Using Golog for Deliberation and Team Coordination in Robotic Soccer

Alexander Ferrein Christian Fritz Gerhard Lakemeyer

Robotic soccer provides an interesting and nontrivial testbed for many aspects of mobile robotics. From a high-level decision making point of view, central issues are how to get the robots to choose intelligently among various possible courses of actions and how to get them to coordinate their actions with other members of the team. In this paper we report on our efforts to address these issues within the framework of the Golog action language.

1 Introduction

Robotic soccer [1] provides an interesting and nontrivial testbed for many aspects of mobile robotics. From a high-level decision making point of view, which is our main concern, central issues are how to get the robots to choose intelligently among various possible courses of actions and, soccer being a team sport, how to get them to coordinate their actions with other members of the team.

In this paper we report on our efforts to address these issues within the framework of the Golog action programming language, originally developed by Reiter and his colleagues [26]. While Golog has been applied previously to the control of single robots, robotic soccer is particularly challenging as it not only calls for coordinated actions among several robots but also poses hard real-time constraints in a highly dynamic environment where uncertainty abounds. In particular, actions often need to be selected within a fraction of a second and the execution of a plan may fail more often than not. To address these issues we have developed the Golog dialect Readylog, which offers useful features such as event-driven action initiation and probabilistic actions. Readylog also includes a novel kind of decision-theoretic planning. In particular, it allows to monitor at execution time whether a plan, also called a policy, is still valid or should be abandoned because of unforeseen circumstances. In case the planning process does not yield a result in time, deliberation is complemented with a fast reactive action selection mechanism based on decision-tree learning as a fall-back.

The rest of the paper is organized as follows. After a brief introduction to Golog we discuss our system architecture allowing to combine reactivity with deliberation. In Section 4 we give a high-level description of Readylog and present example programs for coordinated actions among robots. Section 5 focuses on the decision-theoretic extensions of Readylog. Before we conclude, we give a brief overview of the related work in Section 6.

2 Golog

Golog is based on Reiter's variant of the Situation Calculus [33, 28], a second-order language for reasoning about actions and their effects. Changes in the world are only due to actions so that a situation is completely described by the history of actions starting in some initial situation. Properties of the world are described by *fluents*, which are situation-dependent predicates and functions. For each fluent the user defines a successor state axiom specifying precisely which value the fluent takes on after performing an action. These, together with precondition axioms for each action, axioms for the initial situation, foundational and unique names axioms, form a so-called *basic action theory* [33].

Golog emerged to an expressive language over the recent years. It has imperative control constructs such as loops, conditionals [26], and recursive procedures, but also less standard constructs like the nondeterministic choice of actions. Extensions exist for dealing with continuous change [19] and concurrency [10], allowing for exogenous and sensing actions [11] and probabilistic projections into the future [18], to name just a few.

A recent extension, DTGolog [6], introduces decision-theoretic planning. DTGolog uses basic action theories to give meaning to primitive actions and it inherits all of Golog's programming constructs. From MDPs DTGolog borrows the notion of a *reward*, which is a real number assigned to situations indicating the desirability of reaching that situation, and *stochastic actions*. To see what is behind the latter, consider the action of intercepting a ball in robotic soccer. Such an action routinely fails and we assign a low probability such as 0.2 to its success. To model this in DTGolog, we define a stochastic action $intercept$. It is associated with two non-stochastic or deterministic actions $interceptS$ and $interceptF$ for a successful and failed intercept, respectively. Instead of executing $intercept$ directly, nature chooses to execute $interceptS$ with probability 0.2 and $interceptF$ with probability 0.8. The effect of $interceptS$ can be as simple as setting the robot's position to the position of the ball. The effect of $interceptF$ can be to teleport the ball to some arbitrary other position and setting the robot's position to the old ball position.(While this

model is certainly simplistic, it suffices in most real game situations, since all that matters is that the ball is not in the robot's possession after a failed intercept.) These ingredients basically form the language Readylog which is suitable to program robots in dynamic domains like robotic soccer.

3 The DR-Architecture

While deliberation has many advantages for decision making of a robot, it has the disadvantage of being slow compared to generating actions in a reactive fashion. In [13] we proposed a hybrid architecture which allows the combination of deliberation with reactivity. In this Section we give an overview of our architecture.

