Utrecht University

University of Edinburgh

Msc Thesis Cognitive Artificial Intelligence

Agent Organization Framework for Coordinated Multi-Robot Soccer

Author:
Gwendolijn Schropp
3345319

Supervisors:
Prof. Dr. John-Jules Meyer (UU)
Dr. Subramanian Ramamoorthy (UoE)

A thesis submitted in fulfilment of the requirements of 30 ECTs for the degree of Master of Science

carried out for the

Department of Philosophy, Faculty of Humanities (UU)

in the

Robust Autonomy and Decisions group, Faculty of Informatics (UoE)

May 2014

UTRECHT UNIVERSITY

Abstract

Faculty of Humanities
Department of Philosophy

Master of Science

Agent Organization Framework for Coordinated Multi-Robot Soccer

by Gwendolijn SCHROPP

In this thesis my final Msc research project in Cognitive Artificial Intelligence is presented. The main issue adressed in this work is the problem of 'ad hoc coordination'. Coordination in this context mainly is cooperation in teamwork: interaction between multiple agents that share a certain environment or system, whilst trying to achieve certain goals or objectives together. When coordination is 'ad hoc', an agent does not know what to expect of the other agents and their plans, but nevertheless has to contribute to their teamwork in achieving goals. In this project, the domain of robot soccer is taken as a specific application of the problem of ad hoc coordination, with special attention to the coach robot. The contribution of this work to the problem is twofold: from an agent theory point of view, a formal framework for the robot soccer society of the RoboCup Standard Platform League is designed using the OperA methodology for agent organizations. This framework is grounded in a combination of deontic and temporal logics and provides structures for coordination while still being flexible and allowing for extension with various agent architectures and other lower level implementations. In order to ground the concepts used in the framework, a sensor data-driven module is developed to infer an agent's plans. In order to be able to coordinate ad hoc, the coach first has to learn the ways and plans of his teammates and/or opponents, before deciding on how to adapt his strategy in order to improve the team's performance. In collaboration with the University of Edinburgh's Robust Autonomy and Decisions robotics group, a plan recognition module has been developed. As an extension of the current methodology towards a more high-level approach of multi-agent interaction, the domain is approached from a logical, multi-agent theory point of view, aiming at structured coordination and teamwork. This thesis yields a thorough agent organization model of the robot soccer society combined with a plan recognition module and suggestions on their connection.

Contents

Chapter 1

Introduction

The main inspiration for the work in this thesis is the problem of *ad hoc coordination in a multi-agent system*. Ad hoc coordination (section 1.2.2) is coordination, for instance collaboration between multiple entities to achieve some goal, without prior knowledge of the other entities. Multi-agent systems (section 1.1) are models or applications consisting of multiple entities or *agents* that co-exist in a shared environment. The domain of this work is that of *robot soccer*, specifically the humanoid Standard Platform League of the RoboCup organization (section 1.2.1). The work in this thesis consists of two parts. The parts are different in how they approach the problems researched here, but very relevant to the same field. The combination of different perspectives is what characterizes research in Artificial Intelligence (AI), especially in robotics, where contributions are made in fields ranging from kinematics and motor control, computer vision, machine learning, agent architectures, logic, reasoning and (cognitive) behaviour to linguistics (natural language processing and speech synthesis for example) and philosophical research like the studies of mind and knowledge. A lot of those different views and techniques are explored here, although the focus will be on **multi-agent theory** and **(probabilistic) machine learning**. In this chapter, several concepts are introduced to provide an appropriate background for this thesis' content.

1.1 Multi-Agent Theory

The concept of an *agent* has been popular in AI and computer science for years. The reason for that is that agents and agent theory are very powerful paradigms in the design, representation, simulation and understanding of various (real-world) domains. Agent theory is the field in which mathematical or logical formalisms for both *representing* and *reasoning* about agent properties is investigated [59]. Multi-agent theory is concerned with groups (or *societies*) of agents, where it is often the case that these agents have to cooperate and interact with each other to achieve (common and individual) *goals*. Multi-agent systems are extensions of single-agent systems, adding infrastructures for interaction and communication [31]. These aspects are needed to enable the agents to work together and negotiate about who is supposed to do what, in order to achieve goals. It can also be the case that agents in a system are not cooperating but competing. However, competition is also a form of interaction between agents: for example if one agent is impeding another agent from achieving its goal, they are of mutual influence as much as when they would be working together.

Agents are especially suitable for open and dynamic systems, because of their ability to support representation, coordination and cooperation between heterogeneous processes [31]. One obvious example here is the robot soccer domain, where the robots are the agents of the system.

1.1.1 Agents

One of the best known definitions of an agent is the one by Wooldridge and Jennings [59]:

> 'An agent is an encapsulated computer system that is situated in some environment and that is capable of flexible, autonomous action in that environment in order to meet its design objectives.'

We can view the agent as a problem-solving entity. The definition can be explained using the following characteristics:

- autonomy: an agent should be able to operate without (human) intervention, having some form of control over its actions and internal states.
- social ability/communication: agents should be able to interact with each other or with humans, e.g. via some *agent communication language* (section 2.3.2).
- reactivity: agents should be able to perceive their environment (physical or via a user interface or otherwise/a combination) and act upon the changes that occur.
- pro-activeness: agents should also be able to not merely respond, but take the initiative to achieve goals.

Sometimes *rationality* is added to this list: the assumption that an agent has knowledge and beliefs that he can act upon in order to achieve a certain goal [72]. Such agents are sometimes called 'cognitive' if they are endowed with mental attitudes representing the world and motivating their actions [31, 105]. According to Wooldridge and Jennings, mentalistic notions as knowledge, belief, intention or even emotional notions correspond to the concept of 'strong AI', a field of AI-research aiming to approach human-like intelligence instead of developing new ways to achieve intelligent behaviour (as in 'weak AI'). One of the aspects of human-like intelligence that is aimed for in recent research is *learning*. Learning is what enables an agent to exhibit intelligent behaviour: from imitating a person or another robot to being able to solve problems that have never been encountered before. Learning is sometimes seen as the skill agents need to be deemed 'truly intelligent' [31].

Several kinds of agents can be distinguished by their architecture, for example logic-based agents, BDI-agents, reactive agents and layered agents [31]. In logical agents, or knowledge-based agents, knowledge and reasoning is used to deal with partially observable environments [80]. The agent should be provided with a knowledge base, containing sentences in a knowledge representation language (First Order Logic (FOL) for instance), describing states of affairs in the environment. Combining these with current percepts creates the possibility to infer new aspects of the current state, which can help the agent decide about his actions. Besides knowledge, agents can also have *beliefs* (an attitude towards a proposition, for example 'Player 1 believes that [Player 2 is in possession of the ball]'), *desires* and *intentions*: such agents are known as BDI-agents [77]. Based on these mental or internal states, they interact with their environment. In contrast, there are reactive agents, which behave in a more basic way of merely sensing their environment and acting directly upon those percepts, meaning that they don't employ any reasoning step in between. This can be used for example in obstacle avoidance tasks in which higher-level reasoning is not necessary to achieve goals. Layered agents are agents that make decisions via several software layers of different levels of abstraction.

Why agents?

As mentioned before, the agent paradigm is particularly suitable for robotic systems because robots clearly are encapsulated (or embodied) computer systems, situated in an environment in which they interact in order to achieve certain goals. Agent theory provides a natural, intuitive way to view or describe robotic systems. Moreover, since agents are autonomous entities in general (the definition should not restrict to computer systems per se), the ideas of agent theory apply to humans or even entire organizations as well [62]. Multi-agent systems are widely used to handle organizational problems, like the collective achievement of tasks [33].

1.1.2 Agent Organizations

As in Virginia Dignum's OperA framework, which is the main inspiration and source for the framework in this thesis, we adopt an organization-oriented view on the design of multi-agent systems. Organizations of humans or other agents can be seen as sets of entities that are *regulated* by mechanisms of social order and designed to achieve common goals. Moreover, 'the role of any society is to allow its members to coexist in a shared environment and pursue their respective roles in the presence and/or in cooperation with others' [31].

The idea of a robot soccer team as an agent society is fairly straightforward: the team as a whole has the common goal to win a match (which can be decomposed in 'score more goals than the opponent team'). The structure, for example the formation of the team and how agents could cooperate, is determined by specific *roles*, interaction rules and a communication language. Furthermore, there should be *norms* describing the desirable behaviour of the agents, like the rules and regulations of a RoboCup match.
The specification of this organizational structure for a robot soccer team, its roles, norms and interaction rules, covers the majority of this research.

Logic

The robot soccer environment can be modelled as a logic-based multi-agent system, which will provide for a solid and consistent structure. Logic is a powerful and appropriate way of formalizing real-world concepts and situations and specify reasoning patterns of the agents therein. Using the input from the environment via the robot's sensors as knowledge, logic can reason about appropriate actions or changes in the robot's goals on a higher level of abstraction. Moreover, formalizing the robot soccer society is an important development step towards implementing reusable and shared knowledge representations, reasoning and interaction patterns.

1.2 Robot Soccer

The aim of the RoboCup competition is to win against a human team by 2050, which led to a recent interest in analysis of human soccer teams to improve the tactics and coordination of robot soccer teams [9, 10, 107]. Clearly, in human soccer, the players interact and communicate with each other and enact different roles with according tactics during a match. This makes an organizational framework such as OperA well-suited for formalizing robot soccer, with human soccer as inspiration and including human-like interaction structures.

1.2.1 RoboCup and Edinferno

RoboCup is an international robotics competition founded in 1997[1]. It consists of a rescue domain, a home domain, a junior domain for children and of course the soccer domain. The aim of the competition is to promote robotics and Artificial Intelligence research in a way that is appealing to the greater public. The ultimate goal is to have a team of fully autonomous humanoid robot soccer players able to win a soccer game against a professional human soccer team.
The RoboCup soccer domain is divided into categories based on the size and properties of robots. The Standard Platform League is especially for teams of NAO robots[2]. Contestants to the SPL competition focus on multi-agent research in dynamic adversarial environments and software development, without changing the hardware of the robots.

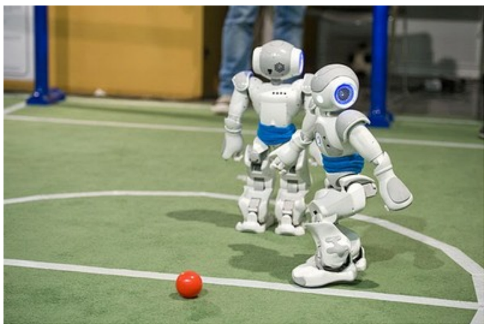


FIGURE 1.1: NAO's in action

NAO's, developed by Aldebaran Robotics[3], are 57cm tall humanoid robots that have 25 degrees of freedom with joints at the head, shoulder, elbow, wrist, hip, knee and ankle. Currently, the hands and fingers are not used in robot soccer (although they will be in the near future). They are equipped with cameras, infrared sensors and tactile sensors ('whiskers') to help them determine their environment and move around in it.

[1]http://www.robocup2014.org/
[2]http://www.informatik.uni-bremen.de/spl/bin/view/Website/WebHome
[3]http://www.aldebaran.com/en

This thesis project is conducted in (collaboration with) University of Edinburgh's Team Edinferno[4], consisting of undergraduate and graduate students combined with experienced researchers from Informatics, AI and Neuroscience. The focus of this group lies on autonomous and robust decision making mechanisms in continually changing and strategically rich environments, while also working on robot control and motion synthesis. Edinferno has 7 H21 V4 NAO's for developing and testing the code in the in-house small-scale pitch and 5 more (newer) H25 NAO's that can be used in the competition. They have debuted in 2011 and reached the quarter-finals in 2012. This year, Edinferno will compete again at RoboCup in Brazil (19-25 July).

FIGURE 1.2: Edinferno logo

FIGURE 1.3: The human part of team Edinferno, 2014

1.2.2 Ad Hoc Coordination

As a proof-of-concept, one aspect of a robot soccer team as an organizational structure is explored in more detail. The aim of this part is to start the development of a more *ad hoc* approach to teamwork. Each year, the RoboCup organisation publishes challenges for the SPL. Besides improving overall robustness, strategy and teamwork, the participating teams can attempt to solve the problems presented in these challenges. One of the most recent challenges is the 'drop-in player' challenge. The main point of this challenge is to develop players that can be good teammates and play well with a team of unknown players. This corresponds to the work of for instance [7, 8, 41, 94, 95, 97] on *ad hoc coordination*. Coordination is the interaction and communication between agents in a multi-agent system which enables them to achieve both individual and global goals, in collaboration. For coordination to be 'ad hoc' means that the

[4]http://wcms.inf.ed.ac.uk/ipab/robocup/home

agents do not share a common protocol, but have to learn and adapt to their teammates and adversarials depending on their plans, actions and strategies. The ad hoc agent must be able to recognize plans or tactics of his teammates (or opponents - this is the subject of the related field of *opponent modelling*) and subsequently *decide* about his own plans to complement those of his teammates in order to achieve common goals (section 4.1.1). This is the problem of *plan recognition*, which is studied and implemented in this project as a first step towards an ad hoc agent.

In the process of designing an ad hoc agent there is a great overlap with modules that would be needed for another useful RoboCup character: a *coach*. The coach is an extra, non-playing NAO robot watching the game from the sideline and giving tips to the players in the team. A coach can be a very useful addition to the team, because he can focus on merely observing what everyone is doing and where the ball is, without also having to keep track of its own position. Tips he could give are for instance which players are to be trusted, who should have the ball and whether or not there is a scoring opportunity.

1.3 Problem Description

One of the things that make robotics a challenging field of research, is the fact that the world or environment of the robot is dynamic: it changes over time. This results in incomplete or even inconsistent views of that environment. Especially in multi-agent domains, these characteristics have to be taken into account: simply fitting individual agents with precomputed coordination plans will not do, for their inflexibility can cause failures in teamwork [97]. Furthermore, in ad hoc coordination, there are two main problems: plan recognition and adaptation (related to (machine) learning). The focus of this thesis is twofold: researching plan recognition methods for robot soccer in the literature and designing and implementing a module that can be expanded into a full ad hoc agent, and designing an abstract logical framework for communication and coordination in the agent society of the robot soccer domain.

1.3.1 Relevance of the Subject

The two parts of research each have a different approach and therefore are relevant to AI in general and the master Cognitive Artificial Intelligence in slightly different ways. The plan recognition part includes methods of robotics, probability theory and machine learning, whereas the abstract framework part is based on theories of logic, intelligent agents and multi-agent systems.
In terms of relevance to RoboCup and Team Edinferno, the aim is to deliver a beginning of an ad hoc agent architecture in a grounded, logical organizational system for the team to use in the competition in the future.

Robot Soccer Agent Organizations

Related work on agent organization MAS specifically for robot soccer can be found in [33] for both simulated (via RoboCup's official simulator SoccerServer) and real playing fields for small and medium sized robots. Three levels of behaviours are proposed: functional, relational and organizational, where the organizational behaviours define for example group formation behaviours or interactions. In [23], RoboCup's simulation league is taken as a use case to compare several organizational models, including Agent/Group/Role [39] and OMNI (which is an extension of OperA). The organizational aspects (roles, norms, coordination) of robot soccer are the focus of this review as the models are compared on their modeling options on four 'dimensions': structural, dialogical, functional and normative. OMNI is one of the highest scoring methods.
The use of the OperA methodology specifically is new to the RoboCup domain. Ontologies have been applied for object recognition and categoriztaion, also in soccer robots [63, 72]. Roles have been used in multiple (robot/human/simulation) soccer applications (section 2.1.1), as have behaviour libraries [14]. OperA is novel in this field as it provides the means to design the complete human - robot system of a soccer match from an organizational point of view, including norms and violations to regulate agent behaviour. The main advantage of this framework is that it is founded in logic, yielding a consistent, grounded model that can be implemented with several logic-based agent programming languages or otherwise. Also, this framework is extendable to include specific agent designs.

1.3.2 Research Questions

Now that the fields of research have been introduced, we reach the following research questions:

- How to model an abstract agent organizations framework for the human-robot soccer domain using OperA architecture?
- How to do plan recognition on robot soccer players from visual and numerical information only?
- How to connect the framework and plan recognition via the ad hoc agent or coach?

Chapter 2 will introduce relevant work on abstract agent frameworks and Multi-Agent Systems. The actual Robot Soccer Society framework will be elaborately presented in chapter 3. Through the agent roles in the framework, the connection to the current state of affairs of Edinferno's RoboCup team will be possible. The second question of extending that methodology with a plan recognition module to be applied to the also novel coach role, will be presented and discussed in a related research chapter on plan recognition methods (4), followed by the proof-of-concept module developed for this work 5. In chapter 6, suggestions on how to integrate the framework and the plan recognition module will be given. Conclusions and remarks for future research can be found in that same chapter.

Chapter 2

Related Research - Agent Organizations

The first research question of this work considers the use of an ‘abstract framework’ to model robot soccer. In this chapter, an overview of the relevant components of such a framework is given.

2.1 Agent Societies or Organizations

The concepts of *agent societies* as well as *agent organizations* are based on human phenomena and research in sociology and psychology, but can be and have been linked to agent systems [1]. The term *society* can be used to describe groups of entities (i.e. humans, animals, robots) that coexist in an environment and aim to achieve goals in cooperation or other coordinated behaviour patterns [31, 70]. An *organization* is a set of entities, regulated by social order and in which agents are meant to achieve common goals [31]. The difference lies in the focus: in societies, the focus lies on the social interaction of the *agents*, while an agent organization focusses on the structure of the *system*. *Coordination mechanisms* are an important part of the notion of the organization-oriented Multi-Agent Systems (MASs). Take for example the ‘mutual adjustment’ mechanism of agents in an ad hoc setting (‘adhocracy’) in which the decision-making process is decentralized (handled by special ‘managing’ agents) and the execution of tasks depends on agent *negotiation* and adjustment of their own plans [1, 4, 31, 69]. For an elaborate overview of coordination strategies in MASs, see [1] (chapters 2 and 3) and Weigand et al. [101]. In the latter, the idea of coordination through communication is analyzed, based on organization theory and human coordination mechanisms. They argue that when communication is used for mutual understanding it is a coordination mechanism, and that interdependence of agents requires communication in order to coordinate their actions. These statements also occur in work on animal coordination [67], where experiments on strategic animal behaviour showed that ‘acting in a coordinated manner required communication’.

There are multiple methods for developing a MAS. In [33], Drogoul and Collinot present a generic design method (‘Cassiopeia’) which is conveniently applied to robot soccer as a use case. Both the dynamics of the game and the ‘unpredictable’[1] actions of the opponents make robot soccer a challenge for the design of a multi-agent system for it. Other examples of agent organizations and how to model them can be found in [31, 36, 70, 105] and [38]. Esteva et al. [36] researched *electronic institutions* using the concepts of *norms* and institutions (laws) for the design of robust open agent organizations, Odell et al. [70] examine the notion of *roles* and social structures. One of the first works on roles and groups and how they interact would be the Agent/Group/Role (AGR) model by Ferber and Gutknecht [38, 39]. Wooldridge’s Gaia model defines roles with responsibilities, permissions and protocols and also defines protocols for inter-role interactions [1, 105]. Gaia handles both the societal or *macro* level and the agent or *micro* level aspects of MAS design. However, Gaia is not suited for open domains and the organizational aspects of the society are only implicitly defined within roles and interaction models. For further review of organization-oriented MAS methods and models, see [4, 23].

2.1.1 Roles

A *role* in the context of agent societies is comparable to the role a character in a book or film can play: in fact, the term stems from theater analogies [70]. It is an abstract representation of the *function* of the agent that ‘plays’ or *enacts* the role, consisting of the behaviour that he can or should perform and the goals that he should achieve if he adopts that role [25]. Roles can be seen as simpler units in a complex system that represent parts of (organization) objectives [31] or smaller parts of team behaviour [41]. This idea is based on the notion of ‘bounded rationality’, presented by Simon in his research on human rationality and decision making [86]. The idea is that individual agents might have limited ability to acces information, for example they only can see the world from their own position, while division of labor over multiple agents with different functions (e.g. a team) would achieve a task very efficiently [101]. The *global objectives* of a system can be decomposed [91] into *role objectives* which can be further specified into *sub-objectives* that describe the intermediate states in achieving that corresponding objective. In a system like robot soccer, there are certain *rules* or regulations that the players should follow, for example what each player should do in a kick-off situation or in what situations they would get a penalty. Such rules also have to be accounted for in the roles that can be enacted.

Besides this abstract, organization-oriented use of roles, the concept has also been applied in several fields of soccer research as a way to identify the different players and their tactics. In those works, the role a player has is based on either his absolute position on the field [64], his position relative to the ball [10, 11, 60] or trajectories of the player on the field, including relative position to the ball and/or other players over time [98, 107]. This role assignment can either be predefined by the player’s position at the beginning of a game (in which case an initial formation is devised, for instance by a coach, and the players stay in their role for the entire match), switch dynamically depending on those relative positions, or switch according to a structure of ’role evolutions’ or a fixed sequence according to which roles should be played [14]. Another approach could be to let the formation be adjusted by self-organization algorithms to a more optimal

[1]See chapter 4 for researches in action prediction.

formation [70]. Yoshimura et al. [106] specify role-dependent, strategic behaviour for both agent and team coordination dynamically, based on current states of the world (beliefs of the agents).

The dynamic role switching system by Weigel et al. [102] is based on an utility system: each player constantly calculates its utility to enact a certain role and communicates that utility to its teammates, after which they all compare results and decide which roles should be played by whom to yield the highest team utility. This is similar to the approaches by Genter et al. [41, 42], in which agents have a certain ability to perform each role and roles have different values for the team at a certain moment. Based on the team value, an agent chooses the role that adds the most value to team performance, while still fitting its own abilities. Other dynamic role switching techniques can be found in [60, 91, 97].

The notion of agents adapting their own role to the roles of other agents, for example their teammates, introduces *role dependencies*, leading to coordinated interaction to achieve goals together.

2.1.2 Interaction and Coordination

Interaction is a term used for many kinds of activities; the main requirement for interaction is the involvement of multiple agents. The most basic form of interaction in a multi-robot setting like soccer is avoiding collisions with the other agents. In case of unknown agents in a partially observable environment, fields such as plan recognition, opponent modelling and intent inference come into play (chapter 4). *Coordination* is basically structured or organized interaction between agents, usually in cooperation.

When a system is designed with a fixed or implied set of conventions, protocols, strategies and plans, for each member to adhere to, and each member knows or assumes the other members will adhere to the same structure, we say that the systems works with a 'locker room agreement' [91]. In small strategic games and toy examples, the locker room agreement can be dropped, but in real-world applications this would not be realistic [14]. Also, even when it may look like teamwork for an observer, the behaviour of robot soccer agents might technically merely be the performance of separate individual tactic behaviours [102]. The ad hoc setting is more realistic than a locker room setting, but difficult to solve because of its complex dynamic character (1.2.2). However, higher-level coordination structures in combination with flexible methods of learning on the agent level might be a promising approach to ad hoc teamwork.

2.2 Knowledge representation

For agents to operate and interact in an environment and with other agents, a means of representing their knowledge of that environment is needed. This is known as *knowledge representation* and it can be approached from many fields of research (logic, agent theory, robotics, philosophy). We will give a short overview of the parts relevant to multi-robot soccer.

2.2.1 Symbol grounding

Representing knowledge about its environment, an agent needs to be able to *ground* the things he perceives in an abstract, symbolic way in order to *reason* with them. The problem of *symbol grounding* as posed by Harnad is a recurrent challenge in both philosophy of AI and actual agent or robotic systems. The grounding of a symbol is to connect 'reality' (internal states) to sensorimotor activities of an agent, or to link a symbol or name to some activity of object in the environment [47]. He proposed three stages of grounding, resulting in symbols: iconization (representing 'analogue signals' or percepts), discrimination (distinguish different signals) and identification (assign a (class) name to them) [47, 100]. In robotics research, symbol grounding can be done with object categorization via pattern recognition and semantic libraries or ontologies to match observed patterns to known ones [51]. A way to determine the necessary and sufficient conditions for something to belong to a certain class is using numerical thresholds. In their work, Mendoza and Williams for example used ontologies for object recognition in AIBO robots playing soccer [63].

2.2.2 Ontologies

In addition to low-level feature and object recognition techniques, there is a need for a way to connect this perceptual level to a more symbolic level on which reasoning can be done. An *ontology* is a well-known 'tool' to help bridge this semantic gap and in doing that, grounding a robot's sensory information. From a philosophical point of view, ontology is the study of *what there is*[2]. This not only encompasses the things that exist but also what the most general features and relations of these things are. According to [45], ontologies are 'formal descriptions of entities and their properties, relationships, constraints and behaviours'. They are also seen as 'explicit specification of a conceptualization' [44], in which a *conceptualization* is an abstract view of the world (or a part of the world) that we want to represent. An ontology can for instance be viewed as a hierarchical structure of concepts (or categories, classes) with their properties (or relationships), for example depicted as a graph, tree or 'semantic network', with 'is-a' or *subsumption* relations between nodes. One category 'subsumes' another if the latter is a subset of the former [80] (also known as the hyponymy relation in linguistics [84]). Ontologies can be used to represent information, which make them very suitable for automated information processing or communication between agents in a multi-agent system - even sharing actions and intentions [51].

Two main kinds of ontologies can be found in the literature: frame-based and semantic network-based [68, 72]. Ontologies as described in this section are like semantic networks. Frames describe entities as a list of slots that can be filled in with values denoting the properties of that entity. In the development of an ontology, the concepts that we want to describe need to be categorized. Categorization is the partitioning of concepts or objects into useful groups or categories [90]. In this process a trade-off has to be made between expressivity and complexity: the domain that needs to be represented must be described, but the level of detail should only be as deep as necessary [68].

[2] http://plato.stanford.edu/entries/logic-ontology/

Besides formalizing domain concepts from scratch in some logic representation language, there are also ontology development tools and several logic-based markup languages that can be used to develop an ontology. Examples of such tools are Ontolingua[3], Chimaera[4] and Protégé[5]. Markup languages like OWL (Web Ontology Language)[6] or DAML (DARPA Markup Language)[7] can be used to encode the concepts in the ontology in a formal way, based on description or first order logic.

2.2.3 Logic

Formal logic languages are a very expressive and powerful means for describing concepts and their relations, which make them well-suited for developing ontologies. The two common logics used in ontologies are Description Logics (DL) and First Order Logics (FOL). DL is related to FOL, but consisting only of its decidable fragments [48]. Decidability refers to the decision problem of finding a method to determine set membership.

Besides a formal language for developing ontologies, a language to represent the actual multi-agent system in should be defined. That is, the environment, the agents, possible actions, situations and conditions need to be represented in order to reason with and about it. An example that will return in our framework in chapter 3 is using CTL* (based on propositional logic), possibly extended with STIT logics as done by Wooldridge [104]. CTL* (CTL = Computational Tree Logic) is a branching time logic, meaning that formulae are interpreted over a tree-like structure which represents all possible ways the system could evolve. A path through such a tree is a history or course of events, nodes represent system states and arcs the actions of an agent. STIT is an abbreviation for 'sees to it that' and provides the means to relate CTL* to the actual agents and situations: with STIT expressions, specific agents are made responsible for ensuring a certain state of affairs. The notion of an specific agent 'seeing to it that' something becomes true is an intuitive way of grounding abstract properties, for example the desire to achieve some goal (more in chapter 3).

Where Wooldridge's paper considers single-agent systems, Dignum [31] applies an extension of this CTL* + STIT with Deontic logic to her framework for multi-agent systems. Deontic logic is the logic of norms and rights, allowing agents in a system to choose whether or not to adhere to regulations rather than forcing them with system constraints [50]. Deontic logic reasons about ideal versus actual states, which makes it attractive for application in social organizations, simulating the norms that regulate human societies.

2.3 Languages

For agents to coordinate their actions, some way of communication is needed. In an agent society with heterogeneous agents, communication can provide the means to ensure interaction between

[3]http://www.ksl.stanford.edu/software/ontolingua/
[4]http://www.ksl.stanford.edu/software/chimaera/
[5]http://protege.stanford.edu/overview/
[6]http://www.w3.org/TR/owl-guide/
[7]http://www.daml.org/

them [92]. Inspired by human organizations and natural language, certain abstract languages on different levels of communication have been proposed and developed for usage in multi-agent systems.

2.3.1 Knowledge Representation Language

Knowledge representation languages are the means to express statements concerning concepts from ontologies or knowledge bases combined with internal behaviour of agents. They can also be called 'content languages' [31]. Different languages are developed, starting from different logical bases. We will only mention a few here since there is a wide range of such agent languages. ALICA, the language used in [72], is based on description logics. Readylog, a variant of Golog, is based on Reiter's *situation calculus* [58]. Situation calculus uses descriptions of properties of a state and conditions for reaching successive states [35]. Dylla's Readylog programmes are directly executable on soccer robots. An example of a content language based on first order predicate logic is KIF (Knowledge Interchange Format). It also supports *nonmonotonic reasoning*, allowing for the addition of new information (change) to a model without making existing inference rules inconsistent. Nonmonotonic or abductive reasoning uses the notion of deducting most likely explanations instead of classical consequences, by revision of information and consequences as new information enters a model [31, 75].

