Autonomous Agents and Multi-Agent Systems, 1, 113–129 (1998)
© 1998 Kluwer Academic Publishers. Manufactured in The Netherlands.

Applying an Agent-Oriented Methodology to the Design of Artificial Organizations: A Case Study in Robotic Soccer

A. DROGOUL AND A. COLLINOT {anne.collinot, alexis.drogoul}@lip6.fr
LIP6-OASIS-MIRIAD, Université Paris VI, 4 place Jussieu, Case 169, F-75252 Paris Cedex 05, France

Abstract. The multi-agent paradigm is widely used to provide solutions to a number of organizational problems related to the collective achievement of one or more tasks. All these problems share a common difficulty of design: how to proceed from the global specification of a collective task to the specification of the local behaviors to be provided to the agents? We have defined the Cassiopeia method whose specificity is to articulate the design of a multi-agent system around the notion of organization. This paper reports the use of this method for designing and implementing the organization of a soccer-playing robotic team. We show why we chose this application and how we designed it, and we discuss its interest and inherent difficulties in order to clearly express the needs for a design methodology dedicated to DAI.

Keywords: collective robotics, multi-agent organization, agent-oriented design

1. Introduction

The multi-agent paradigm is widely used to provide solutions to a variety of organizational problems related to the collective achievement of one or more tasks: computer supported cooperative work, flexible workshop or network management, distributed process control, or coordination of patrols of drones [3, 10, 24]. All these problems share a common difficulty of design: how to proceed from a global specification of the collective task to the specification of the individual behaviors, which is to be provided to the agents that achieve the task. A problem of organization has to be solved, most of the time in a dynamic fashion, so as to obtain the collective achievement of the considered task. In previous works, we have defined the Cassiopeia method [8] whose purpose is to provide a methodological framework to design multi-agent systems (MAS). The underlying principle of Cassiopeia is to articulate the design activity around the notion of organization. In this paper, we report the use of the Cassiopeia method in the context of a research project (MICROB) on collective phenomena of organization in robot societies.

The paper is organized as follows: In Section 2, we present the MICROB project and the application we are concerned with, that is, the design of a soccer robot team. We discuss the interests and difficulties that characterize this application, which directly sets the requirements for a methodology dedicated to DAI. In Section 3, we give an overview of the methodological approach of Cassiopeia. In Section 4, we report the use of the Cassiopeia method for the design and implementation of the soccer robot team. Finally, in Section 5, we discuss our current results.

2. The MICROB project: Soccer-playing robots

The aim of the MICROB [11] project is to investigate collective phenomena of organization in societies of robots. The project relies on the use of a simple test-bed, which enables us to conduct experiments that focus on the organizational features of agents. So far, the first instantiation of this project has taken place within the RoboCup [2] and MiroSot competitions of soccer-playing robots, both on simulated and real playing fields. The simulation experiments make use of the RoboCup official simulator, SoccerServer [14], while the real experiments rely on two different types of robots.

2.1. The MICROB experimental testbed

The MICROB medium- and small-size robots (see Figure 1) are made of remote-controlled cars, which thus have no sensor and no on-board decision module. Each car receives its commands through a remote-control pilot, which transmits the instructions that have been worked out either by a software agent or a human agent. A camera is filming the scene while an image processing system keeps analyzing the position and the speed of all the robots and objects that are on the

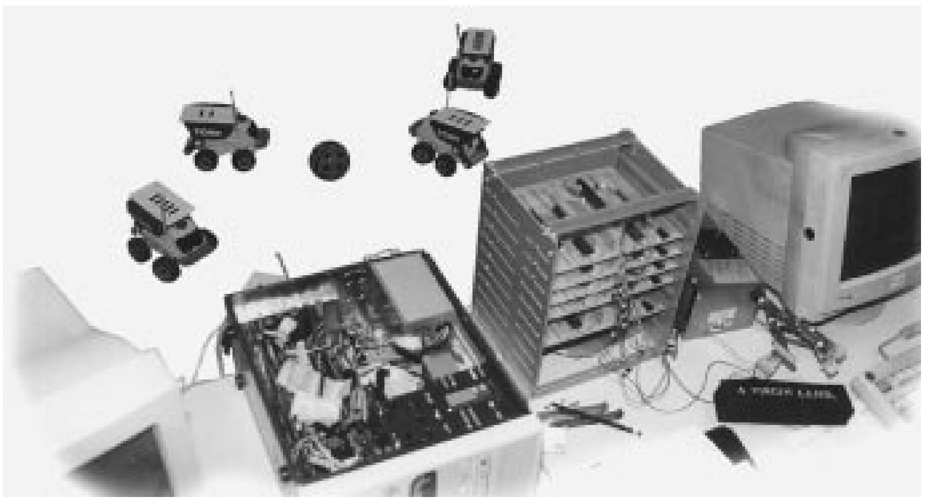
Figure 1. The MICROB project (medium-size robots).

playing field. The data are then placed into a simulated environment within which the software agents are operating to make appropriate decisions that will be sent to the remote-control pilots. Given this test-bed, the main issue is to design the software agents so that the robots they command can exhibit capabilities to organize themselves and collectively achieve a task.

