The Dirty Dozen Team and Coach Description

Sean Buttinger, Marco Diedrich, Leo Hennig, Angelika Hoenemann, Philipp Huegelmeyer, Andreas Nie, Andres Pegam, Collin Rogowski, Claus Rollinger, Timo Steffens, and Wilfried Teiken

Institute of Cognitive Science, University of Osnabrueck, Germany

Abstract. The Dirty Dozen team consists of the player agents and an online coach. The players' low-level-skills and world-model are based on the publicly available code of the CMU-99 team [1]. The team behavior is specified using a new strategy formalization language (SFL), an extension of the standard coach language. This allows to modify the behavior easily and makes the whole team coachable, even for online coaches developed by different teams. We introduce the main concepts of SFL and one possible system with modules that interprete and run the SFL specifications. We also give an outline of the online coach.

1 Introduction

Specifying the behavior of agents in multi-agent-systems is often possible only for domain experts [4]. This paper proposes a new method for specifying a team's strategy in a way that makes adjustments by a human designer or an online coach easy. The Strategy Formalization Language (SFL) is an extension of the standard coach language (Clang) [2] which enables coaches and players of different teams to communicate with each other. SFL extends Clang in terms of expressibility, low-level- and high-level-concepts. The Dirty Dozen team is implemented as an SFL-System which is described below.

2 Strategy Formalization Language System (SFLS)

In order to easily handle coach-messages, SFL completely subsumes Clang and thus retains its syntactical structure. There are five main message types of which we will focus on two in this paper, namely info- and advice- messages. These are syntactically the same, just the semantics differ in that info-messages contain knowledge about the behavior of one of the teams, and advice-messages contain directives about how to behave in certain situations. The syntax of these messages is:

$(\{info \| advice\}\ TOKEN_1\ TOKEN_2 \ldots TOKEN_n)$

where TOKENs are

$(T\ CONDITION\ DIRECTIVE_1\ DIRECTIVE_2 \ldots DIRECTIVE_n)$

T denotes the time-to-live or timespan in which the token will be valid. CONDITION denotes a situation and the DIRECTIVEs denote directives for a set of players. Basically, these tokens are production rules. In the following some of the SFL-extensions to CONDITION- and DIRECTIVE-tokens are described.

2.1 Language Concepts

Clang is not sufficient to implement the behavior of a team. Its concepts are too general, so SFL contains several concepts that are important for implementing teams. It also makes many Clang-advices that may be uttered by an online-coach more elegant and concise.

Due to the extendibility of Clang, SFL was created by adding new concept primitives with clear semantics. First of all, it introduces abstractions of the constant uniform-numbers used in Clang to denote players: variables and symbols for situation-specific player denotation (e.g. ClosestPlayerToBall). These extend the practical expressibility of Clang, because it was not possible to concisely express "mark the ballowner". One would have to list all possible ballowners. In SFL this becomes "If opponent X has the ball, mark X."

A lot more complicated in Clang are advices that refer to different players in certain situations, e.g. the closest player to the ball. Since there is no concept for distances in Clang, the advice would have to simulate distance by several increasing circles around the ball. In SFL it's a short statement using the primitive symbol "ClosestPlayerToBall" instead of constant uniform numbers.

Other SFL-concepts extend the set of situation specification primitives, e.g. by making conditions depend on the stamina of an agent, the ball's velocity or the fact that the ball is interceptable. The set of actions was also extended with new features like "interceptball" and parameters for existing actions, e.g. velocities for positioning-actions and modes for passing.

Since Clang was continually extended, some concepts of SFL, like relative positions, were already included before publishing this paper. For the complete syntax of SFL see the final documentation [3].

2.2 Implementation

The actual implementation of our team is just one possible implementation about how to handle the SFL-specification. The Strategy Formalization Language System works as follows.

The behavior of our team is specified by a set of advice-tokens, i.e. rules with a condition and several effects. In each cycle a module, called matcher, evaluates the conditions of all rules with the world-model of the agent. Rules whose conditions are true in the actual cycle and that contain directives for the agent are labeled as "active".

Most of the time several rules are active. Thus, another module, called selector, has to decide which rule should be executed. The basic idea is that rules are heuristically evaluated on each of the three Clang levels (actions, directives and conditions), being assigned three fitness values that are summed up. So, certain actions seem more promising, e.g. interceptball has a higher fitness than marking on the action-level. Directives refer to different sets of players and the more specific a player set, the higher the directive's fitness on the directives level. E.g. the fitness of a directive that refers to the whole team has less fitness than a subset, which again has less fitness than a situation-specific player-symbol.

Since the rules specified in the team-implementation are fixed, the fitness-assignment is done manually, but automated assignment will be straight-forward.

