OMBO: An opponent modeling approach

Agapito Ledezma, Ricardo Aler,
Araceli Sanchis and Daniel Borrajo
Universidad Carlos III de Madrid
Avda. de la Universidad, 30
28911, Leganés (Madrid). Spain
E-mail: {ledezma,aler,masm}@inf.uc3m.es,
dborrajo@ia.uc3m.es

Abstract In competitive domains, some knowledge about the opponent can give players a clear advantage. This idea led many people to propose approaches that automatically acquire models of opponents, based only on the observation of their input-output behavior. If opponent outputs could be accessed directly, a model can be constructed by feeding a machine learning method with traces of the behavior of the opponent. However, that is not the case in the Robocup domain where an agent does not have direct access to the opponent inputs and outputs. Rather, the agent sees the opponent behavior from its own point of view and inputs and outputs (actions) have to be inferred from observation. In this paper, we present an approach to model low-level behavior of individual opponent agents. First, we build a classifier to infer and label opponent actions based on observation. Second, our agent observes an opponent and labels its actions using the previous classifier. From these observations, machine learning techniques generate a model that predicts the opponent actions. Finally, the agent uses the model to anticipate opponent actions. In order to test our ideas, we created an architecture, OMBO (Opponent Modeling Based on Observation). Using OMBO, a striker agent can anticipate goalie actions. Results show that in this striker-goalie scenario, scores are significantly higher using the acquired opponent's model of actions.

Keywords: Opponent modeling, Learning about agents, Hybrid learning

1. Introduction

A very important issue in multi-agent systems is that of adaptability to other agents, be it to cooperate or compete. In competitive domains, the knowledge about the opponent can give players a clear advantage [35]. A powerful way of achieving that knowledge consists of using machine learning to automatically build models of other players to be able to intelligently interact with other agents (either cooperating or opposing [27,37]).

Sometimes the inputs and outputs of agents to be modeled are available, and the model can be constructed from them. In previous papers, we have presented results for agents whose outputs are discrete [1], agents with continuous and discrete outputs [19], and an implementation of the acquired model in order to test its accuracy [18]. Among other robotic domains, we used the RoboCup Soccer 2D Simulator (RoboSoccer). There, we used the logs produced by another RoboCupSoccer simulation team's player to predict its actions using an action/parameter learning scheme [16]. In those papers, we considered that we had direct access to the opponent's inputs and outputs. However, in most domains (including Robosoccer) agents cannot see directly other agent's inputs (information sensed by the agent) and actions (actions performed by the agent). Instead, both have to be inferred by observing the behavior of the agent to be modeled.

The goal of this paper is to show an approach that automatically generates models of other agents when the specific inputs-outputs of that agent are unknown. We use an input-ouput approach in order to model opponent agents. Our approach, called OMBO (Opponent Modeling Based on Observation), uses machine learning (ML) techniques in order to map observations of the opponent agents into actions that they have executed (i.e. to label opponent actions from sensory data). In the case of many domains this model should be independent of the agent that is being modeled, given that it only accounts for changes in the environment caused by that action. For instance, in the RoboSoccer simulator domain, kicking the ball results in approximately the same effects on the environment (i.e. the ball changes its position), no matter what agent performs the action. This is somehow equivalent of building a model of how the simulator executes actions.

Once the action that the opponent has performed is labeled, a model about the other agent input-output behavior is built, also using inductive ML techniques.

Finally, the model of the other agent is used to improve the agent's performance. In this paper, we propose two ways of using the model. First, an expert on the domain designed a decision-making algorithm that uses the knowledge supplied by the model prediction. Second, we propose to automatically obtain the decision-making process itself. We generated random situations, and a ML technique learned which is the best action to perform, according to the model predictions.

In summary, our OMBO approach uses ML for opponent modeling at three levels to: label actions of any agent from observation (i.e. mapping sensory data into discrete actions); build the opponent model; and generate the decision-making algorithm. We have used the RoboSoccer as a domain where to test our approach. Results of the performed experiments show that using the acquired opponent model can improve the agent's performance.

The remainder of the paper is organized as follows. Section 2 presents our test domain, the RoboCup. Section 3 presents a summary on our learning approach to modeling. Actual results are detailed in Section 4. Section 5 discusses the related work. The paper concludes with some remarks and future work, in Section 6.

2. The RoboCup Soccer Simulator

The base of the soccer simulation league of RoboCup is the Soccer Server System [21]. This is a client-server software system. The Soccer Server provides a virtual field in which two soccer teams can play a match. The server simulates all movements of the ball and players (clients) and when the clients send a request to execute an action (e.g. turn, run, kick), the server modifies the current state of the world. Additionally, the server periodically sends information about the environment to each agent. This information is noisy and incomplete. The player only receives information about the objects in its vision range. The team's members are eleven independent players as well as a coach agent. The brain of each player on the field is a client that controls the actions of the player and communicates with the server through a standard network protocol with well-defined actions.

A standard game lasts for about 10 minutes. The server works with discrete time intervals known as "cycles". A second has ten cycles, thus, a standard game has 6000 cycles. In each cycle, an agent can send multiple actions to the server, and also receives information about the world from the server every 6 or 7 times per second. In our approach we want to model actions that can be sent one per cycle.

3. Opponent Modeling Based on Observation (OMBO)

Our approach carries out the modeling task in two phases (see Figure 1). In the first phase we create a generic module that is able to label the last action (and its parameters) performed by any robosoccer opponent based on the observations performed by the agent that is going to build the model (Action Labeling Module - ALM). As most activity recognition tasks, in order to obtain a model of other agent's behavior we need a log that matches sensory data with performed actions. In most of the work on activity recognition, actions are manually tagged by looking at sensory data. We propose an automatic way of tagging actions for the robosoccer simulator. In other words, we need this module given that in a game in the soccer simulator of the RoboCup, an agent does not have direct access to the other agents' inputs and outputs (what the other agent is really perceiving through its sensors and the actions that it executes at each moment). This module can be used for labeling later any other agent's actions. In a second phase, once we have a tagged log of sensory data from our agent and performed actions of the opponent, we create the model of the other agent based on ALM data inside of the Model Builder Module - MBM.

As we can see in Figure 1, after the opponent's model has been constructed, it can be used by the Reasoner Module REM to make the most suitable decision in order to react to the opponent's behavior.

The implementation of the modules that OMBO comprises is discussed in detail below.

