Flexible Coordination of Multiagent Team Behavior Using HTN Planning

Oliver Obst and Joschka Boedecker

Universität Koblenz-Landau, AI Research Group, 56070 Koblenz, Germany
{fruit, jboedeck}@uni-koblenz.de

Abstract. The domain of robotic soccer is known as a highly dynamic and non-deterministic environment for multiagent research. We introduce an approach using Hierarchical Task Network planning in each of the agents for high-level coordination and description of team strategies. Our approach facilitates the maintenance of expert knowledge specified as team strategies separated from the agent implementation. By combining high level plans with reactive basic operators, agents can pursue a grand strategy while staying reactive to changes in the environment. Our results show that the use of a planner in a multiagent system is both possible and useful despite the constraints in dynamic environments.

1 Introduction

Coordination among different agents in a multiagent system (MAS) is considered as one of the most import challenges in order to achieve a common goal. This task becomes increasingly difficult in highly dynamic, partially observable, and adversarial environments such as robotic soccer. Noisy sensor data and imperfect actuators further add to this problem.

For our work, we have in mind a multiagent system where the user specifies not only a general task for the system, but also the way specific situations are handled. On the other hand, the user can expect a certain set of available primitives that handle simple tasks on their own. The idea is that a designer can change the team behavior for specific situations. With the widely used reactive approaches, changes are complicated because of interdependencies so that simple changes can have impact on more than one situation. Another observation we made with previous approaches was the difficulty to specify strategies for reactive multiagent teams. Even though reactivity is a key quality in soccer, long term strategies seem to be as useful. Classical planning approaches are a solution to compute actions that lead towards given goals, but are in general regarded as not applicable to highly dynamic multiagent scenarios.

In this work, we suggest to use Hierarchical Task Network (HTN) planners in each of the agents in order to achieve coordinated team behavior, while at the same time our agents should always follow the strategy as suggested by the human expert. The expert knowledge should be separated from the rest of the agent code, in a way that it can be easily specified and changed. While pursuing the given strategy, the agents should keep as much of their reactiveness as possible. Different levels of detail in the description of strategies facilitate the generation of useful information for debugging or synchronization.

HTN planning is based upon work presented in [15]. It makes use of domain knowledge to speed up the planning and is able to solve classical planning problems orders of magnitude more quickly than classical planners if provided with a good set of HTNs. In our work, we followed the formalization of soccer domain knowledge in [5] but adapted it for the use in HTN planning.

Furthermore, we show how it is possible to use an HTN planner in the domain of robotic soccer, even though it is very different from environments used in classical planning with deterministic operators and a single planning agent being the only reason for changes in the world. For our approach, we have chosen a team of agents in the RoboCup Soccer Simulation League 3D [8].

We start by reviewing relevant related work, and then describe our approach in detail, outlining problems that arose while trying to adapt the planner to be used in the MAS, and the solutions to overcome them. We present and discuss the results of our first tests and give directions for future work.

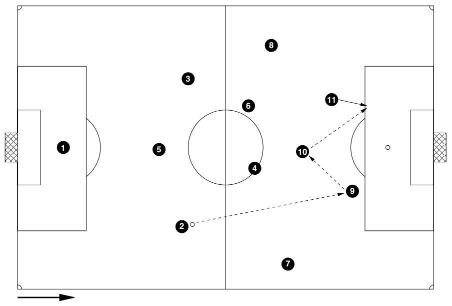
2 Related Work

Several approaches that use a planning component in a MAS can be found in the literature. In [4], the authors describe a formalism to integrate the HTN planning system SHOP [13] with the IMPACT [17] multiagent environment (A-SHOP). While the environment of this work clearly is a multiagent system, the planning is carried out centralized by a single agent. This is a contrast to our approach, which uses a planner in each of the agents to coordinate their actions.

Bowling *et al.* [2] presents a strategy system that makes use of *plays* to coordinate team behavior of robots in the RoboCup Small Size League. Multiple plays are managed in a *playbook* which is responsible to choose appropriate plays, and evaluate them for adaption purposes. Effects of individual plays are not specified due to the difficulties in predicting the outcome of operators in the dynamic environment. This is in contrast to our approach, as we use *desired effects* of the operators in our plans as described in section 3. The planning component in [2] is also centralized.

