Chapter 3

Team Member Agent Architecture

In order to create a coherent *team of agents*, the entire agent architecture must be designed with the team in mind. "Collaboration must be designed into systems from the start; it cannot be patched on. [Grosz 96]"

A multi-agent system which involves several agents that collaborate towards the achievement of a joint objective is viewed as a *team* of agents. Most proposed teamwork structures (e.g. joint intentions [Cohen *et al.* 99], shared plans [Grosz 96]) rely on agents in a multi-agent system to negotiate and/or contract with each other in order to initiate team plans. However, in dynamic, real-time domains with unreliable communication, complex negotiation protocols may take too much time and/or be infeasible due to communication restrictions.

Simulated robotic soccer provides a time-critical environment in which agents in a team alternate between periods of limited and unlimited communication. Before games and at half-times, the team can effectively communicate with no limitations: each agent can be given the entire internal decision-making mechanisms of all of its teammates. However, as described in Section 2.2, during the course of a game, the agents must act independently in a dynamic, real-time, low-bandwidth communication environment: if the agents take the time to fully synchronize while playing, they may miss critical action opportunities and concede an advantage to the opponents.

These domain characteristics motivate the introduction of the concept of *periodic team synchronization* (PTS) domains. In PTS domains, during the limited communication periods, agents need to act autonomously, while still working towards a common team goal. Time-critical environments require real-time response and therefore eliminate the possibility of heavy communication among team agents. However, in PTS domains, agents can periodically synchronize in a safe, full-communication setting.

In this chapter, I describe our general team member agent architecture suitable for creating teams of agents in PTS domains [Stone and Veloso 99b]. This architecture includes a mechanism for defining pre-determined multi-agent protocols accessible to the entire team, called *locker-room agreements*. Within this team member agent architecture and using the locker-room agreement, we define our flexible teamwork structure that allows for task decomposition and dynamic role assignment in PTS domains [Stone and Veloso 99a]. In addition, we define a communication protocol in service of the locker-room agreement that is suitable for use during low-communication periods in a class of PTS domains: domains such as the

soccer server with single-channel, low-bandwidth communication environments [Stone and Veloso 98b].

The team member agent architecture described in this chapter defines a complete agent, including perception, cognition, and action. It is fully implemented as a simulated robotic soccer team.

This chapter is organized as follows. Section 3.1 expands upon the introduction of PTS domains and the two main contributions of this chapter: a flexible teamwork structure and a low-bandwidth communication paradigm. Section 3.2 presents the general agent architecture within which both the flexible teamwork structure and the low-bandwidth communication paradigm are situated. Sections 3.3 and 3.4 formally present the teamwork structure and the communication paradigm respectively. Section 3.5 gives details of our full implementations of both main contributions of this chapter within the simulated robotic soccer domain. Section 3.6 presents extensive empirical results testing the effectiveness of these implementations. Section 3.7 describes the implementation of the team member agent architecture and flexible formations within a second domain, namely real robotic soccer, and Section 3.8 is devoted to discussion and related work.

3.1 PTS Domains

We define periodic team synchronization domains as domains with the following characteristics:

- There is a team of autonomous agents A that collaborate towards the achievement of a joint long-term goal G.
- Periodically, the team can synchronize with no restrictions on communication: the agents can in effect inform each other of their entire internal states and decision-making mechanisms with no adverse effects upon the achievement of G. These periods of full communication can be thought of as times at which the team is "off-line."
- In general (i.e. when the agents are "on-line"):
  - The domain is *dynamic* and *real-time* meaning that team performance is adversely affected if an agent ceases to act for a period of time: G is either less likely to be achieved, or likely to be achieved farther in the future. That is, consider agent a_i. Assume that all other agent behaviors are fixed and that were a_i to act optimally, G would be achieved with probability p at time t. If a_i stops acting for any period of time and then resumes acting optimally, either:
    - G will be achieved with probability p' at time t with $p' < p$; or
    - G will be achieved with probability p at time t' with $t' > t$.
  - The domain has *unreliable communication*, either in terms of transmission reliability or bandwidth limits. In particular:
    - If an agent $a_i \in A$ sends a message m intended for agent $a_j \in A$, then m arrives with some probability $q < 1$; or

* Agent a_i can only receive x messages every y time units.

In the extreme, if $q = 0$ or if $x = 0$, then the periods of full communication are interleaved with periods of *no* communication, requiring the agents to act completely autonomously. In all cases, there is a cost to *relying* on communication. If agent a_i cannot carry on with its action until receiving a message from a_j, then the team's performance could suffer. Because of the unreliable communication, the message might not get through on the first try. And because of the dynamic, real-time nature of the domain, the team's likelihood of or efficiency at achieving G is reduced.

The soccer server provides a PTS domain since teams can plan strategies before the game, at halftime, or at other breakpoints; but during the course of the game, communication is limited. Its communication protocol involves a single, low-bandwidth, unreliable communication channel for all 22 agents (see Chapter 2).

In PTS domains, teams are long-term entities so that it makes sense for them to have periodic, reliable, private synchronization opportunities in which they can form off-line agreements for future use in unreliable, time-critical environments. This view of teams is complementary to teams that form on the fly for a specific action and keep communicating throughout the execution of that action as in [Cohen *et al.* 99]. Instead, in PTS domains, teams define coordination protocols during the synchronization opportunity and then disperse into the environment, acting autonomously with limited or no communication possible.

It has been claimed that pre-determined team actions are not flexible or robust to failure [Tambe 97]. In the context of PTS domains, a key contribution of our work is the demonstration that pre-determined multi-agent protocols can facilitate effective teamwork while retaining flexibility. We call these pre-determined protocols *locker-room agreements*. Formed during the periodic synchronization opportunities, locker-room agreements are remembered identically by all agents and allow them to coordinate efficiently.

In this chapter, I present the *team member agent architecture*, an agent architecture suited for team agents in PTS domains. The architecture allows for an agent to act collaboratively based on locker-room agreements.

A first approach to PTS domains is to break the task at hand into multiple rigid roles, assigning one agent to each role. Thus each component of the task is accomplished and there are no conflicts among agents in terms of how they should accomplish the team goal. However such an approach is subject to several problems: inflexibility to short-term changes (e.g. one robot is non-operational), inflexibility to long-term changes (e.g. a route is blocked), and a lack of facility for reassigning roles.

We introduce instead *formations* as a teamwork structure within the team member agent architecture. A formation decomposes the task space defining a set of roles with associated behaviors. In a general scenario with heterogeneous agents, subsets of homogeneous agents can flexibly switch roles within formations, and agents can change formations dynamically.

Within these PTS domains and our flexible teamwork structure, several challenges arise. Such challenges include:

- how to represent and follow locker-room agreements;
- how to determine the appropriate times for agents to change roles and/or formations;

- how to ensure that all agents are using the same formation; and
- how to ensure that all roles in a formation are filled: since the agents are autonomous and do not share memory, they could easily become uncoordinated.

Also within the team member agent architecture, we introduce a communication paradigm appropriate for agents in PTS domains with single-channel, low-bandwidth, unreliable communication during the dynamic, real-time (on-line) phases of operation. Not all PTS domains have such communication environments, but agents operating in those that do can implement this communication paradigm within their locker-room agreements.

In a nutshell, the contributions of this chapter are: the introduction of the concepts of PTS domains and locker-room agreements; the definition of a general team member agent architecture for defining a flexible teamwork structure; the facilitation of smooth transitions among roles and entire formations; a method for using roles to define pre-compiled multi-step, multi-agent plans; and techniques for dealing with the obstacles to inter-agent communication during the low-communication periods of PTS domains with single-channel, low-bandwidth, unreliable communication during the "on-line" periods.

In addition to simulated robotic soccer, there are several other examples of PTS domains, such as hospital/factory maintenance [Decker 96b], multi-spacecraft missions [Stone 97], search and rescue, and battlefield combat [Tambe 97]. There are also several other domains with similar communication requirements to the ones considered here. For example, aural communication in crowded settings is one. Both people and robots using aural sensors (e.g. [Fujita and Kageyama 97]) must contend with multiple simultaneous audible streams. They also have a limit to the amount of sound they can process in a given amount of time, as well as to the range within which communication is possible. Another example of such a communication environment is arbitrarily expandable systems. If agents are not aware of what other agents exist in the environment, then all agents must use a single universally-known communication channel, at least in order to initiate communication.

3.2 Architecture Overview

The team member agent architecture is suitable for PTS domains. Individual agents can capture locker-room agreements and respond to the environment while acting autonomously. Based on a standard agent paradigm, our team member agent architecture allows agents to sense the environment, to reason about and select their actions, and to act in the real world. At team synchronization opportunities, the team also makes a locker-room agreement for use by all agents during periods of limited communication. Figure 3.1 shows the functional input/output model of the architecture.

The agent keeps track of three different types of state: the *world state*, the *locker-room agreement*, and the *internal state*. The agent also has two different types of behaviors: *internal behaviors* and *external behaviors*.

The world state reflects the agent's conception of the current state of the real world, both via its sensors and via the predicted effects of its actions. This conception can be represented as a belief state in terms of probability distributions or confidence values.

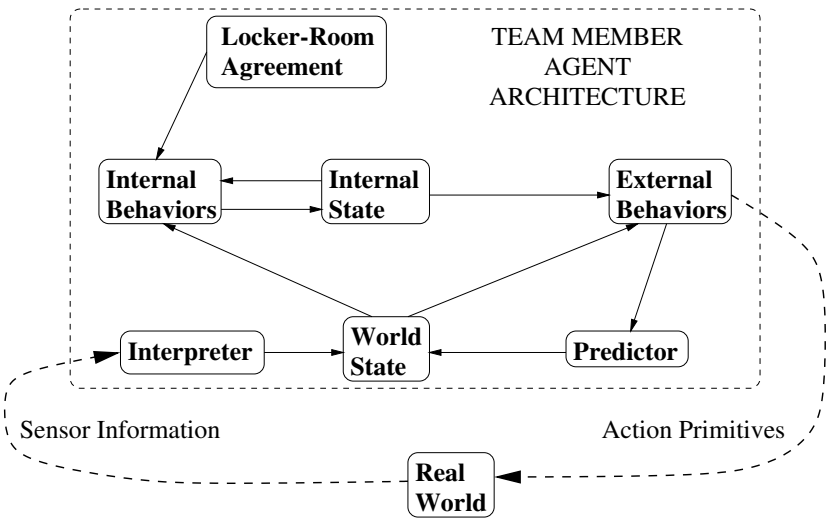


Figure 3.1: A functional input/output model of the team member agent architecture for PTS domains.

The world state is updated as a result of interpreted sensory information. It may also be updated according to the predicted effects of the external behavior module's chosen actions. The world state is directly accessible to both internal and external behaviors.

The locker-room agreement is set by the team when it is able to privately synchronize. It defines the flexible teamwork structure and the inter-agent communication protocols, if any. It is identical for all team members. The locker-room agreement is accessible only to internal behaviors.