2.2. *The application to a soccer-playing robotic team*

The first application of the MICROB project has been the construction of a team of soccer-playing robots. The playing field in which the robots are moving is rectangular and comprises a goal at each end. Each team is composed of four robots. In our first experiments, the cars of the first team were remote-controlled by software agents, while the cars of the opposing team were remote-controlled by human agents. In the RoboCup and MiroSot competitions, both teams are either autonomous or controlled by software agents.

2.3. *Designing a soccer-playing robotic team: Interests and difficulties*

Obviously, in such a typical team game, the aim of the designer is that the team of software agents scores goals while defending its side. Thus, the problem is to design the software agents so that they can play with what we might call team spirit, that is, a set of explicit or implicit social rules and constraints that enables them to play collectively and prevents them from disturbing their teammates while they are playing. Consequently, besides their individual soccer-playing skills, the agents should have capabilities to organize themselves in order to win. From the standpoint of multi-agent systems, the main difficulty is then to express *locally* (i.e., at the agent level) the behaviors which allow the *collective* achievement by the team of the task to be obtained. Two other difficulties make this game a very interesting, though very challenging, standard problem for multi-agent systems:

- The dynamics of the game makes it impossible both to define the organization of the robots in advance and to control the game in a centralized way.
- The operations of the opposing team are, by definition, unpredictable and, consequently, require a high level of real-time adaptability.

From the standpoint of software engineering, this problem is interesting as well since it covers most of the issues of analysis, design, implementation, experimentation, and validation of artificial organizations in a strictly controlled while moderately abstracted world [14]. This is the reason why this application has been chosen as the first test-bed for Cassiopeia, enabling us to undertake the design of a large MAS project and to evaluate its contribution in a more comprehensive manner than in the case of a toy application.

3. The methodological approach of Cassiopeia[1]

3.1. Requirements for a methodology dedicated to DAI

Most software engineering projects, especially multidisciplinary projects like MICROB, require the use of a methodology, whose main role is to identify the necessary steps that permit us to proceed from the definition of the project requirements to their fulfilment (i.e., the project life cycle). More concretely, a methodology supplies the tools for transforming an always intuitive and subjective vision of a system to be built (the client's requirements, for instance) into a formalized and objective (i.e., which can be shared and reused) definition of the same system once it has been implemented. It thus provides a project with "something" that will stand and remain somewhere between the original blueprint and the final code. And we claim that this "something," despite many valuable studies, does not yet exist in the MAS community, thus preventing software engineers from using the MAS technology in routine operation. First of all, let us examine what this "something" usually contains:

(a) A structured set of guidelines, which includes the aforementioned steps, advice for each of these steps, and how to proceed from one step to another.
(b) A unified way to document the design process. This is used for sharing the experience gained during this process among designers and/or across time, whenever, for instance, the design has to be undertaken by others.
(c) The use of a homogeneous terminology, which has a meaning at each step of the cycle and supports the transitions from step to step (it usually includes as well a graphical terminology based on diagrams and flowcharts).
(d) The use of *operational* conceptual abstractions; that is, conceptual structures abstract enough to allow a sufficient choice of techniques when it comes to implementing the system, but are operational enough to prevent the designer from using unrelated or outdated techniques.
(e) A comprehensive and incremental history of the project, which gives the possibility to backtrack from any step to previous ones without losing what has been done.

In the case of DAI, the basic project requirements consist of having a number of agents achieve a collective task. To fulfil these requirements, one must design the agents with specific attention to their capabilities for organizing themselves. The existing methodologies, especially the object-oriented methodologies [13] that can be considered because of some similarities such as distribution or locality,[2] provide an interesting basis of analysis [1, 6], since they enable the distribution of the initial requirements along the structural and behavioral axes. However, they do not offer any methodological framework for taking the various organizational issues into account, because the organization is not considered as an object of analysis by itself [4]. A number of very interesting studies on building agent-oriented methodologies as extensions of object-oriented ones have nevertheless been undertaken (see for example [18]), but what they can really provide so far is still unclear: Agents and objects differ in a number of important respects, the most noticeable being the capability of agents to dynamically and autonomously change their organization, something that objects are not supposed to be allowed to do. We believe that although object-oriented languages are the target of choice for implementing MAS, it would be an error to consider agents as only "superobjects" (see, for instance, Wooldridge's arguments about the importance of the *intentional stance* for building MAS [26]).

On the other hand, the few DAI works (e.g., [17, 25]) that deal with methodological aspects are not very satisfying, since they either indirectly address the organizational issues through the use of specific negotiation or coordination techniques, or impose certain agents' architectures, which in fact are only particular methods of implementation. We tend to think that the peculiarities of multi-agent systems (with respect to both objects and single-agent AI systems) require any methodological framework to integrate:

1. The descriptive and operational aspects of the organization as early as the analysis step, both for implementation and documentation reasons.
2. The possibility to combine a bottom-up (designing the agents before the organization) as well as a top-down (designing the organization prior to the agents) approach.

3.2. *Overview of Cassiopeia*

The Cassiopeia method is primarily a way to address a type of problem-solving where collective behaviors are put into operation through a set of agents. It is not targeted at a specific type of application nor does it require a given architecture of agents. However, it is assumed that, although the agents can have different aims, the goal of the designer is to make them behave cooperatively. The main idea is that a MAS should be designed in terms of agents provided with three levels of behavior:

1. Elementary behaviors, in other words, the different functions or actions that the agents are individually able to perform, regardless of the policy they will choose to perform them with.
2. Relational behaviors, that is, how they interact with one another (by enabling/disabling some elementary behaviors) with respect to both the elementary behavior they are engaged in and its influence on the other agents, and the influence exerted by other agents.
3. Organizational behaviors, or how the agents can manage their interactions to become or stay organized (by enabling/disabling some relational behaviors).