In [9], the use of *coordination graphs* [7] to coordinate a team of agents in dynamic environments without explicit communication is proposed. The coordination graphs are applied to the continuous domain of robotic soccer where a discretization of the state is achieved by assigning roles to the different agents, similar to the approach in [16]. The coordination graphs are then built on the derived set of roles.

In [11, 12], the behavior of agents in a Multiagent System is specified using UML statecharts. Agents are designed in a top-down manner with a layered architecture. At the highest level global patterns of behavior are specified in an abstract way. Explicit specification of cooperation and multiagent behaviors can be realized.



```
method diagram_4
pre [ is_current_formation(F),
      #formation(F,A,back,_,Side),
      #formation(F,B,forward,_,Side),
      #formation(F,C,_,attacking,center),
      #formation(F,D,forward,_,OtherSide),
      #different(Side,OtherSide),
      #different(Side,center)
    ]
subtasks [pass(A,B),
          pass(B,C),leading_pass(C,D)].
```

Fig. 1. Building up play with Diagram #4 from [10] (left) and its notation as HTN method for our planner (right). Preconditions prefixed with '#' denote function calls.

3 A Planner for Soccer Agents

The main focus of our work was to achieve high-level coordination for a team of several agents in a dynamic environment. All planning should be done in a distributed fashion. The system should allow for easy specification of team plans, and automatically generate individual actions for the agents during execution.

In [5], we present an approach to model soccer knowledge, as it can be found in soccer theory books. The focus of this work was the representation of diagrams used in [10] to describe moves to be used in specific phases of the match. We derive an ontology from [10] which breaks down the top level task *play soccer* into subtasks by means of aggregation and specialization. A subset of this ontology can be found in Tab. 1. The hierarchy of the tasks and subtasks facilitates the collection of useful debugging output during development.

In Hierarchical Task Network (HTN) planning, the objective is to perform tasks. Tasks can be complex or primitive. HTN planners use *methods* to expand complex tasks into subtasks, until the tasks are primitive. Primitive tasks can be performed directly by using planning operators. Changes to basic HTN planning algorithms are necessary in order to use it for the soccer domain, because here the outcome of operators is not deterministic.

Because it is impossible to foresee how the world will look like after a few actions, our planner generates what is called *plan stub* in [1], a task network with a primitive task as the first task. As soon as a plan stub has been found, an agent can start executing its task. The algorithm in Fig. 2 expands a list of

Table 1. Some complex tasks with subtasks ('|': alternatives; ',': sequences)

play_soccer	**offensive_phase** | defensive_phase | *nothing*
offensive_phase	upon_ball_possession, **build_up_play**, final_touch, shooting
build_up_play	**build_up_play_long_pass** | build_up_play_diagonal_pass
build_up_play_long_pass	diagram_3 | **diagram_4** | ...
diagram_4	pass(A,B), **pass(B,C)**, leading_pass(C,D)
pass(B,C)	do_pass(B) | do_receive_pass | do_positioning

```
Function: plan($s_{now}, < t_1, ..., t_k >, O, M$)
Returns: $(w, s)$, with $w$ an ordered set of tasks, $s$ a state; or failure
---------------------------------------------------------------------------
if $k = 0$ then return $(\emptyset, s_{now})$  // i.e., the empty plan
if $t_1$ is a pending primitive task then
    active $\leftarrow \{(a, \sigma) | a$ is a ground instance of an operator in O,
                   $\sigma$ is a substitution such that $a$ is relevant for $\sigma(t_1)$,
                   and $a$ is applicable to $s_{now}\}$;
    if active $= \emptyset$ then return failure;
    nondeterministically choose any $(a, \sigma) \in$ active;
    return $(\sigma(< t_1, ..., t_k >), \gamma(s_{now}, a))$;
else if $t_1$ is a pending complex task then
    active $\leftarrow \{m | m$ is a ground instance of a method in $M$,
                 $\sigma$ is a substitution such that $m$ is relevant for $\sigma(t_1)$,
                 and $m$ is applicable to $s_{now}\}$;
    if active $= \emptyset$ then return failure;
    nondeterministically choose any $(m, \sigma) \in$ active;
    $w \leftarrow$ subtasks$(m).\sigma(< t_1, ..., t_k >)$;
    set all tasks in front of $t_1$ to pending, set $t_1$ to expanded;
    return plan($s_{now}, w, O, M$);
else
    // $t_1$ is an already executed expanded task and can be removed
    return plan($s_{now}, < t_2, ..., t_k >, O, M$);
```