2.3.2 Agent Communication Languages

Besides a language to represent knowledge content, the format in which messages are expressed should also be shared. Agent Communication Languages (ACL's) extend knowledge representation formalisms with communication primitives, but can also be used to specify coordination strategies [31]. ACL's like KQML (Knowledge Query and Manipulation Language) and FIPA-ACL[8] are examples of ACL's based on Speech Act Theory: a philosophical theory that interprets utterances of human language as actions like requests, commitments and replies. The idea is that in stating a sentence, an action is performed as well [85]. KQML is a declarative language on the level of knowledge communication, representing message contents using ontologies to define speech domains [24]. FIPA-ACL contains a structured Communicative Act Library and a semantic characterization of those acts [40].

2.4 OperA

In the next chapter the abstract framework for our robot soccer society will be presented. The methodology used for its development is that of Dignum's OperA, Organizations per Agents, which provides a formal model for organizational interaction for multi-agent systems [31]. The reasons for chosing this framework rather than one of the other MAS models discussed in this chapter are the following.

[8]FIPA is the Foundation for Intelligent Physical Agents (`http://www.fipa.org/`)

In general, multi-agent system and the agent paradigm are a suitable field of research for robotic soccer as the robotic soccer setting consists of multiple autonomous entities situated in a shared environment, with tasks to perform that require coordinated behaviour and interactions. As (robot/human) soccer is a domain functioning on rules and regulations combined with heterogeneity of the agents, its description should reflect such organizational characteristics and structures. Regulations can be captured by deontic logic and the heterogeneity of agents can be represented in terms of roles.

OperA is a general framework design methodology with two main requirements: the *collaboration autonomy requirement* and the *internal autonomoy requirement*. The former states that activity and interaction in a society must be specified without completely fixing interaction deterministically. General 'scripts' are designed that can be adjusted and instantiated to the specific needs of each interaction moment. The latter requirement states that interactions and structure of the model should be represented independently from the internal design of the participating agents: the framework is developed from an organizational point of view and in principle any kind of agents should be able to participate. This corresponds to the idea of *open societies*, in which autonomous individuals, each with limited resources and knowledge, inhabit a shared environment and collaborate in or with that environment. In the context of roles, an open society requires roles to be *separate* from the actual agents that can enact them, which makes role dynamics an important part of the design of such societies [25]. Meeting OperA's requirements allows for extensions to the model, flexibility for the agents and reusability of structures within or between similar societies [31].

Since our application in the robot soccer domain yields a system in which humans ánd robots inhabit and collaborate in the same environment, performing a dynamic, human-inspired game which is highly structured and coordinated, OperA's organizational but flexible framework is an appropriate choice for this application. It will yield formal, grounded and structured coordination interactions for the RoboCup Standard Platform League society in general, with descriptions of roles, norms, game situations, violations and ways to apply them to specific agents and soccer matches.

Chapter 3

Robot Soccer Society Framework

This chapter contains the organization framework for our robot soccer society. Following the development steps from the OperA methodology, the three organizational layers (Organizational Model, Social Model and Interaction Model) will be specified. As the entire game of soccer would be too complex to describe completely, and moreover, since it is dynamic in such a way that there may exist situations or successions of situations that cannot be predefined in a formal structure, we will work with the most likely situations and generalizations. However, since soccer is a structured game, with rules and referee decisions to adhere to and coordinated strategies, for example based on relative field positions of its players, it can be formalized up to a certain level while still allowing dynamic situations.
The definitions and examples in this chapter are based on the 2014 rules for the RoboCup Standard Platform League, in which requirements and forbidden actions and their sanctions are described. The SPL rules are inspired by regulations and situations in human soccer, which conveniently adds to our aim of using knowledge from human soccer games in improving robot soccer team performance.

3.1 Logic for Contract Representation

Before we start with the specification of the OperA architecture for our robot soccer society, its language and notation should be defined. The language used in the framework is OperA's Logic for Contract Representation (LCR). This logic is an extension of BTLcont [30] which is an extension of CTL* ([31], p. 102). CTL* is a branching time logic based on classical propositional logic. This branching time logic is in OperA further extended with the STIT-operator $E_a\varphi$, meaning that 'agent a sees to it that φ' ([31], p. 102) and its *achieved* form $D_a\varphi$, 'agent a saw to it that φ'. The last part of the extension is the addition of deontic expressions to indicate what should or should not happen (obligations, prohibitions, permissions) and also what will happen if that does not happen (violations, sanctions). Deontic logic allows for reasoning about ideal states versus actual states of behaviour [50]: it adds to the autonomy of agents in a system by stating what would be ideal and guiding them (back) towards those ideal states, while still allowing other choices. In the sections below, we use both a semi-formal and formal notation for norms, roles and interaction scenes. In order to verify our model, the formal notation is needed. More on verification can be found in chapter 3.5.

The syntax of LCR contains the classical proposition connectives $\vee$ ('or') and $\neg$ ('not'), $\wedge$ ('and'), $\rightarrow$ (logical implication) and $\leftrightarrow$ (logical equivalence). It also contains the constants *true*, *false* and the CTL* operators: A = always in the future (inevitable, on every path); S = since; X = next state; Y = previous state; U = until; $\leq$ = before; and the STIT operators E (sees to it that) and D (saw to it that) (see section 3.1.2). For a complete formal definition of LCR's syntax and semantics we refer to Chapter 4 of [31].

3.1.1 Deontic expressions

Norms make up a large part of the framework. Norms are deontic expressions, describing the commitments agents make to each other, but also the rules that they should adhere to on a more global level: they regulate the behaviour of agents in a system. Note that we are only concerned with the external behaviour of agents, since we do not include specifics for agent implementation and their internal states. A distinction can be made between *role norms*, *scene norms* and *transition norms*. LCR is used to model deontic expressions as follows: given an expression $\varphi \in \mathtt{L}_D$ and a role-enacting agent $i \in Reas_D$ (where Reas$_D$ is the set of role-enacting agents in domain D; see also 3.3) , $O_i\varphi, F_i\varphi, P_i\varphi \in \mathtt{L}_D$ are deontic expressions meaning 'agent i is *Obliged*, *Forbidden/prohibited* or *Permitted* to bring about φ'.

Obligation is defined as an expectation for agent a to bring about a certain result (or state of affairs) ρ before a certain condition (or deadline) δ has occurred:
$O_a(\rho \leq \delta) =_{def} A((\neg\delta \wedge viol(a,\rho,\delta))U((E_a\rho \wedge X(A\Box\neg viol(a,\rho,\delta))) \vee X(\delta \wedge viol(a,\rho,\delta))))$,
where A= inevitable, U=Until, X=in the next state and $\Box\varphi =_{def} \neg(trueU\neg\varphi)$: 'always'.

Permission (P) and prohibition (F) are defined as abbreviations of obligation (O):

1. $P_i\varphi =_{def} \neg O_i\neg\varphi$
2. $F_i\varphi =_{def} O_i\neg\varphi$

Here, permission has the 'weak' annotation; that is, permission to do something means that there is no obligation to not do that something: it can be done but not necessarily has to be done.

The set of all deontic expressions $Deon_D$ is given as:

1. $\forall\varphi \in L_D, OPF_i\varphi \in Deon_D$
2. if $\alpha \in Deon_D$ then also $OPF_i\alpha \in Deon_D$, where $OPF \in \{O, P, F\}$

A deontic expression or *norm* can be built like shown below, where `form` is a formula (e.g. φ) in the domain language:

```
 <Norm> ::= OBLIGED(<id>, <Norm-form>) | PERMITTED(<id>, <Norm-form>)
| FORBIDDEN(<id>, <Norm-form>) | IF <Achieved-form> THEN <Norm>
<Norm-form> ::= <Form> | <Form> BEFORE <Form>
<Achieved-form> ::= DONE(<id>, <Form>)
```

3.1.2 Achievement expressions

Achievement expressions are logical sentences using the STIT-operators E and D to represent the *results* of abstract actions. The set Act_D of all achievement expressions given a domain language $\mathtt{L}_D$ (3.1.3) is the smallest set of STIT-expressions: $\forall\varphi \in \mathtt{L}_D$, $E_i\varphi \in Act_D$.
Achievement expressions are: $E_i\varphi$ and $D_i\varphi$ given expression φ in $\mathtt{L}_D$ and i $\subseteq$ Reas; or $E_r\varphi$ and $D_r\varphi$ where $r \in Roles$ indicate an achievement for any `rea` of that role r.

1. $E_r\varphi \rightarrow \exists i \in Reas$: `rea`(i,r,s) $\wedge E_i\varphi$
2. $D_r\varphi \rightarrow \exists i \in Reas$: `rea`(i,r,s) $\wedge D_i\varphi$

Achievement expressions can also be used with the notion of *deadlines*: before a certain expression holds, the agent should have seen to it that something has happened. However, in our society, we are not concerned with deadlines as in other (human) organizations, but merely with a partial ordering on achievements. For example, in the Conference Society given in [31] it is necessary for a paper reviewer to have reviewed papers before a certain deadline, as in a fixed point in time. In robot soccer, achievements can depend relatively: the robots are only allowed to start playing after the playing signal has been given. So in our model, δ can be substituted for another achievement expression, e.g. $D_i\varphi \leq D_i\psi$. We give the achievement axioms including deadlines δ with this in mind:

1. $\models E_i\varphi \rightarrow XD_i\varphi$
2. $\models D_i\varphi \rightarrow \varphi$
3. $\models (D_i\varphi \leq \delta) \wedge (D_i\psi \leq \delta) \leftrightarrow (D_i\varphi \wedge D_i\psi) \leq \delta$
4. $\models (D_i\varphi \leq \delta) \vee (D_i\psi \leq \delta) \leftrightarrow (D_i\varphi \vee D_i\psi) \leq \delta$
5. $\models \neg(D_i\varphi \leq \delta) \rightarrow \neg D_i\varphi \leq \delta$

These expressions are the same as in [31] (chapter 5) but with a new connotation for deadlines δ.

3.1.3 Domain Language

The general definition of a domain language mentioned in the previous sections is given here. Signature Σ, the set of first order formulas that form the domain language, consists of

$< Pred_D, Func_D, Id_D >$: predicates, functions and identifiers (constants) for the domain of robot soccer. They will be gradually introduced in this chapter. Most of them will be clear from context, others will be explained in more detail. The same holds for the set of variables and the terms that can be built from Σ.

Terms are defined as $\forall i \in Id_D, i \in \mathtt{Term}_D, \forall x \in \mathtt{Var}_D, x \in \mathtt{Term}_D$. Also, $\forall t_1, ..., t_n \in \mathtt{Term}_D, \forall f \in \mathtt{Func}_D, f(t_1, ..., t_n) \in \mathtt{Term}_D$. As we specify the language for just our robot soccer society domain, the subscript D is redundant and will be omitted from here. `L` is defined as follows:

- if $p \in \mathtt{Pred}$, of arity n, and $t_1, ..., t_n \in \mathtt{Term}$; then $p(t_1, ..., t_n) \in \mathtt{L}$
- if $t_1, t_2, \in \mathtt{Term}$, then $t_1 = t_2 \in \mathtt{L}$
- if $\varphi \in \mathtt{L}$, then $\neg\varphi \in \mathtt{L}$
- if $\varphi, \psi \in \mathtt{L}$, then $\varphi \wedge \psi \in \mathtt{L}$
- if $\varphi \in \mathtt{L}$, then $\forall x(\varphi) \in \mathtt{L}$

Formulas of the form $p(t_1, ..., t_n)$ are *atoms* (in the set `Atom`). Variables can be free or bound - we will use capitalized names for free variables and lowercase names for bound variables. When referring to an unspecified member of a set, we use that set as a free variable (e.g. any $p \in Players$ can bind the free variable 'Players').

3.1.4 Illocutionary LCR

Next up is a short introduction to *Illocutionary LCR*; its explanation and application will be given in 3.2.2. Illocutionary LCR is an extension of LCR, with the addition of Communicative Acts to represent interaction between agents by means of their communication.

If φ, ψ are expressions in LCR and i,j agents $\in$ Ags, then $inform(i, j, \varphi)$ and $request(i, j, \varphi)$ are formulas of ILCR. Furthermore, $\neg\varphi, \varphi\wedge\psi, E_i\varphi, A\varphi, \varphi U\psi, \varphi \leq \psi, X\varphi, viol(i, \varphi, \psi), inform(i, j, \varphi)$ and $request(i, j, \varphi)$ are formulas of ILCR, using the standard LCR operators.

Now that all formal introductions have been given, the model for our robot soccer society can be developed. On some points, this model will vary from the OperA framework due to domain specifics. Design choices and examples will be illustrated in depth.

3.2 Organizational Model (OM)

The first part of designing an agent society using the OperA methodology is defining the highest organizational level: the Organizational Model (OM). The OM is an example of how coordination in a Multi-Agent System can be modelled via social interaction and dependence. On this level, the aims or *objectives* of the society and the means needed to achieve those objectives are captured. The OM is the most elaborate layer of the design, in which the characteristics of our robot soccer society will be given in four parts or *structures*: social, interaction, normative and communicative. The social structure consists of the specification of society objectives and roles, groups, dependencies and the coordination type of the society. In the interaction structure, *scenes* representing interaction moments will be described for the tasks that require coordination. The normative structure gives norms for the roles and interactions defined in the first two structures, and the communicative structure consists of *ontologies* and the communication language used in the society.

In the design of our framework, we maintain the terminology and methodology as shown in Dignum's 'Chapter 6: Designing Agent Societies' [31]. This means we have divided the OM into a Coordination level, defining the coordination type of our society, an Environment level, in which the social structure will be specified in terms of roles and a domain ontology. Also, the normative and communicative structures will be introduced on this level. The last level of the OM is the Behaviour level, describing the interaction structure of the society.

3.2.1 OM: Coordination Level

Coordination type and facilitation roles

In the first step of building the OM, the *coordination type* of our robot soccer society is determined. The coordination type determines what kind of relationships and dependencies exist between the enactors of roles. The types to choose from in the OperA methodology are *hierarchy, market* and *network*, each with their characteristic properties. Alternatively, a combination or adjusted version of those types can be defined to fit a specific application. In short, a market is an open society based on self interest of the agents, a hierarchy is a closed society based on controlled dependencies between agents, and a network society works on trust and collaboration through mutual interest.
A soccer team should play in collaboration, with what we could call 'team spirit' [33]. This corresponds with the characteristic of *mutual interest*: all players in the team, including their coach and human teammembers, share the interest of winning matches and trust that all teammates work towards that aim. A soccer team would be best described as network coordination type. However, we also include the referees in our society in order to help structure its coordination. Referees in the RoboCup games have different status than participants (robots or humans): they should be obeyed at all times and have the final say in whatever kind of dispute that might occur. This corresponds more to the hierarchy type of coordination. Combining multiple types of coordination is likely to be the best choice for many kinds of societies ([31], p171); this is also our choice for the robot soccer society.

Let's call our combination of mostly network, partly hierarchy coordination the *'Robot Soccer Coordination Type'* or RSCT. In relation to this coordination type several *institutional* or *facilitation* roles can be defined. Facilitation roles, in contrast to *external* or *operational* roles (3.2.3), have fixed actors and are designed to enforce the social behaviour of the other society agents and global society activity. Actors of facilitation roles are typically mutually trusted, impartial agents. Operational roles can be enacted by (almost; see 3.2.2) any agent and they describe the domain related objectives of the society. The facilitation roles characteristic for the coordination type are given concisely in table 3.1; they will be further described in section 3.2.3 and Appendix B. Specifics of the dependency relations within the RSCT will be discussed in section 3.2.3.

Role	**Objectives**	**Abstract Norms**
Head-Referee	decide about violations and penalties decide about requests (keep shoot-out time)	obliged to inform the society about decisions allow Assistant-Referees to enter the field
Assistant-Referee	apply decisions of the H.-Referee	obliged to apply
GameController-operator	manage clock inform Robots of RobotStates	obliged to keep the time of the game, time-outs etc.

Table 3.1: Facilitation roles for the Robot Soccer Coordination Type.

3.2.2 OM: Environment Level

We continue with the specification of the social structure. In the Environment Level, the global requirements, *roles* and a *domain ontology* are described, based on the relation between the system and its environment; meaning the (expected) functionality of the robot soccer system.

Domain Ontology

In a domain ontology, domain concepts are formalized as formulas in a knowledge representation language, for example in First-Order Logic. In the domain of robot soccer, we have to define physical environment concepts such as the ball and parts of the field, but we also need representations for valid and forbidden actions. Following the example of Stephan Opfer's work on formalization of RoboCup's Middle-Sized League, we used Stanford's ontology development tool 'Protégé' [1] in combination with OWL (Web Ontology Language) and the Hermit reasoner to define a taxonomy of domain concepts or categories. In the concept graph in Appendix A, a part of this taxonomy can be found. Specification of some of the concepts (i.e. actions, violations) is omitted due to the size and complexity the graph would have otherwise: they will be further refined in the *Norm Library* (appendix D).

In order to describe the objects in the domain of our SPL robot soccer society, we mainly need to define a 'isPartOf' function, which is a mapping from elements to sets (note that the element y can itself also be a set): $\forall x, y : isPartOf(y, x) \rightarrow Set(x)$. Also, this relation is *transitive*: $\forall x, y, z : isPartOf(y, x) \wedge isPartOf(z, y) \rightarrow isPartOf(z, x)$. In this manner, the domain concepts of the field environment are defined as follows:

Constants: ball, field, OppArea = {oppHalf, oppPenaltyArea, oppGoalArea},
OwnArea={ownHalf, ownPenaltyArea, ownGoalArea}
Predicates/functions(arity): Area(1), isPartOf(2), Ball(1)
Formulas: Area(field), Ball(ball), OwnArea(ownHalf; ownPenaltyArea; ownGoalArea),

[1] http://protege.stanford.edu/

OppArea(oppHalf; oppPenaltyArea; oppGoalArea)
Let x be of type OppArea or OwnArea, such that:
$\forall x.(isPartOf(x, OppArea) \vee isPartOf(x, OwnArea)) \leftrightarrow isPartOf(x, Field)$
$\forall x.isPartOf(x, OppArea) \leftrightarrow isPartOf(x, OppHalf)$
$\forall x.isPartOf(x, OwnArea) \leftrightarrow isPartOf(x, OwnHalf)$

Informally, there is one instance of category Area which is field, one instance of Ball which is ball. OwnHalf, OwnPenaltyArea and OwnGoalArea are of the category OwnArea and similar for OppArea. OppArea and OwnArea combine the whole field (an area must be either of type OppArea or OwnArea), all instances of areas are part of Field, and instances of OppArea are part of OppHalf (similar for OwnArea). We write free variables with a capital ('Ball') and specific instances or constants in lowercase ('ball'), such that 'Ball(ball)' means that the specific instance called 'ball' *is an* instance of the category 'Ball'.

Besides a domain ontology, there are three more ontologies to be used in an OperA framework. Those can be reused, directly or after small adjustments, to fit the design of our robot soccer society framework. These are the OperA level ontology (describing roles, dependency, interaction scripts on a conceptual level), Model level ontology (describing concepts of coordination types) and the Communication ontology (describing the illocutions to be used, e.g. 'inform', 'request'). The latter two can be adjusted as to include only the chosen coordination type (3.2.1) and illocutions (3.2.2). The Model level ontology has been adapted to this specific system by not using the standard network facilitation roles of Gatekeeper, Notary and Monitor but representing them as Head-Referee, Assistant-Referee and GameController-operator as shown in table 3.1. They will be further specified in the role tables in the role section 3.2.3 and Appendix B.

OperA identifiers

Identifiers are in fact 'names' in the domain language which can be used to refer to the different entities defined in our model (for a more formal explanation please see page 117 of [31]). As a part of the framework identifiers Id, the set of Agents, $Agents \subseteq Id$, can be defined: $Agents = \{a_1, ...a_{11}\}$, which is divided in a set of $Robots = \{b_1, ..., b_6\}$ and $Humans = \{h_1, ..., h_5\}$, $Robots \cap Humans = \varnothing$. We assume here 5 role-enacting human agents; however it can be possible to have more than one instance of the 'human teammember' role. We take the set of robots to be only own 'teammembers': the opponent's robots are not included in this set. We refer to any opponent agent as simply 'Opponent', since we will not need any more detail of the opponent team in modelling the society from our own team's point of view at this level. This framework can be extended to include Opponent models when needed. The set of Robots consists of a subset of $Players = \{p_1, p_2, p_3, p_4, p_5\}$ of which 4 out of 5 are $FieldPlayers(FP) = \{p_2, p_3, p_4, p_5\}$. Here, we assume that b_1 is playing the goalkeeper role and b_6 the coach role[2]; specific formations of defender roles and attacker roles determine the number of field players playing those respective roles. The other identifiers to be defined are $Roles \subseteq Id$ and $Scenes \subseteq Id$.

[2]This corresponds to the jersey numbers of the team according to the RoboCup rules; however, $b_1 ....b_6$ are merely variables and do not refer to specific robots: we just take this representation to avoid confusion, for those familiar with player number conventions.

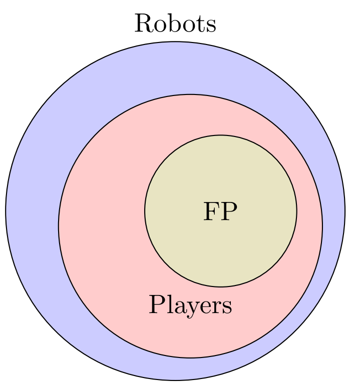


FIGURE 3.1: Subsets of Robots

Again, we distinguish between roles that can be played by human agents and those that can be played by robots. Let $Roles_H$={head-referee, assistant-referee, GameController-operator, human-teammember} and the set $Roles_R$={goalkeeper, defender, attacker, coach}, such that $Roles_H \cap Roles_R = \varnothing$. Note that the number of agents and the number of roles do not correspond: this is because there can exist multiple *instances* of players of a single role. However, it is not arbitrary which roles can have multiple instances. This will be specified in table 3.3. In our domain, *every agent should enact one role and one role only.*

Stakeholders

Besides facilitation roles, there are operational roles in the society, which are related to the different *stakeholders* and provide a link between the society and the environment. Stakeholders can be seen as the agents or entities that have some interest or goals in the functionality of the society. Stakeholder tables (table 3.2) are the first step in specifying the operational roles, loosely describing the role objectives and dependencies.

Stakeholder	**Objectives**	**Dependencies**
RoboCup Staff	ensuring quality of matches, ensuring rules are followed	Robots, Human-Teammembers ,,
Robots	score goals follow rules	other Robots, Human-Teammembers Staff, Human-Teammembers
Human-TMs	ensuring Robots function properly follow rules	Staff, Robots Staff

TABLE 3.2: Stakeholders of the robot soccer society.

In our society, the stakeholders are all dependent on each other for their objectives. This need not be the case but in this application it makes sense: the robots depend on the human teammembers to function correctly and if they don't, the humans have the objective to repair them again. Both robots and human teammembers depend on decisions of the staff. An overview of all Roles of the robot soccer society, including their relation to the society via stakeholders, is presented in table 3.3. Also, in this table, a start with defining role dependencies is made for the objectives per role. This table will be the basis for the further specification of coordinated interaction scenes (section 3.2.3).

Note in this table that the GameController-operator, even though that role is typically enacted by one of the human teammembers of a participating team, represents the stakeholder of RoboCup staff. As this agent is assigned the GC-op role, he actually ceases to be part of his team but acts like an impartial member of staff like the referees: his goals are more of global interest for all participants and staff members of that match than specifically for the team he came from. So, the *role* of GC-op is a staff role, even though the enactor of the role is no different than the

Role(instances)	**Relation to society**	**Role Objectives**	**Role Dependencies**
coach(1)	repr. stakeholder: Robots	message-tactics follow-rules	GC-operator h-ref
goalkeeper(1)	Robots	defend-goal follow-rules	FP, Opponents h-ref
defender(1≤x≤4)	Robots	help-defend-goal block-player follow-rules	Players, Opponents FP, Opponents h-ref
attacker(4-x)	Robots	score-goal help-FP follow-rules	FP, Opponents FP, Opponents h-ref
human-tm(≥1)	Human-TM	maintain-robots	Robots, h-ref, a-ref
head-referee(1)	Staff	penalty-decisions request-decisions keep-shootout-time *robot-acceptance*	Robots h-tm - Robots
assistant-referee(2)	Staff	apply-requests apply-penalties	h-ref h-ref, Robots
GC-operator(1)	Staff	magage-gameclock communicate-robotstates communicate-coach-message	- h-ref, Robots coach, Players

Table 3.3: Role table for the robot soccer society. The number of instances only considers the 'own' team; the formation of opponent teams is unknown.

enactors of the Human-TM role. We assume here that the kind of role enactment in our society is *total adoption*, meaning that the agent adopts and prioritizes all objectives and norms of his role (more in section 3.3). For the Head-Referee role, the objective 'robot-acceptance' is not really a goal that can directly be achieved, but is achieved through his other objectives 'penalty-decisions' and 'request-decisions': with these, the Head-Referee implicitly decides about the removal and return of robots on the field. The 'robot-acceptance' objective is however included in the Head-Referee's definition to emphasize the corresponence to the typical network coordination type's facilitation role of 'Gatekeeper', to which the Head-Referee is related most.

Before specifying the roles of the robot soccer society, we continue with the normative structure of the OM. As introduced in section 3.1, deontic logic forms a large part of the specification of an OperA framework: how it is used is described in the following section.

Normative structure

Norms are expressions stating the rules of the society, specifically the things a certain role-enacting agent ought (not) to do. Through norms, the behaviour of the agents can be regulated to ensure they do not violate society rules. Note here that the agents still should be able to violate rules, to represent the concept of *choice* and ensure *autonomy*, but that it has been made less attractive for them to do so. However if they do, norms also provide a means for them to return to more preferable states, and so 'repair' their violation. To specify norms, society expectations and requirements should be captured and analyzed. In the OperA methodology this is done using the Norm Analysis Method [31, 81].

This analysis method results in semi-formalized versions of concrete society norms. By means of example, a few of the norms are specified in norm analysis table 3.4. The other norms have been directly integrated in the role tables in appendix B.

Norm Analysis	
1. Description	Coach wants to send a message to the players
Responsibilities	Initiation: Coach Action: Coach, GC-op
Resources	Plan-Rec Module, Plans, Tactics, [Message-Requirements]
Triggers	Pre: Players need tactic advice Post: Players follow the advice of the coach
Norm specification	**whenever** coach-message-meets-requirements **then** GC-op **is** *obliged* **to do** inform(GC-op,Players, message)
2. Description	A-ref applies requests from H-TM, after H-ref decision
Responsibilities	Initiation: H-TM Action: H-ref, A-ref
Resources	requests (, decisions)
Triggers	Pre: H-TM requests a time-out or pick-up Post: A-ref applies the request
Norm specification	**whenever** request-granted(h-ref, h-tm, request) **then** A-ref **is** *obliged* **to do** apply-request(h-tm, request)
3. Description	privilege of goalkeeper
Responsibilities	Initiation: goalkeeper Action: goalkeeper
Triggers	Pre: goalkeeper walks towards ownPenaltyArea Post: goalkeeper is in ownPenaltyArea
Norm specification	**always** goalkeeper **is** *permitted* **to do** in(goalkeeper, ownPenaltyArea)

Table 3.4: A few of the norms in a norm analysis table.

Besides the aspects specified in the table above, for some norms *sanctions* can be defined as well. The notion of sanctions that can be imposed on an agent violating norms is represented in the robot soccer society by *penalties* that the head referee can charge an agent with. For example, if any player other than the goalkeeper were to walk into his penalty area while the goalkeeper is still in there, the 'illegal defender' rule is violated by that player, which induces a 'standard penalty removal' judgement by the head referee (by default; the h-ref decision is final and can differ from the defaults described in the rules). The conditions, as far as they were given in the RoboCup rules, on which the head referee bases his/her decisions are described in the Norm Library (appendix D). Also, formalized versions of global rules can be found there.

On this level, stakeholders and norms have been identified and the domain ontology described. The next step in the design process is the development of the precise structures of roles, interactions and dependencies.

Communicative structure

In OperA, interaction is seen as communication between agents. In order to communicate, agents need a *communication language* that they both possess and a way to represent *domain knowledge*, like a *knowledge representation language*. Knowledge representation can be based on the domain ontologies developed earlier in the model, which in our case are represented in First Order Logic. Agent Communication Languages (ACL) can be used as a wrapper or 'umbrella' language that implements the way to communicate (protocols) without taking into account the specific content or ontology (see Chapter 4 on ontologies and ACLs). Clearly, this ACL would need to be shared by all agents in the society, to ensure they have the same ways and means of communicating and are able to 'understand' each other. In an ad hoc setting, this is problematic, since the aim is the exact opposite: for agents to be able to coordinate without sharing the same language (or without communication at all). However, from an organizational point of view, communication and knowledge representation are important aspects of society modelling.