Figure 1 shows the DR-Architecture. From the sensory input we build our world model. It contains data like the own position, or the ball. The world model integrates also the perceptions made by the team-mates, yielding information about the positions of the other team members, the opponents (by classifying dynamic obstacles as opponents), and better information about the ball position (by fusing the local ball perception). Using a global world model improves the quality of the available data for the robot, e.g. if it does not see the ball by itself it still knows where it is. From these basic information a high-level world model is calculated. It contains derived information like $bestGoalCorner$ which indicates the unoccupied side of the opponent goal.

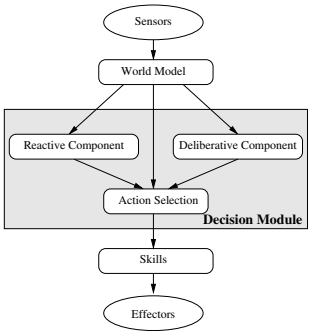

Figure 1: The DR-Architecture

The decision module in the DR-Architecture is divided into three modules. To be able to settle an action immediately the *Reactive Component* computes the next action to be executed based on the current game situation. It evaluates a decision tree which was learned using Quinlan's C4.5 [32]. We will not go into details about the reactive component here. For an application of C4.5 as decision module in robotic soccer confer to [24], which describes an application of C4.5 in RoboCup's simulation league. We will defer the detailed discussion of *Deliberative Component* until the next section. For now it is sufficient to know that it also provides an action to be executed next.

Having two components proposing possible actions to be performed, one needs an *Action Selection* method, deciding which action is going to be executed. Currently, our focus is on evaluating the deliberative component. Therefore, we use the reactive component only as a kind of fall-back system if the deliberative component could not propose an action in time. So far, our action selection strategy is rather trivial: whenever there is a suggestion from the deliberative component we use it.

Having calculated an action to be performed from the decision layer it has to executed. The action we use in RoboCup's Middle-size league are actions like *dribbleTo*, *kick*, *defendPosition*. The module *Skills* implements these complex behaviors and translates them into effector commands. To be as flexible as possible the actions are executed following a least commitment strategy. The arguments can take qualitative arguments like $dribbleTo(bestGoalCorner)$, denoting the free goal corner, which are offered by the world model. The best goal corner is evaluated at the last moment when the action is performed. This is especially important for goals which cannot be satisfied immediately.

4 Readylog

Readylog is an extension of Golog useful for specifying the behavior of robots in highly dynamic real-time domains. It offers the following control constructs:

$a_1; a_2$	sequence
$a_1 \mid a_2$	nondeterministic choice
$solve(p, h)$	(offline) dt-planning
$?(c)$	test
$waitFor(c)$	event-interrupt
$if(c, a_1, a_2)$	conditional
$while(c, a_1)$	loop
$withCtrl(c, a_1)$	condition-bounded execution
$pconc(a_1, a_2)$	concurrent actions
$prob(val_{prob}, a_1, a_2)$	probabilistic actions
$pproj(c, a_1)$	probabilistic (offline) projection
$proc(name(parameters), body)$	procedures

To illustrate the use of these constructs we give some examples from the soccer domain. We will only hint at the formal semantics of program execution at the end of the section. To make the following programs more readable we use a slightly more intuitive notation for the above constructs.

The control loop of our robots can be specified using concurrency and condition-bounded execution. To control different phases of the game special signals like *beforeKickOff* or *KickOffLeft* are sent from an external computer by the referee. According to the signal a particular procedure is executed by the robot. The $withCtrl$ statement is used for this purpose. Intuitively it means that as long as the condition holds, i.e. the referee's signal in this example, the particular procedure is executed:

proc mainloop
  while *gameRunning* **do**
    $pconc(withCtrl(\text{beforeKickOff}, \text{positionOnField}),$
      $withCtrl(\text{gameOn}, \text{playSoccer}), \ldots$
  endwhile
endproc

The following programs are examples of how robots can be coordinated in our framework. In particular, we show the

coordination of a double pass between two robots making use of probabilistic projections in order to evaluate possible pass partners.[1]

At some point in the high-level program the robot comes to evaluate the possibility to play a double-pass. In this particular situation, suppose an opponent staying right in front of the player with the ball is to be outplayed with a double pass.

