Spatial Agents Implemented in a Logical Expressible Language

Frieder Stolzenburg, Oliver Obst, Jan Murray, Björn Bremer

Universität Koblenz-Landau, Fachbereich Informatik
Rheinau 1, D–56075 Koblenz, GERMANY
{stolzen,frvit,murray,moddy}@uni-koblenz.de

Abstract. In this paper, we present a multi-layered architecture for spatial agents. The focus is laid on the declarativity of the approach, which makes agent scripts expressive and well understandable. They can be realized as (constraint) logic programs. The logical description language is able to express actions or plans for one and more autonomous and cooperating agents for the RoboCup (Simulator League). The system architecture hosts constraint technology for qualitative spatial reasoning, but quantitative data is taken into account, too.
The basic (hardware) layer processes the agent's sensor information. An interface transfers this low-level data into a logical representation. It provides facilities to access the preprocessed data and supplies several basic skills. The second layer performs (qualitative) spatial reasoning. On top of this, the third layer enables more complex skills such as passing, offside-detection etc. At last, the fourth layer establishes acting as a team both by emergent and explicit cooperation. Logic and deduction provide a clean means to specify and also to implement teamwork behavior.

1 Introduction

Naturally, tasks to be solved by a team of autonomous agents are many-sided and complex. In order to achieve a goal, a single agent has to use a set of complementary subtasks. On the one hand, some of these actions can be performed in a purely reactive manner, meeting real-time requirements. On the other hand, tasks may require a certain amount of planning and reasoning. So, we were led to the idea of combining both the advantages of procedural and logic programming and decided on a hybrid system with a layered architecture.

1.1 Implementing Agents in Logic

In contrast to other approaches that provide an architecture for (multi-)agent systems (see e.g. [16, 24]), we use different logical and deductive formalisms not only as a specification language but also as an implementation language. Widespread in this context is the use of a Belief-Desire-Intention (BDI) architecture (see e.g. [7]), which has been originally specified by means of modal logics. A first-order axiomatization has been proposed for this kind of architecture only recently [24]. However, it seems that it is not actually used as implementation language there.

M. Veloso, E. Pagello, and H. Kitano (Eds.): RoboCup-99, LNAI 1856, pp. 481−494, 2000.

We will now describe our system architecture and show how different deductive processes—including constraint solving—can be used for the RoboCup [20]. The system combines the BDI approach with a multi-layered architecture, allowing multiple agents to perform collective actions. Nevertheless, each agent is autonomous and can be implemented in a manner similar to (Constraint) Logic Programs (CLP) [15]. This combines the advantages of being declarative and efficient to a certain extent.

The major goals of the RoboLog project, undertaken at the University of Koblenz, Germany, are the following:

- A flexible, modular system architecture should be established, meeting the various requirements for RoboCup agents. For example, on the one hand, agents have to be able to react in real-time. But on the other hand, it is also desirable that more complex behavior of agents can be programmed easily in a declarative manner.
- It should be possible to handle different representation formats of knowledge about the environment. Information may be quantitative or qualitative in nature. Therefore, we propose a deductive framework, that is expressible in plain first-order logic (possibly plus constraint technology components), that integrates axiomatic approaches in geometry, spatial constraint theories, and numerical sensor data.
- Agents should not only be able to act autonomously on their own, but also to cooperate with other agents. For this, we develop a multi-agent script language for the specification of collective actions or intended plans that are applicable in a certain situation. These scripts can be translated into logic programs in a straightforward manner.

1.2 Outline of the Approach

In the following, we discuss our layered system architecture and the functionality of the respective layers. Fig. 1 shows the complete architecture of RoboLog. The lowest layer—the RoboLog kernel, which is implemented in C++—essentially is the interface between the SoccerServer [9] and Prolog, since all other layers are implemented in this logic programming language.

The basic layer hosts reactive behavior. It is implemented in the RoboLog Prolog extension [21, 22]. This extension is an enhanced RoboCup SoccerServer interface for ECLiPSe-Prolog [14]. Time critical tasks are handled within the RoboLog module, as well as the exchange of data. The module provides the atomic SoccerServer commands and some more complex actions. Hence already at this level, logic (programming) formalisms are available. Also position determination is settled in this layer (see Sect. 2.1). It also provides more specific facilities, e.g. dribbling and ball interception. For these actions, (almost) no spatial cognition is required.

Spatial cognition is the contents of the second layer. For example, players have to recognize when passing the ball is possible or a player is offside. Many approaches (see e.g. [8, 26]) propose purely qualitative reasoning, i.e. disregarding quantitative information after it has been transferred into a qualitative representation. But this may be too inexact and too vague sometimes. Since we use logic as connecting formalism in all layers, we can access low-level data at all levels of abstraction. This implies,

reasoning can be as exact as required. We will present our approach in more detail in Sect. 3.

