has two main limitations. First, it does not support explicit representation of interaction models. Second, CT does not provide the basic tools for experimentation with the model being simulated.

Chapter 3

A New Framework for Studies of Agent Interaction Models

The purpose of this chapter is to explain the reasons that motivated the research addressed in this thesis, describe the research problem, and present the solution strategy. In Section 3.1, I explore the challenges in the design and quantitative analysis of agent interaction models that need to be addressed. In Section 3.2, I present the rationale for a new software framework that will serve as a design tool for the development of certain types of interaction models in agent teamwork. In Section 3.3, I formulate and explain some of the design principles that I will use in the construction of the new framework in order to ensure its effectiveness in its immediate application domain, and in order to keep it open for different application domains. Finally, my approach for presenting the framework in this thesis is briefly explained in Section 3.4.

3.1 The Challenges in Studying Agent Interactions

Agent interaction is one of the fundamental aspects of multiagent systems and agent-oriented design. It allows agents to coordinate their activities, cooperate, collaborate, and negotiate while trying to achieve their individual or collective goals. The study of agent interaction models has long been an active area of research in MAS; in recent years its practical significance has been on the rise, as many developing areas of MAS application require concrete engineering solutions (Section 2.2). The present section outlines some of the challenges encountered in such research and explains how they motivate the topic of this thesis.

We use the term *interaction model* to denote a specific pattern of interaction that agents use in order to achieve a certain objective; for instance, an auction mechanism can be used to allocate a resource on a competitive basis. An interaction model is described in terms of communication acts by which the participating agents exchange information. Multiagent systems use several types of communication mechanisms. Some of the common ones are: message passing, where agents send messages that are explicitly addressed to other agents; shared-storage communication, where agents post information in a storage area that is accessible to other agents; and implicit communication through the environment, where agents simply act upon the environment and modify its state in a way that is perceived by other agents (Section 2.2). Interaction models based on message passing are called *agent interaction protocols (AIP)*. They specify the structure and sequencing of the messages involved in the interaction, as well as the behavior of the agents that exchange them. The techniques by which agent interaction protocols can be specified and analyzed are derived in part from the techniques developed for protocols in networking and distributed computing (Section 2.4).

The development of an agent interaction model to the point that it can be effectively applied in practical system engineering is a challenging process. The model designer typically faces a large number of decisions with many possible outcomes, whose impact upon the properties and performance of a multiagent system as a whole is often difficult to predict. Once an agent interaction model has been conceived, and before it has been fully specified in detail, its designer needs to quantitatively analyze its impact on the system behavior and performance, determining its properties, advantages, and disadvantages in a specific context. In such studies, the performance of the multiagent system depends not only on the choice of the interaction model and its parameter values, but also on many other factors, such as: the characteristics of the environment in which agents are situated and perform their tasks; the task structure and complexity, including subtask assignment and resource allocation strategies; and the overall system organization. In order to evaluate an interaction model in context, one needs to study a multiagent system as a whole in a number of different configurations and situations.

In evaluating the early design of an interaction model, there is a wide variety of questions that the researcher needs to answer. Does the model design have the necessary formal properties, such as the absence of deadlock? When should an agent decide to initiate an interaction? When should an agent engage in an interaction initiated by another? How well does a MAS that uses the interaction model perform in different situations and scenarios? Given a specific MAS context, how does the interaction model compare with alternative models with respect to the desired system performance? Which parameter settings for the interaction model result in a better (or optimal) performance of the MAS? The answers to these and other questions are needed in order to better understand and evaluate the properties of a model in relation to its use in real-world environments. The feedback from the evaluation studies leads

to iterative refinements of the original design of the interaction model, enabling the researcher to draw conclusions regarding its suitability for a specific purpose, and to adapt it to a specific set of requirements.

While some of the questions outlined above can be addressed through formal specification of interaction models and their mathematical studies, simulation has proven to be the most widely applicable and effective method for conducting such evaluations. Simulation has been a key method in the evaluation of possible designs since the earliest studies of agent interaction models (e.g., the Contract Net Protocol introduced by Smith [1980]). It is a powerful and flexible tool in studying the properties and predicting the behavior of a complex system such as a MAS in situations of practical interest (Section 2.4).

Despite the growing research interest in agent interaction models, to the best of our knowledge, universal or wide-scope simulation tools that would effectively support all the necessary aspects of their quantitative studies in MAS context have not yet appeared (Section 2.5). This situation gives rise to three kinds of research challenges.

First, when embarking on a study of a relatively novel class of interaction models, one needs specialized simulation tools, based on a suitable set of abstractions arising from an analysis of the nature of the model, and a suitable set of facilities arising from an analysis of the experimenter's practical needs in mastering the design decision complexity. Typically, different researchers or research teams develop their own simulators for the particular types of interaction models that they study and the types of questions that they intend to pursue (Section 2.5). Such tools are then gradually perfected through ongoing research on interaction models within the selected class. The research on agent interactions is still in an early stage and various new models are expected to appear. For the time being, those new models are likely to require

the development of new specialized simulation tools.

Second, a major challenge in simulation studies of agent interaction models comes from the complexity of the world in which the agents are situated. Even an apparently simple real-world situation can result in a fairly complex simulation model. This slows down the simulation and limits the entire research process. The multitude of factors represented in the complex model also makes it difficult to infer the impact of individual factors on different aspects of the model's behavior and performance. This difficulty has long been recognized in the studies of artificial intelligence. A successful method of overcoming it has been the construction of a *microworld*, an abstract world model that represents the key aspects of the problem being studied in a highly simplified and yet relevant manner (Section 2.5). For each problem class, finding the right abstractions that achieve both simplicity and relevance requires a deep understanding of the problem and represents a major research challenge.

Third, there is the challenge of widening the application scope of the simulation tools. While developing support for specialized interaction models, one can try to keep the adopted solutions general, flexible, and extendible, and thus contribute towards a better understanding of what a more general simulation environment for a wider variety of interaction models would need to support and how it could be constructed. An important aspect of such generality is the openness towards different microworlds. This type of research may pave the way for the eventual construction of more general simulation frameworks for the study of agent interaction models.

The purpose of this thesis is to address all three challenges, in the specific scope outlined in the next two sections.

3.2 The Rationale for a New Framework

In this thesis, I propose a new software framework for early simulation studies during the design stage of agent interaction protocols for helpful behavior in teams of artificial agents. The framework design addresses the research challenges identified in the previous section by providing specialized support within its restricted application domain, by providing a core microworld open to variation, as well as by favoring architectural solutions that are open to generalization and extension beyond this restricted domain.

The motivation for the new simulator has developed gradually through my participation, as a member of the MAS research group at UNBC, in simulation studies of interaction protocols for helpful behavior in agent teamwork. The subject matter of those studies has been a new family of protocols, introduced in [Polajnar et al., 2011, 2012, Nalbandyan, 2011, Dalvandi, 2012], and further expanded in ongoing research. The simulation experiments have mainly aimed at the testing of key design ideas in the early stages of protocol development. The emphasis has been on investigating the impact of protocol design decisions upon the performance of an agent team that employs the protocol. One common experiment scenario involves comparisons between several agent teams that address identical tasks in identical environments, and have identical designs except that they employ different versions of the interaction protocol. In another common experiment scenario, one seeks to evaluate the protocol under study by comparing the performance of a team that employs it and the teams that employ alternative solutions (of a different nature) for the same purpose under the same circumstances. As the work progressed, it became apparent that the existing tools were not well-suited to the research tasks. A new specialized simulation tool needed to be developed, and the ideas shaping its design gradually became clear.

The proposed software framework is intended to serve as a *design* tool for interaction models in agent teamwork and differs from other kinds of simulators that one might use for agent teamwork. For example, in an operating search and rescue system in which robots interact with people, one may employ a simulator to train the human personnel in a realistic virtual environment that mimics the real-world experiences. That kind of simulator requires comprehensive modeling of the MAS in a real-world environment. In contrast, the framework introduced in this thesis is built for an entirely different purpose, namely for facilitating the design of agent interactions, which favors simple and abstract MAS models. In the rest of this section, I identify some of the key requirements arising from this particular orientation of the proposed framework.

The process of designing an agent interaction model typically involves series of incremental refinements. Each step includes a set of simulation experiments that provide an information basis for a specific design choice over a potentially large decision space. In order to properly guide the design evolution, the experimentation must be *interactive* and provide *early feedback* that allows fast elimination of undesirable model features, adjustments of the relevant model parameters, and avoidance of unproductive experiment setups. These and other core requirements need to be addressed at the level of the basic architecture of the software framework, as well as at the levels of its detailed design and implementation.

In order to have statistically significant results, one needs to repeat the same experiment a large number of times. On the one hand, while experimenting with a lower number of runs provides the results faster, such results cannot be used to draw reliable quantitative conclusions, although they may sometimes help roughly identify the qualitative trends. On the other hand, simulation may be computationally

expensive, and a high number of runs may require substantial computational power, or a long computation time, in order to produce the simulation results with the desired statistical significance. An experimenter in an early stage of the design process, who seeks to quickly move away from unproductive options, can greatly benefit from fast identification of unfavorable qualitative trends, despite their low level of statistical accuracy, while the assessments of near-satisfactory candidate solutions may require a high degree of statistical confidence. In order to be used effectively in a design process, the simulation framework needs to provide the experimenter with interactive dynamic control over repetitive runs based on the observation of the intermediate results. The user also needs to be able to dynamically modify different parameters of the MAS model under study, and to observe their effect on the system behavior and performance.

The proposed specialized simulation framework should not aspire to incorporate the functionality that is already well supported by available software packages, such as mathematical optimization or statistical analysis. Moreover, the complex functionality that will be required in the anticipated deployment of the model, such as specific reasoning engines or knowledge bases, should preferably not be imported and integrated into the framework. In both cases it is generally more appropriate to enable the framework to interact with external systems that provide the required functionality. Such interactions may involve scenarios in which the framework employs services of other systems, as well as scenarios in which it provides simulation services to other systems.

The proposed software framework should be adaptable and extendible so that one can easily widen its scope as a simulation tool. At a minimum, one should be able to modify the world model in order to experiment with different phenomena

and environments, and allow extensions that support other types of communication mechanisms and interaction models.

The framework also needs to provide the means for transition towards further stages of interaction model development, namely the prototyping of the model in concrete multiagent platforms and languages. This is required in order to allow the designer to make sure that the experimental results can be reproduced and applied in more realistic settings.

Many MAS applications in engineering and other fields involve agent teams with specialized individual roles that rely on different knowledge bases, similar to multidisciplinary human teams. In an *expert team* (as discussed in Section 2.3) each member has a unique set of skills and knowledge that distinguishes it from other team members and that may not be easily transferable to other members. In our studies of helpful behavior among team members, the multiagent system model involves agents that have their own expertise, which makes the cost of an action dependent on who performs it.

Teamwork of artificial agents has become a mainstream research area in MAS; within it, the study of helpful behavior increasingly attracts the interest of researchers (Section 2.3), but adequate simulation tools for the design of necessary AIPs are not readily available (Section 2.5). Having identified the resulting challenges and the requirements for overcoming them, we can now address the design principles for the proposed new software framework.

3.3 The Design Principles

The previous section has highlighted a number of characteristics that the new framework needs to have in order to fulfill its purpose. Those characteristics shape my approach to the design of the framework, from its high-level architectural definition to its detailed design and implementation. In the present section, I identify and explain the principles that will guide my design decisions presented in the next two chapters.

Interactive Experiment Control. The experimenter should be able to directly manipulate the models during simulation experiments. In support of this type of interactivity, the system should be able to display the simulation results as the experiment progresses, providing the feedback necessary for further decisions. By following this principle, the framework helps to reduce the decision space by allowing the designer of an interaction model to detect and eliminate undesirable features of the model.

Concurrent Simulation of an Experiment Scenario in Multiple Agent Teams. In order to address the challenges in comparative studies of interaction models, multiple agent teams can be simulated concurrently. As each team employs its own interaction model, using this approach the experimenter can study and compare multiple interaction models, or different variations of the same model, at the same time in identical environments and scenarios, and monitor all the results at once as the experimentation progresses.

Interactive Visualization During Simulation. The results of the simulation, as well as the state of the world, can be visualized and provided to the experimenter as they are being generated and updated. This visualization should be supported by a graphical user interface (GUI) provided by the framework.

Interactive Control Over Gradual Refinement of Accuracy. The statistical accuracy of the simulation results for a series of experiments can be refined gradually as the simulation process continues. In order to have statistically significant results, experiments need to be repeated a large number of times, causing the simulation process to take longer. This design principle seeks to balance the generation speed and the statistical accuracy of the results as a particular stage of the design process may require. For instance, the simulation might start with a relatively low number of runs for each experiment (which can be done in a reasonable time). This produces the initial results which represent the preview of the trend of the results in a timely manner but with a low statistical accuracy. Next, the simulator repeats the same experiments and consequently the accuracy of the results improves with time.

Microworld-based MAS Modeling. In order to provide a simple yet representative world model, the framework should employ a MAS model that is based on the microworld approach. This principle reduces the need to incorporate complex domain knowledge, and lets the agent reasoning focus on the essentials.

Low Coupling Between the Simulation Environment and the MAS Model. The framework should support modifying and replacing the MAS model through localized programming without the need to modify the rest of the framework. This is essential as supporting different MAS models makes the framework suitable for studying interaction models in a wider range of contexts.

Interoperability with other Systems. The framework should support interactions with other systems both as a client and as a server. This is required in order to use the functionalities that are already provided by such external systems but needed by the experimenter. In turn, the framework should allow external systems to define and run experiments and access the simulation results. The framework should also support

the employment of external reasoning engines or knowledge bases by its agents.

Distributable Architecture and Implementation. The framework should have a distributable architecture that allows the simulation to be either executed on a single computer or distributed across multiple nodes in a network. The main objective is to overcome the computational complexity and produce the simulation results faster. In particular, this implies that early feedback is delivered much faster to the interactive user, accelerating the design process.

3.4 The Approach in This Thesis

The framework proposed in this thesis can be used to build customized simulators for agent interaction models. Such simulators are tailored to simulate a specific group of agent interaction models in a desired context chosen by their designer. For that

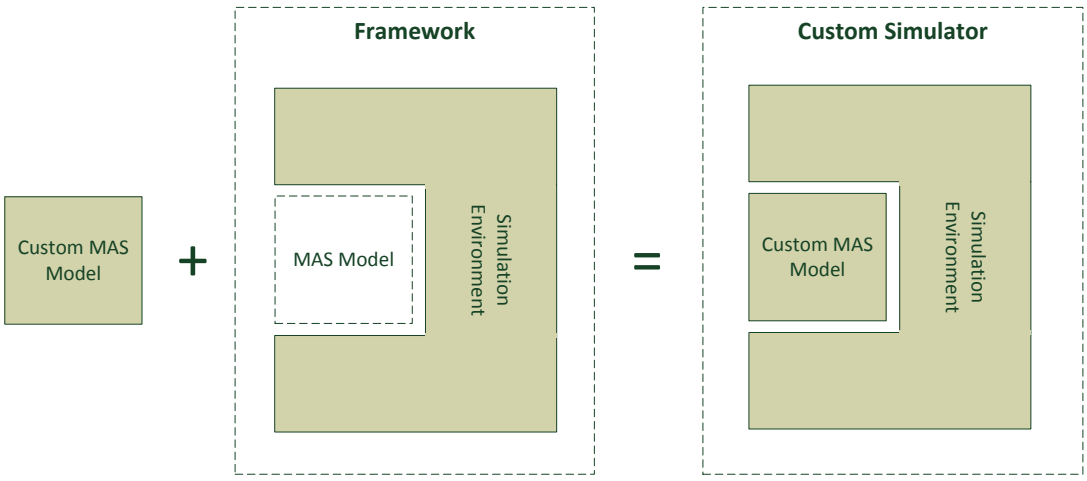

Figure 3.1: Building customized simulators using the framework

purpose, the framework includes a simulation environment that supports different

MAS models. By incorporating a specific MAS model to the framework, one builds a customized simulator (Figure 3.1).

In the next chapters, the framework is presented by elaborating its generic simulator architecture and the approaches used for building a simulation model. The MAS model presented in this thesis is specifically designed to support our studies of helpful behavior in agent teamwork. In order to elaborate the framework in more concrete terms, as well as to evaluate it, two different simulators that are built using the framework are introduced and explained.

Chapter 4

The Generic Simulator Architecture

This chapter presents the generic simulator's architecture of the framework. This architecture can be used to build simulators that are customized for different interaction models. The chapter begins by identifying the system requirements in terms of functional, non-functional, and domain-specific requirements in Section 4.1. The system structure and its high-level decomposition are explained in Section 4.2. In Section 4.3, the behavioral view of different components of the generic simulator is elaborated. Finally, in Section 4.4 the role of this generic architecture in creating customized simulation tools is explained.

4.1 The System Requirements

In order to specify the requirements of the proposed system, I have used the approach suggested in [Sommerville, 2004] and have divided the requirements in three categories: *functional requirements*, *non-functional requirements*, and *domain-specific requirements*. In the rest of this section, different elements of each group are explained.

4.1.1 Functional Requirements

The generic functions of the simulator are captured in the use cases represented in Figure 4.1. It illustrates the functionality of the simulator without including its interoperability features. Each use case is explained in more detail in the following paragraphs.

1. *Set up and Run Experiment:* The experimenter can set up an experiment and run it. This functionality is further decomposed into other use cases as follows:

    (a) *Load Experiment:* The experimenter can load a previously saved experiment setup from a file.

    (b) *Configure Experiment:* The experimenter can configure different aspects of an experiment.

        i. *Configure Team:* The experimenter can configure the parameters regarding the team(s) he/she wishes to study. This includes selecting the participating interaction model(s) and configuring them. In addition, other properties of a team can be configured in this use case (e.g. the team composition).

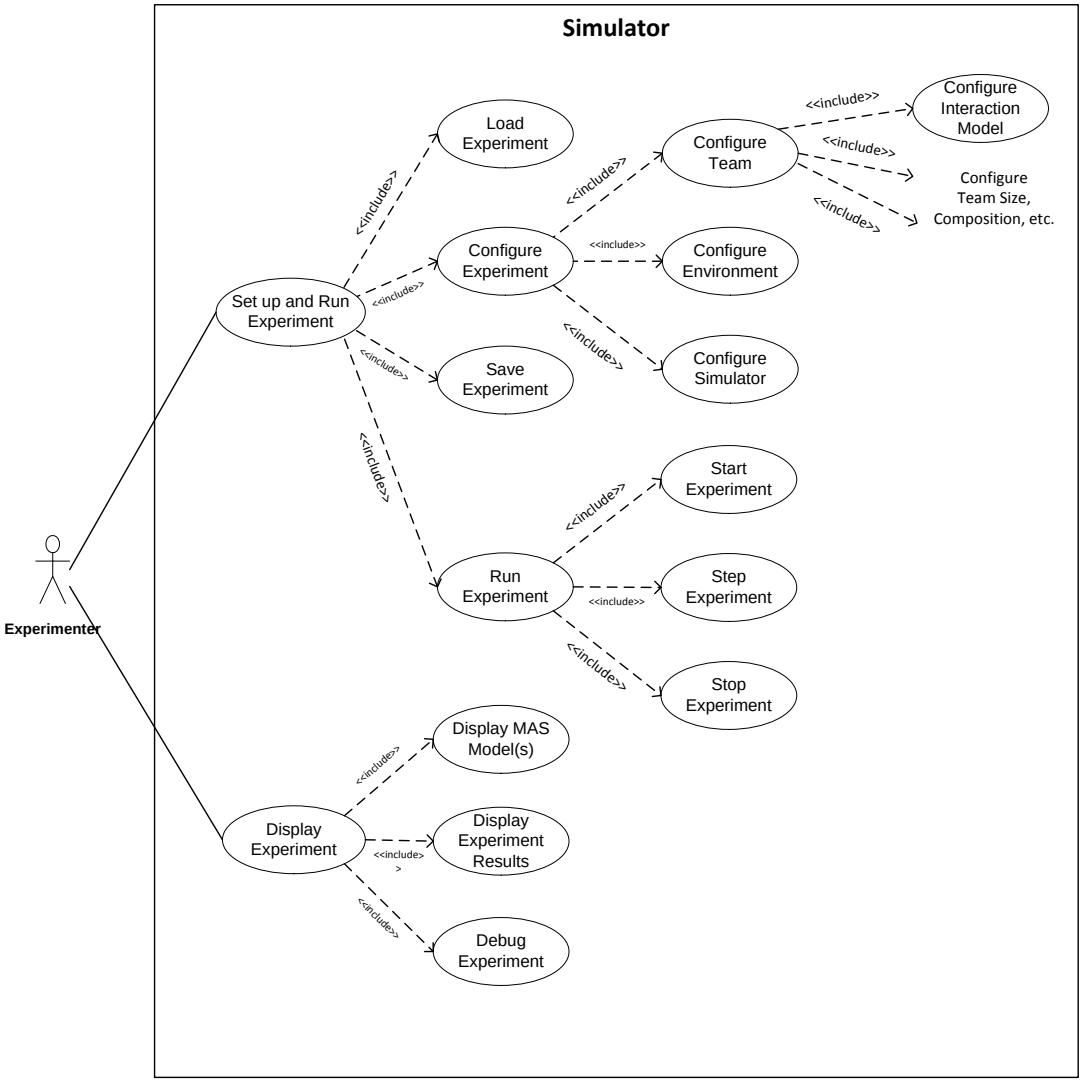

Figure 4.1: The use case diagram of the simulator as a stand-alone system

ii. *Configure Environment:* The experimenter can configure the properties of the environment in which the team(s) would be situated.

iii. *Configure Simulator:* The experimenter can configure different parameters of the simulator.

(c) *Save Experiment:* The experimenter can save the current experiment setup into a file.

(d) *Run Experiment:* The experimenter can control the execution of the simulation. experimenter can start and stop an experiment. In addition, he/she can execute the simulation step-by-step.

2. *Display Experiment:* The *Display Experiment* use case includes the following three use cases:

   (a) *Display MAS Model(s):* Upon experimenter's request, the system displays the MAS model(s) being simulated in a graphical representation to the experimenter. This includes the environment state, agents, and team related information.

   (b) *Display Experiment Results:* Upon experimenter's request, the system provides the experiment results to the experimenter. Results can be either visualized in graphs or displayed in a numerical format.

   (c) *Debug Experiment:* The experimenter can access and study the logs generated by the interaction model(s) and the environment he/she studies.

Simulator as a Client of an External System. The simulator can use the services provided by an external system. For example, the experimenter can request an external statistical software to process the simulator's results or use an external visualization package to draw customized charts based on the simulation results. This type of interoperability of the simulator is presented by the use case diagram in Figure 4.2 and explained below:

1. **The Simulator**

   (a) *Configure External System Interface:* The experimenter can configure the interface that is required to interact with the external system and use its services.