First, the robot tries to find possible pass partners which are provided by the world model. For each pass partner it projects the program tryDoublePass with the current candidate partner. If the probability of success for the pass with the respective partner is above 0.9 it tries to play the pass by executing the playPass program.

```
proc DoublePass(Own)
  getNextPassPartner(Own);
  while ∃p.passPartner(p) do
    if pproj(hasBall(Own), TryDoublePass(Own, p))
     ≥ 0.9 then playPass(Own, p)
    endif
  endwhile
endproc
```

The procedure tryDoublePass$(Own, Teammate)$ first tries to find a free space behind an opponent staying right in front of the player with ball. The action lookForFreeSpace alters the value of the fluent *freePosition* to an unoccupied position in that free region of the field. The player with the ball (Own) passes the ball to his free teammate and starts running to the free position in order to receive the second pass. In the meantime, the teammate tries to receive the first pass taking also the possibility into account that the opponent which is going to be outplayed tries to intercept the ball. Therefore, the teammate has to check if the reception of the pass was successful, i.e. if the ball is kickable for itself after the reception of the pass. If this does not hold in the simulation the whole procedure fails meaning that with the current pass partner a safe double pass cannot be tried. Otherwise the double pass is executed.

```
proc tryDoublePass(Own, Teammate)
  lookForFreeSpace(Own, Teammate);
  directPass(Own, Teammate);
  pconc(receivePass(Teammate),
        interceptPass(closestOpponent(Teammate),
        if ballIsKickable(Teammate)
        then passTo(Teammate, freePosition);
             interceptPass(closesOpponent(Own))
       endif
        moveToPos(Own, freePosition);
        receivePass(Own)
      )  % endpconc
endproc
```

Note that we do not use explicit communication in this example. Under the assumption that the perception of agents does not differ too much, as is the case in the Simulation league, this form of coordination can be applied.

[1] Note that the example is intended for and has successfully been tested in our Simulation league team. In the Middle-size league, such complicated maneuvers are yet too challenging.

In addition to programs like the above, the user needs to specify the underlying basic action theory to give meaning to primitive actions and to specify an initial situation. Note also that the actions directPass and interceptPass used above are probabilistic. The respective probabilities were derived empirically. For a complete axiomatization of the double pass scenario we refer to [14].

The formal semantics of Readylog is an adaptation of the *transition semantics* proposed for ConGolog [10]. We will not go over the details and only hint at the main ideas. To define the meaning of a program, a special predicate $Trans(\delta, s, \delta', s')$ is introduced denoting a transition from configuration $\langle \delta, s \rangle$, i.e. a program δ in a situation s, to configuration $\langle \delta', s' \rangle$. All language constructs are defined in terms of the predicate $Trans$. To illustrate the transformation of a program, consider the definition of a while-loop from [10]:

$$Trans(while(\varphi, p), s, \delta', s') \equiv$$
$$\exists \delta''. Trans(p, s, \delta'', s') \wedge \varphi[s] \wedge \delta' = \delta''; while(\varphi, p)^2$$

Intuitively, the *while* construct with φ as the loop condition and p as the loop body is transformed into the program p with the *while* construct concatenated, if there exists a legal transformation to program p in the case where φ holds. Thus, the while loop is executed as long as the condition holds or there does not exist a transformation for program p.

Besides $Trans$, the semantics also makes use of a special predicate $Final(p, s)$ which is true just in case the program p can legally terminate in situation s. For example, the empty program nil is always final and a primitive action never is. If $Final(p, s)$ is true, $\langle p, s \rangle$ is called a final configuration. The meaning of a successful execution of a program starting in some initial situation can then be defined as a sequence of transitions leading to a final configuration.

5 Decision-theoretic Planning

In the previous section we showed the use of several important Readylog features needed to control and to coordinate the robots. In this section we concentrate on how to evaluate candidate plans in order to execute the most promising one. For this purpose we integrated decision-theoretic planning into Readylog based on the work of [6]. As pointed out in the introduction DTGolog calculates a policy maximizing the expected cumulated reward. The search for a policy can be restricted by a DTGolog program. The output is a conditional DTGolog program which for each state the system is in provides an optimal action to perform.