The internal state stores the agent's internal variables. It may reflect previous and current world states, possibly as specified by the locker-room agreement. For example, the agent's role within a team behavior could be stored as part of the internal state. A window or distribution of past world states could also be stored as a part of the internal state. The agent updates its internal state via its internal behaviors.

The internal behaviors update the agent's internal state based on its current internal state, the world state, and the team's locker-room agreement.

The external behaviors reference the world and internal states, and select the actions to send to the actuators. The actions affect the real world, thus altering the agent's future percepts and predicted world states. External behaviors consider only the world and internal states, without direct access to the locker-room agreement.

Internal and external behaviors are similar in structure. They are both sets of condition/action pairs where conditions are logical expressions over the inputs and actions are themselves behaviors as illustrated in Figure 3.2. In both cases, a behavior is a directed acyclic graph (DAG) of arbitrary depth. The leaves of the DAGs are the behavior types' respective outputs: internal state changes for internal behaviors and action primitives for external behaviors. One leaf is illustrated in Figure 3.2.

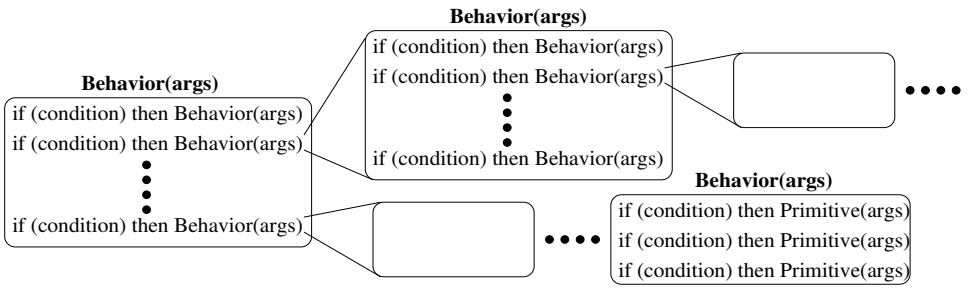


Figure 3.2: Behaviors in the team member agent architecture. Both internal and external behaviors are organized in directed acyclic graphs.

This notion of behavior is consistent with that laid out in [Mataric 94a]. In particular, behaviors can be nested at different levels: selection among lower-level behaviors can be considered a higher-level behavior, with the overall agent behavior considered a single "do-the-task" behavior. There is one such *top-level* internal behavior and one top-level external behavior; they are called when it is time to update the internal state or act in the world, respectively.

The following section introduces the teamwork structure that builds upon this team member agent architecture. The teamwork structure is designed for use in PTS domains. It exploits the locker-room agreement and the behavior definitions of the team member agent architecture.

3.3 Teamwork Structure

Common to all players, the locker-room agreement includes the team structure used by team members while they are acting in a time-critical environment with limited or no communication. In this section, I present our teamwork structure. It defines:

1. Flexible agent *roles* with protocols for switching among them;
2. Collections of roles built into team *formations*; and
3. Multi-step, multi-agent plans for execution in specific situations: *set-plays*.

The teamwork structure indirectly affects the agents' external behaviors by changing the agents' internal states via internal behaviors.

3.3.1 Roles

A *role*, r, consists of a specification of an agent's internal and external behaviors. The conditions and arguments of any behavior can depend on the agent's current role, which is a function of its internal state. At the extreme, a top-level external behavior could be a switch, calling an entirely different behavior sub-graph for each possible role. However, the role can affect the agent's overall behavior at any level of its behavior graph: it could affect just the arguments of a behavior deeply embedded in the behavior graph.

Roles may be *rigid*, completely specifying an agent's behavior. Or they may be *flexible*, leaving a certain degree of autonomy to the agent filling the role. For example, consider an agent that has access to a clock and that can blow a whistle. Role r could rigidly specify that the agent filling it must blow a whistle on the hour every hour. On the other hand, role r could leave some flexibility to the agent that fills it, specifying that no fewer than 25% but no more than 75% of the times that the hour changes, the agent must blow a whistle. In this case, the agent must stay within a parametric range in order to successfully fill the role, but on every given hour change, it can choose for itself what to do. By specifying ranges of parameters or behavior options, the agent filling role r can be given an arbitrary amount of flexibility. In this sense, a role can be thought of as a "soft constraint" on an agent's behavior.

A role in the robotic soccer domain, can be a position such as a midfielder. In the hospital maintenance domain, a role could specify the wing of the hospital whose floors the appropriate agent should keep clean, while in the web search domain, it could specify a server to search.

3.3.2 Formations

We achieve collaboration between agents through the introduction of *formations*. A formation decomposes the task space defining a set of roles. Formations include as many (possibly redundant) roles as there are agents in the team, so that each role is filled by one agent. In addition, formations can specify sub-formations, or *units*, that do not involve the whole team. A unit consists of a subset of roles from the formation, a *captain*, and intra-unit interactions among the roles.

For a team of n agents $A = \{a_1, a_2, \ldots, a_n\}$, any formation is of the form

$$F = \{R, \{U_1, U_2, \ldots, U_k\}\}$$

where R is a set of roles $R = \{r_1, r_2, \ldots, r_n\}$ such that $i \neq j \Rightarrow r_i \neq r_j$. Note that there are the same number of roles as there are agents. However, it is possible to define redundant roles such that the behavior specification of r_i is equivalent to that of r_j $(i \neq j)$. Each unit U_i is a subset of R: $U_i = \{r_{i1}, r_{i2}, \ldots, r_{ik}\}$ such that $r_{ia} \in R$, $a \neq b \Rightarrow r_{ia} \neq r_{ib}$ and r_{i1} is the *captain*, or unit leader. The map $A \mapsto R$ is not fixed: roles can be filled by different homogeneous agents. A single role may be a part of any number of units and formations.

Units are used to deal with local problem solving issues. Rather than involving the entire team in a sub-problem, the roles that address it are organized into a unit. Captains are unit-members with special privileges in terms of directing the other unit members.

Roles and formations are introduced independently from the agents that are to fill them. The locker-room agreement specifies an initial formation; an initial map from agents to roles; and run-time triggers for dynamic changing of formations. At any given time, each agent has an opinion as to what formation the team is currently using. Agents keep mappings $A \mapsto R$ from teammates to roles in the current formation. All this team structuring information is stored in the agent's internal state. It can be altered via the agent's internal behaviors.

Since agents are autonomous and operating in a PTS domain, during the periods of limited communication there is no guarantee that they will all think that the team is using

the same formation, nor that they have accurate maps $A \mapsto R$. In fact, the only guarantee is that each agent knows its own current role. Thus, in our implementation of the teamwork structure, we create robust behaviors for team agents which do not depend upon having correct, up-to-date knowledge of teammates' internal states: they degrade gracefully. When limited communication is available, efficient low-bandwidth communication protocols can allow agents to inform each other of their roles periodically. Figure 3.3 illustrates a team of agents smoothly switching roles and formations over time.

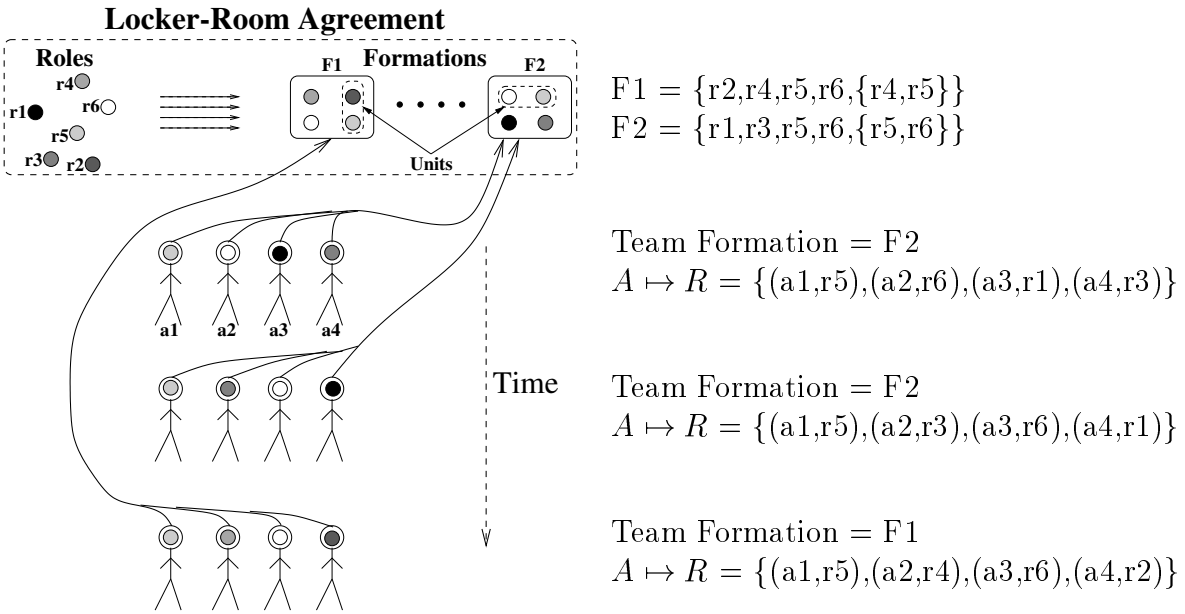


Figure 3.3: A team of agents smoothly switching roles and formations over time. Different roles are represented as differently shaded circles. Formations are possibly overlapping collections of roles. Units within the formations are indicated within a dotted enclosure. The definitions of all roles, formations, and units are part of the locker-room agreement and are known to all agents. An agent's current role is indicated by the shaded circle in its head and its current formation is indicated by an arrow to the formation. The agents first switch roles while staying in the same formation; then they switch to an entirely new formation.

3.3.3 Set-Plays

As a part of the locker-room agreement, the team can define multi-step, multi-agent plans to be executed at appropriate times. Particularly if there are certain situations that occur repeatedly, it makes sense for the team to devise plans for those situations ahead of time. We define a set-play as the combination of:

- A *trigger* condition indicating the set of states in which the set-play is activated; and
- A set of *set-play roles* $R_{sp} = \{spr_1, \ldots, spr_m\}$, $m \leq n$ defining the actions to be taken by the participants in the set-play. Each set-play role spr_i includes:
  - A *set-play behavior* to be executed; and

– A *termination* condition indicating the set of states in which an agent should cease filling the set-play role and resume its normal behavior.

The set-plays are defined in the locker-room agreement so that they are known to all agents on the team. Note that a set-play need not involve the entire team: $m \leq n$. The locker-room agreement also includes a general function to map roles in a formation to roles in a set-play: $R \mapsto R_{sp}$. Thus set-play roles are not assigned to pre-determined agents; instead they are filled by whichever agent is filling the appropriate role in the team's current formation.

3.4 Communication Paradigm

The teamwork structure defined in Section 3.3 is designed to be appropriate for all PTS domains. In the subclass of PTS domains with single-channel, low-bandwidth, unreliable communication during the periods of limited communication, such as the soccer server, the communication paradigm defined in this section is also appropriate. The communication paradigm further illustrates the use of the locker-room agreement within the team member agent architecture. Recall that PTS domains may have no communication possible during the "on-line" periods. In those cases, of course, the communication paradigm presented here does not apply.