These three levels of behaviors are quite common in the MAS literature and, to put things more simply, we can describe them as answers to the following three questions (which are, at varied degrees, the core questions of DAI):

1. How to behave or what to do
2. How to interact or what to do *in the presence of* other agents
3. How to cooperate or what to do *with* these agents

The underlying principle of Cassiopeia is that the problem of organizing individuals to undertake collective problem-solving must be addressed as such by the designer and/or the agents. Such organizational issues arise when *functional dependencies* are inherent to the collective achievement of the considered task: The activation of an individual problem-solving behavior affects and is affected by problem-solving behaviors activated by other agents. The whole set of these dependencies determines the *coupling* of the organizational problem that underlies the task achievement.[3] We distinguish two types of coupling: (1) When the individual behaviors are not competing with one another, the coupling is said to be static, which means that the organization of the agents remains unchanged at run time and can be defined in advance at design time (e.g., the multi-expert systems in which there is exactly one expert per domain of expertise); (2) when some sort of competition is to be managed (individual behaviors exist that are equivalent for a given situation, or a same behavior can be operated by different agents), the coupling is said to be dynamic, which means that the organization cannot be defined in advance since it depends on the run-time context. Therefore, the designer can only consider the organizational structures that can be built between specific types of agents.[4] These structures will in turn be instanced (or not) during the run-time context. This is the case in the MICROB project, where the soccer-playing robots should be able to organize themselves dynamically, because of the unpredictability of the game (see Section 2.3).

Although Cassiopeia does not yet offer all the aforementioned components that one could expect to find in a complete methodology, it provides a methodological framework to better understand and plan the design of a computational organization. It then proceeds from the definition of the collective task to the design of the MAS along three steps that reconcile both local and global views of the MAS (Figure 2):

1. The definition of the required elementary behaviors in order to define the types of agent.
2. The structural description of the organizations that can be built upon these different types of agents.
3. The description of the dynamics of these organizations, that is, the organizational behaviors the agents have to perform to make them appear, evolve, or disappear.

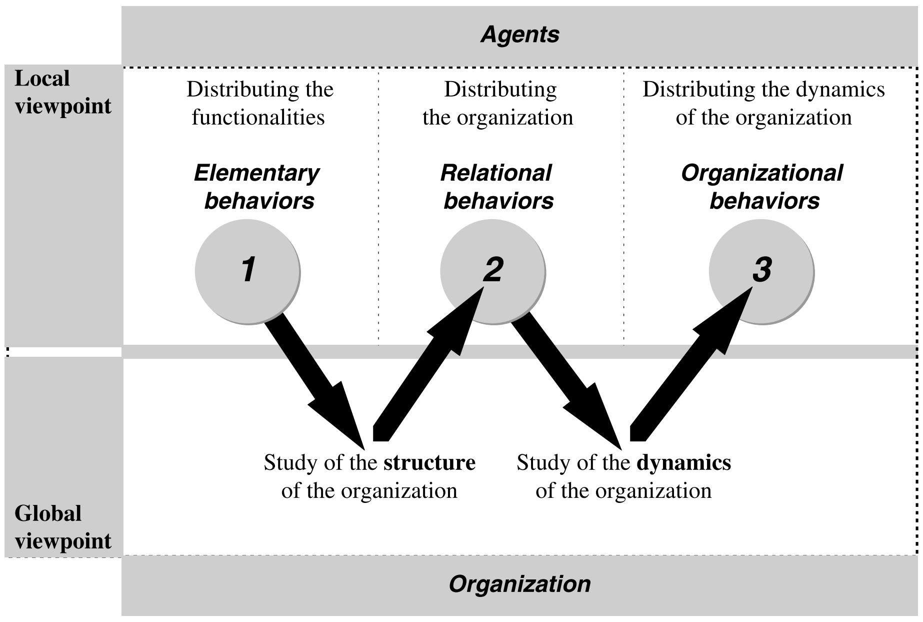
Agents
Local viewpoint
Distributing the functionalities
Distributing the organization
Distributing the dynamics of the organization
Elementary behaviors
Relational behaviors
Organizational behaviors
1
2
3
Study of the structure of the organization
Study of the dynamics of the organization
Global viewpoint
Organization

Figure 2. The Cassiopeia steps and how they allow one to alternatively proceed from the local viewpoint (that of the agents) to the global viewpoint (that of their organization).

3.3. Step 1. Elementary behaviors

The first step of Cassiopeia consists of listing the elementary behaviors that are required for the achievement of the collective task. The identification of these behaviors does not come under Cassiopeia: it can result from a functional (e.g., [27]) or object-oriented (e.g., [1, 6, 20]) analysis step. The designer is free to define different types of agents based on these behaviors: he can choose to design agents that are either homogeneous (all the agents are provided with the same set of elementary behaviors) or heterogeneous (some agents are supplied with only a subset of these behaviors).