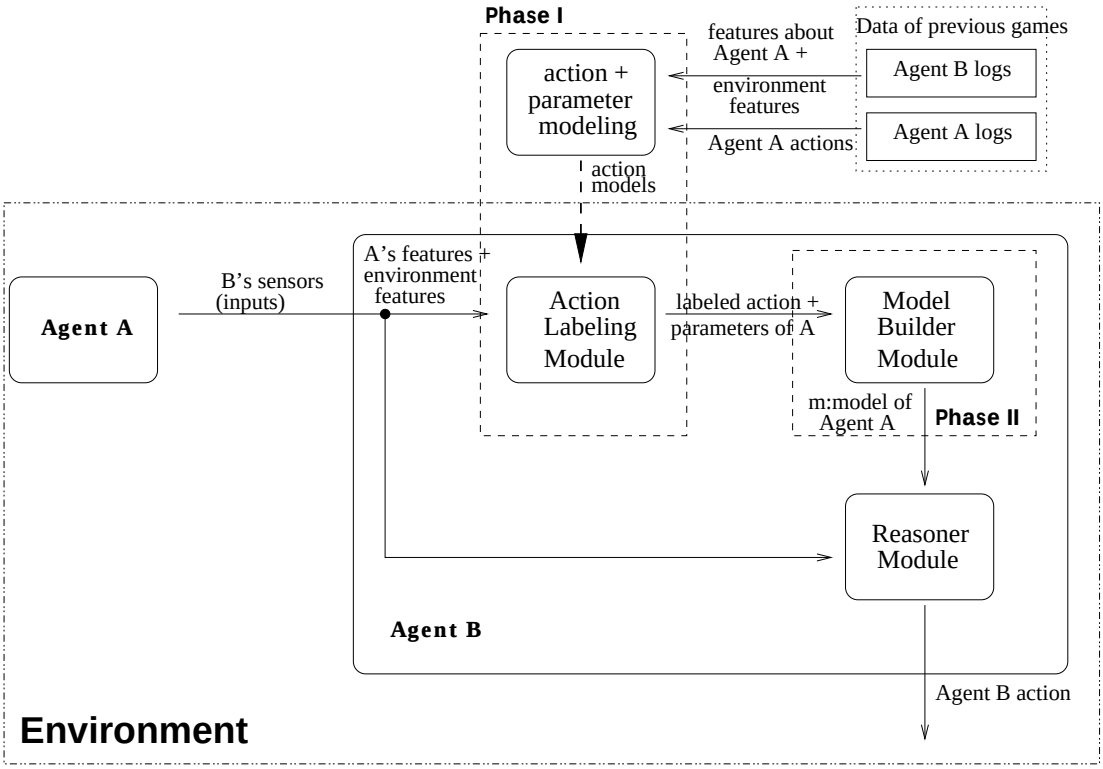

Figure 1. Architecture for opponent modeling.

*3.1. Action Labeling Module (*ALM*)*

In order to predict the behavior of the opponent (*Agent A*), it is necessary to obtain many instances of the form (Input Sensors, Output Actions), so that they can be used for learning. However, in a real match, *A*'s inputs and outputs are not directly accessible by the modeler agent (*Agent B*). Rather, *A*'s actions (outputs) must be inferred by *Agent B*, by watching *A*. For instance, if *Agent A* is besides the ball at time 1, and the ball is far away at time 2, it can be concluded that *A* kicked the ball. Noise can make this task more difficult.

The purpose of the ALM module is to classify *A*'s actions based on observations of *A* made by *B*. This can also be seen as a classification task. In this case, instances of the form (*observations of A's from B perspective,A's actions*) are required. Since we are building an agent-independent labeling module, we can use any *Agent A* for performing this training. We are somehow capturing the simulator behavior from a playing agent perspective.

The detailed steps for building the Action Labeling Module (ALM) are shown in Figure 2 and detailed below:

1. The *Agent A* plays against *Agent B* in several games. At every instant, data about *A* and some environment variables calculated by *B* are logged to produce a trace of *Agent A* behavior from *Agent B* point of view. On the other hand, the actual actions carried out by *AgentA* can be extracted from its logs or from the server logs.
2. Each example I in the trace is made of three parts in a given simulation step t:
   - a set of features, F, about *Agent A*,
   - some environment features, E, and
   - the actual action, C, that *Agent A* performed.

   In other words, given a simulation step t, each example is composed of $I_t = F_t, E_t, C_t$. From the trace of all simulation steps, $t = [1, 2, ..., n]$, it is straightforward to obtain a set of examples D so that *Agent B* can infer, through ML techniques, the action car-

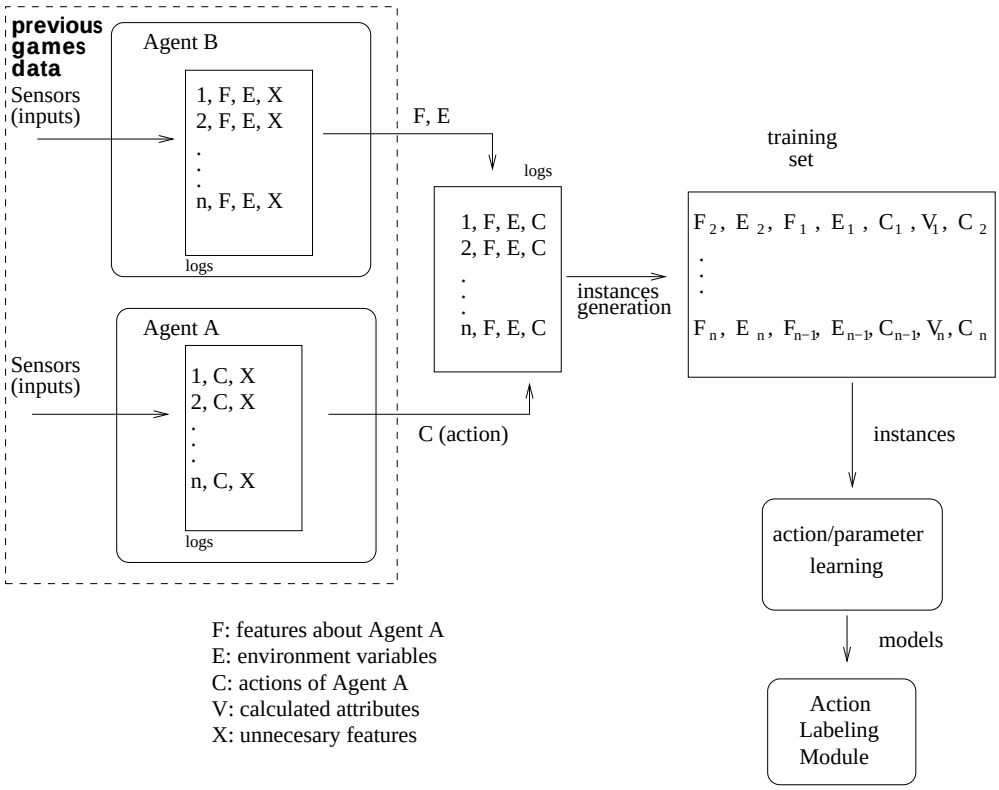

Figure 2. Action Labeling Module creation.

ried out by *Agent A* using its sensoring data from two consecutive simulation steps. Sometimes, the soccer server does not provide information for two contiguous time steps. We ignore these situations.

