Specifying Multirobot Coordination in ICPGolog — From Simulations towards Real Robots —

F. Dylla and **A. Ferrein** and **G. Lakemeyer**
Knowledge-based Systems Group, Ahornstr. 55, RWTH Aachen Germany
Phone: +49 241 80-21534, Fax: +49 241 80-22321
{dylla, ferrein, gerhard}@cs.rwth-aachen.de

Keywords: reasoning about actions and change, knowledge representation

Abstract

Deliberation in highly-dynamic domains such as robotic soccer requires a rich representation language that can deal with continuous change, uncertainty, and multiple agents, among other things. For this purpose we have developed the language ICPGOLOG, a variant of the logic-based action language GOLOG. We show how to specify plans for soccer agents such as playing a double pass in ICPGOLOG and report on experimental results in the ROBOCUP SIMULATION league. We have also recently integrated ICPGOLOG as the high-level control language into our MID-SIZE soccer team. We discuss the software architecture and some of the differences between agent modeling in the SIMULATION and MID-SIZE league.

1 Introduction

Highly dynamic real-time domains like robotic soccer place stringent requirements on the decision making process of agents. An action must be settled nearly immediately after new sensory information is received. There is little time to reason about the next action to perform. Especially in soccer, it is better to do something, even if it is not optimal, than to stay around on the field thinking about what to do. On the other hand, for clever team play, there is a need to, say, calculate the game position. Imagine you want to attack over the right-hand side of the field but the opponent defense is in good position while the left wing is open to advance. Estimating the game situation like this the agent holding the ball should play a pass back to a free midfielder that is in turn in a position to serve a left forward.

It seems that only taking the currently available game data into account, there is not enough information available for such decisions. We therefore think that besides reactivity a good decision module for a soccer agent needs a deliberative component in one form or another. By deliberation we understand the possibility to reason about actions and make plans about future actions to perform. Moreover, it is not enough to think only about one's own actions. Due to soccer being a team play it has to take possible actions of teammates into account. As in the example of the wing change, the ball holding agent should also reason about what would be the best action for the midfielder receiving the back pass to be able to estimate that the midfielder has a good chance to complete the wing change.

The logic-based programming language GOLOG [Levesque *et al.*, 1997] is an approach to reason about action effects combining explicit agent programming as in imperative programming languages with deliberation. Based on the situation calculus [McCarthy, 1963] GOLOG is able to reason about the world evolving from situation to situation. Over the past years many extensions like dealing with concurrency, exogenous and sensing actions, a continuous changing world and probabilistic projections [G. Lakemeyer, 1999; de Giacomo und H.J. Levesque, 1999; Giacomo *et al.*, 2000; Grosskreutz and Lakemeyer, 2000b; Grosskreutz, 2000; Grosskreutz and Lakemeyer, 2000a; 2001] made GOLOG into an expressive agent programming language. It was used as high-level controller in robotic's applications as a museum tour-guide [Burgard *et al.*, 1998], in computer animation [Funge, 1998], and for low-cost robots like the Lego Mindstorms [Levesque and Pagnucco, 2000].

Combining many of the above mentioned features into a single language, we propose ICPGOLOG as an agent programming language suitable for highly-dynamic multiagent domains like robotic soccer. Using a double pass as an example, we demonstrate in the paper how multirobot coordination can be achieved using ICPGOLOG. We also report on experimental results obtained in the SIMULATION league. However, our main goal is to use the language to control and coordinate the actions of real robots. To this end we have recently integrated ICPGOLOG into our MID-SIZE league robots. We discuss the software architecture and some of the differences in agent modeling in simulated versus physical robots. At the time of the workshop we hope to report on our first experiences using ICPGOLOG on real robots.

The rest of the paper is organized as follows. After briefly introducing relevant aspects of agent control languages in Section 2, we present our system architecture in Section 3 as a hybrid approach combining both the reactive and the deliberative paradigm. As the deliberative part of this architecture, ICPGOLOG is presented in Section 4. We present experimental results in Section 5. Section 6 gives a brief overview of our MID-SIZE software architecture with ICPGOLOG as the high-level controller. We further discuss some of the differences between the SIMULATION and MID-SIZE league regarding high-level control. In Section 7, we give a brief summary and an outlook on future work.

2 Related Work

Below we give a short summary of several architecture approaches applied in ROBOCUP. The classification is based on [Dorer, 1999b] and [Wooldridge, 1999].

Reactive architectures are based on an immediate assignment between perception and action without an explicit description of how a goal can be achieved [Maes, 1990]. Examples are the Subsumption-Architecture [Brooks, 1986], the Situated-Agents [Agre and Chapman, 1990], the Dual-Dynamics approach by Jäger and Christaller [Jaeger and Christaller, 1998], or UML-Statecharts [Arai and Stolzenburg, 2002]. *BDI architectures* are based on the work of Bratman on practical reasoning [Bratman, 1987]. 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 [Georgeff and Lansky, 1987] and the recent Double-Pass Architecture based on mental models [Burkhard, 2001]. *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 [McCarthy, 1963]. 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 [Iocchi *et al.*, 2000]. 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 [Ferguson, 1994] or InterRAP [Müller, 1996]. Our own DR-Architecture [Dylla *et al.*, 2002] belongs to this class. *Integrated architectures* aim not only for the combination but the integration of reactive and goal directed behavior. The Situated Automata [Kaelbling and Rosenschein, 1990] and Enhanced Behavior Networks proposed by Dorer [Dorer, 1999a] are examples of this approach.

3 The DR-Architecture

While deliberation has many advantages for decision making of an agent, it has the disadvantage of being slow compared to generating actions in a reactive fashion. In [Dylla *et al.*, 2002] we proposed a hybrid architecture which allows the combination of deliberation with reactivity. In this Section we give an overview of our architecture. This architecture is not only the basis for our SIMULATION team but for the MID-SIZE team as well.

