Predicting agent behavior by declarative agent modelling: The Dirty Dozen 2002

Timo Steffens, Marco Diedrich, Leonhard Hennig, Philipp Huegelmeyer, and Andreas G. Nie (robocup@cl-ki.uni-osnabrueck.de)

Institute of Cognitive Science, University of Osnabrueck, Germany

Abstract. Coordination of agents within a team and using knowledge about the behavior of adversary agents is an important and challenging research topic in multi-agent systems. This paper proposes a method which allows agents to consider the future actions of their teammates and opponents, thereby improving coordination among teammates and adaptation to the opponent, respectively. This method works in both centralized and decentralized systems and retains player autonomy even if the coordination is directed by a centralized agent.

1 Introduction

In multi-agent systems with agents which share common or have adversary goals, coordination between cooperating agents and reacting to competing agents is crucial [2]. This claim about coordination is especially true, if the common goal cannot be achieved by a single agent alone, but only by a team. Often this coordination is achieved by letting a designated agent direct the others. Yet, this requires a lot of communication. If the bandwidth is low or communication is expensive or unreliable, other techniques have to be employed. Previous work mostly focussed on monitoring agents and detecting failures in team coordination [3]. Work on reacting to competing agents includes modelling the opponent and taking counter-measures [2]. This paper proposes a method for coordination between teammates and for benefitting from opponent modelling information. Both are done by declarative behavior specification and reacting to agent actions. The modelling takes place centralized in a designated coach-agent and decentralized in all agents. RoboCup is an example for domains, in which teams of agents compete against each other. The Dirty Dozen team is an implementation of declarative behavior specification. The remainder of this paper is organized as follows: Section 2 describes how coordination is achieved by modelling-structures within the team. Section 3 outlines how the same method can be used to benefit from opponent modelling information provided by an online coach. In section 4 it is outlined how players retain autonomy against other agents, and finally section 5 concludes.

2 Teammate modelling

When working in teams, executing a task by more than one agent is suboptimal if one agent is enough. On the other hand not doing an important task at all must be avoided. To coordinate teammates it is helpful to model teammates in order to predict the other's actions. These predictions can be used in the decision process. The behavior of the Dirty Dozen is specified in an extension of the standard coach language (Clang) [1], called SFL [4]. It is a production system with rules consisting of conditions and actions, stating that the given action will be executed if the condition is true. In SFL it is possible to include other agent's actions as condition. The idea proposed in this paper is to facilitate coordination like closing a defense line if one of the defenders has to break out of formation. In this scenario, the defenders will have a rule that becomes true if e.g. one defender goes to the ball, and states that the defense line must be closed. That leaves the problem of finding out what the teammates' action is. In complex domains like RoboCup observations are ambiguous. For example a positioning obervation might be consistent with the hypothesis that the agent goes to the ball or marks the ballowner. To overcome this problem, several methods are described in the following.

Player communication: Being the most prominent method for coordination, communication appears now as a special case of inferring the other's action. An agent can broadcast its future action, so that the modelling agents can fill the action-condition slot. Yet, in RoboCup this costs at least one cycle and only one agent can talk per cycle.

Continual observation: By continually observing an agent, its inferred actions become less ambiguous. All possible actions are stored in a set. After each observation actions which are not consistent with it are removed. But in dynamic and fast-changing domains the observed agent is likely to change its plans and actions, so that continual obervation has to trade-off ambiguity reduction towards likeliness of changing plans by choosing a time intervall. Another disadvantage is that the observation time is lost for taking counter actions.

Modelbased action inferring: All players in the Dirty Dozen team share the same behavior specification. So, given a player's belief it is possible to predict its actions. Tambe proposed to model other agents with the agent's own architecture [2]. This is similar to what the Dirty Dozen players do. They evaluate the production system from the modelled player's perspective and infer its next action. This action can be filled in the action-condition slot of its own rules. The advantage to observating actions is that the agent does not need to wait until the action was executed, but can act simultaneously with the modelled agent. In order to avoid recursive modelling and to save computation time, the team is organized in so-called modelling trees. Each agent is assigned maximally one other player to model, but may be modelled by several other agents. This way several trees exist that correspond to subteams like defenders, midfielders and forwards. It is also possible to create only one tree, where subtrees correspond to the afore-mentioned subteams but are linked to each other by e.g. a midfielder modelling a forward.