3.4. Step 2. Relational behaviors

The second step consists of analyzing the organizational structure based on the dependencies between the elementary behaviors. These dependencies define the *coupling graph* underlying the considered collective task. The dependencies between the types of agents[5] are elaborated based on this coupling graph by projecting it onto these different types and removing the inconsistent or unnecessary dependencies.[6] This refined projected graph contains the sole dependencies that are supposed relevant to the achievement of the task. Such dependencies between the different types of agents are called *influences* (somewhat close to the notion of social dependencies introduced in [5]), and the refined projected graph is called the *influence graph*. The paths and the elementary circuits of this influence graph define a number of potential groups of agent types and, therefore, provide a global representation of the organizational structure.

Based on this analysis, the designer specifies the *relational behaviors* that enable the agents to identify and handle the influences. In this task, the designer determines the *influence signs*, which are produced by each type of *influencing agent*, and specifies how an *influenced agent* takes into account an influence sign; that is, which elementary behaviors it activates and in which way. These signs can be anything, from a complex message to an action on the environment—it depends on the techniques that will be used when implementing the MAS. A reactive agent-based architecture would typically use some sort of indirect communication (i.e., an agent is influenced whenever it perceives a change in its environment), while a cognitive approach could consider signs as being KQML-like messages.

3.5. Step 3. Organizational behaviors

The third step addresses the dynamics of the organization. It consists of specifying the behaviors that will enable the agents to manage the formation, durability, and dissolution of groups. The relational behaviors may allow the formation of several groups that are redundant with respect to their end. When such a redundancy is useless or costly, it is necessary to determine the criteria that affect the choices to form one group rather than another. As described in the previous step, when an agent needs to form a group, it produces the relevant influence signs. The occurrence of redundant groups thus indicates a redundancy of means to meet the need of this particular agent, called the *trigger* agent. Such a trigger agent belongs to all potential groups meeting its need and should evaluate them to decide which one is the most appropriate in the current context. The designer should thus (1) identify the trigger agents (according to the potential groups that have been identified in the influence graph) and (2) determine for each of them, the selection methods allowing control of the formation of these groups. There exist a number of techniques for making such choices [9], based on task announcement such as all the techniques deriving from the Contract Net [21], the notion of consensus and negotiation among agents belonging to concurrent groups [19, 23], priorities to allow ordering the potential groups [12], or the use of a supervisor or a hierarchy [16]. These techniques define a first type of organizational behavior: the *group formation behaviors*.

The designer next specifies the *commitment signs*, which are produced by the trigger agents to indicate to other agents that a group is formed to meet their need. These signs allow the agents that are not members of the formed groups to control (e.g., to inhibit) their relational behaviors. The designer thus defines for each type of agent a second type of organizational behavior, which takes the commitment signs into account to organize the relational behaviors (i.e., inhibit or activate a given set of interactions with others): the *joining behaviors*.

Finally, the choices resulting from the group formation behaviors may need to be revised. A group ceases to exist when it has carried out its commitments or a group can be replaced by a more efficient one. The designer thus defines for each type of agent a third type of organizational behavior for dissolving a group: *the group dissolution behaviors*.

4. Using Cassiopeia to design a soccer robot team

In this section, we report how we followed the steps of the methodology in order to design the multi-agent system that pilots the MICROB soccer robot team. The point of this section is more to follow the method step by step rather than to give the details of implementation of the application. Its interest does not then reside in the algorithmic solutions (although the application is performing quite well), but in the way these solutions are introduced by (and integrated in) the design process.

4.1. Step 1. The elementary behaviors of the soccer robots

The elementary behaviors we are describing in this section result from a functional analysis stage of the soccer game. These behaviors may appear fairly abstracted with respect to the lower-level actions one might expect to find in this game, but, as we wanted to be able to express dependencies between them, we needed them to be at least context-dependent and/or goal-oriented behaviors. Of course, the difficulty is always to find the adequate level of abstraction, but, as far as we know, this difficulty is shared by all the existing techniques of analysis (see, for instance, [27]).

Four elementary behaviors have then been defined to enable our robots to play soccer individually *and* collectively:

1. *Shoot* the ball in a given direction (usually that of the opponents' goal).
2. *Place* oneself on the playground, waiting for a pass.
3. *Block* an opponent's way.
4. *Defend* one's goal against the opponents' attacks.

The details of the algorithms that implement these behaviors need not be given in this paper. Again, note that the behaviors are *elementary* with respect to the MAS design level, but not with that of the robots'. Indeed, they are at a more abstract level than the standard behaviors of a robot since they represent a combination of them (e.g., the shooting behavior). This is true in the context of the simulated robots as well, because the client/server architecture of the SoccerServer simulator only accepts low-level orders (such as kick, turn, dash, etc.).

For the time being, the MICROB software agents are all provided with the four behaviors. However, they activate only one of them at a given time, which is referred to as their *active elementary behavior*. The agents that shoot, block, defend, or place themselves are then, respectively, called a shooter, blocker, defender, or placer. This led us to design four different types of agent. Thus, at a given time, an agent has a particular *role* in its team, which may be dynamically determined by the role of the others. This kind of collective control will be held by the relational and organizational behaviors.