OperA's communicative structure consists of the domain language, ontology, an ACL and role illocutions. The notion of *illocutions* is based on Speech Act Theory [85], a communication theory from the field of philosophy of language, that proposes illocutions or *speech acts* that can succeed or fail instead of propositions that can be true or false. The advantage of this is that the *intention* of the speaker is included in these speech acts: an agent can for instance *inform* an other agent of something, but he can also *request* something of him, *commit* himself to doing something, *permit* or *prohibit* actions.

Usually, speech acts are used in combination with action logics, described from agent perspective ([31], p.133). As we adopt an abstract view of the externally observable effects of communication, we use achievement expressions rather than actions in our illocutions. In OperA, the Communicative Acts (CAs) *inform, request, commit* and *declare* are defined. For our model, we only use *inform* and *request.* A communicative act is $CA(s, r, \varphi)$, where s is the sender role, r is the receiver role and φ the content of the act. Throughout this chapter, such CA's can be found in the roles and scenes, for example when a norm is violated, the head-referee will inform the society of the violation and the agent that committed it, along with the resulting penalty, with $\mathbf{inform}(\textit{h-ref}, \textit{society}, \textit{decide-penalty}(\textit{Robot}, \textit{violation}(\textit{Robot}, \textit{Norm}), \textit{Penalty}))$. The possible illocutions per role are described within the role tables in appendix B.

3.2.3 OM: Behaviour Level

In the behaviour level, the definitions and tables from the previous levels are refined to construct the formal conceptual model for the OM of our robot soccer society. For readability, mostly semi-formal versions of the actual model structures are included in this chapter, following examples from Dignum's chapter 7. However, from the semi-formal notation it is only a small translation step to completely formalize, as shown in B.9. The roles, groups and dependencies, together with the coordination type specified in 3.2.1, form the *social structure* of the OM.

Roles

The analysis of role objectives results in the refinement of objectives into sub-objectives and the specification of *rights* for that role. Roles in the robot soccer domain are defined as tuples $role(r, Obj, Sbj, Rgt, Nor, tp)$ where $r \in Roles$ is the identifier of a role, $Obj \subseteq Act$ is the set of objectives of the role, $Sbj \subseteq Act$ is the set of sub-objectives sets of the role, $Rgt \subseteq Deon$ are the rights of the role and $Nor \subseteq Deon$ the norms of the role. $tp \in \{operational, institutional\}$ is the type of the role.

Furthermore, roles have the following properties (for a society S):

- $\forall R_1, R_2 \in R_S : id(R_1) = id(R_2) \leftrightarrow objectives(R_1) = objectives(R_2)$
- $\forall R \in R_S : objectives(R) \neq \varnothing$

That is, roles should have different objectives and each role should have at least one objective. A role objective is represented by $\rho = p(t_1, ...t_n)$, where $p(t_1, ...t_n)$ is a predicate in the domain language. The set of objectives for a role r is P_r. An objective γ can be described in more detail using a set of sub-objectives $\Pi\gamma = \{\gamma_1, ...\gamma_n\}$. An objective can have multiple sets of sub-objectives: they represent the various ways in which that objective can be achieved.

Role: Coach	
Role id	coach

Objectives	o1 := messaged-tactics
	o2 := followed-rules

Sub-objectives	Πo1 = ({∀p∈ Players: executed-plan-rec-module(p, role(p), t), got-plan(p, plan)), got-tactic-list(plan, formation, Tactics), decided-tactic(Tactics, tactic),got-msg(tactic, msg), message-sent(coach, GC-op, msg), wait(10s)} Πo1' =({∀p∈ Players: executed-plan-rec-module(p,t), got-role-map(plan(p), role(p))), got-formation-map(role(p), Formations), got-team-tactics(formation, TeamTactics), decided-tactic(TeamTactics, tactic), got-msg(tactic, msg), message-sent(coach, GC-op, msg), wait(10s)}
Rights	message-via-GC-op, decide-tactic(coach, (Team)Tactic)

Norms	PROHIBITED(coach, move(¬(head∧arms)))
	PROHIBITED(coach, communicate(coach, Robots, direct))
	PERMITTED(coach, have-clothes(anyColor, anyPattern))
	OBLIGED(coach, meet-msg-requirements(Msg, [Msg-Requirements]))

Type	operational

TABLE 3.5: Role definition for Coach; t = window of observation, msg = message. subobjectives o1 are assuming that the coach knows the roles and formations of all players; o1' are assuming he has to map those first, according to the plan he recognizes. Please note that these subobjectives are just conceptual, to convey what could be desirable states in order to achieve the 'messaged-tactics' state eventually. Precise plans and their implementation should be specified on agent design level.

The coach role is given in table 3.5 as an example (the other roles can be found in appendix B). Suffice it to say that the predicate called 'execute-plan-rec-module' activates the plan recognition module, after which its output is retrieved with the 'get-plan(plan)' statement. The sub-objectives in this example are meant as conceptual suggestion of how the lower-level modules and the role level could be connected.

In these role tables, some of the used predicates are defined in the domain ontology (parts of the field, the ball, penalties and violations for example), some of them should be clear just from their names (have-clothes, move) and there are several related to illocutionary acts (inform, request) or norms (obliged, permitted, prohibited), which were discussed in sections 3.2.2 and 3.2.2.

Groups

Roles can be categorized in groups if they share the same norms, in order to refer to them collectively. The formal definition of a group is as follows:

> Given society S and set of roles R_S in it, a group is a tuple $group(g, Rls, Nor)$ where $g \in Groups$ is the group identifier, $Rls \subseteq \{\rho \in Roles : \exists r \in R_S, id(r) = \rho)\}$ is the set identifiers of roles in the group and $Nor \subseteq Deon$ are the norms for the group.

As can be seen, groups do not have objectives and roles do. Furthermore, as the norms of the roles of a group should be equal, we cannot for example consider the roles Goalkeeper, Attacker and Defender to be a group in this sense, even though they are all Players and all member of the set Robots. In our model we can distinguish one group of agents with the same norms: the field players (table 3.6). Field players are the robots that play either attacker or defender but not goalkeeper or coach.

Group: Field Player	
Group id	FP
Roles	attacker, defender

Norms	PROHIBITED(FP, hold(ball, ≥0s))
	PROHIBITED(FP, move(¬(bipedal∧human-like)))
	PROHIBITED(FP, damage(field) OR leave(field))
	PROHIBITED(FP, push(Opponent))
	OBLIGED(FP, have-clothes(teamColor), teamPattern))
	IF is-in(goalkeeper, ownPenaltyArea) THEN
	    PROHIBITED(FP, is-in(ownPenaltyArea))

TABLE 3.6: Group specification for Field Player

Dependencies

Role dependencies define the relations between roles. These relations indicate between which roles and in what way objectives can be passed. Dependencies determine the interactions in the society. For example, agent A can *request* from another agent, B, that he (B) helps him with achieving his (A) objective. Dependencies between roles are based on the power relations between roles, where these power relations in their turn are determined by the coordination type of the society. These power relations determine how agents react on such requests: whether they have to commit themselves or can choose not to help.

In general, a dependency relation $r_1 \underline{\phi}_\gamma r_2$ describes that role r_1 depends on role r_2 to realize its objective γ. The relation $\underline{\phi}_\gamma \subseteq R \times R$ is reflexive and transitive, that is $\forall r_1, r_2, r_3 \in R$:
1. $r_1 \underline{\phi}_\gamma r_1$
2. $r_1 \underline{\phi}_\gamma r_2 \wedge r_2 \underline{\phi}_\gamma r_3 \rightarrow r_1 \underline{\phi}_\gamma r_3$

In our robot soccer society, we determined the coordination type to be a combination of *network* and *hierarchy*, which we called RSCT (3.2.1). The network relation is defined as $r_1 \underline{\phi}_\gamma^N r_2$, where both `rea` r_1 and r_2 can request the other for some objective γ (e.g. a state of affairs to be achieved, an action to be done). Similarly, the hierarchical relation $r_1 \underline{\phi}^H_\gamma\ r_2$ means that r_1 delegates γ to r_2 and the market relation, $r_1 \underline{\phi}^M_\gamma\ r_2$, means that `rea` r_2 can request for γ to r_1. These relations can be explained using the notions of *power, authorization* and *request.* If an agent i has power over another agent j, for example in hierarchical societies, agent j has to accept requests for a certain γ by agent i: $power(i, j, \gamma)$. Authorization relations, $auth(i, j, \gamma)$, state that i is authorized by j to do γ. Thirdly, an agent j might answer to a request from agent i without being obliged to do so, which can be seen as 'charity'. Using these notions, the dependency relations can be defined as follows:

Given $r_1, r_2 \in$ Roles, the following axioms hold:
1. $r_1 \underline{\phi}^H_\gamma\ r_2 \rightarrow power(r_1, r_2, \gamma)$
2. $r_1 \underline{\phi}^M_\gamma\ r_2 \rightarrow auth(r_2, r_1, request(r_2, r_1, \gamma))$
3. $r_1 \underline{\phi}^N_\gamma\ r_2 \rightarrow auth(r_2, r_1, request(r_2, r_1, \gamma)) \wedge auth(r_1, r_2, request(r_1, r_2, \gamma))$

In terms of our soccer society roles, we define a network dependency between the players with the roles Attacker, Defender and Goalkeeper; a hierarchy dependency between the Head-Referee and all the other roles; and a market dependency between the Head-Referee and Assistant-Referee roles. Let, for our *Robot Soccer Coordination Type R*:

$$r_1 \underline{\phi}^R_\gamma\ r_2 \rightarrow \begin{cases} auth(r_2, r_1, request(r_2, r_1, \gamma)) \wedge & \\ \quad auth(r_1, r_2, request(r_1, r_2, \gamma)) & \text{when } r_1, r_2 \in \{Attacker, Defender, Goalkeeper\} \\ power(r_1, r_2, \gamma) & \text{when } r_1 = \textit{Head-Referee} \text{ and} \\ & r_2 \in Roles \setminus \{\textit{Head-Referee}\} \\ auth(r_2, r_1, request(r_2, r_1, \gamma)) & \text{when } r_1 = \textit{Human-Teammember} \text{ and} \\ & r_2 = \textit{Head-Referee} \end{cases}$$

Players can request things from each other, the Head-Referee has power over all agents, and a Human-Teammember is allowed to request the Head-Referee to do γ; where γ can only be a request for pickup or time out [22].

Role dependencies can be depicted as in figure 3.2. In this dependency graph, the dependencies between two roles are depicted as directed arrows, labeled with the objective that determines the dependency. The source of an arrow is the role where the objective is defined and the target is the role that handles the objective. In our robot soccer society, this does not always apply: it can also be the case that the source role itself handles the objective but needs the target role to achieve it together (rather than that the target role takes over the whole objective). Consider the 'help-defend' objective between defender and goalkeeper, or the 'communicate-coach-msg' objective between GC-op and the Players. Furthermore, the objectives 'defend-goal(ownGoal)', 'block-player' and 'score-goal' are not given since they depend on the Opponents and their actions, which are not modelled here. Also, the facilitation objectives of 'keep-time' and 'manage-game-clock' are not depicted to keep the graph readable; those are mutually dependent on h-ref and GC-op.

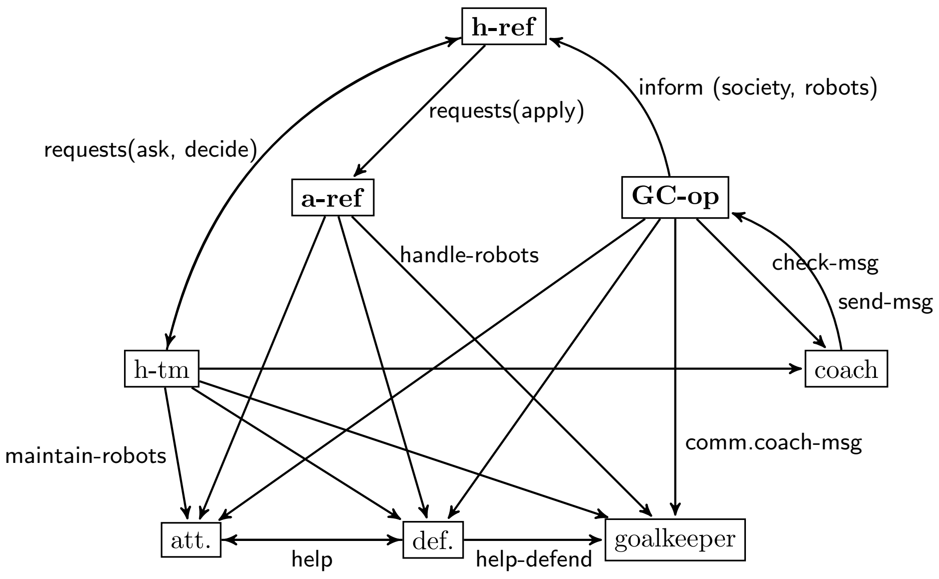


FIGURE 3.2: Role dependency graph

Interaction structure

Based on these role dependencies we can define *interaction scenes* using the role norms as guidance for how the scene should develop. The resulting interaction scene scripts can be seen as the coordination of such interactions to achieve goals together. Note that the interactions described in this work are not a complete set of all possible interactions in the dynamic domain of robot soccer: the given scenes formalize only the standard situations given in the rules and current code.

Scene scripts

Scene *scripts* describe the way an interaction scene should be performed. The scripts for the interaction scenes are defined as a tuple **scene(s, Rls, Res, Ptn, Nor)** with s $\in$ Scenes the identifier of the scene, Rls $\subseteq \{\rho \in Roles : \exists r \in R_S, id(r) = \rho\}$ the identifiers of the roles that enact the scene, Res $\subseteq$ Act the results of the scene (achievement expressions), Ptn $\subseteq$ Act the set of interaction patterns (the subachievements that make up the scene) and Nor $\subseteq$ Deon the relevant norms of the agents in the scene. Examples of interaction scene script tables can be found in 3.8, 3.9 and C.

Landmarks

The OperA framework uses the notion of *landmarks* and *landmark patterns* to represent the states in a scene. Landmarks are sets of propositions that are true in a certain state, to describe for example the state that is to be achieved. Achievement expressions as defined in the scene scripts form the landmarks of that scene. A sequence of landmarks (and by extension, the states that they represent) can be partially ordered with the LCR operators $\leq$ (before) and $\vee$ (or) which makes them a pattern. These patterns can be seen as the intermediate states to pass in order to achieve the scene result. Formally, patterns are described in terms of achievement expressions. The actions of the `reas` enacting the scene provide for *transitions* between the states in a landmark pattern. Note here that the specific actions (*how*) to achieve objectives need not be defined on this level, but rather *that* these landmarks have been reached. Actions are to be defined at agent implementation level.

First, the overview scene table for the robot soccer society model is presented (3.7). Subsequently, scene scripts and landmark patterns are given for two example scenes (other scene scripts can be found in appendix C).

Scene id	**Roles**	**Connected to**
apply-penalty	a-ref, h-ref, r	penalty-decision
apply-request	a-ref, h-ref, r	request-decision, maintain-robots
communication(CoachMsg, Players)	GC-op, Players, coach	message-tactics
penalty-decision	h-ref, r	follow-rules, apply-penalty
request-decision	h-ref, h-tm	maintain-robots, apply-request
maintain-robots	h-tm, r	request-decision
help-defend-goal	d, goalkeeper a	defend-goal, block-player
block-player	d, a, Opponent	help-defend-goal, score-goal
score-goal	a, Opponent	block-player, help(FP)
help(FP)	a, d	score-goal, block-player
defend-goal	goalkeeper, a, d	help-defend-goal, block-player
message-tactics	coach, Players, GC-op	comm.(CoachMsg, Players) follow-rules

TABLE 3.7: Overall scene table. a ∈ Attackers, d ∈ Defenders, r ∈ Robots: at least one of those roles should be enacted in the scene.

Formally, a scene S_i have the following properties:

$$\forall S_1, S_2 \in S_S : id(S_1) = id(S_2) \leftrightarrow results(S_1) = results(S_2)$$
$$\forall S \in S_S : results(S) \neq \varnothing$$

That is, different scenes have different results and a scene should have at least one result. Moreover, since any interaction occurs to achieve goals, the results of a scene should correspond to (sub-)objectives of one of the roles involved the scene.

Examplary scene scripts and landmark patterns

An example of a filled out (partially *instantiated*) interaction scene script is given in table 3.8.

The landmarks and their pattern in the penalty-decision example are the following:
$\lambda_1 : DONE(h\text{-}ref, check(h\text{-}ref, b, follow\text{-}rules(b, n)) \wedge DONE(b, \neg follow\text{-}rules(b, n1))$
$\lambda_2 : DONE(h\text{-}ref, decide\text{-}penalty(b, viol(b, n1), SRP))$
$\lambda_3 : DONE(h\text{-}ref, request(h\text{-}ref, a\text{-}ref, apply\text{-}penalty(b, SRP)))$

Interaction scene:penalty-decision	
Description	the *Pushing*-norm is violated by Robot b; h-ref decides the penalty
Roles	h-ref(1), r(1), a-ref(1)
Results	r1: DONE(h-ref, decide-penalty(h-ref, b, viol(b, pushing), SRP)) r2: DONE(h-ref, request(h-ref, a-ref, apply-penalty(b, SRP))
Patterns	{ $\forall b \in Robots; \forall n \in NormLibrary$: DONE(h-ref, check(h-ref, b, follow-rules(b,n)) $\wedge \exists n1 \in NormLibrary, n1 =' pushing'$: DONE(b, ¬follow-rules(b, n1)), BEFORE DONE(h-ref, decide-penalty(b, viol(b, n1), SRP)) } BEFORE DONE(h-ref, request(h-ref, a-ref, apply-penalty(b, SRP)))

Norms	IF decide-penalty(Robots, Violation, Penalty)
	THEN OBLIGED(h-ref,
	(inform(h-ref, society,decide-penalty(Robots, Violation, Penalty))
	$\wedge$ request(h-ref, a-ref, apply-penalty(Robots, Penalty))))

TABLE 3.8: Interaction scene script, filled out for the scene where robot r pushes another robot and is sanctioned by the head-referee for that violation. SRP = standard removal penalty (zie Special Scene:SRP)

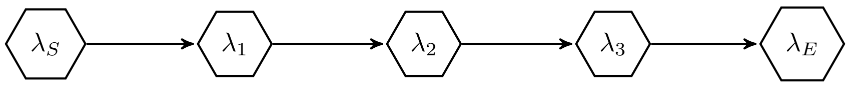


FIGURE 3.3: Landmark pattern for the penalty-decision scene: $\lambda_1 \leq \lambda_2 \leq \lambda_3$

λ_S and λ_E denote the start and end of a scene. This scene is pretty straightforward: if a robot violates a rule, he will get a penalty. More complex scenes result in more complex patterns (table 3.9, figure 3.4). The different patterns in the table and parallel paths in the figure denote the alternative ways of achieving the main result of the scene.

Interaction scene: help(FP)	
Description	For example: two attackers, one has the ball
Roles	Attackers(2)
Results	DONE(Attackers, help(Attackers, FieldPlayers))

Patterns	{ $\exists a_1, a_2 \in Attackers$:
	DONE(a_1, gain-ballPossession(a_1, ball))
	DONE(a_2, walk(a_2, supportPos))
	OR [DONE(a_2, is-near(a_2, Opponent)) AND
	DONE(a_2, block-player(a_2, Opponent))] }

Norms	All the 'Attacker'-norms and global norms apply (table B.3)

TABLE 3.9: Conceptual idea of two attackers in a coordinated 'helping'-interaction.

Landmarks:
$\lambda_1 : DONE(a_1, gain\text{-}ballPossession(a_1, ball))$
$\lambda_2 : DONE(a_2, walk(a_2, supportPos))$
$\lambda_3 : DONE(a_2, is\text{-}near(a_2, Opponent)) \vee DONE(a_2, block\text{-}player(a_2, Opponent))$

Note that this is just a conceptual example of the higher level coordination of two attackers. The idea of a first attacker dribbling the ball and a second, 'supporting' attacker assuming a free, recipient position (a relative 'supportPos' with respect to the first attacker), or blocking an opponent if one is near, is based on the current code for Edinferno's striker (attacker), as well as on ideas from [102].

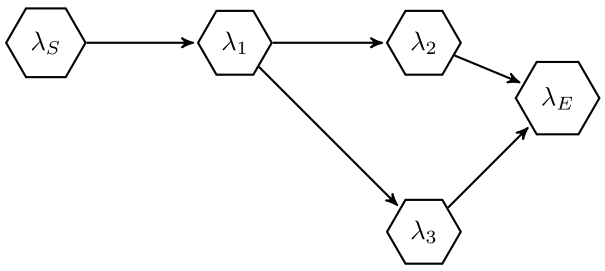


FIGURE 3.4: Landmark pattern for the 'help(FP)' scene: $\lambda_1 \leq (\lambda_2 \vee \lambda_3)$.

Norm Library, Special Scenes

In the roles of our model there are a couple of objectives that do not necessarily require interaction scenes to be achieved. These are related to *facilitation aspects* of the society and can better be described using libraries or *special scenes*. These special scenes are not related to role objectives, but are too complex to describe in terms of a single deontic expression. We specified the special scenes *StandardRemovalPenalty* and *KickOff*, as those where the ones described in most detail.

The forbidden actions and standard game situations in a soccer match however can be described as norms, which we have done as precise as possible, through analysis of the RoboCup regulation text [22]. These society norms can be found in their formal LCR translations in appendix D. We explicitely represent the forbidden actions as norms in LCR, in order to be able to use the notion of *violation* of a norm: $viol(agent, rule, (deadline))$. Consider the following example:

Locomotion
$\forall p \in Players : D_p moves(p, \neg bipedal) \vee moves(p, \neg humanlike)$
$\rightarrow O_{href} decide\text{-}penalty(p, locomotion, HrefDecision)$

The locomotion rule states that all players should move bipedal and humanlike, or if they do not, the head-referee can assign them a penalty. There is no default penalty for this violation, so 'HrefDecision' is a decision 'instance' which can be different for each specific violation event [22]. Needless to say, all roles and all scenes implicitly include the global norms from the Norm Library.

Connections, Transitions, Evolution

The last task in developing the behaviour level is to specify the order of interaction scenes and how the roles evolve throughout these scenes. For the scenes formalized in appendix C, their structures would be a straightforward diagram of the order in which the scenes occur (figure 3.5). The structures that could be defined on an organizational level are those of *penalties*, *requests* (as in the figure) and *coach communication*. Further specification of agent interactions and coordination structures depends largely on the implementation of agent plans (in the case of the player objectives (defend-goal, score-goal, help(FP), help-defend-goal, block-player)). Also, in our society, the scenes mostly occur in parallel or only when a certain situation is the case. Consider the 'follow-rules' objective that should always be happening, or the coach communication interaction structure. Furthermore, for each new occurrence of an interaction, a new scene instance should be created. These characteristics make it very hard to draw one clear interaction structure for the complete society.

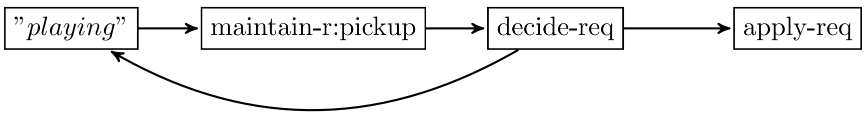


FIGURE 3.5: Interaction structure for 'requests'. After decision, the game either continues or the request is applied.

A scene *connection* is a relation $st(s_1, s_2)$ for two scenes s_1, s_2 and $st \subseteq S \times S$. This is a 1:1 relation between a source and a target scene. For example, $st(\textit{message-tactics}, \textit{communicate-coach-message})$ is a scene connection. A *transition* is a 1:M or N:1 relation between multiple source or target scenes, which can form networks of scenes. Moreover, in a connection between two scenes, *role evolution* can be determined. Role evolution describes how roles can change into other roles as a consequence of the actions in a scene. For example in OperA's Conference Society example, the 'registration' scene has an agent enacting the role of *applicant*. In the next scenes, when this applicant is registered, the same agent will now enact the role of *participant*. In the robot soccer society, role evolution or role switching can only occur between Players. Although their objectives are not specified in scenes, consider the following conceptual scene connection with role evolution: attacker a1 has possession of the ball, attacker a2 is the supporting attacker. Somehow, a1 loses the ball or passes it to a2, after which a2 is closest to the ball. This makes it necessary for a2 to play first attacker and attempt to score a goal in the next scene. How role switching is handled depends on implementation. It can be handled through a dynamic assignment system as mentioned in Chapter 2.1.1 based on for example relative (ball) position. An alternative could be to let the coach decide and assign roles to the players. Currently, role evolution in the robot soccer society does not occur as defined in OperA.

Summary OM

A lot of definitions and specifications are given in the Organizational Model. We started with determining the coordination type of our society (a combination of network and hierarchy, coined 'RSCT'; 3.2.1). A domain ontology (3.2.2) has been developed, identifiers, stakeholders, facilitation and operational roles have been specified (3.2.3, B). Norms have been captured and analyzed and role dependencies determined; this formed the social structure and normative structure of our OM. In the interaction structure, scenes, landmarks and scene transitions with role evolutions are given and discussed (3.2.3).

3.3 Social Model (SM)

Where the OM consists of the actual, formal framework that models a society, the Social Model (SM) continues from there with the explicit representation of how an agent will enact a role. On this level, the requirements, conditions and any optional internal states of the actual agents in this society can be taken into account. *Social contracts* provide the link between these agents and the general role and scene definitions from the OM.

3.3.1 Social Contracts

A social contract is an abstract description of the results and the behaviour that can be expected from role-enacting agents in the society. It allows for verification of role enactment: an agent can for example negotiate to play a role in a slightly different version to better match his own goals or abilities. For example, consider the Conference Society where a reviewer can negotiate to only review two papers instead of the five papers that he should review according to the 'reviewer'-role. In agreeing on a social contract that states how the agent enacts the role, the other agents in the society will again know what to expect from him (assuming the contracts are overt like the roles). Formally: a social contract is a tuple $SC = (a, r, CC)$ where a is an agent, $\exists a \in Agents$, $r \in Roles(S)$ is a role from society S and $CC \subseteq Deon$ is a set of *contract clauses*. These contract clauses are deontic expressions that give the conditions for a specific norm that the agent enacting the role meets. When the agent enacts the role exactly as it is given in the OM, we speak of a *trivial social contract* and no clauses need to be specified.

3.3.2 Role-enacting Agents

It is actually only here that we can properly introduce the role-enacting agent, `rea`. The term has been used throughout this chapter as 'an agent that plays a role in the society', but from here on out we use the following specification:

> Given a society S and a social contract SC = (a, r, CC): $\forall s \in Scenes(S)$ such that $r \in roles(s)$, **rea(a,r,s)** is a role-enacting agent relation, meaning that $\exists a \in Agents$ such that a enacts role r (with contract clauses $CC \subseteq Deon$) in scene s.

The difference is that now we speak of the specific agent that enacts an instance of a certain role in that instance of a scene: for example *rea(b_6, coach, communicate-coach-message)* for the scene shown in table C.5.

3.3.3 Contract instantiation

The process of forming a social contract can be described in a special kind of interaction scene script, where the result is the contract and the patterns are replaced by *plans*: landmarks describing the agreements that have been made before enacting the role. In OperA, it is assumed that these special interaction scenes to set up contracts only occur at the beginning of every interaction structure. For example, before the scene structure 'penalties' (scenes: violating a norm - decision on a penalty - application of the penalty) can be played, its `reas` and their contracts need to be specified in such a special interaction scene. This happens before the actual interaction in a separate 'start' scene. Similarly, the ending of a contract, after the interaction is played out, happens in an 'end' scene. Ending a social contracts comes naturally when all clauses have been fulfilled, but it may also occur that an agent wants to end the contract earlier or that the society wants the agents to dispose of the contract if he has failed to realize its objectives.

The SM is mostly focussed on open societies, in which agents have to be considered for participation in the society and can leave it again if allowed by their contract clauses or the agents enacting facilitation roles. However, in a robot soccer match, all agents in the society are there from the beginning and do not leave before the match is over (removal penalties and disqualification could be argued to be exceptions; the only actual way to leave a match prematurely is through forfeit of an entire team). In this case, when all roles are instantiated or assigned to agents in a society, we speak of a **full** instantiation of the society. Furthermore, the kind of role enactment for our society is that of *total adoption*, that is, agents adopt all the norms and goals associated with the role they enact. They can keep their own goals and norms, which should not conflict, but this way we can ensure that every agent in the society will eventually fulfill the objectives of its role. Entirely closed societies will not have negotiation scenes as the agents are specified as part of the society design, having the same characteristics as the role they are to enact; this might be closer to the robot soccer society, especially if we don't consider removal penalties and disqualification to be leaving the society.