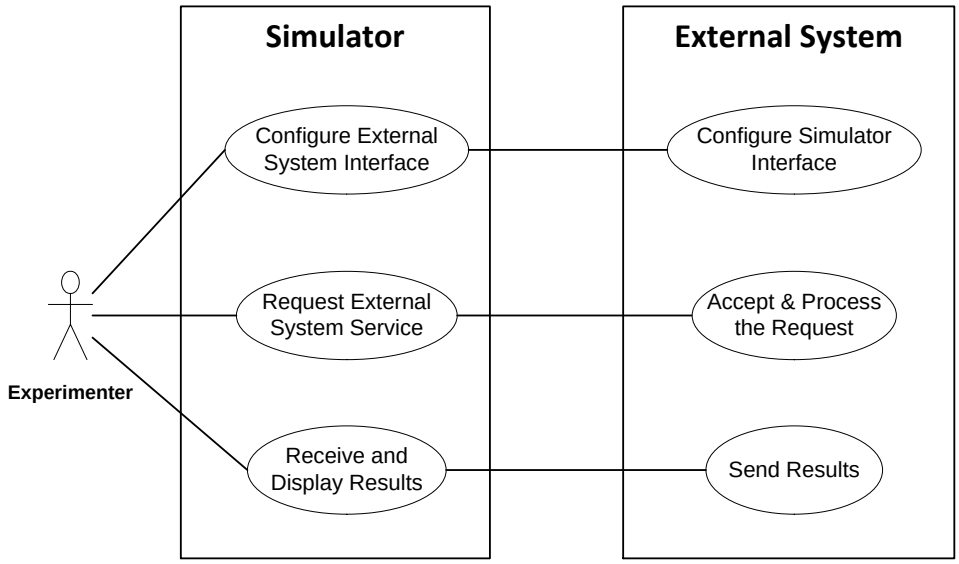

Figure 4.2: Simulator as a client of an external system

(b) *Request External System Service:* The experimenter can request a service from the external system.

(c) *Receive and Display Results:* The experimenter can ask simulator to receive and display the results generated by the external system.

2. **The External System**

(a) *Configure Simulator Interface:* The experimenter can configure the simulator interface to the external system through *Configure External System Interface*.

(b) *Accept & Process the Request:* The experimenter can ask the external system to accept and process the request through *Request External System Service*.

(c) *Send Results:* The experimenter can ask the external system to send the results to the simulator through *Receive and Display Results* interface.

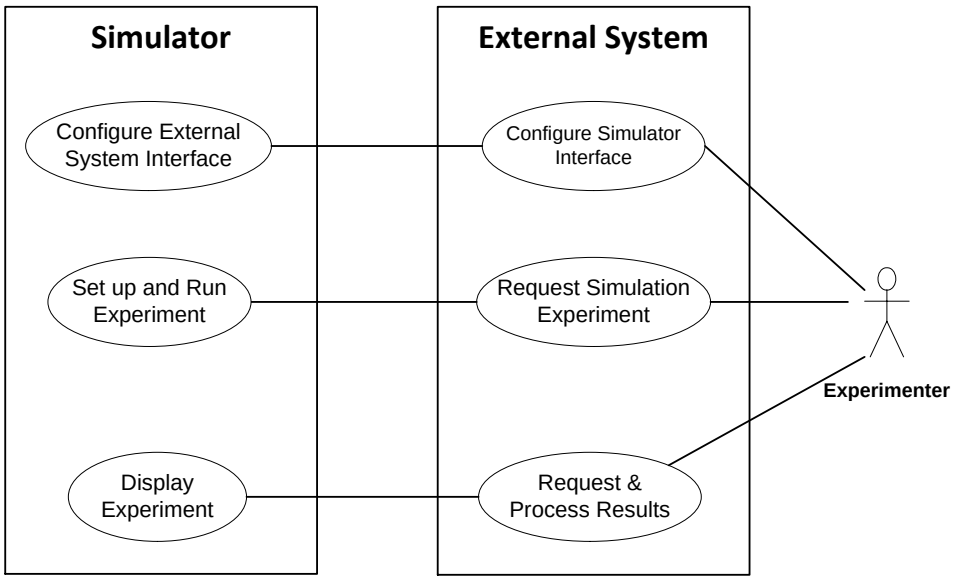

Figure 4.3: Simulator as a server of an external system

Simulator as a Server of an External System. The simulator can act as a server to an external system. The external system can invoke the simulator to run an experiment with a specified setup. The use cases for this form of interoperability are presented in Figure 4.3 and explained below.

1. **The External System**

   (a) *Configure Simulator Interface:* The experimenter can configure the external system to comply with the required interface to interact with the simulator.

   (b) *Request Simulation Experiment:* The experimenter can request an experiment from the simulator by providing the required properties of the experiment.

   (c) *Request & Process Results:* The experimenter can request the results of the experiment from the simulator, and have them processed by the external

system.

2. **The Simulator**

   (a) *Configure External System Interface:* The experimenter can configure external system interface to the simulator through *Configure Simulator Interface*.

   (b) *Set up and Run Experiment:* The experimenter can ask the simulator to run an experiment based on the given experiment setup through *Request Simulation Experiment*.

   (c) *Display Experiment:* The experimenter can ask the simulator to display the results for further processing in the external system through *Request & Process Results*.

Simulator in a Bilateral Client-Server Relation with an External System. The simulator can interact with an external system in a bilateral client-server manner. The experimenter can request a service from the external system through the simulator. In return, the external system, upon the experimenter's activation through the simulator, can specify and request an experiment to be done by the simulator. Once the simulation results are ready, the external system can further process them. At the end, the simulator can receive and display the external system's service results to the experimenter. For example, the experimenter can ask an external system to optimize a number of parameters of the model he wishes to study. The external system can create a number of different experiments, request that they be performed by the simulator, and use their results in order to find the optimal values of those parameters using its services. The use cases in Figure 4.4 represent this relationship and are explained in the following:

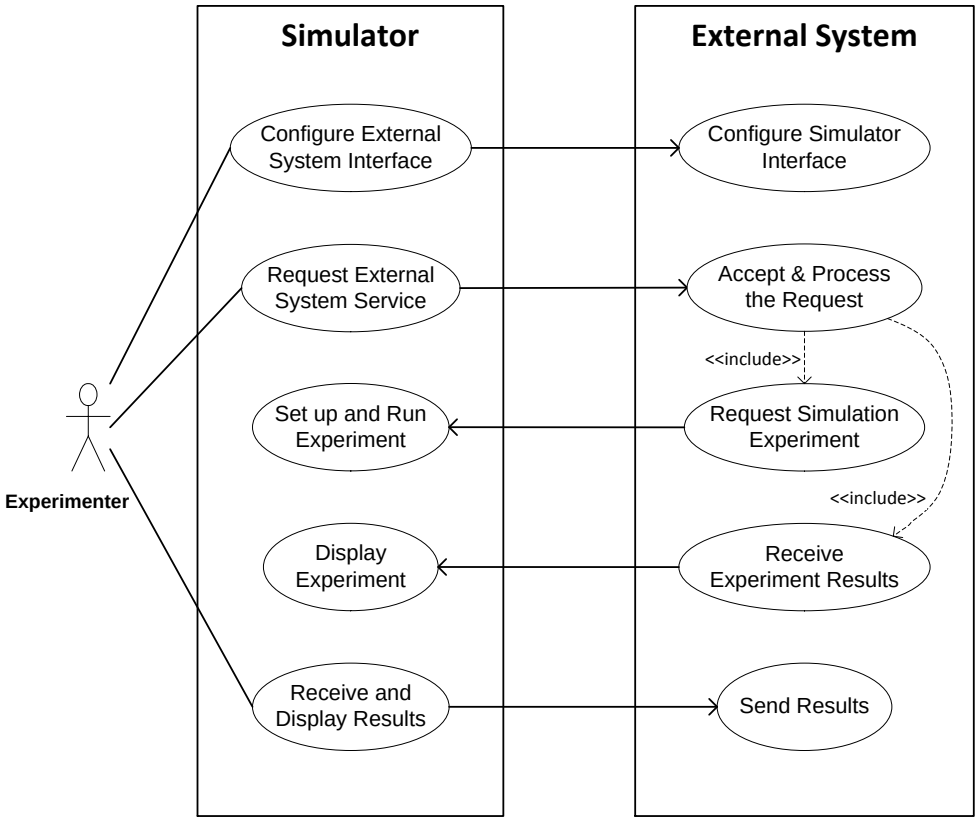

Figure 4.4: Simulator in a bilateral client-server relation with an external system

1. **The Simulator**

   (a) *Configure External System Interface:* The experimenter can configure the interface that is required in order to interact with the external system and use its services.

   (b) *Request External System Service:* The experimenter can request a service from the external system.

   (c) *Set up and Run Experiment:* The experimenter, through *Request External System Service* and *Accept & Process the Request*, asks the external system to create an experiment setup and pass it to the simulator to run it through *Request Simulation Experiment*.

(d) *Display Experiment:* The experimenter can ask the simulator to display its results through *Request External System Service* functionality of the simulator and *Accept & Process the Request* and *Receive Experiment Results* functionalities of the external system.

(e) *Receive and Display Results:* The simulator can receive the results of the external system's service and display it to the experimenter.

2. **The External System**

(a) *Configure Simulator Interface:* The experimenter can configure simulator interface to the external system through *Configure External System Interface*.

(b) *Accept & Process the Request:* The experimenter can ask the external system to accept and process the request through *Request External System Service*. This includes two other use cases:

i. *Request Simulation Experiment:* Upon the experimenter's request through *Request External System Service* and *Accept & Process the Request*, the external system can create an experiment setup and request the simulator to run it.

ii. *Receive Experiment Results:* Upon the experimenter's request through *Request External System Service* and *Accept & Process the Request*, the external system can receive the experiment results from the simulator.

(c) *Send Results:* The experimenter can request the external system to send the results to the simulator through *Receive and Display Results*.

4.1.2 Non-functional Requirements

The non-functional requirements of the simulator define the constraints on the functionalities that the simulator provides [Sommerville, 2004]. It is necessary to satisfy these requirements in order to make the system usable and reliable. These requirements are summarized below.

1. The simulator shall support modifications in the microworld, including its replacement with a different microworld, without the need to change the simulator's architecture.

2. The simulator has to provide message-passing mechanisms for direct communication between agents. In addition, its architecture shall allow the extensions for supporting implicit communication through environment and shared storage communication.

3. The simulator shall support a distributable architecture. In principle, the architecture should allow the simulation to be distributed on a cluster. One of the benefits of a distributable architecture is that one can increase the simulation speed through parallel execution. This increases the overall usability of the simulator.

4. The simulator's architecture shall support interoperability with external programs that provide various mathematical, statistical, optimization, and visualization techniques (e.g. Octave, Matlab, etc.). This is needed in order to provide the support for the processing and analysis of simulation results. It shall also be possible for such external systems to invoke the simulator's functionalities and run simulation experiments as a part of their processing or analysis tasks.

5. The simulator shall provide user interfaces that comply with the functional requirements of the simulator. At a minimum, this includes a graphical user interface (GUI) that allows interactive experimentation and a command-line interface that allows batch-mode experimentation.

6. The simulator shall support interoperability with multiagent platforms and agent reasoning engines. This can be used to allow the users of the simulator to use actual agent code and reasoning, rather than its simulated representation, whenever it is required. The purpose of this requirement is to allow a more realistic behavior of agents as well as to ease the transition towards an AIP prototype implementation on a full multiagent software platform.

7. The simulator shall use an AIP representation technique that makes the protocol descriptions intuitively understandable, easy to implement, and amenable to formal studies.

4.1.3 Domain-specific Requirements

The domain-specific requirements specify what is needed with respect to the application domain that the simulator is intended for. These requirements explain the constraints that reflect the fundamental MAS and agent teamwork concepts and have to be considered in the design and implementation of the simulator. In our case, as we are interested in studies of helpful behavior in agent teamwork, the models of multiagent systems presented here are tailored to ease such studies. As the simulator is intended to be used as a design tool for agent interaction models, the focus on designing the simulator shall be on agents interactions. In order to decrease the influence of other agent's components on the evaluation of agent interaction models, assumptions

and requirements on agent's deliberation, reasoning, and any domain knowledge have to be kept as minimum.

1. The simulator shall support a model of a multiagent system that includes representation of agents, environment, and task structure.

2. The MAS model shall present a team of agents. For our purpose, we make no assumptions about the team organization, subtasks assignment, and resource allocation.

3. The MAS model shall represent individual agents. Such agents need to have the ability to perform actions, interact with other agents, and maintain both local beliefs, formed through perception, and context beliefs, formed through communication with team members. In addition, for the purpose of studying helpful behavior, each agent shall have its distinct set of skills with respect to the set of possible actions in the environment, creating diversity in the team regarding the members' specializations.

4. The environment itself shall provide communication facilities to agents as well as a microworld model that can be used for studies of helpful behavior.

5. In order to use the environment for performance analysis of an interaction model, the communication facilities shall allow modeling communication costs.

6. The microworld should follow simple rules that do not require any particular domain knowledge or complex reasoning. This makes it suitable for studies of agent interactions and the required decision makings without the need to deal with any specific domain ontologies and agent programming.

7. The microworld shall define a set of possible actions, sets of rules for actions and perceptions, and a source of dynamism. Agents can perform actions on

the environment according to the provided rules. Each action may require a specific skill from agents. In addition, the environment should support actions that agents perform specifically to help another team member. The environment shall represent the dynamism that exists in real-world situations in a way that it affects agents' plans and actions. Different sources can be defined that generate such dynamics in the environment.

8. The microworld shall define a scoring metric that can be used for quantitative evaluations of the system performance.

9. Different elements of the microworld shall be parameterized so that the complexity of the environment or the level of its dynamism can be modified during experimentation by the system experimenter.

10. For our purposes, a team task structure shall be modeled. This task can be decomposed into subtasks. Once the task is assigned to the team, each agent in the team is assigned a particular subtask and has to complete it through performing atomic actions on the environment. These actions are determined through agent's autonomous planning abilities and the characteristics of the environment. This brings about the need to allow an agent to perform its own planning and possibly change its plan at any time.

11. The team shall be given a specific amount of resources initially which can be distributed among agents. Agents spend their resources in exchange for performing actions on the environment.

4.2 The System Structure

This section describes the high-level structural design of the simulator that provides the basis for its detailed design and implementation.

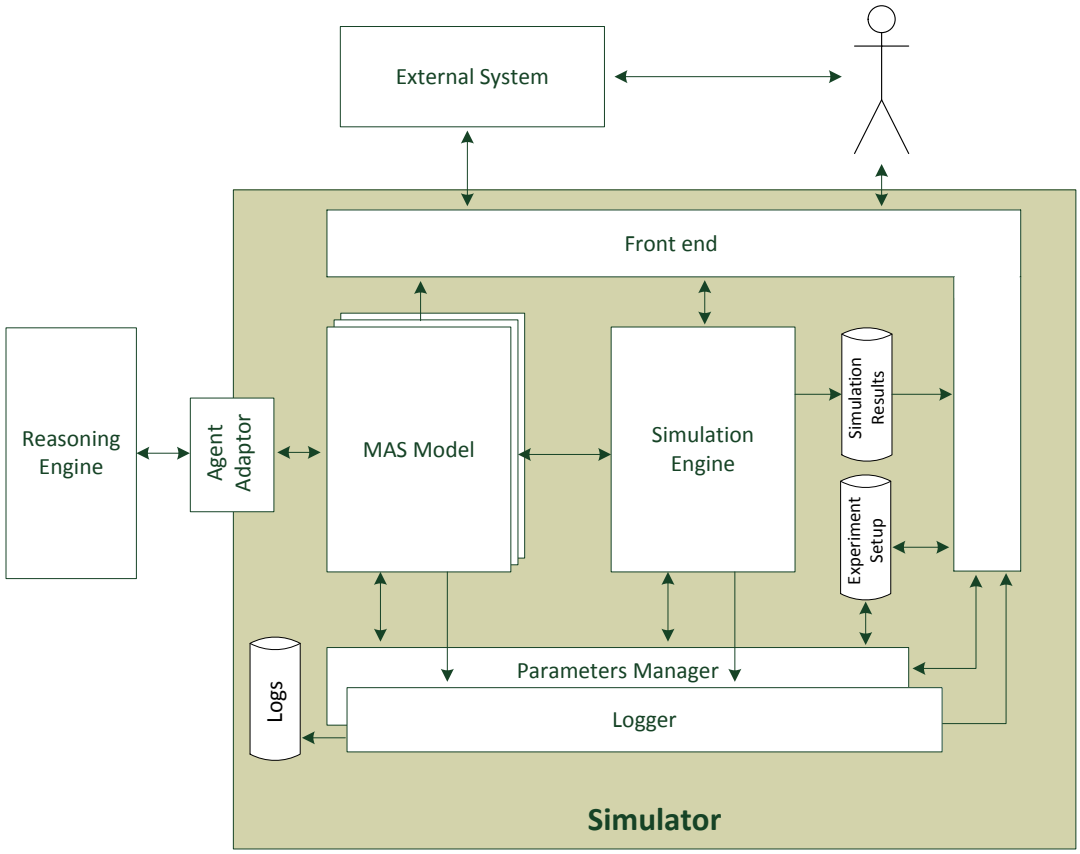

Figure 4.5: The high-level structure of the simulator and its external relations

At the top-level the simulator is decomposed into three main components: the *simulation engine*, the *MAS model(s)*, and the *front end* (Figure 4.5). The simulation engine controls the execution of the MAS model(s) and is responsible for coordinating the course of simulation. A MAS model represents a model of a multiagent system that would be used for simulation studies. The front end allows the user to control the simulation process and the presentation of the results. It is also responsible for

allowing the user to access all of the system functions specified by the use cases. Two other components of the simulator are responsible for managing system parameters and systems logs. These components are used by other components of the system.

The simulator provides the interoperability with other systems in two directions. First, the simulator can be invoked by external systems. For example, another program can control the simulator to automate creation and execution of a series of experiments. In another case, the simulator can be embedded within a system. Second, the simulator can use external reasoning engines to allow agents to use more sophisticated agent reasoning. The details of such interoperabilities are discussed throughout this chapter.

The simulator's architecture is designed to provide loosely coupled components. The benefit of this design is that it allows independent development of different components and simplifies its maintenance. In particular, different MAS and microworld models can be implemented within the simulator; as long as their designs comply with some general assumptions, there is no need to modify the other component of the simulator (e.g. the simulation engine). In the rest of this section, we describe the internal structure of the top-level components and outline the interfacing assumptions on which their designs are based.

4.2.1 The Simulation Engine

The simulation engine is the central component of the simulator. It is designed to control the execution of the model and to coordinate the operations of different components of the simulator. Its design is based on the assumption that the activity of each agent team can be viewed as a cooperative game that proceeds in discrete syn-

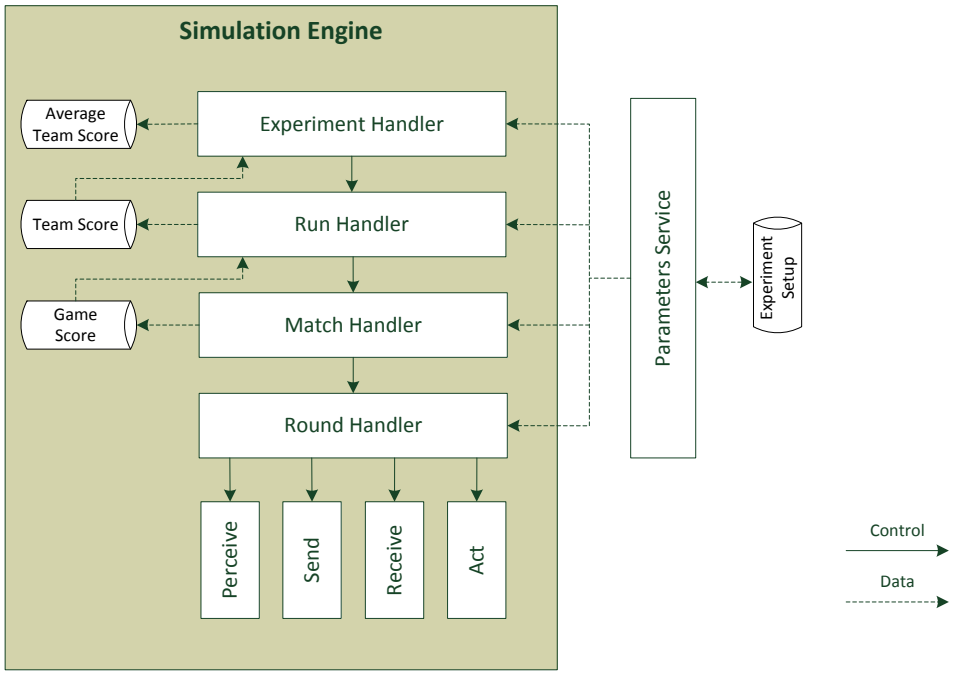

Figure 4.6: The simulation engine

chronous rounds, is guaranteed to end after a finite number of rounds, and at the end produces a final team score that indicates the level of the team's success in pursuing its task. The team's task involves a number of variable parameters whose values can be randomly generated or specifically set by the experimenter. An instance of the game, with a particular selection of task parameter values, is called a *match*.

The simulation engine executes simulation *experiments*. An experiment may involve a number of agent teams that play their matches concurrently. An experiment *run* consists of the concurrent execution of a fixed number of matches, with varying task parameters, by each agent team and the averaging of each team's scores. An experiment consists of a number of runs. That number must be sufficiently large so as to guarantee the desired level of statistical significance of the averaged team scores.

The *experiment handler* is responsible for managing the execution of an exper-

iment. This includes dealing with parameters, activating runs, and calculating the final team score. This module is activated by the front-end of the simulator and at the end produces the average team score over multiple runs.

The *run handler* manages the parameters and the execution of each run of an experiment. Once activated by the experiment handler, it generates a new combination of the environment settings and starts the simulation process. For each run, there is a score associated with the team which is passed to the experiment handler. The run handler activates the match handler for a fixed number of times defined by the experiment setup.

The *match handler* is responsible for the concurrent execution of the games played by individual teams. At the beginning, it initializes each team and the environment for a new game. At the end of the game, it reports the team score of each team for that game to the run handler. The game is played in discrete steps until it is over. For every team, each step is controlled by the team's round handler explained next.

The *round handler* executes the MAS model for one step in the game. In each step, this handler activates different handlers to cause agents to perceive, send, receive, and act. This module ensures proper synchronization between these activities.

4.2.2 Service Components

Parameter Management

The simulator uses a central parameter repository to store and manage all the experiment configurations and parameters of the simulation. Each component of the

simulator can use this repository to define, set, and read parameters. For each experiment, the user can set these parameters through this component. This includes all the parameters of the simulation engine, the environment, agents, interaction models, etc. Different entities can access their required parameters through this interface.

For easier identification, disambiguation, and safety, parameters are defined using a namespace convention. The entity that the parameter belongs to can be specified as prefix in the name of the parameter. For example, the cost of a unicast message that is used by the communication module can be prefixed by *comm* and be defined as "*comm.unicastcost*".

One of the main advantages of this component is to allow better interactive experimentation. Changes to any parameter shall be done through this component and whenever, at runtime, any component needs to access the value of a parameter it is done through this component. The front end can perform changes to the model at runtime without the need to restart the simulation.

Logger

The logger component collects and manages all the logs that other components of the simulator generate. Such logs can show the progress of simulation, or they can represent the internal state of an agent in its decision making process. Logs can be helpful for debugging agent interaction models and further analysis of the simulation. They can be set to be recorded in a file or be redirected to the GUI for user access. Also, logs can be classified into different levels, which can be individually set by the user to be recorded or displayed.

4.2.3 The Front End

The front end is responsible for providing the experimenter's access to the simulator. On the one hand, the user may directly interact with the simulator to design and perform experiments and access the results. On the other hand, external programs may use the simulator to perform experiments and process the simulation results.