4.2. Step 2. The relational behaviors of the soccer robots

The four elementary behaviors can be viewed in many cases as depending on one another. For instance, the shot of a robot depends on the position of the other robots on the playground: one should not make a pass to a misplaced robot. It is true, however, that there are a number of possible types of dependencies between these behaviors:

1. *Functional dependencies*. Performing a behavior depends on the previous or simultaneous activation of another behavior (by the same or another agent).
2. *Resource-based dependencies*. A behavior can be inhibited because of a conflict of resources with another behavior, or its activation depends on a resource created by another behavior.
3. *Organizational dependencies*. Contrary to the two categories above, these are linked to the type of organization the designer wants to build. He can, for example, decide to make behaviors depend on one another in order to implement tactics or strategies that cannot simply emerge from the "objective" dependencies described previously.

Cassiopeia does not yet offer the possibility to distinguish between these different types of dependencies,[7] so the ones inherent to the game as well as those designed to fulfil the goal of the designer appear on the same coupling graph (Figure 3).

Seven different dependencies have then been identified (an arrow from A to B means that B potentially depends on A):

1. Blocking an opponent can help another robot (including oneself) to better place itself.
2. Defending can help oneself or another robot to better place itself.
3. It may be necessary to place oneself if another robot wants to shoot.
4. Defending may allow catching the ball of the opponent.
5. Blocking can help oneself or another robot to shoot the ball.

6. Shooting can help oneself or another robot to shoot (this is the pass).
7. Defending depends on the other robots' defense strategy.

In order to define the influences between the soccer behaviors, the coupling graph of Figure 3 is projected on the four types of soccer agents. Although we considered only the dependencies whose target is an active behavior, the resulting graph is intricate and, following Cassiopeia, we explicitly made assumptions about the relevance of some dependencies. For example, we decided to ignore the dependencies whose origin is a passive blocking, shoot, or defense (except when their target is an active defense behavior). This choice is clearly discretionary, but it is explicit and we know where it has been done in the design cycle.[8]

The resulting influence graph (Figure 4) is used to determine which influence signs should be produced. Once again, some choices are to be made, in particular regarding the agent communication protocol, which is how the influence signs are communicated. In the current implementation, we chose a simple object-oriented synchronous message passing: an influence sign is a message sent to an agent; its interpretation is the execution of the corresponding method, which returns a value to the sender. Here are the influences considered, along with their associated

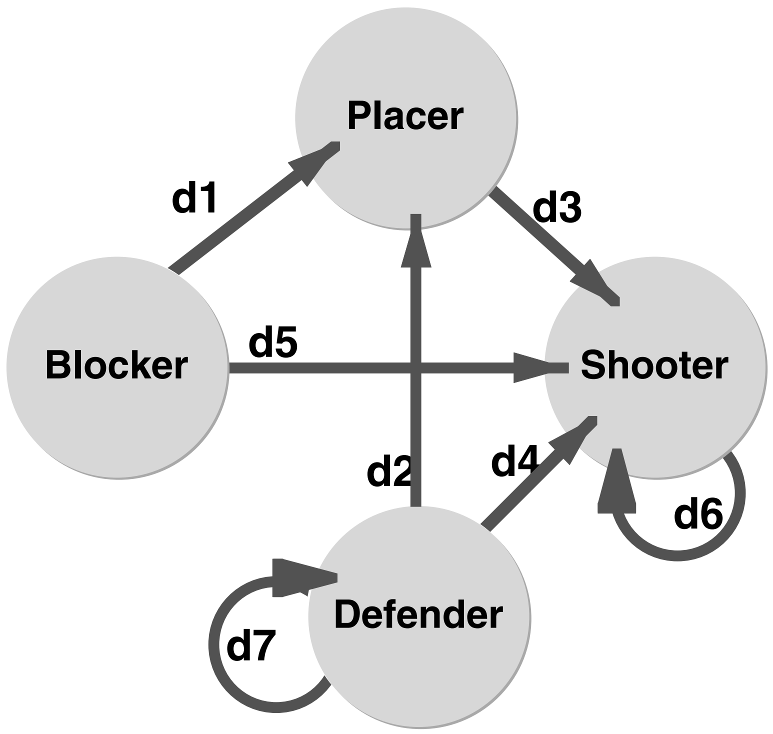

Figure 3. The coupling graph of the robotic soccer game.

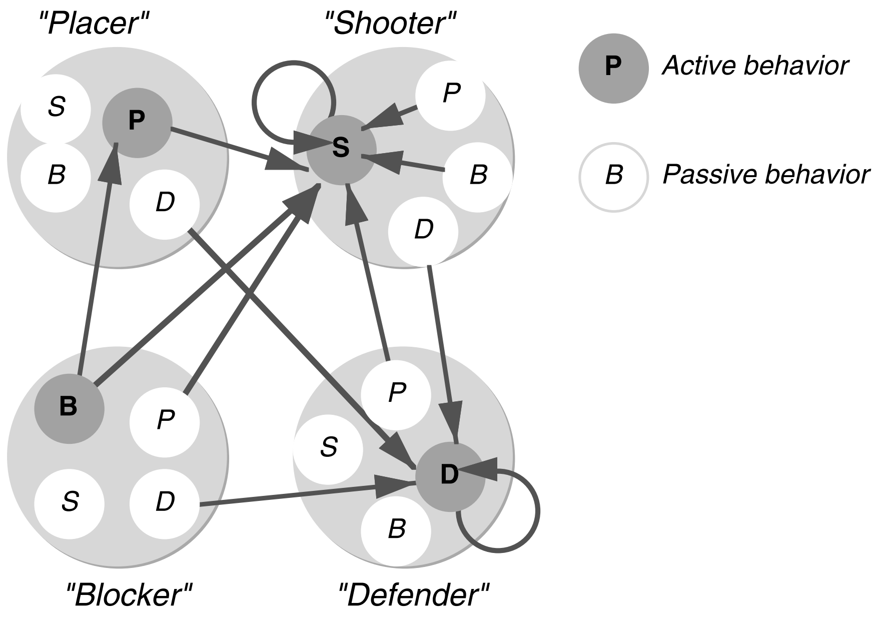

Figure 4. The influence graph for the soccer robotic team.