By these tree-structures a team- or subteam captain naturally arises at the root of the tree. The captain's actions influence the other agent's decisions. For example, the movement of the defenders' captain causes the whole defense line to move given that each defender is involved in the modelling tree. For the defense line it is no longer necessary to observe all other defenders in order to stay in line. Instead, each agent only observes one other agent. Modelling one agent is very cheap in terms of computation time due to model-sharing proposed by Tambe [2]: Most of the rules in the production system are independent of the evaluating player. Since the player has to evaluate the rules for its own decision process anyways, it can also use most of the evaluations unchanged for modelling the other agent.

A problem that remains is transitive modelling. Since the actions of an agent depend on the actions of all agents above it, the agent might end up with more than one agent to model. In the worst case one agent that is several layers above the modelling player might be spatially too far away in order to model it from its perspective. For this reason we focus on modelling-trees that have only one layer under the root node, in order to coordinate subteams like forwards, midfielders and forwards. Future work will be spent on combining communication and the afore-mentioned action observation in order to handle multi-layer trees.

Of course it is crucial that the modelling agent has similar information about the world like the modelled agent. For now this is no problem, since they are spatially close to each other and share information about the same regions.

3 Modelling the opponent

Opponent modelling is done by an online coach for the Dirty Dozen team. The coach can send advice or information about the opponent's behavior to the players. Info-messages fit smoothly into the afore-mentioned modelling methods, so that knowledge about the opponent's behavior can be used. Info-messages are rules with a condition- and action part and describe the opponent's tendency to execute an action if the condition is satisfied. So just like when evaluating the production system, players can evaluate the info-messages from the coach and predict some actions of the opponent. These can then be filled into action-condition slots in the production system.

For example: Let a player have the following rule in its production system: "If an opponent is going to do a long pass along the wings, move the defense line to that wing." The online coach now does not have to come up with counter actions for long passes along the wings, but can just acquire data about the opponent and send its results as info-messages. So, if it finds situations in which such passes are executed, it can send these to the players who can benefit from this information.

This procedure facilitates interoperability between coaches and teams that were designed by different research groups, because the coach can focus on modelling the opponent without having to come up with appropriate counter actions.

These counter actions depend heavily on the abilities of the team, so that its designers know better, if it is for example stronger in tackling or marking.

This is to our knowledge the first attempt for an architecture that can handle the full expressivity of CLang opponent modelling information (although ChaMeleons use it to send information about heterogenous player types [5]).

4 Player autonomy

Agents that are dependent on other agent's actions or advices must make sure that they will not be negatively affected by failures of their captain or advisor. In the coach competition 2001 a situation arose in which an online coach sent inverted player positions as advise, so that the team's forwards moved to their own goals and its defenders to the opponent goal. This shows that players have to retain a great amount of autonomy in order to overcome failures like that.

A large part of suboptimal behavior can be detected by humans by common sense. A very simple method to implement this into software agents is to express parts of this common sense as DO-NOT-rules. These are part of CLang and express the fact that an action should not be executed in certain situations. CLang actions can be expressed in general terms, so that rules like "Do not pass to the goalie" or "Do not pass in front of your penalty area" can be expressed. In the Dirty Dozen several of these rules form a special constraint rulebase against which each action that an agent can execute is tested. These constraints are ranked and the action that violates the fewest constraints will be executed.

5 Conclusion

It was shown that observing and reasoning about the actions of other agents can be used for achieving coordination among teammates and for incorporating opponent modelling information. Including other agents' actions into one's own decision process is facilitated by specifying the behavior explicitly in declarative production systems. Reasoning about one's own behavior by means of constraints that model common sense increases player autonomy.

References

1. Noda et al.: Soccerserver Manual v7. 2001.
2. Milind Tambe, Paul S. Rosenbloom: Architectures for Agents that Track Other Agents in Multi-Agent Worlds. in Proceedings on the IJCAI Workshop on Intelligent Agents II : Agent Theories, Architectures, and Languages. 1996
3. Gal Kaminka, D. V. Pynadath, Milind Tambe: Monitorin Deployed Agent Teams. In Proceedings of the Fifth International Conference on Autonomous Agents (Agents-2001). Montreal, 2001.
4. Sean Buttinger et al.: The Dirty Dozen Team and Coach Description. In RoboCup-2001: Robot Soccer World Cup V. Springer, Berlin, 2002.
5. Patrick Riley et al.: ChaMeleons and OWL Simulator Team and Coach. In RoboCup-2001: Robot Soccer World Cup V. Springer, Berlin, 2002.