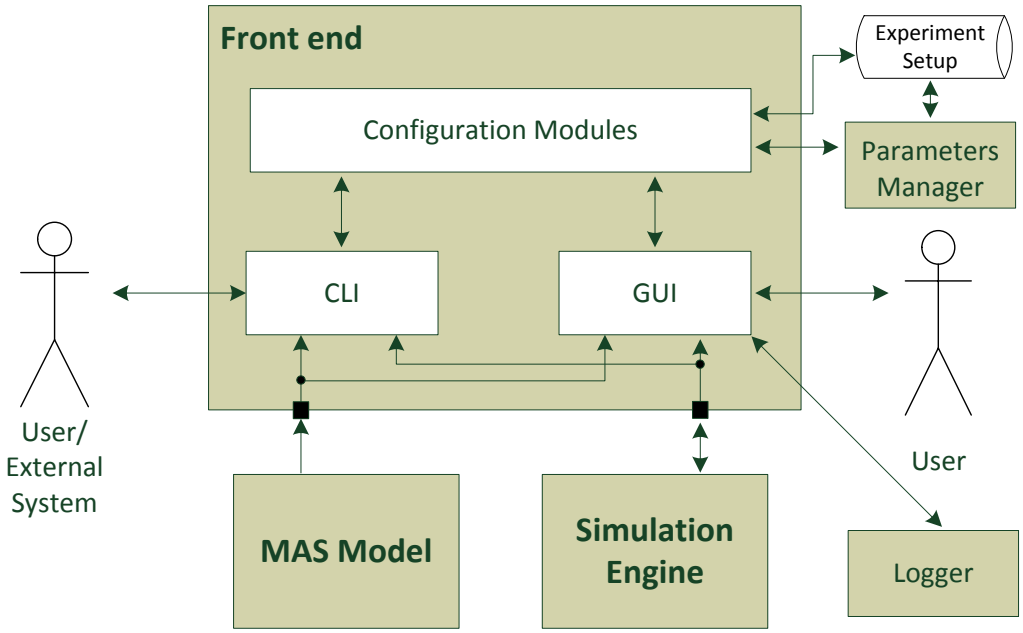

Figure 4.7: The Front-end component

The internal structure of the front end is presented in Figure 4.7. It consists of a configuration module, a graphical user interface (GUI), and a command line interface (CLI). The configuration module enables the experimenter to create the experiment setup required for simulation. The graphical user interface provides a graphical and interactive interface of the functionalities of the simulator to its users. The CLI allows the simulator to be invoked from the command line.

The configuration module sets the parameter values for the simulation. This in-

cludes simulator parameters, interaction model parameters, and the microworld parameters. Other components of the front end use this component to set up experiments.

The GUI provides an interface for interactive user access to the simulator. The user can create and manipulate experiment setups interactively and use the visualizations provided by the GUI to get feedback from the simulation model.

The CLI provides a non-interactive access to the simulator. Using CLI, the simulator can be invoked from command line and the experiment setup and the simulation results can be set to be read from or saved to files. The GUI is suitable for interactive experimentation by researchers while the command line front end is suitable for batch mode operations in text-based environments or where interactive experimentation is not required.

Simulator's Interoperability with External Systems

The simulator is designed to allow interactions with external systems. The interaction can be done in situations where the simulator is a client of an external system, the simulator is a server for an external system, or the simulator interacts bilaterally with an external system. This can be used to automate experiments and further analyze the results by other systems and increase the overall functionality of both systems. The general structure of invoking the simulator from external systems, and vice versa, is represented in Figure 4.8 and is described as follows.

In order to work with each external system, based on its structure and requirements, an adaptor can be developed to provide the interface between the simulator

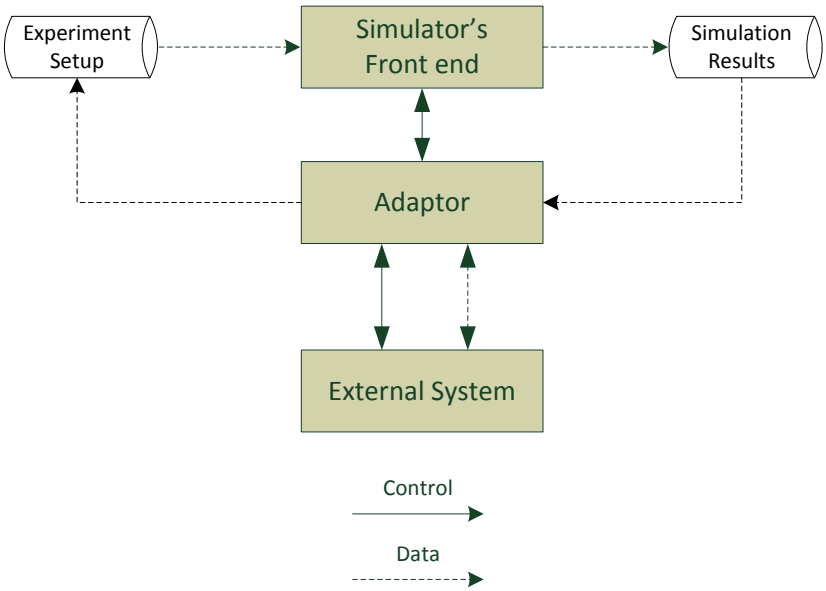

Figure 4.8: The interoperability of the simulator with external systems through its front end

and that system. The adaptor is responsible for the requirements of both systems and translates their interactions and data. Such an adaptor can interact with the simulator through its front end. Experiment setup can be stored in a file by the adaptor for simulator's use. Also, the simulation results can be stored in a file so that they can be processed by the adaptor and translated to the external system's input format.

4.2.4 The MAS Models

The MAS model component represents a multiagent system. This model is used for simulation and includes representations of the agents, the team context, the communication module, and the microworld (Figure 4.9). In addition, the simulator supports simulating multiple MAS models concurrently. First, the structure of a single MAS

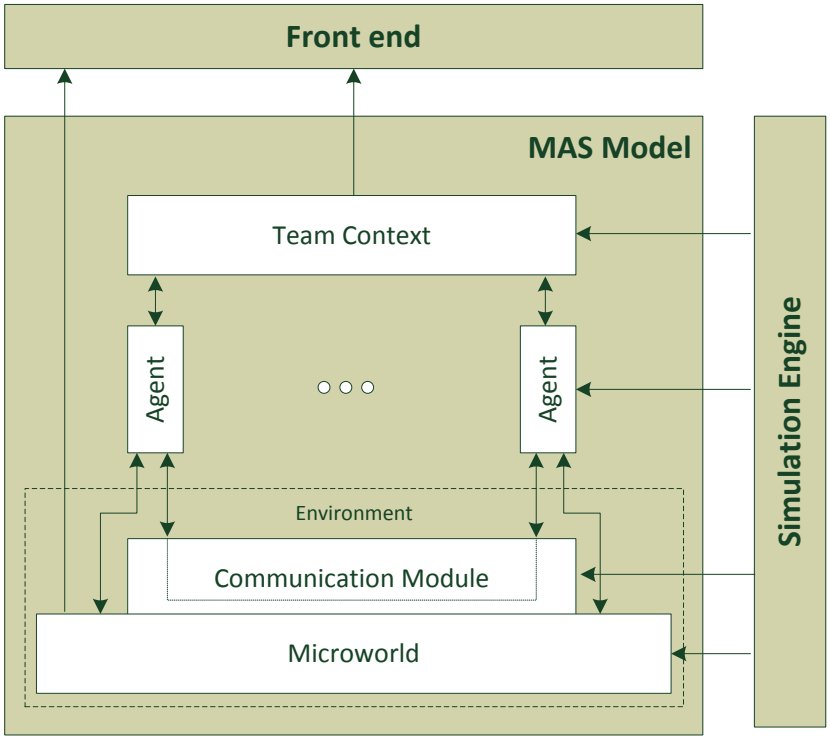

Figure 4.9: A MAS model within the simulator

model and its components, including agent, team context, the communication module, and the microworld, that represents a single team is described. Later, concurrent architecture of the simulation of multiple MAS model based on the presented single MAS model is explained.

Agent

The simulator employs a simple agent architecture which is built to support the execution of agent interaction models (Figure 4.10). The agent is activated by the simulation engine and is connected to the microworld and the communication module within the MAS model. The agent architecture consists of a belief base, a belief

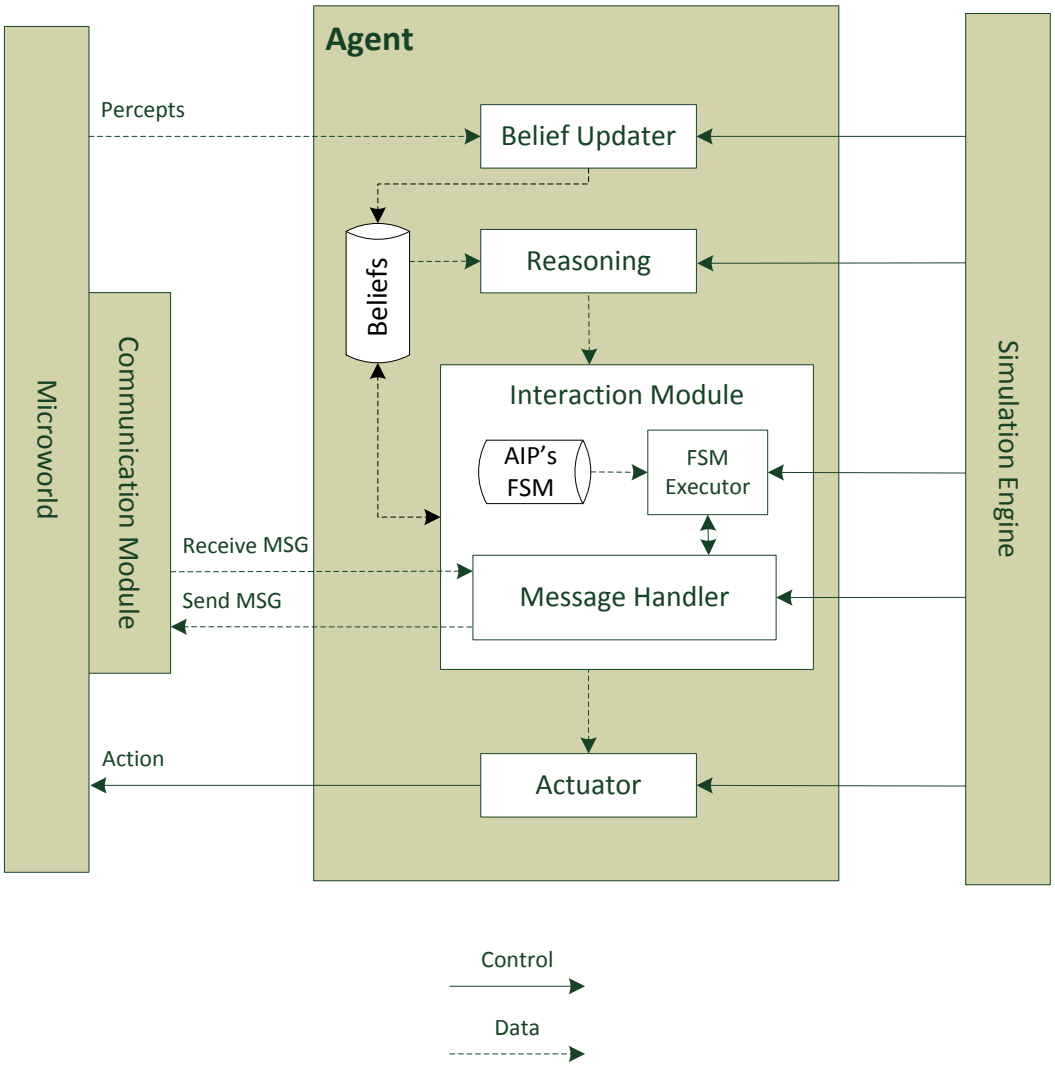

Figure 4.10: The agent architecture of the simulator

updater, a reasoning component, an interaction module, and an actuator component.

The belief base maintains all the beliefs of the agent. Agent's beliefs can be divided in two groups: local beliefs that are generated based on agent's percepts, and context beliefs that result from the agent's interactions with the rest of the team. The belief updater component receives the percepts from the microworld and updates the belief base and forms local beliefs. The interaction module also updates the belief base and

forms context beliefs based on the incoming messages.

The reasoning component allows the agent to deliberate about its current situation and make decisions. The agent uses its reasoning abilities for a number of different purposes. First, once a subtask is assigned to an agent, the agent can use its reasoning component to create a plan in order to achieve the subtask. Second, in each round of the game, the agent can reason whether to initiate an interaction model. Also, the agent can decide whether to engage in interactions initiated by other agents. Third, during executing interaction models, the agent can use its reasoning abilities to decide about its course of action in the protocol decision points. Finally, if the agent can choose dynamically which interaction model to use, the reasoning component will perform the deliberation.

The interaction module is a key component of our agent architecture. It contains a specification of an AIP, where the AIP is defined as a finite state machine (FSM); an FSM executor, which can execute the AIP's FSM; and a message handler, which allows the agent to create, send, and receive messages using the communication module of the MAS model.

The actuator component performs the chosen action, which is specified as a result of agent's reasoning and interactions, using the interface provided by the microworld (explained later in this section).

The reasoning component of the agent architecture supports reasoning that can be specified using regular imperative programming languages. However, it does not provide built-in support for agent-oriented programming languages and reasoning. The design cases considered in this thesis that use the framework need simple reasoning that is implemented using an imperative programming language. However,

some research may require the use of an agent-oriented language to perform more realistic reasoning. The agent architecture supports the use of external agent reasoning engines in order to enable agents to execute complex reasoning implemented in an agent-oriented programming language. This feature is built as an option in the simulator so that one can enable it when it is needed for a particular research problem.

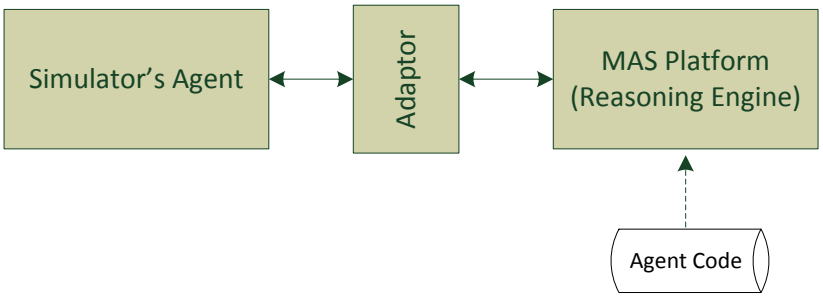

Figure 4.11: Using an adaptor to connect to external reasoning engines

The simulator's agent architecture can connect to an external reasoning engine (found in MAS platforms) using an adaptor (Figure 4.11). The purpose of using an adaptor is to translate the interactions between simulator's agent architecture and the reasoning engine. As long as such adaptor complies with the interface the simulator's agent needs, one does not need to modify the simulator's architecture in order to develop a new adaptor for a reasoning engine. This allows using different reasoning engines without being restricted to a particular one and at the same time providing the same interface to the agent architecture of the simulator. As a result, one can implement a specific adaptor for a reasoning engine he wishes to use without modifying the simulator.

Team Context

The team context module represents an agent team in a MAS. Agents in the same team are situated in the same environment and are able to interact with each other. The team context module conceptually holds the agents together by maintaining team-related information such as team's task structure, resources, and performance metrics.

The Communication Module

The communication module provides the facilities that are needed for agent communication. The current version of the simulator provides an implementation for a message passing system to support AIPs. In addition, the architectural support for a shared storage module (that can be used to implement a blackboard system) and communication through the environment is discussed here.

The message passing module consists of a message structure and a direct communication network between agents. In order to support AIPs, agents shall be able to create and recognize different types of messages. In addition, they shall be able to send messages directly to other agents or broadcast a message to all team members. In the following, the details of the message structure and the communication network is described.

A message is defined by the structure represented in Figure 4.12. Each message is built based on a set of fields in the form of $< key, value >$. The first three fields are mandatory as they identify its sender, receiver, and the message type. The sender and receiver fields contain the identification of agents involved in the message passing. The

Message	
Sender	...
Receiver	...
MSG_Type	...
key	*value*
key	*value*
...	
key	*value*

Figure 4.12: The message structure

message type specifies the type of the message among all possible types of messages that a protocol includes. This can be specified by the AIP designer. The rest of the fields can be used as they are needed based on the message type constraints. In order to have the least assumptions on the ontology and language, the contents of the *key* and *value* can be anything that can be encoded in a string.

Agents in the same team can exchange messages through a complete synchronous network with single-message unidirectional channels (as described in [Lynch, 1996]). In order to send a message from agent A to agent B, A has to put the message on the channel that exists between A and B. Afterwards, B can access the message once it needs it. In a similar manner, an agent can broadcast a message by putting it to all the channels connecting it to other agents.

A typical blackboard system consists of three main components: a group of knowledge base, the control unit, and the shared storage (blackboard). Applied to a MAS, each agent acts as a knowledge base, carrying special expertise to solve the given problem. Agents have to be able to post directly to the shared storage and to retrieve any

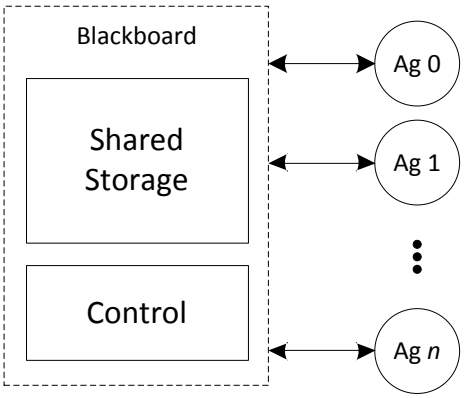

Figure 4.13: The architecture of a blackboard for the simulator

desired information from it. In addition, they may also receive notifications from the blackboard's control system. This overall architecture is represented in Figure 4.13.

The current architecture of the simulator supports incorporating the blackboard design described above. Agents can directly send messages to the blackboard, contributing to the common knowledge in the shared storage in the same manner as they exchange message to each other. Notifications from the blackboard can be sent through messages as well. Information retrieval can also be implemented through a pair of messages being sent and received.

The support for environment mediated communication shall come from both the environment and the agent model. A proposed design for supporting such communication by the simulator is represented as follows. Implicit communication can be done thorough changing the environment. Agents can create and use cues in the environment in order to implicitly communicate with each other and coordinate their activities. The environment should support creation of such cues. The simulator design supports such environment characteristics. In a finite gird-based environment,

each field (cell) can hold a number of different cues. Each cue would be implemented by an object and might have different attributes associated with it.

In this model, one agent's modifications in the environment becomes another agent's cue. An agent must be able to (1) perceive the cues on the environment, and (2) make modification in the environment (create cues) through performing actions. In the simulator's design, agents are allowed to perceive their environment, this includes the properties of individual cells which can include a list of all possible cues. Also, specific actions can be defined which make such modifications in the environment by creating cue objects.

The Microworld

The microworld, in this version of the simulator, models a cooperative game, played in discrete steps. The game is played on a rectangular board of colored cells. It is inspired by the Colored Trails (CT) game [Gal et al., 2010], designed and implemented independently. It models a dynamic environment which is designed to be used for studies of helpful behavior.

The microworld component contains a number of different modules as demonstrated in Figure 4.14. These modules interact with each other and provide the required functionality of the microworld. The microworld provides interfaces to the MAS model and the simulation engine to access and control these modules. The microworld modules are described as follows:

Board The board is a rectangular grid of colored squares and is where agents are situated. It allows agents to perform their actions using its *act* interface.

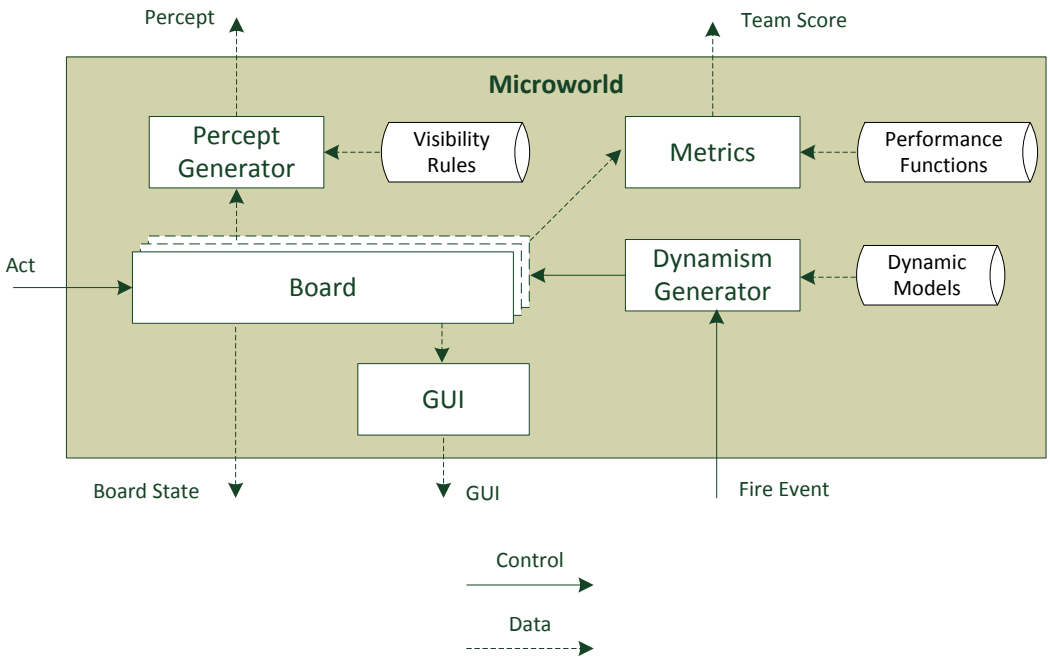

Figure 4.14: The structure of microworld

Percept Generator The microworld generates percepts per each agent. During the perception stage, a set of percepts generated by this module is passed to each agent. These percepts are generated based on the visibility rules of the microworld allowing the modeling of scenarios with partially observable environment.

GUI The microworld can supply an implementation of a graphical component that can be displayed in the GUI front end. This component provides a graphical representation of the microworld including environment, agents, and task structure.

Metrics The microworld provides metrics for teams regarding their performance and other qualitative measures. These rules for measuring team performance are specified separately.

Dynamism Generator The microworld models a dynamic environment. Therefore, the state of the environment, in particular the board, is subject to stochastic changes. These changes are generated based on a model of dynamism supplied by the user. In addition, the microworld allows the simulation engine to control the generation of these changes by firing events.

Concurrent Simulation of Multiple Teams

In order to simulate multiple teams concurrently, the simulator needs to conduct the exact same experiments on different MAS models. Each MAS model might use a different interaction model or the same interaction model with different parameters. Constructing the same experiments for each MAS model is necessary in order to ensure the fairness of the comparisons and validity of the studies. It also helps in getting early feedback from experiments by allowing the user to access the simulation results from each simulation step for all MAS models or observing the behavior of different MAS models on the same phenomena at the same time. The dynamic environment requires special support from the MAS models in order to replicate the same experiment.

The dynamic characteristic of the environment brings a challenge in replicating experiments. While the initial conditions of each MAS model's environment should be the same, the dynamic patterns of the environment over time should also be exactly reflected in each MAS model's copy of environment. There are two types of sources of the environment dynamics: the ones that are beyond the control of agents (external) and the ones that are direct or indirect result of agents actions (internal). Based on this distinction, an architecture is designed to support constructing the same dynamic patterns in each MAS model's environment as follows.