influence signs:

- A blocker can provide help to a shooter. A shooter is able to send a *help* message, which can be interpreted by any type of agent to return a value representing its capacity to block, based on its perception of the game.
- A shooter may make a pass to itself or another shooter. The shooter sends a *shoot* sign and the receiver of the message returns its capacity to shoot. The shooter then adjusts the direction of its shot.
- A shooter must take the other robots' positions into account before shooting. The *place* sign is sent by the shooter, and the receiver of the message returns its capacity to shoot and place itself.
- Any agent can influence a defender because of its position on the playground. The *position* sign is produced by the defender, and the receiver of the message returns its position, which is then used by the defender to adjust the direction of its move.

In this current implementation, the relational behaviors are simply handling the transmission of the messages and the activation of the associated methods.

4.3. Step 3. The organizational behaviors of the soccer-playing robots

The behaviors allowing the agents to dynamically organize themselves, according to their mutual influences, are based on the contract net mechanism. In this implementation, the *trigger agent* is then the manager that proposes the contracts.

The analysis of the influence graph (Figure 4) provides all the information required for deciding which agents should be considered as potential candidates for becoming trigger agents. This is done by following a very simple rule: Each agent whose active behavior is influenced by more than one other behavior may become a trigger agent. In other words, and quite obviously, an agent whose active behavior *depends* on others' behaviors is a good candidate for forming a group around itself.

This led us to decide, not surprisingly, that the two types of agents likely to become managers would be the defender and the shooter. However, we chose to restrict this possibility solely to the shooter in the real robots games, since the teams are made up of only four robots, which makes it difficult to split them into two groups. In the simulation games, where up to 11 agents can play simultaneously in the same team, things are much easier—and realistic: The goalie is a permanent defender agent (it is not provided with any other behavior), while the shooting behavior is triggered dynamically (with respect to the relative positions of the ball and the other players).

- The *group formation behaviors* are thus defined for the shooter and defender agents:
  1. When an agent becomes a shooter, it first evaluates the capabilities of its teammates to take up their position and block opponents, and then establishes a placing contract and a shooting contract with those which return the best values. In the implementation of the real robots, the shooter also evaluates the capabilities to shoot of the other robots, and, when one of them is better placed than itself, establishes a *shooting* contract with it.
  2. In the simulation implementation, a defender always tries to establish placing and blocking contracts with its nearest teammates by evaluating their ability to place themselves between itself and the ball, or between itself and a given opponent.
- The associated *commitment signs* are simply the roles of the robots, which are determined by the contracts they are engaged in. These signs are messages that are sent by the current trigger agent to the other robots (including itself). In the real robot games, by default, any nonshooter robot with no contract becomes a defender.
- The *joining behaviors* of a robot simply consist of behaving according to the role associated with the commitment sign it has received.
- The *dissolution behaviors* are defined for the shooter and defender robots. They cover two main situations:
  1. The shooter and defender can cancel any previous contract they have passed with a blocker or a placer if it happens that other robots have a better capability to place themselves or to block an opponent.

2. When a shooter passes a shooting contract with another robot, it cancels all the contracts it has passed previously.

Figure 5 shows an example of the dynamics of group formation taken from a SoccerServer game. The robots are labelled after their active behavior (S for shooter, P for placer, D for defender, B for blocker). The initial groups built around the defenders and the shooter evolve substantially during the game with respect to the position of the opponents and the ball. One can see, for example, that the attacking group is being dynamically reorganized around the shooter robot, whatever its identity. The defending groups also changes, although the one on the left side appears to be quite static—for a good reason: The defender (the goalie) remains the same throughout the game.