Figure 1 shows the DR-Architecture. From the sensory input we build our world model (gray box in Figure 1). It contains data like the positions of teammates, opponents, or the ball (*Basic World Model*), classification data about the current situation, e.g. good possible pass partners (*Situation Classification*) and group resp. team-level information, e.g. the basic formation of the team (*Group- and Team-level tactics*).

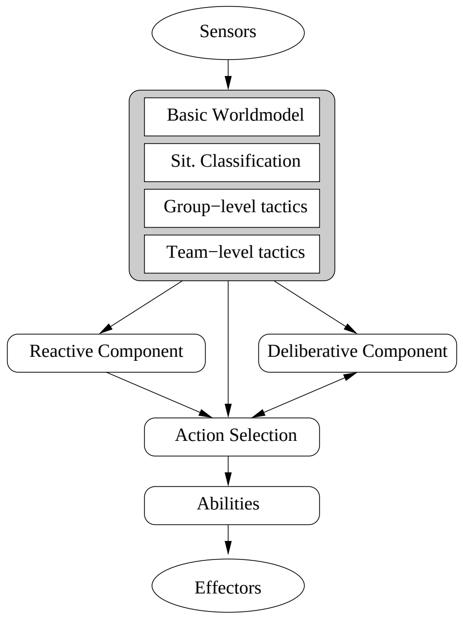


Figure 1: DR-Architecture

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. The *Deliberative Component* calculates a plan projecting into the future choosing among the possible action sequences. Section 4 covers this in detail. Having an immediate action and a plan concurring for executing one needs to decide which of both to use. A simple strategy for the *Action Selection* is to use the plan whenever there exists one or execute the action provided by the Reactive Component otherwise. More complex approaches are currently being tested in [Riedel, 2003].

Abilities are the basic actions the agent can perform in the world. In the SIMULATION league the system of UvA-TRILEARN provides actions on three layers of granularity. The fine grained layer contains actions like *kick* or *dash*, which are given by the SOCCERSERVER while the coarse model provides high-level abilities like *dribble* or *intercept ball*. For a detailed description we refer to [de Boer and Kok, 2002]. We use the UvA-TRILEARN system as our basic system.

4 From GOLOG to ICPGOLOG

GOLOG [Levesque *et al.*, 1997] is a high-level programming language for specifying complex tasks like those typically found in robotic scenarios. Similar to ordinary imperative languages, GOLOG offers constructs like *sequence*, *if-then-else*, *while*, and *recursive procedures*. In addition, a *nondeterministic choice* operator allows the robot to choose at runtime from the given alternatives. Another important difference compared to most other programming languages is the notion of a *test condition*, which, in general, can be an arbitrary first-order sentence.

We will not go into any technical details of the semantics of GOLOG except to note that it is based on the *situation calculus* [Reiter, 2001]. From a user's point of view, the following needs to be specified: a set of so-called *fluents*, which are predicates and functions that may change over time like the position of a robot, a set of *primitive actions* like moving to a certain location, *preconditions* and *effects* of primitive actions, and a (first-order) description of the initial state or situation. The user can then write programs which use the given primitive actions and where test conditions refer to fluents. We will see example programs below. Perhaps the most interesting aspect of GOLOG is the ability to project (or simulate) the outcome of a program before actually executing it. This way an agent can evaluate different alternatives and choose the best one for execution.[1]

The features offered by the original GOLOG were soon found to be insufficient for realistic robot application. The first major extension was CONGOLOG [Giacomo *et al.*, 2000], which added the ability to execute actions concurrently and the notion of an *exogenous action*, which is useful to model events outside the robot controller like a low-level routine reporting the current position of the robot. INDIGOLOG [de Giacomo und H.J. Levesque, 1999] extends CONGOLOG by allowing the interleaving of on-line execution and projection.

For robotic domains, however, at least two things were still missing, actions that describe continuous change to, say, reason about the movement of a mobile robot, and noisy sensors and effectors. These features were added in CCGOLOG [Grosskreutz and Lakemeyer, 2000b; 2001] and PGOLOG [Grosskreutz, 2000; Grosskreutz and Lakemeyer, 2000a], respectively. Among other things, these offer constructs that allow an agent to wait for a certain event to occur (*waitFor*), to execute a sub-program which can be blocked at any time once a given condition is no longer satisfied (*withCtrl*), and to execute an action with some probability p and an alternative with probability $1 - p$.

In our current work, we have integrated all these features into a single framework called ICPGOLOG, whose Prolog-implementation is based on an existing implementation of INDIGOLOG. Before turning to applying ICPGOLOG to robotic soccer, here is a brief summary of the language features and the notation used:

- Sequence: a_1, a_2
- Nondeterministic Choice: $a_1; a_2$
- Test: $?(c)$
- Event-Interrupt: $waitFor(c)$
- If-then-else: $if(c, a_1, a_2)$
- While-loops: $while(c, a_1)$
- Condition-bounded execution: $withCtrl(c, a_1)$
- Concurrent actions: $pconc(a_1, a_2)$
- Probabilistic actions: $prob(val_{prob}, a_1, a_2)$
- Probabilistic (off-line) projection: $pproj(c, a_1)$
- Procedures: $proc(name(parameters), body)$

[1]We remark that the language provides a solution to the frame problem [Reiter, 2001].

4.1 A soccer agent in ICPGOLOG

For specifying an agent in ICPGOLOG one starts with a description of the domain knowledge. One has to specify the actions the agent can perform. ICPGOLOG distinguishes between *primitive*, *sensing*, and *exogenous* actions. The primitive actions are those the agent can perform in the world, sensing actions are used to get information about the world. A fluent is assigned to each sensing action which the agent can test to get information about the world. Exogenous actions are actions which occur in the world and to which the agent can react. For each action one has to specify a precondition to state when the action is possible. To describe the effects of an action, one must provide effect axioms, describing which fluents are changed when performing this particular action. Following, we give an example of such a description for the soccer domain.