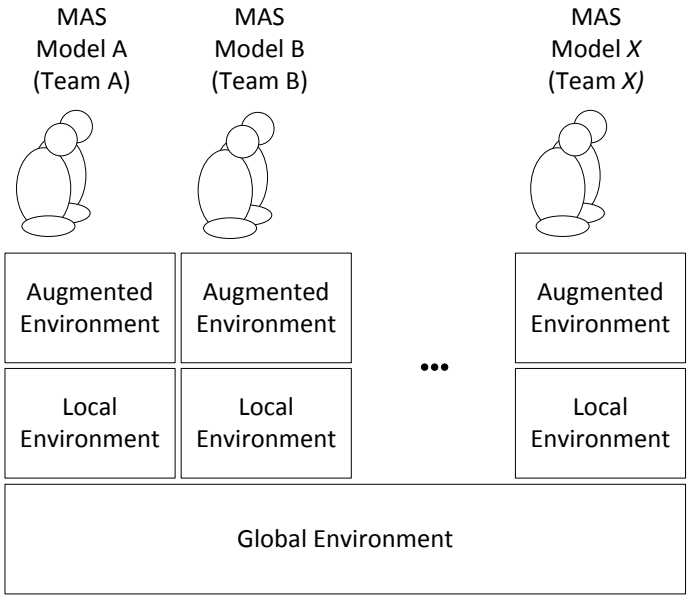

Figure 4.15: The board architecture for parallel MAS models, each representing a different team. The global environment is perceivable by agents from all MAS models. Each MAS model has a distinct layer which maintains the MAS specific information. Agents have access to both layers through an augmented board which is a virtual layer combining two other layers.

Conceptually, the environment and its features are broken into three layers. A *global* environment layer, a *local* environment layer, and an *augmented* environment layer (Figure 4.15). The *global* environment consists of all the common features of the environment which can be shared among the MAS models. This includes any external events that might occur in the environment that are beyond the control of agents and independent of their actions. No matter how agents behave in the environment, these events have their own effects. As a result, each MAS will face the same pattern of events regardless of the interaction model they employ.

Each MAS model is given a *local* environment which is specific to that MAS. This layer of environment handles all the features that belong to the MAS model. For instance, the outcomes of agents' actions are handled in this layer of the environment.

However, agents do not have access to the *local* environments that belong to other MAS models. By combining the first two layers, a third layer is created that is called the *augmented environment* layer. The environment represented by this layer contains the information from both the *global* environment and the MAS model's *local* environment, and it is what the agents perceive. While the *augmented* environment contains the perceptions propagated from both other layers and is used for agents' perceptions, the outcomes of agents' actions are only reflected on their own MAS model's *local* environment. In this way, each MAS model has an identical yet isolated environment which is subject to the same external change patterns. An example of applying this architecture to a grid-based environment in which agents perform implicit communication through environment is illustrated in Figure 4.16.

The concurrent simulation of multiple teams is supported by the simulator's MAS model architecture as follows. For each team, there is a separate MAS model providing team, agents, and communication module representations. In addition, each MAS model includes a microworld. However, in order to replicate the same experiments, and the same environments, for each team, the microworld of each MAS model need to share parts of their state. This belongs to the state of the board in which agents would be situated. Thus, the board module of the microworld is broken into two layers: a global board in which the common properties of the board that are going to be shared among multiple MAS models are represented; and a local copy of the board for each MAS model, excluding the common properties and only maintaining team-specific properties (Figure 4.17).

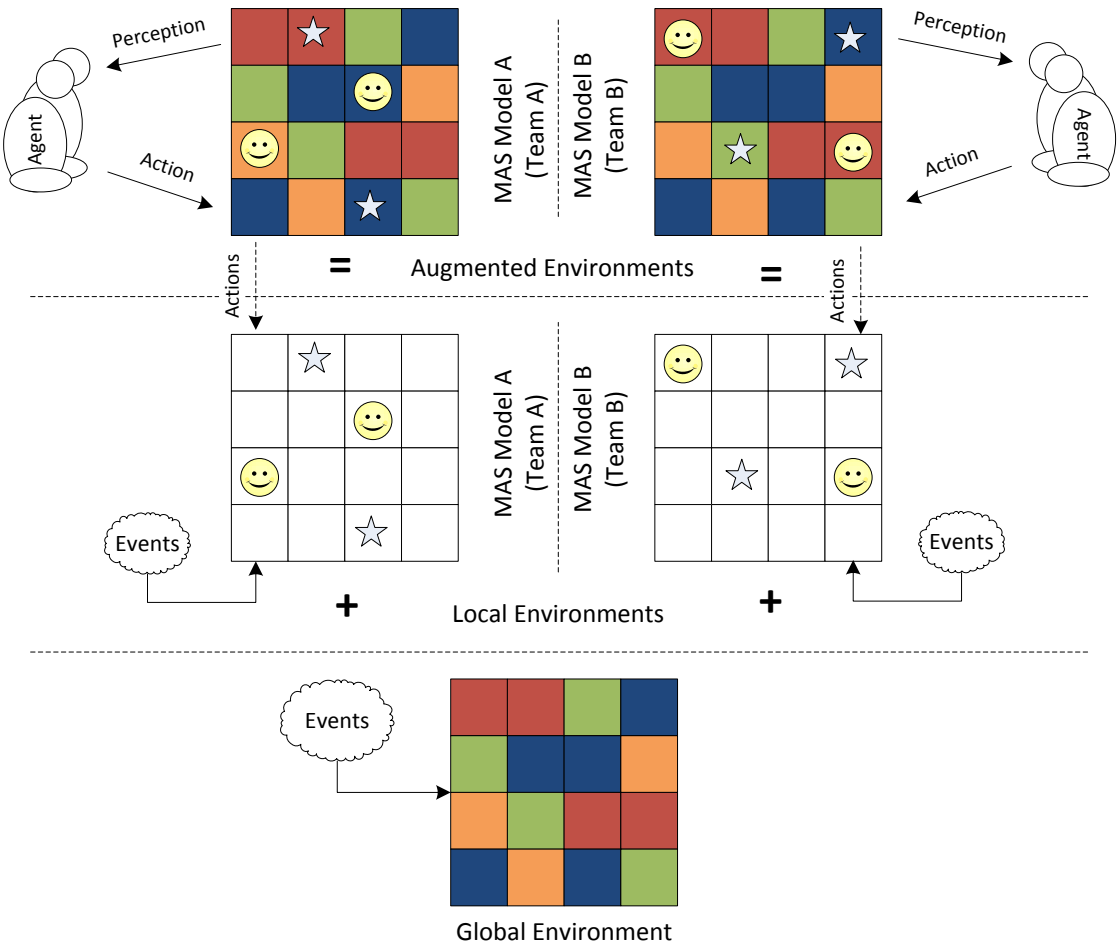

Figure 4.16: Parallel MAS models using a grid-based environment. The global board consists of colored cells. An event in such a board could be a change in the cell colors. Each local board maintains the position of agents and other results of agents' actions. In general, any property of the environment that is the result of agent's actions, or can be affected by agents' actions is represented in the local environment and any other properties that are not affected by agents are represented in the global environment. The augmented environment provides a unified view of both layers.

4.2.5 Distributed Simulation

The proposed simulator uses a distributed simulation architecture (as shown in Figure 4.18). In this architecture, the distribution is based on executing different runs of the same experiment on different nodes of the network. In other words, as for each

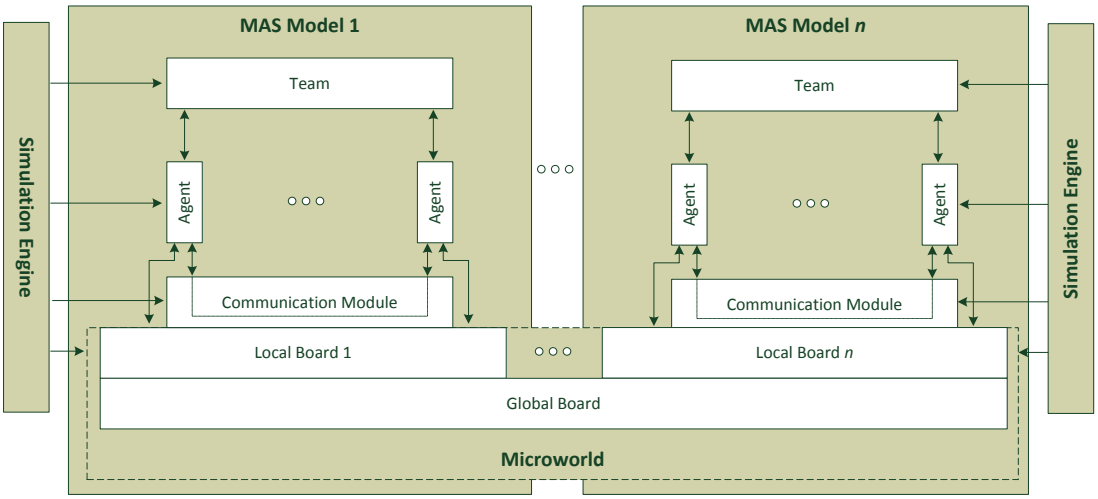

Figure 4.17: Multiple MAS models and the shared microworld

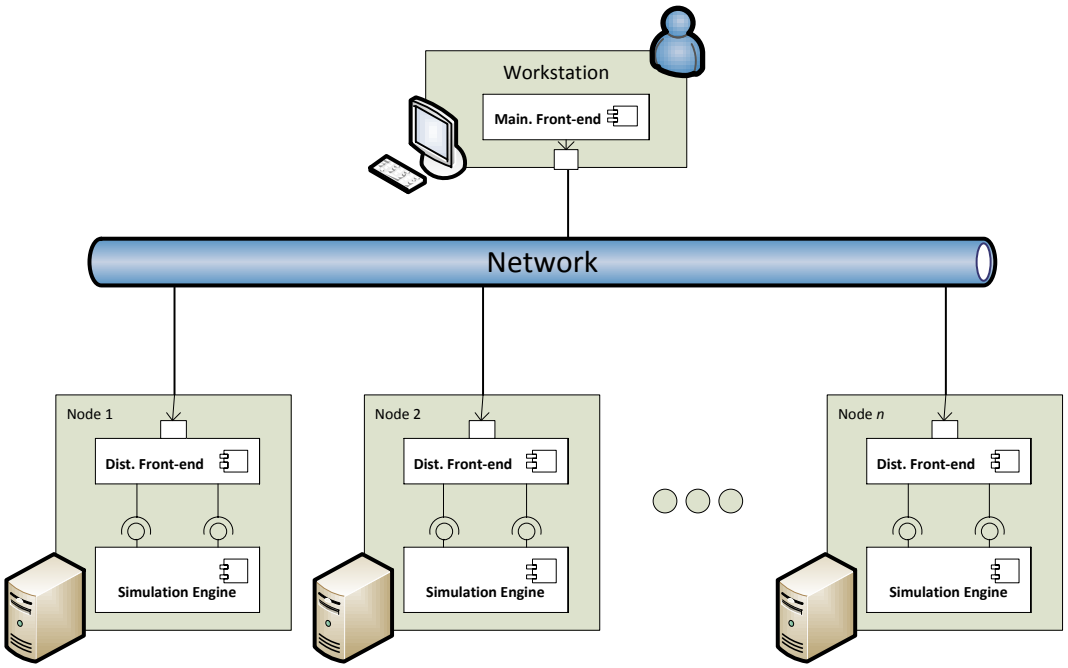

Figure 4.18: The distributed simulation architecture

experiment there are a large number of runs needed, these runs are distributed to different instances of the simulator on different nodes of a network. The rationale

behind this distribution scheme is that these runs are independent from one another. Therefore, there is no need for synchronization and communication between instances of the simulator. As a result, this scheme provides a simple and effective distribution architecture for the simulator.

4.3 The System Behavior

In this section, the behavioral aspects of different components of the system are explained. However, this only includes the components whose behavioral aspects play important roles in the simulation process.

4.3.1 The Simulation Engine

The simulation engine design is based on simulating time as discrete values. Accordingly, the simulation model is represented as a discrete system whose state transitions occur at discrete time steps. The simulation model supported by this framework is a cooperative game which can be played in discrete steps by different members of the same agent team. The course of the game, from its beginning to its end, is performed in a *match*. The game starts by assigning a *task* to the team and is played in multiple *rounds*. In every *round*, each agent can perform one action on the environment. These *rounds* are repeated until, based on the rules of the game, the game is considered as finished. The framework also supports modeling a long-term memory for the agents involved in the game through allowing each *run* to consist of multiple *matches* and allowing the agents to retain some state information from completed *matches* within the same run. The behavior of the simulation engine is explained below.

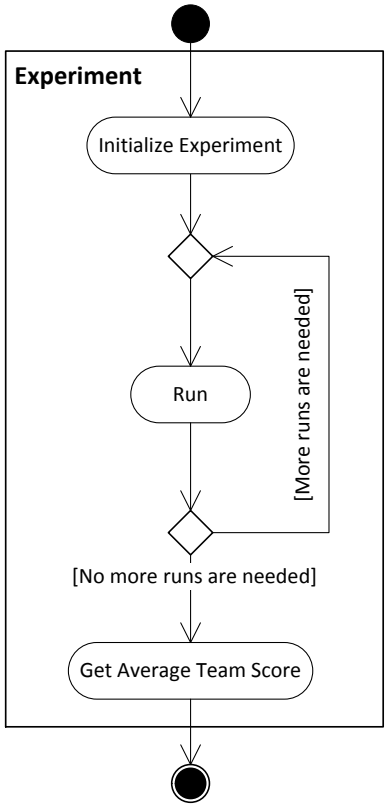

Figure 4.19: The mechanics of the *experiment*

At the beginning of an *experiment*, the Experiment Handler initializes the simulation engine and the model by using the configuration specified in the experiment setup. Next, based on the specified number of runs for the experiment, it activates the run handler. At the end of an experiment, it calculates the average score of the team over all runs (Figure 4.19).

After its invocation by Experiment Handler, the Run Handler first initializes the microworld and the agents for a new experiment run. Next, it activates the Match Handler for executing a new game. After the game is over, it checks whether more matches are required. If yes, it will reactivate the Match Handler. Otherwise, it will calculate the average team score for all games the team played in the run (Figure 4.20).

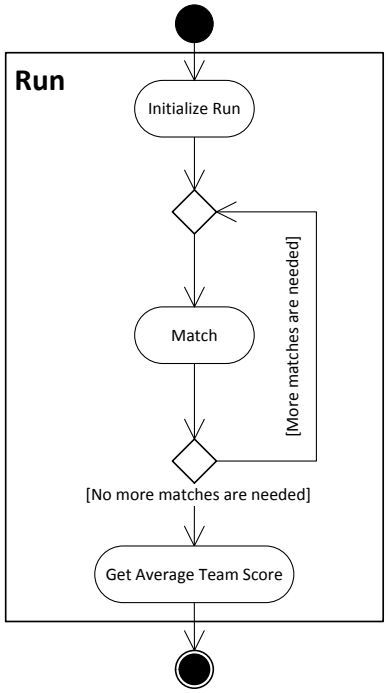

Figure 4.20: The mechanics of the *run*

The Match Handler starts each *match* by initializing the parameters of different parts of the model for a *match*. Next, it starts executing each round by first firing the global events of the environment (defined in the microworld component). At the end of each round, the Match Handler checks for each of the concurrently executing teams whether, based on the rules of the game, the game is over or not. If the game is not over yet, it will execute another round of the game, otherwise it will terminate the game and calculate the team's score for that game through the microworld (Figure 4.21). The invocation of the Match Handler terminates when the matches of all teams are completed.

The game is played in synchronous cycles called *rounds*. As demonstrated in Figure 4.22, each cycle starts with agents perceiving their environment. The percepts are generated by the microworld per each agent and are passed to them individually.

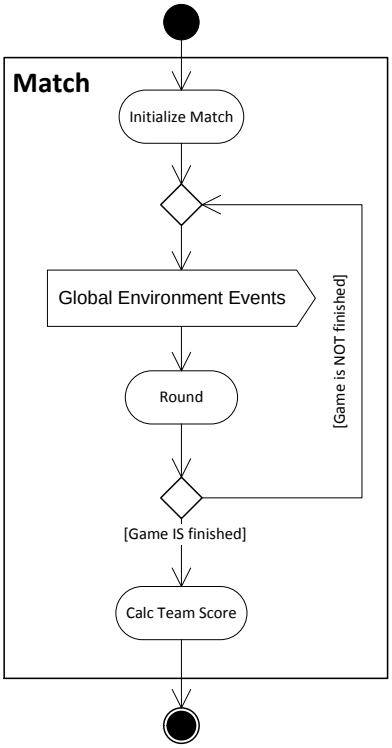

Figure 4.21: The mechanics of the *match*

Following the perception, agents enter into a number of communication cycles where they can interact with each other through message passing. A communication cycle starts by a *send* phase and ends with a *receive* phase. The alternating *send-receive* cycles repeat until agents decide, as dictated by the protocol they are employing, that there is no more communication required. Finally, each *round* ends with agents performing actions (if any). Each stage occurs for all the agents within the same team concurrently. The synchronization of agents' activities is represented in Figure 4.23.

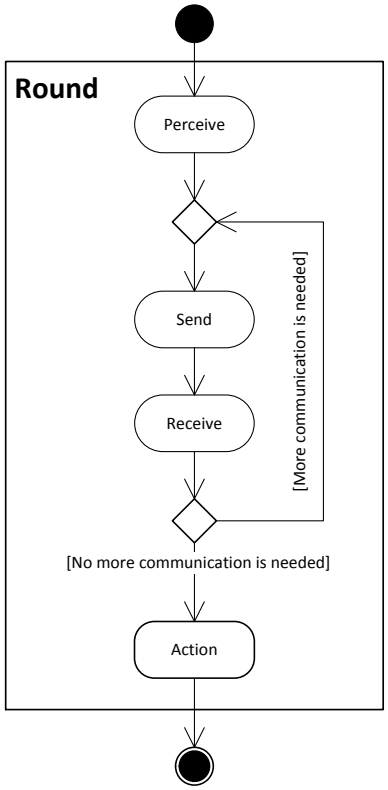

Figure 4.22: The mechanics of the *round* with message passing cycles

4.3.2 The Front End

GUI

The Graphical User Interface (GUI) front end allows an intuitive access to the simulator and provides various unique features that facilitate the study of interaction models through the simulator. The main window of the GUI is represented in Figure 4.24. In response to the functional requirements of the system, the fundamental goals of designing the GUI are summarized in providing the following functionalities:

- Experiments setup

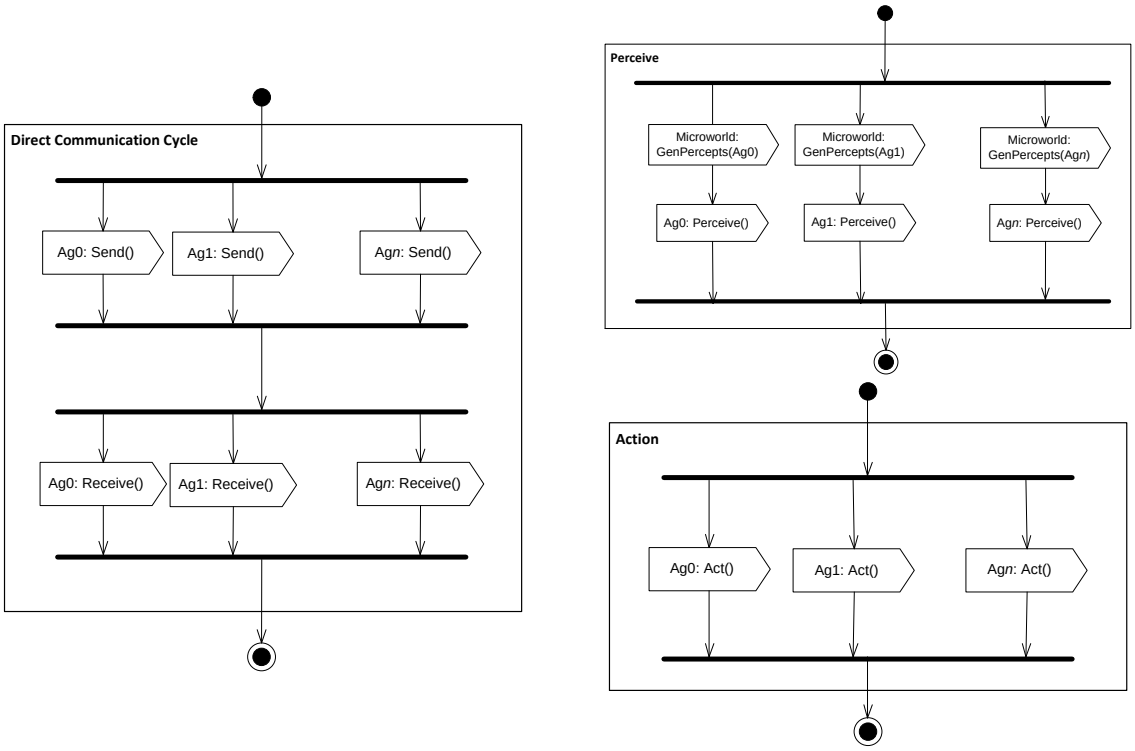

Figure 4.23: How agents are handled in different stages of each *round*

- Experiments control
- Online simulation
- Dynamic experimentation
- Visualization
- Statistical analysis

The GUI consists of toolboxes that support different aspects of the simulation, namely: the control box, the console, the visualization box, and the experiment setup box. The functionality of each of these components is explained below.

The control toolbox (Figure 4.25) allows the user to control the simulation. It provides basic start/stop functionalities as well as settings for running the simulation

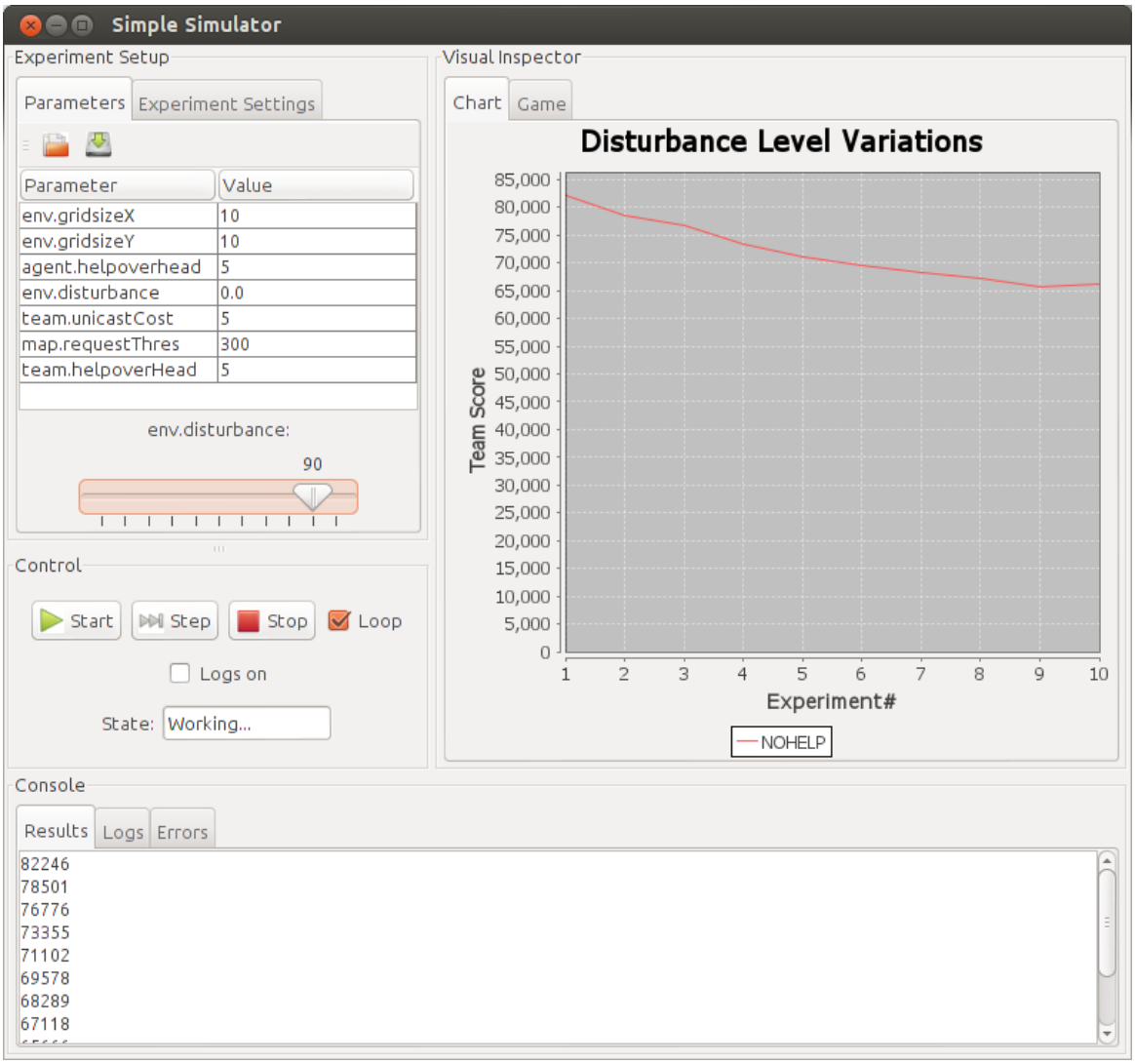

Figure 4.24: The main window of the simulator's GUI

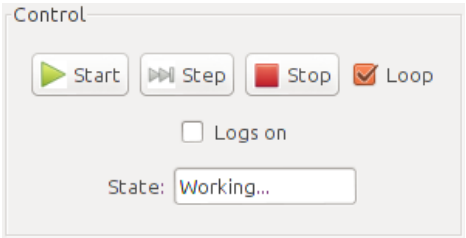

Figure 4.25: The control toolbox

in an infinite loop.