In DTGolog planning is undertaken by means of nondeterministic choice of actions. The best action according to the optimization theory is selected. When an action is stochastic the robot does not know which of the possible

[2] $\varphi[s]$ denotes the situation calculus formula obtained from φ by restoring situation variable s as the suppressed situation argument for all fluent names mentioned in φ. Also note that free within formulas are implicitly universally quantified.

effects occurs in the real word. Therefore, the resulting policy must provide a branch for each possible outcome of an action. DTGolog represents this tree as a Golog program.

To illustrate how a policy is calculated we show the definition of a nondeterministic choice of action $(p_1|p_2)$ taken from [6].

$$\begin{aligned}&BestDo((p_1|p_2);p,s,h,\pi,v,pr) \stackrel{def}{=}\\ &\quad \exists\pi_1,v_1,pr_1.BestDo(p_1;p,s,h,\pi_1,v_1,pr_1) \wedge\\ &\quad \exists\pi_2,v_2,pr_2.BestDo(p_2;p,s,h,\pi_2,v_2,pr_2) \wedge\\ &\quad ((v_1,p_1) \geq (v_2,p_2) \wedge \pi = \pi_1 \wedge pr = pr_1 \wedge v = v_1) \vee\\ &\quad (v_1,p_1) < (v_2,p_2) \wedge \pi = \pi_2 \wedge pr = pr_2 \wedge v = v_2)\end{aligned}$$

The semantics of DTGolog is defined in terms of the macro $BestDo$. For both choices a policy is constructed. The best one is selected by optimizing the value and success probability. For how the preference of (v,p) can be defined we refer again to [6].

As DTGolog is an offline interpreter, i.e. a policy is calculated up to a given horizon before it is executed, it is not suitable for highly dynamic domains. The reason is that the world may have changed too much before a policy is executed and becomes inapplicable soon. What is needed is a so-called on-line version of DTGolog. On-line means that an actions is executed immediately after it has been calculated.

Soutchanski [35] proposed an on-line version of DTGolog which overcomes the problem of planning too far ahead before executing the first action by interleaving planning with plan execution. The problem in his interpreter is, though, that he assumes *active* sensing to update world model information. Active sensing means, roughly, that sensing is initiated explicitly by special actions in the control program. As the outcome of sensing is unknown at planning time, it is in general not possible to calculate a policy beyond sensing actions. As sensing in highly dynamic domains such as ours is ubiquitous and happens many times a second, the use of active sensing and hence Soutchanski's approach is not feasible.

To overcome this problem, Readylog uses passive sensing as proposed by Grosskreutz and Lakemeyer in [19]. The idea is that sensor values are updated "in the background" by exogenous actions, which are not part of the program controlling the robot's behavior. To be able to project into the future or to plan when using this form of sensing one needs to build models of the actions which are used for planning. A simple example are computed trajectories of a robot's movements.

In Readylog we combine the model-based passive sensing approach with DTGolog. The models used in Readylog for the soccer domain are as simple as the example of intercepting a ball given in the introduction. The planning then works as follows. The interpreter is switched into an off-line mode, where the policy is calculated. Afterwards, the interpreter is turned on-line again to execute the policy. For this purpose we introduced the operator $solve$, taking a program and a horizon up to which depth the policy should be created as arguments.

$$\begin{aligned}&Trans(solve(p,h),s,\delta',s') \equiv\\ &\quad \exists\pi,v,pr.BestDo(p,s,h,\pi,v,pr)\\ &\quad \wedge\ \delta' = applyPol(\pi) \wedge s' = s.\end{aligned}$$

The predicate $BestDo(p,s,h,\pi,v,pr)$ evaluates a policy for program p in situation s up to a fixed horizon h, in a way very similar to [6]. The policy is denoted by π, the value of π is v, and pr stands for the probability of success. A complete definition of $BestDo$ for all constructs of DTGolog is given in [6]. After the policy is calculated it is executed using a special $applyPol$ transition. This special transition is needed because of the way we represent the policy. Using abstract models of the world also means to make assumptions of how the world might evolve. To check the validity of a policy when executing it against the assumptions made during planning we must keep track of all assumptions made along the way. Consider the case where the robot first tries to intercept the ball before it tries to shoot a goal. According to the model used in the planning phase for the intercept action the robot is in ball possession afterwards. Only then is the shoot action possible. Sadly, during actual execution things often go wrong, invalidating the current policy. For instance, the intercept action can fail for numerous reasons such as an opponent capturing the ball.