3.3.4 Social Contracts in the Robot Soccer Society

Because OperA is based on human organizations, it is more elaborate on the social front than we need. OperA agents are assumed to be socio-cognitive entities: entities with mental attitudes towards the environment and assuming other entities also have mental attitudes [28]. In contrast, we assume our agents base their interactions solely on the expectations and rules that they all know via the specification of the OM: roles, scenes and objectives that are defined in general. The robot soccer society is a collaboration of humans and robots wherein allowed and forbidden actions are quite strictly regulated. Also, as we are not considering internal states of our agents, but we know all our robots have the same mechanics, and moreover, currently no reasoning systems other than a finite-state machine like 'if-then-else' structure; there is no such thing as 'personal conditions and requirements' that could require special clauses in a social contract. As for the human participants, we can only speculate about what they might want to do differently while still performing all the crucial tasks of their roles. For example, a GC-operator might only want to play that role for half of a match, or might want to share the task of checking coach messages with another human teammember - if the RoboCup staff agrees. Note that this would be negotiation of a facilitation/institutional role instead of an operational role, whereas OperA only considers negotiation of operational roles.

That leaves us with the bare minimum of the Social Model for now: trivial social contracts for an example instance of a soccer match. Instead of complex start scenes where agents negotiate their role-enactment, the robots in our society are assigned their roles by the human teammembers and the humans are assigned their roles by the RoboCup staff. The human roles (h-ref, a-ref, h-tm, GC-op) never evolve during a match and the same holds for the robot role of coach. The robot roles (goalkeeper, attacker, defender) can be dynamically assigned and switched at run time, by using relative positions to goals, the ball and the other agents in utility computations (2.1.1) to determine what would be their optimal role at that point in the game.

This happens to a certain extent in the current Edinferno code, mainly between Field Players: throughout the game, as the ball position varies, these role assignments switch as well. However, having them negotiate each time when a switch is coming up, would take too much time as well as it will not change much in their enactment. It would be more efficient to provide them with a coordination mechanism, including all roles and the situations in which they would be optimal.

To give an example of a social contract in the robot soccer society, consider the contract $\textit{social-contract}(b_6, \textit{coach}, \{\})$, where $b_6 \in \textit{Robots}$ instantiates the one possible instance of the coach role with no extra clauses. We could do the same for the other roles and scenes:
Let an initial formation of the robot team be the instantiations $\textit{social-contract}(b_1, goalkeeper, \{\})$, $\textit{social-contract}(b_2, defender, \{\})$, $\textit{social-contract}(b_3, attacker, \{\})$,
$\textit{social-contract}(b_4, attacker, \{\})$, $\textit{social-contract}(b_5, attacker, \{\})$, which can be seen as an *offensive* team formation because the field players are mainly attackers [60]. The human agents can be instantiated as $\textit{social-contract}(h_1, \textit{h-ref}, \{\})$, $\textit{social-contract}(h_2, \textit{a-ref}, \{\})$,
$\textit{social-contract}(h_3, \textit{a-ref}, \{\})$, $\textit{social-contract}(h_4, \textit{GC-op}, \{\})$, $\textit{social-contract}(h_5, \textit{h-tm}, \{\})$.
Here we assume that there is only one human teammember (h_5, playing h-tm) actually participating in the society. Since we use only trivial contracts this does not give us much information. Moreover, the instantiation of specific agents to specific roles will only become important when an actual game with actual robots and actual humans is to be played (such that we can further refine b_6 to be the robot called Dunlop or h_1 to be a guy called Stewart, for instance). By way of illustration: the example of the GC-op wanting to play only half a game would have the contract $\textit{social-contract}(h_4, \textit{GC-op}, \{\text{IF clockState(half-game) THEN PERMITTED}(h_4, \text{leave-match})\})$.

But, even though it isn't necessary for our agents to literally negotiate their roles before they play them, the SM represents the expectations of their behaviour as enactor of those roles (for trivial contracts: expect the agent to play the role as given in the OM). This can again be used by the other agents to know how to interact with them. Especially if the internal states of agents are unknown, social contracts provide a means to explicitely represent behaviour expectations and enable prediction of the society behaviour.

It is hard to give a concrete application of the Social Model. As described, the creation of the SM depends on characteristics and plans of specific agents, which cannot be defined in a static formal framework. Depending on the different agents that might play in a certain soccer match, the same Organizational Model will give different Social Models. Provided we have agent designs (outside OperA), in which the OM roles can be integrated and checked for consistency and compatibility.

3.4 Interaction Model (IM)

The Interaction Model (IM) takes the agreements between `reas` from the SM and combines them with enactment in interaction scenes from the OM. The scenes have the same kind of generic description which can be 'applied to' specific role-enacting agents to form 'personalized' interactions; this can be done in the same way as the instantiation of roles in the SM. That is, when `rea`s come together in an interaction, an *interaction contract* should be negotiated to describe the actual interpretation of the script for that interaction scene, according to those specific `rea`s. The advantage of using contracts for interaction is that is allows for non-rational agents to participate in the society; interaction occurs as a consequence of performing a contract, and not as a consequence of internal agent states.

3.4.1 Interaction contracts

An interaction contract is a formal LCR representation of the conditions and rules that apply to a certain interaction. For society S, scene $s \in scenes(S)$, an interaction contract IC is a tuple $interaction\text{-}contract(A, s, CC, P)$ where A is the set of agents such that A= $\{a \in Agents : \exists rea(a, r, s) \mid r \in roles(s)\}$; CC is a set of contract clauses, P is the *protocol* to follow. Again, CC is given in deontic expressions. The protocol represents the actual interaction by means of a communication pattern using the scene script and the possible illocutions the `rea`s can use. In OperA protocols are interpreted as conversations between agents before interacting a scene together, to decide how to play that scene. Protocols can be depicted using Petri Nets or UML sequence diagrams [31]; trivial contracts can be represented by standard protocols.

3.4.2 Interaction contracts in the Robot Soccer Society

For several scenes in the robot soccer society, interaction contracts could be specified. Since interaction contracts are based on communication and illocutions, these contracts are currently only applicable to the scenes involving human `rea`s. Moreover, since the specification of the SM depends on specific agents but our framework is generic and our social contracts are trivial, it is equally hard to apply the IM at this step. However, we will give a standard protocol[3] P_1 for the interaction scene ‘message-tactics’ (table C.4) in appendix E.

Let’s take the example of a robot b_6 playing the coach (rea(b_6, coach, ‘message-tactics’)), a human h_4 playing the GC-op (rea(h_4, GC-op, ‘message-tactics’)) and just one robot b_3 playing a Player (let’s say an attacker: `rea`(b_3, attacker, ‘message-tactics’)). The interaction contract for this scene with these `rea`s would be $interaction\text{-}contract(\{b_6, h_4, b_3\}, `message\text{-}tactics', \{\}, P_1)$: no contract clauses to refine the generic script, protocol is identified as P_1.

3.5 Verification

In order to check the design of an OperA framework, certain requirements should be met. The aim of this verification is to ensure that global society objectives will be achieved and interactions between the agents occur as desired.

[3]UML-diagram is made following examples in [31] and from general UML introduction `http://www.ibm.com/developerworks/rational/library/3101.html` - some symbols may have been used in a inconventional way.

Because of the open nature of OperA models (Chapter 4 on 'open societies'), verification of the model can only be done in terms of observable behaviour and not on internal states of agents. The three models of the framework should affirm the following questions ([31], pp.146):

1. Does the society design comply with its requirements?
2. Does society instantiation (social contracts) comply with the society design and is it sufficitent to guarantee society activity as specified?
3. Does society activity (interaction contracts) comply with the society design; are there interaction contracts compliant with scene descriptions?

Verifying these questions requires the logical notation of the framework specifications to check for inconsistencies and conflicts. Actual verification can be done while running the model on the final, implemented system and monitoring that all scenes and norms are applied as designed. Checking for inconsistencies and conflicts throughout the framework is done per level and described in the following sections. Besides formally checking the OperA framework, a note on the validation of its contents is also in order. All roles, rules and examples formalized in this chapter are based firmly on the official RoboCup regulations [22], supplemented with advice, information and confirmation of the members of team Edinferno.

3.5.1 Verification of the OM

At the OM level, the society structure can be verified by checking whether the formal descriptions of roles, scenes and dependencies represent the objectives of the society. Since these objectives have been constantly used in the design phase of the framework, checking this is trivial. The objectives of the society have been divided into separate objectives for the facilitation and operational roles. Dependencies between those roles have been analyzed and consequently, interaction scenes have been defined for all objectives. The following properties are checked for all scenes s in our society S:

1. $\forall s \in S_S$, $\forall r \in roles(s) : \exists R \in \mathtt{R}_S : id(R) = r$, where $r \in Roles$ are role *identifiers* and $R \in \mathtt{R}_S$ denote roles.
2. $\forall s \in S_S$, $\forall \gamma \in results(s), \exists R \in \mathtt{R}_S : id(R) \in roles(s) \wedge \gamma \in (objectives(R) \cup sub\text{-}objectives(R))$
3. $\forall R \in \mathtt{R}_S, \forall \rho \in objectives(R) \exists s \in S_s : id(R) \in roles(s) \wedge \rho \in results(s)$
4. $\forall G \in \mathtt{G}_S, \forall r \in roles(G), \forall \varphi \in norms(G), \forall \psi \in norms(R) :$
   $\varphi \wedge \psi$ is consistent, where $R \in \mathtt{R}_S : id(R) = r$. That is, $\mathtt{G}$ are groups and the norms of a group should be consistent with the norms of the roles in the group.

Informally: for all scenes, the participating roles should be specified: all participating roles have been specified in appendix B. The roles mentioned per interaction scene can be looked up there. The second and third property check the feasibility of the society objectives. For all scenes, the results must be in the (sub-)objectives of at least one of the participating roles, which can be verified by comparing the (sub-)objectives of the roles in B and the results of the scenes in C. For example, the result of the scene 'apply-penalty' is $D_{aref}apply\text{-}penalty(Robots, Penalty)$, which corresponds to the objective o2 of the a-ref role: apply-penalty(Robots, Penalty). Similarly, result r2 of 'maintain-robots' ($D_{htm}repair(h\text{-}tm), Robots)$) corresponds to the sub-objective $repair(h\text{-}tm, Robots)$ of the human teammember role. Furthermore, for all role objectives, a scene should be specified. This is not entirely the case, as some of the objectives (defend-goal, score-goal, block-player, help-defend-goal) depend entirely on agent design, and others (manage-game-clock, keep-time) are purely institutional as in the roles with those objectives do not depend on other roles to achieve them (3.2.3). The last property, for groups, is verifyed by comparing the norms of table 3.6 with those of tables Attacker and Defender in appendix B.

3.5.2 Verification of the SM

There are several requirements that should be met in the application of the Social Model specifically. That is, agents and roles in a society should be *internally coherent*, meaning that there is no internal conflict between their components (goals and plans of agents, objectives (sub-objectives) and norms of roles). Furthermore, not just any agent can enact any role. An agent and a role should be **compatible** and **consistent**. Given an internally coherent agent a and an internally coherent role r, a is compatible with r if the goals of a are a subset of the objectives of r, and all plans of a can be formed using the sub-objectives of r. Furthermore, a is consistent with r if the goals and rules of the agent and the role do not conflict.

Whether an *agent* is internally coherent depends on its specific implementation; we assume this is the case. We can verify to some extent whether the robot soccer society *roles* are internally coherent in specifications the OM. Take for example the coach role in table 3.5. We need to check whether there is a conflict between its objectives: 'messaged-tactics' and 'follow-rules'. In itself, these objectives are not conflicting: the coach just needs to make sure he does not violate the rules while messaging tactics. The same thing holds for sub-objectives. Sub-objectives in the same sub-objective set should not conflict, and they don't: in fact, in this example, the sub-objectives sets are both alternative ways to achieve the same objective. Even if they would conflict, that will (in this case) not be a problem because the actual agent enacting the role only has one of these sub-objective sets available at a time. Furthermore, sub-objectives shouldn't conflict with their objective. Here, one could argue that the 'wait' sub-objective is in conflict with the 'message-tactics' objective, but since it is necessary for the coach to wait in between messages in order not to violate the jamming rule or get his messages transferred at all, this is not a conflict. Then, the objectives and norms of the role should not conflict, which they do not; they merely restrict the coach in the manner of communicating his messages (not directly to the robots, and only if they meet the message requirements). Lastly, for each objective, there should be an interaction scene in the society which enables the realization of that objective. We did not specify interaction scenes for *all* the objectives of *each* role, but the 'message-tactics' objective of the coach can be found in the 'Interaction scene: message-tactics' table (C.4).[4]

[4]The reason for not specifying all the objectives in scenes is that most of them depend too much on specific plans designed on agent level (e.g. defend-goal, score-goal and other Player objectives).

A distinction should be made between robots and robot roles and humans and human roles respectively. Clearly, we want the robots to be compatible and consistent with the robot roles and the humans to be compatible and consistent with the human roles: it should not be possible that a robot can enact a human role. This can be ensured simply by providing the robots only with the set of robot roles.

With the examplary instantiation given in the previous section, we established a *complete* Social Model:

> A social model $SM(OM, Agts, SCs)$ where Agts is the set of agents that enact roles in the society and SCs the set of social contracts between those agents and roles in the OM ($Agts \subseteq Agents$, $SCs = \{\textit{social-contract}(a, r, CC) : a \in Agts, r \in roles(OM)\}$); a SM is *complete* iff: $\forall r \in roles(OM)$, $\exists c \in SCs : c = \textit{social-contract}(a, r, cc)$.

That is, a SM is complete if and only if there are social contracts for all roles in the OM.

3.5.3 Verification of the IM

Just like the SM, the IM can only be fully applied when agent designs are available to integrate the framework with. A similar definition of a *complete* IM can be given to formalize the idea that an IM is complete if and only if there is an interaction contract for every scene script in the OM:

> An interaction model $IM(SM, ICs)$, where $ICs = \{\textit{interaction-contract}(Parties, scene, CC) : Parties \subseteq Reas, scene \in Scenes\}$ is a set of interaction contracts between `rea`s given in the SM, is *complete* iff $\forall s \in scenes(OM)$, $\exists c \in \textit{interaction-contract}(Parties_s, s, cc)$, where $Parties_s = \{rea(a, r, s) \mid r \in roles(s)\}$.

The verification of the SM and IM is recommended work for the actual implementation of this robot roccer society framework into a robot team (chapter 6).

3.5.4 Summary

To answer the verification questions given at the beginning of this section, we analyzed our framework design given in this chapter. On this high, abstract level of framework design, concrete contracts between actual agents and the described roles and interactions cannot be verifyed since those actual agents are not defined. However, the means to verify and test whether the society objectives will be achieved using this framework have been given. The subject of agent designs using this framework should be covered in future work.

Chapter 4

Related Research - Plan Recognition

As a first step towards integrating the agent framework of the previous chapter, we focus on the role of the *coach* as a new addition to Edinferno's team. In order to provide the players with strategic advice and thus improve the team's performance from observations, the coach is considered a kind of ad hoc agent. One of the coach's tasks is to determine current strategies of the players in order to decide on better moves for them. In this chapter, the concept of *plan recognition* will be explained in terms of possible methods to approach it. The methods discussed here are collected from a machine learning point of view, using real-world practical situations as testing domains and numerical rather than logical techniques to estimate the plans of other agents. For comparison, logical approaches to plan recognition are also discussed.

4.1 Ad Hoc Coordination

As introduced in Chapter 1, ad hoc coordination is a special kind of coordination. Coordination in general is for agents to cooperate in structured interaction, achieving some goal by working together according to a set of regulated or agreed upon plans [13]. Coordination in multi-agent systems is an important aspect of robotics, game theory and AI in general.

Ad hoc coordination is coordination *without* such a predefined set of plans: agents have to find ways to cooperate with unknown agents and without prior agreements on how to work together [7, 8, 41, 94, 95]. In these related researches on ad hoc coordination in robot soccer, the scenario for a single *ad hoc agent* to cooperate with unfamiliar teammates, without pre-coordination, is considered. In such a ad hoc or *impromptu* setting, there is a core team of players and one agent with the task of adapting its behaviour to the team [14]. *Roles* can aid in this task by providing recognizable functions or behaviours, when considering a whole team of agents [13, 14, 17, 41] (see Chapter 2.1.1 on roles).

4.1.1 Plan Recognition

In order to achieve ad hoc teamwork or even to design a single ad hoc agent, many issues have to be solved. Not only does the ad hoc agent need to adapt his behaviour 'on the fly' in unknown situations, before he can do so he needs to be able to infer or recognize the behaviour of the other agents in the field. This problem, *plan recognition*, is a hard problem since the (ad hoc) agent can only perceive the physical actions of the acting agents and environment states in which they perform those actions. A 'plan' in this sense represents the intention of the agent, how he plans to achieve some goal [17, 82]. These intentions are 'hidden' from the observer but can be inferred based on observated actions of the agent. Here we assume the recognition to be obstructed or 'keyhole' recognition, in which the observed agent either deliberately hides his intentions or is not aware of being watched [27]. Plan recognition, in contrast to *planning*, is concerned with representing actual situations rather than hypothetical explanations of actions, together with uncertainty: instead of choosing *any* plan that achieves a desired goal, the *specific* plan that is currently performed has to be identified [55].

Traditionally, the observing agent is provided with a *library* of domain-specific actions and models that predefine sequences of actions [17, 98]. To infer the plan of the acting agent, the observing agent constructs a possible sequence of actions that connects the observed actions to one of the possible goals. For human football, recognizing tactical intentions can be done from observable behaviour only, when considering not just the player's own actions, but also his interactions with teammates and opponent players [9]. Related to plan recognition, there is the problem of *opponent modelling* [6, 18, 73, 78, 79, 89] which can be considered as a more elaborate version: observations are not only used to estimate an agent's plan or behaviour, but also to build a complete model of that player, for example what strategies he has in situations other than the current one or what his individual goals are. Considering individual goals is sometimes referred to as *intent inference*: to estimate the internal state of another agent [88, 98]. Opponent modelling can be used to infer an optimal strategy against a given model. Such a model can for example be learned from the opponent's behaviour in the past [18]. When it is not possible to collect many past interactions to learn from, the opponent's future actions can be predicted from the optimal behaviour in its current situation [93]. Alternatively, a prior distribution over possible strategies can be assumed [6]. Applications of plan recognition and opponent modelling are not only useful in robot soccer, but also in games like poker [6, 89]. Furthermore, human-robot interaction could benefit from intent recognition to improve cooperation [27], for example in learning by demonstration [99].

There are many ways to approach the problem of plan recognition and these approaches can be applied to a wide scale of (real-world) domains, of which robot soccer is of main interest for this thesis. In the following sections several methods and their applications will be presented.

4.2 Machine Learning Approaches

4.2.1 Heuristics

Heuristic techniques stem from the field of psychology originally, where they typically are efficient rules to explain how people make decisions and solve problems [86]. In Artificial Intelligence, heuristics are mentioned mainly in the context of problem solving via *searching* [80]. For environments that are static, observable, discrete and deterministic, the state space can be searched strategically, for example with breadth-first or depth-first search through a search tree. These are examples of uninformed search; informed, or *heuristic* search on the other hand extends the use of such search strategies by adding *knowledge* of the problem to the search algorithm. For example in *pruning* an ordering on parts of the search tree and means to decide which nodes nót to expand is imparted in the system. To eliminate possibilities from consideration without having to examine them is an important technique in AI as it makes problem solving considerably more efficient. This all relates to the well-known *frame problem*: representing all relevant facts about a robot's environment and considering if and how they change over time [80]. A heuristic function is an estimation of the expected cost of the cheapest path from a given node n to a goal node. These functions can be learned from experience (solutions to similar problems solved before), or devised from 'relaxed' (simplified) versions of the problem to which an optimal solution is easily found.

An example of the use of heuristics in a plan recognition method can be found in [78]. An agent is assumed to be provided with a library of possible plans through which he can search and try to match his observations to the pre-defined plans. The best match of plan is found using a combination of hill-climbing, an evaluation function and a naive Bayes classifier. The low-level plan recognition system in [29] is based on a library of pattern templates to look up agent features like velocity, heading and position. In this sense, a plan only represents where the agent is headed. It can be used for single agents or agent-pairs and is applied to training battle data from maneuvering human army troops.

Many of the plan recognition related works make use of probabilistic methods, often combined with Bayesian belief updating and/or classification. In the following, the most recurrent methods are presented.

4.2.2 (Dynamic) Bayesian Networks

The seminal work of Charniak and Goldman states the plan recognition problem is largely a problem of inference under conditions of *uncertainty* [15, 19]. They propose an agent's execution of a plan should be represented as a Bayesian network, using Bayesian probability theory to infer candidate plans and update the network. Plan recognition is then defined as the process of constructing and evaluating these networks. According to Carberry's review [17], Bayesian Networks are most appropriate for domains where prior and conditional probabilities can be reliably estimated and causal influence between nodes reliably determined. Charniak and Goldman applied the system to the problem of understanding a character's action in a story (written text rather than moving objects).

While the order of actions is not considered in Charniak and Goldman's paper, other researchers attempted to capture the influence of temporal aspects in a dynamic version of a Bayesian network: Dynamic Belief Networks or Dynamic Bayesian Networks (DBN) [15, 17, 49, 66]. In a DBN, multiple nodes are used to represent the status of one variable at different instances of time. These networks are used in the construction of a plan inference system by Albrecht et al. [2, 17], in order to infer an agent's plan during an adventure game. They conclude that, if sufficient training data can be collected and the causal structure of the network can be clearly identified, a Dynamic Belief Network-based system is appropriate for this application. DBN are also used in Nicholson and Brady's work for monitoring robot vehicles and people in a restricted dynamic environment in the field of tracking in the Data Association Problem (deciding which agent gives rise to an observation) [66].

An alternative network to represent plans, like a 'Simple Temporal Network', can effectively capture the temporal dependencies between the plan steps in a directed graph style [78]. Riley and Veloso's research is focussed on opponent models in robot soccer, which are in this case probabilistic representations of the opponent's predicted movements. In order to handle the uncertainty in the environment and to allow models to represent more than only a predicted location, their models also contain information about the ball's movement and the observed player's initial location.

Bayesian networks are a suitable method to choose plan from predefined libraries, because under the assumption that an agent behaves according to a known plan, these networks can handle the uncertainty when a set of observations can be explained by different plans [103].

4.2.3 Markov Models

However, according to Bui [15], online plan recognition using DBNs will be unable to scale up when the belief state space becomes too large or when plan hierarchies become more detailed. He proposes to use a Hidden Markov Model (HMM) for representing the execution of a hierarchy of *policies* or plans and using an approximate inference scheme to do *policy recognition*. The idea of a Markov Model in general is that it models a process where a state depends on previous states in a non-deterministic way [96]. A policy is a mapping from states to actions, defining what actions would be optimal in each state to reach a certain goal.

Hidden Markov Models

Frameworks based on Hidden Markov Models (HMM) can be used to respresent and recognize strategic behaviours of robotic soccer agents [46]. In Han and Veloso's work, a robot is assumed to act according to partially or fully predefined sets of behaviours and another robot has the task of identifying which behaviour is executed. High-level strategies like 'go-to-ball' and 'go-to-defend' are considered, in both simulation and actual robots. To model a behaviour as an HMM, they represented their system as a set of discrete states. These Markov states could map to the physical location of the agent, but this is not necessarily the case. In this work they correspond to 'sub stages' of the behaviour, for example the stages $\{beginningofbehaviorexecution, rotatingtowardsball, infrontofball, besidetheball, behindtheball\}$ being states of the 'go behind ball' behaviour. To do plan recognition using HMM representations of behaviours, the observing robot can only infer the probability of the acting robot being at certain state. This probability represents the likelihood of that state to be the actual internal state of the acting robot. The observing robot is interested in the chance that the observed state is the actual (hidden) state, given some observations and possibly some parameters.

Another way to do inference on a HMM is to use an approximation algorithm like the Rao-Blackwellised Particle Filter in Bui et al.'s work [15, 16]. They implemented an 'Abstract HMM' with a dynamic Bayesian network structure and used the filter to combine exact inference by updating belief states with approximate sampling-based inference. The network structure is applied to model the online and dynamic aspect of an otherwise static HMM. In [15] this model is extended to allow policies with internal memory which can be updated in a Markovian way. Memory in policies allows for representing an uninterrupted sequence of sub-plans and use *histories* of sub-plans instead of only the current state. Both methods are implemented in a surveillance domain. The main advantage of the methods in these papers is that they are scalable, dynamic and hierarchical, which makes them realistic and general frameworks.

Saria and Mahadevan continued their work by extending those abstract models from a single agent to a multi-agent setting. They present a hierarchical dynamic Bayes network that allows reasoning about the interaction among multiple cooperating agents, using similar hierachical abstract policies and Rao-Blackwellised Particle Filter sampling and inference techniques. They call their framework a 'Hierarchical Multiagent Markov Process', which is a combination of an HMM and a Markov Decision Process, in order to model hierarchical policy execution in a multi-agent systems for robot soccer [82].

Markov Decision Processes

Where Hidden Markov Models make use of the notion of hidden (internal) states, a typical Markov Decision Process (MDP) is again fully observable. Markov Decision Processes are specifically suitable for solving optimization problems related to dynamic programming and reinforcement learning [13, 34, 96]. Reinforcement learning techniques can be used by agents to estimate expected rewards for individual or joint actions based on past experience, in order to adapt their plans to the environment [20, 56, 103]. In a typical MDP, the world consists of a set of possible states and actions permissible in those states. A *policy* maps a state to an action (deterministic) or a distribution over a set of actions (stochastic). A policy determines what states are visited using which actions. If a policy is fixed, the resulting sequence of states behaves again like a Markov Chain. The difference with a Markov Chain is the addition of *choice* and *reward*. Choice in the sense that there is a *set* of actions to choose from: at each state, an action is chosen which leads to one of all possible next states. Which action is chosen depends on the *transition function* and *reward function*: the probability of reaching a next state given any current state and any action, and the expected reward for that transition. *Solving* an MDP is to find its optimal policy, which is the policy with the highest rewards for all its states [13, 34, 53].

Because MDPs are frameworks for decision making, they are often applied in (multi-) agent systems and robotic domains. The problem of multi-agent coordination is a suitable application. Boutilier [13] uses Multi-agent MDPs (MMDP) to model the process of interacting agents, much like an MDP of which the actions (and possibly decisions) are distributed among multiple agents. This MMDP is then expanded by combining it with a randomization protocol: a learning mechanism that requires agents to select actions (from their subset of interesting actions) randomly until coordination is achieved. Another variant of MDP, Semi-MDP, has been applied by [56]. The difference with a regular MDP is that an SMDP provides the basis for learning to choose among temporally abstract actions, rather than executing actions at discrete time steps.

For *partially observable* domains like the dynamic realistic robot soccer domain several techniques can be applied that work well for a small problem, for example a repeated game like the Prisoner's Dilemma, but quickly grow intractable for more complex systems [14]. In a system where the agent decides according to an MDP, but where the underlying state of the system cannot be directly observed, a Partially Observable Markov Decision Process (POMDP) can be used as a model [43, 53, 99]. Because the agent does not know in which state it is, it maintains a probability distribution (belief state) over the possible states. In addition to an MDP, a POMDP has a set of observations and a set of conditional observation probabilities. POMDPs can again be found in many different versions (Interactive POMDP, Decentralised POMDP etcetera), but a full review of those falls outside the scope of this thesis.

4.3 Logical Approaches

If one does not want to 'enter the depths of probabilistic inference' [55], plan recognition can also be approached with classical deductive inference when the observing agent is provided with a library of actions and a closed world (and possibly other simplicity constraints) is assumed. The closed-world assumption states that the knowledge base is complete, meaning that there exists no more entities or concepts than the one that have a description in the knowledge base (or ontology - see 3.2.2) [61, 80].
Kautz and Allen [55] proposed a formal theory of plan recognition in the domain of story understanding via *circumscription* [61, 75]. Circumscription, presented by McCarthy in 1985 as means to formalize common sense knowledge, is used to transform a first order theory of action into an action taxonomy. This taxonomy can then be used to deduce the actions an agent is performing, assuming the taxonomy is an exhaustive description of actions and how they can be performed. This is not suitable for a dynamic domain like robot soccer, but if these assumptions can be met, it yields exact and certain answers to the plan recognition problem. A not strictly logical, but otherwise abstract symbolical approach is presented in [5, 54], where behaviour graphs and Feature Decision Trees are used to match observations to known behaviours in the domain of robot soccer. Their approach is even said to be able to handle lossy observations and interleaved plans, that do not exactly match the known behaviours. Game theory has also been applied to infer models of other agents based on past interactions and adapt behaviour ad hoc [3, 18]. For

example in [3], the concept of a Bayesian Nash equilibrium is used to find optimal actions in an human-machine experiment with repeated games (Prisoner's Dilemma and Rock-Paper-Scissors).

Another interesting logic-based approach to plan or intention recognition is the 'mental state abduction' method by Sindlar et al. [88], in which a set of explanations for an agent's behaviour is computed, based on observations and knowledge of the agent's rules. This technique uses the BDI-based programming language 2APL combined with nonmonotonic reasoning (see section 2.3.1). The idea is that agents' behaviour depends on their roles, which are known and can be used to infer *why* they behave this way, e.g. what their plans or intentions are on which their observable behaviour is based.