Fig. 2. Creating an initial plan stub (Notation according to [6])

tasks to a plan stub. At the same time, the desired successor state is computed and returned. The second algorithm (see Fig. 3) removes executed tasks from the plan and uses the first algorithm then to create an updated plan stub.

To handle non-determinism, we treat a plan as a stack. Tasks on this stack are marked as either *pending* or as *expanded*. Pending tasks are either about to be executed, if they are primitive, or waiting to be further expanded, if they are complex. Tasks marked as expanded are complex tasks which already have been expanded into subtasks. If a subtask of a complex task fails, all the remaining subtasks of that complex task are removed from the stack and it is checked if the complex task can be tried again. If a task was finished successfully, it is simply removed from the stack.

Our plan operators realizing primitive tasks describe only the *desired effects* of an action. Using desired effects of an action we can check if the action was executed successfully. Simultaneously executed actions which are not part of the desired effects of the operator are simply ignored in the description.

To give an example, we assume that the planner is about to plan for a situation like in Fig. 1 and the complex task `play_soccer` has already been partially expanded as shown in Fig. 4 (left). All the pending tasks in Fig. 4 are still complex tasks. At this level of expansion, the plan still represents a team plan, as seen from a global perspective. The team task `pass(2,9)` will expand to `do_pass(9)` for agent #2, agent #9 has to do a `do_receive_pass` for the same team task. The other agents position themselves relatively to the current ball

```
Function: step($s_{expected}, s_{now}, < t_1, ..., t_k >, O, M$)
Returns: $(w, s)$, with $w$ a set of ordered tasks, $s$ a state; or failure
if $k = 0$ then return $(\emptyset, s_{now})$  // i.e., the empty plan
if $t_1$ is a pending task then
    if $s_{expected}$ is valid in $s_{now}$ then
        $i \leftarrow$ the position of the first non-primitive task in the list;
        return plan($s_{now}, < t_i, ..., t_k >, O, M$);
    else
        // $t_1$ was unsuccessful; remove all pending children of our parent task
        return step($s_{expected}, s_{now}, < t_2, ..., t_k >, O, M$);
else
    // $t_1$ is an unsuccessfully terminated expanded task, try to re-apply it
    active $\leftarrow \{m | m$ is a ground instance of a method in $M$,
                $\sigma$ is a substitution such that $m$ is relevant for $\sigma(t_1)$,
                and $m$ is applicable to $s_{now}\}$;
    if active $= \emptyset$ then
        // $t_1$ cannot be re-applied, remove it from the list and recurse
        return step($s_{expected}, s_{now}, < t_2, ..., t_k >, O, M$);
    else
        nondeterministically choose any $(m, \sigma) \in$ active;
        $w \leftarrow$ subtasks$(m).\sigma(< t_1, ..., t_k >)$;
        set all tasks in front of $t_1$ to pending, set $t_1$ to expanded;
        return plan($s_{now}, w, O, M$);
```

Fig. 3. Remove the top primitive tasks and create a new plan stub

```
pending-pass(2,9)
pending-pass(9,10)
pending-leading-pass(10,11)
expanded-diagram-4
expanded-build_up_long_pass
expanded-build_up_play
pending-final_touch
pending-shooting
expanded-offensive_phase
expanded-play_soccer
```

```
method pass(A,B)
pre [my_number(A)]
subtasks [do_pass(B) with pass(we,A,B),
          do_positioning].

method pass(A,B)
pre [my_number(B)]
subtasks [do_receive_pass with pass(we,A,B)].

method pass(A,B)
pre [my_number(C),#\=(A,C),#\=(B,C)]
subtasks [do_positioning with pass(we,A,B)].
```