In order to be able to monitor whether a policy becomes invalid due to unanticipated changes in the world, we insert special markers $\mathfrak{M}(\varphi,v)$ into the policy, where φ is a condition and v its respective truth value at planning time. For instance, a conditional in Readylog is then defined as

$$\begin{aligned}&BestDo(if(\varphi,p_1,p_2);p,s,h,\pi,v,pr) \stackrel{def}{=}\\ &\varphi[s] \wedge \exists\pi_1.BestDo(p_1;p,s,h,\pi_1,pr) \wedge \pi = \mathfrak{M}(\varphi,\top);\pi_1 \vee\\ &\neg\varphi[s] \wedge \exists\pi_2.BestDo(p_2;p,s,h,\pi_2,v,pr) \wedge \pi = \mathfrak{M}(\varphi,\bot);\pi_2\end{aligned}$$

In the case where φ holds (in our model) a policy for program p_1 is calculated otherwise the policy is created for p_2. In both cases we prefix the policy with the marker stating whether φ holds or not. Note that except for the marker the definition of the $BestDo$ predicate is the same as in DTGolog.

When executing a policy we must treat the marker in a special way. The condition is evaluated again (denoted by $\varphi[s]$) and compared with the value stored at planning time. If they have the same truth value, the policy is further executed, otherwise we know that some assumption made during planning is not valid in the real world any more rendering execution of the policy impossible. In the case where $\varphi[s]$ and v mismatch, the policy is discarded. Please note that the situations s in the definition of the conditional above and in the following definition of the $Trans$ predicate are not the same.

$$\begin{aligned}&Trans(applyPol(\mathfrak{M}(\varphi,v);\pi),s,\delta',s') \equiv s = s' \wedge\\ &\quad ((v = \top \wedge \varphi[s] \vee v = \bot \wedge \neg\varphi[s]) \wedge \delta' = applyPol(\pi) \vee\\ &\quad (v = \top \wedge \neg\varphi[s] \vee v = \bot \wedge \varphi[s]) \wedge \delta' = nil)\end{aligned}$$

In addition, we must take care of primitive and stochastic as well as conditionals in the monitoring process. The details of how to treat those are given in [16].

In the following we give an example from our Middle-

```
solve(nondet(
 [kick(ownNumber, 40),
  dribble_or_move_kick(ownNumber),
  dribble_to_points(ownNumber),
  if(isKickable(ownNumber),
     pickBest(var_turnAngle, [-3.1, -2.3, 2.3, 3.1],
       [turn_relative(ownNumber, var_turnAngle, 2),
        nondet([[intercept_ball(ownNumber, 1),
           dribble_or_move_kick(ownNumber)],
           [intercept_ball(no_ByRole(supporter), 1),
            dribble_or_move_kick(no_ByRole(supp.))]])]),
     nondet([[intercept_ball(ownNumber, 1),
           dribble_or_move_kick(ownNumber)],
           intercept_ball(ownNumber, 0.0, 1)]) ) ]), 4)
```

Figure 2: The *bestInterceptor* program performed by an offensive player. Here, $nondet(\ldots)$ refers to the nondeterministic choice of actions.

size league team. The robot can choose between alternative plans by using the nondeterministic choice of actions. Fig. 2 shows a part of our attacker program we used for our RoboCup Middle Size robot team at the world championships 2003 in Padova, Italy, 2004 in Lisbon, Portugal, and at the German Open 2004 in Paderborn.

The robot selects between kicking the ball directly (if possible), and dribbling the ball. In the case that the ball is in the kicking device ($isKickable$, line 5) it moreover evaluates if it can pass the ball aside so that its teammate can overtake the ball (l. 10). This is modeled by a turn movement where the ball rolls to the side of the supporting teammate. The $pickBest$ construct chooses the optimal angle for the turn movement. In this case the robot also evaluates if the supporting player has a good opportunity to dribble the ball towards the opponent goal (l. 11). Here, the coordination is similar to the double pass example. The robot plans from the view of the supporting player. This is again possible because the global world model yields the needed information.

Fig. 3 depicts the policy for the case where the robot plays the pass to its supporting teammate.