1. $prim\_fluent(seeBall)$
2. $prim\_fluent(playmode)$
3. $exog\_action(changePlaymode(PM))$
4. $poss(changePlaymode, true)$
5. $causes\_val(changePlaymode(PM),$
   $playmode, PM, legal(PM))$
6. $prim\_fluent(passPartner)$
7. $prim\_action(search\_next\_Passpartner(Player))$
8. $senses(search\_next\_Passpartner(Own),$
   $passPartner)$
9. $cont\_fluent(ballPosition)$
10. $prim\_action(directPass(Sender, Recipient))$
11. $poss(directPass(X, Y),$
    $and(ballOwner(X), seePlayer(Y)))$
12. $causes\_val(directPass(X, Y), ballOwner,$
    $Y, not(passIntercepted(X, Y)))$
13. $initial\_val(ballPosition, [0, 0])$
14. $initial\_ll(ll\_linearBallMove)$

The relational fluent $seeBall$ tells the agent whether it sees the ball or not. It is a primitive fluent in contrast to continuous fluents which will be explained below.

The fluent of Item 2 denotes the server's playmode. Every time the playmode is changed by the SOCCERSERVER an exogenous action is generated by the basic system. The agent notices a playmode change by testing the playmode fluent. The effect axiom of this action (5) changes the value of the playmode fluent when the exogenous action was generated by the basic system and received by the ICPGOLOG agent. The playmode is only changed if a legal playmode is transmitted to the agent otherwise its value remains unchanged. This is denoted by the last argument of the $causes\_val$ predicate in

(5). An example for a precondition is given in (4) stating that a playmode change can always happen.

To play a pass, an agent has to determine to which teammate it should play the pass. Next, the description for a possible pass partner follows. The fluent $passPartner$ is affected by the sensing action $search\_next\_Passpartner(Player)$.

The predicate $senses$ instructs the interpreter to set the fluent $passPartner$ to the respective value.

Fluent $ballPosition$ holds the position of the ball. It changes continuously every cycle and is therefore stored in a special continuous fluent.

In (10) the primitive action for playing a direct pass from the $Sender$ to a pass $Recipient$ is defined. The action only seems reasonable if the sender is in ball possession and knows where the recipient is (11). The action has the effect that the ballholder changes from sender to the recipient, but only if the ball was not intercepted by an opponent player.

Additionally, one has to define the fluent's initial values, e.g. the ball being in the center of the field before kick-off (12). For continuous fluents one has to specify a so-called low-level model (13). Those models are used in projections to determine the correct fluent value. For the ball position a linear approximation of the ball movement is used.

```
proc(soccer_agent,
  pconc( [ withCtrl(playmode=pm_BeforeKickOff,
                place_on_field(playmode) ),
            withCtrl(playmode=pm_Null, wait(1) ),
            withCtrl(playmode=pm_FreeKick_Left,...),
            ...
            withCtrl(playmode=pm_PlayOn,
                play_soccer) ],
  [ execute(next_action), fail].
```

Figure 2: Top-Level structure of the soccer agent

The next step is to provide ICPGOLOG programs to describe agent's behavior. The playmode is the most important parameter for determining a next action. If the mode changes the agent has to react immediately. Within the top level procedure $soccer\_agent$ (see Figure 2) this is modeled by the $withCtrl$ statement. $withCtrl(\text{playmode} = \text{pm\_BeforeKickOff}, \ldots)$ means that the agent should take its position on the field before the game is started. As long as this conditions holds, the agent will call the $place\_on\_field$ procedure. At that moment a condition fails, the respective procedure is interrupted. To be able to catch all possible playmodes we use the $pconc$ statement. By this, the different $withCtrl$ statements are executed concurrently.

```
proc(play_soccer,
  [ try_goal_shot(Own);
    ...;
    try_double_passes(Own);
    try_direct_passes(Own);
    ...        ]).
```

Figure 3: Selecting a behavior

In the case of $playmode = pm\_Play\_On$ the procedure $play\_soccer$ is called (see Figure 3). This procedure encodes the behavior of the agent in normal play. If a plan is applicable with respect to the low-level models in the situation the projection is based on, it will be selected for execution. So far, the order actions are evaluated is fixed. The first successful complex action in this order will be selected, the others below will be ignored. Methods changing this order in a reasonable way are currently under investigation.

```
proc(try_double_passes(Own),
  [ initializePassPartner,
    search_next_Passpartner(Own),
    while ( not(passPartner=nil),
      if ( pproj(has_ball(Own),
                try_double_pass(Own, passPartner)) >= 0.9,
        [ print("PLANNED DOUBLE PASS."),
          set_next_action(double_pass(Own, passPartner)),    ],
      search_next_Passpartner(Own))
    )]).
```

Figure 4: Module for evaluating several double pass opportunities

In Figure 4 we display the procedure for evaluating double pass possibilities with several pass partners. The agent projects a double pass with a specific partner by $pproj$ and checks whether the projected probability is higher than 90%. In this case the double pass with the computed pass partner is selected and executed otherwise a next pass partner is tried. To coordinate both agents, the pass partner uses the same plan descriptions and projects from the ballholder's view.

```
proc(try_double_pass(Own, TargetPlayer),
  [ look_for_free_space(Own, TargetPlayer),
    directPass(Own, TargetPlayer, pass_NORMAL),
    pconc( [ pconc( receivePass(TargetPlayer),
             intercept_direct_pass(closestOppToPass(TargetPlayer),
                    TargetPlayer)),
               if( isBallKickable(TargetPlayer),
                 [ kickTo(TargetPlayer, freePos, 0.8),
                   intercept_direct_pass(closestOppToPass(Own),
                     Own)])],
             [ moveToPos(Own, freePos),
               receivePass(Own)])]).
```

Figure 5: Module for projecting one double pass taking opponents into account.