Fig. 4. Plan stack during planning (left) and different methods to reduce the team task `pass(A,B)` to agent tasks (right)

position with `do_positioning` at the same time. The desired effect of `pass(2,9)` is the same for all the agents, even if the derived primitive task is different depending on the role of the agent. To express that an agent should execute the `do_positioning` behavior while taking the effect of a simultaneous pass between two teammates into account, we are using terms like `do_positioning with pass(we,2,9)` in our planner. Figure 4 (right) shows methods reducing the team task `pass(A,B)` to different primitive player tasks.

```
pending-(do_receive_pass with          pending-(do_positioning with
        pass(we, 2, 9)),                       pass(we, 2, 9)),
expanded-pass(2, 9),                   expanded-pass(2, 9),
pending-pass(9, 10),                   pending-pass(9, 10),
pending-leading_pass(10, 11),          pending-leading_pass(10, 11),
expanded-diagram-4,                    expanded-diagram-4,
...                                    ...
```

Fig. 5. Step 1: Plan Stubs for player 11 and player 9 (see also Fig. 1)

```
pending-(do_pass(10) with              pending-(do_positioning with
        pass(we, 9, 10)),                      pass(we, 9, 10)),
pending-do_positioning,                expanded-pass(9, 10),
expanded-pass(9, 10),                  pending-leading_pass(10, 11),
pending-leading_pass(10, 11),          expanded-diagram_4,
expanded-diagram_4,                    ...
...
```

Fig. 6. Step 2: Plan Stubs for player 11 and player 9

In different agents, the applicable methods for the top team task `pass(2,9)` lead to different plan stubs. This is an important difference to the work presented in [1]. The plan stubs created as first step for agent 9 and agent 11 are shown in Fig. 5. When a plan stub is found, the top primitive tasks are passed to the `C++` module of our agent and executed. The agent has to execute all pending primitive tasks until the next step in the plan starts. If there are pending primitive tasks after one step is finished, these agent tasks are simply removed from the plan stack and the next team task can be expanded. Figure 6 shows the plan stub for the second step from the diagram in Fig. 1. For player 11, the expansion leads to a plan stub with two primitive tasks in a plan step while for player 9 there is only one task to be executed. Each step in plans for our team stops or starts with an agent being in ball possession. If any of the agents on the field is in ball possession, we can check for the desired effect of the previous action.

4 Results and Discussion

For our approach of generating coordinated actions in a team we implemented an HTN planner in Prolog which supports interleaving of planning and acting. Our planner supports team actions by explicitly taking the effects of operators simultaneously used by teammates into account. The planner ensures that the agents follow the strategy specified by the user of the system by generating individual actions for each of the agents that are in accordance with it. The *lazy evaluation* in the expansion of subtasks which generates plan stubs rather than a full plan, makes the planning process very fast and enables the agents to stay reactive to unexpected changes in the environment. The reactiveness could, however, be increased by adding a situation evaluation mechanism that is used prior to invoking the planner. This would improve the ability to exploit sudden, short-lived opportunities during the game.

Creating strong teams is possible with many approaches. We strongly believe that our approach leads to a modular behavior design and facilitates rapid specification of team behavior for *users* of our agent architecture. Our plans can describe plays as introduced in [2], which have shown to be useful for synchronization in a team. Our approach supports different levels of abstraction in plans. That means there are different levels of detail available to describe what our team and each single agent is actually doing. Additionally, the planner can find alternative ways to achieve tasks. The approach in [2] was used for Small Size League; but for larger teams, more opportunities are possible for which an approach using fixed teammates seems to restrictive. On the other hand, the approach in [2] supports adaptation by changing weights for the selection of successful plays. In our approach, the corresponding functionality could be achieved by changing the order in which HTN methods are used to reduce tasks. At this point in time, our approach does not support this yet. As soon as we do have an adaptive component in our approach, it makes sense to compare results of our team with and without adaptation.

Although more detailed evaluations have to be carried out, the first tests using the planner seem very promising and indicate that our approach provides a flexible, easily extendable method for coordinating a team of agents in dynamic domains like the RoboCup 3D Simulation League.