The third level is based on the conditions and is basically a way to save world knowledge from the rules and incorporate them into the selector. So it is not necessary to specify in a rule that the agent should only mark an opponent if no teammate is already there. Certain common sense heuristics can be used on this level to assign a fitness value to each rule. This has not yet been implemented, so the SFL-rules in our team contain certain amounts of this common sense knowledge explicitly.

The selector will then execute the rule with the highest fitness, unless one of the rules contains the "force"-flag which denotes that this rule should always be executed if its condition is true.

The action is then handed over to the effector-module which encapsulates the strategic actions and executes the appropriate low-level-skills. In the case of a pass-action for example, the effector has to take care that the agent gets close enough to the ball or is turned in the proper direction.

Coach advices are added to the rulebase as they arrive via the communication channel and will be evaluated in the following cycles like the other rules. As of now, the assigned fitness is high, yet fixed. So it is possible for the team designer to specify rules that will not be overwritten by coach advices by assigning a higher fitness than the coach advice fitness.

3 The Online Coach

One of the main concerns of designing the Dirty Dozen online coach was to make it compatible with foreign teams. This was made possible by Clang at RoboCup 2001 for the first time. We use the coach to contribute in two ways: a) opponent/team modelling and subsequentally advising the agents to change their behaviour appropriately and b) facilitating coordination thanks to the coach's central role in communication and the global and noiseless information that it receives.

The coach looks for repeting patterns in the setplay[1] of the opponent, e.g. player positions during goalie-kickoffs. This information is sent to the agents so that they can position themselves strategically in the next setplay. Unfortunately, this requires a lot of inference on the agents' side and one can not rely on the team's designers having incorporated this mechanism into their players.

So a more promising way is to send not only information, but explicit advice about how to behave in certain situations. This encapsulates the inference about tactics and formations on the team level within the coach. E.g. our coach assigns tasks as to which agent should mark which opponent in defense situations. Since the coach does not know a priori which players are defenders, it has to model both teams in order to identify opponent forwards and its own team's defenders. The assignment of defenders to forwards is based on their spatial distribution on the field, so that each player is assigned an opponent that tends to be closeby in most defensive situations. Experiments showed that it is not wise to change only parts of the tactics, because most teams use well-tuned tactics that the coach might not be able to detect in the first place. In the marking assignment the coach therefore also advices new formation positions for the defenders based on the most likely position of the assigned opponent players. So the defensive formation is tuned to work with the marking assignments. Note that this would be hard to achieve in a distributed way if each agent had to decide for itself which player to mark, because the agents have only incomplete and noisy visual information and would have to communicate in order to avoid marking opponents twice or not at all.

4 Future Work

Considering the rather small set of primitive features in both Clang and SFL it seems that playing soccer successfully is not about having the best concepts, but about the domain knowledge regarding how to handle the existing concepts in given situations. So we will elaborate on the selector module to choose between rules more dynamically. This will also lead to a better integration of coach advices which are handled with fixed fitness values now. Because the rule-conditions typically formulated in SFL are more detailed and specific than Clang can express, the rule-selection needs more work.

Until now the players can only handle explicit advices from the coach and ignore opponent modelling information. An inference mechanism that also uses this information, i.e. can infer how to behave given certain information about the opponent's tactics, could also be useful for implementing the coach itself.

Additionally, as the coach contest showed, the coach needs to observe how its own messages influence the team's behavior and react appropriately. The opponent modelling has to be supported by modelling its own team. This year, all coaches kept on sending messages throughout the whole game, although it was obvious that the information only confused the foreign coached team. The online coach and the standard coach language are rather new concepts in the RoboCup domain and need more experiments and evaluation.

References

1. Peter Stone, Patrick Riley, and Manuela Veloso: The CMUnited-99 champion simulator team, In Veloso, Pagello, and Kitano, editors, RoboCup-99: Robot Soccer World Cup III , pages 35–48. Springer, Berlin, 2000.
2. Mao Chen, Ehsan Foroughi, Fredrik Heintz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Itsuki Noda, Oliver Obst, Patrick Riley, Timo Steffens, Yi Wang and Xiang Yin: Soccerserver Manual v7. 2001.
3. Sean Buttinger, Marco Diedrich, Leo Hennig, Angelika Hoenemann, Philipp Huegelmeyer, Andreas Nie, Andres Pegam, Collin Rogowski, Claus Rollinger, Timo Steffens, Wilfried Teiken: ORCA project report, 2001, (to appear)
4. Scerri P. and Ydren J.: End User Specification of RoboCup Teams, in RoboCup-99: Robot Soccer World Cup III, Springer. 2000