3. Let D be the whole set of available examples from *Agent A* trace. Each example $d_i \in D$ is made of two parts: an n-dimensional vector representing the attributes $a(d_i)$ and a value $c(d_i)$ representing the class it belongs to. In more detail, $a(d_i) = F_t, E_t, F_{t-1}, E_{t-1}, C_{t-1}, V$ and $c(d_i) = C_t$. V represents attributes computed based on comparison of feature differences between different time steps. We use 24 such attributes (e.g. *Agent A* position differences, ball position differences, etc. See appendix).
4. When the actions $c(d_i)$ in D are a combination of discrete and continuous values (e.g. *dash 100* - dash with power 100), we create a set of instances $\hat{D}$ with just the discrete part of the actions, and a set $\hat{D}_j$ for each parameter of the action j using only the examples corresponding to the same action. That is, the name of the action and the parameter of the action will be learned separately. For instance, if the action executed by the player is "dash 100" only *dash* will be part of $\hat{D}$ and the value 100 will be in $\hat{D}_{dash}$ with all the instances whose class is dash.
5. The set $\hat{D}$ is used to obtain a model of the action names (i.e. classify the action that the *Agent A* carried out at a given simulation step). The $\hat{D}_j$ are used to generate the parameters with continuous values associated to its corresponding action j. We have called this approach for learning the action and its parameter separately *action/parameter learning* (see Figure 3).
6. Finally, in order to label the action carried out by *Agent A*, we apply all classifiers in two steps. First, the action classifier label the action a_j that the agent has just performed. Second, the action a_j parameter classifier labels the value of the parameter of action a_j. This set of classifiers composes the Action Labeling Module (ALM).

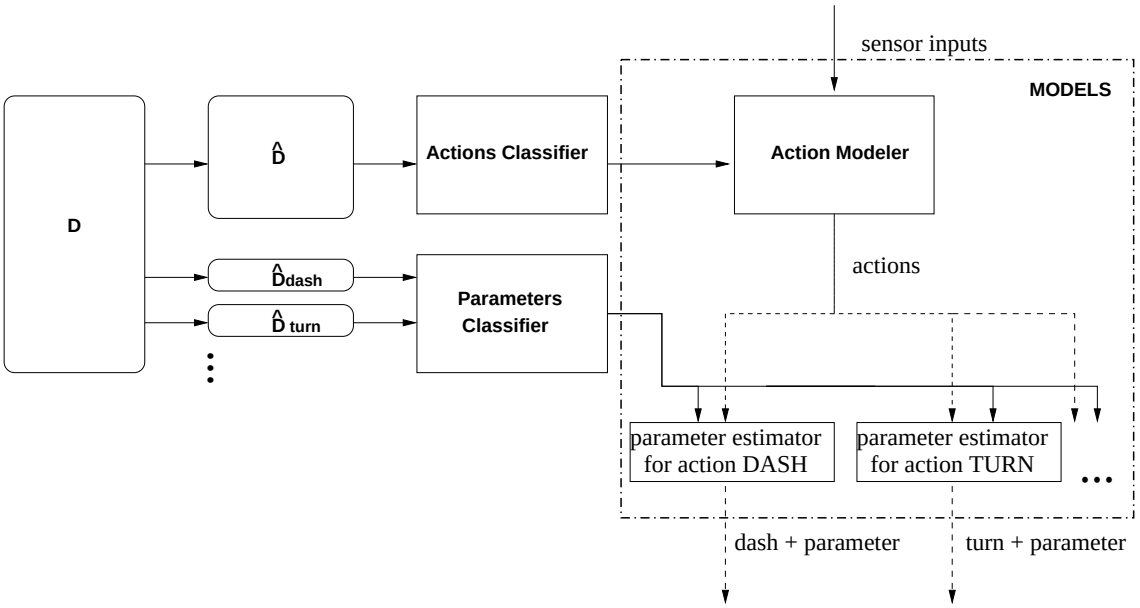

Figure 3. Action/Parameter Learning used by the ALM and MBM.

Kick, dash and turn are generic actions that have the same effect in the simulator independently of the agent that executes them (the action models are the same for all players). For instance, kicking the ball results in the same effects in the environment (the ball changes its position), irrespectively of the agent that performs that action. For this reason, the ALM is independent of *Agent A*, and could be used to infer the actions of other agents as well. In any case, a stronger agent-independent ALM could be obtained using a set of heterogeneous agents in order to construct it.

We assume we can access the real action of an agent whose behavior with respect to sensory actions is equal to the one we are going to predict later on. This is the case of RoboSoccer. In other domains this assumption does not hold and one would have to devise ways of obtaining those executed actions. For instance, one could manually tag a small set of instances and then run the ALM for the rest of the logs.

*3.2. Model Builder Module (*MBM*)*

So far, a classifier to label opponent's action has been obtained: ALM, and it can be used for the next step. Our next goal is to learn a classifier to predict a specific *Agent A*'s actions based on observations of *A* from *Agent B*'s point of view. It will be obtained from instances (F_t, E_t, ALM_t) recorded during the match, where ALM_t is the action classified by ALM from observations at t and $t-1$. The aim of this classifier is to predict *Agent A*'s behavior.

More specifically, data consists of tuples (I) with features coming from observing *Agent A*, some environment variables, and the action that *B* guessed the *Agent A* has performed, together with its parameter labeled by ALM. Instead of using a single time step, we have considered several of them in the same learning instance. Therefore, the learning tuples will have the form $(I_t, I_{t-1}, ..., I_{t-(w-1)})$, where w is the window size. Like in ALM construction, we used computed attributes, V.

The detailed steps taken for obtaining the model of *Agent A* are as follows:

1. The ALM is incorporated into *Agent B*, so that it can label (infer) *Agent A*'s actions.
2. *Agent A* plays against *Agent B* in different situations. At every instant, *Agent B* obtains information about *Agent A* and the guessed performed action of *Agent A*, labeled by ALM, as well as its parameter. All this information is logged to produce a trace of *Agent A* behavior.
3. Like in the ALM construction, every example I, at a given simulation step t in the trace, is made of three parts:

- a set of features, F, about *Agent A*,
- some environment features, E, and
- the action, C (labeled by ALM), of *Agent A*.

In other words, at simulation step t every example is $I_t = F_t, E_t, C_t$. We use information about *Agent A* from some previous simulations w steps.

4. Let D be the whole set of instances available at a given simulation step. Each instance $d_i \in D$ is made of two parts: an n-dimensional vector representing the attributes $a(d_i)$ and a value $c(d_i)$ representing the class it belongs to. In more detail,

$$a(d_i) = (F_t, E_t, F_{t-1}, E_{t-1}C_{t-1}, ..., F_{t-(w-1)}, E_{t-(w-1)}, C_{t-(w-1)}, V)$$

and $c(d_i) = C_t$.

5. Similarly, as in ALM, when the actions $c(d_i)$ in D are a combination of discrete and continuous values (e.g. dash 100), we create a set of instances $\hat{D}$ with only the discrete part of the actions, and a set $\hat{D}_j$ for each parameter of the action j using only the examples corresponding to the same action. From here on, action/parameter learning is used, just like in ALM, to produce action and parameter classifiers. They can be used later to predict the actions performed by *Agent A*.

Although the Model Builder Module could be used on-line, during the match, the aim of the experiments in this paper is to show that the module can be built and is accurate enough. Therefore, in our present case, learning occurs off-line.

*3.3. Reasoner Module (*REM*)*

Predicting opponent actions is not enough. Predictions must be used somehow by the agent modeler to anticipate and react to the opponent. For instance, in an offensive situation, an agent may decide whether to shoot the ball to the goal or not, based on the predictions about the opponent. The task of deciding when to shoot has been addressed by other researchers. For instance, CMUnited-98 [30] carried out this decision by using three different strategies based on: the distance to the goal; the number of opponents between the ball and the goal; and on a decision tree. In CMUnited-99 [31], Stone et al. [29] considered the opponent to be an ideal player, to perform this decision. Unlike Stone's work, in our approach we build first a general action-recognition classifier, an explicit model of the current opponent and we learn how to use the acquired opponent model.