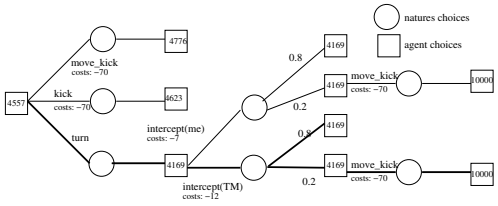

Figure 3: A (pruned) example decision tree for the *bestInterceptor* program.

For readability we pruned some similar branches. The root node stands for the situation were the agent switched to off-line mode, i.e., the current situation. The boxes denote agent choices, i.e. the agent can decide which of the alternatives to take. The circles are nature's choices, denoting the possible outcomes of stochastic actions. Numbers of outgoing edges in these nodes are the probabilities for the possible outcomes. The numbers in the boxes are the rewards for the corresponding situation. The best policy in the situation of the example is marked by a thick line.

The reward function we used here is a rather simple one. We took into account the velocity, position and distance of the ball towards the opponent goal, giving a high positive reward for position in front of the opponent goal and high negative reward in front of the own goal.

The time needed to evaluate the plans depends on the number of choices the agent has to take into account. In the presented example of the multi-agent plan presented above the agent spent much more time for calculating the policy. The computation times spent for deliberating differed on wheter the robot was in ball possession or not. The minimum time the robot needed to decide what to do without the ball amounts to 0.01 seconds, the maximal time about 0.450 seconds. On average the robot spent 0.094 seconds computing the policy in 694 examples. With ball the minimal time was 0.170, the maximal 2.11, on average 0.536 seconds, taking 117 examples into account.

The hardware used was an on-board Pentium III-933. With the described decision making method we shot 13 goals at the world championships and missed the final round in both world championships. At the recent GermanOpen 2004 Championships we were able to advance to the quarter finals.

6 Related Work

Approaches to high-level robot programming can roughly be classified into those making use of some form of deliberation and those which do not. These systems are embedded into system architectures ranging from reactive to deliberative. Examples for *Reactive architectures* following Brooks [8] are the Situated-Agents [2], the Dual-Dynamics approach by Jäger and Christaller [22], or UML-Statecharts [3]. These kinds of systems are based on an immediate assignment between perception and action without an explicit description of how a goal can be reached [27]. *BDI architectures* are based on the work of Bratman on practical reasoning [7]. Following Bratman the internal state of an agent is determined by its knowledge about the environment (beliefs), the action facilities the agent is able to choose from (desires) and the current goals (intentions). Representatives for this approach are PRS by Georgeff and Lansky [17] and the recent Double-Pass Architecture based on mental models [9]. *Logic-based architectures* try to obtain a goal-directed plan (sequence of actions) by using a symbolic description and a theorem prover. This manipulation of symbolic data is also called deliberation. One of the most influencing approaches is McCarthy's Situation Calculus [28]. Because deliberation is a complex, time consuming process, an optimal plan is only obtained for the situation where planning started, but not necessarily for the current situation. Another system based on Plan Description Languages is introduced in [21]. Systems following the *hybrid approach* try to combine the advantages of reactive and goal-directed aspects of other architectures. Layers found in most of these models are a reactive layer, a deliberative layer and a modeling layer. Due to using separate modules a coherence problem arises, i.e. there has to be a module making them work together reasonably. Examples are Touringmachines [15] or InterRAP [30]. Our own DR-Architecture [13] belongs to this class.

Examples are [4, 12, 23].

In [4] Beetz describes *structured reactive controllers* based on the language RPL [29]. It supports to synchronize concurrent reactive behavior with high-level control programs and gains adaptivity by applying plan revision techniques. It was tested and applied for office delivery tasks in indoor environments. Another example where reactivity was successfully combined with deliberation is described in the WITAS project [12]. Here the control rests strictly with the reactive component, which is understandable given the unmanned aerial vehicle scenario. The deliberative component, which includes a planner, among other things, is only activated on demand when the reactive controller cannot achieve a goal. This is different from our architecture, since we never rely on the deliberative component producing a plan, mainly because the environment of a soccer playing robot requires constant vigilance on the part of the robot. Jensen and Veloso describe an hybrid approach in [23], an example from the Robocup domain. Here, the simulation-league soccer agents also mix reactive and deliberative decision making. Among other things, the authors propose that an agent switches from deliberation to reactive control when an opponent moves too close to the agent. This fits well with our notion of dropping the deliberative plan once the world changes too much compared to the world model used by Golog. Despite these similarities, there are significant differences as well. For one, Jensen and Veloso use Prodigy [36], a nonlinear planner, which runs as a central deliberative service and which derives a multi-agent plan for the whole team and then sends each agent its corresponding sub-plan. To make this work, severe restrictions in the expressiveness of the plan language are necessary. For example, it is assumed that every action takes the same unit of time, which seems to limit the usefulness of the plans derived. Besides, employing a full-scale planner like Prodigy imposes a heavy burden computationally.