Domains with single-channel, low-bandwidth, unreliable communication raise several challenges for inter-agent communication. The locker-room agreement can be used to make inter-agent communication more efficient and reliable. The five challenges addressed by our communication approach are:

1. Team members need some method of identifying which messages on the single channel are intended for which agent.

2. Since there is a single communication channel, agents must be prepared for active interference by hostile agents. A hostile agent could mimic messages it has previously heard at random times.

3. Since the communication channel has low bandwidth, the team must prevent itself from "talking all at once." Many communication utterances call for responses from all team members. However, if all team members respond simultaneously, few of the responses will get through.

4. Since communication is unreliable, agents must be robust to lost messages: their behaviors cannot depend upon receiving communications from a teammate.

5. Teams must determine how to maximize the chances that they are using the same team strategy (formation) despite the facts that each is acting autonomously and that communication is unreliable.

The characteristics and challenges of this communication environment are summarized in Table 3.1.

Communication Environment	**Challenges**
• Many agents, teams • Single-channel • Low-bandwidth • Unreliable	• Message targeting and distinguishing • Robustness to active interference • Multiple simultaneous responses • Robustness to lost messages • Team coordination

Table 3.1: The characteristics and challenges of the type of communication environment considered in this section.

In order to meet these challenges, we specify that a team uses messages with the following fields, all of whose syntax and semantics are defined within the locker-room agreement:

- The <team-identifier> identifies messages from within the team as opposed to another team in an adversarial environment.

- The <unique-team-member-ID> is a different sequential integer assigned to each team member.

- The <encoded-time-stamp> is a security code that can be used to verify a message's authenticity.

- The <time-stamped-team-strategy> indicates the current formation that the sender believes the team is using.

- The <selected-internal-state> contains portions of the sender's internal state.

- The <message-type> and <message-data> contain the semantic content of the individual message. The messages can use any syntactic and semantic codes (KQML [Finin *et al.* 94] and KIF [Genesereth and Fikes 92] for example). The locker-room agreement also includes a mapping from message type to response requirements.

- The <target> indicates the intended recipient(s) of the message. It could be intended for a single team member identified either by <unique-team-member-ID> or by role within the team's current formation; for a unit of the current formation; or for all team members.

In addition to this protocol which is defined within the locker-room agreement, some internal state variables need to be devoted to communication. When an agent hears a message, it interprets it and updates the world state to reflect any information transmitted by the message. It also stores the content of the message as a special variable `last-message`. Furthermore, based on the locker-room agreement, an internal behavior then updates the internal state. If the message requires a response, three variables in the internal state are manipulated by an internal behavior: `response`, `response-flag`, and `communicate-delay`. `response` is the actual response that should be given by the agent as determined in part by the locker-room agreement. All three of these variables are then referenced by an external behavior to

determine when a response should be given. For example one condition-action pair of the top-level external behavior might be: `if (response-flag set and communicate-delay==0) then say(response)`.

These message fields and internal state variables are components of our novel communication paradigm and represent contributions of our team member agent architecture. The remainder of this section details how these particular message fields and internal state variables can be used to meet the challenges summarized in Table 3.1.

3.4.1 Message Targeting and Distinguishing

Since there is a single communication channel, agent a_i hears messages sent by all agents whether or not they are intended for it. Messages sent by agents from another team are completely ignored. Messages sent by a teammate are identified by the <team-identifier> field. Since all team members know the locker-room agreement, agents monitor all messages from teammates to determine their teammates' internal states, even if the content of the message is intended for another teammate.

Agents can distinguish messages that are intended for them by checking the <team-identifier> and <target> fields. An agent a_i pays attention to a message from a member of the same team that is targeted to a_i, to the entire team, or to some subset of the team that includes a_i. The <target> field could identify an individual agent either by its unique ID number or by the role that it is currently playing. Thus, a message could be sent to the agent playing a particular role without knowing which agent that is. Similarly, a message could be targeted towards all agents in a unit of the team's current formation.

3.4.2 Robustness to Active Interference

The only further difficulty related to an agent distinguishing which messages are intended for it arises in the presence of active interference. Consider a hostile agent h which hears a message that is directed to a_i at time t. h has full access to the message since all agents use the same communication channel. Thus if h remembers the message and sends an identical message at time u, agent a_i will mistakenly believe that the message is from a teammate. Although the message was appropriate at time t, it may be obsolete at time u and it could potentially confuse a_i as h intends.

This potential difficulty is avoided with the <encoded-time-stamp> field. Even a simple time stamp is likely to safeguard against interference since h is not privy to the locker-room agreement: it does not necessarily know which field is the time stamp. However, if h discovers which field is the time stamp by noticing that it always matches the time of the message, it could alter the field based on the time elapsed between times t and u. Indeed, if there is a globally accessible clock, h would simply have to replace t with u in the message. However, the team can safeguard against such interference techniques by encoding the time-stamp using an injective function chosen as a part of the locker-room agreement. This function can use any of the other message fields as arguments in order to make decryption as difficult as possible. The only requirement is that a teammate receiving the message can invert the function to determine the time at which the message was sent. If the time at which it was

sent is either too far in the past or in the future (according to the locker-room agreement), then the message can be safely ignored. In particular, the locker-room agreement has a variable `message-lag-tolerance` encoding this time. If a message sent at time t arrives at time u with $u - t >$ `message-lag-tolerance`, then the message is ignored.

By observing enough messages and comparing them with the actual time, it is theoretically possible for hostile agents to crack simple codes and alter the <encoded-time-stamp> field appropriately before sending a false message. However, the function can be made arbitrarily complex so that such a feat is intractable within the context of the domain. If secrecy is critical and computation unconstrained, a theoretically safe encryption scheme can be used. The degree of complexity necessary depends upon the number of messages that will be sent after the locker-room agreement. With few enough messages, even a simple linear combination of the numerical message fields suffices.

3.4.3 Multiple Simultaneous Responses

The next challenge to meet is that of messages that require responses from several teammates. However, not all messages are of this type. For example, a message meaning "where are you?" requires a response, while "look out behind you" does not. Therefore it is first necessary for agents to classify messages in terms of whether or not they require responses as a function of the <message-type> field. Since the low-bandwidth channel prevents multiple simultaneous responses, the agents must also reason about the number of intended recipients as indicated by the <target> field. Taking these two factors into account, there are six types of messages, indicated here as a1,a2,a3,b1,b2, and b3:

	Response requested	
Message Target	no	yes
Single agent	a1	b1
Whole team	a2	b2
Part of team	a3	b3

When hearing any message, the agent updates its internal belief of the other agent's status as indicated by the <time-stamped-team-strategy> and <selected-internal-state> fields. However, only when the message is intended for it does it consider the content of the message. Then it uses the following algorithm in response to the message:

1. If the message requires no response (types a1–3), the agent simply updates its internal state.

2. If the message requires a response then set `response` to the appropriate response message, `response-flag` = 1 and

   - If the agent was the only target (type b1), respond immediately: `communicate-delay` = 0;
   - If the message is sent to more than one target (types b2 and b3), set `communicate-delay` based on the difference between the <unique-team-member-ID> of the message sender and that of the receiver. Thus each teammate responds at a different time, leaving time for teammate messages to go through.

An internal behavior keeps decrementing `communicate-delay` as time passes. An external behavior uses the communication condition-action pair presented above:

`if (response-flag set and communicate-delay==0) then say(response)`

where say is an actuator primitive. Players also set the `communicate-delay` variable in the event that they need to send multiple messages to the same agent in a short time. This communication paradigm allows agents to continue acting in real-time while reasoning about the appropriate time to communicate.

3.4.4 Robustness to Lost Messages

In order to meet the challenge raised by unreliable communication leading to lost messages, agents must not depend on communication to act. Communication is structured so that it helps agents update their world and internal states. But agents do not stop acting while waiting for communications from teammates. As brought up in [Tambe 96a], such a case could cause infinite looping if a critical teammate fails to respond for any reason. As well as continuing to act while waiting for `communicate-delay` to expire, agents ensure that they do not rely on inter-agent communication by continuing to act while waiting for responses from teammates. They also maintain world and internal states without help from teammates. Communication can improve the reliability of an agent's world state by elucidating some of an agent's hidden state; but communication is not necessary for an agent to maintain a reasonable approximation of the world state.

3.4.5 Team Coordination

Finally, team coordination is difficult to achieve in the face of the possibility that autonomous team members may not agree on the <time-stamped-team-strategy> or the mapping from teammates to roles within the team strategy. Again, there are no disastrous results should team members temporarily adopt different strategies; however they are more likely to achieve their goal G if they can stay coordinated.

One method of coordination is via the locker-room agreement. Agents agree on globally accessible environmental cues as triggers for switches in team strategy. Another method of coordination which complements this first approach is via the time stamp. When hearing a message from a teammate indicating that the team strategy is different from the agent's current idea of the team strategy, the agent adopts the more recent team strategy: if the received message's team strategy has a time-stamp that is more recent than that on the agent's current team strategy, it switches; otherwise it keeps the same team strategy and informs its teammate of the change. Thus changes in team strategy can quickly propagate through the team.

In particular, suppose that agent a_i hears at time t that the team formation is F_1. It then hears a message from agent a_j indicating that the team formation was set to F_2 at time u. If $t < u$, then F_2 is a more recent team decision and it updates its notion of the team's formation to F_2. However, if $u < t$, it is agent a_j that has an obsolete view of the formation. a_i then sends a message to a_j indicating in the <time-stamped-team-strategy> field that the formation was set to F_1 at time t, thus causing a_j to update its notion of the

team's formation. In the event that $t = u$, the locker-room agreement must specify an order of precedence among roles in order for the agents to determine which agent's idea of the current formation to regard as correct.

Depending on the available bandwidth in the particular application, the <selected-internal-state> can also be used to facilitate team coordination by helping to keep the team members up-to-date regarding the mapping $A \mapsto R$, and perhaps regarding object locations that might be hidden to individual agents.

3.5 Implementation in Robotic Soccer

Robotic soccer is a very good example of a PTS domain: teams can coordinate before the game, at half-time, and at other break points, but communication is limited during play. In addition, as described in Chapter 2, the soccer server models a single-channel, low-bandwidth, unreliable communication environment.

This section provides a detailed description of a specific implementation of the team member agent architecture, the teamwork structure, and the communication approach presented in Sections 3.2–3.4. The implementation is that of the CMUnited-98 simulated robotic soccer team.

The implementation is described in great detail. In particular, Section 3.5.1 presents the low-level action cycle, skills, and world model of CMUnited-98 agents. Further details, particularly implementation details of the skills, are available in Appendix B. The low-level details presented in Section 3.5.1 are important for gaining an understanding of the basic agent perception and action capabilities. In the context of the team member agent architecture, they represent the interpreter, the world state, the predictor, and the external behaviors. These portions of the agent architecture are concerned principally with issues relating to an individual agent. On the other hand, the teamwork structure and communication paradigm implementations, presented in Sections 3.5.2 and 3.5.3, deal with collaborative, team issues. They are defined within the locker-room agreement, internal behaviors, and internal state.