A short introduction in several related researches in the fields of plan recognition and opponent modelling have been given. In the following chapter, the methods we actually applied in our plan recognition module will be discussed in more detail.

Chapter 5

Plan Recognition Module

As a proof of concept, a part of the framework is further explored by means of implementation and testing. This part was a plan recognition module, inspired by the RoboCup Drop-In Player challenge [21]. The first step a drop-in player or coach has to do to be able to adapt to a team of players or give advice to them respectively, is identifying what they are doing. As described in Chapter 4, there are many ways to do so. In this chapter, our approach to the problem of plan recognition is presented.

5.1 Idea

Our implementation is inspired by the concept of the coach role, and the question whether a coach agent would be able to identify the behaviour of player agents. We interpret *behaviour* as a certain set of characteristic *goals* that an agent can have and the specifc plans and actions leading to those goals. For example, an agent in an attacking role should adopt an attacking behaviour, which is, intuitively, expressed in the field as a set of movements concentrated near the ball and the opponent's goal, with a focus on shots on the goal rather than dribbling. Since RoboCup players are autonomous agents, their 'true' plans, intentions or strategies are private or 'hidden' states. A coach agent can only use the from the outside observable actions to reason about those internal plans.

The identification of a certain behaviour type (*class*) given a set of observations (*instances*) is a typical *classification* problem. This means we need to define some numerical *features*: measurable properties of the instance that can help distuinguishing between classes. Ultimately, our goal is taking a step towards a general approach for identifying many different behaviours, including complex interactive behaviours of multiple agents. Naturally, we have to start at the beginning: one observer, the coach, and one player.

We define two basic behaviours with the suggestive labels *'Go to your own goal'* and *'Go to the opponent's goal'*. For this first attempt we take target positions that have static positions on the soccer field. Behaviours in our experiment are defined by *trajectories*: typical paths traveled by the robot executing plans 'belonging' to that behaviour. What makes a path characteristic for a specific behaviour is its direction, which of the goals he is approaching and from what angle, whether or not the robot gets closer to that goal and whether the position where he ends up is near that goal.

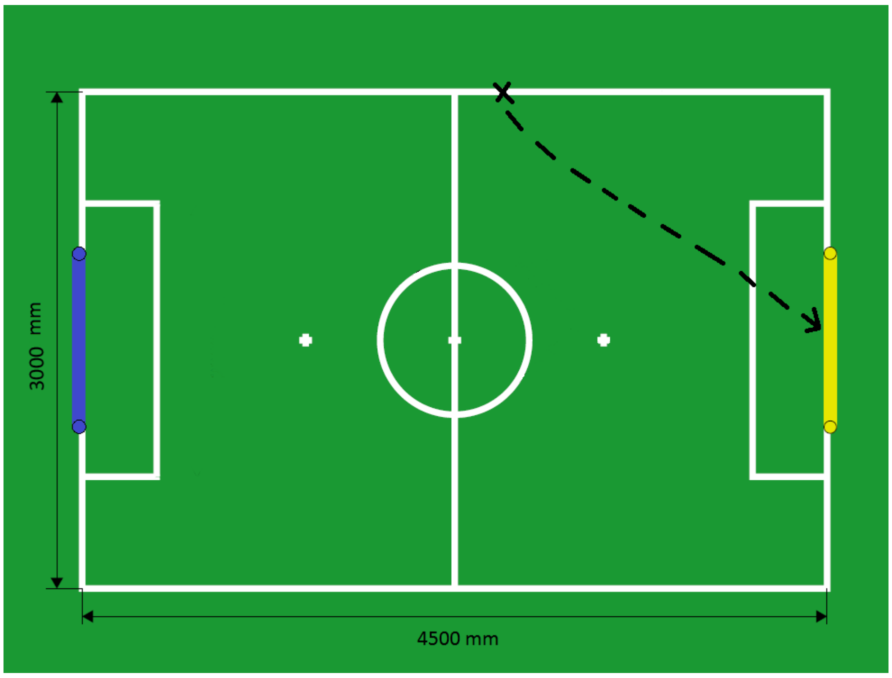


FIGURE 5.1: Our RoboCup soccer field with example trajectory. The dimensions from the borders (instead of the field lines) are 5500x4000 mm. The field used in Edinburgh is slightly smaller than the official RoboCup field, which has borders at 6000x4000 mm.

A trajectory consists of states and transitions between those states, starting at the initial position in which the robot is placed and ending at the position where he stops moving (which, ideally, is near a goal position). States can correspond directly to the physical position of the player on the field, but this need not necessarily be the case. In our case, it seemed useful to use relative position and orientatiom of the player with respect to its target, rather than absolute position and orientation with respect to the field dimensions.

As the coach observes the player, executing some plan, he has *beliefs* over what behaviour he thinks the player is performing. These beliefs are defined as (log)likelihoods over the set of behaviours, based on the observations that the coach does. Beliefs can be calculated over a complete trajectory or updated for each sequence of transitions of a set length, provided that the information about the player's path can be collected and analyzed real-time. Our goal is to compute those likelihoods and decide which behaviour $b \in B$ is more likely the true 'state of nature' given the observation O [34]. Here, B is the set of possible behaviours $B = \{b_1, b_2\}$, with $b_1 = GoToOwnGoal$ and $b_2 = GoToOppGoal$, and O is a sequence of states $\mathbf{s}$ of length N (which may, but does not need to, be the length of the entire trajectory) and transitions $\mathbf{s}, \mathbf{s}'$. In short, we are interested in $argmax_b\ p(b|O)$, which can be calculated using Bayes' decision rule:

$$argmax_b\ p(b|O) = argmax\ p(O|b)\ p(b) \tag{5.1}$$

This means we need to calculate $p(O|b)$, which is $p(b|O) = \prod_i^N p((\mathbf{s_i}, \mathbf{s_{i+1}})|b)$. We could obtain these by either defining all transitions $(\mathbf{s}'|\mathbf{s}, b)$ for each behaviour and each possible state $\mathbf{s}$ by hand, or by collecting training data to estimate those probabilities from. Of course the latter is less time-consuming and therefore preferable.

5.2 Data Collection

For both behaviour classes a number of trajectory log files were collected, with trajectories starting from 7 different positions in the field to cover as much of the field positions as possible. To ensure the robot would walk exactly the trajectories designed for these experiments, and because this robot's path planning was not the focus of this experiment, it was remotely controlled using a joystick (X-Box controller) and located in the different starting positions manually, instead of walking autonomously. The log files contain the player's absolute x- and y-position on the field and its orientation θ for every time step of 0.3 seconds. Relative versions of these features are used to distinguish between behaviour classes (section 5.3). Additional features could be logged, like whether or not the ball was seen, whether or not there are other robots in the field, and if so, where they are and what team they are on and so on. Such features have not been used for this classifier.

5.2.1 Self-localization

The way in which the robot estimates its own position and orientation on the soccer field relies mostly on visual cues and *odometry*, which is the offset since the last motion update: $\Delta x_t, \Delta y_t, \Delta\theta_t$ [57]. The information the robot gets from its environment are processed by a Monte Carlo-based Particle Filter module. The modules used by Edinburgh's RoboCup-team are based on the ones released by the German team B-Human in 2011 [37].

The Monte Carlo-based Particle Filter self-localization module, implemented by B-Human as Augmented Monte Carlo Localization, is a version of Markov localization, using fast sampling techniques to approximate probability distributions over possible positions of the robot [102]. Information from both the robot's motions and sensors is used to update *beliefs* the robot has about its position at each time step. Motion information, odometry, is provided by the walking engine, based only on the motion of the legs. The idea of MCL is to represent the posterior belief about the robot's position by a set of weighted random samples (*particles*), a sample set constituting a discrete approximation of a probability distribution. These samples are generated from the previously computed set and weighted according to their likelihood of being the actual position.

This self-localization method is however still not very robust, which causes the robot to be 'lost' sometimes. There are multiple reasons for it to be lost. One major issue especially for RoboCup is that the environment of the robot, the soccer field, is symmetrical: it looks the same from different viewpoints. Especially in the official RoboCup field, where both goals are yellow, it is hard for the robots to be certain at what side of the field they are. In an attempt to restrain the consequences of the unstable self-localization module in our experiment, we used one yellow goal and one blue goal, similar to the situation in RoboCup before 2012, instead of two yellow ones[1]. Another reason for the robot to be lost is the so-called 'kidnapped robot problem', which occurs if the robot is replaced and it does not recognize this state change. It has to re-estimate its position, which can take longer than usual since it is not in a place where it last thinks it was. A similar thing happens to the odometry measures if the robot falls: over time, the more the robot moves, the less reliable the measures become. According to Laue et al. [57], their MCL-algorithm can deal with the kidnapped robot problem a lot better than earlier modules. In a soccer game where the robots act autonomously, this is a rare problem. However, in our data collection sessions, we 'kidnap' the robot quite often to place it in a new starting position. These issues of self-localization can however not be solved within the scope of this project, so for the time being, we assume the logs of the robot to be accurate and trustworthy.

Due to the variable performance of the self-localization and logging modules, some of the collected data turned out to be too noisy to use or not properly logged. This resulted in uneven sets of data for the two behaviour classes: a total of 60 trajectories for *GoToOppGoal* and 46 for *GoToOwnGoal*. The collected log files were divided into a set of training data and validation data according to a 80/20 ratio.

5.3 Preprocessing

To get a useful representation of the training data for our classifier some preprocessing methods were applied.

5.3.1 Smoothing

Not all of the collected trajectories represented their actual paths correctly. What looked like a reasonably straight path on the field could have log files showing big 'jumps' between coordinates. This could be caused by the mirrorring issue mentioned before or other reasons for the robot to mislocate his position on the field. Such random errors in the data are characterized as *noise*. To compensate for noise without compromising the underlying information we smoothed them using the low-pass Savitzky-Golay filter [83]. The idea behind this filter is to make for each point a least-square fit with a polynomial of high order over a odd-sized window centered at that point. The filter is based on the assumption that the time steps are equally spaced. The filter uses a *convolution* process in combination with the method of *least squares* to accomplish such smoothing. Convolution computes *moving averages* for a fixed number of points, by averaging over a group of points, then dropping one point at one end of the group and adding the next point at the other end and repeating that process for all datapoints. By multiplying datapoints by a corresponding *convolution integer*, or weight, before averaging (the central point having the largest weight), *convolutes* are obtained for a group of points, which can thus be seen as weighted averages. The method of least squares minimizes the sum of squared differences between the

[1] http://www.tzi.de/spl/pub/Website/Downloads/Rules2012.pdf

data values and their corresponding modelled values to form a curve that fits the data best. The modelled values in this case are the weighted averages computed in the convolution step.
This yields an approximation of the true data values, in our case the absolute coordinates, removing the noise without degrading the wanted information. We used an implementation from SciPy.org[2] with windowsize 13 and polynomial order of 3. These parameter values were chosen based on a few trials on trajectories of variable noise. These values yielded a reasonable amount of smoothness without making the original trajectory unrecognizable.

5.3.2 Relative Distances and Angles to Goals

For the classification procedure, we describe each state as a feature vector. The features used in this experiment are relative distances and angles to the goals, obtained from the (smoothed) absolute information from the log files. Distance to both the goals is computed as Euclidean distance in the 2D plane, between the robot's coordinates (x, y) and the static goal coordinates, $(2625, 0)$ and $(-2625, 0)$ for the 'own' goal and 'opponent' goal respectively. These positions are the exact centre of the goals, beyond the ground line. The field dimensions are given in millimeters, the x-axis ranging from -2750 to 2750, the y-axis from -2000 to 2000, with point $(0, 0)$ being the middle of the field. These are the dimensions of the entire field, so including the area outside the field lines up to the 'walls' surrounding the entire arena. Note that these dimensions are slightly different from the ones in the official RoboCup arena, due to the limited space available in the Edinferno lab. Note that, if we were to change ends (the static position $(-2625, 0)$ now denoting 'own' goal and vice versa), the approach described in this chapter is still applicable if only the output likelihoods are swapped as well as the goal positions.

The original orientation of the robot on the field is defined for the robot being in the middle of the field: the middle (origin) of the circle, $(0, 0)$, is taken as the robot's core. Instead of having radians from 0 to 2π like in a unit circle, Edinferno's code uses radians 0 to π for the upper half of the circle and $-\pi$ to 0 for the bottom half, where $\theta \approx 0$ is the orientation facing the opponent goal (the yellow goal) and $\theta = \pi$ or $-\pi$ facing the own goal (the blue goal), as shown in figure 5.2.

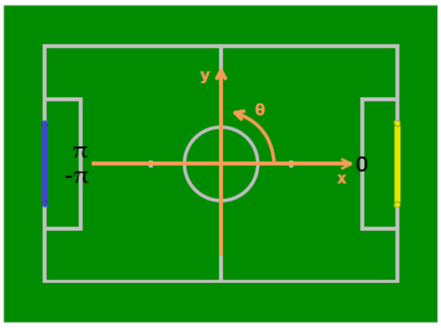


FIGURE 5.2: Global robot pose orientation (edited from B-Human code release 2011)

[2]http://wiki.scipy.org/Cookbook/SavitzkyGolay

To compute the relative angle to the goal we use the orientation and position of the robot and the position of the goal. We shift the coordinate system to have the robot's core as its origin. For each goal and robot position, a reference point can be defined that shares its x-coordinate with the goal and its y-coordinate with the robot, forming a right angle with legs $C = |x|$ and $A = |y|$. We can compute the relative angle from the robot to the reference point (say, θ_2) using the cosine inverse of the distance to the reference point (C) divided by the distance to the goal (A): $\theta_2 = \cos'(C/A)$. The angle from the robot to the goal, θ_3, can be computed using the current orientation of the robot θ and the angle to the reference point θ_2. Based on θ ranging from 0 to π and $-\pi$ to 0, we can do some simple additions and substractions between the absolutes of π, θ and θ_2, conditioned on quadrant of the goal, to get θ_3. For example in figure 5.3, θ_3 can be computed as $\theta - \theta_2$.

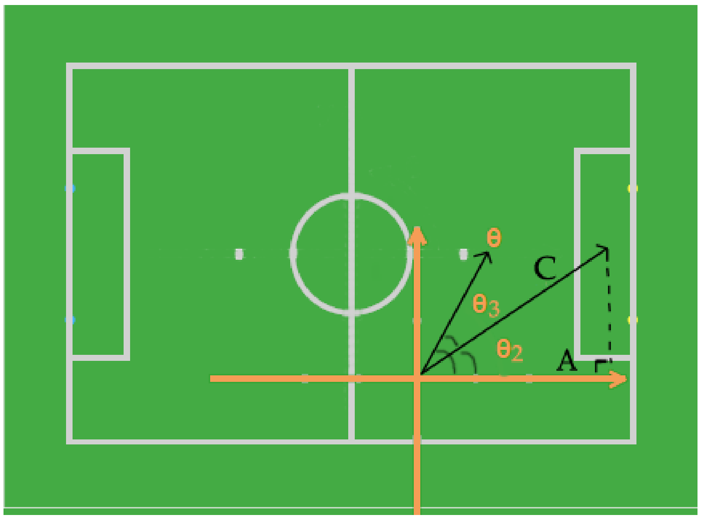


FIGURE 5.3: Example: goal and current orientation θ lie in the same quadrant.

The angle from the player robot to a goal is a feature that represents to what extent the player is facing that goal, which is a major clue to predict whether he will move in that direction or not. This is of course based on the real-world assumption that people 'follow their noses', meaning that a person tends to go in the direction that he is facing. Moreover, initial tests with the NAO player showed that he can move faster and steadier if he is facing the direction in which he moves: sidestepping or going backwards is possible, but is harder on his stabilizers and motors, causing him to fall over more easily.
This converted data, showing how angles and distances with respect to the goals change over time, will give the coach the minimum information it needs to decide on the player's plan.

5.4 Representation and Implementation

We have two agents: the coach, who merely observes in this experiment, and a player, who interacts with the environment. The environment is everything outside the player, everything that it interacts with. In this case, it is the soccer field and the goals (in the complete RoboCup environment this would also include the ball and other players). Interactions of the player with the environment are the actions that he does, which can change the environment, presenting the agent with new situations to deal with. These interactions are logged at each discrete time step $t = 0, 1, 2...$, with a set interval of 0.3 seconds between them.

Ultimately, the coach should use his visual information to get actual observations; for now we use only the self-localization information of the player, communicated to the coach. Let's assume the coach can get the information the player is logging about his position in real-time. In section 5.1, the notions of states, transitions and observations were introduced which will be defined in more detail in this section.

5.4.1 Modelling

Before presenting our behaviour model, a quick recap of two common models is in order. Both are *Markov Processes*, meaning they are a stochastic process with the *Markov property*: future behaviour of the model depends only on the current state of the model and not on a complete history of states and actions. Here a history is a sequence of past events $(\mathbf{s}_t, \mathbf{s}_{t-1}, ..., \mathbf{s}_0)$ and the complete probability distribution in the case where a new state depends on everything that has happened earlier is $Pr(\mathbf{s}_{t+1} = \mathbf{s}' | \mathbf{s}_t, \mathbf{s}_{t-1}, ..., \mathbf{s}_0)$; for all $\mathbf{s}'$ and all possible values of the past events. If, on the other hand, an environment has the Markov property, this distribution is simply $Pr(\mathbf{s}_{t+1} = \mathbf{s}' | \mathbf{s}_t)$ for all $\mathbf{s}', \mathbf{s}_t$. In short, if and only if these probabilities are equal for all $\mathbf{s}'$ and histories, the model has the Markov property [96].

Markov Chain

Traditionally the term 'Markov Chain' is used to indicate a discrete-time stochastic process, although continuous-time Markov processes also exist.
Markov Chains can be characterized by an initial distribution over states, $p(X_1 = i)$, and a state transition matrix, $p(X_t = j | X_{t-1} = i)$. If the observed variables are discrete ($X_t \in \{1, ..., K\}$, where $X_1, ..., X_T$ is a sequence of observations of length T), the conditional distribution $p(X_t | X_{t-1})$ can be written as a $K \times K$ transition matrix A, where $A_{ij} = p(X_t = j | X_{t-1} = i)$ is the probability of going from state i to state j [65]. If a state transition is *stationary*, it means that the transition function is independent of time: $p(X_{t+1} = j | X_t = i) = p(X_t = j | X_{t-1} = i)$.

A Markov Chain can be drawn as a directed graph, with the nodes representing states and the arcs representing legal transitions (non-zero elements of A). The weights associated with the arcs represent the transition probabilities.

Markov Decision Process

A Markov Decision Process is an extension of a Markov Chain, where the major difference lies in the addition of actions and rewards. In the literature, types of behaviours are sometimes modelled as *Markov Decision Processes* (MDPs) or variants of it (see Chapter 4). States are then assumed to be fully observable and the agent executing the task is said to 'solve' the MDP that he is given. Let a finite Markov Decision Process M be a tuple (S, A, T, R), where:

S: finite set of states $s \in S$
A: finite set of actions $a \in A$
T: transition function; for any state and action $s \in S$ and $a \in A$, the probability of each possible next state s' is: $T(s'|s,a) = Pr(s_{t+1} = s'|s_t = s, a_t = a)$ [3].
R: reward function; $R(s'|s,a)$ denotes the expected next reward ($r \in \mathbb{R}$) for taking action a in state s leading to any next state s': $R(s'|s,a) = E\{r_{t+1}|s_t = s, a_t = a, s_{t+1} = s'\}$[4]. Sometimes a discount factor $\gamma \rightarrow [0,1)$ is added to the definition to prevent infinite rewards [76].

A (stationary) *policy* π is a mapping from each state $s \in S$ and action $a \in A$ to the probability $\pi(s,a)$ of taking action a in state s [96]. An optimal policy is the policy for which the expected reward is maximized. To 'solve' an MDP is to compute its *optimal policy*, which is done by maximizing the reward function for all states in the policy.

Parameterized Markov Chain

Since our robot is joystick-controlled and therefore does not choose its actions autonomously, we do not need the full machinery of a Markov Decision Process: the reward function and discount factor can be omitted. However, as our states are vectors of continuous features, a discrete-state Markov Chain would also not fit the profile. To model a behaviour as something that is not as elaborate as a Markov Decision Process, yet not as basic as a Markov Chain, we adopt the notion of a *parameterized Markov Chain*. Let a PMC M be a tuple (S, A, T), where:
S is a finite set of states,
A is a finite set of actions,
T the transition function: for any state and action $\mathbf{s} \in S$ and $\mathbf{a} \in A$, the probability of each possible next state s' is: $T(\mathbf{s'}|\mathbf{s}, \mathbf{a})$.

Our *state space* consists of feature vectors containing measures of the player's (physical) state or situation, relative to the goals. Let a state be a 4-dimensional feature vector $\mathbf{s}$ with real-valued components: $\mathbf{s} = (\text{distanceToOwnGoal}, \text{angleToOwnGoal}, \text{distanceToOppGoal}, \text{angleToOppGoal})$, representing the relative location of the player with respect to both goals. We assume these features to be *independent*, meaning that for any distance, the robot can be in any angle (formally, statistical independence is denoted by $P(dist, ang) = P(dist)P(ang)$ [34]).

A transition $(\mathbf{s}, \mathbf{s'})$ between a state $\mathbf{s}$ and a next state $\mathbf{s'}$ can be represented as the difference between the corresponding vector features, which itself is again a vector $\mathbf{a}$:
$\mathbf{a} = (\text{diffDistToOwnGoal}, \text{diffAngleToOwnGoal}, \text{diffDistToOppGoal}, \text{diffAngleToOppGoal})$. This vector represents the *action* that is taken at $\mathbf{s}$, leading to a possible next state $\mathbf{s'}$. Intuitively, an action is a move in a certain direction, travelling a certain distance. An action is static if the transition $(\mathbf{s}, \mathbf{s'})$ is such that $\mathbf{s'} = \mathbf{s}$.

Note that for our two behaviours *GoToOwnGoal* and *GoToOppGoal* the state and action spaces are equal but the transition functions T are different: the probability with which transitions

[3] T can also be written as P [96] or Pr [13]
[4] The reward function can also be written as $R(s,a)$, giving the reward for taking action a in state s, independent of next state [76].

$(\mathbf{s}, \mathbf{s}')$ occur is conditioned on the specific PMC in which it takes place. These transition probability distributions have been learned from training data of their corresponding PMCs. Let the behaviour library B be the modelled version of the set defined in section 5.1, with M_{own} being the PMC for the behaviour class *GoToOwnGoal* and M_{opp} the PMC for the class *GoToOppGoal*.

5.4.2 Fitting Gaussians

Let the probabilities of reaching a next state $\mathbf{s}'$ given a current state $\mathbf{s}$, action $\mathbf{a}$, $T(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, be normally distributed, $T(\mathbf{s}'|\mathbf{s}, \mathbf{a}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. In learning this function T from the training data we have a problem in that there are infinitely many pairs $(\mathbf{s}, \mathbf{s}')$ possible but only a limited amount collected. This is known as *data sparseness*, which can be solved by abstraction, for instance by discretizing the field into grid cells. We divide the field into eight cells (4x2), $G = \{g_0, g_1, ..., g_7\}$, and group states into these grid cells according to their (x, y) positions. Now we can define T per grid cell by using a multivariate Gaussian with feature vector $\mathbf{s}$, action vector $\mathbf{a}$ and next state vector $\mathbf{s}'$, which gives us the probability of going to $\mathbf{s}'$ from $\mathbf{s}$ doing $\mathbf{a}$.

For a group of states $S_g = \{\mathbf{s}, \mathbf{s_1}, ..., \mathbf{s_n}\}$ in a grid cell $g \in G$, the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ of their according Gaussian can be learned from the training data. Here $\boldsymbol{\mu}$ is the mean of the action vectors $\mathbf{a}$, $\boldsymbol{\mu_a}$, plus the originating state $\mathbf{s}$. The mean of the Gaussian depends on the originating state, as shown in figure 5.4. $\boldsymbol{\Sigma}$ is $diag(\boldsymbol{\sigma})$, where $\boldsymbol{\sigma}$ is a vector of variances of each feature. By adding $\boldsymbol{\mu_a}$ to the input state vector $\mathbf{s}$, we get the states S' where the robot will probably go from $\mathbf{s}$, and with that, the actual distribution per state $\mathbf{s}$. By fitting a probability distribution

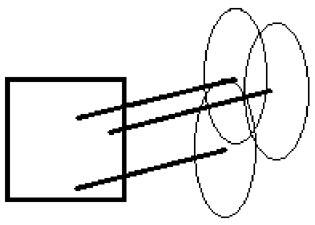


FIGURE 5.4: Distribution of next states for observed data points grouped in the same cell: the lines denote the $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$ of $\mathbf{a}$; the ellipsoids the distribution of next states from $\mathbf{s}$.

to known data points it is possible to estimate the probabilities of the possible transitions, also for unknown data points that did not occur in the training data. In order to fit distributions, we needed to assume a family of distributions for our data. The assumption of our data belonging to the family of Gaussian distributions is justified by the *Central Limit Theorem*, stating that many training data patterns can be viewed as prototype patterns corrupted by a large number of random processes, which often makes the Gaussian distribution a good model for the actual probability distribution. The reason for this is that, as the number of random variables increases, the distribution of their sum converges to the standard normal, Gaussian, distribution [34, 65].

In this case, because we are concerned with vectors instead of scalars, we need the *multivariate* normal distribution, that describes the Gaussian law in the k-dimensional Euclidean space. Instead of the usual Bell curve for univariate distributions, it gives us ellipsoids with the mean located in their centre and the variance determining their vertical and horizontal width (figure 5.4).

Even though we collected data from multiple starting positions and used a very coarse grid representation of the field, it is still possible that there is a grid cell for which we do not have training data to directly compute the average transitions from. For those we used the transition distributions of all other cells given that behaviour, yielding an average $\boldsymbol{\mu_a}$ and $\mathbf{\Sigma}$ for the behaviour on the whole field. The advantage of this method over a method that uses the Gaussian of a nearest known state (for example by calculating the k-Nearest Neighbours or Mahalanobis distance to all other states), is that it can be done during the preprocessing, so offline, and does not have to be computed anew for each input state and each possible neighbour.

5.5 Classification

To classify which behaviour is being executed based on an observation of state transitions, we use the multivariate normal probability density function [34] (eq. 5.2) to compute probabilities for each transition, given the mean and variance of a certain behaviour and grid cell.

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}} exp[-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^t\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})] \tag{5.2}$$

Substituting $\mathbf{x}$ for $\mathbf{s}'$ in 5.2 and using $\boldsymbol{\mu_a}+\mathbf{s}$ and $\mathbf{\Sigma}$ from the observed $\mathbf{s}$ conditioned on behaviour model $M_{\{opp,own\}}$ and grid cell g_j gives us the desired probability. The covariance matrix $\mathbf{\Sigma}$ should be positive definite (every column of the transpose of the matrix is positive) and the diagonal elements σ_{ii} are the variances of the respective x_i (i.e. σ_i^2), and the off-diagonal elements σ_{ij} are the covariances of x_i and x_j. Moreover, because of the independence of features, the covariance matrix $\mathbf{\Sigma}$ is diagonal, that is, non-zero elements only appear along the main diagonal of the matrix [52]. We take log-likelihoods because the probabilities tend to be very small, which would lead to numeric underflow if we want to multiply them to get likelihoods. Also, probabilites would need to be multiplied, whereas log-probabilities can be added to get log-likelihood, which is a computationally faster operation [52].

Summarizing, we were interested in the maximum likelihood of behaviour classes given observations: $argmax_b\, p(b|O) = argmax\ p(O|b)\ p(b)$. We have no knowledge about prior probabilities for our behaviour classes B. We could make them uniform, but this would be uninformal, so we can simply leave $P(b)$ out of this equation. The log-likelihoods $p(O|b)$ are computed by taking the logarithms of the probabilities of observation O of length $n \leq N$ (N being the total length of the trajectory), calculated using 5.2, and summing them.
Then the only step left to estimate which behaviour is more likely based on an observation is to compare the log-likelihoods of both behaviour classes and decide according to Bayes' rule:

$$\text{Decide}\ \ b_1\ \ \text{if}\ \ p(b_1|O) > p(b_2|O);\ \ \text{otherwise, decide}\ \ b_2. \tag{5.3}$$

5.5.1 Preliminary Results

This classifier was run on the validation set of the collected data, yielding a 91.6% and 44.4% precision for the *GoToOppGoal* and *GoToOwnGoal* behaviour classes respectively. These scores are retrieved when testing on complete trajectories, so observations of their total length N, in an offline manner. Note the rather large difference between the classes. We suspect the size of our data set for *GoToOwnGoal*, which was only 3/4 of the size of the data set for *GoToOppGoal*, might be the main cause of this difference. When looking more closely at the trajectories that were misclassified, several other characteristics might have contributed to that misclassification:

- The length of the trajectory: are shorter trajectories harder to classify? However, just in two of the six misclassified files, the length of the trajectory was considerably shorter than in the correctly classified ones.
- Mirrorring and coordinate jumps: despite of the preprocessing procedure, these issues could not be smoothed out entirely. Especially if a relatively long part (or multiple shorter parts) of a trajectory was logged incorrectly, the parts that were correct could not compensate for that large amount of noise. Although a lot of trajectories with similar issues (even after smoothing) were already left out of the training and validation sets, a few of the ones that were in that process considered acceptable were misclassified nonetheless.