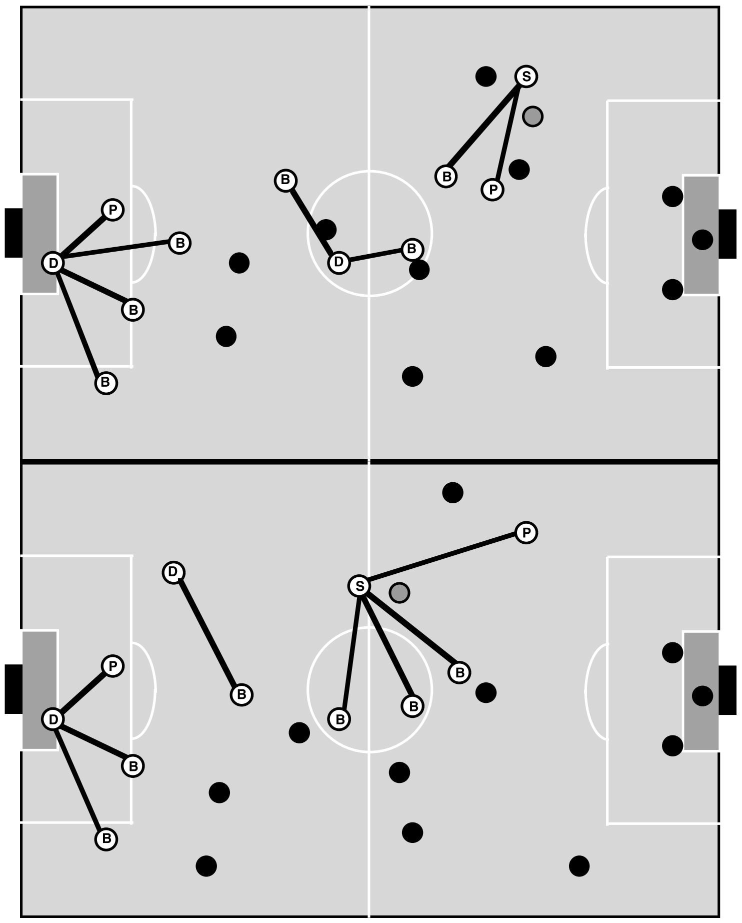

Figure 5. An example of the dynamics of group formation.

The competition between groups (for recruiting agents) is only ruled by the clock: The first offer is accepted, whatever its value and, until the commitment to its current group has been released, an agent will not accept other offers.

4.4. Preliminary results and work in progress

The real robots as well as the simulated ones have been demonstrated and tested during the MiroSot'96 [11] and RoboCup'97 [22] competitions. Although we did not win any of these competitions, the results appeared to be very satisfying: We won several games, and the ones we lost were performed against teams that paid a great deal of attention to the implementation of low-level behaviors (such as, for example, anticipating the movement of the ball, computing efficient trajectories, etc.), which we did not do in this current implementation (our elementary behaviors are very simple). However, Cassiopeia as a method for designing a MAS (that is, before the implementation step) has definitely proved useful since it has allowed us to evaluate various types of agent architectures (see next paragraph), without having to revise our analysis choices. We will use this important property for designing the second generation of Microb robots and agents that should enter the RoboCup'98 competition.

In a more theoretical way, we are currently using Cassiopeia to design two other types of soccer robot teams: a "reactive" and a "learning" one. When designing a reactive robot team, it is obviously not relevant to consider the third step of our method. Indeed, in reactive MAS, organizational behaviors are expected to implicitly emerge from the interactions between agents. The use of Cassiopeia is an attempt to clarify this kind of hypothesis. Regarding the design of the learning robot team, we want the agents to learn to play with "team spirit" (how to coordinate themselves, how to cooperate) given their elementary behaviors. The relational and organizational behaviors are then replaced by methods for learning them. By doing this, our aim is to prove that the three levels of behavior of Cassiopeia help in determining a simple, but powerful, ontology of the types of learning that can occur in a multi-agent system (see [7] for more details).

5. Conclusion

The major contributions of Cassiopeia are the abilities (1) to manipulate conceptual abstractions (e.g., influence, group) in a homogeneous way from the analysis step to the implementation step; (2) to interface the languages of the expert in robotics and the computer scientist by providing a conceptual level that is neither too sketchy nor too technical—this is mainly due to the use of an operational multi-agent oriented terminology[9]; (3) to clearly distinguish, at design time, between domain-dependent and organization-dependent behaviors.

Use of the method has also shown the limits of a simple methodological guideline. We need to refine the current terminology, especially when the terms are used by different DAI researchers with different meanings.[10] It is also necessary to integrate Cassiopeia, as a design method, with existing analysis methods (in particular object-oriented methods), which could allow a first step toward a domain-dependent structural and behavioral distribution. These improvements should enable Cassiopeia, currently used as a methodological guideline, to become a complete *agent-oriented design* method, qualified to provide the DAI community with methodological tools allowing (1) the development of ambitious multi-agent projects,[11] (2) more fruitful interactions between research, engineering, and industry, and (3) a more efficient re-use of their work.

Acknowledgments

The MICROB project is conducted by D. Duhaut (Robotics Laboratory of Paris) and A. Drogoul. This work was partially supported by special grants from the University of Paris 6, the CNRS, and the University of Versailles Saint-Quentin.

Notes

1. More details on the Cassiopeia method can be found in [1].
2. Also because most agent-based systems are programmed using object-oriented languages.
3. Although the existence of functional dependencies influences the type of organization that can be chosen, the notion of organization does not simply boil down to solving these dependencies. It can be a way of improving the quality of the collective behavior or a set of structural (and not functional) constraints thrust upon the agents. This is why the ability to use Cassiopeia in a top-down manner is important.
4. The expression "type of agent" is to be understood as an agent performing an elementary behavior at a given time, and not as a class (in the object-oriented way) of agent.
5. Which can then include the dependencies between elementary behaviors *within* an agent.
6. Inconsistent dependencies are dependencies that *structurally* cannot exist given constraints already defined at the level of the organization (for instance, dependencies between types of agents that are expected to operate in distant places and are unable to communicate or interact with one another); unnecessary dependencies are determined according to domain-dependent heuristics and with respect to the kind of organization the designer wants to obtain. He is explicitly responsible for their definition (which appears in the documentation of the design process).
7. Although we currently work on this issue. See [7] for a preliminary report.
8. Such a choice has to be written down in the documentation of the design process, since ignoring some dependencies may lead to a poor organization. However, considering all the dependencies would prevent seeing any potential groups.
9. The justification of each term is grounded in the role the terms have within the method.
10. For example, *dependency* and *organization*, for which a number of definitions may be found in the literature.
11. Cassiopeia has already been chosen by two major French companies (Peugeot SA and Dassault Aviation) to be incorporated into their software engineering processes. These projects are partially supported by the French Departments of Industry and Research and the two companies themselves.