3.5.1 Action Cycle and World Model

This section defines the portions of the CMUnited-98 simulated robotic soccer agent implementation that relate to an individual agent. In the context of the team member agent architecture presented in Section 3.2, this section covers the interpreter, the world state, the predictor, and the external behaviors.

Timing Actions

CMUnited-98 agents are capable of perception, cognition, and action. By perceiving the world, they build a model of its current state. Then, based on a set of behaviors, they choose an action appropriate for the current world state.

A driving factor in the design of the agent architecture is the fact that the simulator operates in fixed cycles of length `simulator_step` (100 msec). As presented in Section 2.2, the simulator accepts commands from clients throughout a cycle and then updates the world

state all at once at the end of the cycle. Only one action command (dash, kick, turn, or catch) is executed for a given client during a given cycle.

Therefore, agents (simulator clients) should send exactly one action command to the simulator in every simulator cycle. If more than one command is sent in the same cycle, a random one is executed, possibly leading to undesired behavior. If no command is sent during a simulator cycle, an action opportunity has been lost: opponent agents who have acted during that cycle may gain an advantage. In the team member agent architecture, sending an action corresponds to executing the top-level external behavior.

Since the simulator updates the world at the end of every cycle, it is advantageous to try to determine the state of the world at the end of the previous cycle when choosing an action for the current cycle. As such, the basic agent loop during a given cycle t is as follows:

- Assume the agent has consistent information about the state of the world at the end of cycle $t-2$ and has sent an action during cycle $t-1$.
- While the server is still in cycle $t-1$, upon receipt of a sensation (see, hear, or sense_body), process the sensation in the interpreter and store the new information in temporary structures. Do not update the world state.
- When the server enters cycle t (determined either by a running clock or by the receipt of a sensation with time stamp t), use all of the information available (temporary information from sensations and predicted effects of past actions) to **update the world state** to match the server's world state (the "real world state") at the end of cycle $t-1$. Then **choose and send an action** to the server for cycle t: execute the top-level external behavior.
- Repeat for cycle $t+1$.

While the above algorithm defines the overall agent loop, much of the challenge is involved in updating the world state effectively and choosing an appropriate action. The remainder of this section goes into these processes in detail.

The World State

When acting based on a world model, it is important to have as accurate and precise a model of the world as possible at the time that an action is taken. In order to achieve this goal, CMUnited-98 agents gather sensory information over time, extracting its meaning via the interpreter, and process the information by incorporating it into the world state immediately prior to acting.

There are several objects in the world, such as the goals and the field markers which remain stationary and can be used for self-localization. Mobile objects are the agent itself, the ball, and 21 other players (10 teammates and 11 opponents). These objects are represented in a type hierarchy as illustrated in Figure 3.4.

Each agent's world state stores an instantiation of a stationary object for each goal, sideline, and field marker; a ball object for the ball; and 21 player objects. Since players can be seen without their associated team and/or uniform number, the player objects are

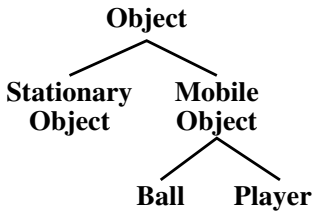


Figure 3.4: The agent's object type hierarchy.

not identified with particular individual players. Instead, the variables for team and uniform number can be filled in as they become known.

Mobile objects are stored with confidence values within [0,1] indicating the confidence with which their locations are known. The confidence values are needed because of the large amount of hidden state in the world: no object is seen consistently. While it would be a mistake to only remember objects that are currently in view, it is also wrong to assume that a mobile object will stay still (or continue moving with the same velocity) indefinitely. By decaying the confidence in unseen objects over time, agents can determine whether or not to rely on the position and velocity values [Bowling *et al.* 96].

All information is stored as global coordinates even though both sensor and actuator commands are specified in relative coordinates (angles and distances relative to the agent's position on the field). Global coordinates are easier to store and maintain as the agent moves around the field because the global coordinates of stationary objects do not change as the agent moves, while the relative coordinates do. It is a simple geometric calculation to convert the global coordinates to relative coordinates on demand as long as the agent knows its own position on the field.

The variables associated with each object type are as follows:

Object :

- Global (x, y) position coordinates
- Confidence within [0,1] of the coordinates' accuracy

Stationary Object : nothing additional

Mobile Object :

- Global (dx, dy) velocity coordinates
- Confidence within [0,1] of the coordinates' accuracy

Ball : nothing additional

Player :

- Team
- Uniform number
- Global θ facing angle
- Confidence within [0,1] of the angle's accuracy

Updating the World State

Information about the world can come from

- Visual information;
- Audial information;
- Sense_body information; and
- Predicted effects of previous actions.

Visual information arrives to the interpreter as relative distances and angles to objects in the player's view cone. Audial information could include information about global object locations from teammates. Sense_body information pertains to the agent's own status including stamina, view mode, and speed.

Whenever new information arrives at the interpreter, it is stored in temporary structures with time stamps and confidences (1 for visual information, possibly less for audial information). Visual information is stored as relative coordinates until the agent's exact location is determined.

When it is time to act during cycle t, all of the available information is used to best determine the server's world state at the end of cycle $t-1$. If no new information arrived pertaining to a given object, the velocity and actions taken are used by the predictor to predict the new position of the object and the confidence in that object's position and velocity are both decayed.

When the agent's world state is updated to match the end of simulator cycle $t-1$, first the agent's own position is updated to match the time of the last sight; then those of the ball and players are updated.

The Agent Itself: Since visual information is given in coordinates relative to the agent's position, it is important to determine the agent's exact position at the time of the sight. When updating the world state to match the end of simulator cycle $t-1$, there may have been visual information with time stamp $t-1$ and/or t (anything earlier would have been incorporated into the previous update of the world state at the end of cycle $t-1$).

If the latest visual information has time stamp $t-1$, then the agent's own position is not updated until after the other objects have been updated since their coordinates are given relative to the old agent position. On the other hand, if the latest visual information has time stamp t, or if there has been no new visual information since the last world-state update, the agent's status can be updated immediately.

In either case, the following process is used to update the information about the agent:

- If new visual information has arrived:
  - The agent's position can be determined accurately by using the relative coordinates of one seen line and the closest stationary object.
- If no visual information has arrived:
  - Bring the velocity up to date, possibly incorporating the predicted effects of any actions (a dash) taken during the previous cycle.

  - Using the previous position and velocity, predict the new position and velocity.
- If available, reset the agent's speed as per the sense_body information. Assume velocity is in the direction the agent is facing.
- Bring the player's stamina up to date either via the sense_body information or from the predicted action effects.

The Ball: As the key focus of action initiative in the domain, the ball's position and velocity drives a large portion of the agents' decisions. As such, it is important to have as accurate and up-to-date information about the ball as possible.

The ball information is updated as follows:

- If there was new visual information, use the agent's absolute position at the time (determined above), and the ball's temporarily stored relative position to determine the ball's absolute position at the time of the sight.
- If velocity information is given as well, update the velocity. Otherwise, check if the old velocity is correct by comparing the new ball position with the expected ball position.
- If no new visual information arrived or the visual information was from cycle $t-1$, estimate its position and velocity for cycle t using the values from cycle $t-1$. If the agent kicked the ball on the previous cycle, the predicted resulting ball motion is also taken into account.
- If the ball should be in sight (i.e. its predicted position is in the player's view cone), but isn't (i.e. visual information arrived, but no ball information was included), set the confidence to 0.
- Information about the ball may have also arrived via communication from teammates. If any heard information would increase the confidence in the ball's position or velocity at this time, then it is used as the correct information. Confidence in teammate information can be determined by the time of the information (did the teammate see the ball more recently?) and the teammate's distance to the ball (since players closer to the ball see it more precisely).

Ball velocity is particularly important for agents when determining whether or not (or how) to try to intercept the ball, and when kicking the ball. However, velocity information is often not given as part of the visual information string, especially when the ball is near the agent and kickable. Therefore, when necessary, the agents attempt to infer the ball's velocity indirectly from the current and previous ball positions.

Teammates and Opponents: The biggest challenge in keeping track of player positions is that the visual information often does not identify the player that is seen (see Figure 2.4). One might be tempted to ignore all ambiguously-specified players. However, for strategic planning it is very useful to have a complete picture of the player positions around the field.

In general, player positions and velocities are determined and maintained in the same way as in the case of the ball. A minor addition is that the direction a player is facing is sometimes available from the visual information.

When a player is seen without full information about its identity, previous player positions can be used to help disambiguate the identity. Knowing the maximum distance a player can move in any given cycle, it is possible for the agent to determine whether a seen player could be the same as a previously identified player. If it is physically possible, the agent assumes that they are indeed the same player.

Since different players can see different regions of the field in detail, communication can play an important role in maintaining accurate information about player locations.

From the complete set of player locations, an agent can determine both the defensive and offensive offsides lines. It is particularly important for forwards to stay in front of the last opponent defender in order to avoid being called offsides. Forwards periodically look towards the opponent defenders in order to increase the reliability of their location information.

Agent Skills: Low-Level External Behaviors

Once the agent has determined the server's world state for cycle t as accurately as possible, it can choose and send an action to be executed at the end of the cycle. That is, it must execute its external behavior. At the top level, it first chooses its *behavior mode* or its local goal within the team's overall strategy. The behavior mode determines the subgraph of the external behavior to be executed. The subgraphs contain several low-level *skills* which provide the agent with basic capabilities. The output of the skills are primitive movement commands.

The skills available to CMUnited-98 players include kicking, dribbling, ball interception, goaltending, defending, and clearing. The implementation details of these skills are described in Appendix B.

The common thread among these skills is that they are all *predictive, locally optimal skills* (PLOS). They take into account predicted world states as well as predicted effects of future actions in order to determine the optimal primitive action from a local perspective, both in time and in space.

One simple example of PLOS is each individual agent's stamina management. The server models stamina as having a replenishable and a non-replenishable component. Each is only decremented when the current stamina goes below a fixed threshold. Each player monitors its own stamina level to make sure that it never uses up any of the non-replenishable component of its stamina. No matter how fast it should move according to the behavior the player is executing, it slows down its movement to keep itself from getting too tired. While such behavior might not be optimal in the context of the team's goal, it is locally optimal considering the agent's current tired state.

Even though the skills are predictive, the agent *commits* to only one action during each cycle. When the time comes to act again, the situation is completely reevaluated. If the world is close to the anticipated configuration, then the agent will act similarly to the way

it predicted on previous cycles. However, if the world is significantly different, the agent will arrive at a new sequence of actions rather than being committed to a previous plan. Again, it will only execute the first step in the new sequence.

Behavior Modes: The Top-Level External Behavior

Given all of the individual skills available to the CMUnited-98 agents, it becomes a significant challenge to coordinate the team so that the players are not all trying to do the same thing at the same time. Of course one and only one agent should execute the goaltending behavior. But it is not so clear how to determine when an agent should move towards the ball, when it should defend, when it should dribble, or clear, etc.