A first approach for using the model is to program the reasoner module by hand. Then, it uses the prediction given by the model in a specific situation, in order to make a decision and execute the right action. In section 4 we will propose one way to achieve this, but there are many other possibilities.

The previous approach depends on the expert knowledge, as well as programming time and skills. It would still be better if the reasoning module could be automatically generated. One alternative would be using reinforcement learning (RL) and adding the prediction of the model as a new feature to the state vector [9]. Thus, the policy learned by RL would take into account both the information coming from the sensors and the prediction about the opponent.

In our case, we have followed a simpler approach shown in Figure 4:

1. First, we randomly generated many different scenarios in which the modeler is involved, that include the ball and other players. The potential scenarios are limited to those that are interesting for the learning task. For instance, we could have a striker and a goalie. In that case, the scenarios of interest are those where the striker starts at some distance of the goal, it is kicking the ball, and the goalie is near the goal.
2. Then, the modeler agent executes a random action, and the result is observed. Random actions would be restricted to those that make sense in a particular learning situation (for instance, a striker can either kick the ball or continue advancing towards the goal). According to the result, the action would be marked as positive or negative. The meaning of positive and negative depends on the particular learning task. For instance, in the case of the striker, scoring a goal would be marked as positive. By repeating this process many times, a sequence of *state, opponent prediction, action, class* will be generated. "State" is represented by the features constructed from data coming from the sensors. "Opponent prediction" is the action pre-

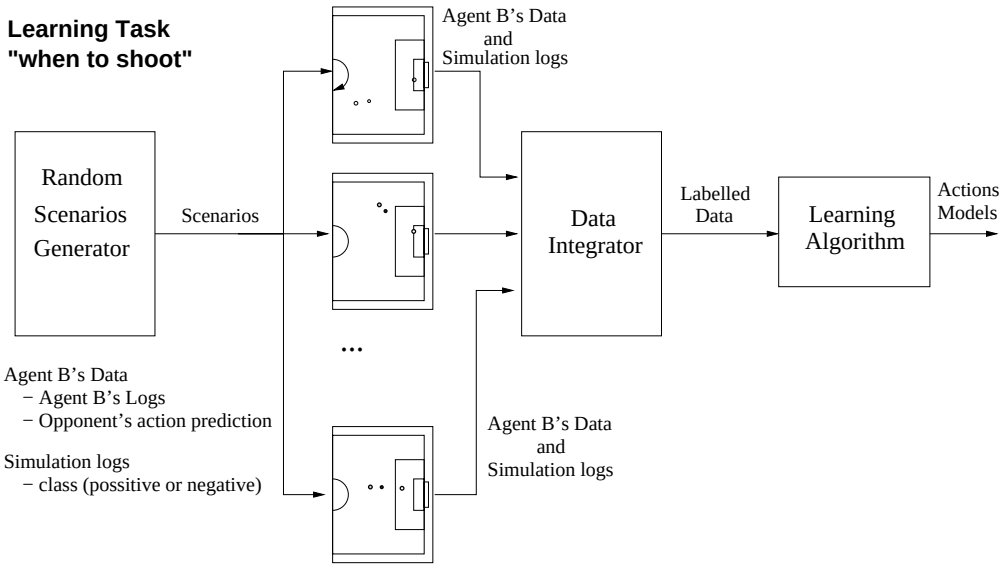

Figure 4. Scheme followed to generate the REM kernel.

dicted for the opponent by the model. "Action" is the one that was (randomly) executed in that particular situation. And "Class" is either positive or negative, depending on the outcome of performing the action.

3. Finally, these tuples are fed to a learning algorithm (for example C4.5 [22]) in order to construct a classifier that decides whether an action should be executed or not in a specific state taking into account the opponent prediction.

4. Experiments and Results

This section describes the experimental setup and results obtained. We have carried out three phases: first a player of a soccer simulator team (*Agent A*) plays against an *Agent B* to build the ALM; second, the model m of *Agent A* is built; and third, the model is used by B against A.

We have used two different players as the agent whose actions will be predicted (*Agent A*); the ORCA team [20] goalie and the Uva Trilearn [15] goalie. On the other hand, the *Agent B* (modeler agent) is based on the public part of the source code of CMUnited-99 [31]. The CMUnited99 team was simulation league champion in the RoboCup-99, and it is one of the most used source codes as base for other teams. *Agent A* will act as a goalie and *Agent B* as a striker, that must make the decision of shooting to the goal or continue advancing towards it, depending on the predictions about the goalie.

The techniques to model *Agent A* actions have been, in both, ALM and MBM, PART [10] and M5 [23]. PART is a descendant of C4.5 [22], and generates rules from a decision tree, and M5 generates regression trees. The latter are also rules, whose then-part is a linear combination of the values of the input parameters. PART has been used to predict the discrete actions (kick, dash, turn, etc.) and M5 to predict their continuous parameters (kick and dash power, turn angle, etc.). PART and M5 were chosen because we intend to analyze the models obtained in the future, and rules and regression trees obtained by them are quite understandable. In these experiments we use the implementation of PART provided by WEKA [39]. In relation to M5, we use a variant named M5RULES also implemented in WEKA. This algorithm implements routines for generating a decision list using M5 model trees and the approach used by the PART algorithm.

4.1. ALM *Construction*

As it is detailed in the previous section, the data used to generate the ALM, is a combination of the perception about *Agent A* from the point of view of *Agent B* and the actual action carried out by *Agent A*. Once the data has been generated, we

Table 1
Results of ALM creation. (R: correlation coefficient)

			ORCA GOALIE		UVATRILEARN GOALIE	
Labeling task	Attributes	Classes	Instances	Accuracy	Instances	Accuracy
Action	68	5	5095	70.81%	4545	77.14%
Turn angle	68	Continuous	913	0.007 R	2476	-0.038 R
Dash power	68	Continuous	3711	0.21 R	2067	0.32 R

use it to construct the set of classifiers that will be the ALM. Results of ALM creation are displayed in Table 1.

There are three rows in Table 1. The first one displays the classification of the action performed by *Agent A*, while the other two rows show the classification of the numeric parameters of two actions: turn and dash. As *Agent A* is a goalie, for experimental purposes, we only considered relevant the parameters of these actions. Columns two and three represent the number of attributes and the number of classes (continuous for the numeric attributes) used in these experiments. The rest of the columns show the number of instances used for learning and the accuracy obtained in the experiments with different goalies. These results have been obtained using a ten-fold cross validation.

ALM obtains between 70% and 77% accuracy for the discrete classes, which is a good result, considering that the simulator adds noise to the already noisy task of labeling the performed action.

Although a majority class algorithm (like ZeroR) obtains a 72% accuracy in the Orca goalie's case and a 70% in the UvaTrilearn agent's case, ALM is able to obtain a precision rate higher than 0 for all the classes (not only the majority one). Table 2 shows the True Positive (TP) and the False Positive (FP) rates per class. We can see that the error is distributed among all classes. In addition, the Precision (TP/TP+FP) is above 30% except in a case (class none/Orca goalie). Therefore, we can say that although the accuracy rate is similar to the one of a majority rule classifier, the learned model is more adapted because it considers all classes.