The description of a double pass with a specific teammate is shown in Figure 5. In a double pass situation an opponent is staying right before our agent. To determine a good position where to receive the pass back from the teammate the agents look for a free space behind the opponent. The action $look\_for\_free\_space$ does right this setting the fluent $freePos$ to a particular position in the free region. After that the agent passes the ball to its partner it starts to run towards the free position. Meanwhile, the ball is moving towards his teammate (modeled by $pconc$). It is expected that the closest opponent tries to intercept the moving ball ($intercept\_direct\_pass$). When the second player receives the ball, it kicks it back to the position where the initiator waits for the ball. The opponent's attempt to intercept the ball is modeled for this pass as well.

```
proc(execute_double_pass(Own, TargetPlayer),
  [ look_for_free_space(Own, TargetPlayer),
    directPass(Own, passPartner, pass_NORMAL),
    receivePass(passPartner),
    kickTo(passPartner, freePos, 0.8),
    moveToPos(Own, freePos),
    receivePass(Own),
    setPassFinished,
    setTrySucceeded(true) ].
```

Figure 6: Module for executing a double pass

The execution of the projected path is shown in Figure 6. The ballholder has to look again where his teammate and the free space is, because the world might have changed. The actions with argument Own are executed by the agent itself while the actions with $passPartner$ as argument are supposed to be executed by the respective teammate.

To illustrate two more important features we show in Figure 7 two procedures for projecting a direct pass with two possibilities to model opponent behavior. By the $waitFor$ construct in Figure 7, the agent waits until it is able to stop the ball or the ball is too far away, probably it was intercepted and the current action is not reasonable anymore. With this construct we can test the end of the pass action. Another concept is the $prob$ statement. We can test two different models of an opponent intercepting our pass. With probability 0.7 the pass may be intercepted using model $intercept\_direct\_pass1$ and with probability 0.3 using $intercept\_direct\_pass2$. In the first alternative the model can be very pessimistic, i.e. the opponent will perform a real sophisticated intercept action, or optimistic in the other case.

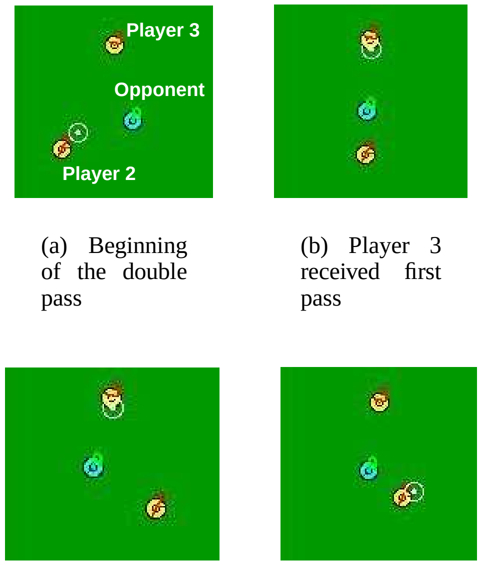
4.2 Executing a Double Pass

To illustrate how ICPGOLOG works in practice we give an example execution of a double pass. We refer to the procedures given above. We first introduce the example setting and show how a multiagent plan is generated and executed in ICPGOLOG by an execution trace of the program *try_double_passes*.

Our scenario is the following. Player 2 (the lower yellow player in Figure 8) wants to outplay the opponent with a double pass. Player 3 (upper yellow player in Figure 8) is in a good position to play a double pass with Player 2. Player 2

```
proc(try_direct_pass(Own, TargetPlayer),
  [ directPass(Own, TargetPlayer, pass_NORMAL),
    pconc(prob_intercept_direct_pass(closestOpp-
          ToPass(TargetPlayer), TargetPlayer),
    waitFor(or(ball_near_player(TargetPlayer),
                 ball_far(TargetPlayer))))]).
proc(prob_intercept_direct_pass(Opp, TargetPlayer),
    prob(0.7, intercept_direct_pass1(Opp, TargetPlayer),
    intercept_direct_pass2(Opp, TargetPlayer))).
```

Figure 7: Module for projecting a direct pass with probabilistic opponent behavior



(a) Beginning of the double pass

(b) Player 3 received first pass

(c) Player 2 moves to receive position

(d) Player 2 receives the second pass

Figure 8: Double pass scenario

therefore initiates the double pass by playing a direct pass to Player 3. Thereafter, Player 2 has to run to the position where it can receive the pass from Player 3 (Figure 8(b)). Player 3 receives the ball and should pass it back to Player 2 if Player 2 itself is near the reception position (Figure 8(c)). Finally, Player 2 receives the ball (Figure 8(d)).

To make it more concrete, we show the ICPGOLOG execution trace of the described scenario in Figure 9. Although we are able to reason about the behaviors of opponents by appropriate models as well, we leave out this detail here. The left column of this figure shows the trace for Player 2 which initiates the pass, the right one for Player 3. Player 2 starts by getting a new world model and intercepting the ball to be able to play the first pass (lines 1 – 6). After the successful intercept action both agents start the procedure $try\_double\_passes(Own)$ (refer to Figure 4). The variable Own is set to Player 2 for both agents. Player 3 therefore plans all actions of the double pass from Player 2's point of view. Of course, in the execution each agent performs only actions regarding itself.