A player's top-level external behavior decision is its behavior mode. Implemented as a rule-based system, the behavior mode determines the abstract behavior that the player should execute. For example, there is a behavior mode for the set of states in which the agent can kick the ball. Then, the decision of what to do with the ball is made by way of a more involved decision mechanism represented as a subgraph of the external behavior. On each action cycle, the first thing a player does is re-evaluate its behavior mode.

The behavior modes include:

Goaltend: Only used by the goaltender.

Localize: Find own field location if it's unknown.

Face Ball: Find the ball and look at it.

Handle Ball: Used when the ball is kickable.

Active Offense: Go to the ball as quickly as possible. Used when no teammate could get there more quickly.

Auxiliary Offense: Get open for a pass. Used when a nearby teammate has the ball.

Passive Offense: Move to a position likely to be useful offensively in the future.

Active Defense: Go to the ball even though another teammate is already going. Used in the defensive end of the field.

Auxiliary Defense: Mark an opponent.

Passive Defense: Track an opponent or go to a position likely to be useful defensively in the future.

The detailed conditions and effects of each behavior mode are described in Appendix C. However, they will also become more clear as the role-based flexible team structure is described in Section 3.5.2.

The remainder of this section details the implementations of both main contributions of this chapter: the flexible teamwork structure and the communication paradigm. Both implementations are unified within the CMUnited simulated robotic soccer system, building upon the action cycle and world state model presented here.

3.5.2 Teamwork Structure Implementation

One approach to task decomposition in the soccer server is to assign fixed coordinate positions to agents.[1] Such an approach leads to several problems: i) short-term inflexibility in that the players cannot adapt their positions to the ball's location on the field; ii) long-term inflexibility in that the team cannot adapt to opponent strategy; and iii) local inefficiency in that players often get tired running across the field back to their positions after chasing the ball. Our formations allow for flexible teamwork and combat these problems. (The term "position" is often used to denote the concept of "role" in the soccer domain. In this section I use the two terms interchangeably.)

This section describes the CMUnited simulator team implementation of the teamwork structure presented in Section 3.3. In the context of the team member agent architecture in Section 3.2, it covers the locker-room agreement, the internal behaviors, and the internal state.

Domain Instantiations of Roles and Formations

Figure 3.5 shows a simplified top-level external behavior used by a team agent. The agent's top priority is to locate the ball. If the ball's location is known, it moves towards the ball or goes to its position (i.e. to assume its role), depending on its internal state. It also responds to any requested communications from teammates. The sub-behaviors of the top-level behavior are all behavior modes.

External Behavior: Play Soccer()

If (Ball Lost)	Face Ball()
If (Ball known AND Chasing)	Handle Ball(args1)
If (Ball known AND Not Chasing)	Passive Offense(args2)
If (Commuincate Flag Set)	Communicate()

Figure 3.5: An example of a simplified top-level external behavior for a robotic soccer player.

The referenced "handle ball" and "passive offense" behaviors may be affected by the agent's current role and/or formation. Such effects are realized by references to the internal state either at the level of function arguments (args1, args2), or within sub-behaviors. None of the actions in the condition-action pairs here are action primitives; rather, they are calls to lower level behaviors.

The definition of a position includes *home coordinates*, a *home range*, and a *maximum range*, as illustrated in Figure 3.6. The position's home coordinates are the default location to which the agent should go. However, the agent has some flexibility, being able to set its actual home position anywhere within the home range. When moving outside of the max range, the agent is no longer considered to be in the position. The home and max ranges of different positions can overlap, even if they are part of the same formations.

A formation consists of a set of positions and a set of units (as defined in Section 3.3.2). The formation and each of the units can also specify inter-position behavior specifications

[1] One of the teams in Pre-RoboCup-96 used and depended upon these assignments: the agents passed to the fixed positions regardless of whether or not there was a teammate there.

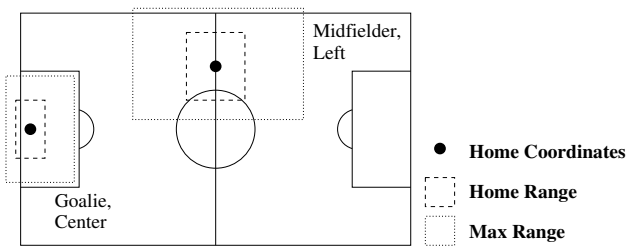
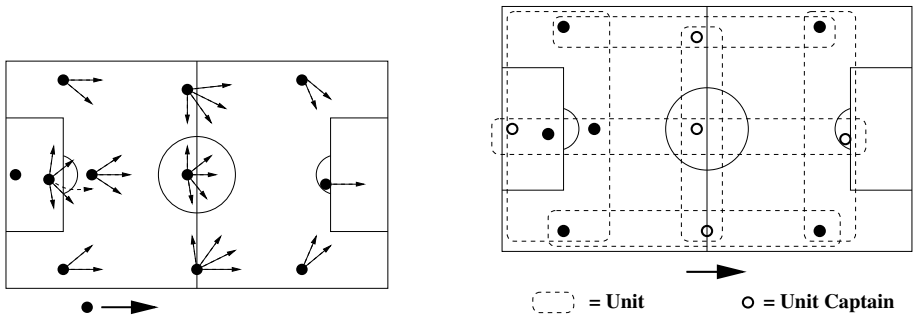


Figure 3.6: Different positions with home coordinates and home and max ranges.

for the member positions, as illustrated in Figure 3.7(a). In this case, the formations specify inter-role interactions, namely the positions to which a player should consider passing the ball. We use decision tree learning to enable players to decide where to pass from among these options (see Chapter 6). Figure 3.7(b) illustrates the units, the roles involved, and their captains. Here, the units contain defenders, midfielders, forwards, left players, center players, and right players.



(a) (b)

Figure 3.7: (a) A possible formation (4-3-3) for a team of 11 players. Arrows represent passing options. (b) Positions can belong to more than one unit.

Since the players are all autonomous, in addition to knowing its own role, each one has its own belief of the team's current formation along with the time at which that formation was adopted, and a map of teammates to positions. Ideally, the players have consistent beliefs as to the team's state, but this condition cannot be guaranteed between synchronization opportunities.

For example, consider the passive offense behavior in Figure 3.5. This external behavior references the agent's internal state via a series of function calls. Specifically, the agent is to move to some location within the home range of its current position in the team's current formation:

$$\texttt{TargetLocation} \in \texttt{HomeRange}(\texttt{MyPosition}(\texttt{CurrentFormation}())) \tag{3.1}$$

where `HomeRange(p)` returns the home range of position p; `MyPosition(f)` returns a player's own current position in formation f; and `CurrentFormation()` returns the player's own opinion of the team's current formation. Thus the internal behaviors that determine the player's current position and formation affect its external behavior. Notice that by specifying

a range of possible locations, the role leaves some flexibility to the autonomous player: it can choose which specific `TargetLocation` to move to within the range.

Different roles can also have entirely different external behaviors. As presented in Section 3.3.1, each role could have an entirely different external behavior subgraph.

Our teamwork structure for PTS domains allows for several significant features in our simulated soccer team. These features are: (i) the definition of switching among multiple formations with units; (ii) flexible position adjustment and position switching; (iii) and pre-defined, special-purpose plays (set-plays).

Dynamic Switching of Formations

We implemented several different formations, ranging from very defensive (8-2-0) to very offensive (2-4-4).[2] The full definitions of all of the formations are a part of the locker-room agreement. Therefore, they are all known to all teammates. However during the periods of full autonomy and limited communication, it is not necessarily known what formation the rest of the teammates are using. Three approaches can be taken to address this problem:

Static formation: the formation is set by the locker-room agreement and never changes;

Run-time formation switch: during team synchronization opportunities, the team sets globally accessible run-time evaluation metrics as formation-changing indicators.

Communication-triggered formation switch: one team member decides that the team should switch formations and communicates the decision to teammates.

Both run-time formation switches and communication-triggered formation switches are internal behaviors. The run-time triggers and communication protocols are defined in the locker-room agreement. When a run-time evaluation metric indicates that the formation should change, or when a heard communication triggers a formation change, an internal behavior changes the player's opinion of the team's formation in its internal state.

This change in internal state can then affect external behaviors. For example, a switch in formations changes the output of the `CurrentFormation()` function in Equation 3.1. The outputs of `MyPosition()` and `HomeRange()` are also altered: the new formation consists of a different collection of roles with different home ranges. Thus the passive offense external behavior changes as a result of the formation switch.

The CMUnited simulator team uses run-time formation switches. Based on the amount of time left relative to the difference in score, the team switches to an offensive formation if it was losing near the end of the game and a defensive formation if it was winning. Specifically, the team starts out in a 4-4-2 formation. If $\Delta Minutes$ is the number of minutes left in the game, and $\Delta Score$ is the difference in score ($\Delta Score > 0$ if the team is winning; $\Delta Score < 0$ if the team is losing), then the team uses the following run-time algorithm:

- If $\Delta Score < 0$ and $-\Delta Score \geq \Delta Minutes$, then switch to a 3-3-4 formation;

[2]Soccer formations are typically described as X-Y-Z where X, Y, and Z are the number of defenders, midfielders, and forwards respectively. It is assumed that the eleventh player is the goaltender [LaBlanc and Henshaw 94].

- If $\Delta Score > 0$ and $\Delta Score \geq \Delta Minutes$, then switch to a 8-2-0 formation;
- Otherwise switch to (or stay in) a 4-4-2 formation.

Since each agent is able to independently keep track of the score and time, the agents are always able to switch formations simultaneously.

Communication-triggered formation switches have also been implemented and tested. Details are presented in the context of the communication paradigm implementation (Section 3.5.3).

Flexible Positions

As emphasized throughout, homogeneous agents can play different positions. But such a capability raises the challenging issue of when the players should change positions. In addition, with teammates switching positions, a player's internal player-position map $A \mapsto R$ could become incorrect and/or incomplete. The locker-room agreement provides procedures to the team that allow for coordinated role changing. In CMUnited's case, the locker-room agreement designates an order of precedence for the positions within each unit. Unless their pursuit of the ball leads them from their position, players only switch into a more important position than their current position.

By switching positions within a formation, the overall joint performance of the team can be improved. Position-switching has the potential to save player energy and to allow them to respond more quickly to the ball. However, switching positions can also cause increased player movement if a player has to move across the field to occupy its new position. Players must weigh the possible costs and benefits before deciding to switch positions.

Like switching formations, switching positions can change external behaviors via their references to the internal state. In Equation 3.1, switching positions changes the value returned by `MyPosition()`, thus also affecting the value of `HomeRange()`: the player executing the passive offense external behavior chooses its location from a different range of possible positions.

In addition to having the flexibility to switch to a different position, CMUnited-98 agents also have flexibility *within* their positions. That is, the external behavior references the internal state to determine a range of possible locations that are determined by the player's current position. However, within this range, the role does not specify the player's precise location. For example, in the passive offense external behavior (equation 3.1), the player must choose its `TargetLocation` from within the home range of its current position.