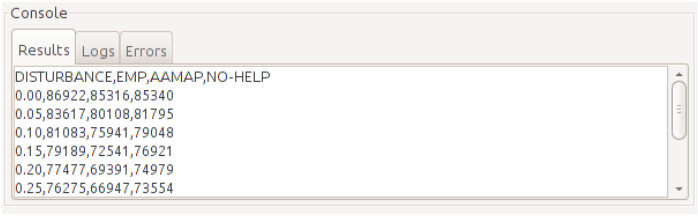

Figure 4.26: The console toolbox

The console toolbox (Figure 4.26) provides the means to access the raw results, agents' logs, and the errors.

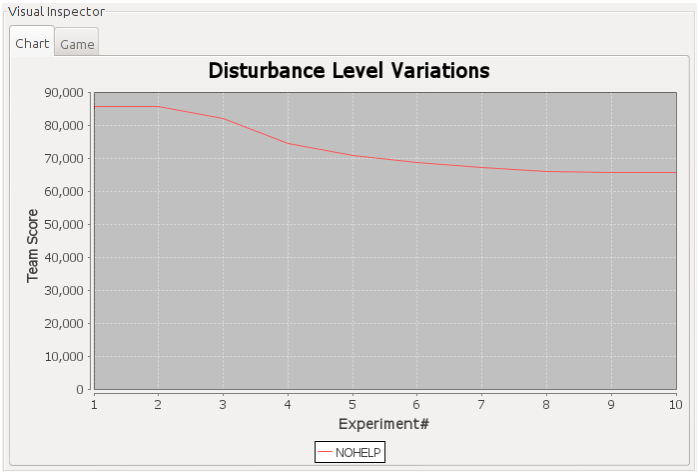

Figure 4.27: A chart created in the visualization toolbox

The visualization toolbox provides the support for drawing charts, dynamically, in user-defined ways[1]. Examples of different charts are demonstrated in Figure 4.27.

The experiment setup toolbox allows the user to set up experiments and modify any parameter that is defined within the system. The first tab of the toolbox has the controls for setting up experiments. In the second tab, the user can adjust any parameter that is defined in the system (Figure 4.28). This list is interconnected to the simulation engine's parameter interface. This makes any changes in the list to be immediately applied to the (running) system. This unique feature allows dynamically

[1]The simulator uses the charting library JFreeChart (http://jfreechart.org) for drawing charts on the GUI.

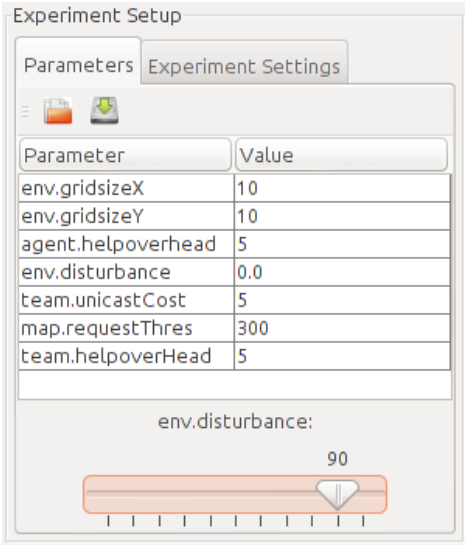

Figure 4.28: The parameter editor in the experiment setup toolbox

studying the effect of individual parameters within an experiment that is already running.

CLI

In the Command-Line Interface (CLI), the input parameters can be read from files or from the standard input. The results also can be written into a file or just printed on the standard output. This version of the front end can be easily used to support functionalities such as distributed simulation or interoperability with other programs.

4.3.3 The MAS Models

Agent

The agent architecture presented in this framework enables the basic behavior of agents such as perceiving the environment, reasoning, communicating, and performing actions. However, the detailed mechanisms of the agent's behavior are left unspecified so that they can be defined based on specific requirements of a research problem. In particular, the agents within the current version of the framework are expected to work as a team in such a way that they try to maximize their team benefit rather than their own benefit. They are designed to support interaction protocols for performing helpful behavior and are capable of modeling different types of reasoning (e.g. rational, empathic, etc).

Concurrent Simulation of Multiple Teams

The concurrent teams simulation model easily fits into the general simulator architecture (Figure 4.29). At the beginning of every *match*, the events that affect the *global* environment are fired. These events cause the *global* environment to change in certain ways. Then, in a (pseudo) parallel setting, the control is passed at the same time to each MAS model, which starts by firing local events in its local environment followed by a new *round* of the game. These local events affect the *local* environment of each MAS model individually. Using this method, it is guaranteed that all MAS models face the same global events in their environment at each round while they are allowed to have their local events. It is worth mentioning that, in this architecture, from the agents' perspective there is only one environment and the layers are invisible to the

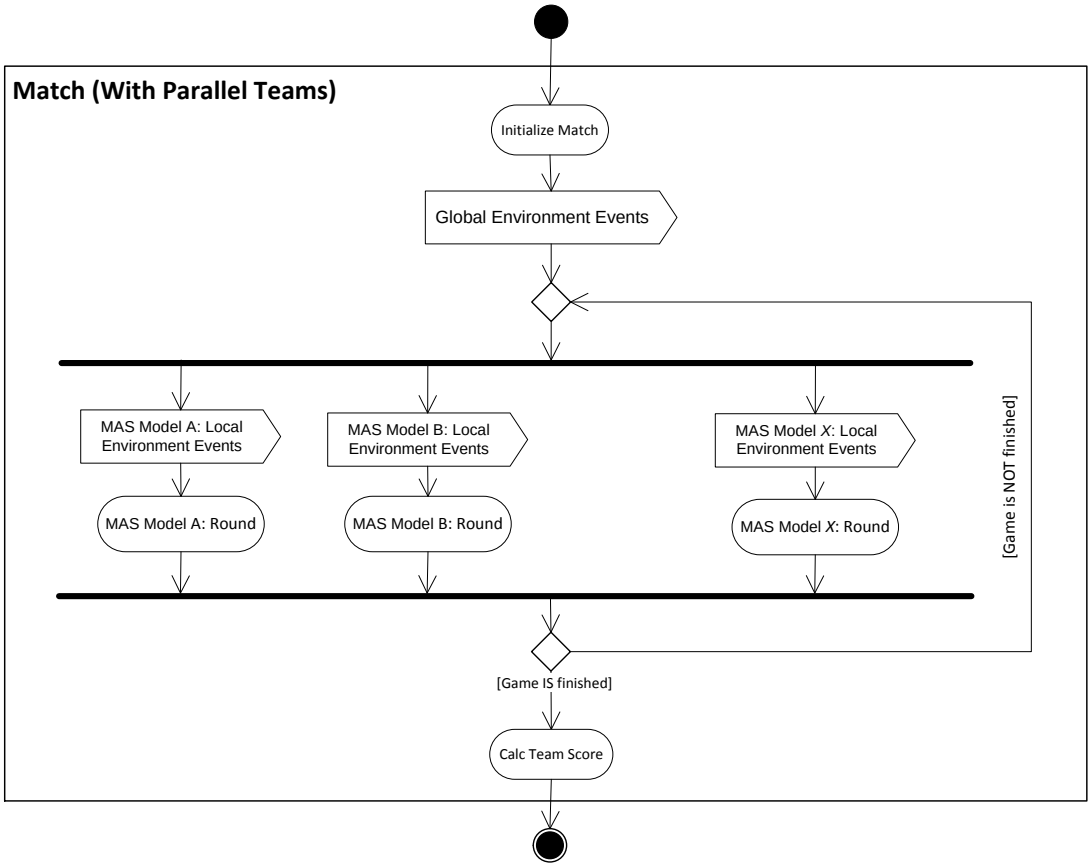

Figure 4.29: Handling multiple MAS models in parallel in each *match*.

agents.

4.3.4 Distributed Simulation

The simulator provides scalable simulation by using a distributable simulation architecture. Speed and valid results are the two main requirements of this simulator. However, precise simulation demands a great amount of computational power which reduces the speed of the simulator. In order to overcome this barrier and achieve both of the mentioned objectives, a distributed simulation mechanism is designed and

provided with the simulator.

The main factor that affects the speed of the simulation is the large number of runs that are required for more (statistically) significant simulations. As the size of the sample the simulator is generating is always less than the size of the population (all possible combinations of settings), there is a natural sampling error associated with each experiment result. This error can be reduced by increasing the sample size which, in our case, is the number of runs. For instance, one might need to repeat the same experiment for more than 25000 times in order to get a reasonably small error. The impact of this factor can be reduced by breaking down the *runs* that are being repeated, and run them in parallel. At the end, by aggregating the results, one can achieve both the performance and the precision required.

The simulator is designed to support distribution of the simulation load on a Linux cluster in order to achieve scalability. Based on the nature of these simulations and factors that affect the performance, the main ideas behind the distributed simulation design are as follows. As the runs of the same experiment do not depend on each other, they can happen in parallel. While the simulator runs as a single process to simulate each instance, each of these instances can be run in parallel with more than one simulator process. This process can be replicated on a Linux cluster as follows. A copy of the simulator which is setup to be used in batch mode would be available on each node of the Linux cluster. The nodes of the cluster are able to communicate with each other, for exchanging both data and commands, using the Secure Shell (SSH) protocol. The SSH allows sending remote commands and making basic remote I/O between different network systems. A main front end is executed on the master node, where the user can set up, run, and analyze the experiments. The rest of the system is invisible to the user. The whole process is done in three steps (Figure 4.30)

as follows:

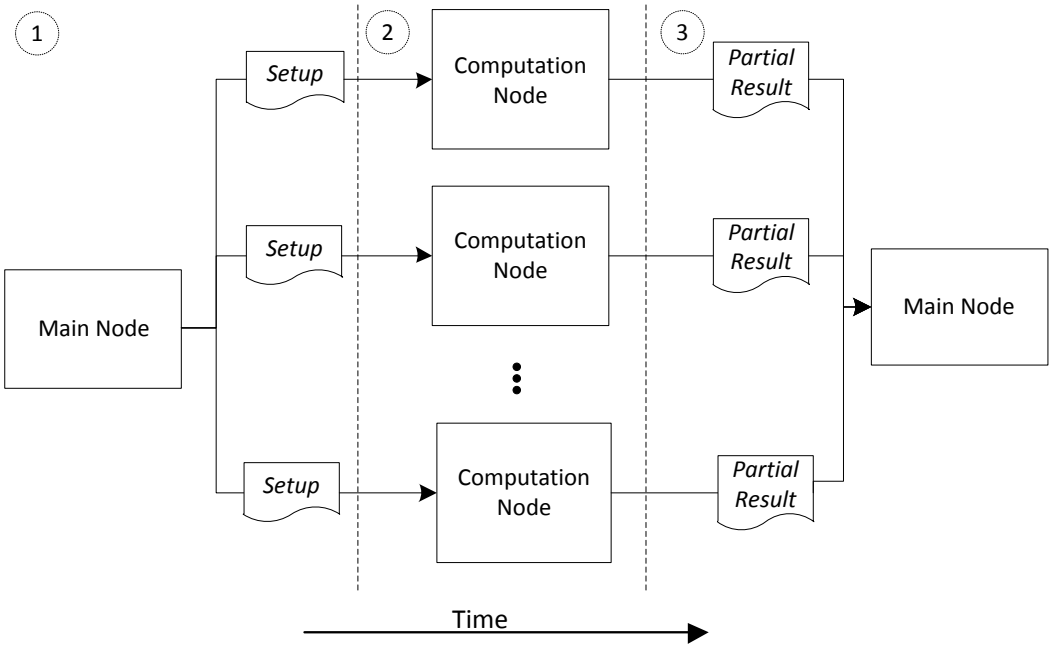

Figure 4.30: The three steps of distributed simulation.

1.1. The master node, which only runs a front end, distributes the experiment setup file to each node within the cluster. In the setup, each node is set to run for a fraction of the total required number of runs. If N is the number of nodes and r is the total number of required runs, then each node is set to run $\frac{r}{N}$ times.

1.2. The master node remotely executes the simulator instances on each node of the cluster and waits for them to finish. It also creates a Unix pipe for each connection which can be used to retrieve information from each node.

2. Each node starts executing the simulation.

3.1. Upon generating a new partial result each node writes the results on its standard output which is redirected to the master node through the pipe.

3.2. The front end in the master node gathers all the results, aggregates them, and displays them to the user as if they were all generated in the same machine.

The overall characteristics of this approach is summarized in the following:

- *reliability*: A failure in any node has a limited effect upon the simulation process as it just misses a group of runs and thus just reduces the precision.
- *flexibility*: This distributed architecture can be set up to run on any Linux cluster with potentially heterogeneous processor hardware.
- *cost-effectiveness*: As this design requires no synchronization and communication between nodes, it could be used on any high speed computer network and does not require the more expensive, low-latency network infrastructures.
- *simplicity*: The internal architecture of the simulator does not need to be changed in order to be used in this model.

4.4 Instantiation of the Generic Simulator

Specialized simulators for studying specific classes of AIPs can be instantiated from the generic simulator architecture and MAS model, presented in this chapter, by incorporating the models of the desired AIPs as well as a customized MAS model.

In the next chapter, the approaches to the modeling of AIPs and MAS are explained. In particular, a common microworld model for the studies of helpful behavior in agent teamwork that can be incorporated in the framework is introduced. In order

to present the framework in more concrete terms, as well as to evaluate its usefulness, we describe two separate simulators that have been created using the framework for our studies of two classes of AIPs for helpful behavior in teamwork. Chapter 5 presents the MAS and AIP models that are used to build those simulators and Chapter 6 the experiments that has been conducted using them.

The examples presented in the next two chapters are the test cases for evaluating the proposed framework and its flexibility and usability for instantiating specialized simulators for different classes of AIPs and conducting experiments with them.

Chapter 5

Modeling for Simulation

This chapter first presents the framework's approach for modeling agent interaction protocols in Section 5.1. Next, Section 5.2 presents the modeling of a multiagent system in the framework by elaborating a common world model that is built to support our research on helpful behavior. Sections 5.3 and 5.4 present the specializations of this world model that lead to two separate simulators, each designed for studying a different class of helpful behavior AIPs.

5.1 Modeling of Interaction Protocols

In this section, I explain how one can model agent interaction protocols within the framework.

5.1.1 Representing Protocols

Following one of the common techniques in protocol specification (Section 2.4), we represent agent interaction protocols by interacting finite state machines (FSM). Each FSM represents the behavior of an agent, and their interactions are represented by messages passed through a communication network. In our model this network is synchronous in the sense that the agents alternate between their sending and receiving

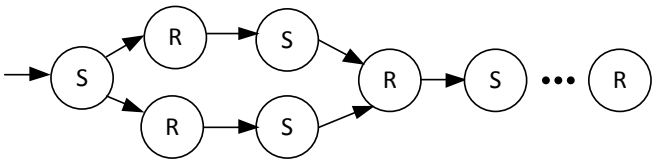

Figure 5.1: The alternating *sending* (S) and *receiving* (R) states of an AIP's FSM

phases in lockstep. As protocol deliberations occur between those phases, we prefer to present them through separate states. Accordingly, each state of an FSM is labeled as either a *sending* (S) or *receiving* (R) state. An agent can only send in an S state, and receive in an R state. A state transition occurs in each synchronous cycle, and it always leads to a state with an opposite label (Figure 5.1).

Each AIP, in order to be defined in the framework, should be modeled following the described FSM. In other words, the protocol should be defined as a state machine with alternating send and receive states guiding the agent through its interactions. At each send state, agents can send messages to others based on the protocol they employ. Accordingly, agents receive and process their incoming messages in receive states. In this model, the state transition is determined by the protocol, based on agent's beliefs, incoming and outgoing messages, and current state.

5.1.2 Multi-Protocol Interaction Models

Some MAS architectures allow dynamic selection of interaction protocol. In such architectures, an agent can select between alternative protocols dynamically. In these situations, the agent needs to be able to model multiple protocols and activate one dynamically. The FSM model described above supports such models by combining different FSMs. As represented in Figure 5.2, the state machines of individual AIPs

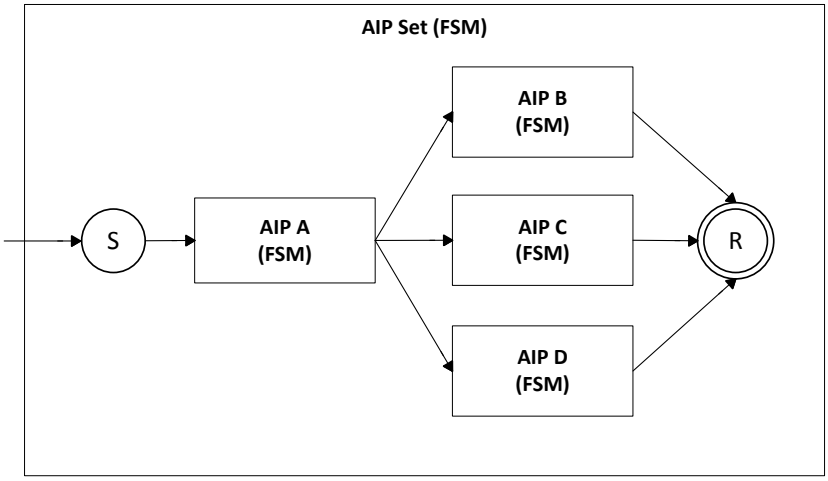

Figure 5.2: The augmented FSM that combines several AIPs

can be combined based on common practices for combining state machines to create a new FSM that models a set of AIPs. In this case, the agents initially perform a negotiation protocol A in order to agree on which of the protocols B, C, and D they wish to use.

5.2 A Model of a Multiagent System

This multiagent system (MAS) model is designed as a game along the lines of the Colored Trails [Gal et al., 2010] game. However, it is designed and developed independently to provide a specific microworld for studying helpful behavior in a teamwork context. The model represents the environment, the agents in a team context, and the task and subtask specifications. Each entity is described in the following.

5.2.1 Environment

The environment is a rectangular board that consists of colored squares (Figure 5.3) on which the players of the game (software agents) are situated. The colors of the squares are chosen from a fixed set of colors $C_1, ..., C_m, m > 1$. The environment is

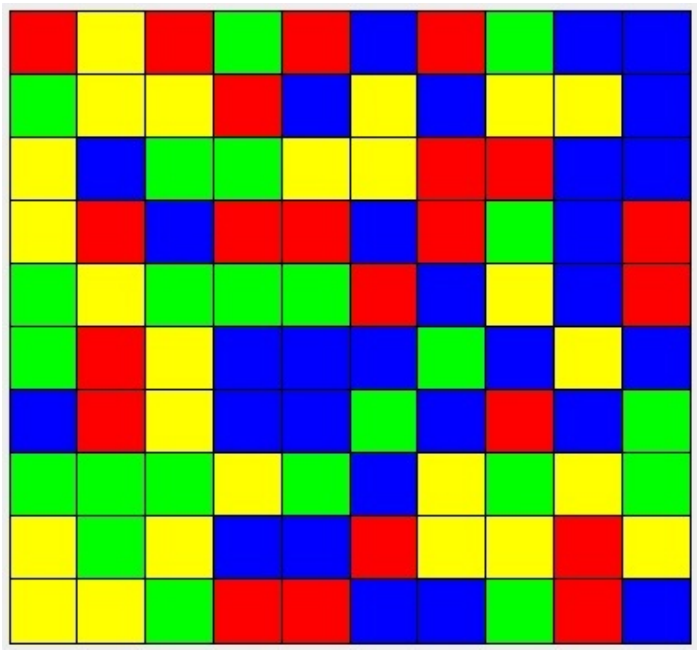

Figure 5.3: The board of the game

dynamic and evolves in each round of the game. The evolution of the environment is

represented by the changes in the colors. The color of any given square can change at any time according to a prescribed stochastic model.

To model different phenomena in the environment, it is often required to model different types of disturbances. Therefore, the user can design different disturbance models and plug them into the microworld. Each disturbance model can affect the environment, and consequently the behavior of agents, in a different way. The disturbance in the environment occurs at the beginning of each round of the game when the simulation engine activates the global events of the microworld.

5.2.2 Task

At the beginning of each game, a task is assigned to a team. The main objective of the team is to achieve the given task. The task can be broken into subtasks in such a way that achieving the task requires achieving all the subtasks. Each subtask is represented by the location of a square as the starting point and by the location of another square as the goal. A subtask is then considered to be completed if and only if all the squares of one of the paths connecting its starting and goal locations have been marked by one or more agents. In other words, each subtask is done if and only if a set of certain actions have been performed by agents. However, the progress towards completion of the task can be measured by a scoring system and, if the completion cannot be achieved it may still be important to reach a high team score.

Actions are represented by agents making a move from one square to another. These moves are on a path from the initial location of a subtask to its goal location. Subtasks are initially assigned to agents at the beginning of the game and can be dynamically reassigned later based on any user-defined (re)assignment algorithm.

5.2.3 Resources

At the beginning of each game, the team is given an amount of resource points. The team should consume some of its resource points in order to perform each action.

5.2.4 Agent Team

Each team consists of agents $A_1, ..., A_n, n > 1$. The agents are situated in the environment in the sense that each agent is located on a certain square of the board. It is allowed that multiple agents have the same position on the board.