On the other hand, the results obtained for parameters values could be improved. These results are consistent with the results obtained by Blaylock & Allen [3] when using statistical methods for goal recognition. Besides using the m5Rules algorithm, we have also used artificial neural networks and linear regression techniques obtaining similar results, and in some cases even worse. It might be that better results could probably be achieved by discretizing the continuous values in order to apply classification algorithms. Moreover, it is not necessary to predict the numeric value with high accuracy; only a rough estimate is enough to take advantage of the prediction. For instance, it would be enough to predict whether the goalie will turn left or right, rather than the exact angle. In Section 4.3.1, where a reasoner module is programmed by hand, only the main action is considered. But in Section 4.3.2, where the reasoner module is generated automatically, both the main action and the parameters are used.

4.2. Model Construction

The ALM can now be used to label opponent actions and build a model, as explained in previous sections. PART and M5 have been used again for this purpose. Results are shown in Table 3.

Table 3 shows that the main action that will be performed by the opponent can be predicted with high accuracy (more than a 80%) considering the simulator noise.

Like in ALM construction, we compare our results with a majority rule classifier. This classifier obtains 79% and 73% accuracy for the Orca and Trilearn agents, respectively. In Table 4 we can see the TP and FP rates and the Precision per class. As we can see, the Precision rate is over 50% in all cases.

4.3. Reasoner Module (REM)

Once the model m has been constructed and incorporated into the *Agent B* architecture, it can be used to predict the actions of the opponent in any situation. The task selected to test the acquired model is *when to shoot* [29]. When the striker player is approaching the goal, it must decide whether to shoot right there, or to continue

Table 2
Results details per class in ALM creation. The "-" implies that the action was not executed by the agent

	ORCA GOALIE			UVATRILEARN GOALIE		
Class	TP Rate	FP Rate	Precision	TP Rate	FP Rate	Precision
turn	0.248	0.097	0.359	0.331	0.082	0.381
kick	-	-	-	0.303	0.003	0.435
dash	0.877	0.616	0.793	0.499	0.076	0.541
catch	0	0	0	0.256	0.002	0.524
unknown	0.274	0.048	0.361	0.925	0.294	0.88
none	0.067	0.002	0.091	-	-	-

Table 3
Model creation results.

			ORCA GOALIE		UVATRILEARN GOALIE	
Predicting	Attributes	Classes	Instances	Accuracy	Instances	Accuracy
Action	72	5	5352	81.13%	3196	85.39%
Turn angle	72	Continuous	836	0.67 R	278	0.46 R
Dash power	72	Continuous	4261	0.41 R	551	0.76 R

R: correlation coefficient

Table 4
Results details per class in MBM creation. The "-" implies that the action was not executed by the agent

	ORCA GOALIE			UVATRILEARN GOALIE		
Class	TP Rate	FP Rate	Precision	TP Rate	FP Rate	Precision
turn	0.4	0.069	0.519	0.454	0.042	0.514
kick	0	0	0	0.125	0	1
dash	0.919	0.576	0.862	0.739	0.057	0.728
catch	-	-	-	0	0.001	0
unknown	0.365	0.014	0.564	0.934	0.226	0.919
none	0	0	0	-	-	-

with the ball towards the goal. In our case, *Agent B* (the striker) will make the decision based on a model of the opponent goalie.

When deciding to shoot, our agent (B) first selects a point inside the goal as a *shooting target*. In this case there are two shooting targets: the two sides of the goal. The agent then considers its own position and the opponent goalie position to select which one of the shooting targets to shoot at. Once the agent is near the goal, it uses the opponent goalie model to predict the goalie reaction and decides to shoot or not at a given simulation step. For example, one potential algorithm would be that if it predicts the goalie will remain still, the striker advances with the ball towards the goal.

In order to test the effectiveness of our modeling approach in a simulation soccer game, we ran 100 simulations in which only two players take part. The *Agent A* is either an ORCA Team goalie or a Uva Trilearn goalie, whereas the *Agent B* is a striker based on the CMUnited-99 architecture with ALM and the model of *Agent A*. For each simulation, the striker and ball were placed in 30 different positions in the field randomly chosen. This makes a total of 3000 goal opportunities. The goalie was placed near to the goal. The task of the striker is to score a goal while the goalie must prevent it. Figure 5 shows an example of a scenario.

4.3.1. REM *coded by hand*

To test the model utility, we compare a striker that uses the model with a striker that does not. In all situations, the striker dribbles the ball towards the goal until deciding when to shoot. The striker that does not use the model decides when

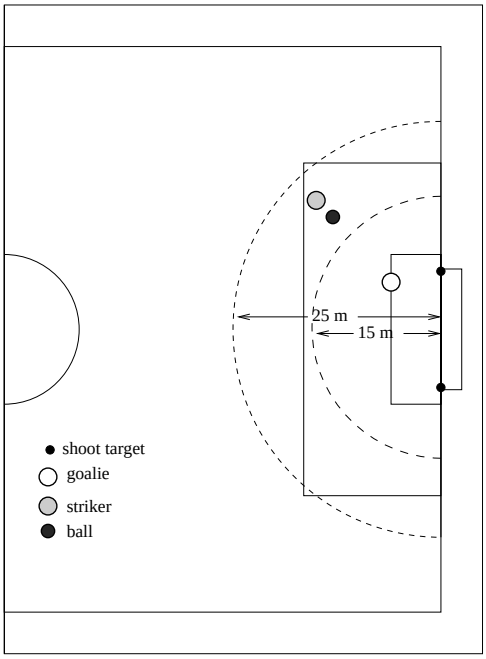

Figure 5. Example of simulated scenario.

to shoot based only on the distance to the goal, while the striker that uses the model, considers this distance and the goalie predicted action. The algorithm coded by hand that uses the prediction about the opponent is the following:

1. When the striker gets to a distance of 25 meters of the opponent goal, it begins to consider whether to shoot or to continue advancing.
2. In order to make the decision, the striker uses the model to predict whether the opponent goalie will move (dash). In that case, the striker will shoot towards the goal. Otherwise, it will continue advancing and consider whether to shoot or not in the next time step.
3. In any case, when the striker gets to a distance of 15 meters, it always shoots.
4. The shooting angle is computed considering the nearest shooting target and the angle between the selected shooting target and the opponent goalie. If the angle is greater than 20 degrees, then the striker shoots at the selected target. Otherwise, the striker shoots at the opposite target.

When no model is used, the striker always shoots at a distance of 25 meters, in case it has the ball. The shooting angle is computed in the same way as before.