In the CMUnited multi-agent approach, the player positions itself flexibly such that it *anticipates* that it will be useful to the team, either offensively or defensively. The agents can exercise this flexibility within its external behaviors in three ways:

- Opponent marking;
- Ball-dependent positioning;
- Strategic positioning using attraction and repulsion (SPAR).

When marking opponents, agents move next to a given opponent rather than staying at the default position home. The opponent to mark can be chosen by the player (e.g. the closest opponent), or by the unit captain which can ensure that all opponents are marked, following a preset algorithm as part of the locker-room agreement. The low-level agent behavior used when marking an opponent is specified in Appendix B (Section B.1.6).

When using ball-dependent positioning, the agent adjusts its location within its range based on the instantaneous position of the ball. For example, when the ball is on the same side of the field as the agent, the agent tries to move to a point on the line defined by its own goal and the ball. When the ball is on the other side of the field, the player adjusts its position back towards its own goal.

Ball-dependent positioning is an improvement over rigid roles in which agents stay in a fixed home position. However, by taking into account the positions of other agents as well as that of the ball, an even more informed positioning decision can be made. The idea of *strategic position by attraction and repulsion* (SPAR) is one of the novel contributions of the CMUnited-98 software. It was developed jointly in simulation and on the CMUnited-98 small-robot platform [Veloso *et al.* 99b].

When positioning itself using SPAR, the agent uses a multi-objective function with attraction and repulsion points subject to several constraints. To formalize this concept, we introduce the following variables:

- P - the desired position for the passive agent in anticipation of a passing need of its active teammate;
- n - the number of agents on each team;
- O_i - the current position of each opponent, $i = 1, \ldots, n$;
- T_i - the current position of each teammate, $i = 1, \ldots, (n-1)$;
- B - the current position of the active teammate and ball;
- G - the position of the opponent's goal.

SPAR extends similar approaches of using potential fields for highly dynamic, multi-agent domains [Latombe 91]. The probability of collaboration in the robotic soccer domain is directly related to how "open" a position is to allow for a successful pass. Thus, SPAR maximizes the distance from other robots and minimizes the distance to the ball and to the goal according to several forces, namely:

- *Repulsion* from opponents, i.e. maximize the distance to each opponent: $\forall i, \max dist(P, O_i)$
- *Repulsion* from teammates, i.e. maximize the distance to other passive teammates: $\forall i, \max dist(P, T_i)$
- *Attraction* to the active teammate and ball: $\min dist(P, B)$
- *Attraction* to the opponent's goal: $\min dist(P, G)$

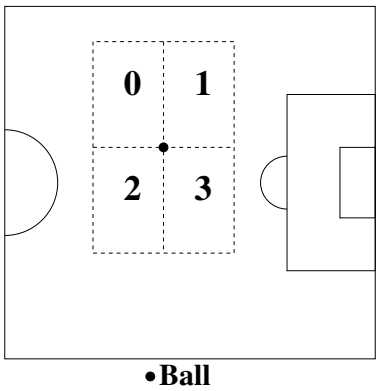


Figure 3.8: The four possible rectangles, each with one corner at the ball's location, considered for positioning by simulator agents when using SPAR.

This formulation is a multiple-objective function. To solve this optimization problem, we restate the problem as a single-objective function. As each term may have a different relevance (e.g. staying close to the goal may be more important than staying away from opponents), we want to apply a different weighting function to each term, namely f_{O_i}, f_{T_i}, f_B, and f_G, for opponents, teammates, the ball, and the goal, respectively. Our anticipation algorithm then maximizes a weighted single-objective function with respect to P:

$$\max(\sum_{i=1}^{n} f_{O_i}(dist(P, O_i)) + \sum_{i=1}^{n-1} f_{T_i}(dist(P, T_i)) - f_B(dist(P, B)) - f_G(dist(P, G)))$$

In our case, we use $f_{O_i} = f_{T_i} = x$, $f_B = 0$, and $f_G = x^2$. For example, the last term of the objective function above expands to $(dist(P, G))^2$.

One constraint in the simulator team relates to the position, or role, that the passive agent is playing relative to the position of the ball. The agent only considers locations that are within one of the four rectangles illustrated in Figure 3.5.2: the one closest to the position home of the position that the agent is currently playing. This constraint helps ensure that the player with the ball will have several different passing options in different parts of the field. In addition, players do not need to consider moving too far from their positions to support the ball.

Since this position-based constraint already encourages players to stay near the ball, we set the ball-attraction weighting function f_B to the constant function $y = 0$. In addition to this first constraint, the agents observe three additional constraints. In total, the constraints in the simulator team are:

- Stay in an area near one's home position;
- Stay within the field boundaries;
- Avoid being in an offsides position;
- Stay in a position in which it would be possible to receive a pass.

This last constraint is evaluated by checking that there are no opponents in a cone with vertex at the ball and extending to the point in consideration.

In our implementation, the maximum of the objective function is estimated by sampling its values over a fine-grained mesh of points that satisfy the above constraints.

Using this SPAR algorithm, agents are able to *anticipate* the collaborative needs of their teammates by positioning themselves in such a way that the player with the ball has several useful passing options.

Pre-Planned Set-Plays

The final implemented improvement facilitated by our flexible teamwork structure is the introduction of set-plays, or pre-defined, special-purpose plays. As a part of the locker-room agreement, the team can define multi-step, multi-agent plans to be executed at appropriate times. Particularly if there are certain situations that occur repeatedly, it makes sense for the team to devise plans for those situations.

In the robotic soccer domain, several situations occur repeatedly. For example, after every goal, there is a kickoff from the center spot; and when the ball goes out of bounds, there is a goal-kick, a corner-kick, or a kick-in. In each of these situations, the referee informs the team of the situation. Thus all the players know to execute the appropriate set-play. A particular referee's message is the trigger condition for each set-play. Associated with each set-play role is a set-play behavior indicating a location on the field as well as an action to execute when the ball arrives. The player in a given role might pass to the player filling another role, shoot at the goal, or kick the ball to some fixed location. The termination condition for each role is either the successful execution of the prescribed action or the passage of a specified amount of time from the beginning of the set-play.

The locker-room agreement specifies that the roles in the current team formation are mapped to the set-play roles in the way requiring the least movement of agents from their position homes. That is $F \mapsto R_{sp}$ is chosen to minimize

$$\sum_{spr \in R_{sp}} Dist(r, spr)$$

where $Dist(r, spr)$ is the distance from the home location of role r to the home location of its associated set-play role spr. This assignment of roles to set-play roles is part of each agent's internal behavior.

For example, Figure 3.9 illustrates a sample corner-kick set-play. The set-play designates five set-play roles, each with a specific location, which should be filled before the ball is put back into play. Based on the home positions of the current formation, each individual agent can determine the best mapping from positions to set-play locations, i.e. the mapping that requires the least total displacement of the 5 players. If there is no player filling one of the necessary formation roles, then there must be two players filling the same role, one of which must move to the vacant role. In the event that no agent chooses to do so, the set-play can proceed with any single set-play role unfilled. The only exception is that some player must fill the set-play role responsible for kicking the ball back into play. A special-purpose protocol is incorporated into the set-play behaviors to guarantee such a condition.

Once the set-play roles are filled, each player executes the action associated with its set-play role as an external behavior. As illustrated by the role starting the corner-kick in Figure 3.9, a player could choose among possible actions, perhaps based on the opponent positions at the time of execution. No individual player is guaranteed of participating in the play. For example, the uppermost set-play position is there just in case one of the other players misses a pass or shoots wide of the goal: no player will pass directly to it. Each player leaves its set-play role to resume its former role either after successfully kicking the ball, or after a pre-specified, role-specific amount of time.

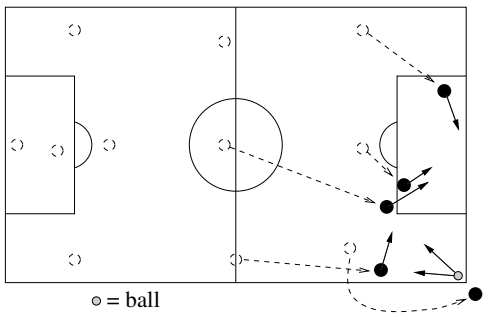


Figure 3.9: A sample corner-kick set-play. The dashed circles show the positions in the team's current formation and dashed arrows indicate the locations of the set-play roles—black circles—that they would fill. Solid arrows indicate the direction the ball is to be kicked as part of each set-play role.

The set-plays significantly improved CMUnited's performance. During the RoboCup-97 and RoboCup-98 simulator competitions, several goals were scored as a direct result of set-plays.

3.5.3 Communication Paradigm Implementation

In our teamwork structure, players are organized into team formations with each player filling a unique role. However players can switch among roles and the entire team can change formations. Both formations and roles are defined as part of the locker-room agreement, and each player is given a unique ID number. It is a significant challenge for players to remain coordinated, both by all believing that they are using the same formation and by filling all the roles in the formation. Since agents are all completely autonomous, such coordination is not guaranteed.

In PTS domains with limited communication (as opposed to no communication) possible during the dynamic, real-time periods, inter-agent communication can help the team stay coordinated. Communication protocols defined in the locker-room agreement combine with heard messages to trigger internal behaviors that alter the agent's internal state.

This section describes the CMUnited simulator team implementation of the communication paradigm presented in Section 3.4. All of the agent messages are of the format:

(CMUnited <Uniform-number> <Encoded-stamp> <Formation-number>
<Formation-set-time> <Position-number> <target> <Message-type>
[<Message-data>])

For example, player 8 might want to pass to player 6 but not know precisely where player 6 is at the moment. In this case, it could send the message `(CMUnited 8 312 1 0 7 ----> 6 Where are you?)`. "`CMUnited 8`" is the sender's team and number; "`312`" is the <Encoded-stamp>, in this case an agreed-upon linear combination of the current time, the formation number, and the sender's position number; "`1 0`" is the team formation player 8 is using followed by the time at which it started using it; "`7`" is player 8's current position; "`----> 6`" indicates that the message is for player 6; and "`Where are you?`" is a message type indicating that a particular response is requested: the recipient's coordinate location. In this case, there is no message data.

All teammates that hear such a message update their internal states to indicate that player 8 is playing position 7. However only player 6 sets its `response` and `response-flag` internal state variables. In this case, since the target is a single player, the `communicate-delay` flag remains at 0. Were the message targeted towards the whole team or to a subset of the team, then `communicate-delay` would equal:

- IF (my number > sender number)
  ((my number − sender number − 1) × 2) × `communicate-interval`
- ELSE (((sender number − my number − 1) × 2) + 1) × `communicate-interval`

where `communicate-interval` is the time between audible messages for a given agent (`hear_decay` = 2 simulator cycles in the case of the soccer server). Thus, assuming no further interference, player 8 would be able to hear responses from all targets.

Once player 6 is ready to respond, it might send back the message `(CMUnited 6 342 1 0 5 ----> all I'm at 4.1 -24.5)`. Notice that player 6 is using the same team formation but playing a different position from player 8: position 5. Since this message doesn't require a response (as indicated by the "`I'm at`" message type), the message is accessible to the whole team ("`----> all`"): all teammates who hear the message update their world states to reflect the message data. In this case, player 6 is at coordinate position (4.1, −24.5).