5 Conclusion and Future Work

We presented a novel approach that uses an HTN planning component to coordinate the behavior of multiple agents in a dynamic MAS. We formalized expert domain knowledge and used it in the planning methods to subdivide the given tasks. The hierarchical structure of the plans speeds up the planning and also helps to generate useful debugging output for development. Furthermore, the system is easily extendable as the planning logic and the domain knowledge are separated.

In order to use the system in RoboCup competitions, we plan to integrate a lot more subdivision strategies for the different tasks as described in the diagrams in [10]. A desirable enhancement to our work would be the integration of an adaption mechanism. Monitoring the success of different strategies against a certain opponent, and using this information in the choice of several applicable action possibilities, as e.g. outlined in [2], should be explored. The introduction of *durative actions* into the planner (see for instance [3]) would give a more fine grained control over the parallelism in the multiagent plans. Finally, we want to restrict the sensors of the agents to receive only partial information about the current world state, and address the issues that result for the distributed planning process.

References

1. Thorsten Belker, Martin Hammel, and Joachim Hertzberg. Learning to optimize mobile robot navigation based on HTN plans. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4136–4141, 2003.
2. Michael Bowling, Brett Browning, and Manuela Veloso. Plays as team plans for coordination and adaptation. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, Vancouver, June 2004.
3. Alex M. Coddington, Maria Fox, and Derek Long. Handling durative actions in classical planning frameworks. In John Levine, editor, *Proceedings of the 20th Workshop of the UK Planning and Scheduling SIG*, pages 44–58, 2001.
4. Jürgen Dix, Héctor Muñoz-Avila, and Dana Nau. IMPACTing SHOP: Planning in a Multi-Agent Environment. In Fariba Sadri and Ken Satoh, editors, *Proceedings of CLIMA 2000, Workshop at CL 2000*, pages 30–42. Imperial College, 2000.
5. Frank Dylla, Alexander Ferrein, Gerhard Lakemeyer, Jan Murray, Oliver Obst, Thomas Röfer, Frieder Stolzenburg, Ubbo Visser, and Thomas Wagner. Towards a league-independent qualitative soccer theory for RoboCup. In Daniele Nardi, Martin Riedmiller, Claude Sammut, and José Santos-Victor, editors, *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 611–618, Berlin, Heidelberg, New York, 2005. Springer.
6. Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning Theory and Practice.* Morgan Kaufmann, San Francisco, CA, USA, 2004.
7. Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored MDPs. In *In 14th Neural Information Processing Systems (NIPS-14)*, 2001.
8. Marco Kögler and Oliver Obst. Simulation league: The next generation. In Polani et al. [14], pages 458–469.
9. Jelle R. Kok, Matthijs T. J. Spaan, and Nikos Vlassis. Multi-robot decision making using coordination graphs. In A.T. de Almeida and U. Nunes, editors, *Proceedings of the 11th International Conference on Advanced Robotics, ICAR'03*, pages 1124–1129, Coimbra, Portugal, June 2003.
10. Massimo Lucchesi. *Coaching the 3-4-1-2 and 4-2-3-1.* Reedswain Publishing, 2001.
11. Jan Murray. Specifying agent behaviors with UML statecharts and StatEdit. In Polani et al. [14], pages 145–156.
12. Jan Murray, Oliver Obst, and Frieder Stolzenburg. RoboLog Koblenz 2001. In Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Artificial Intelligence*, pages 526–530. Springer, Berlin, Heidelberg, New York, 2002. Team description.
13. Dana S. Nau, Yue Cao, Amnon Lotem, and Héctor Muñoz-Avila. Shop: Simple hierarchical ordered planner. In *Proceedings of IJCAI-99*, pages 968–975, 1999.
14. Daniel Polani, Brett Browning, Andrea Bonarini, and Kazuo Yoshida, editors. volume 3020 of *Lecture Notes in Artificial Intelligence*. Springer, 2004.
15. Earl D. Sacerdoti. A structure for plans and behavior. American Elsevier, 1977.
16. Peter Stone and Manuela Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 1999.
17. V. S. Subrahmanian, Piero Bonatti, Jürgen Dix, Thomas Eiter, Sarit Kraus, Fatma Ozcan, and Robert Ross. *Heterogeneous Agent Systems.* MIT Press, 2000.