The first action in this procedure is to find a pass partner. After resetting the fluent $passPartner$, the agents projects all possible partners for playing a pass. This is expressed by the $pproj$ statement in Figure 7. This corresponds to the lines 7 to 13 in Figure 9. If this projection is successful with probability 0.9 the procedure $execute\_double\_pass$ is called. As one can observe in lines 14 and 15 in Figure 9, both players are executing the action `directPass`. The execution system can determine by the command `nextSkill(2)` that this action is for Player 2. Player 3 will not perform the action in the real world.

```
            Player 2                                Player 3

send(getBasicWorldModel, true)
send(nextSkill(2), intercept)
WAITING FOR EXOGENOUS ACTIONS...
setPlayerProj(2,[-28.34,-2.33],...)
waitedIntercept
send(getBasicWorldModel, true)          send(getBasicWorldModel, true)
initializePassPartner                   initializePassPartner
setPassPartner(3)                       setPassPartner(3)
Prob. Proj. Test                        Prob. Proj. Test
 (# of initial configs: 1,                (# of initial configs: 1,
 unsorted/sorted # of traces:1/1).        unsorted/sorted # of traces:1/1).
Prob. Proj. Test (cached result).
write(PLANNED DOUBLE PASS.)             write(PLANNED DOUBLE PASS.)
send(nextSkill(2),                      send(nextSkill(2),
  [directPass, [3, pass_NORMAL]])         [directPass, [3, pass_NORMAL]])
setBallProj([-27.50,-3.88],...)         setBallProj([-28.50, -3.00], ...)
send(nextSkill(3), receivePass)         send(nextSkill(3), receivePass)
                                        WAITING FOR EXOGENOUS ACTIONS...
setBallProj([-23.27,-11.44],...)        setBallProj([-23.19, -11.43],...)
send(nextSkill(3),                      send(nextSkill(3),
  [kickTo, [[-18.71,-1.88],0.4])          [kickTo, [[-21.42, 0.98],0.4])
                                        WAITING FOR EXOGENOUS ACTIONS...
setBallProj([-27.50,-3.88],...)         setBallProj([-23.26, -8.75], ...)
send(nextSkill(2),                      send(nextSkill(2),
  [moveToPos,[[-18.71,-1.88],...])        [moveToPos,[[-21.42,0.98],...])
WAITING FOR EXOGENOUS ACTIONS...
setPlayerProj(2,[-18.71,-1.88],...)     setPlayerProj(2,[-21.42,0.98],...)
send(nextSkill(2), receivePass)         send(nextSkill(2), receivePass)
WAITING FOR EXOGENOUS ACTIONS...
setBallProj([-20.57,3.19],...)          setBallProj([-3.37, 2.97],...)
send(getBasicWorldModel, true)          send(getBasicWorldModel, true)
send(nextSkill(2), intercept)           send(nextSkill(2), intercept)
setPlayerProj(2,[-20.96,3.47],...)      setPlayerProj(2,[-23.30,-6.25],...)
waitedIntercept                         waitedIntercept
setPassFinished                         setPassFinished
setTrySucceeded(true)                   setTrySucceeded(true)
send(nextSkill(2), intercept)
WAITING FOR EXOGENOUS ACTIONS...
setPlayerProj(2,[-21.01,3.62],...)
waitedIntercept
```

Figure 9: Execution traces of the pass sender and receiver in the double pass situation

We now enter phase 2 of our double pass (Figure 8(b)) where the first pass is to be received by Player 3. Again, both player settle the same action (`receivePass`). To synchronize actions of both players the execution system waits until some condition meets denoting the end of the respective action. In our example the reception of the first pass is acknowledged by an exogenous event "*received pass*".

This is modeled by an exogenous action (line 18 in Figure 9, `WAITING FOR EXOGENOUS ACTION`). The pass back from Player 3 to Player 2 is not modeled by a direct pass. Instead, a $kickTo$ action is performed to a position calculated by $look\_for\_free\_space$ in Figure 6, i.e. a position in a free region behind the opponent. Note that the goal position of the $kickTo$ command slightly differs in both traces. This can be explained by the different world models of the resp. agent based on which this calculation is done.

Figure 8(c) shows the situation when Player 2 is near the calculated receive position. Finally, Player 2 receives the pass (Figure 8(d)). As stated above, there can be small differences in values derived from agent's world model. Therefore, to ensure that Player 2 receives the ball, it performs an intercept action in the end.

In this example we show a multiagent plan for a double pass. This plan does not use explicit communication to coordinate the agents involved. The execution of this plan is possible because both player reason about the same actions. Player 3 in the example generates the plan from the ballholder's point of view and comes to the same conclusion as Player 2. So, Player 3 identifies itself to be the best pass partner for Player 2. Multiagent coordination like this only works if agents' world models are similar and not too uncertain.

For our implementation we used the PROLOG system ECLIPSE [ECLiPSe, 2002] for the ICPGOLOG interpreter. The primitive ICPGOLOG actions we used in the double pass example are sent to the basic agent that is connected to the SOCCERSERVER. Our basic agent is an extension of the UvA-TRILEARN system from the University of Amsterdam [de Boer and Kok, 2002]. The PROLOG system communicates with the basic agent via shared memory. The high-level agent receives its world model from the basic agent and sends the next action to be executed to it which in turn translates it to SOCCERSERVER commands.

Finally, we remark that for our implementation of ICPGOLOG, we needed to solve a problem that is common to most GOLOG variants, but which becomes quite critical in real-time environments like robotic soccer. The issue is that in order to evaluate a test condition, the GOLOG interpreter usually applies what is called *regression*, which means, roughly, that the interpreter needs to transform the test condition to one that can be evaluated in the initial situation, taking into account *all* the actions which have happened so far. In our case the number of actions to be considered in this process quickly grows into the thousands, which leads to severe computational problems. As an alternative, Lin and Reiter [Lin and Reiter, 1997] proposed what they call *progression*, which means that after an action has been executed, the description of the initial situation is updated to reflect the effects of the action. This way tests can always be evaluated directly against the current state of the world. Unfortunately, Lin and Reiter's proposal does not lend itself to an efficient implementation because the size of the world description easily grows exponentially when applying progression. For our domain we developed a simpler, set-based form of progression which is more restricted than Lin and Reiter's version but which is computationally viable. In fact we were able to gain an exponential speedup compared to using regression. For details we refer the reader to [Dylla *et al.*, 2003].