Were player 8 not to receive a response from player 6 before passing, it could still pass to its best estimate of player 6's location: should the message fail to get through, no disaster would result. Such is the nature of most communication in this domain. Should there be a situation which absolutely requires that a message get through, the sending agent could repeat the message periodically until hearing confirmation from the recipient that the message has arrived. However, such a technique consumes the single communication channel and should be used sparingly.

Notice that in the two example messages above, both players are using the same team-formation. However, such is not always the case. Even if they use common environmental cues to trigger formation changes, one player might miss the cue. In order to combat such a case, players update the team formation if a teammate is using a different formation that was set at a later time as detailed in Section 3.4. For example, if player 6's message had begun "`(CMUnited 6 342 3 50 ...`" indicating that it had been using team formation 3 since time 50, an internal behavior in player 8 would have changed its internal state to indicate the new team strategy.

Other examples of message types used in our implementation of simulated robotic soccer players include:

- Request/respond ball location
- Request/respond teammate location
- Inform pass destination
- Inform going to the ball
- Inform taking/leaving position

3.6 Results

Although the flexible teamwork structure and communication paradigm implementations are merged into a single robotic soccer system, we are able to isolate the effects of each contribution through controlled testing. This section presents empirical results demonstrating the effectiveness of both main contributions of this chapter: the flexible teamwork structure and the low-bandwidth communication paradigm.

3.6.1 Teamwork Structure Results

The flexible teamwork structure improves over a rigid structure by way of three characteristics: flexible positioning within roles, set-plays, and changeable formations. We tested the benefits of the first two characteristics by playing a team with flexible, changeable positions and set-plays against a "default team" with rigid positions and no set-plays. The behaviors of the players on the two teams are otherwise identical. The advantage of being able to change formations—the third characteristic—depends on the formation being used by the opponent. Therefore, we tested teams using each defined formation against each other.

Standard games in the soccer server system last 10 minutes. However, due to the large amount of noise, game results vary greatly. All reported results are cumulative over several games. Compiled statistics include the number of 10-minute games won, the total cumulative goals scored by each team, average goals per game, and the percentage of time that the ball was in each half of the field. The last statistic gives a rough estimate of the degree to which each team was able to control the ball.

Flexible Positions and Set-Plays

In order to test the flexible teamwork structure, we ran a team using ball-dependent flexible positions with set-plays against one using rigid positions and no set-plays. Both teams used a 4-4-2 formation. As shown in Table 3.2, the flexible team significantly outperformed the default team over the course of 38 games.

Further experimentation showed that both aspects of the flexible team contribute significantly to the team's success. Table 3.3 shows the results when a team using flexible positions but no set-plays plays against the default team and when a team using set-plays but rigid positions plays against the default team, again over the course of 38 games. Both characteristics provide a significant advantage over the default team, but they perform even better in combination.

(Game = 10 min.)	Flexible and Set-Plays	Default
Games won	34	1
Total goals	223	82
Avg. goals	5.87	2.16
Ball in own half	43.8%	56.2%

Table 3.2: Results when a flexible team plays against a rigid team. The flexible team won 34 out of 38 games with 3 ties.

Only Flexible Positions

(Game = 10 min.)	Flexible	Default
Games won	26	6
Total goals	157	87
Avg. goals	4.13	2.29
Ball in own half	44.1%	55.9%

Only Set-Plays

(Game = 10 min.)	Set-Plays	Default
Games won	28	5
Total goals	187	108
Avg. goals	4.92	2.84
Ball in own half	47.6%	52.4%

Table 3.3: Results when only using flexible positions and only using set-plays. Each individually works better than using neither.

Formations

In addition to the above tests, we tested the various formations against each other, as reported in Table 3.4. Each entry shows the goals scored for and against when a team using one formation played against a team using another formation over the course of 24 10-minute games. The right-most column collects the total goals scored for and against the team using that formation when playing against all the other teams. In all cases, the teams used flexible positions, but no set-plays.

formations	4-3-3	4-4-2	3-5-2	8-2-0	3-3-4	2-4-4	totals
4-3-3		68–60	68–54	24–28	59–64	70–65	289–271 (51.6%)
4-4-2	60–68		68–46	22–24	51–57	81–50	282–245 (53.5%)
3-5-2	54–68	46–68		13–32	61–72	75–73	249–313 (44.3%)
8-2-0	28–24	24–22	32–13		27–28	45–36	156–96 (61.9%)
3-3-4	64–59	57–51	72–61	28–27		87–69	308–267 (53.6%)
2-4-4	65–70	50–81	73–75	36–45	69–87		293–385 (43.2%)

Table 3.4: Comparison of the different formations. Entries in the table show the number of goals scored. Total (and percentage) cumulative goals scored against all formations appear in the right-most column.

The results show that the defensive formation (8-2-0) does the best in terms of the percentage of goals scored for versus those allowed against. However the total goals scored when using the defensive formation is quite low. On the other hand, the 3-3-4 formation performs well with a high goal total.

This study allowed us to devise an effective formation-switching strategy for the RoboCup competitions. For example, our RoboCup-97 team [Stone and Veloso 98a] used a 4-4-2 formation in general, switching to an 8-2-0 formation if winning near the end of the game, or a 3-3-4 formation if losing. This strategy, along with the flexible teamwork structure as a

whole, and the communication paradigm, helped us to perform well in the RoboCup-97 and RoboCup-98 tournaments (see Chapter 8).

3.6.2 Communication Paradigm Results

While contributing to the overall success of the CMUnited simulator team, our communication paradigm is also demonstrably effective in controlled experimentation. In this section, I report results reflecting the agents' robustness to active interference, their ability to handle messages that require responses from multiple team members, and their ability to maintain a coordinated team strategy.

Robustness to Interference

Relying on communication protocols involves the danger that an opponent could actively interfere by mimicking an agent's obsolete messages: since there is a single communication channel, opponents can hear and mimic messages intended for teammates. However, the <Encoded-stamp> field guards against such an attempt. As a test, we played a communicating team (team C) against a team that periodically repeats past opponent messages (team D). Team C set the <Encoded-stamp> field to <Uniform-number> $*$(`send-time` + 37). Thus teammates could determine `send-time` by inverting the same calculation (known to all through the locker-room agreement). Messages received more than a second after the `send-time` were disregarded (`message-lag-tolerance` = 1 sec). The one-second leeway accounts for the fact that teammates may have slightly different notions of the current global time.

In our experiment, agents from team D sent a total of 73 false messages over the course of a 5-minute game. Not knowing team C's locker-room agreement, they were unable to adjust the <Encoded-stamp> field appropriately. The number of team C agents hearing a false message ranged from 0 to 11, averaging 3.6. In all cases, each of the team C agents hearing the false message correctly ignored it. Only one message truly from a team C player was incorrectly ignored by team C players, due to the sending agent's internal clock temporarily diverging from the correct value by more than a second. Although it did not happen in the experiment, it is also theoretically possible that an agent from team D could mimic a message within a second of the time that it was originally sent, thus causing it to be indistinguishable from a valid message. However, in this case, the content of the message is presumably still appropriate and consequently unlikely to confuse team C.

Handling Multiple Responses

Next we tested our method of handling multiple simultaneous responses to a single message. Placing all 11 agents within hearing range, a single agent periodically sent a "where are you" message to the entire team and recorded the responses it received. In all cases, all 10 teammates heard the original message and responded. However, as shown in Table 3.5, the use of our method dramatically increased the number of responses that got through to the sending agent. When the team used `communicate-delay` as specified in Section 3.5, message responses were staggered over the course of about 2.5 seconds, allowing most of

the 10 responses to get through. When all agents responded at once (no delay), only one response (from a random teammate) was heard.

	Number of Responses			Response Time (sec)		
	Min	Max	Avg	Min	Max	Avg
No Delay	1	1	1.0	0.0	0.0	0.0
Delay	6	9	8.1	0.0	2.6	0.9

Table 3.5: The number of responses that get through to agents when responses are delayed and when they are not. When the team uses `communicate-delay` as specified in Section 3.5, an average of 7.1 more responses get through than when not using it. Average response time remains under one second. Both experiments were performed 50 times.

Team Coordination

Finally, we tested the team's ability to maintain coordinated team strategies when changing formations via communication. One player was given the power to toggle the team's formation between a defensive and an offensive formation. Announcing the change only once, the rest of team had to either react to the original message, or get the news from another teammate via other communications. As described in Section 3.5, the <Formation-number> and <Formation-set-time> fields are used for this purpose. We ran two different experiments, each consisting of 50 formation changes. In the first, a midfielder made the changes, thus making it possible for most teammates to hear the original message. In the second experiment, fewer players heard the original message since it was sent by the goaltender from the far end of the field. Even so, the team was able to change formations in an average time of 3.4 seconds. Results are summarized in Table 3.6.

	Entire Team Change Time (sec)				Heard From
Decision-Maker	Min	Max	Avg	Var	Decision-Maker
Goaltender	0.0	23.8	3.4	17.8	46.6%
Midfielder	0.0	7.9	1.3	2.8	80.6%

Table 3.6: The time it takes for the entire team to change team strategies when a single agent makes the decision. Even when the decision-making agent is at the edge of the field (goaltender) so that fewer than half of teammates can hear the single message indicating the switch, the team is completely coordinated after an average of 3.4 seconds.

3.7 Transfer to the Real Robots

As described in Chapter 2, the CMUnited-97 real robot platform differs from the soccer server platform in many ways, including agent perception and action capabilities. In addition, the robots do not have any inter-agent communication abilities. Nonetheless, the real robot soccer domain is a PTS domain and many components of the CMUnited simulator implementation described in Section 3.5 are directly transferable to the CMUnited-97 small-robot software. In particular, the teamwork structure transfers very easily and has been used

successfully [Veloso and Stone 98]. Since the robots do not use inter-agent communication, the communication paradigm is not appropriate for transfer.

The specific software components that transfered from the simulation to the real robot implementations include:

- The world model;
- Skill functionalities;
- Behavior modes;
- Team structure including the locker-room agreement

The objects in the world are defined according to the same type hierarchy depicted in Figure 3.4 and the same information is stored about each object. The one exception is that opponent orientation is not discernible by the small-robot vision system.

Since the movement capabilities and action command syntax of the robots differ greatly from those of the simulated agents, the low-level agent skill implementations are entirely different. However, the *functionalities* of the skills are similar. For example, like the simulated agents, the real robots also have ball-interception and goaltending skills. The implementation details of the robots' skills are provided in Appendix B.

With slightly less complex behaviors than the simulated agents, the robots need fewer behavior modes. However, they still choose one of several behavior modes as the top-level action decision. The default position-holding behavior occurs when the robot is in an *inactive* state. However, when the ball is nearby, the robot changes into an *active* state. In the active state, the robot moves towards the ball, attempting either to pass it to a teammate or to shoot it towards the goal based on an evaluation function that takes into account teammate and opponent positions. A robot that believes itself to be the intended receiver of a pass moves into the *auxiliary* state in which it tries to intercept a moving ball towards the goal. There is also a special *goaltend* mode for the goaltender. The CMUnited-97 decision function sets the robot that is closest to the ball into the active state; other robots filling a forward role (if any) into the auxiliary state; and all other robots (other than the goaltender) into the inactive state.