There are some approaches which also integrate decision-theoretic planning into their language. Poole [31] incorporates a form of decision-theoretic planning into his independent choice logic distinguishing also between active and passive sensing. This issue is, though, not regarded in the context of decision-theoretic planning. Other action logics addressing uncertainty include [34], where abduction is the focus, and [20], which addresses symbolic dynamic programming and which itself is based on [5]. Finally, [25] also discuss ways of replacing sensing by models of the environment during deliberation.

7 Conclusion

In this paper we gave a brief overview of our efforts in developing Readylog, a dialect of Golog suitable for the high-level decision making component of robotic soccer agents. The highly dynamic nature of this application and a high degree of uncertainty are among the main challenges we had to face. While this work is still ongoing we feel that we already succeeded in showing that logic-based reasoning methods can play an important role even in time-critical environments like soccer.

Acknowledgments
This work was supported by the German National Science Foundation (DFG) in the Priority Program 1125, *Cooperating Teams of Mobile Robots in Dynamic Environments* and by a grant of the Ministry of Science and Research of North Rhine-Westphalia.

References

[1] The Robocup Federation. http://www.robocup.org.

[2] P. E. Agre and D. Chapman. What are plans for? In P. Maes, editor, *Designing Autonomous Agents*, pages 17–34. MIT Press, 1990.

[3] T. Arai and F. Stolzenburg. Multiagent systems specification by UML statecharts aiming at intelligent manufacturing. In *Proceedings of the 1st International Joint Conference on Autonomous Agents & Multi-Agent Systems*, pages 11–18. ACM Press, 2002. Volume 1.

[4] M. Beetz. Structured reactive controllers. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 228–235. ACM Press, 1999.

[5] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDPs. In *IJCAI*, pages 690–700, 2001.

[6] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *Proc. of AAAI-00*, pages 355–362. AAAI Press, 2000.

[7] M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press, 1987.

[8] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, pages 14–23, 1986.

[9] Hans-Dieter Burkhard. Mental models for robot control. Dagstuhl Workshop on Plan-based Control of Robotic Agents, 2001.

[10] G. De Giacomo, Y. Lésperance, and H. J. Levesque. ConGolog, A concurrent programming language based on situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

[11] G. De Giacomo and H. Levesque. An incremental interpreter for high-level programs with sensing. In H. J. Levesque and F. Pirri, editors, *Logical Foundation for Cognitive Agents: Contributions in Honor of Ray Reiter*, pages 86–102. Springer, 1999.

[12] P. Doherty, G. Granlund, K. Kuchcinski, E. Sandewall, K. Nordberg, E. Skarman, and J. Wiklund. The witas unmanned aerial vehicle project. In *ECAI'2000*, 2000.

[13] F. Dylla, A. Ferrein, and G. Lakemeyer. Acting and Deliberating using Golog in Robotic Soccer – A Hybrid Approach. In *Proc. 3rd International Cognitive Robotics Workshop*. AAAI Press, 2002.

[14] F. Dylla, A. Ferrein, and G. Lakemeyer. Specifying multirobot coordination in ICPGolog – from simulation towards real robots. In *Proc. of the Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments: World modeling, planning, learning, and communicating (IJCAI 03)*, 2003.

[15] I. A. Ferguson. Integrated control and coordinated behavior: A case for agent models. In M. J. Wooldridge and N. R. Jennings, editors, *Intelligent Agents: ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, pages 203–218. Springer, 1994.

[16] A. Ferrein, C. Fritz, and G. Lakemeyer. On-line decision-theoretic golog for unpredictable domains. In S. Biundo, T. Frühwirt, and G. Palm, editors, *KI2004: Advances in Artificial Intelligence*. Springer, 2004.