5 Experimental Results

We tested the above described framework in the scenario of ROBOCUP SIMULATION league. Our agent programming language is expressive at the cost of higher runtimes. We therefore tested on which level of abstraction deliberation is reasonable. Three models differing in their level of granularity were implemented. The classification is roughly adopted from the UvA-TRILEARN system. Fine granular actions are the basic actions provided by the SOCCERSERVER, e.g. $kick$, $dash$ or $turn$ valid for one cycle each. The medium granular model contains actions like $dashToPoint$ and the coarse consists of action on the level $interceptBall$ or $directPass$ continuing for several server cycles. In all tests the task was to project and execute an action. The results are based on tests where only few agents connected to the SOCCERSERVER, not two whole teams.

The $GoalShot$ tests a shot in each corner of the goal, while concurrently simulating the behavior of the goalie and the closest opponent to the ball's trajectory. The $DirectPass$ procedure models a direct pass to two different teammates with an opponent trying to reach the pass-way. The test of more pass options leads to a linear runtime increase per additional teammate. Within $Doublepass$ two different possibilities of playing a double pass are tested with the opponent closest to the pass-way trying to intercept the ball. For coordination purposes the teammate calculates his behavior by putting himself in the ballholder's place. At last the $DirectPassWithPO$ models the same as $DirectPass$ with the difference the opponent having four probabilistic choices of how to behave. Some test results are shown in Figure 10. For a complete overview we refer to [Jansen, 2002; Dylla *et al.*, 2003].

The runtime results in Figure 10 suggest that the use of ICPGOLOG's projection mechanism is currently only feasible computationally at a high level of abstraction like in the coarse model. Here, the use of continuous fluents for projecting the ball position, for example, brings runtime advantages over the fine grained model, where those continuous fluents could not be used.

6 From Simulations towards Real Robots

While the SIMULATION league certainly provides a rich environment to test ideas in multiagent coordination, our main goal is to apply them to real robots. For that purpose we recently acquired a team of robots for the MID-SIZE league. Given the experience of other teams that "off-the-shelf" robots must be completely reengineered for ROBOCUP's MID-SIZE to be competitive, we decided to develop our own robotic platform from scratch [Wunderlich and

GoalShot	fine	middle	coarse
average no of actions / plan	101	80	38
time [s]	1.91	1.12	0.31

DirectPass	fine	middle	coarse
average no of actions / plan	93	40	39
time [s]	1.8	0.48	0.25

Doublepass	fine	middle	coarse
average no of actions / plan	319	319	86
time [s]	6.16	5.8	0.69

DirectPassWithPO	fine	middle	coarse
average no of actions / plan	95	42	41
time [s]	7.22	2.95	0.67

Figure 10: Runtime results in the SIMULATION-League

Dylla, 2002]. The intention was to develop robots competitive in ROBOCUP which can also be used in office domains for service-robot applications. The platform has a size of 39 cm × 39 cm × 40 cm (Figure 11). For power supply we have two 12 V lead-gel accumulators with 15 Ah each onboard. The battery power lasts for approximately one hour at full charge. The robot has a differential drive, the motors have a total power of 2.4 kW. This power provides us with a top speed of 3 m/s and 1000°/s by a total weight of approximately 50 kg.

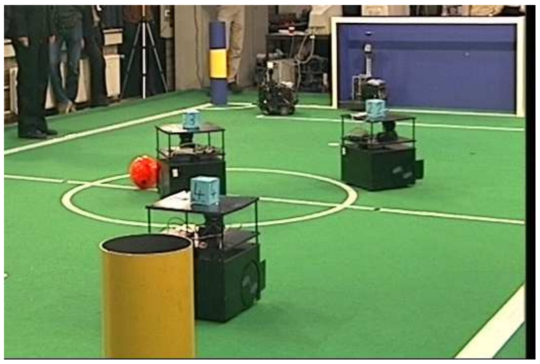


Figure 11: Robots of the AllemaniACs MID-SIZE team

Onboard we have two Pentium III PC's at 500 MHz running Linux, one equipped with a framegrabber for a Sony EVI-D100P camera mounted on a pan/tilt unit. Our other sensor is a 360° laser range finder with a resolution of 0.75 degree at a frequency of 20 Hz. For communication a WLAN adapter based on IEEE 802.11b is installed.

Figure 12 gives a schematic overview of the different software components and the data and command flow between them. The modules *motor*, *kicker* and the *camera's pan/tilt* are drivers for the actuators, *laser* and *camera* are driver modules for the sensors. The collision avoidance module *colli* uses the A^* algorithm to compute a collision free path to a

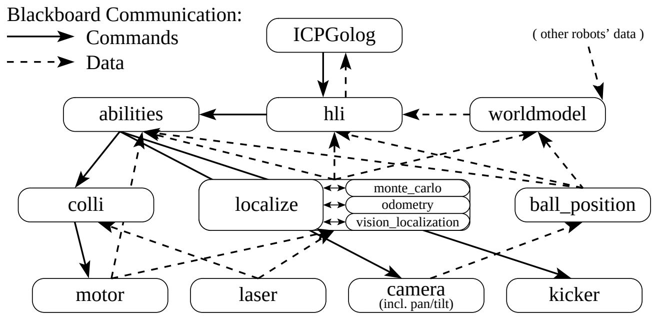


Figure 12: Components of our software system

target point every 50 ms. Localization is based on the fusion of several methods. The module is dominated by a Monte Carlo approach using the laser data. Due to complexity of the Monte Carlo approach update frequencies are lower for the localization. It turned out that a frequency of 2 up to 4 is sufficient for good localization. Between position updates of *localize* the robot uses it's odometry for position calculations. Additionally, information extracted from a vision algorithm using probability maps [Jones and Rehg, 1999] is used for resolving field symmetries. The ball is perceived by the *ball_position* module. The *ability* module defines different basic actions the robot can perform. Abilities such as *goto* and *dribble* send respective commands to the collision avoidance module which in turn actuates the motor, camera and kicker.