Most significantly within the context of this chapter, the team structure implementation described in Section 3.5.2 transfered directly to the real robot software. Changing nothing in the code except for the actual formation configurations (to accommodate teams of 5 rather than teams 11), the robots are instantly able to benefit from the advantages of the teamwork structure:

- Dynamic formations;
- Flexible positions; and
- Set-plays.

As in the simulator implementation, the formation defines a set of roles, or positions, with associated behaviors. It is specified within the locker-room agreement, a part of the team

member agent architecture described in Section 3.2. The robots are dynamically mapped into the positions. Each robot is equipped with the knowledge required to play any position in each of several formations.

Positions are defined as flexible regions within which the player attempts to move towards the ball. For example, a robot playing the "right-wing" (or "right forward") position remains on the right side of the field near the opponents' goal until the ball comes towards it. Positions are classified as defender, midfielder, or forward based on the locations of these regions. They are also given behavior specifications in terms of which other positions should be considered as potential pass-receivers.

In the CMUnited-97 small-robot team, only the ball-dependent flexible positioning is implemented. However, the subsequent CMUnited-98 software was used in conjunction with the simulator to develop the SPAR flexible positioning algorithm [Veloso *et al.* 99b].

At any given time each of the robots plays a particular position on the field. However, each robot has all of the knowledge necessary to play any position. Therefore the robots can—and do—switch positions on the fly. For example, robots A and B switch positions when robot A chases the ball into the region of robot B. Then robot A continues chasing the ball, and robot B moves to the position vacated by A.

The pre-defined positions known to all players are collected into formations, which are also commonly known. An example of a formation is the collection of positions consisting of the goaltender, one defender, one midfielder, and two attackers. Another possible formation consists of the goaltender, two defenders and two attackers. For illustration, see Figure 3.10.

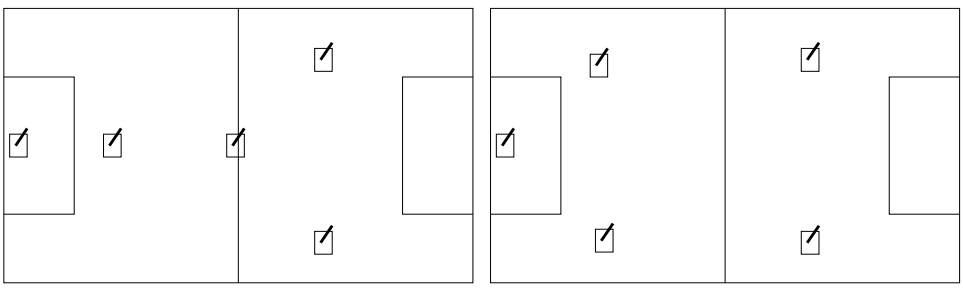


Figure 3.10: Two different defined formations. Notice that several of the positions are reused between the two formations.

The CMUnited-97 robots only used a single set-play: on a kickoff, one robot would pass the ball up the wing to another robot which would then shoot towards the goal. However, the concept of set-plays was originally developed in a previous real-robot soccer implementation [Achim *et al.* 96]. In that case, our 3-robot team was equipped with about 10 set-plays for several different restart situations.

3.8 Discussion and Related Work

This chapter has presented a team member agent architecture appropriate for PTS domains. While the implementation described here is in robotic soccer, it is easy to see how the architecture would apply to other sports, such as American football. In that case, the synchronization opportunities are more frequent, but formations can change during the course

of a game, roles are defined with some flexibility so that agents can adjust to opponent behaviors on the fly, and agents must communicate efficiently both between plays on a drive and during plays.

There are several other examples of non-sports-related PTS domains. Having successfully developed and deployed an autonomous spacecraft [Pell *et al.* 98], NASA is now interested in multi-spacecraft missions, or constellations [Stone 97]. Since spacecraft pointing constraints limit the communication both between the spacecraft and ground control, and among the spacecraft, the spacecraft must be able to act autonomously while still working towards the constellation's overall goal. Using interferometry missions—in which several spacecraft coordinate parts of a powerful imaging instrument to view distant objects—as an example, the locker-room agreement could be used to define several formations to be used for viewing objects that are at various distances or in different parts of the sky. Depending on the relative locations of these objects, the various spacecraft might interchange roles as they image different objects.

Search and rescue scenarios could also be formulated as PTS domains. If several robotic agents are trying to locate victims in a remote disaster sight, they may have to act quickly and autonomously. Nonetheless, before beginning the search, they could define several formations corresponding to different geographical areas of focusing their search. Within these formations, agents would need to be assigned flexible roles given that the precise situation may not be known or may change unexpectedly. The agents might also agree, as part of their locker-room agreement to switch formations either after a certain time or as a result of some limited communication, perhaps from a unit captain.

Other PTS domains that could be applications for the team member agent architecture are hospital/factory maintenance [Decker 96b] and battlefield combat [Tambe 97]. While in this thesis network routing is used to generalize the TPOT-RL algorithm (see Chapter 7), it could also be formulated as a PTS domain if the network nodes are permitted to freely use network bandwidth during periods of otherwise low usage. They could then exchange policies and feedback with regards to network performance.

The remainder of this section summarizes the previous work most closely related to the teamwork structure and communication paradigm as presented in this chapter.

3.8.1 Teamwork Structure

Two popular multi-agent teamwork structures, joint intentions [Cohen *et al.* 99] and shared plans [Grosz 96], consider a team to be a group of agents that negotiate and/or contract with each other in order to initiate a team plan. Both of these teamwork structures as well as STEAM [Tambe 97], another teamwork structure based on joint intentions and shared plans, include complex communication protocols for forming and disbanding a team in pursuit of a goal. The team forms dynamically and stays in close communication until the execution of the plan is completed. In contrast, the teamwork structure presented in this chapter supports a persistent team effort towards a common high-level goal in the face of limited communication.

The concept of the locker-room agreement facilitates coordination with little or no communication. Taking advantage of the property of PTS domains that the team and its long-

term goal are persistent, our teamwork structure eliminates the need for the overhead inherent in previous paradigms.

Although it has been claimed that pre-determined team actions are not flexible or robust to failure [Tambe 97], the locker-room agreement provides a mechanism for pre-defining team actions with enough flexibility to succeed. In particular, set-plays are pre-determined team actions that can be executed without the need to negotiate or use extensive inter-agent communication: the locker-room agreement provides enough flexibility that the agents are able to seamlessly assume the appropriate roles.

While I use the term "formation" to refer to the largest unit of the teamwork structure, soccer formations are not to be confused with military-type formations in which agents must stay in precise relative positions. Despite this dual usage of the term, I use it because formation is a standard term within the soccer domain [LaBlanc and Henshaw 94]. For an example of a multi-agent system designed for military formations, see [Balch and Arkin 95].

Castelfranchi [95] classifies different types of commitments in multi-agent environments. In this context, locker-room agreements can be viewed as C-commitments, or commitments by team members to do the appropriate thing at the right time, as opposed to S-commitments with which agents adopt each other's goals. In the context of [Conte *et al.* 99], the creation of a locker-room agreement is norm acceptance while its use is norm compliance. Within the framework presented in [Müller 99], the architecture is for interactive software and hardware multi-agents.

As mentioned in Section 3.2, the concept of behavior in the context of our team member agent architecture is consistent with that laid out by Mataric [Mataric 94a]. There, "behavior" is defined as "a control law with a particular goal, such as wall-following or collision avoidance." Behaviors can be nested at different levels with selection among lower-level behaviors consisting of a higher-level behavior. Similarly, internal and external behaviors in our system maintain team coordination goals, physical positioning goals, communication goals, and environmental information goals (such as knowledge of where the ball is). These behaviors are combined into top-level internal and external behaviors.

3.8.2 Communication Paradigm

Most inter-agent communication models assume reliable point-to-point message passing with negligible communication costs. In particular, KQML assumes point-to-point message passing, possibly with the aid of facilitator agents [Finin *et al.* 94]. Nonetheless, KQML performatives could be used for the content portions of our communication scheme. KQML does not address the problems raised by having a single, low-bandwidth communication channel.

When communication is reliable and the cost of communication relative to other actions is small, agents have the luxury of using reliable, multi-step negotiation protocols. For example, in Cohen's convoy example [Cohen *et al.* 99], the communication time required to form and maintain a convoy of vehicles is insignificant compared to the time it takes the convoy to drive to its destination. Similarly, message passing among distributed information agents is typically very quick compared to the searches and services that they are performing. Thus, it makes sense for agents to initiate and confirm their coalition while guaranteeing that they will inform each other if they have trouble fulfilling their part of the joint action.

With only a single team present, a situation similar to the one considered here is examined in [Maio and Rizzi 95]. In that case, like in the soccer server, messages sent are only heard by agents within a circular region of the sender. Communication is used for cooperation and for knowledge sharing. Like in the examples presented in the soccer domain, agents attempt to update each other regarding their own internal states when communicating. However, the exploration task considered there is much simpler than soccer, particularly in that there are no opponents using the same communication channel and in that the nature of the task allows for simpler, less urgent communication.

Although communication in the presence of hostile agents is well studied in military contexts from the standpoint of encryption, the problem considered here is not the same. While any encryption scheme could be used for the message content, the work presented here assumes that the adversaries cannot decode the message. Even so, they can disrupt communication by mimicking past messages textually: presumably past message have some meaning to the team that uttered them. Our method of message coding based on a globally accessible clock circumvents this latter problem.

Even when communication time is insignificant compared to action execution, such as in a helicopter fighting domain, it can be risky for agents to rely on communication. As pointed out in [Tambe 96a], if the teammate with whom an agent is communicating gets shot down, the agent could be incapacitated if it requires a response from the teammate. This work also considers the cost of communication in terms of risking opponent eavesdropping and the benefits of communication in terms of shifting roles among team members. However, the problems raised by a single communication channel and the possibility of active interference are not considered.

Another approach that recognizes the danger of basing behaviors upon multi-step communication protocols is ALLIANCE [Parker 94]. Since a primary goal of this work is fault-tolerance, only broadcast communications are used. Agents inform each other of what they are currently doing, but never ask for responses. In ALLIANCE, the team uses time-slice communication so that each agent periodically gets exclusive use of the single communication channel.

A possible application of our communication method is to robots using audio communication. This type of communication is inherently single-channel and low-bandwidth. An example of such a system is the Robot Entertainment Systems which uses a tonal language [Fujita and Kageyama 97]. Agents can communicate by emitting and recognizing a range of audible pitches. In such a system, the number of bits per message would have to be lowered, but the general techniques presented in this chapter still apply.

Another example of such a communication environment is arbitrarily expandable systems. If agents aren't aware of what other agents exist in the environment, then all agents must use a single universally-known communication channel, at least in order to initiate communication.