However, it should be noted that in all of the misclassified *GoToOwnGoal*-trajectories, the difference between the log-likelihoods (which already translated to extremely small probabilities) for both behaviours is extremely small. Due to our Bayes' decision rule, we strictly choose the highest log-likelihood, but this might not be the best answer in these cases.

5.6 Possible Improvements

One way to improve this system is to collect more data to train and test on to get more reliable results. It would be interesting to see the system's performance on a ball-related behaviour, where the target position is not static with respect to the field, but dynamic. Since we already work with relative angles and distances, the approach would not be too different. We could also try using another field representation, for example a more fine-grained grid or a non-discretized field. An important improvement would be to test the system online instead of offline. We should be able to get the logs of the player real-time. If we amplify the current code with a belief update module, we would be able to get the coach's beliefs about the possible behaviours at any time. We could then test the effect of the length of a history on the coach's assessment.

Furthermore, the decision procedure, which might be too strict when dealing with these very small likelihoods, could also be adjusted. One solution is to add a third category for the event that the difference between the likelihoods is too small to make a strict distinction. It might be a good idea to include more or other distinguishing features. Alternatively, the decision procedure using Bayes' rule might be too naive, which calls for another approach to decide.

Chapter 6

Application and Conclusions

In order to extend the current RoboCup methodology into a more high-level reasoning direction, an agent organization framework has been designed for the robot soccer domain. In this final chapter, possible connections between the framework and the plan recognition module as a part of it are suggested. The work in this thesis is discussed and ideas for further research are provided.

6.1 Application

6.1.1 Coach and Plan Recognition

Starting from the current Edinferno methodology, both the coach role and the plan recognition module are entirely new additions. As argued, the coach is a kind of ad hoc agent who, instead of adapting his own behaviour, provides players in the team with strategic advice on how to adapt theirs. In order to improve teamwork and coordination, he should be able to identify currently executed player behaviours and decide what would be best for the team to do next. In this work, we focussed on the plan recognition part of that process - the decision making and adaptation challenges could cover several follow-up research reports. As described in the coach role table (3.5), we suggest the coach executes the plan recognition module as a sub-objective. The output of the module can function as the input for the subsequent modules that determine the role of the observed player, the formation of the entire team, the optimal tactic in this specific situation and the corresponding message to send. This is an abstract plan that can be implemented in several ways, which depends on the choice of the actual *agent* (in contrast to *organization*) designer. We suggest the following interpretations:

Tactics as independent sub-objectives

The tactic plans the coach can choose from are collected in a library, together with conditons and situations that determine which tactic is optimal in which situation. Optimality of a tactic can for instance be determined through a utility value for each situation-tactic pair (perhaps in a manner like [41]'s task-team utility). The tactics in this library are role-independent, meaning that they are not part of the general role descriptions and can in principle be executed by any role-enacting agent. They can, as such, be seen as contract clauses that can be fitted to a specific role-enacting agent in a certain situation. The difference with OperA's social contract instantiation would be that instead of negotiation about the contract, the coach's advice would be obligatory, leading to automatic adoption of that plan upon receiving the message. How the `rea` would then prioritize his original objectives and the new tactic is subject to discussion. In this situation, the coach does not need to know the player's roles since the tactics can be imposed on any role. The plan recognition module would serve as a means to learn the players' intended movements and decide the appropriate tactics based on that information combined with the overall game situation.

The advantage of this approach is that the players only have to trust the coach, who has a more complete view of the players' and game situation than the players themselves, due to their localization issues while moving. The disadvantage is that, by disregarding the differences in roles, the coach might decide for instance that all players should attack or that the goalkeeper is in the best position to go and block some opponent, leaving the goal open. Also, the coach could give advice that is inconsistent with the norms of the role-enacting agent, which would lead to violations and undesirable agent behaviour.

Tactics as dependent sub-objectives

Again, tactics are interpreted as sub-objectives in terms of the logic framework. In this case, the players' role specifications are already provided with subsets of the possible tactics in some sensible order or priority: these subsets are role-dependent in the sense that, for example, a goal-keeper can never get scoring-tactics. The coach uses the plan recognition module in combination with some plan-to-role mapping to learn which agent has which role (team formation) and with that, he knows which tactics they would be able to perform. The assumption that tactics depend on field positions and game situations still holds. Now, instead of assigning tactics to agents, the coach assigns agents to tactics: whenever a certain tactic would be optimal, but the `rea` whose role includes that tactic is not in the right position to perform it, the coach can re-assign roles to ensure that that tactic is performed by an agent that ís in the right position. If that agent already has the correct role, the coach simply gives him the tactic. This is an alternative to the dynamic, reactive role switching as it happens currently, relative to the position of the ball and otherwise performing some standard behaviours. However, we do not suggest to abandon that method, since it is likely that the coach's messages come too late or not at all. A robust, individual default behaviour should be at hand. This also explains why the coach needs to keep checking the team's formation instead of just keeping track of his own role assignments.

In comparison to the first approach, this one has the addition of the coach reassigning roles to fit the optimal tactics for their positions. This obviously is an advantage over the first approach, as with this addition consistency of norms and (sub-)objectives of the agent, his role and the new tactic will be maintained. In these first two interpretations we assume that the coach can assign roles and/or tactics to separate agents, but this might not be feasible due to the time it would take per player.

Team Tactics

In this last suggested interpretation, the coach again infers the players' roles and, with that, the team formation. Instead of single agent tactics, the library of plans contains tactics for the entire team (or a subset, e.g. only attackers). Based on the formation, game situation and possibly players' positions, the coach decides the optimal tactic and sends that to the team without specifying which role-enacting agent should perform should perform what specific sub-objective. That could be handled in a similar reactive way as the current role-switching: the agent in the appropriate position to do a certain sub-objective is to perform that sub-objective.

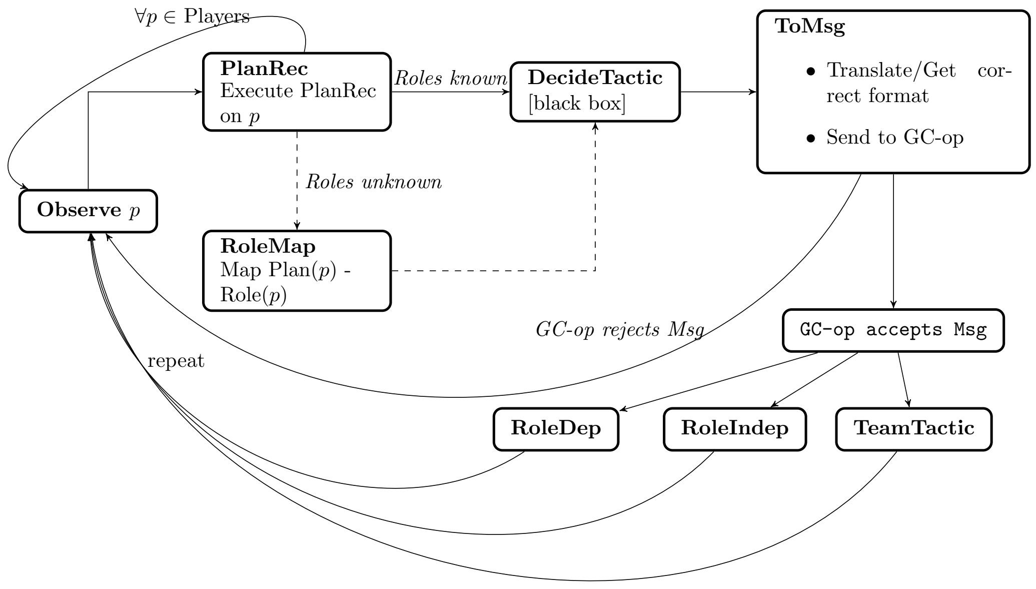
The advantage of this method is that if the coach fails for some reason, the team will continue as usual using their own tactics and role-switching methods, which makes it a robust approach. A possible disadvantage is that, depending on how this is implemented, the players could spend a long time on negotiating who plays what sub-objective of that tactic.

In all these, we assume the coach is aware of the game situation (through the general `inform`-actions of the GameController-operator or head referee), and that he can perceive the necessary (spatial) information to do plan recognition. We mentioned a plan-to-role mapping the coach could use to make the step from the low-level plan recognition output to the abstract level of roles and objectives. The idea of such a mapping is similar to the idea of libraries to match observations to, containing for each possible role a set of possible plans and conditions that the observed plans and conditions can be matched to. The goalkeeper role, for example, could correspond to plans that take place within his own penalty area or plans that involve diving for the ball.

In order to formally determine which agent enacts which role, a mapping or role-enacting function similar to the one in [25] can be defined, as it also includes mapping the agent's plans and objectives to those of his role. For P the set of role-enacting Players, a player is a tuple $< b, (\sigma), \Gamma, \Pi, \Omega, t >$: b is the behaviour as yielded by the plan recognition module, σ is the set of that player's beliefs (if he has any), Γ is his set of objectives, Π his set of plans (sub-objectives), Ω his norms and t the player type: in this case t is the set of robot roles, $Roles_R$ *without* the robot role 'coach' (see 3.2.2): $Roles_{R'} = \{\text{attacker, defender, goalkeeper}\}$. Let a role be tuple $< (\sigma_i), \gamma_i, \omega_i >$ and $r_i \in Rname$ a role instantiation name. Then, a mapping of Players $p \in P$ to roles $r \in Roles_{R'}$ would be a function $\mathcal{F} : P \times Roles_{R'} \times Rname \to P$:
$\mathcal{F}(< b, (\sigma), \Gamma, \Pi, \Omega, t >, < (\sigma_i), \gamma_i, \omega_i >, r_i) = < (\sigma \wedge \sigma_i), \Gamma', \Pi, \Omega', t >$. For notation and precise definitions, please see [25]. We included beliefs here in the BDI-sense and not as probabilities over an agent's behaviour, as used in 5. BDI beliefs might be included in future implementations.

In figure 6.1, a diagram for the coach robot is depicted where the three possible tactic decision modules are included in the loop.

Figure 6.1: Diagram showing the possible flows of the coach robot.



Another approach to plan recognition, which has shortly passed the review in 4.3, is plan recognition via *mental state abduction* (MSA) [88]. In this approach, the agents are assumed to be BDI-agents with internal states (belief/knowledge, desire/goals, intentions/plans), and it introduces an explanatory abstraction for reasoning about other agents' behaviour. The idea is similar to the numerical plan recognition method (5) in that a set of possible explanations is computed for an agent's observed behaviour, based on knowledge of its rules (in contrast to: computation of likelihood for each known behaviour/plan given observation). Also, is it assumed that behaviour is determined by the *role* of the agent, which provides a connection to the idea of roles in our OperA framework.

MSA uses 'answer set programming' for nonmonotonic, abductive reasoning (chapter 2.3.1) in the agent programming language (2)APL [26]. In APL, a goal achievement *rule* has the form $n : \gamma \leftarrow \beta \mid \pi$, meaning that the rule with identifier n is suitable for achieving goal γ if β is believed, in which case plan π can be chosen by the agent to execute. A plan generates an *observable sequence* of primitive actions α, $\delta ::= \alpha \mid \delta_1\delta_2$. Like in our current module, only complete observations are considered here (the observed sequence of actions must be the prefix of the sequence of some plan), but an extension to handle 'gaps' or otherwise incomplete observations is given in [87]. From these known agent rules and the observed action sequences, hypotheses can be abduced as explanations for that behaviour.

A distinction is made between (possible) observable actions $\mathbf{o}(\alpha, n)$ and actually observed (seen) actions $\mathbf{s}(\alpha, n)$: in order for an action to be seen, it must be observable. Rules state which actions are observable, that is, a plan generates observable sequences (OS). However, if a single plan can generate multiple OSs, then it also generates multiple *computation sequences* (CS). They can include *test actions* that should succeed before the actual action sequence of a plan can occur. These test actions can be seen as a form of belief/goal introspection: a test action $\mathbf{B}\phi?$ (ϕ is a proposition in the agent's proposition language) is evaluated with respect to the agent's beliefs and similar for goals. If the preconditions β, γ or a certain rule $n : \gamma \leftarrow \beta \mid \pi$ are met, the agent can perform the corresponding plan π of that rule. If we know the rule that generates some seen and observable actions, and we know that in order to be able to do those actions, certain test actions must have succeeded, then we also have information about the agent's mental state.

In order to implement this, the APL rules $\mathcal{R}$ are translated into a logical theory $\Theta_{\mathcal{R}}$, which is then translated into a logical program $\mathcal{P}_{\mathcal{R}}$ of Anser Set Programming. In first translation step, the following predicates are introduced [88]:

$\mathtt{r}(n, c)$: the agent applies rule n and performs CS c.
$\mathtt{b}(\psi)$: the agent's belief base entails ψ.
$\mathtt{g}(\psi)$: the agent's goal base entails ψ.

Observables (seen observations) and abducibles (possible explanatory hypotheses) are added such that all seen observations are facts in the answer set and that single distinct instances of rules are considered as explanations respectively (please see [88] for detailed proofs).

$\mathtt{o}(\alpha, n)$: action α is observable as the n'th action.
$\mathtt{s}(\alpha, n)$: action α is seen as the n'th action.

Clearly, *seen* actions must also be *observable* according to the theory. The translation from $\mathcal{R}$ to $\Theta_{\mathcal{R}}$ is then defined as follows:

$$\forall(n : \gamma \leftarrow \beta \mid \pi) \in R, \forall\pi' \in CS(\pi) : \mathtt{r}(n, \iota(\pi')) \rightarrow (\mathtt{g}(\tau(\gamma)) \wedge \mathtt{b}(\tau(\beta))) \in \Theta_R \tag{6.1}$$

Subsequently, Θ_R can be translated into an logic program P_R:

$$\phi \rightarrow (\psi_1 \wedge \cdots \wedge \psi_n) \in \Theta_R \Longrightarrow n\{\psi_1, \ldots, \psi_n\}n : -\phi \in P_R \tag{6.2}$$

It is explicitely assumed that observed actions stem from a single computation sequence of some plan that belongs to a single rule: a single rule has one single plan, but that plan can give rise to multiple observation and computation sequences. This represents the notion that one plan can be executed in different ways.

The connection to our framework is as follows: MSA rules can be seen as the plans of our agents' roles (the actions/behaviours executed as a consequence of (sub-)objectives and norms), where goals γ are the (sub-)objectives. Beliefs β are not really included as such currently, although one could argue that the agent's knowledge of the environment (via ontologies and sensory input) are 'beliefs' in some sense. The observed action sequences as considered in the current plan recognition module are the actual steps our robot takes in a certain direction, forming a trajectory towards its goal. A simplified example in case of our goalkeeper role, assuming he's implemented as a BDI-agent, would be as follows. Consider the rule $R = \{1$: `hold-ball` $\leftarrow$ `in-ownPA(g) and in-ownPA(b)` $\mid$ `move(g,b); if B(opponent-near) then pickup(g,b) else skip`$\}$, which would lead to the answer set program P_R:

```
2{g(hold-ball,0), b(conj(in-ownPA(g), in-ownPA(b)),0)}2 :- r(1,1)
```

```
2{g(hold-ball,0), b(conj(in-ownPA(g), in-ownPA(b)),0)}2 :- r(1,2)
                         2{o(move(g,b),1), pickup(g,b),2)}2 :- r(1,1)
                                          1{o(move(g,b),1)}1 :- r(1,2)
                           1{r(1,1), r(1,2)}1 :- s(A,T), not o(A,T),
```

where the last statement says that candidate answer sets that were seen, but not deemed observable at step `T` should be discarded. The reason that `r(1,1)` and `r(1,2)` are the same, is because the plan of this rule has two possible computation sequences, represented by the two possible observations given (depending on the belief of 'opponent-near').
Let's say we've seen the goalkeeper moving towards the ball: $P' = P \cup \{$`s(move(g,b),1)`$\}$. This can be explained by both `r(1,1)` and `r(1,2)`. Next, we see him picking up the ball: $P'' = P' \cup \{$`s(pickup(g,b), 2)`$\}$; this can only be explained by `r(1,1)`. We can now infer $P'' \models$ `goal(hold-ball)`$\wedge$`bel(conj(in-ownPA(g), in-ownPA(b)))`$\wedge$`bel(opponent-near)`, revealing the mental state of our goalkeeper based on observed actions.

As shown, when extending our logic-based framework with logic-based BDI-agents, for example programmed in an agent programming language like 2APL, we can formally infer agents' plans using abductive reasoning on observed actions. This approach can be considered as the logical alternative to the numerical plan recognition module developed for this thesis. It depends on the choice of agent design which of these would be more appropriate.

6.1.2 General Framework Application and Discussion

Since OperA's methodology yielded an organizational framework wherein actual agents are not specified, our work should be extended with implementable agent designs. These already exist in the form of the currently functioning soccer team. How to connect the two or how to implement the framework on those agents is subject for further research; the BDI-agents suitable for MSA as introduced above are one possible option. We've looked into some related work for some inspiration.

A more recent extension of OperA called OMNI [32] seems to be an option for further research. According to [4], OMNI translates the norms from an abstract to a procedural level, making the social norms implementable. This approach of extending the organizational framework into implementable methods is one way to proceed; another way would be to investigate agent designs suitable for implementation within the framework. An example of the latter could be Bellifemine's JADE agent middleware framework, which provides a general agent model allowing for the addition of more specific agent models like BDI-architectures or reactive Finite State Machine-like architectures like the current implementation [12, 77]. An advantage of JADE is that is suitable for agent communication via an ACL; a disadvantage in terms of the current methodology is that it's developed for Java and not `C++`, although that might not be a problem. Another motivation would be that Virginia Dignum used JADE to implement a part of her OperA prototype as well [31](chapter 7). BDI-agents as presented in the previous section is another option for agent designs.

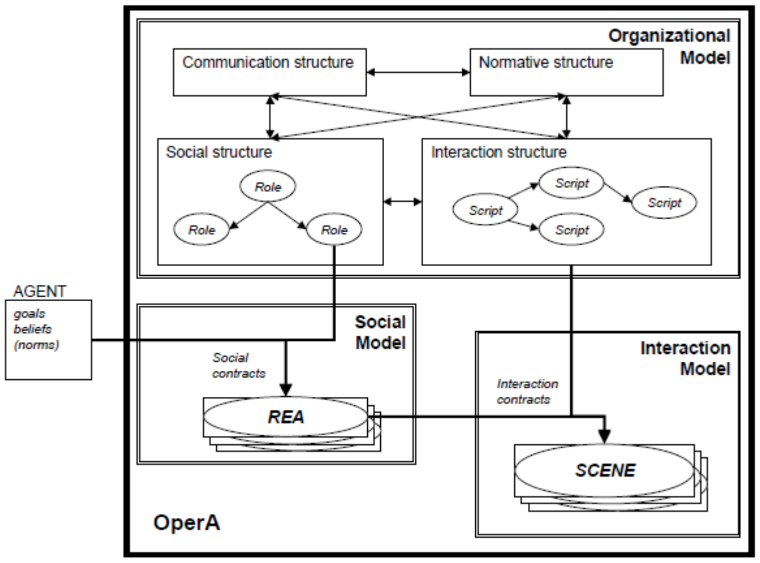


FIGURE 6.2: OperA's architecture (source: [31], chapter 3). The coach and other roles fit in the social structure of the OM.

In general, applying a MAS framework requires three steps [74]: problem analysis, architecture modelling and agent modelling. The first two are provided in this work, only agent modelling remains. Agent models as described by Park correspond largely to the roles of the OperA framework, containing goals (objectives), plans and 'beliefs' (knowledge about the environment and the agent itself - as provided by sensory information and a shared ontology). Agents can for example be implemented using an Agent Programming Language like GOAL or 2/3APL [26, 31]. Instead of designing entirely new agents, another option is to extend the current player implementations with the OperA libraries for roles, norms and the shared ontology. Checking conditions that would determine role switches or that trigger revision of priorities within norms or plans can be handled in a similar finite state machine style as the current behaviour switches.

Coordination in the robot soccer society can be divided into a human part and a robot part. The human part is based mainly on the functionality of the head-referee and is described in detail in our framework. The robots' coordination however is only partly organizational: this is the normative part as described in the framework. Actual interactions between the robots still need to be defined within their roles and internal architecture. The boundaries for these interactions are given in the framework, but specific interaction strategies still need to be investigated.

As for communication, there are two lines of communication within the robots of our society: one between players, one from the coach to the players. For the coach, a strict protocol is provided with requirements for his messages (appendix B) [22]. Messages between the players simply contain their 'beliefs' about their position (relative to the ball); since they have the same internal representation for those beliefs, they can communicate in that exact format. A more sophisticated method of communication could be devised, however, as all communication is handled via a limited WiFi connection, it would be best to keep it 'light' and simple.

There are two drawbacks to the OperA methodology with respect to the application in this work: a corresponding agent design method and general implementation guidelines are not included. However, OperA has proven to be applicable to our mixed, dynamic human-robot society which yielded a formal conceptual model of the system from an agent organizations point of view. This model should be extended with agent designs, for example adjustments of the current implementations. Developing means to implement an OperA framework was already suggested as follow-up work in [31]. Meanwhile, a design tool by the name of OperettA[1] has been developed [71]. Besides the design and verification of OperA models, this tool is also said to generate simulations of systems built with it. Still, this is not the same as actual, physical implementation in our robot soccer team: this remains a challenge for future work.

6.2 Conclusion

Let's look back at the research questions posed at the beginning of this work:

- How to model an abstract agent organizations framework for the human-robot soccer domain using OperA architecture?
- How to do plan recognition on robot soccer players from visual and numerical information only?
- How to connect the framework and plan recognition via the ad hoc agent or coach?

These questions have been researched in the corresponding three parts: chapters 2 and 3 about agent organization frameworks and our own newly developed robot soccer society framework in OperA's architecture, chapters 4 and 5 on plan recognition as part of the coach role in an attempt at ad hoc coordination and the link between those parts in this chapter, by means of suggestions on the application of the framework and the plan recognition module to the current robot soccer methodology.

In answering the first question, we explored [31]'s OperA framework design methodology for agent organizations, based on deontic temporal logic and with a focus on coordinated interaction and roles. We choose to apply such a method of Multi-Agent Systems theory, and OperA's organizational point of view specifically, because our system, consisting of both humans (referees, human teammembers) and robots (players and the coach) in the regulated domain of soccer, is a suitable application for such an approach. We used the descriptions of the RoboCup regulations [22] and took inspiration from Edinferno's current player implementation and the team's advice in the design of our organizational framework, following [31]'s examples and explanations to develop a very elaborate Organizational Model 3.2.2. The Social and Interaction Model (3.3, 3.4) depend largely on specific agent designs, which is why they couldn't be fully developed at this stage, but the means to continue and verify those models have been provided.
OperA proved to yield an expressive and detailed framework which allows for coordinated interaction between its role-enacting agents while being flexible enough that we believe it would be possible and indeed useful to extend it into the current robot player designs. The fact that OperA itself did not provide the means to design agents seemed a disadvantage at first; however,

[1]http://www.cs.uu.nl/research/projects/opera/

it actually is an advantage as it makes our model reusable (within the RoboCup SPL), flexible (different agent designs can be integrated) and dynamic (the individual goals of agents can be considered, new interactions can be added or current interaction scenes adjusted to the agent's needs).

In order to answer the second question we choose to make use of the information the robots can give by logging perceived information. The plan recognition problem is approached from a probabilistic machine learning point of view, using training data and a Markov chain to model the problem of identifying the most likely behaviour that could cause the observed trajectory. A coarse grid representation of the field, in combination with some loss of training trajectories due to the unsteady self-localization technique, might have caused a slightly inaccurate distribution of the robot's movement per behaviour. Also, we only tested with static goals, while it would be more interesting to infer plans concerning the ball.
In this first attempt at behaviour recognition, we classified the 'GoToOpponentGoal' behaviour with 91.6% precision and the 'GoToOwnGoal' behaviour with only 44.4%. As discussed in 5 this is probably due to the difference in training set size. However, as the actual difference in log-likelihoods between correctly and incorrectly classified trajectories was very small, it might also be that Bayes' decision rule, which we used to classify, is too naive for this task.
As this was just a proof-of-concept module to obtain better insight and understanding of the problem and to provide a basis for further research into ad hoc coordination, we have several suggestions for the future. For our module specifically, we suggest adjusting it to work in an online rather than the current offline fashion. Furthermore, this method can also be applied to ball-related behaviour if training data could be collected for those behaviours. We actually begun collecting such trajectories, but due to some hardware problems the collected data was never used in the tests. Other adjustments could be a different classification rule, a finer field representation and simply more training data. However, in the bigger picture, it might be better to change the approach alltogether since it is based entirely on the robot's logs: these will not be available to the coach in the actual ad hoc setting. It depends on the way in which the coach would then collect information of the players whether or not the current module can be used still. For example, if the coach can translate the information he receives purely from his own visual feedback into the format of the currently used numerical features, our module will still be relevant. As there are many different ways of collecting information and processing it, it really depends on how the other, to be developed, coach modules function whether or not this one still applies.

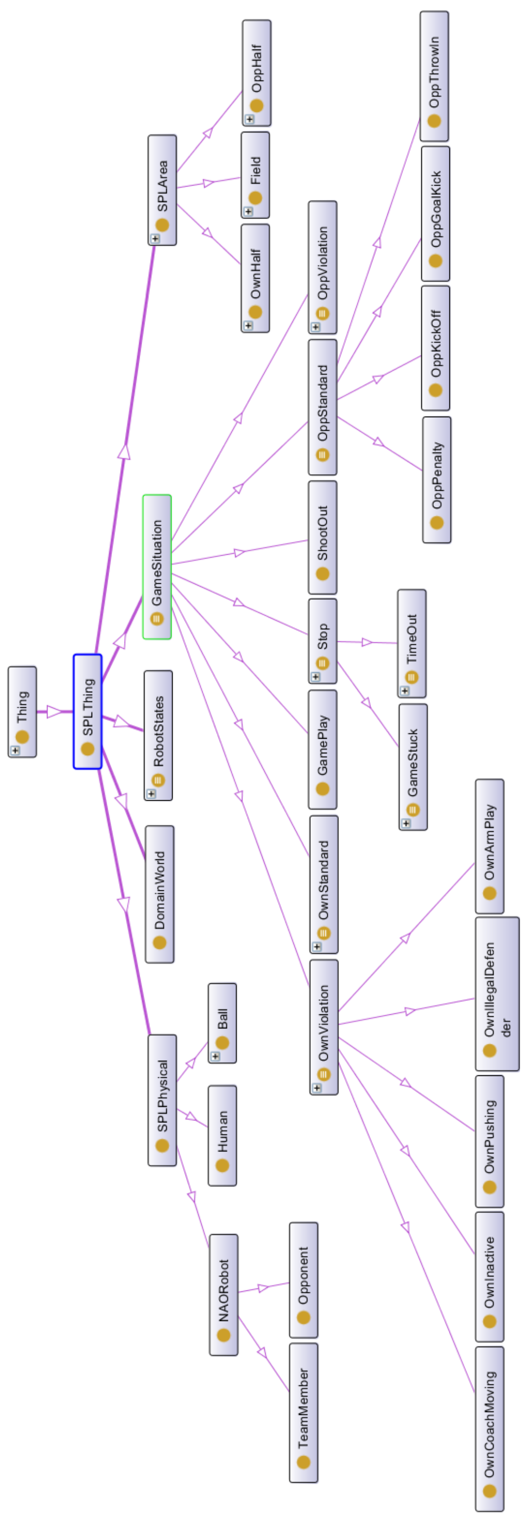
The last question, aiming to connect the plan recognition to the framework via the coach, yielded the suggestions in the first sections of this chapter. We proposed and compared three possible ideas for further work on the coach. These ideas deal with the situation after a coach message has been accepted by the GameController-operator, sketching how that message could influence the roles and plans of the team. As in the answer to the previous question, this largely depends on how the coach and the players will be implemented, but we've compared the ideas on a conceptual level.
Furthermore, we introduced an alternative method of plan recognition that would be an appropriate bridge between the logical framework and the agents if they would be implemented as BDI-agents. Using abductive reasoning in an agent language like 2APL for BDI-agents, mental

state ascription seems to be a connection well worth exploring further. Major advantages are that an extension to the method in [88] to handle incomplete observations already exists [87] and that a clear and complete description of how to implement it are provided. Futhermore, it fits the desire to amplify the current robots with reasoning methods in order to try to make their behaviour more human-like. Due to the highly regulated character of robot soccer, where all the rules are known across the team, exploring the possibilities of intention recognition via abduction seems a promising next step.

Overall, we can say the OperA framework is indeed suitable for modelling the robot soccer society. Agent design remains an open question, but OperA is flexible anough to allow different approaches for this. We gained a deeper understanding of the needs and characteristics of the robot soccer system, interaction and coordination between robots and methods of plan recognition specifically. Some suggestions for further work in connecting plan recognition to this abstract framework and implementing agent modules to handle more ad hoc and human-like coordination methods have been presented.