[17] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *The Proceedings of AAAI-87*, pages 677–682, 1987.

[18] H. Grosskreutz. Probabilistic projection and belief update in the pgolog framework. In *Proc. 2nd Int. Cognitive Robotics Workshop*, pages pages 34–41. 2000.

[19] H. Grosskreutz and Gerhard Lakemeyer. cc-Golog – An Action Language with Continous Change. *Logic Journal of the IGPL*, 2002.

[20] A. Großmann, S. Hölldobler, and O. Skvortsova. Symbolic dynamic programming with the Fluent Calculus. In *Proc. IASTED (ACI-2002)*, pages 378–383, 2002.

[21] L. Iocchi, D. Nardi, and R. Rosati. Planning with sensing, concurrency, and exogenous events: Logical framework and implementation. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *KR2000: Principles of Knowledge Representation and Reasoning*, pages 678–689. Morgan Kaufmann, 2000.

[22] H. Jaeger and T. Christaller. Dual dynamics: Designing behavior systems for autonomous robots. *Artificial Life and Robotics*, 1998.

[23] R. Jensen and M. Veloso. Interleaving deliberative and reactive planning in dynamic multi-agent domains. In *Proceedings of the AAAI Fall Symposium on on Integrated Planning for Autonomous Agent Architectures*. AAAI Press, 1998.

[24] S. Konur, A. Ferrein, and G. Lakemeyer. Learning decision trees for action selection in soccer agents. In *Proc. of Workshop on Agents in dynamic and real-time environments*, 2004.

[25] Y. Lespérance and H.-K. Ng. Integrating planning into reactive high-level robot programs. In *Proc. 2nd Int. Cognitive Robotics Workshop*, pages 49–54, 2000.

[26] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.

[27] P. Maes. Situated agents can have goals. In Patti Maes, editor, *Designing Autonomous Agents*, pages 49–70. MIT Press, 1990.

[28] J. McCarthy. Situations, actions and causal laws. Technical report, Stanford University, 1963.

[29] D. McDermott. A reactive plan language. Research Report YALEU/DCS/RR-864, Yale University, 1991.

[30] J. P. Müller. The design of intelligent agents. In *Lecture Notes in AI*, volume 1177. Springer, 1996.

[31] D. Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56, 1997.

[32] J. Quinlan. *C4.5 Programs for Machine Learning*. Morgan Kaufmann, 1993.

[33] R. Reiter. *Knowledge in Action*. MIT Press, 2001.

[34] M. Shanahan. The event calculus explained. *Lecture Notes in Computer Science*, 1600, 1999.

[35] M. Soutchanski. An on-line decision-theoretic golog interpreter. In *Proc. IJCAI-2001*, 2001.

[36] M. Veloso, J. Carbonell, A. Pérez, D. Borrajo, E. Fink, and J. Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):81–120, 1995.

Kontakt

Knowledged-based Systems Group,
Department of Computer Science
Ahornstr. 55, D-52056 Aachen
e-mail: gerhard, ferrein, fritz@cs.rwth-aachen.de
WWW: http://www-kbsg.informatik.rwth-aachen.de/

Bild

Alexander Ferrein studied Computer Science at RWTH Aachen and entered the Knowledge-based Systems Group in 2001. Since then he works the project "Readylog: A Real-time Deliberative Component for cooperative multi-robot systems in highly dynamic domains " in the DFG Priority Program *Cooperating Teams of Mobile Robots in Dynamic Environments* and heads the "AllemaniACs" RoboCup team.

Bild

Christian Fritz studied Computer Science at RWTH Aachen and was a team member of the "AllemaniACs" RoboCup team. He joined the Knowledge-based Systems group in 2004 working also in the project "Readylog: A Real-time Deliberative Component for cooperative multi-robot systems in highly dynamic domains". Currently, he participates in the Ph.D. program of the University of Toronto, Canada.

Bild

Gerhard Lakemeyer is Associate Professor and Head of the Knowledge-Based Systems Group at RWTH Aachen. His research interests focus on Knowledge Representation and Cognitive Robotics. He has been on the program committee of numerous international conferences and he is an Associate Editor of the Journal of Artificial Intelligence Research.