References

1. R. J. Abbott, "Program design by informal English sentences," *Comm. ACM*, vol. **26**(11), pp. 882–894, 1983.

2. M. Asada, M. Kuniyoshi, A. Drogoul, H. Asama, M. Mataric, D. Duhaut, P. Stone, and H. Kitano, “The RoboCup physical agent challenge: Phase-I,” *Appl. Artif. Intell. J.*, to appear.
3. N. Avouris and L. Gasser (Eds.), *Distributed AI*: *Theory and Praxis*, Kluwer Academic, Boston, 1992.
4. G. Booch, *Object-Oriented Analysis and Design*, Benjamin Cummings, Redwood City, CA, 1994.
5. C. Castelfranchi, M. Miceli, and A. Cesta, “Dependence relations among autonomous agents,” in *Decentralized A. I. 3*, E. Werner and Y. Demazeau (Eds.), North-Holland, Amsterdam, 1992.
6. P. Coad and E. Yourdon, *Object-Oriented Analysis*, Yourdon Press, Englewood Cliffs, NJ, 1991.
7. A. Collinot and A. Drogoul, “Using the Cassiopeia method to design a soccer robot team,” Special Issue on RoboCup, *Appl. AI J.* to appear.
8. A. Collinot, P. Carle, and K. Zeghal, “Cassiopeia: A method for designing computational organizations,” *Proc. First Int. Workshop on Decentralized Intell. Multi-Agent Systems*, Poland, 1995.
9. K. S. Decker, “Distributed problem-solving techniques: A survey,” *IEEE Trans. Syst.*, *Man*, *Cybernetics* vol. 17, 1987.
10. Y. Demazeau and J.-P. Muller (Eds.), *Decentralized A.I. 2*, North-Holland, Amsterdam, 1991.
11. A. Drogoul and D. Duhaut, “MICROB: Making intelligent collective ROBotics,” in *Proc. MiroSot’96*, Taejon, Korea, 1996.
12. M. Erdmann and T. Lozano-Pérez, “On multiple moving objects,” in *Proc. IEEE Int. Conf. on Robotics and Automation*, 1986.
13. I. Graham, *Object-Oriented Methods*, Addison-Wesley, Reading, MA, 1994.
14. H. Kitano, A. Minoru, Y. Kuniyoshi, I. Noda, and E. Osawa, “RoboCup: The robot world cup initiative,” in *Proc. Workshop on Entertainment and AI/Alife*, *IJCAI*, 1995.
15. H. Kitano, M. Asada, E. Osawa, I. Noda, Y. Kuniyoshi, and H. Matsubara, “RoboGup: A challenge problem for AI,” *AI Mag.* vol. **18**(1), 1997.
16. C. Le Pape, “A combination of centralized and distributed methods for multi-agent planning and scheduling,” in *Proc. IEEE Int. Conf. on Robotics and Automation*, Cincinnati, 1990.
17. B. Moulin and L. Cloutier, “Collaborative work based on multi-agent architectures: A methodological perspective,” in *Soft Computing*: *Fuzzy Logic*, *Neural Networks and Distributed Artificial Intelligence*, F. Aminzadeh and M. Jamshidi (Eds.), Prentice-Hall, Englewood Cliffs, NJ, 1994.
18. M-J. Pont and E. Moreale, “Towards a practical methodology for agent-oriented software engineering with C++ and Java,” Leicester University, Dept. of Engineering, Leicester University, Technical Report 96-33.
19. J. Rosenschein and M. Genesereth, “Deals among rational agents,” *Proc. 9th Int. Joint Conf. on Artif. Intell.*, 1985.
20. J. Rumbaugh, M. Blaha, F. Eddy, W. Premerlani, and W. Lorensen, *Object Oriented Modeling and Design*, Prentice-Hall, Englewood Cliffs, NJ, 1991.
21. R. G. Smith and R. Davis, “The contract net protocol: High-level communication and control in a distributed problem-solver,” *IEEE Trans. Comput.*, vol. **C29**(12), 1980.
22. P. Stone and M. Veloso, “A layered approach to learning client behaviours in the robocup soccer server,” *Appl. Artif. Intell. J.*, to appear.
23. K. Sycara, “Multiagent compromise via negotiation,” in *Distributed Artificial Intelligence II*, Gasser and Huhns (Eds.), Morgan Kaufmann, San Mateo, CA, 1989.
24. E. Werner and Y. Demazeau (Eds.), *Decentralized A.I. 3*, North-Holland, Amsterdam, 1992.
25. M. Wooldridge, “The logical modelling of computational multi-agent systems,” UMIST, Manchester, England, Ph.D. Thesis, 1992.
26. M. J. Wooldridge, “Agent-based software engineering,” *IEEE Proc. Software Eng.* vol. **144**(1), pp. 26–37, 1997.
27. E. Yourdon, *Modern Structured Analysis*, Yourdon Press, Englewood Cliffs, NJ, 1989.