4.3.2. REM *generated automatically*

In this section we will apply the algorithm described in Section 3.3 to the striker-goalie case. In order to generate the REM, the following steps were carried out:

- Thirty different scenarios that involve a striker and a goalie are randomly generated. The striker always appears in the opponent field side with the ball. The goalie always appears near its goal.
- When the striker gets into the 25 meter area, it decides randomly whether to shoot or to continue advancing.
- If the striker scores after shooting, a positive training instance is generated. Otherwise, it generates a negative one.
- Every training instance is made of 18 attributes, that include the striker, opponent, ball, and goal positions. Also, the prediction about the opponent is included by means of three attributes, the predicted action and its parameters (angle and power).
- Then, the PART algorithm is trained using the instances and the resulting classifier is incor-

porated to the REM. The striker will decide to shoot when the classifier predicts class positive (goal) if it shoots.

Our goal here is to automatically learn a model so that our agent can decide what action to perform next. In the experiments with the ORCA goalie, the training instances included 7414 negative cases and 2082 positive ones. In order to obtain good results for both classes, positive instances were replicated 4 times, obtaining a training set with 7414 negative and 8328 positive instances. For comparison purposes, we generated two classifiers. One of them with 15 attributes (i.e. no predictions are included), and another one with the full 18 attribute set (i.e. including the three attributes that represent the predictions about the opponent's next action). The 10-fold cross-validation accuracy results are 63.49% (no predictions) and 68.06% (with predictions), respectively. This implies that using the predictions given by the model helps to better discriminate positive and negative instances. In addition, we analyzed the rules generated by PART. We observed that 72.6% of the rules (138 out of 190 rules) actually used the three prediction attributes. This implies that the prediction attributes are important for deciding what action to perform next.

On the other hand, in the experiments with the Uva Trilearn goalie, the original training instances were composed of 14702 negative cases and only 292 positive ones. After the balancing process, we worked with a training set of 19666 instances with 4694 positive instances. We obtain a 98.18% accuracy in the cross-validation process with an accuracy rate of 0.934 for positive cases and 0.999 for negative ones. The model obtained has 199 rules of which 60 include the opponent action prediction.

4.3.3. Results

The average results of the 100 simulations (30 different scenarios per simulation) are shown in Table 5.

Analyzing the results obtained against the ORCA goalie, these show that the goals average using the model (both by hand and automatically) is higher than not using the model. Shots outside the goal are also reduced (10.47 and 9.61 using the model versus 11.18 without the model).

It can also be seen that when the REM is generated automatically, results improve over the handmade algorithm. We carried out a *t-test* to determine that these differences are significant at $\alpha = 0.05$. Using the model is always significatively better than not using it with respect to both goals average and shots outside average.

On the other hand, analyzing the results obtained against the UVA TRILEARN goalie, the score average using the opponent model is significatively higher than not using the opponent model again. UVA's goalie is much better than Orca's and that is why fewer goals are scored against UVA Trilearn even though the accuracy of the models is quite similar.

The difference between using the opponents models or not may seem small (between 0.48 and 1.96 goals every 30 opportunities). However, looking at the results of the soccer simulation league in the year 2007 [11], the goal difference in the four matches of the quarterfinals were between one and two goals. In other words, using the opponent model could be decisive in the outcome of a match.

5. Related Work

Our approach follows Webb feature-based modeling [37] that has been used for modeling user behavior. Webb's work can be seen as a reaction against previous work in student modeling, where it was attempted to model the internal cognitive processing of students, that cannot be observed. Instead, Webb's work models the user in terms of inputs and outputs, that can be directly seen. This approach can be applied to any domain where reaction is important, and the internal state of agents cannot be accessed or it is not important.

Besides the work in user modeling, there are other approaches in order to model the behavior of an agent. For example, in game theory, Carmel and Markovitch [4] propose a method in which the opponent model is represented as a deterministic finite automaton (DFA). Since the DFA uses an observations table in order to maintain a consistent model with the behavior of the opponent, this approach is limited to discrete domains and can not be used in complex domains like RoboCup. Moreover their approach has a high sensitivity to noise.

The *Recursive Modeling Method, RMM* [8,7] is another method for opponent modeling. Using *RMM* an agent can model the internal state of another agent and its action selection strategies in

Table 5
Simulations results using different strikers.

	Orca Goalie		UvaTrilearn Goalie	
Striker	Goals average	Shots outside average	Goals average	Shots outside average
Without model	4.65	11.18	0.38	4.64
Rem coded by hand	5.88	10.47	2.34	5.85
Rem generated automatically	5.97	9.61	0.86	4.17

order to predict its action. This method is recursive because the modeled agent might similarly be modeling the modeler agent. This modeling strategy can lead to an arbitrary depth of reasoning. Some work has been carried out in order to limit this depth [6,36]. One problem of *RMM* is that it constructs the agent's payoff matrices assuming that it knows the internal state of other agents and their actions capabilities.

Suryadi and Gmytrasiewicz [32] present a framework where an agent can model other agents in a multi-agent environment based on observed behavior. In their work, they start from models of agents represented as *influence diagrams* and a history of observed behavior in order to model other agents. They modify one of the available models when none of them are likely to be correct, based on observed behavior. In this work, the authors start from a model or a set of models, and they try to adjust the model or to select the model that describes best the opponent behavior. Our approach differs in that our aim is to build the model about the opponent from scratch.

Takahashi et al. [33] present an approach that builds a state transition model about the opponent (the "predictor"), that could be considered a model of the opponent, and uses reinforcement learning on this model. They also learn to change the robot's policy by matching the actual opponent's behavior to several opponent models, previously acquired. The difference with our work is that we use ML techniques to build the opponent model and that opponent actions are explicitly labelled from observation.

Another way to face the modeling agents task is through plan recognition [13,34]. Tambe [34] presents an algorithm for tracking agents in flexible and reactive environment (RESC: Real-Time Situated Commitments). Using RESC an agent can carry out the tracking of another agent inferring a hierarchy of operators (of the modeled agent) taking advantage of its own architecture. Avrahami-Zilberbrand and Kaminka [2] use decision trees for matching observations to the plan library in a method for symbolic plan recognition. Like in the work of Suryadi and Gmytrasiewicz, the plan recognition task starts from predefined models or plans stored in plan libraries. In our case, we do not have "a priori" models of other agents behaviors.

There has also been some work on agent modeling in the RoboCup soccer simulation domain. Most of the work focuses on the coaching problem (i.e. how the coach can give effective advice to the team based on, among other things, the opponent modeling). For example, Druecker et al. [5] use neural networks to recognize team formation in order to select an appropriate counter-formation, that is communicated to the players. Another example of formation recognition is described in Riley et al. [26] that use a learning approach based on player positions.

Riley and Veloso [25] presented an approach that generates plans based on opponent plans recognition and then communicates them to its teammates. In this case, the coach has a set of "a priori" opponent models.

Based on [24], Steffens [27] presents an opponent modeling framework in multi-agent systems. In his work, he assumes that some features of the opponent that describe its behavior can be extracted and formalized using an extension of the coach language. In this way, when a team behavior is observed, it is matched with a set of "a priori" opponent models.

Kaminka et al. [12] focus on learning sequential behaviors from observations of the agents behaviors. Their technique translates observations in a domain, like RoboCup games, into a time-series of recognized atomic behaviors. Then this time-series are analyzed in order to find repeating subsequences that characterize each team behavior.

The main difference with all these previous works is that we want to model opponent players

in order to improve low level skills of the agent modeler rather than modeling the high level behavior of the whole team.