Agents in the same team are allowed to help each other. Currently two forms of help are modeled: *action* help and *resource* help. In action help agents can perform actions on behalf of their teammates. An instance of such help, termed as *help act*, involves an agent A_i that wants to move to a square of color C_k, and a helper agent, A_j that executes that action on behalf of A_i. The result is that A_i moves with its budget r_i unchanged, and A_j's budget r_j is reduced by $c_{jk} + h$, where c_{jk} is the cost of an action on a cell with color C_k for agent A_j, and $h \in N$ is a team-wide constant called the *help act overhead*. Note that if A_j has a higher capability (i.e., lower cost) for actions of type C_k than A_i, the help act saves resource points to the team, provided that the difference in costs is higher than the help act overhead. The help act overhead quantifies the extra work involved when an agent performs an action for another agent rather than for itself.

In resource help, agents can exchange resource points. For example, suppose agent A_i has a higher capability of doing an action of type C_k than agent A_j, but A_i does not have enough resource points in order to perform C_k. In this situation, A_j can

send the necessary resource points to A_i so that A_i can finish its designated action. This will also help the team save more resource points. It is also possible for A_i to receive resource points from multiple sources.

5.3 Example: The Mutual Assistance Protocol (MAP)

Mutual Assistance Protocol (MAP) [Nalbandyan, 2011] is a protocol for incorporating helpful behavior in agent teamwork. In MAP, an agent can help another agent who is requesting help if they can jointly agree that the outcome of the help is in the interest of their team. The decision about performing help act in thus based on a rational bilateral distributed agreement between the two agents. The agent use their individual beliefs and interact through a bidding sequence in order to make their decisions. A complete description of MAP can be found in [Nalbandyan, 2011] and [Polajnar et al., 2012].

In this section, the approach for using the proposed framework in modeling a multiagent system and building a simulator for studying and developing a version of MAP called Action MAP is presented. First, the microworld that was used for experiments is described. Second, the approach for modeling Action MAP and other help strategies that have been developed for the comparison studies of MAP is explained.

5.3.1 The Microworld Configuration

The microworld that is used for studying and validating MAP is described in [Polajnar et al., 2012]. This microworld is carefully tailored to model the essential aspects

of a teamwork environment needed for experimenting with helpful behavior. This microworld is the basis for our experimentation. Here, I quote the description directly from the paper:

> The players are software agents $A_1, \ldots, A_n, n > 1$, situated on a rectangular board divided into colored squares. The game proceeds in synchronous *rounds*. Each agent can move to a neighboring square in each round. Each move represents the execution of an action. The types of actions $\alpha_1, \ldots, \alpha_m$ are represented by the available colors, and their costs to individual agents by the $n \times m$ matrix *cost* of positive integer values.

The task structure and the planning capabilities of agents are modeled in the microworld component as follows:

> At the start of the game, each agent A_i is assigned its initial location on the board, a unique goal with a specified location and amount g_i of *reward points*, and a budget $r_i = d_i a$ of *resource points*, where d_i is the shortest distance (i.e., number of squares) from the agent's initial location to its goal, and a a positive integer constant. Whenever A_i moves to a field of color α_j, it pays $cost_{ij}$ from its resource budget; if the budget is insufficient, the agent is blocked. Each agent chooses its own path to the goal, which represents the choice of its own local plan. The paths can intersect; it is legal for multiple agents to be on the same square at the same time. The game ends when no agent can make a move (because it has either reached the goal or lacks the resources).

The game's objective is to maximize the team's score. The team score is used as the metric for evaluating different MAS models for these experiments and is calculated based on the scoring rules defined as follows:

> All agents remain in the game until the end, when their *individual scores* are calculated as follows: if A_i has reached the goal, its score is the goal achievement reward g_i plus any remaining resource points (as a savings bonus); if A_i has failed to reach the goal, its score is $d_i a$, where d_i is the number of moves A_i has completed, and a is a positive integer constant representing the reward for each move. The *team score* is the sum of all individual scores.

The microworld is designed to model dynamics of the environment that can affect agents' actions. The changes in the environment are modeled by a disturbance model:

> As a representation of environment dynamics, the color of any square can be replaced, after each round, by a uniformly random choice from the color set. The change occurs with a fixed probability D, called the *level of disturbance*.

Most MAP experiments conducted so far use the following disturbance model. For any given square, in each round, the color revision occurs with probability D; the new color is a uniformly random choice among all available colors (including the current one). Note that, since the new color can be the same as the old one, the probability that a given square actually changes color in a given round is $\bar{D} = (1 - 1/m)D$.

The disturbance model described above is defined as a global environment event. All the teams within the experiment will face such changes in their environments

in the exact same manner. At the beginning of each *round*, the simulation engine triggers this global environment event and causes the colors of the board to change based on the *level of disturbance* introduced above. Level of disturbance is defined as a parameter and can be set in the experiment setup. It is possible to use other models of disturbance for other experimentation with MAP.

In simulations of MAP, the helpful act is modeled essentially as described in 5.2.4:

> The requester A_i faces a move to a square of color α_k , charged at $cost_{ik}$; if A_j agrees to help, A_i moves at no cost to itself, with the $cost_{jk}$ charged to A_j . Protocol interactions involve explicit computation and communication costs, and the help act has a fixed overhead cost. While the specific decision criteria and protocols for help transactions may vary, the general intent of such transactions is to advance the performance of the team as represented by the team score.

Finally, the rules of agents' perception in the environment that are used for generating percepts for each agent are defined as follows:

> Each agent sees the entire board. The agent knows its own cost vector. It knows the range of all color costs, and thus its own level of expertise for each action type, relative to what may exist in the team. The agent has probabilistic beliefs about the cost vectors of other agents. The quality of its probabilistic beliefs about teammates' abilities is modeled by a team-wide constant probability p_M , called the *mutual awareness*. For a given agent A_j , $j \neq i$, and color α_k, agent A_i believes, with probability p_M , that $cost_{jk}$ has the value that it actually has, and with the probability $1 - p_M$ that all possible color cost values are equally likely. This

uncertainty reflects the idea that, in many realistic teamwork situations, the performance of a teammate with a different expert profile on a specific problem type may be hard to estimate reliably.

5.3.2 Modeling Protocols

Action MAP

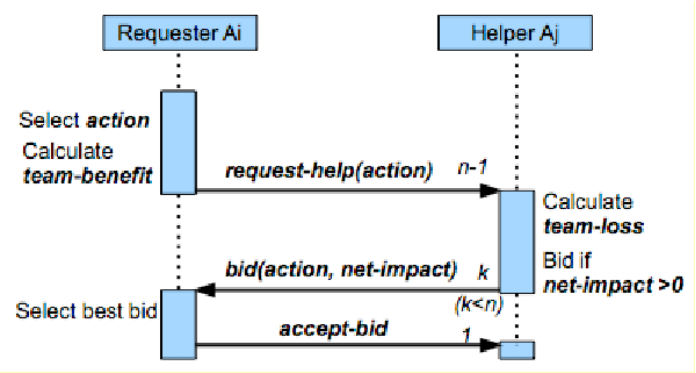

Figure 5.4: The Action MAP. Reprinted from: [Polajnar et al., 2012]

As a general procedure in the framework, in order to model a protocol (Action MAP in this example), after it has been generally conceived initially (e.g. by the diagram in Figure 5.4), it needs to be represented in terms of interacting finite state machines (FSM), based on the framework's AIP modeling scheme described in Section 5.1, so that it can be employed by the agents in the framework. The result of this representation is displayed in Figure 5.5. This FSM follows alternating send and receive states, which is required by the framework's FSM executer. Such FSM is then supported by a full definition of the agent's behavior in each state and the transition rules. An example of the definitions for some of the states of the Action MAP's FSM is presented in Figure 5.6. The FSM definition is then translated into code that is executable by the framework.

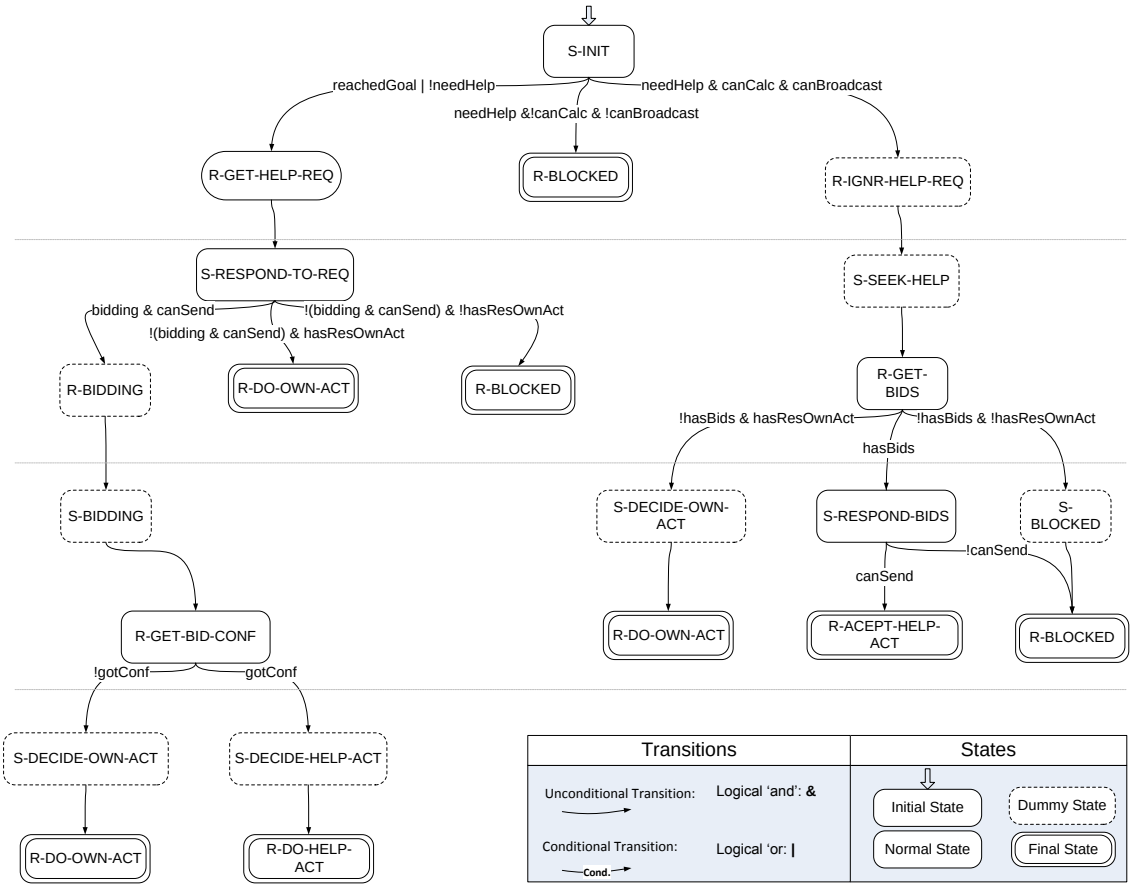

Figure 5.5: The FSM model of Action MAP. Note that some states are shown more than once for the purpose of better presentation.

Unilateral Protocols

In order to study and evaluate Action MAP, two other protocols with different help strategies have been modeled: The Unilateral Requester-Initiated Protocol (URIP) and the Unilateral Helper-Initiated Protocol (UHIP). These protocols represent help strategies that use unilateral decision making in contrast to Action MAP's bilateral approach. The framework easily accommodates the concepts required for incorporating these help strategies. First, the protocols are modeled in the simulator without further modifications in its generic AIP model. Second, the framework supports incorporating the concept of *probabilistic beliefs* that is introduced in this MAS model

State *R-GET-BIDS*:

Receive $q(q \geq 0)$ bids $(Bid_1, \ldots, Bid_q)$

If $q == 0$ (no bids):
- If resources suffice for next action:
  - Set *State* to *S-DECIDE-OWN-ACTION*;
  - Return *waitingToCommit*;
- Else:
  - Set *State* to *S-BLOCKED*;
  - Return *waitingToCommit*;

Else:
- Select the bid with highest Net Team Benefit Δ_{ij};
- Set *State* to *R-RESPOND-TO-BIDS*;
- Return *waitingToCommit*;

State *S-BIDDING* [dummy]:

Set *State* to *R-GET-BID-CONFIRMED*;
Return *awaitingResponse*;

State *R-GET-BID-CONFIRMED*:

If receives confirmation message for bidding
- Set *State* to *S-DECIDE-HELP-ACTIONS*;
- Return *waitingToCommit*;

Else:
- Set *State* to *S-DECIDE-OWN-ACTIONS*;
- Return *waitingToCommit*;

Figure 5.6: Example: Part of state definitions for the Action MAP FSM.

which is required for modeling unilateral help strategies.

5.4 Example: The Empathic Help Model

A model of empathy as a mechanism for triggering help in agent teamwork is introduced in [Polajnar et al., 2011] and [Dalvandi, 2012]. A simulator is built using the

proposed framework in order to investigate whether and how incorporating this model into agent teamwork can improve the team's performance.

In this section, the special microworld configuration that is built for this simulator is explained. The modeling of the protocol is similar as in Section 5.3.

5.4.1 The Microworld Configuration

The simulator built for studies of the empathic help model uses almost the same microworld configuration as the MAP simulator described in Section 5.3. The main difference between the microworlds of the two simulators is explained below.

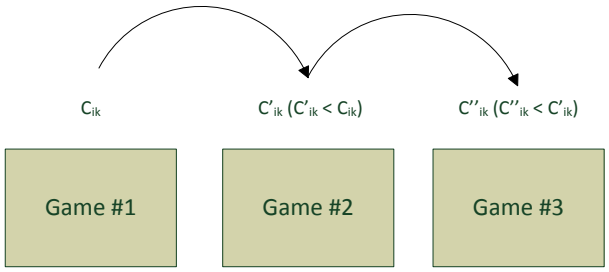

Figure 5.7: Modeling agent's experience carried over multiple games. The cost of performing an action on the cell with color C_k by agent A_i is denoted as c_{ik}. As the figure demonstrates, c_{ik} decreases as the agent plays more games.

The empathic help model requires agents to acquire experience from the actions that they perform. To accommodate this concept, a notion of experience is modeled in this simulator's microworld in the following way. As described in Section 4.3, each experiment run consists of a fixed number of matches. In each match, agents play a new game but some information from previous games is allowed to be maintained

between games within the same experiment run. In this microworld, as agents perform actions, they gain more experience. The notion of experience is modeled by a reduction in the action cost for that agent. As the costs of agents can be carried over from previous games, agents can gain experience from the previous games in the current game (Figure 5.7).

The same concept explained here can be used for other types of research where a long-term memory for agents, as well as a longer-term retention of the changes to the environment, may be required (e.g. machine learning applications).

Chapter 6

The Simulation Process

This chapter outlines a selection of simulation experiments that have been conducted with the simulators developed within the software framework introduced in this thesis. These experiments constitute a part of the ongoing research into agent interaction protocols within the MAS research group at UNBC. Their significance in the context of that research has been described in two defended theses, [Nalbandyan, 2011] and [Dalvandi, 2012], as well as in two conference papers [Polajnar et al., 2011, 2012]. In the context of this thesis they are cited as test cases that illustrate the potential of the software framework to support different MAS models and different requirements of the experimenters. In particular, Section 6.1 describes the experiments for studying the MAP interaction protocol using the simulator introduced in Section 5.3; Section 6.2 describes the experiments for studying the Empathic Help Model, which is another AIP for helpful behavior in teamwork, using the simulator introduced in Section 5.4. Parts of the experiment descriptions, and the diagrams displaying the simulation results, are reproduced directly from the cited publications.

6.1 Example: The Mutual Assistance Protocol (MAP)

The experiments presented in this section have been conducted in order to study different properties of Action MAP and its advantages over other help strategies. First, the general experiment setup that has been used for these experiments is explained. Next, two experiments that demonstrate the framework's potential in modeling and experimentation with different aspects of Action MAP and the agents' environment are presented. These experiments were conducted by Narek Nalbandyan using the software described in this thesis.

6.1.1 The Experiment Setup

The parameters defined in the experiment setup that has been used for the experiments presented in this section are summarized below:

- The board size is 10 by 10.
- Each square on the board can have one of 6 different colors.
- Each team has 8 agents.
- The reward points for achieving each goal is 2000.
- The reward points for accomplishing each step on the chosen path to the goal square is 100 reward points.
- The initial allocation of resources for each agent is 200 points per each step in its chosen path towards its goal square.

- The overhead cost of performing a help act is 30 points.
- Four agent teams are considered: Action MAP, URIP, UHIP, and No-Help,
- The number of runs for each experiment is set to 10,000.

6.1.2 The Impact of Computation and Communication Costs

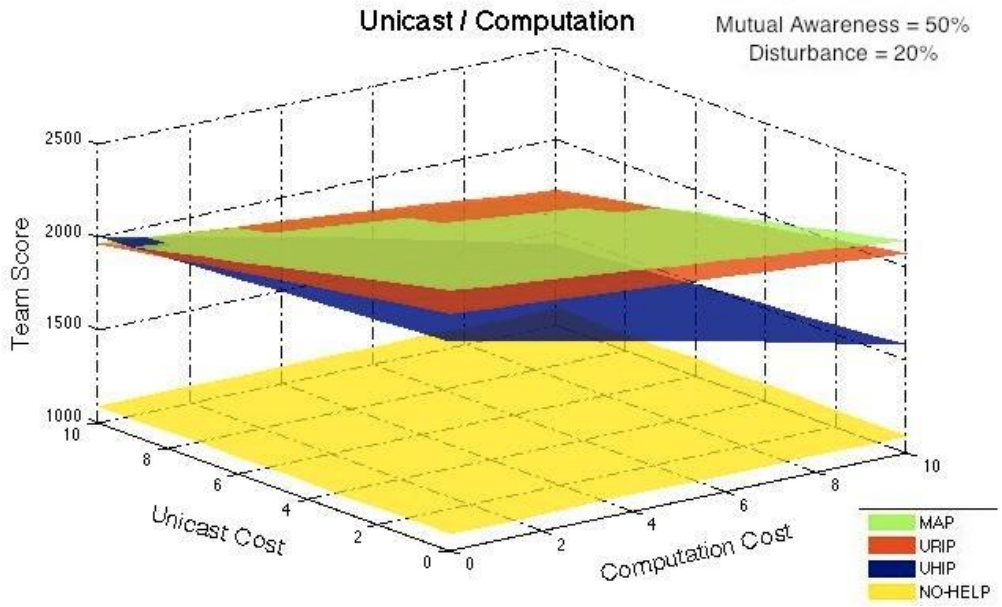

Figure 6.1: Team scores vs computation and communication (unicast) costs. Reprinted from: [Polajnar et al., 2012]

This experiment is designed to study the performance of Action MAP compared to other help strategies with respect to different costs that exist in realistic settings. In particular, this experiment looks at the costs associated to performing computations in assessing help requests and offers and sending and receiving messages in a network. These costs are referred to as *computation* and *communication (unicast)* costs.

The comparative performance of all four teams based on different computation and communication (unicast) costs is shown in Figure 6.1. The first observation is that

the three teams that use help protocols (URIP, UHIP, and MAP) perform better than the one that does not use any help model. However, their scores decrease as the costs of computation and communication increase. If the costs continue to increase beyond the area shown in the graph, the team that uses no help will eventually outperform the teams that use help. Furthermore, as observed in [Polajnar et al., 2012]:

> UHIP uses the most computation (leading to a sharper performance drop at the high-cost end) and the least communication (making it dominant for high communication and low computation costs). MAP scores are best overall. The relative scores of MAP vs URIP when communication costs are dominant depend on the relative cost of broadcast vs. unicast [...]. Our implementation of broadcast as $n - 1$ unicast messages favors URIP in the critical area.

This experiment demonstrates the potential of the framework in modeling and experimentation with some aspects of a multiagent system. First, because of the fact that performing computations affects the system's performance, it has been modeled as a cost (penalty) in the system. Second, the performance of the system has been measured in different levels of computation costs. Similarly, another penalty in the system's performance has been modeled as communication cost and been varied in the experimentation.

6.1.3 The Impact of Mutual Awareness and Disturbance

This experiment gives more insights about in what situations, based on two different parameters of the environment, teams that employ Action MAP perform better than

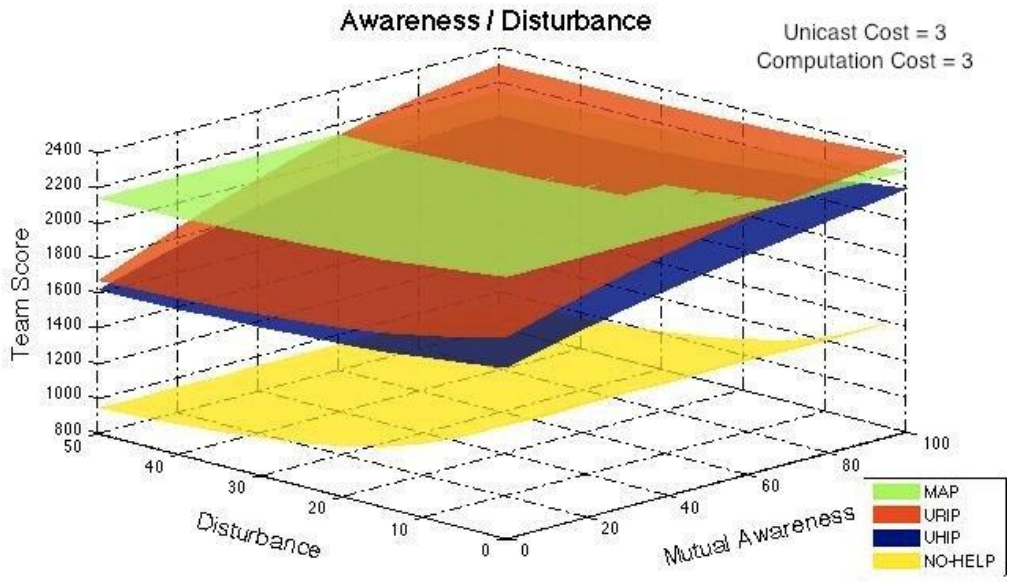

Figure 6.2: Team scores vs mutual awareness and environmental disturbance. Reprinted from: [Polajnar et al., 2012]

other teams. These parameters are the level of mutual awareness between agents, about the information they know from their teammates' skills; and the level of disturbance, representing the frequency of changes in the environment that occur dynamically. The experiment varies the level of mutual awareness and the environmental disturbance in order to compare the performance of agents using Action MAP with other unilateral strategies in different conditions of the environment.

The results of this experiment are shown in Figure 6.2. Their analysis in [Polajnar et al., 2012] is as follows:

> URIP and UHIP rely on mutual awareness, while MAP and NO-HELP do not. URIP and UHIP perform significantly worse than MAP for low mutual awareness, and improve as the rising knowledge of their teammates abilities improves unilateral judgment. At high mutual awareness, a URIP agent requests help from the right teammates, and owing to lower

communication costs (partly caused by our costly broadcast) outperforms the otherwise dominant MAP. Dynamic disturbance in the environment adversely affects all methods, because the stochastic color changes distort the effects of the initial planning (i.e., lowest-cost path selection). The distortion is most significant in the low-disturbance range, as evidenced by steeper performance drop; as the color composition of the paths gets closer to random, further stochastic disturbance causes less degradation. Throughout the range, mutual help partly compensates the effects of disturbance, as the very costly steps in an agent's path can be performed at lower cost by helpful teammates. As the figure shows, the performance of NO-HELP indeed degrades more significantly compared to other methods. MAP and UHIP degrade the least.

This experiment demonstrates how an experimenter can model and manipulate different aspects of a MAS environment in order study the behavior of agents in each configuration. This illustrates the capability of the framework in accommodating different characteristics of real-world environments in a simplified microworld model that is suitable for experimentation.