Our world model basically holds (quantitative) information about the positions of the robots and the ball. Moreover, information about the game state like the robot's role in the play or the current play mode is represented as well. Our interface to the high-level control is encoded in the module *hli* (high-level interface) This interface is a wrapper program between the PROLOG system of our high-level control and the basic robot system which is implemented in C++. For interprocess communication we use a blackboard communicating via shared memory. For inter-robot communication UDP is used.

To ensure safe communication via UDP the blackboard system has its own security layer. This feature offers the possibility to implement a global world model. Figure 13 shows the communication infrastructure of inter-robot data exchange. An external data synchronization module (*sync*) receives local world information from each robot and transmits them to a global blackboard to which the global world model is connected. Each robot in turn receives the global world model data in the opposite direction. To control the local software modules, *rccc* (*r*obo*c*up *c*ontrol *c*enter) was implemented. It offers the possibility to start the software on each robot by a different communication channel using remote procedure calls.

The possibilities to model robot behavior are much more restricted compared to SIMULATION league. One obvious reason is that the world model the robot can rely on is much more uncertain than in SIMULATION league, e.g. regarding one's own position or the ball position. Moreover, it is harder to get informations about, say, opponent positions. Another

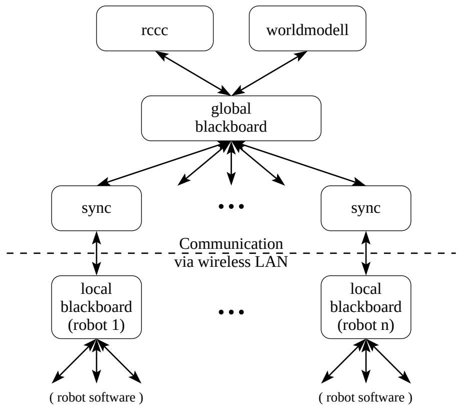

rccc
worldmodell
global blackboard
sync
sync
Communication via wireless LAN
local blackboard (robot 1)
local blackboard (robot n)
(robot software)
(robot software)

Figure 13: Global connectivity of the robots' local software systems

problem is related to the actuators. While in SIMULATION league it is rather simple to kick the ball or to play a pass it is much more complicated in MID-SIZE league. On the other hand, the coordination of the team should be easier than in the SIMULATION league since only four robots have to be coordinated. For a pass only two teammates have to be considered as possible pass receivers whereas in SIMULATION this number is normally much higher. The robots, in general, have more time to "think" about what action to perform because there is not a strict decision cycle as in SIMULATION league. Thus, the calculation times shown in Figure 10 seem not to be too problematic. Another difference between the SIMULATION and MID-SIZE league concerns communication. In the MID-SIZE league no restriction of what amount of data may be communicated between the robots via wireless LAN exists whereas it is limited to a few bytes in the SIMULATION league. Therefore communication may be used for more reliable coordination of the robots.

7 Discussion

We introduced the framework of ICPGOLOG for developing deliberative agents. By integrating features like concurrency, exogenous actions, continuous change and the possibility to project into the future we are able to model agents for highly-dynamic environments like robotic soccer. We showed how multiagent coordination can be achieved without communication. The requirement for coordination without communication is that the world models of the agents do not differ too much. In the tested cases of the SIMULATION league the agents' world models are not too uncertain due to the good sensory information given by the SOCCERSERVER. Therefore, this kind of multiagent coordination works well. In the MID-SIZE league, where world models are not that elaborate and more uncertain due to sensor noise, this approach to multiagent coordination might be problematic. It is likely that communication between robots will play a much bigger role here. It remains to be seen how much communication is necessary for coordinated actions of real robots. However, it does not necessarily only get harder when moving from simulations to real robots. For example, in the MID-SIZE league the real-time requirement for deciding on the next action is not as strict as in the SIMULATION league. Therefore, the robot has more time for generating plans with ICPGOLOG. Moreover, fewer agents have to be coordinated. We are currently evaluating and testing what works best for our MID-SIZE robots and hope to report on our experiences at the time of the workshop. Ultimately we also want to get deeper understanding of the similarities and differences between the SIMULATION and MID-SIZE leagues.

We end the paper with remarks on two other research issues currently under investigation. The first concerns the action selection mechanism. Having two decision components (reactive and deliberative) competing for the next action to be executed we have to select which action to use. This affects also the problem of the validity of ICPGOLOG plans over time. Once a plan is generated, it will be executed for a number of cycles. The action selection also has to check if the world evolved as anticipated in the plan. In [Riedel, 2003] those questions are investigated. Again it is likely that there will be significant differences between simulated and real soccer agents.

The second is about how to select optimal actions. In this regard Boutilier et al. [Boutilier *et al.*, 2000a] recently proposed a decision-theoretic GOLOG dialect based on Markov Decision Processes. We plan to integrate decision-theoretic concepts into our ICPGOLOG framework as well. To this end we are currently working on speeding up the computation of optimal policies in decision theoretic GOLOG [Ferrein *et al.*, 2003] by incorporating the notion of macro actions considered in the MDP literature [Boutilier *et al.*, 2000b; Hauskrecht *et al.*, 1998; Sutton *et al.*, 1999].

References

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

[Arai and Stolzenburg, 2002] Toshiaki Arai and Frieder 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, Bologna, Italy, 2002. ACM Press. Volume 1.

[Boutilier *et al.*, 2000a] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *AAAI'2000*, 2000.

[Boutilier *et al.*, 2000b] C. Boutilier, R. Reiter, M. Soutchanski, and S. Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *Proc. AAAI-2000*, 2000.