On the other hand, Stone et al. [29] propose a technique that uses opponent optimal actions based on an ideal world model to model the opponent future actions. This work was applied to improve the agent low level skills. Our work addresses a similar situation, but we construct a real model based on observation of an specific agent, while Stone's work does not directly construct a model.

Steffens [28] propose a similarity-based approach to model high-level opponents actions (e.g. shoot towards goal). In its approach, Steffens propose the use of Case-based reasoning (CBR) in order to predict the opponent's actions from the coach point of view. He increases the classification accuracy including in the similarity measure some derived attributes from imperfect domain theories. As in our work, Steffens does not use predesigned models in order to predict the opponents actions. Nevertheless, unlike our approach, it uses the global information that can be received by the online coach and not the information that can be obtained by a player.

6. Conclusions and future work

In this paper we have presented and tested an approach to modeling low-level behavior of individual game opponent agents. Our approach follows three phases. First, we build a general classifier to label opponent actions based on observations. This classifier is constructed off-line once and for all future action labeling tasks in the same domain and type of agent. Second, our agent observes a specific opponent and labels its actions using the classifier. From these observations, a model is constructed to predict the opponent actions. This can be done on-line. Finally, our agent takes advantage of the predictions to anticipate the opponent actions.

In this paper, we have given a proof-of-principle of our approach by learning a model of two different goalies, so that our striker gets as close to the goal as possible, and shoots when the goalie is predicted to move. Our striker obtains a higher score by using the model against a fixed strategy. We have shown that the model that decides which action to perform can be built in two ways: by hand (using knowledge from an expert about the best way to use predictions about the opponent), and automatically (generating random situations and classifying them as positive or negative depending on the outcome). In both cases, using the model is significantly better than not using it.

In the future, we would like to do on-line learning, possibly using the game breaks to learn the model although this approach wouldn't work against the teams that change their behavior over the break. Moreover, we intend to use the model for more complex behaviors like deciding whether to dribble, shoot, or pass. Also, we would like to explore other ML approaches, such as RL techniques for some of the learning tasks (in particular, to build the Reasoner Module). Finally, we would like to extend our techniques to other domains, specially interactive ones, so that the computer agents can predict human player actions.

Acknowledgements

This work has been partially supported by the Spanish MCyT under projects TRA2007-67374-C02-02 and TIN-2005-08818-C04. Also, it has been supported under MEC grant by TIN2005-08945-C06-05. We thank anonymous reviewers for their helpful comments.

References

[1] R. Aler, D. Borrajo, I. Galván, and A. Ledezma, Learning models of other agents, in *P*rocs. of the Agents-00/ECML-00 Workshop on Learning Agents, Barcelona, Spain, 2000, pp. 1–5.

[2] D. Avrahami-Zilberbrand and G. Kaminka, Fast and Complete Symbolic Plan Recognition, in *P*rocs. of the International Joint Conference on Artificial Intelligence (IJCAI 2005)

[3] N. Blaylock and J. Allen, Recognizing instantiated goals using statistical methods, in *P*rocs. of the Workshop on Modeling Others from Observations, 2005, pp. 79–86.

[4] D. Carmel and S. Markovitch, Learning Models of Intelligent Agents, in *P*rocs. of Thirteenth National Conference on Artificial Intelligence (AAAI96), pp. 65–67, 1996.

[5] C. Druecker, C. Duddeck, S. Huebner, H. Neumann, E. Schmidt, U. Visser and H. Weland, *V*irtualweder: Using the online-coach to change team formations (Technical Report), TZI-Center for Computing Technologies, University of Bremen, 2000.

[6] E. H. Durfee, Blissful ignorance: Knowing just enough to coordinate well, in *P*rocs. of the First International Conference on Multi-Agent Systems (ICMAS-95), Menlo Park, CA: AAAI Press, 1995, pp. 406–413.

[7] E. H. Durfee, P.J. Gmytrasiewicz, and J. S. Rosenschein, The utility of embedded communications: Toward the emergence of protocols, in *P*rocs. of the Thirteenth International Distributed Artificial Intelligence Workshop, 1994, pp. 85–93.

[8] E. H. Durfee, J. Lee and P. J. Gmytrasiewicz, Overeager reciprocal rationality and mixed strategy equilibria, in *P*rocs. of the 11th National Conference on Artificial Intelligence, Washington, DC, USA: The AAAI Press/The MIT Press, 1993, pp. 225–230.

[9] F. Fernández and M. Veloso, Probabilistic Policy Reuse in a Reinforcement Learning Agent, in *P*rocs. of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'06), 2006.

[10] E. Frank, and I. Witten, Generating accurate rule sets without global optimization, in *P*rocs. of the Fifteenth International Conference on Machine Learning, Morgan Kaufmann, 1998, pp. 144–151.

[11] Georgia Institute of Technology, RoboCup wiki, [In-line] Available http://wiki.cc.gatech.edu/robocup/index.php/Main_Page

[12] G. Kaminka, M. Fidanboylu, A. Chang and M. Veloso, Learning the Sequential Behavior of Teams from Observations,*R*obot Soccer World Cup VI, Springer-Verlag, 2002, pp. 111–125.

[13] A. Kautz and J.F. Allen, Generalized Plan Recognition, in *P*rocs. of the Fifth National Conference on Artificial Intelligence (AAAI), AAAI Press, pp. 32–37, 1986.

[14] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda and E. Osawa, RoboCup: The robot world cup initiative, in *P*rocs. of the First International Conference on Autonomous Agents (Agents'97), New York: ACM Press, 1997, pp. 340–347.

[15] J. Kok, N. Vlassis and F.C.A. Groen, UvA Trilearn 2003 team description in *P*rocs. of the CD RoboCup 2003, Springer-Verlag, Padua, Italy, 2003.

[16] A. Ledezma, R. Aler, A. Sanchis and D. Borrajo, Predicting opponent actions in the robosoccer in *P*rocs. of the 2002 IEEE International Conference on Systems, Man and Cybernetics, 2002.

[17] A. Ledezma, R. Aler, A. Sanchis and D. Borrajo, Predicting opponent actions by observation, in *R*oboCup-2004: The Seventh RoboCup Competitions and Conferences. Berlin: Springer Verlag, 2005.

[18] A. Ledezma, A. Berlanga and R. Aler, Automatic symbolic modelling of co-evolutionarily learned robot skills, in *C*onnectionist Models of Neurons, Learning Processes, and Artificial Intelligence (IWANN, 2001), pp. 799–806.

[19] A. Ledezma, A. Berlanga, and R. Aler, Extracting knowledge from reactive robot behaviour in *P*rocs. of the Agents-01/Workshop on Learning Agents, Montreal, Canada, 2001, pp 7–12.

[20] A. G. Nie, A. Honemann, A. Pegam, C. Rogowski, L. Hennig, M. Diedrich, P. Hugelmeyer, S. Buttinger, and T. Steffens, *T*he osnabrueck robocup agents project (Technical Report), Institute of Cognitive Science, Osnabrueck, 2001.

[21] I. Noda. Soccer server: a simulator of robocup, in *P*rocs. of AI symposium'95, 1995, pp. 29–34.

[22] J. R. Quinlan, *C*4.5: Programs for machine learning, San Mateo, CA: Morgan Kaufmann, 1993.