Appendix A

SPL Domain Ontology Graph



Appendix B

Role Tables

Notation: parameters written with a capital are variables that can be filled in; for example, 'Players' means that one of the elements of the set Players can be filled in in that spot. Parameters in lower case are already filled in instances (like 'field' refers to the one and only possible instance of the Field area).
Time parameters are defined whenever the RoboCup-rules stated timing specifics; how to implement such constraints is dependent on the agent design.

The sets mentioned in these tables are defined as follows:
ClockState = {timed-penalty-ended, timed-penalty-started, timed-penalty-10sec-left, half-game, end-game, Minutes[0,...,30]}
RobotState = {ready, set, playing, penalized$_r$, finished}
GameSituation = {KickOff, ShootOut, GoalSaved, GoalScored, GameStuck, Stoppage, {Violations}} (please see Appendix D)
Violations = {Locomotion, BallHolding, Jamming, IllegalDefender, Pushing, ArmPlay, LeavingField, DamagingField, CoachMotion, Inactive} (please see Appendix D)
Request = {pickup(Robots), timeout}
Penalty = {standardRemovalPenalty, pickup-removal-noreturn, disqualification}
Stoppage = {globalGameStuck, localGameStuck, GoalScored}
Msg-Requirements = [human-readable, noNumbers, size$\leq$20bytes, time-since-last-msg$\geq$10sec]
Note that all msg-requirements should be met, whereas from the other here defined sets, only one element per instance can be the case.

Tactics and TeamTactics are part of the implementation; they are used here in the example of how to possibly work with a coach.

Role: Coach	
Role id	coach

Objectives	o1 := messaged-tactics
	o2 := followed-rules

Sub-objectives	Πo1 = ({∀p∈ Players: executed-plan-rec-module(p, role(p), t), got-plan(p, plan)), got-tactic-list(plan, formation, Tactics), decided-tactic(Tactics, tactic),got-msg(tactic, msg), message-sent(coach, GC-op, msg), wait(10s)} Πo1' =({∀p∈ Players: executed-plan-rec-module(p,t), got-role-map(plan(p), role(p))), got-formation-map(role(p), Formations), got-team-tactics(formation, TeamTactics), decided-tactic(TeamTactics, tactic), got-msg(tactic, msg), message-sent(coach, GC-op, msg), wait(10s)}
Rights	message-via-GC-op, decide-tactic(coach, (Team)Tactic)

Norms	PROHIBITED(coach, move(¬(head∧arms)))
	PROHIBITED(coach, communicate(coach, Robots, direct))
	PERMITTED(coach, have-clothes(anyColor, anyPattern))
	OBLIGED(coach, meet-msg-requirements(Msg, [Msg-Requirements]))

Type	operational

Table B.1: Role definition for Coach; t = window of observation, msg = message. subobjectives o1 are assuming that the coach knows the roles and formations of all players; o1' are assuming he has to map those first, according to the plan he recognizes. Please note that these subobjectives are just conceptual, to convey what could be desirable states in order to achieve the 'messaged-tactics' state eventually. Precise plans and their implementation should be specified on agent design level.

Role: Goalkeeper	
Role id	goalkeeper

Objectives	o1 := defended-goal(ownGoal)
	o2 := followed-rules

Sub-objectives	Πo1 = { in(goalkeeper, (ownPenaltyArea ∨ ownGoalArea)),
	adaptedPosition(ownPos, ballPos, estimatedBallPos) }

Rights	in(goalkeeper, ownPenaltyArea), in(goalkeeper, ownGoalArea), hold(goalkeeper, Ball)

Norms	OBLIGED(goalkeeper, have-clothes(teamColor, teamPattern))
	IF DONE(goalkeeper, receive(goalkeeper, coach, Msg))
	THEN OBLIGED(goalkeeper, apply(Msg))
	$\forall i \in Reas$: IF DONE(i, in(Ball, ownPenaltyArea)
	AND DONE(goalkeeper, in(goalkeeper, ownPenaltyArea))
	THEN PERMITTED(goalkeeper, hold(ball, <10s))
	PROHIBITED(goalkeeper, move(goalkeeper, ¬(bipedal ∧ human-like))

Type	operational

Table B.2: Role definition for Goalkeeper

Role: Attacker	
Role id	attacker

Objectives	o1 := scored-goal(attacker, oppGoal)
	o2 := helped(attacker, FieldPlayers)
	o3 := followed-rules

Sub-objectives	Πo1 = { gained-ballPossession(Attacker, Ball),
	executed-default-attack(direction(oppGoal), velocity(fast)),
	avoided-collision(attacker, (Players∧Opponents)),
	is-near(attacker, oppPenaltyArea),
	kicked(attacker, Ball, oppGoal) }
	Πo2 ={$\exists f \in$ FP: is-near(f, Ball), is-near(attacker, Opponent),
	blocked-player(attacker, Opponent)}

Rights	

Norms	OBLIGED(attacker, have-clothes(teamColor, teamPattern))
	IF DONE(attacker, receive(attacker, coach, Msg)
	THEN OBLIGED(attacker, apply(Msg))
	IF DONE(goalkeeper, in(goalkeeper, ownPenaltyArea))
	THEN PROHIBITED(attacker, in(attacker, ownPenaltyArea))
	$\forall p \in Players$: IF DONE(p, ¬in(p, ownPenaltyArea))
	THEN PERMITTED(attacker, in(attacker, ownPenaltyArea)) }
	PROHIBITED(attacker, hold(Ball, ≥0s))
	PROHIBITED(attacker, move(attacker, ¬(bipedal ∧ human-like))
	PROHIBITED(attacker, damage(Field))
	PROHIBITED(attacker, leave(Field))
	PROHIBITED(attacker, push(attacker, Players∨Opponents))

Type	operational

Table B.3: Role definition for Attacker. The default attack sub-objective corresponds to the 'dribble' state of Edinferno's striker. Bear in mind that subobjectives are not plans since they don't consider timing; these predicates depict desirable substates.

Role: Defender	
Role id	defender

Objectives	o1 := helped-defend-goal(defender, ownGoal)
	o2 := blocked-player(defender, Opponents)
	o3 := followed-rules

Sub-objectives	Πo1 = {is-near(defender, ownPenaltyArea),
	gained-ballPossession(defender, Ball),
	blocked-player(defender, Opponents) }

Rights	

Norms	OBLIGED(defender, have-clothes(teamcolor, teampattern))
	IF DONE(defender, receive(defender, coach, Msg))
	THEN OBLIGED(defender, apply(Msg))
	IF DONE(goalkeeper, in(goalkeeper, ownPenaltyArea))
	THEN PROHIBITED(defender, in(defender, ownPenaltyArea))
	$\forall p \in Players$: IF DONE(p, ¬in(p, ownPenaltyArea))
	THEN PERMITTED(defender, in(defender, ownPenaltyArea)) }
	PROHIBITED(defender, hold(Ball, ≥0s))
	PROHIBITED(defender, move(defender, ¬(bipedal ∧ human-like))
	PROHIBITED(defender, damage(Field))
	PROHIBITED(defender, leave(Field))
	PROHIBITED(defender, push(defender, Players∨Opponents))

Type	operational

Table B.4: Role definition for Defender. 1 FieldPlayer (Defender or Attacker) may defend the goal from ownPenaltyArea iff otherwise the area would be empty (the goalkeeper is out and no other FP is in).

Role: Human-Teammember	
Role id	h-tm

Objectives	o1 := maintained-robots
	o2 := followed-rules

Sub-objectives	Πo1 = { ∀b ∈ Robots: check(b, broken(b)∨penalized(b)∨inactive(b)),
	request(h-tm, h-ref, pickup(b)), repair(h-tm,b) }

Rights	forfeit, request(h-tm, h-ref, TimeOut), request(h-tm, h-ref, pickup(Robots))

Norms	OBLIGED(h-tm, finish(TimeOut, <5min))
	IF DONE(h-ref, decide-stoppage(Stoppage, begin)
	∧ ¬DONE(h-tm, request(h-tm, h-ref, TimeOut))
	THEN PERMITTED(h-tm, request(h-tm, h-ref, TimeOut), t)
	IF DONE(h-tm, finish(h-tm, TimeOut, ≤5min)) ∨
	DONE(h-ref, inform(h-ref, society, decide-stoppage(Stoppage, end)))
	THEN OBLIGED(h-tm, ready-to-play(r, ≤2min))
	$\forall b \in \mathit{Robots}$: IF robotState(b, 'playing')
	THEN [PROHIBITED(h-tm, communicate(h-tm, b,
	{wireless, acoustic, visual}))
	AND PROHIBITED(h-tm, touches(h-tm, b))]
	PERMITTED(h-tm, request(h-tm, h-ref, pickup(Robots)))

Type	operational

Table B.5: Role definition for Human-Teammember.

Role: Assistant-Referee	
Role id	a-ref

Objectives	o1 := apply-request(Applicant, Request)
	o2 := apply-penalty(Robots, Penalty)

Sub-objectives	
Rights	handle-robots (e.g. place(a-ref, $< \mathit{Robots}, \mathit{Ball} >$, $< \mathit{Pos}, \mathit{Area} >$))

Norms	IF DONE(h-ref, request(h-ref, a-ref, apply-request(Applicant, Request)))
	THEN OBLIGED(a-ref, apply-request(Applicant, Request))
	IF DONE(h-ref, request(h-ref, a-ref, apply-penalty(Robots, Penalty)))
	THEN OBLIGED(a-ref, apply-penalty(Robots, Penalty)))
	OBLIGED(h-ref, have-clothes({black,blue}, noPattern))

Type	institutional

Table B.6: Role definition for Assistant-Referee.

Role: Head-Referee	
Role id	h-ref

Objectives	o1 := *handle(robot-acceptance(Robots, {removal, return}))*
	o2 := *handle*(penalty-decision(Robots, Violation, Penalty))
	o3 := *handle*(request-decision(Applicant, Request, {accept, reject}))
	o4 := keep-time(shootout-time)

Sub-objectives	Πo2 = {∀b ∈ Robots: checked(b, Violation),
	decided-penalty(b, Violation, Penalty),
	informed(h-ref, society, Decision),
	requested(h-ref, a-ref, apply-penalty(b, Penalty)}
	Πo3 = {received(h-ref, Applicant, Request),
	decided-request(Applicant, Request, {accept, reject})
	informed(h-ref, society, Decision),
	requested(h-ref, a-ref, apply-request(Applicant, Request))}

Rights	decide-penalty, decide-request, decide-stoppage(Stoppage, {begin, end})

Norms	IF DONE(h-ref, decide-penalty(Robots, Violation, Penalty))
	THEN OBLIGED(h-ref, (inform(h-ref, society,
	decide-penalty(Robots,Violation,Penalty))
	∧ request(h-ref, a-ref, apply-penalty(Robots, Penalty)))
	IF DONE(Applicant, request(Applicant, h-ref, Request))
	THEN OBLIGED(h-ref, decide-request(Applicant, Request, Decision))
	IF DONE(h-ref, decide-request(Applicant, Request, Decision))
	THEN OBLIGED(h-ref, inform(h-ref, society,
	decide-request(Applicant, Request, Decision)))
	IF DONE(h-ref, decide-request(Applicant, Request, 'accept'))
	THEN OBLIGED(h-ref, request(h-ref, a-ref,
	apply-request(Applicant, Request)))
	IF DONE(h-ref, decide-stoppage(Stoppage, {begin, end}))
	THEN OBLIGED(h-ref, inform(h-ref, society,
	decide-stoppage(Stoppage, {begin, end})))
	OBLIGED(h-ref, inform(h-ref, society, gameSituation(GameSituation)))
	OBLIGED(h-ref, keep-time(shootout-time))
	OBLIGED(h-ref, have-clothes({black,blue}, noPattern))

Type	institutional

TABLE B.7: Role definition for Head-Referee. Applicant usually is a Human-Teammember. The same norms apply for own and opponent team.

Role: GameController-operator	
Role id	GC-op

Objectives	o1 := manage-game-clock
	o2 := *handle*(communication(RobotState, Robots))
	o3 := *handle*(communication(Coach-Message, Players))

Sub-objectives	Πo1 = {adjusted-clock(ClockState), informed(GC-op, society, ClockState) }
	Πo2 = {informed(h-ref, society, RobotState),
	adjusted-Robotstate(RobotState), informed(GC-op, Robots, RobotState)}
	Πo3 ={check-coach-message(Msg, [Msg-Requirements])
	[∀p ∈ Players: inform(GC-op, p,Msg)
	OR drop(GC-op, Msg)] }

Rights	check-coach-message, adjust-clock

Norms	IF DONE(GC-op, check-coach-message(Msg, [Msg-Requirements]))
	∧ [Msg-Requirements] =*true*
	THEN OBLIGED(GC-op, inform(GC-op, Players, Msg)
	∧ display(GC-op, Msg))
	IF DONE(GC-op, check-coach-message(Msg, [Msg-Requirements]))
	∧ [Msg-Requirements] =*false*
	THEN OBLIGED(GC-op, drop(GC-op, Msg))
	OBLIGED(GC-op, check-coach-message(Msg, [Msg-Requirements]))
	OBLIGED(GC-op, inform(GC-op, Robots, RobotState))
	IF DONE(h-ref, inform(h-ref, society, RobotState(”playing”, ≥10sec))
	THEN OBLIGED(GC-op, inform(GC-op, h-ref, ball-in-play))
	OBLIGED(GC-op, adjust-clock(ClockState))
	OBLIGED(GC-op, inform(GC-op, society, clockState(ClockState)))
	OBLIGED(GC-op, count-aloud(last-5sec, {half-game, end-game}))

Type	institutional

Table B.8: Role definition for GameController-operator (which can be enacted by a human team member; who is not currently enacting the Human-Teammember role).

Role: Goalkeeper (FORMAL)	
Role id	goalkeeper

Objectives	o1 := defend-goal(ownGoal)
	o2 := follow-rules

Sub-objectives	Πo1 = { in(goalkeeper, (ownPenaltyArea ∨ ownGoalArea)),
	adaptPosition(ownPos, ballPos,estimatedBallPos) }

Rights	in(goalkeeper, ownPenaltyArea), in(goalkeeper, ownGoalArea), holds(goalkeeper, Ball)
Norms	$O_{goalkeeper} have\text{-}clothes(teamColor, teamPattern)$ $D_{goalkeeper} receive(goalkeeper, coach, Msg)$ $\rightarrow O_{goalkeeper} apply(Msg)$ $\forall i, i \in Reas : (D_i in(Ball, ownPenaltyArea)$ $\wedge D_{goalkeeper} in(goalkeeper, ownPenaltyArea))$ $\rightarrow P_{goalkeeper} hold(ball, < 10s)$ $F_{goalkeeper} move(goalkeeper, \neg(bipedal \wedge human\text{-}like))$
Type	operational

Table B.9: Role definition for Goalkeeper

Appendix C

Scene scripts & Structures

The scene scripts in this appendix are the ones that can be formalized in general, on facilitation level, without restricting implementation details. Any specifics are by means of example and can be adjusted to fit the ideas of further design.

Structure 1: penalties

Interaction scene:penalty-decision	
Description	a norm violation occurred, h-ref decides a penalty
Roles	h-ref(1), Robots($\geq$1), a-ref(1)
Results	r1: DONE(h-ref, decide-penalty(h-ref, Robot, Violation, Penalty)) r2: DONE(h-ref, request(h-ref, a-ref, apply-penalty(Robot, Penalty)))

Patterns	r1:{ $\forall b \in Robots; \forall n \in NormLibrary$:
	DONE(h-ref, check(h-ref, b, follow-rules(b,n))
	$\wedge$ $\exists n1 \in NormLibrary$: DONE(b, $\neg$follow-rules(b, n1)),
	BEFORE DONE(h-ref, decide-penalty(b, Violation(b, n1), Penalty)) }
	r2: {DONE(h-ref, request(h-ref, a-ref, apply-penalty(b, Penalty))) }

Norms	IF decide-penalty(Robots, Violation, Penalty)
	THEN OBLIGED(h-ref,
	[inform(h-ref, society,decide-penalty(Robots, Violation, Penalty))
	$\wedge$ request(h-ref, a-ref, apply-penalty(Robots, Penalty))])

Interaction scene: apply-penalty	
Description	after a h-ref penalty decision, the a-ref applies the decision
Roles	h-ref(1), a-ref(1), Robots($\geq$1)
Results	DONE(a-ref, apply-penalty(Robots, Penalty))

Patterns	{ $\forall b \in Robots; \forall p \in Penalties$:
	DONE(h-ref, request(h-ref, a-ref, apply-penalty(b, p)))
	BEFORE DONE(a-ref, apply-penalty(b, p)) }

Norms	IF DONE(h-ref, request(h-ref, a-ref, apply-penalty(Robots, Penalty)))
	THEN OBLIGED(a-ref, apply-penalty(Robots, Penalty))

Structure 2: requests

Interaction scene: maintain-robots	
Description	a robot needs to be repaired; h-tm requests pickup
Roles	h-tm(1), Robots($\geq$1), h-ref(1)
Results	r1: DONE(h-tm, h-ref, request(pickup(Robots))) r2: DONE(h-tm, repair(h-tm, Robots))
Patterns	{ $\forall b \in$ *Robots*: (ALWAYS) DONE(h-tm, check(b, (broken(b) $\vee$ penalized(b) $\vee$ inactive(b)))) AND [$\forall b \in$ *Robots*: broken(b) $\vee$ penalized(b) $\vee$ inactive(b)] DONE(h-tm, request(h-tm, h-ref, pickup(b))) BEFORE [DONE(h-ref, decide-request(h-tm, pickup(b), 'accept')) BEFORE DONE(h-tm, repair(h-tm, b))] OR [DONE(h-ref, decide-request(h-tm, pickup(b), 'reject'))] }

Norms	PERMITTED(h-tm, request(h-tm, h-ref, pickup(Robots))
	FORBIDDEN(h-tm, interfere(h-tm, Robots))
	IF DONE(h-tm, request(h-tm, h-ref, Request))
	THEN OBLIGED(h-ref, decide-request(h-tm, Request, {accept, reject}))

Table C.1: In order to maintain robots, the human team member should check at all times if they need repairing, and if they do, request a pickup. If accepted, they can repair, if not, the game continues. NB: repairing can only happen after the request-decision and apply-request scenes have been played.

Interaction scene:request-decision	
Description	a request is made, h-ref decides whether to accept or not
Roles	h-tm(1), h-ref(1)
Results	r1: DONE(h-ref, decide-request(h-tm, pickup(Robots), {accept, reject})) r2: (if 'accept':) DONE(h-ref, request(h-ref, a-ref, apply-request(Applicant, Request)))
Patterns	r1: { $\exists b \in Robots$: DONE(h-tm, request(h-tm, h-ref, pickup(b))) BEFORE [DONE(h-ref, decide-request(h-tm, pickup(B), 'accept')) BEFORE DONE(h-ref, request(h-ref, a-ref, apply-request(h-tm, pickup(b)))) AND DONE(h-ref, inform(h-ref, society, decide-request(h-tm, pickup(b), 'accept')))] OR [DONE(h-ref, decide-request(h-tm, pickup(b), 'reject')) AND DONE(h-ref, inform(h-ref, society, decide-request(h-tm, pickup(b), 'reject')))]}

Norms	IF DONE(h-ref, decide-request(Applicant, Request, 'accept'))
	THEN OBLIGED(h-ref, request(h-ref, a-ref, apply-request(Applicant, Request)))
	IF DONE(h-ref, decide-request(Applicant, Request, Decision))
	THEN OBLIGED(h-ref, inform(h-ref, society,
	decide-request(Applicant, Request, Decision)))
	IF DONE(Applicant, request(Applicant, h-ref, Request))
	THEN OBLIGED(h-ref, decide-request(Applicant, Request, Decision))

Table C.2: If the h-ref is requested something, he has to decide about it, inform the society about his decision, and if he decides to accept the request he should allow the a-ref to handle it. Example here is the pickup of a certain robot r.

Interaction scene: apply-request	
Description	when h-ref decides to accept a (pickup) request and asks a-ref to handle it
Roles	h-ref(1), a-ref(1), Robots(≥1)
Results	DONE(a-ref, apply-request(h-tm, pickup(Robots)))
Patterns	{ $\exists b \in Robots$:DONE(h-ref, request(h-ref, a-ref, apply-request(h-tm, pickup(b)))) BEFORE DONE(a-ref, apply-request(h-tm, pickup(b)))) }

Norms	IF DONE(h-ref, request(h-ref, a-ref, apply-request(h-tm, pickup(Robots))))
	THEN OBLIGED(a-ref, apply-request(h-tm, pickup(Robots)))
	IF DONE(h-ref, decide-request(Applicant, Request, 'accept'))
	THEN OBLIGED(h-ref, request(h-ref, a-ref, apply-request(Applicant, Request)))

Table C.3: An Assistant Referee handles the request that he is requested to apply by the Head Referee.

Structure 3: coach communication

Interaction scene: message-tactics	
Description	coach decides and sends (team) tactics through GC-op
Roles	coach(1), GC-op(1), Players($\geq$1)
Results	DONE(coach, send-message(coach, GC-op, Msg))
Patterns	r1: $\forall p \in Players$:[DONE(coach, execute-plan-rec-module(p, role(p), t))) $\wedge$ DONE(coach, get-plan(p, plan))] BEFORE [DONE(coach, decide-tactic(Tactics, tactic)) $\wedge$ DONE(coach, get-msg(tactic, msg))] BEFORE DONE(coach, send-message(coach, GC-op, msg)) r1': $\forall p \in Players$:[DONE(coach, execute-plan-rec-module(p, t))) $\wedge$ DONE(coach, get-role-map(plan(p), role(p))) $\wedge$ DONE(coach, get-formation-map(role(p), Formations)) $\wedge$ DONE(coach, get-team-tactics(formation, TeamTactics))] BEFORE [DONE(coach, decide-tactic(TeamTactics, tactic)) $\wedge$ DONE(coach, get-msg(tactic, msg))] BEFORE DONE(coach, send-message(coach, GC-op, msg))

Norms	PROHIBITED(coach, communicate(coach, Robots, direct,
	 {wireless, acoustic, visual}))
	 OBLIGED(coach, meet-msg-requirements(Msg, [Msg-Requirements]))
	 OBLIGED(GC-op, check-coach-message(Msg, [Msg-Requirements]))

Table C.4: Coach executes plan recognition, decides which tactic tip he can give the team, and sends the corresponding message (assuming he has a set of predefined messages - this depends on the implementation). Two alternative ways are given here as patterns; others still can be designed to better fit the coach implementation.

Interaction scene: communicate-coach-message	
Description	GC-op receives coach msg, decides whether or not to forward it to Players.
Roles	GC-op(1), coach(1), Players(≥1)
Results	r1: DONE(GC-op, inform(GC-op, Players, Msg)) ∧DONE(GC-op, display(GC-op, Msg)) r1': DONE(GC-op, drop(GC-op, Msg))
Patterns	r1: ∀ m = coach-message: DONE(GC-op, check-coach-message(GC-op, m, [Msg-Requirements])) ∧ [Msg-requirements] = *true* BEFORE [DONE(GC-op, inform(GC-op, Players, m)) ∧ DONE(GC-op, display(GC-op, m))] r1': ∀ m = coach-message: DONE(GC-op, check-coach,message(GC-op, m, [Msg-Requirements])) ∧ [Msg-requirements] = *false* BEFORE DONE(GC-op, drop(GC-op, m))

Norms	OBLIGED(GC-op, check-coach-message(Msg, [Msg-Requirements]))
	IF DONE(GC-op, check-coach-message(Msg, [Msg-requirements]))
	∧ [Msg-Requirements] = *true*
	THEN OBLIGED(GC-op, inform(GC-op, Players, Msg)
	∧ display(GC-op, Msg))
	IF DONE(GC-op, check-coach-message(Msg, [Msg-requirements]))
	∧ [Msg-Requirements] = *false*
	THEN OBLIGED(GC-op, drop(GC-op, Msg))

Table C.5: GC-op checks for each message instance whether it satisfies the requirements (appendix B) and sends it to the Players if it does.

Coordination between Attackers: example

Interaction scene: help(FP)	
Description	For example: two attackers, one has the ball
Roles	Attackers(2)
Results	DONE(Attackers, help(Attackers, FieldPlayers))
Patterns: semi-formal	{ $\exists a_1, a_2 \in \mathit{Attackers}$: DONE($a_1$, gain-ballPossession($a_1$, ball)), DONE($a_2$, walk($a_2$, supportPos($a_1$)) OR [DONE($a_2$, is-near($a_2$, Opponent)) AND DONE($a_2$, block-player($a_2$, Opponent))] }
Patterns: formal	$\{\exists a_1, a_2 \in \mathit{Attackers}$: $D_{a_1} \mathit{gain\text{-}ballPossession}(a_1, \mathit{ball})$, $D_{a_2} \mathit{walk}(a_2, \mathit{supportPos}(a_1))$ $\vee (D_{a_2} \mathit{is\text{-}near}(a_2, \mathit{Opp}) \wedge D_{a_2} \mathit{block\text{-}player}(a_2, \mathit{Opp}))$
Norms	All the 'Attacker'-norms and global norms apply (table B.3)

TABLE C.6: Conceptual idea of two attackers in a coordinated 'helping'-interaction.

Special scene 1: KickOff

Interaction scene: gameSituation(KickOff, (KickOffTeam))	
Description	(re)start of the game
Roles	Players(5), Opponents(5), GC-op(1), A-ref($\leq$2), H-ref(1)
Results	r1: $\forall s \in \mathit{Players} \cup \mathit{Opponents}$: DONE(GC-op, inform(GC-op, s, 'playing'))
Patterns	{ $\forall s \in \mathit{Players} \cup \mathit{Opponents}$: [DONE(GC-op, inform(GC-op, p, 'ready')) AND DONE(p, walk(p, legalPos(p, ownHalf, $\leq$45sec)))] BEFORE [DONE(GC-op, inform(GC-op, p, 'set')) AND DONE(p, ¬walk(p, $< \mathit{area}, \mathit{pos} >$)) AND DONE(a-ref, place(a-ref, ball, center))] BEFORE [DONE(h-ref, inform(h-ref, society, gameSituation('kickOff'))) AND DONE(GC-op, inform(GC-op, p, 'playing'))] }

Norms	OBLIGED(GC-op, inform(GC-op, Robots, RobotState))
	IF DONE(h-ref, inform(h-ref, society, RobotState("playing", $\geq$10sec))
	THEN OBLIGED(GC-op, inform(GC-op, h-ref, ball-in-play))
	OBLIGED(h-ref, inform(h-ref, society, gameSituation(GameSituation)))

TABLE C.7: KickOff can be initial, halfway, or when the h-ref decides an intermediate kick off is in order.

Special scene 2: Standard Removal Penalty

Interaction scene: Standard Removal Penalty (SRP)	
Description	A robot violated a norm, h-ref decides SRP
Roles	h-ref(1), a-ref(1), GC-op(1), Player(1)
Results	DONE(a-ref, apply-penalty(Player, SRP))
Patterns	{$\exists p \in Players$: DONE(h-ref, decide-penalty(h-ref, p, SRP)) AND DONE(h-ref, request(h-ref, a-ref, apply-penalty(p, SRP))) BEFORE DONE(a-ref, apply-penalty(p, SRP)) AND DONE(GC-op, inform(GC-op, p, 'penalized')) AND DONE(GC-op, adjust-clock(timed-penalty-started, 0sec)) BEFORE DONE(GC-op, adjust-clock(timed-penalty-10-sec-left, 35sec) AND inform(GC-op, society, clockState(timed-penalty-10-sec-left, 35sec))) BEFORE DONE(GC-op, adjust-clock(timed-penalty-ended, 45sec)) AND inform(GC-op, society, clockState(timed-penalty-ended, 45sec))) AND DONE(a-ref, place(a-ref, p, returnPos))}

Norms	OBLIGED(GC-op, adjust-clock(ClockState))
	OBLIGED(GC-op, inform(GC-op, society, clockState(ClockState)))
	OBLIGED(GC-op, inform(GC-op, Robots, RobotState))
	IF DONE(h-ref, request(h-ref, a-ref, apply-penalty(Robots, Penalty)))
	THEN OBLIGED(a-ref, apply-penalty(Robots, Penalty)))

Table C.8: `apply-penalty(p, SRP)`: remove p from the field for 45 seconds. `returnPos` is the robot's position when he was given the penalty, facing the opposite sideline, on its own half.

Appendix D

Norm Library

For the rules that aren't necessarily directly connected to roles and moreover, of which we want to be able to say that an agent violated it, a complementary norm library is given here. For the first couple of norms, both the semi-formal and formal versions are given by means of example.

To clarify, first a table with all predicates and properties that are used in multiple occasions in the framework is given, although they will most likely be interpretable from their occurrences throughout the framework tables. Parameters between the $<,>$ are obligatory and can be substituted with specific instances/agents of that sort ('id can be any agent in the society); parameters between an extra set of parentheses can be left unspecified. When a time parameter is included, it is most likely used to set the period during which the predicate holds (for example the duration of a robot being inactive).