6.2 Example: The Empathic Help Model

This section first examines the framework's interoperability capabilities by explaining the technique that has been used for optimizing the empathic help model using its simulator and MATLAB. Next, I present two experiments about different aspects of the empathic help model that have been conducted by Behrooz Dalvandi using the

software described in this thesis.

6.2.1 Optimizing the Performance of the Empathic Model

The performance of an agent team that employs the empathic help model depends on four different parameters of the model. Three of these parameters, called the emotional state, past experience, and salience, are empathy factors that participate in the formation of affective response in the model of empathy for artificial agents introduced in [Dalvandi, 2012]. The *emotional state* of the subject of empathy influences the readiness to offer help to the object; *passed experience* of their mutual interactions is another influencing factor, as is the *salience* of the object's signals the indicating the need for help. The forth parameter is the *threshold* which the combined effects of the three empathy factors need to exceed in order for a help act to take place. In order to investigate the performance characteristics of empathic help compared to other help model, one needs to first determine which combination of values of the four parameters leads to the optimal team performance. The author has chosen to use Genetic Algorithms (GA) in order to determine the values of these parameters which together lead to the optimal performance of the agent team that employs the empathic help model. For performing the optimization, the Global Optimization (GO) toolbox in MATLAB is used together with the simulator introduced in Section 5.4. In the following, first a brief overview of the optimization process is explained. Next, the role of the simulation in the optimization process is discussed. Finally, the approach to connect the simulator to MATLAB's GO toolbox is presented.

In order to optimize a problem using GA, one needs to represent its possible solutions as *individuals (chromosomes)* (i.e. a string representation of some values) and

define a *fitness function* that can evaluate each solution. In the simulator's realm, the individuals are a tuples of distinguished parameters which affect the performance of the simulated multiagent system (representing a team). Each tuple of these parameter values then represents a possible solution. The fitness function's value is determined by the performance of the team that uses the specified parameter values and is measured by a metric defined by the model and implemented within the simulator.

In this setting, the simulator is used as the *fitness function*. It takes an individual generated by MATLAB and performs a simulation based on the values of parameters defined in the individual. After the simulation process is complete, it returns the value of the team score as the fitness value. Using this combination, the optimization algorithm finds the optimal settings for those four parameters based on the team performance in the simulation experiments.

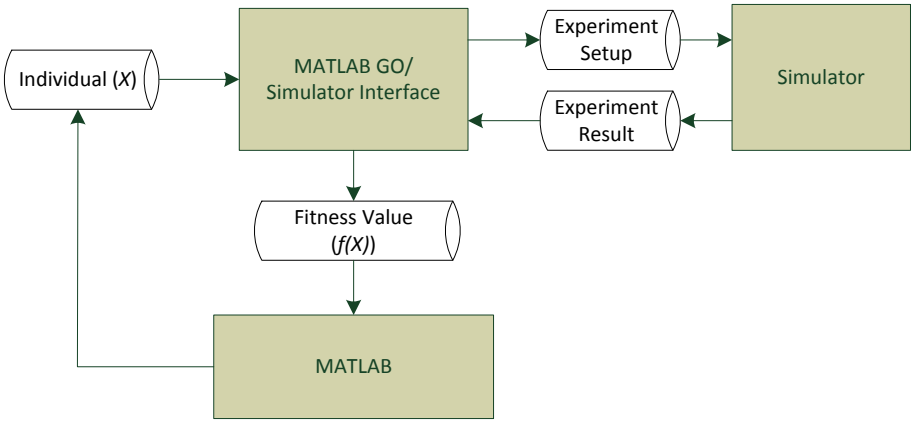

Figure 6.3: Interfacing the simulator with MATLAB's Global Optimization (GO) toolbox. The data flow for a single individual in the population. The fitness value for the rest of the population is evaluated in the same way.

In order to allow MATLAB to interact with the simulator, a specialized interface is developed. This small program is responsible for translating and managing

MATLAB's interactions with the simulator. MATLAB's GO toolbox can write each individual data in a file and read the fitness function's value from a file in each step of its GA algorithm. In each step, the toolbox generates a population of different individuals. Each individual represents a tuple of values of the four empathic model parameters and a fitness value will be associated to it at the end of a simulation process. The interface's job for each individual in the population, once it is invoked by the toolbox, is to read the individual data from the file, generate an experiment setup, and execute the simulator with the generated experiment setup (as presented in Figure 6.3). Upon finishing the simulation, the interface gets the experiment result from the simulator and writes it, as the value of the fitness function, into a file which is accessible by the toolbox. The toolbox uses this fitness value, checks whether a new generation is required, and if so, creates a new generation of individuals. This cycle repeats until the toolbox finds the optimal values based on its configuration. The optimal values determined by this method are then used in the simulation experiments to study the behavior of agents employing the empathic help model.

The results of the genetic algorithm optimization on the four empathic parameters are presented in Figure 6.4. The optimization process uses an initial population of 30 individuals and stops after it faces 50 stall generations. It uses the rank fitness scaling function and the stochastic uniform selection function. Note that because the optimization algorithm is defined so as to minimize the value of the fitness function, we use the negative value of the team score in the optimization process.

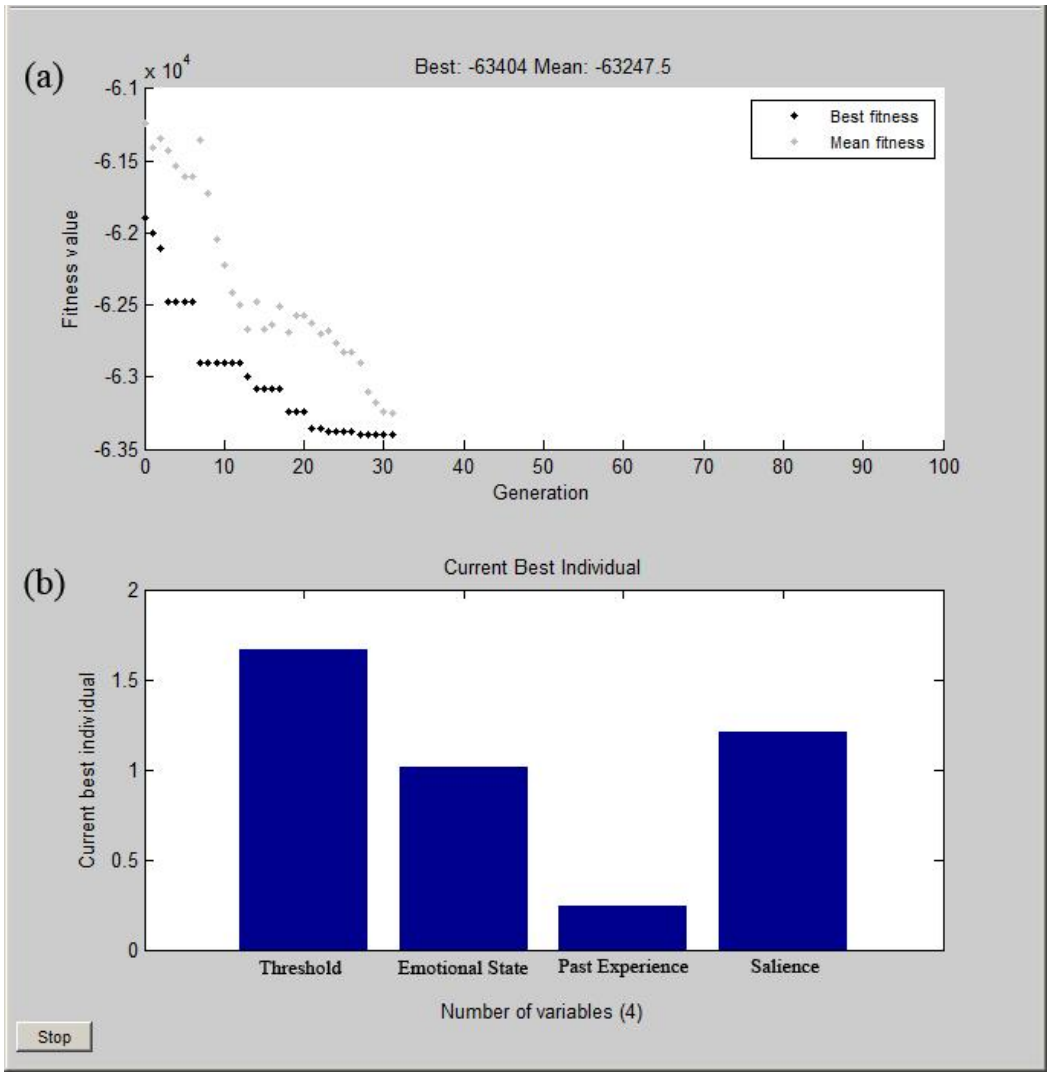

Figure 6.4: The genetic algorithm optimization of four empathic parameters produced by MATLAB. Reprinted from: [Dalvandi, 2012]

6.2.2 The Experiment Setup

The experiments with the empathic help model follow the same experiment setup designed for the Action MAP experiments in order to provide a unified basis for comparison. Also, the values determined by the optimization process are incorporated in the model for the following experiments.

6.2.3 The Validation of Empathy as a Help Trigger

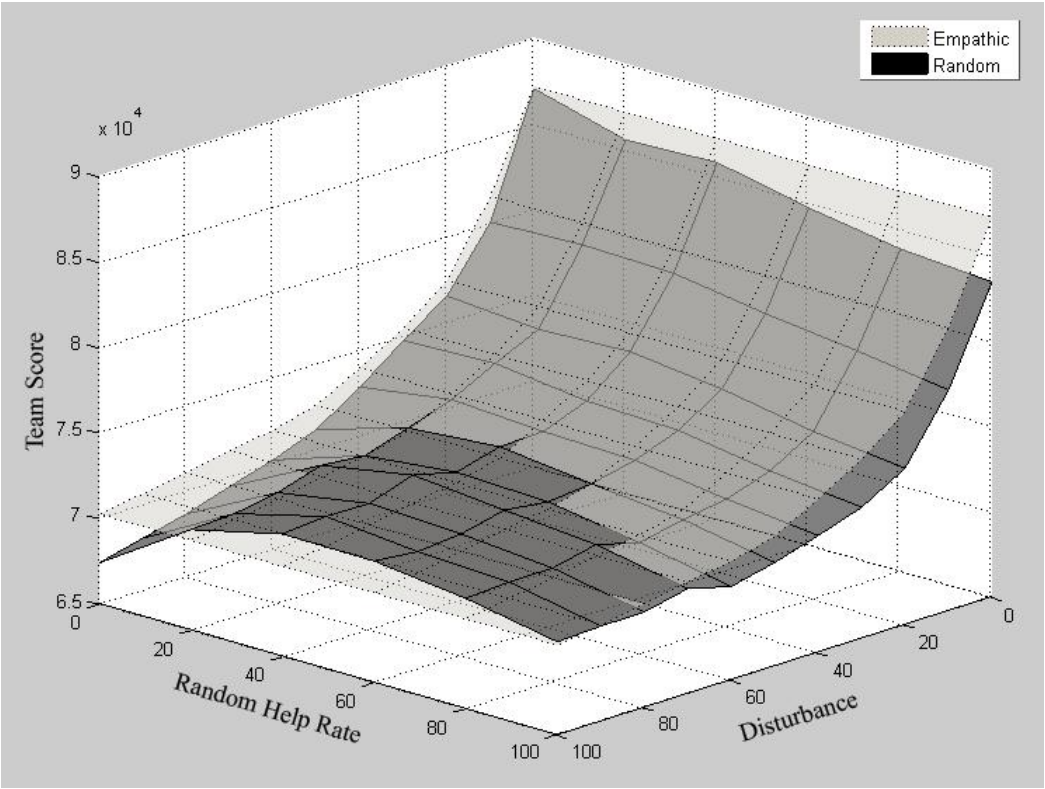

Figure 6.5: The performance of empathic team versus random-help. Reprinted from: [Dalvandi, 2012]

In order to investigate whether or not empathy can be an effective trigger of help in an agent team, a series of experiments are performed to compare the performance of empathic agents with the performance of a team in which agents provide help randomly. In the experiments, the agents from both teams use the same criteria in requesting help. For offering help (answering to the help requests), the agents in the seconds team randomly agree to help based on a fixed probability value, called the *random help rate*. These experiments vary the random help rate in different environmental disturbance levels (as described in Section 6.1).

Figure 6.5 shows the comparative team scores of the empathic team and the random help team for different values of random help rate and disturbance in the envi-

ronment. In low to moderate levels of disturbance the empathic team outperforms the random-help team for all levels of the help probability. This is indeed the area of highest interest for practical applications. A disturbance level of 50% implies that half of the world changes randomly at each step, and the fact that beyond that point most help strategies become ineffective is intuitively expected. This experiment series thus shows that empathy is a valid trigger for help most practically relevant situations. For higher levels of disturbance empathy-based help is no longer effective.

This experiment demonstrates how one can model and manipulate a parameter of an interaction model (random help model in this example) and study the behavior of agents in different configurations with respect to the parameter's value.

6.2.4 A Comparison of Empathic and Rational Help

In order to investigate the comparative performance of teams using empathic and rational help models, a suitable approach would be to realistically model the deliberation of rational agents about help acts in situations where the computational complexity of such deliberations can vary widely. This is because of the expectation that empathic approaches, which require less computation, might outperform rational ones as rational deliberations become increasingly complex. Such experiments can be conducted in future using the framework proposed in this thesis by connecting it to an external reasoning engine, measuring the real computational complexity of deliberations about help, and using it for performance comparisons with empathic help mechanisms.

The current experiment series relies on a simplification which does not allow for realistic comparisons of the actual decision costs in the empathic versus rational help models. However, it allows one to identify the general trends as the cost of rational

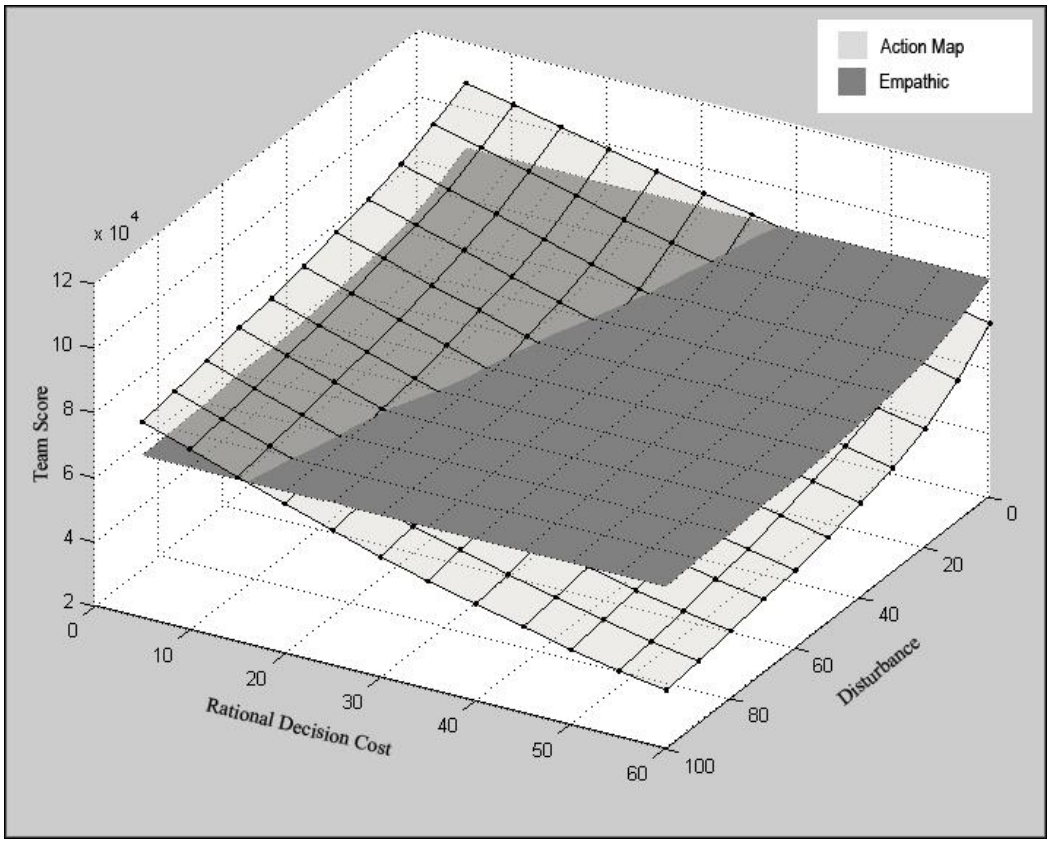

Figure 6.6: The performance of empathic team versus Action MAP (as a rational strategy) Reprinted from: [Dalvandi, 2012]

decision varies in the presence of environmental disturbance. For this purpose the cost of rational decision is modeled by an independent parameter called the *rational decision cost* which affects the team score of agents using Action MAP.

Figure 6.6 shows that the empathic agents exhibit superior performance when rational decisions become complex. The graph shows that this phenomenon is more pronounced at higher disturbance levels. This is because at higher disturbance levels the number of help requests increases and consequently the rational agents need to perform more calculations. The empathic agents have lower decision costs and can deal with such situations more efficiently.

Chapter 7

Analysis and Evaluation

The main objective of this research has been to provide a software framework for building simulators that facilitate the study of agent interaction models in early stages of their development. The framework's purpose has been to help the designer of an agent interaction model by reducing the design decision space through providing an environment for simulation experiments that supports early feedback on comparative performance of alternative solutions. To achieve that, a number of objectives and design principles have been identified in Section 3.3. In this chapter, I analyze the proposed framework with respect to its objectives and principles and evaluate to what extent they are addressed in this thesis.

The graphical user interface (GUI) mode of the front end (presented in Section 4.3) supports interactive control and manipulation of the simulation model. The experiment setup toolbox of the GUI allows the user to modify the simulation parameters at any time. The modifications are handled by the parameter manager service of the framework which guarantees that they apply to all the components of the simulator

and the simulation model. The feedback from the modifications is provided interactively by the visualization and console toolboxes of the GUI. In addition, the user can run an experiment step-by-step in order to study the behavior of agents in more detail. These features together provide interactive experiment control to the users of the simulators built using the framework.

The visualization toolbox of the GUI provides on-line visualization of the simulation results as well as the state of the world. It responds to any changes in the simulation model and results as they occur. Working along with the interactive experiment control feature, the interactive and dynamic visualization helps the designer of an interaction model to reduce the decision space by getting early feedback from simulation experiments.

The framework's design allows the user to preview the simulation results as their precision increases gradually. The front end of the framework that controls the experiment setup and the execution of the experiment can execute a series of experiments in multiple steps, in such a way that in each step all the experiments are executed with a low number of runs — which can be done in a reasonable time — and the results are displayed to the user. In further steps, the results are updated as more experiment runs are being executed. This feature supports the early feedback of the framework by allowing the user to preview the trend of any series of experiments before they get to the desired precision level.

The architecture support for concurrent simulation of multiple teams, as described in Sections 4.2 and 4.3, provides identical environments and experimental scenarios to each team while they are being executed concurrently. In order to achieve that, the architecture uses different layers of environment which allows the separation of global events from local events. The architecture keeps this separation transparent to mul-

tiagent system (MAS) models and their agents, allowing them to interact only with their own environment without knowing about the layers and other teams. Combined with interactive experimentation and visualization, the performance and behavior of all the teams are presented to the user at the same time as a part of the early feedback mechanisms of the framework. This feature has been successfully tested in all the experiments conducted using the simulators built with the framework, such as the ones presented in Chapters 5 and 6. It addresses the objective of reducing the design decision space by providing early feedback on the comparative performance of alternative candidate solutions in identical circumstances. The comparative performance can be observed during the course of the experiment and the parameters that affect it can be modified interactively for all candidates at the same time.

The generic MAS model presented in this thesis is based on a generic microworld model. It only captures the essential elements of expert teamwork (described in Sections 2.3 and 3.2) by representing agents, tasks and subtasks, expertise, plans, and helpful acts in simple forms. While it captures the substance of the problem that the experimenter intends to study, the model remains simple. The environment is represented by a board of colored squares where each color represents a different expertise that is required to perform a specific action. Agents are located on this board and each have a set of individual capabilities. Each agent can individually plan how to move from its initial location to its goal location and thus complete its subtask. In addition, this simple model does not require the agents to employ complex ontologies to deal with the domain knowledge. The agent architecture of this MAS model focuses on the interaction of agents and does not impose any requirements for using complex agent reasoning (although it is possible) in dealing with the environment. These features of the MAS model allow the designers to focus on dealing with agent interaction models rather than the domain knowledge and agent reasoning. This is

valuable in the early stages of the design process; later on, the analysis may require further confirmation or refinement in more elaborate contexts.

The framework's high-level structure (presented in Section 4.2) is designed to provide a low coupling between its MAS models component and the rest of the framework. The MAS models are designed as a separate component of the framework which can be plugged into the framework and interact with other framework's components. The MAS models' design can be flexibly changed as long as it uses the same few interactions with other components of the framework. This feature has been tested in our research projects that have been performed using the framework. It contributes substantially to the extendibility of the application domain of the framework beyond its initial scope.

The framework's interoperability, as presented in Section 4.2, is designed in two directions. First, it allows external systems to manipulate experiment setups and process simulation results through an adaptor program. In this approach, either the simulator or the external system can invoke the other; thus, it allows both client and server configurations for the simulator in interacting with external systems. The interoperability of the simulator has been validated during our research on empathic help model (Section 6.2). The decoupling of the framework from other pre- and post-processing applications, along with its interoperability features, allows the framework design to remain simple and focused on the key tasks, while delegating other potentially complex tasks, such as optimization, advanced visualization, or statistical analysis, to existing specialized software packages. This flexible architecture thus extends the framework's functionality.

Second, the framework allows its agents to employ more complex reasoning capabilities that can be provided by external reasoning engines. The agent architecture of

the framework can activate an external reasoning engine in the deliberation process of an agent to execute an agent program written in an agent programming language. This will delegate the reasoning to the external engine. This type of interoperability can be provided by developing an adaptor program to interact with a specific reasoning engine. This feature supports the transitions in agent interaction model development, from the early stages that favor simplicity and abstraction, to the more mature stages that benefit from more elaborate and realistic MAS contexts.

The architecture of the simulators developed within the framework is distributable in the sense that each simulator can either run on a single processor or be replicated to share the simulation load among a number processor nodes. The performance impact of the distributable architecture has been examined in the following experiment series. A simulator built using the framework was executed on a cluster of identical machines running Ubuntu Linux 10.04. Each machine had an Intel(R) Core(TM)2 Quad CPU Q6600 running at 2.40GHz and the nodes were connected through a 100Mb/s Ethernet network.

The simulator was set to conduct 10 experiments for 10 different levels of environmental disturbance. The experiments included 3 agent teams, each with 8 agents and the size of the board was set to 10x10. In order to get statistically significant results, each experiment was repeated 24000 times.

The experiment involved the execution of the same simulation task on a varying number of nodes in the cluster. The level of distribution was changed in 8 steps, from a single node run to a cluster with 8 nodes, each time adding a new node. The duration of the simulation was measured using the Linux/Unix *time*[1] utility.