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

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

[Burgard *et al.*, 1998] W. Burgard, A.B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. In *Proceedings of the AAAI 15th National Conference on Artificial Intelligence*, 1998.

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

[de Boer and Kok, 2002] Remco de Boer and Jelle Kok. The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team. Master's thesis, University of Amsterdam, Februar 2002.

[de Giacomo und H.J. Levesque, 1999] G. de Giacomo und H.J. Levesque. An incremental interpreter for high-level programs with sensing. In F. Pirri H. Levesque, editor, *Logical Foundations for Cognitive Agents*, pages 86–102. Springer, 1999.

[Dorer, 1999a] Klaus Dorer. Behavior networks for continuous domains using situation-dependent motivations. In Dean Thomas, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol2)*, pages 1233–1238, S.F., July 31–August 6 1999. Morgan Kaufmann Publishers.

[Dorer, 1999b] Klaus Dorer. *Motivation, Handlungskontrolle und Zielmanagement in autonomen Agenten*. PhD thesis, Albert-Ludwigs-Universität Freiburg, Freiburg, December 1999.

[Dylla *et al.*, 2002] 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 (CogRob 2002)*. AAAI Press, 2002.

[Dylla *et al.*, 2003] F. Dylla, A. Ferrein, N. Jansen, and G. Lakemeyer. Progression in the Framework of GOLOG. KBSG, Aachen University, in preparation, 2003.

[ECLiPSe, 2002] ECLiPSe. (Version 5.5) - The ECRC Constraint Logic Parallel System. http://www.icparc.ic.ac.uk/eclipse, 2002.

[Ferguson, 1994] 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, Berlin,, 1994.

[Ferrein *et al.*, 2003] A. Ferrein, C. Fritz, and G. Lakemeyer. Extending DTGolog with options. In *Proc. IJCAI-03*, 2003. to appear.

[Funge, 1998] J. Funge. *Making Them Behave: Cognitive Models for Computer Animation*. PhD thesis, University of Toronto, Toronto, Canada, 1998.

[G. Lakemeyer, 1999] G. Lakemeyer. On sensing and off-line interpreting in golog. In H. Levesque und F. Pirri, editor, *Logical Foundations for Cognitive Agents*. Springer, 1999.

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

[Giacomo *et al.*, 2000] Giuseppe De Giacomo, Yves Lésperance, and Hector J. Levesque. ConGolog, A concurrent programming language based on situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

[Grosskreutz and Lakemeyer, 2000a] H. Grosskreutz and G. Lakemeyer. Turning high-level plans into robot programs in uncertain domains. In *ECAI'2000*, 2000.

[Grosskreutz and Lakemeyer, 2000b] Henrik Grosskreutz and Gerhard Lakemeyer. cc-Golog: Towards more realistic logic-based robot controllers. In *Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00)*, pages 476–482, Menlo Park, CA, 2000. AAAI Press.

[Grosskreutz and Lakemeyer, 2001] H. Grosskreutz and G. Lakemeyer. Online-Execution of ccGolog Plans. In *IJCAI'2001*, 2001.

[Grosskreutz, 2000] H. Grosskreutz. Probabilistic projection and belief update in the pGolog framework. In *Second International Cognitive Robotics Workshop*, 2000.

[Hauskrecht *et al.*, 1998] M. Hauskrecht, N. Meuleau, L. Kaelbling, T. Dean, and C. Boutilier. Hierarchical solutions of MDPs using macro-actions. In *Proc. UAI 98*, 1998.

[Iocchi *et al.*, 2000] Luca Iocchi, Daniele Nardi, and Riccardo 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, San Francisco, 2000. Morgan Kaufmann.

[Jaeger and Christaller, 1998] H. Jaeger and T. Christaller. Dual dynamics: Designing behavior systems for autonomous robots, 1998.

[Jansen, 2002] Norman Jansen. A framework for deliberation in uncertain, highly dynamic environments with real-time requirements. Master Thesis, in German, Knowledge Based Systems Group, Aachen University, Aachen, Germany, 2002.

[Jones and Rehg, 1999] M. J. Jones and J. M. Rehg. Statistical color models with application to skin detection. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, volume 1, pages 274–280, 1999.

[Kaelbling and Rosenschein, 1990] L. P. Kaelbling and S. J. Rosenschein. Action and planning in embedded agents. In Pattie Maes, editor, *Designing Autonomous Agents*, pages 35–48. MIT Press, Cambridge (MA), 1990.

[Levesque and Pagnucco, 2000] H. Levesque and M. Pagnucco. Legolog: Inexpensive experiments in cognitive robotics. In *Proc. 2nd International Cognitive Robotics Workshop (CogRob-00)*. ECAI-00, 2000.

[Levesque *et al.*, 1997] Hector J. Levesque, Raymond Reiter, Yves Lesperance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.

[Lin and Reiter, 1997] F. Lin and R. Reiter. How to progress a database. *Artificial Intelligence*, 92:131–167, 1997.

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

[McCarthy, 1963] J. McCarthy. Situations, actions and causal laws. Technical report, Stanford University. Reprinted 1968 in Semantic Information Processing (M.Minsky ed.), MIT Press, 1963.

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

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

[Riedel, 2003] B. Riedel. Developing similarity measures for the comparison of game situations in Robocup. Master Theses, in progress, in German, Knowledge Based Systems Group, Aachen University, Germany, 2003.

[Sutton *et al.*, 1999] R. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Journal of Artificial Intelligence*, 1999.

[Wooldridge, 1999] Michael Wooldridge. Intelligent agents. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 27–78. The MIT Press, Cambridge, MA, USA, 1999.

[Wunderlich and Dylla, 2002] Jost Wunderlich and Frank Dylla. Technical description of the allemaniacs soccer robots. Technical report, LTI / KBSG, Aachen University, Germany, 2002. in German.