Predicate	**Parameters**
in($< id >$, $< area, pos >$)	$< id >$: r $\in$ Robots, $< area, pos >$: domain areas or precise field positions (e.g. 'center', 'legalPos(Player)'..)
touch($< toucher >, (< bodypart >), < touchee >, (< t >)$)	$< toucher >, < touchee > \in$ Agents
lift/hold($< id >, < object >, (< t >)$)	object is most likely the ball. The difference between holding and lifting is that with lifting, it should be obvious that the object is *off* the ground.
move($< id >, < manner, bodypart >$)	moving in a certain way, moving one's certain bodypart(s) (e.g. arms)
communicate($< sender >, < receiver >$, $(direct), (\{wireless, acoustic, visual\})$)	between Robots or between H-TMs and Robots 'direct' because the coach - player communication is not allowed directly.
is-stuck($< robot, ball >, < in(1), between(2) >, (< t >)$)	e.g. robot is stuck in the goal net or ball is stuck between robot 1 and robot 2
is-lost($< id >, (< t >)$)	e.g. when a robot does nothing or keeps repeating itself
check($(< checker >), < checkee >$, <thing-to-check>)	e.g. whether or not a robot violates a rule
look-at($< id >, < object, robot >, (< t >)$)	mostly: robot looking at ball
chase/search($< id >, < object, robot >, (< t >)$)	mostly: robot chasing/searching ball
stationary/kicking/falling/getting-up($< id >$)	mostly: robots. In the process of that action.
walk/leave($< id >, < area, pos >, (< t >)$)	walking toward or leaving an area or position
damage($< id >, < object, area >$)	e.g. robot damages the field
meet-requirements($< object >, [requirements]$)	e.g. message, list of message-requirements
gameSituation($< GameSituation >, (< Team >)$)	e.g. KickOff, KickOff-Team
place($A\text{-}Ref, < object, robot >, < pos, area >$)	an Assistant-Referee may manually place robots or the ball if necessary also: place(a-ref, robot, ¬field) means that robot is removed from the field
is-near($< id >, < area, pos, object, id >$)	when a robot is near to the ball or a goal, for example

Game Situations

These are standard actions (the ones that aren't integrated in role or scene definitions). O=obliged, P=permitted, F=forbidden/prohibited. $D_r\varphi$=done (agent with role r 'saw to it that' φ). These rules obviously hold for both 'own' and 'opponent' team but are defined here from the point of view of the 'own' team.

GoalSaved
1. Semi-formal: IF lifted(goalkeeper, ball, >1sec) AND in(goalkeeper, ownPenaltyArea) THEN OBLIGED(h-ref, inform(h-ref, society, "goalSaved"))
2. Formal: $D_{goalkeeper}(lifts(goalkeeper, ball, > 1sec) \wedge in(goalkeeper, ownPenaltyArea)) \rightarrow O_{href} inform(href, society, "goalSaved")$

GoalScored; note that the inside of the object 'goal' is *ownGoalArea* whereas the occurence of scoring a goal is called *goal*.
1. IF (in(ball, oppGoalArea) AND NOT touched(Attacker, {hand,arm}, ball) AND NOT gameSituation(KickOff)) THEN OBLIGED(h-ref, (counts(goal, GameSituation) AND inform(h-ref, society, "goalScored"))
2. $\forall a \in Attackers:$
$D_a(in(ball, oppGoalArea) \wedge \neg touches(a, \{hand, arm\}, ball)) \wedge \neg gameSituation(KickOff)$
$\rightarrow O_{href}(counts(goal, GameSituation) \wedge inform(href, society, "goalScored"))$
3. GoalScored but doesn't count because ball is touched OR kick off scene:
$\forall a, a \in Attackers:$
$(D_a in(ball, oppGoalArea) \wedge (D_a touches(a, \{hand, arm\}, ball) \vee gameSituation(KickOff))$
$\rightarrow O_{href}(\neg counts(goal, GameSituation) \wedge inform(href, society, "\neg goalScored"))$

GameStuck; SPR=StandardRemovalPenalty. 'team(half-with-ball)' means that the team that is defending the half on which the ball is during the decision of stoppage will be the kick off-team. The condition of the game having changed can become false due to various reasons.
1. IF NOT game-changed("playing", >15sec) THEN
OBLIGED(h-ref, [decide-penalty(h-ref, Player(nearest-to-ball), SRP))]
OR [decide-stoppage(globalGameStuck, begin) AND gameSituation(kickOff, team(half-with-ball))])
2. $\forall p \in Players: \neg D_p game - changed("playing", > 15sec) \rightarrow$
$\exists p1 \in Players, is\text{-}near(p1, ball): O_{href}$
$decide\text{-}penalty(href, p1, SRP) \vee$
$decide\text{-}stoppage(globalGameStuck, begin) \wedge gameSituation(kickOff, team(half\text{-}with\text{-}ball))$

Violations

These obligations hold for all the agents (or specified subsets of agents) but are not role-specific.
Locomotion: $\forall p \in Players: D_p moves(p, \neg bipedal) \vee moves(p, \neg humanlike)$
$\rightarrow O_{href} decide\text{-}penalty(p, locomotion, HrefDecision)$

BallHolding: $\exists g, Goalkeeper(g) : D_g(holds(g, ball, > 10sec) \land \neg in(g, ownPenaltyArea))$
$\rightarrow O_{href} decide\text{-}penalty(g, ballHolding, SRP)$

Jamming: $\forall b \in Robots : D_r[communicates(b, Robots, wireless) \land (uses(b, \neg correctProtocol) \lor$
$exceeds(b, max\text{-}nr\text{-}msg))]$
$\lor [communicates(b, Robots, acoustic) \land is\text{-}similar(ownCommunication, oppCommunication)]$
$\lor [communicates(b, Robots, visual) \land LEDcolor(r, "orange")]$
$\rightarrow O_{href} decide\text{-}penalty(b, jamming, (disqualification \lor HrefDecision)))$

IllegalDefender: $\exists g, Goalkeeper(g); \forall x \in \{Defenders, Attackers\} :$
$D_g in(g, ownPenaltyArea) \land D_x in(x, ownPenaltyArea)$
$\rightarrow O_{href} decide\text{-}penalty(x, illegalDefender, SRP)$

Pushing: $\exists g, Goalkeeper(g); \forall p \in Players, \forall o \in Opponents :$
$D_p forceful\text{-}contact(p, y) \land (countPerGameHalf \leq 4 \lor 4 + 2 * count)$
$\land \neg(stationary(p) \lor kicking(p) \lor getting - up(p) \lor chases(p, ball) \lor is\text{-}stuck(ball, between(p, y)))$
$\land \neg(p = g \land looking\text{-}at(p, ball) \land in(p, ownPenaltyArea))$
$\rightarrow O_{href} decide\text{-}penalty(p, pushing, SRP)$

ArmPlay: $\forall p \in Players :$
$D_p touches(p, \{arm, hand\}, ball) \lor (p = goalkeeper \land \neg in(p, ownPenaltyArea)$
$\land D_p touches(p, \{arm, hand\}, ball)) \rightarrow O_{href} decide\text{-}penalty(p, armPlay, SRP)$

LeavingField: $\forall p \in Players :$
$D_p(leaves(p, carpet) \lor leaves(p, field) \lor is\text{-}lost(p) \lor is\text{-}stuck(p, in(goalnet), > 5sec))$
$\rightarrow O_{href} decide\text{-}penalty(p, leavingField, SRP)$

DamagingField: $\forall p \in Players :$
$D_p(damages(p, field) \rightarrow O_{href} decide\text{-}penalty(p, damagingField, SRP)$

CoachMotion: $\exists c, Coach(c) : \forall l \in Bodyparts : D_c(moves(c, l) \land l \neq (head \lor arm))$
$\lor D_c leaves(c, seatingPos) \rightarrow O_{href} decide\text{-}penalty(c, coachMotion, disqualification)$

Inactive: 1. $\forall p \in Players : D_p falling(p) \land \neg getting\text{-}up(p, < 5sec)$
$\lor D_p \neg walks(p, anyArea, > 10sec) \lor D_p \neg searches(p, ball, > 10sec)$
$\rightarrow O_{href} decide\text{-}penalty(p, inactive, SRP)$

2. $\forall p \in Players : D_p(falling(p) \land \neg getting\text{-}up(p, > 20sec))$
$\rightarrow O_{href} decide\text{-}penalty(p, inactive, definiteRemoval)$

Appendix E

Interaction Contract Protocol

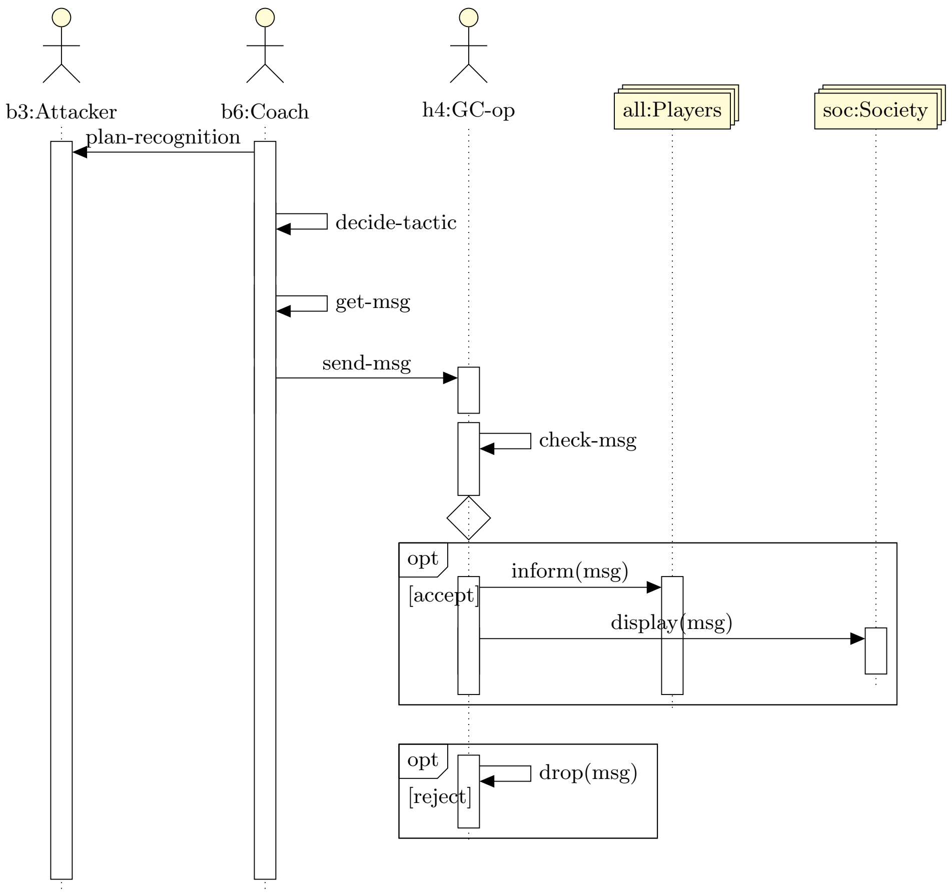


FIGURE E.1: Protocol for the interaction contracts 'message-tactics' and 'communicate-coach-message' as one structure. See section 3.4.

Bibliography

[1] C. Van Aart. *Organizational Principles for Multi-Agent Architectures*. Birkhäuser Verlag, Berlin; part of Springer, 2005.

[2] D.W. Albrecht, I. Zukerman, and A.E. Nicholson. Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction*, 8:5–47, 1998.

[3] S. Albrecht and S. Ramamoorthy. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems*, 2013.

[4] E. Argente, V. Julian, and V. Botti. Multi-agent system development based on organizations. *Electronic Notes in Theoretical Computer Science*, 150:55–71, 2006.

[5] D. Avrahami-Zilverbrand, G.A. Kaminka, and H. Zarosim. Fast and complete symbolic plan recognition: allowing for duration, interleaved execution, and lossy observations. In *Proceedings of the AAAI workshop on Modeling Others from Observations, MOO*, 2005.

[6] N. Bard and M. Bowling. Particle filtering for dynamic agent modelling in simplified poker. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22(1), 2007.

[7] S. Barrett and P. Stone. An analysis framework for ad hoc teamwork tasks. *To appear in Autonomous Agents and Multi-Agent Systems*, 2013.

[8] S. Barrett, N. Agmon, N. Hazon, S. Kraus, and P. Stone. Communicating with unknown teammates. *To appear in Aunomonous Agents and Multi-Agent Systems*, 2013.

[9] M. Beetz, B. Kirchlechner, and M. Lames. Computerized real-time analysis of football games. *IEEE Pervasive Computing*, 4(3), 2005.

[10] M. Beetz, J. Bandouch, and S. Gedikli. Camera-based observation of football games for analyzing multi-agent activities. In *Proceedings of AAMAS '06*, 2006.

[11] S. Behnke, J. Müller, and M. Schreiber. Playing soccer with robosapien. *In: A. Bredenfeld et al. (editors): RoboCup 2005, LNAI 4020, Springer*, pages 36–48, 2006.

[12] F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with a fipa-compliant agent framework. *Softw. Pract. Exper.*, 31:103–128, 2001.

[13] C. Boutilier. Sequential optimality and coordination in multiagent systems. *International Joint Conferences on AI*, 99:478–485, 1999.

[14] M. Bowling and P. McCracken. Coordination and adaptation in impromptu teams. In *Proceedings of AAAI'05*, pages 53–58, 2005.

[15] H.H. Bui. A general model for online probabilistic plan recognition. *International Joint Conferences on AI*, 3:1309–1315, 2003.

[16] H.H. Bui, S. Venkatesh, and G. West. Policy recognition in the abstract hidden markov model. *Journal of Artificial Intelligence Research*, 17:451–499, 2002.

[17] S. Carberry. Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11:31–48, 2001.

[18] D. Carmel and S. Markovitch. Model-based learning of interaction strategies in multi-agent systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 10(3):309–332, 1998.

[19] E. Charniak and R.P. Goldman. A bayesian model of plan recognition. *Artificial Intelligence*, 64:53–79, 1993.

[20] C. Claus and C. Boutilier. Reinforcement learning in cooperative multiagent systems. In *Proceedings of AAAI-98*, pages 746–752, 1998.

[21] RoboCup Technical Committee. Technical challenges for the robocup 2013 standard platform league competition, 2013.

[22] RoboCup Technical Committee. Robocup standard platform league (nao) rule book, 2013. URL http://www.informatik.uni-bremen.de/spl/pub/Website/Downloads/Rules2014.pdf.

[23] L. R. Coutinho, J.S. Sichman, and O. Boissier. Modeling organization in mas: A comparison of models. In *Proceedings of the 1st Workshop on Software Engineering for Agent-Oriented Systems (SEAS 2005)*, 2005.

[24] S. Cranefield, T. Finin, and S. Willmott. Introduction to the special issue on ontologies in agents systems. In *Proceedings of the 2nd International Workshop on Ontologies in Agent Systems, AAMAS 2002*, 2002.

[25] M. Dastani, M. Birna van Riemswijk, J. Hulstijn, F. Dignum, and J-J. Ch. Meyer. Enacting and deacting roles in agent programming. *in: J. Odell (editor): AOSE 2004, LNCS 3382; Springer*, pages 189–204, 2005.

[26] M.M. Dastani. 2apl: a practical agent programming language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.

[27] Y. Demiris. Prediction of intent in robotics and multi-agent systems. *Cognitive Processing*, 8(3):151–158, 2007.

[28] D. Dennett. *The Intentional Stance.* MIT Press, Cambridge, MA, 1987.

[29] M. Devaney and A. Ram. Needles in a haystack: Plan recognition in large spatial domains involving multiple agents. *AAAI-98*, pages 942–947, 1998.

[30] F. Dignum and R. Kuiper. Specifying deadlines with dense time using deontic and temporal logic. In *International journal of Electronic Commerce*, volume 3(2), pages 67–86, 1999.

[31] V. Dignum. *A Model for Organizational Interaction: based on Agents, founded in Logic.* PhD thesis, Utrecht University, 2004.

[32] V. Dignum, J. Vázquez-Salceda, and F. Dignum. Omni: Introducing social structure, norms and ontologies into agent organizations. *In: R.H. Bordini et al.(editors): PROMAS 2004, LNAI 3346*, pages 181–198, 2005.

[33] A. Drogoul and A. Collinot. Applying an agent-oriented methodology to the design of artificial organizations: A case study in robotic soccer. *Autonomous Agents and Multi-Agents Systems*, 1:113–129, 1998.

[34] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification.* John Wiley & Sons, Inc., 2nd edition edition, 2001.

[35] F. Dylla, A. Ferrein, G. Lakemeyer, J. Murray, O. Obst, T. Röfer, S. Schiffer, F. Stolzenburg, U. visser, and Th. Wagner. Approaching a formal soccer theory from behaviour specifications in robotic soccer. *Computers in Sport*, pages 161–185, 2008.

[36] M. Esteva, J.A. Rodríguez-Aguilar, C. Sierra, P. Garcia, and J.L. Arcos. On the formal specification of electronic institutions. *In: F. Dignum, C. Sierra (editors): Agent-Mediated Electronic Commerce (The European AgentLink Perspective), LNAI 1991, Springer*, pages 126–147, 2001.

[37] T. Röfer et al. B-human team report and code release 2011, 2011. URL http://www.b-human.de/downloads/bhuman11_coderelease.pdf.

[38] J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS'98)*, 1998.

[39] J. Ferber, O. Gutknecht, and F. Michel. From agents to organizations: An organizational view of multi-agent systems. *In: P. Giorgini, J.P. Müller, J. Odell (editors): AOSE 2003, LNCS 2935. Springer.*, pages 214–230, 2004.

[40] Foundation for Intelligent Physical Agents. Fipa communicative act library specification. doc. nr. sc00037j, 2002. URL http://www.fipa.org/specs/fipa00037/SC00037J.pdf.

[41] K. Genter, N. Agmon, and P. Stone. Role-based ad hoc teamwork. In *Proceedings of PAIR-11 (workshop at AAAI)*, 2011.

[42] K. Genter, N. Agmon, and P. Stone. Role-based ad hoc teamwork. In *Proceedings of PAIR-13 (workshop at AAAI)*, 2013.

[43] P.J. Gmytrasiewicz and P. Doshi. A framework for sequential planning in multi-agent settings. *Journal of Artificial Intelligence Research*, 24:49–79, 2005.

[44] T.R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Human-Computer Studies*, 43:907–928, 1995.

[45] M. Grüninger and M.S. Fox. Methodology for the design and evaluation of ontologies. In *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95*, 1995.

[46] K. Han and M. Veloso. Automated robot behavior recognition. *Robotics Research - International Symposium*, 9:249–256, 2000.

[47] S. Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42.1: 335–346, 1990.

[48] I. Horrocks. Description logic: A formal foundation for ontology languages and tools, 2010. URL `http://www.cs.ox.ac.uk/people/ian.horrocks/Seminars/seminars.html{#}other`.

[49] S.S. Intille and A.F. Bobick. A framework for recognizing multi-agent action from visual evidence. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1999.

[50] F. P. M. Dignum J.-J. Ch. Meyer, R. J. Wieringa. The role of deontic logic in the specification of information systems. *In: J. Chomicki, G. Saake (editors): Logics for Databases and Information Systems, Kluwer Academics Publishers*, pages 71–115, 1996.

[51] B. Johnston, F. Yang, R. Mendoza, X. Chen, and M. Williams. Ontology based object categorization for robots. *In: T. Yamaguchi (editor): PAKM 2008, LNAI 5345. Springer.*, pages 219–231, 2008.

[52] D. Jurafsky and J.H. Martin. *Speech and Language Processing*. Pearson Education, Inc., Upper Saddle River, New Jersey, pearson international edition edition, 2009.

[53] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Ingelligence*, 101:99–134, 1998.

[54] G.A. Kaminka and D. Avrahami. Symbolic behavior-recognition. In *Proceedings of the AAAI Workshop on Modeling Others from Observations (MOO)*, 2004.

[55] H.A. Kautz and J.F. Allen. Generalized plan recognition. In *Proceedings of AAAI-86*, volume 86, pages 32–37, 1986.

[56] A. Kleiner, M. Dietl, and B. Nebel. Towards a life-long learning soccer agent. *In: G.A. Kaminka, P.U. Lima, R. Rojas (editors): RoboCup 2002, LNAI 2752. Springer*, pages 126–134, 2003.

[57] T. Laue and T. Röfer. Particle filter-based state estimation in a competitive and uncertain environment. In *Proceedings of the 6th International Workshop on Embedded Systems*, 2007.

[58] H. Levesque, F. Pirri, and R. Reiter. Foundations for the situation calculus. *Computer and Information Science*, 3(18), 1998.

[59] N.R. Jennings M. Wooldridge. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10:2:115–152, 1995.

[60] P. MacAlpine, F. Barrerra, and P. Stone. Positioning to win: A dynamic role assignment and formation positioning system. In *Proceedings of the RoboCup International Symposium 2012*, 2012.

[61] J. McCarthy. Applications of circumscription to formalizing common sense knowledge. In *Proceedings from the Non-Monotonic Reasoning Workshop, AAAI*, 1985.

[62] M. Melissen. *Game-theory and Logic for Non-repudiation Protocols and Attack Analysis.* PhD thesis, University of Luxembourg, 2013.

[63] R. Mendoza and M. Williams. Ontology-based object categorisation for robots. *Australasian Ontology Workshop (AOW)*, 2005.

[64] M. Mohr, P. Krustrup, and J. Bangsbo. Match performance of high-standard soccer players with special reference to development of fatigue. *Journal of Sports Sciences*, 21:7:519–528, 2011.

[65] K.P. Murphy. *Machine Learning. A Probabilistic Perspective.* MIT Press, Cambridge, Massachusetts, 2012.

[66] A.E. Nicholson and J.M. Brady. Dynamic belief networks for discrete monitoring. *Systems, Man and Cybernetics, IEEE Transactions*, 24(11):1593–1610, 1994.

[67] R. Noë. Cooperation experiments: Coordination through communication versus acting apart together. *Animal Behaviour*, 71:1–18, 2006.

[68] N.F. Noy and D.L. McGuinness. Ontology development 101: A guide to creating your first ontology, 2001. URL http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html.

[69] H.S. Nwana, L. Lee, and N.R. Jennings. Co-oordination in software agent systems. *British Telecom Technological Journal*, 14(4), 1996.

[70] J.J. Odell, H. Van Dyke Parunak, and M. Fleischer. The role of roles in designing effective agent organizations. *in A. Garcia et al. (editors): SELMAS 2002, Lecture notes in computer science 2603*, pages 27–38, 2003.

[71] D. Okouya and V. Dignum. Operetta: A prototype tool for the design, analysis and development of multi-agent organizations (demo paper). In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*, 2008.

[72] Stephan Opfer. Towards Description Logic Reasoning Support for ALICA. Master's thesis, Universität Kassel, 2012.

[73] Y. Oshrat, R. Lin, and S. Kraus. Facing the challenge of human-agent negotiations via effective general opponent modeling. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, 2009.

[74] S. Park and V. Sugamaran. Designing multi-agent systems: a framework and application. *Expert Systems with Applications*, 28:259–271, 2005.

[75] H. Prakken. Commonsense reasoning, 2012/2013. Utrecht University Msc Cognitive Artificial Intelligence course reader 2012/2013.

[76] S. Ramamoorthy, M.M.H. Mahmud, B. Rosman, and P. Kohli. Latent-variable mdp models for adapting the interaction environment of diverse users. *Technical report: University of Edinburgh*, 2013.

[77] A.S. Rao and M.P. Georgeff. Modeling rational agents within a bdi-architecture. *KR*, 91: 473–484, 1991.

[78] P. Riley and M. Veloso. Coaching a simulated soccer team by opponent model recognition. *AGENTS'01*, 2001.

[79] P. Riley and M. Veloso. Recognizing probabilistic opponent movement models. *RoboCup 2001: Robot Soccer World Cup V. Springer*, pages 453–458, 2002.

[80] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, Inc., Upper Saddle River, New Jersey, 2nd edition edition, 1995/2003.

[81] A. Salter and K. Liu. Using semantind and norm analysis to model organizations. In *In: J. Bráz, M. Piattini, J. Filipe (editors): Proceedings of ICEIS 2002.*

[82] S. Saria and S. Mahadevan. Probabilistic plan recognition in multiagent systems. In *Proceedings of ICAPS-04, AAAI*, pages 287–296, 2004.

[83] A. Savitzky and M.J.E. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

[84] G.Y.R. Schropp, E. Lefever, and V. Hoste. A combined pattern-based and distributional approach for automatic hypernym detection in dutch. In *Proceedings of the 9th International Conference on Recent Advances in Natural Language Processing (RANLP2013)*, pages 593–600, 2013.

[85] J.R. Searle. *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press, 1969.

[86] H.A. Simon. Theories of bounded rationality. *in C.B. McGuire and Roy Radner (editors): Decision and Organization, chapter 8*, 1972.

[87] M.P. Sindlar, M.M. Dastani, F. Dignum, and J-J.Ch. Meyer. Mental state abduction of bdi-based agents. *In: M. Baldoni et al. (editors): DALT 2008, LNAI 5397. Springer.*, pages 161–178, 2008.

[88] M.P. Sindlar, M.M. Dastani, and J-J.Ch. Meyer. Programming mental state abduction. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, volume 1, pages 301–308, 2011.

[89] F. Southey, M. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, and C. Rayner. Bayes' bluff: Opponent modelling in poker. *arXiv Preprint:1207.1411*, 2012.

[90] C. Stanton and M. Williams. Grounding robot sensory and symbolic information using the semantic web. *In: D. Polani et al. (editors): RoboCup 2003, LNAI 3020*, pages 757–764, 2003.

[91] P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 110(2):241–273, 1999.

[92] P. Stone and M. Veloso. Multiagent systems: a survey from a machine learning perspective. *Autonomous Robots*, 8:345–383, 2000.

[93] P. Stone, P. Riley, and M. Veloso. Defining and using ideal teammate and opponent agent models: A case study in robotic soccer. *MultiAgent Systems, 2000. Proceedings of the 4th International Conference on IEEE*, pages 441–442, 2000.

[94] P. Stone, G.A. Kaminka, S. Kraus, and J.S. Rosenschein. Ad hoc autonomous agent team: Collaboration without pre-coordination. In *Proceedings of AAAI'10*, 2010.

[95] P. Stone, G. A. Kaminka, S. Kraus, J. S. Rosenschein, and N. Agmon. Teaching and leading an ad hoc teammate: Collaboration without pre-coordination. *Preprint submitted to Artificial Intelligence*, 2013.

[96] R.S. Sutton and A.G. Barto. *Reinforcement Learning. An Introduction.* MIT Press, Cambridge, Massachusetts, 1998.

[97] M. Tambe. Towards flexible teamwork. *Journal of Artifical Intelligence Research 7*, pages 83–124, 1997.

[98] A. Valtazanos and S. Ramamoorthy. Intent inferencee and strategic escape in multi-robot games with physical limitations and uncertainty. *Intelligent Robots and Systems (IROS), IEEE/RSJ*, 2011.

[99] A. Valtazanos and S. Ramamoorthy. Bayesian interaction shaping: Learning to influence strategic interactions in mixed robotic domains. In *In: Ito, Jonker, Gini, Shehory (editors): Proceedings of the 12th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2013)*, 2013.

[100] P. Vogt. The physical symbol grounding problem. *Cognitive Systems Research*, 3:429–457, 2002.

[101] H. Weigand, F. Van der Poll, and A. De Moor. Coordination through communication. In *Proceedings of the 8th International Working Conference on the Language-Action Perspective on Communication Modelling (LAP2003)*, 2003.

[102] T. Weigel, W. Auerbach, M. Dietl, B. Dümler, J. Gutmann, K. Marko, K. Müller, B. Nebel, B. Szerbakowski, and M. Thiel. Cs freiburg: Doing the right thing in a group. *In: P. Stone, T. Balch, G. Kraetzschmar (editors): RoboCup 2000, LNAI 2019*, pages 52–63, 2001.

[103] T. Weigel, K. Rechert, and B. Nebel. Behavior recognition and opponent modeling for adaptive table soccer playing. *In: U. Furbach (editor): KI 2005, LNAI 3698. Springer.*, pages 335–350, 2005.

[104] M. Wooldridge. Time, knowledge, and choice. *In: Wooldridge, M., Mueller, J. P. ,Tambe, M. (Eds.): Intelligent Agents II, Springer-Verlag*, 1996.

[105] M. Wooldridge, N. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[106] K. Yoshimura, N. Barnes, R. Rönnquist, and L. Sonenberg. Towards real-time strategic teamwork: A robocup case study. *In: G.A. Kaminka, P.U. Lima, R. Rojas (editors): RoboCup 2002, LNAI 2752, Springer*, pages 342–350, 2003.

[107] G. Zhu, C. Xu, Q. Huang, and W. Gao. Automatic multi-player detection and tracking in broadcast sports video using support vector machine and particle filter. In *Proceedings of the International Conference on Multimedia and Expo*, pages 1629–1632, 2006.