[1]http://pubs.opengroup.org/onlinepubs/9699919799/utilities/time.html

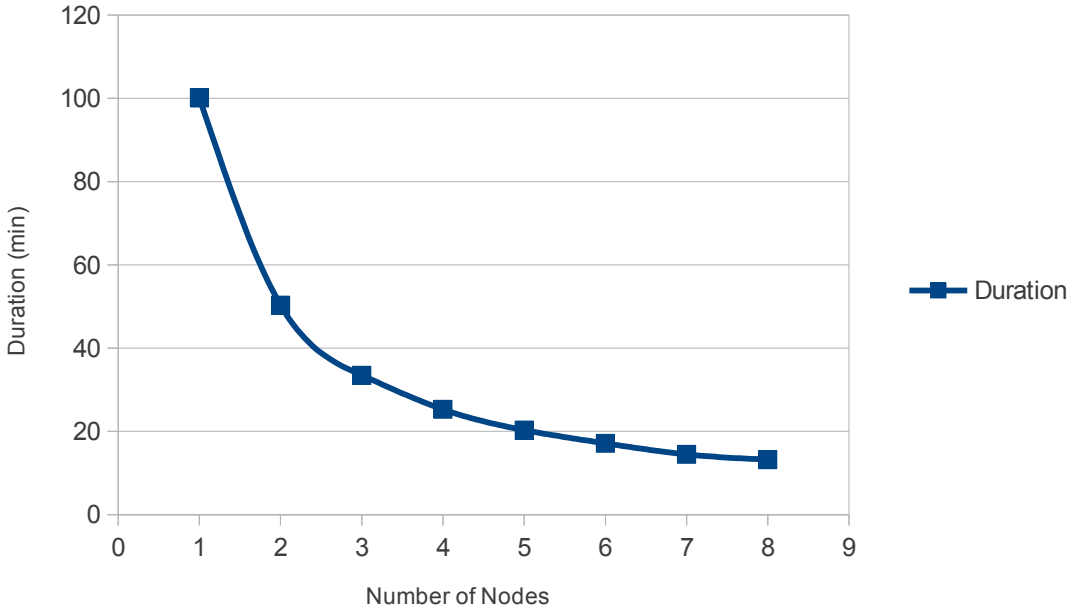

Figure 7.1: The duration of the distributed simulation for 24000 runs, based on the size of the cluster

The experimental results show that the framework's simulator architecture is fully scalable; as the number of nodes on the cluster increases, the speed of the simulation increases almost linearly. In this example, as presented in Figure 7.1, for a single node run, the time it took for the simulator to finish 24000 number of runs was 100.1 minutes whereas the time for the same simulation on a cluster with 4 nodes was 25.29 minutes which is almost $1/4th$ of the time it took with 1 node. The speed gain of the simulator over different numbers of nodes is presented in Figure 7.2. In all the cases, the difference between the speed gain and its ideal linear expectation is less than 5%.

The experimental results agree with the theoretical expectations according to the nature of these simulations. The almost linear speed up can be explained by the fact that the distributed simulation runs are different instances of the same experiment but independent from each other. The total number of runs can be broken into pieces

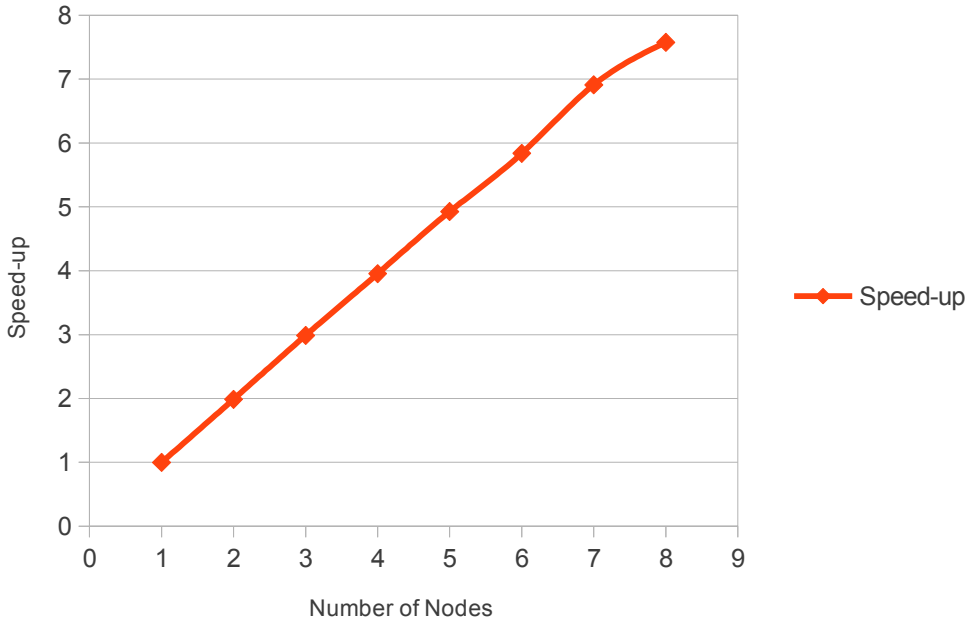

Figure 7.2: The speed-up of the distributed simulator for an experiment with 24,000 runs. The plot demonstrates the relative speed of distributed computation compared to the same computation performed on a single node.

that can be executed in parallel which results in better system performance. The scalability of the results comes from the fact that there is almost no overhead for communication/synchronization between the instances of the simulator distributed over multiple nodes. The only overhead in this approach is for the task distribution which is negligible. As a result, addition of a new node that can process a specific portion of the load will reduce the overall duration of the simulation proportionate to its given portion and linearly increase the simulation speed.

The analysis of the performance impact of the distributed simulation above shows that the method is indeed effective in increasing the speed of simulation and providing the results to the user faster.

The framework has shown a promising usability in the two research projects on

agent interaction protocols (AIPs) demonstrated in Chapters 5 and 6. The framework's capability for instantiating specialized simulators has been verified in those research projects. Various features of the framework have enabled the modeling and studying of different aspects of those AIPs and the flexibility of the framework's architecture for implementing new features has been examined during those studies.

In summary, the above analysis suggests that the proposed framework is a valuable tool for studying agent interaction models and is effective and suitable for a variety of research projects in agent interactions.

Chapter 8

Conclusions and Future Work

This thesis describes a new software framework for studying agent interaction models used in agent teamwork by means of simulation experiments. The framework facilitates the building of custom simulators for various multiagent system (MAS) models, which are then used in design-oriented simulation studies of agent interaction models in their early development stages. The main purpose of such a custom-made simulator is to reduce the design decision space by providing early feedback on comparative performance of alternative solutions in the course of simulation experiments, within a specific research domain.

The results presented in this thesis have helped overcome the methodological challenges that have been identified in the course of our investigation of concrete agent interaction protocols for helpful behavior in agent teamwork. Specifically, these challenges include: the problem of customizing the simulation tools to meet the requirements and emphasis of a particular type of MAS research; the problem of reducing the complexity of simulation models by finding simple and yet representative generic

abstractions for a particular domain; and the problem of making the tools sufficiently general, flexible, and extendible that variation in the MAS model structures does not require substantial changes in the existing tool set.

In order to address these challenges, the framework provides a generic simulator that can be instantiated with concrete MAS models. Each such MAS model is abstract and simple, based on a microworld, with a well-defined research scope and focus. It requires a very modest implementation effort and does not need elaborate support from the simulation environment. It is relatively easy to incorporate into the generic simulation environment provided by the framework in order to produce a custom simulator that effectively supports a specific direction of research.

The generic simulator architecture of the framework provides a novel combination of features that facilitate experimenting with agent interaction models. In particular, its generic simulation environment provides interactive experimentation and visualization, concurrent simulation of multiple teams, low coupling between its MAS model and the simulation environment, distributable architecture, and interoperability with agent reasoning engines and other external systems. The MAS models also share a generic architecture designed for the study of agent teamwork interactions. Its interaction module implements agent interaction protocols specified as interacting finite state machines. The microworld is inspired by the Colored Trails game but designed specifically for studies of helpful behavior in teams of artificial agents. Based on message passing in the current implementation, the communication infrastructure is open to extensions.

Central to the generic simulator architecture is the idea that the designer of a new agent interaction model, facing a variety of decisions with many possible outcomes, needs to be able to choose between alternative solutions by comparing their impacts

on team performance efficiently and directly, so that the design decision space can be substantially reduced early in the development cycle. In support of this idea, the generic simulator allows concurrent operation of multiple agent teams employing the alternative solutions, with immediate visualization of the simulation in progress, and lets the experimenter interactively control the simulation parameters, including the possibility of incrementally adjusting the number of runs in order to control the level of statistical significance of the results.

The framework supports interoperability of the generic simulator with external systems, as a client, server, or both. For instance, a custom simulator, acting as a client, has employed MATLAB in order to provide genetic algorithm (GA) optimization of its MAS model parameters, while at the same time, acting as server, it provided a simulation experiment per each individual (parameter value combination) in every GA generation, and returned their performance scores to the GA algorithm as their fitness values. In general, interoperability allows a custom simulator built within the framework to interactively delegate variety of tasks, such as optimization, statistical analysis, or advanced visualization, to existing specialized systems.

The computational complexity of the concurrent simulation of multiple MAS models can be overcome through a distribution of simulation runs of the same experiment across a potentially large number of nodes in a computing cluster. The achieved speedup in trials using up to eight nodes, has been found to be almost linear. This is attributed to the fact that individual runs are mutually independent and do not need to interact during execution.

The framework has been implemented and used in two research projects at UNBC to derive the custom simulators for two types of agent interaction protocols for helpful behavior in agent teamwork: the Mutual Assistance Protocol (MAP) and the Em-

pathic Help Protocol. MAP lets one team member directly help another based on their bilateral rational assessment that a help act is in the interest of the team, while in the empathic help protocol, the decision relies on the affective response based on a model of empathy for artificial agents. In both cases, the protocol designers have been able to run their own experiments, and the effectiveness of the simulator in achieving our formulated objectives has been confirmed.

With respect to possible future research, the following directions can be considered:

- Although the scope of this thesis is on teamwork contexts, the design of the framework does not exclude the possibility that the developed solutions may have a wider scope and be applicable, for instance, to selfish agents or to individual interactions without an immediate group context. Conducting experiments for applications other than agent teamwork could be an interesting direction for future work.

- Adding the support for a standard agent communication language as an underlying system for message passing would be an interesting part of the future work of this thesis. In most of the agent-oriented programming languages and MAS platforms, agents employ an agent communication language in their communication. For more realistic experimentation, the support for such language would be valuable for the users of the framework.

- As a by-product of the work on this thesis, a direction for its future work is to develop an execution layer for agent interaction models that allows abstracting the interaction models in agent programs. This would separate the interaction mechanisms from the rest of the mainstream agent code. The execution layer can be developed independently from multiagent platforms in order to make it

interoperable with different MAS platforms. This can be an extension of the parts of the framework which allow the agent architecture, considering its AIP execution capability, to employ external agent reasoning engines.

- The current version of the framework only implements the message passing communication mechanism used for AIPs. Communication using environment and communication based on a shared storage are not yet fully implemented. Implementing these mechanisms and providing relevant test-cases for them would be a part of the future work.

Bibliography

Huib Aldewereld, Wiebe van der Hoek, and John jules Meyer. Rational teams: Logical aspects of multi-agent systems. Technical report, Utrecht University, 2004.

J. Alferes, A. Brogi, J. Leite, and L. Pereira. Evolving logic programs. *Logics in Artificial Intelligence*, pages 50–62, 2002.

Fabio Bellifemine, Federico Bergenti, Giovanni Caire, and Agostino Poggi. Jade - a java agent development framework. In Rafael Bordini, Mehdi Dastani, Jürgen Dix, Amal Fallah Seghrouchni, and Gerhard Weiss, editors, *Multi-Agent Programming*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, pages 125–147. Springer US, 2005. ISBN 978-0-387-26350-2.

Rafael H. Bordini, Lars Braubach, Mehdi Dastani, Amal El Fallah Seghrouchni, Jorge J. Gómez Sanz, João Leite, Gregory M. P. O'Hare, Alexander Pokahr, and Alessandro Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica (Slovenia)*, 30(1):33–44, 2006.

R.H. Bordini, J.F. Hübner, and M. Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. Wiley-Interscience, 2008.

B. Chen and S. Sadaoui. *A Generic Formal Framework for Multiagent Interaction Protocols*. Citeseer, 2003.

Wai-Khuen Cheng, Boon-Yaik Ooi, and Huah-Yong Chan. Resource federation in grid using automated intelligent agent negotiation. *Future Gener. Comput. Syst.*, 26(8): 1116–1126, October 2010. ISSN 0167-739X. doi: 10.1016/j.future.2010.05.012. URL `http://dx.doi.org/10.1016/j.future.2010.05.012`.

K. L. Clark and F. G. McCabe. Go!-a multi-paradigm programming language for implementing multi-threaded agents. *Annals of Mathematics and Artificial Intelligence*, 41:171–206, August 2004.

P.R. Cohen and H.J. Levesque. Teamwork. *Nous*, 25(4):487–512, 1991.

R.W. Collier. *Agent factory: A framework for the engineering of agent-oriented applications*. PhD thesis, University College Dublin, 2002.

D.D. Corkill. Blackboard systems. *AI expert*, 6(9):40–47, 1991.

R.S. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng. Modeling agent conversations with colored petri nets. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 59–66, 1999.

Stefania Costantini and Arianna Tocchio. A logic programming language for multi-agent systems. In *Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf., JELIA 2002, LNAI 2424*, pages 1–13. Springer-Verlag, 2002.

Behrooz Dalvandi. A model of empathy for agent teamwork. Masters thesis, University of Northern British Columbia, January 2012.

B. Dunin-Keplicz and R. Verbrugge. *Teamwork in Multi-Agent Systems: A Formal*

Approach. Wiley Series in Agent Technology. John Wiley & Sons, 2011. ISBN 9781119957607.

H.R. Dunn-Davies, R.J. Cunningham, and S. Paurobally. Propositional statecharts for agent interaction protocols. *Electronic Notes in Theoretical Computer Science*, 134 (0):55 – 75, 2005. ISSN 1571-0661. Proceedings of the First International Workshop on Euler Diagrams (Euler 2004).

R. Evertsz, M. Fletcher, R. Jones, J. Jarvis, J. Brusey, and S. Dance. Implementing industrial multi-agent systems using jack tm. *Programming multi-agent systems*, pages 18–48, 2004.

Xiaocong Fan, John Yen, and Richard A. Volz. A theoretical framework on proactive information exchange in agent teamwork. *Artif. Intell.*, 169(1):23–97, November 2005. ISSN 0004-3702.

Nicholas V. Findler and Gregory D. Elder. Multi-agent coordination and cooperation in a distributed dynamic environment with limited resources. *Artificial Intelligence in Engineering*, 9:229–238, 1995.

G. Fortino and W. Russo. Multi-coordination of mobile agents: a model and a component-based architecture. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 443–450. ACM, 2005.

Y. Gal, B. Grosz, S. Kraus, A. Pfeffer, and S. Shieber. Agent decision-making in open-mixed networks. *Artificial Intelligence*, 2010. URL `http://dx.doi.org/10.1016/j.artint.2010.09.002`.

K.V. Hindriks, F.S. De Boer, W. Van der Hoek, and J.J.C. Meyer. Agent programming in 3apl. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

O.E. Holland. Multiagent systems: Lessons from social insects and collective robotics. In *Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium*, pages 57–62, 1996.

Ece Kamar, Ya Gal, and Barbara J. Grosz. Incorporating helpful behavior into collaborative planning. *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, 2:875–882, 2009.

David Keil and Dina Goldin. Indirect interaction in environments for multi-agent systems. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *Environments for Multi-Agent Systems II*, volume 3830 of *Lecture Notes in Computer Science*, pages 68–87. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-32614-4.

H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, pages 340–347. ACM, 1997.

H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada. Robocup rescue: search and rescue in large-scale disasters as a domain for autonomous agents research. In *Systems, Man, and Cybernetics, 1999. IEEE SMC '99 Conference Proceedings. 1999 IEEE International Conference on*, volume 6, pages 739 –743 vol.6, 1999.

I. Kotenko. Simulation of agent teams: the application of domain-independent framework to computer network security. In *23rd European Conference on Modeling and Simulation (ECMS2009). Madrid, Spain*, pages 137–143, 2009.

G.J.M. Kruijff, F. Colas, T. Svoboda, J. van Diggelen, P. Balmer, F. Pirri, and R. Worst. Designing intelligent robots for human-robot teaming in urban search & rescue. In *2012 AAAI Spring Symposium Series*, 2012.

João Alexandre Leite, José Júlio Alferes, and Luís Moniz Pereira. Minerva - a dynamic logic programming agent architecture. In *Revised Papers from the 8th International Workshop on Intelligent Agents VIII*, ATAL '01, pages 141–157, London, UK, UK, 2002. Springer-Verlag.

Jeffrey A Lepine, Mary Ann Hanson, Walter C Borman, and Stephan J Motowidlo. *Contextual performance and teamwork: Implications for staffing*, volume 19. 2000.

Xiong Li, Xianggang Liu, Zhiming Dong, and Kunju Li. Toward an agent-based model of tactical engagement. In *Advanced Management Science (ICAMS), 2010 IEEE International Conference on*, volume 3, pages 218 –223, july 2010. doi: 10.1109/ICAMS.2010.5553251.

Jürgen Lind. Specifying agent interaction protocols with standard uml. In *Revised Papers and Invited Contributions from the Second International Workshop on Agent-Oriented Software Engineering II*, AOSE '01, pages 136–147, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-43282-5.

M. Luck, P. McBurney, and C. Preist. *Agent technology: enabling next generation computing (a roadmap for agent based computing).* AgentLink/University of Southampton, 2003.

Nancy Ann Lynch. *Distributed Algorithms.* Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann Publishers, 1996. ISBN 9781558603486.

Hamza Mazouzi, Amal El Fallah Seghrouchni, and Serge Haddad. Open protocol design for complex interactions in multi-agent systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 517–526. ACM Press, 2002.

F. Mokhati, M. Badri, and L. Badri. A formal framework supporting the specification of the interactions between agents. *Informatica (Slovenia)*, 31(3):337, 2007.

L. Monostori, J. Vancza, and S.R.T. Kumara. Agent-based systems for manufacturing. *CIRP Annals - Manufacturing Technology*, 55(2):697 – 720, 2006. ISSN 0007-8506. doi: 10.1016/j.cirp.2006.10.004.

R. Nair, T. Ito, M. Tambe, and Stacy Marsella. The role of emotions in multiagent teamwork: A preliminary investigation. *Who needs emotions: the brain meets the robot*, (213):1–23, 2003.

Narek Nalbandyan. A mutual assistance protocol for agent teamwork. Masters thesis, University of Northern British Columbia, September 2011.

C. Nikolai and G. Madey. Tools of the trade: A survey of various agent based modeling platforms. *Journal of Artificial Societies and Social Simulation*, 12(2):2, 2009.

J. Odell, H. Van Dyke Parunak, and B. Bauer. Representing agent interaction protocols in uml. In *Agent-Oriented Software Engineering*, pages 201–218. Springer, 2001.

Andrea Omicini and Enrico Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277 – 294, 2001.

Andrea Omicini and Franco Zambonelli. Coordination for internet application development. *Autonomous Agents and Multi-Agent Systems*, 2:251–269, September 1999. ISSN 1387-2532.

Pier Palamara, Vittorio Ziparo, Luca Iocchi, Daniele Nardi, and Pedro Lima. Teamwork design based on petri net plans. In Luca Iocchi, Hitoshi Matsubara, Alfredo Weitzenfeld, and Changjiu Zhou, editors, *RoboCup 2008: Robot Soccer World Cup*

XII, volume 5399 of *Lecture Notes in Computer Science*, pages 200–211. Springer Berlin / Heidelberg, 2009. ISBN 978-3-642-02920-2.

Philippe Pasquier, Ramon Hollands, Iyad Rahwan, Frank Dignum, and Liz Sonenberg. An empirical study of interest-based negotiation. *Autonomous Agents and Multi-Agent Systems*, 22:249–288, 2011. ISSN 1387-2532. 10.1007/s10458-010-9125-6.

Shamimabi Paurobally and Jim Cunningham. Achieving common interaction protocols in open agent environments. In *In Proceedings of the 2nd international workshop on Challenges in Open Agent Environments*, page paurobally.pdf., 2002.

Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. Jadex: A BDI reasoning engine. In J. Dix R. Bordini, M. Dastani and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming*, pages 149–174. Springer Science+Business Media Inc., USA, 9 2005. Book chapter.

Jernej Polajnar, Behrooz Dalvandi, and Desanka Polajnar. Does empathy between artificial agents improve agent teamwork ? In *Proceedings of the 10th IEEE International Conference on Cognitive Informatics & Cognitive Computing*, pages 96–102, Banff, Alberta, 2011.

Jernej Polajnar, Narek Nalbandyan, Omid Alemi, and Desanka Polajnar. An interaction protocol for mutual assistance in agent teamwork. In *Proceedings of the Sixth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS-2012)*, 2012.

Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Proceedings of the 7th European workshop on Modelling autonomous agents in a multi-agent world : agents breaking away: agents breaking away*, pages

42–55, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc. ISBN 3-540-60852-4.

S.J. Russell and P. Norvig. Artificial intelligence: A modern approach, 1995.

R.E. Shannon. *Systems simulation: the art and science*. Prentice-Hall Englewood Cliffs,, New Jersey, 1975.

Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2008. ISBN 9780521899437.

Maarten Sierhuis, Jeffrey M. Bradshaw, Alessandro Acquisti, Ron van Hoof, Renia Jeffers, and Andrzej Uszok. Human-agent teamwork and adjustable autonomy in practice. In *Proceedings of the Seventh International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, 2003.

M.P. Singh. Group ability and structure. *Decentralized AI*, 2:127–146, 1991.

R.G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *Computers, IEEE Transactions on*, C-29(12):1104 –1113, dec. 1980. ISSN 0018-9340.

Ian Sommerville. *Software Engineering*, pages 119–129. International computer science series. Pearson/Addison-Wesley, 2004. ISBN 0321210263.

V.S. Subrahmanian. *Heterogeneous agent systems*. The MIT Press, 2000.

R. Sun. *Cognition and multi-agent interaction: From cognitive modeling to social simulation*. Cambridge Univ Pr, 2006.

Katia Sycara and Gita Sukthankar. Literature review of teamwork models. Technical Report CMU-RI-TR-06-50, Carnegie Mellon University, November 2006.

M. Tambe. Towards flexible teamwork. *Arxiv preprint cs/9709101*, 1997.

M. Tambe, P. Scerri, and D.V. Pynadath. Adjustable autonomy for the real world. *Journal of Artificial Intelligence Research*, 17:171–228, 2002.

M. Thielscher. Flux: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming*, 5(4-5):533–565, 2005.

Adelinde M. Uhrmacher and Danny Weyns. *Multi-Agent Systems: Simulation and Applications*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2009. ISBN 1420070231, 9781420070231.

Tom Wanyama and Behrouz Homayoun Far. A protocol for multi-agent negotiation in a group-choice decision making process. *J. Netw. Comput. Appl.*, 30(3):1173–1195, August 2007. ISSN 1084-8045. doi: 10.1016/j.jnca.2006.04.009. URL `http://dx.doi.org/10.1016/j.jnca.2006.04.009`.

G. Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Intelligent Robotics and Autonomous Agents Series. MIT Press, 2000. ISBN 9780262731317.

Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, second edition, 2009.

Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

J. Yen, X. Fan, and R. Volz. Proactive communications in agent teamwork. *Advances in Agent Communication*, pages 1959–1959, 2004.

N. Yorke-Smith, S. Saadati, L.M. Karen, and N.M. David. The design of a proactive

personal agent for task management. *International Journal on Artificial Intelligence Tools*, 21(01), 2012.