[23] J. R. Quinlan, Combining instance-based and model-based learning, in *P*rocs. of the Tenth International Conference on Machine Learning, Amherst, MA: Morgan Kaufmann, 1993, pp. 236–243.

[24] P. Riley and M. Veloso, On behavior classification in adversarial environments, in *L*ynne E. Parker, George Bekey, and Jacob Barhen, editors, Distributed Autonomous Robotic Systems 4, pp. 371–380, Springer-Verlag, 2000.

[25] P. Riley, and M. Veloso, Planning for distributed execution through use of probabilistic opponent models, in *P*rocs. of the Sixth International Conference on AI Planning and Scheduling (AIPS-2002) 2002.

[26] P. Riley, M. Veloso, and G. Kaminka, Towards any-team coaching in adversarial domains in *P*rocs. of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2002.

[27] T. Steffens, Feature-based declarative opponent-modelling in multi-agent systems, *M*aster's thesis, Institute of Cognitive Science Osnabrück, 2002.

[28] T. Steffens, Similarity-based opponent modelling using imperfect domain theories, in G. Kendall and S. Lucas (Eds.) *I*EEE 2005 Symposium on Computational Intelligence and Games (CIG'05), pp. 285–291, 2005

[29] P. Stone, P. Riley, and M. Veloso, Defining and using ideal teammate and opponent agent models, in *P*rocs. of the Twelfth Innovative Applications of AI Conference, 2000.

[30] P. Stone, M. Veloso and P. Riley, The CMUnited-98 champion simulator team, In Asada and Kitano (Eds.), *R*oboCup-98: Robot soccer world cup II, pp. 61–76. Springer, 1999.

[31] P. Stone, P. Riley and M. Veloso, The CMUnited-99 Champion Simulator Team, In Veloso, Pagello and Kitano (Eds.), *R*oboCup-99: Robot Soccer World Cup III", Springer Verlag, 2000.

[32] D. Suryadi, and P. J. Gmytrasiewicz, Learning models of other agents using influence diagrams, in *P*rocs. of the Seventh International Conference on User Modeling, Banf, CA., pp. 223–232, 1999.

[33] Y. Takahashi, K. Edazawa and M. Asada, Multi-module learning system for behavior acquisition in multi-agent environment, in *P*rocs. of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems pp. 927–931, 2002.

[34] M. Tambe and P. Rosenbloom, RESC: An Approach for Real-time, Dynamic Agent Tracking, in *P*rocs. of the Fourteenth International Joint Conference on Artificial Intelligence, (IJCAI 95), Morgan Kaufmann, pp. 103–111, 1995

[35] H.J. van den Herik, H.H.L.M. Donkers, and P.H.M. Spronck, Opponent Modelling and Commercial Games, in *IEEE 2005 Symposium on Computational Intelligence and Games*, (eds. Graham Kendall and Simon Lucas), pp. 15–25, 2005

[36] J. M. Vidal and E. H. Durfee, Recursive agent modeling using limited rationality. *P*rocs. of the First International Conference on Multi-Agent Systems (ICMAS-95) pp. 376–383, Menlo Park, CA: AAAI Press, 95.

[37] G. Webb and M. Kuzmycz, Feature based modelling: A methodology for producing coherent, consistent, dynamically changing models of agents's competencies, *U*ser Modeling and User Assisted Interaction, **5**, pp. 117–150, 1996.

[38] G. Webb, M. Pazzani and D. Billsus, Machine learning for user modeling, *User Modeling and User-Adapted Interaction*, **11**, 2001.

[39] I. Witten and E. Frank, *D*ata mining: practical machine learning tools and techniques with Java implementations. Morgan Kaufmann, 2000.

Appendix

Table 6 shows the attributes used in the implementation of the ALM. We have used a total of 68 attributes (without the class), 44 of which are obtained directly from the modeler agent (basic attributes) and 24 are calculated attributes. The basic attributes obtained directly from the modeler agent correspond to perceptions and internal state data in two consecutive time steps (22 per time step). These attributes are shown in the upper part of the table. The calculated attributes are based on differences between the basic attributes in two consecutive time steps. In the lower part of the table 6 we can see the list of the calculated attributes and their descriptions.

On the other hand, in the implementation of the MBM, we have used the same attributes used in ALM implementation process except the "opponent number". We remove this attribute because we use only one type of opponent (the goalie). Moreover, we added the output of the ALM to the set of attributes, that is, the opponent class and parameters in two consecutives time steps (6 attributes).

Table 6
Used attributes in the classifier construction in ALM.

Name	Description
	ATTRIBUTES IN t INSTANT
SeeOpponent	Can the agent see the opponent?
OpponentNumber	Opponent number
BallKickableForOpponent	Can the opponent kick the ball?
CanFaceOpponentWithNeck	Can the agent face the opponent turn the neck?
CanSeeOpponentWithNeck	Can the agent see the opponent turn the neck?
BallMoving	Is the ball moving?
BallKickable	Is the ball kickable?
OpponentPositionValid	Certain about opponent position
OpponentDistance	distance to the opponent
OpponentSpeed	opponent speed
OpponentAngleFromBody	opponent angle from agent's body
OpponentAngleFromNeck	opponent angle from agent's neck
BallPositionValid	Certain about ball position
BallSpeed	ball speed
BallDistance	distance to the ball
BallAngleFromBody	ball angle from agent's body
BallAngleFromNeck	ball angle from agent's neck
MyBodyAng	agent's body angle
MySpeed	agent's speed
MyAction	agent's action
MyActionAngle	angle asociated to agent's action
MyActionPower	power asociated to agent's action
	ATTRIBUTES at $t-1$
Same attributes used in t	
	CALCULATED ATTRIBUTES
DIF-BKFO	BallKickableForOpponent difference between two time instants
DIF-CFOWN	CanFaceOpponentWithNeck difference between two time instants
DIF-CSOWN	CanSeeOpponentWithNeck difference between two time instants
DIF-BM	BallMoving difference between two time instants
DIF-BK	BallKickable difference between two time instants
DIF-OX	Opponent X axis difference between two time instants
DIF-OY	Opponent Y axis difference between two time instants
DESP-O	Opponent moves from one time instant to another
DIF-OD	OpponentDistance difference between two time instants
DIF-OS	OpponentSpeed difference between two time instants
DIF-OAFB	OpponentAngleFromBody difference between two time instants
DIF-OAFN	OpponentAngleFromNeck difference between two time instants
DIF-BX	Ball X axis difference between two time instants
DIF-BY	Ball Y axis difference between two time instants
DESP-Ball	Opponent moves from one time instant to another
DIF-BS	BallSpeed difference between two time instants
DIF-BD	BallDistance difference between two time instants
DIF-BAFB	BallAngleFromBody difference between two time instants
DIF-BAFN	BallAngleFromNeck difference between two time instants
DIF-MyX	Agent X axis difference between two time instants
DIF-MyY	Agent Y axis difference between two time instants
DESP-My	Agent moves from one time instant to another
DIF-MyBA	MyBodyAng difference between two time instants
DIF-MyS	MySpeed difference between two time instants
CLASS	Opponent's action in $t-1$