The last two layers host complex situations, possibly requiring teamwork, i.e. single- or multi-agent plans. Nevertheless, the question remains whether teamwork should be invoked explicitly by communication or whether it is sufficient and more robust just to have implicit (emergent) teamwork. The current implementation implicitly exploits knowledge on other implementation of agents. With the exception of the goalkeeper, they are clones of each other. Cooperative behavior may be required even if the implementation details are different or not known. The problem is then, what communication language can be used in this case. See also Sect. 5.2 on this topic.

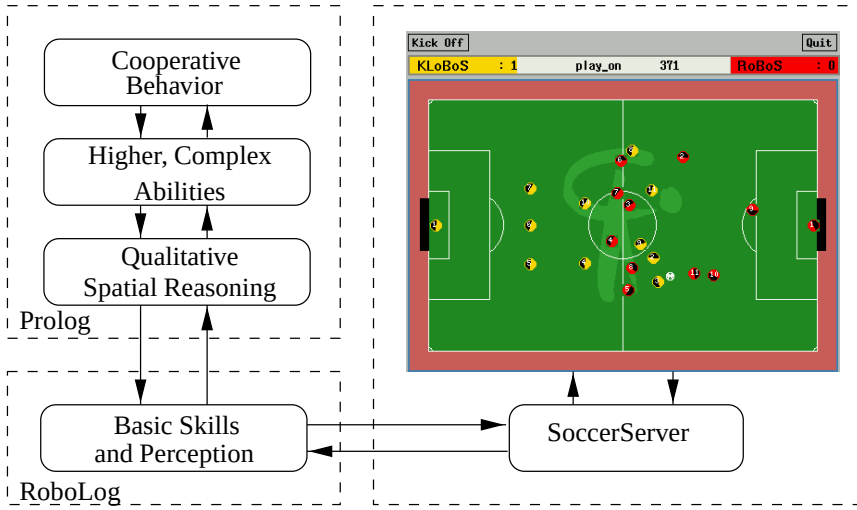

Fig. 1. System Architecture of RoboLog.

2 Basic Abilities and Actions (Layer 1)

The lowest layer in our system architecture handles basic skills and perception of the environment. The basic skills may be actions that can be performed immediately by the agent, e.g. turning around, dashing, kicking the ball etc. In addition, we will allow more complex actions in this layer, that do not need (qualitative) spatial reasoning.

Depending on the hardware used, perception of the environment, including self and object localization is a complex task, requiring more or less processing. In the simplest case, perception just means reading off the data from one of the agent's sensors. Note that we aim at having a (first-order) logic presentation for each agent. The logical description language we are going to introduce allows agent programs (scripts) to be written and interpreted in a manner similar to CLP.

Following the lines of [24], we distinguish two classes of predicates: ACTIONS a and PERCEPTIONS p. When executed successfully, a perception predicate p returns the requested data. We will assume, that this data is quantitative, i.e. some arguments of the predicate are (real) numbers. For example, a perception predicate p may return the distance to a certain landmark, measured in meters and given as a real number. The main matter of an action a is its side-effect, i.e. the performed action. Nevertheless, an action predicate (except the primitive actions of the SoccerServer) also is assigned a truth value, depending on the success or failure of the action. Note that the truth value for all predicates is dependent on the actual time t, when the action or request for data is executed.

In summary, the RoboLog interface provides the following functionality:

- For each agent, it requests the sensor data from the SoccerServer. By this, the agents' knowledge bases are updated periodically. If some requested information about a certain object is currently not available (because it is not visible at the moment), the most recent information can be used instead. Each agent stores information about objects it has seen within the last 100 simulation time steps.
- This low-level data is processed in such a way that more complex and more precise information becomes available, such as global position information (see also Sect. 2.1) or direct relations between objects with or without reference to the actual agent. The relation $is\_left(Obj_1, Obj_2)$, e.g., depends on the relative position of the agent, whereas $is\_between(Obj_1, Obj_2, Obj_3)$ is an agent independent property.
- Last but not least, Prolog predicates are provided that can be used to request the current status of sensor information on demand. The data should be synchronized with the SoccerServer, before an agent's action is initiated.

2.1 Position Determination

An important piece of information for an agent is to know its own position. Therefore, the RoboLog system provides an extensive library that makes precise object localization possible. The whole procedure implemented in the RoboLog kernel is able to work even when only little or inconsistent information is given. In particular, we employ the method for mobile robot localization using landmarks stated in [4].

2.2 Basic Skills

Agents have to be able to move in their environment without collision. This is a basic requirement for many practical robot multi-agent systems. In the RoboCup scenario agents should also be able to handle the ball. This means they must be able to run and kick to a certain position, dribble with the ball etc. Another important task is ball interception. For this, an agent has to recognize and compute the ball trajectory in advance, compute and go to the point where ball interception is possible, and stop the ball. This is a macro task, which could be executed in a certain situation without any qualitative reasoning.

A large set of low-level abilities for the RoboCup scenario is stated in [25]. There, kicking, goal-tending and—as a sub-task—getting sight of the ball among others are

considered as part of the low-level architecture of an agent. Of course, such tasks may require deep computation. However, only quantitative data is used for these actions. This is the reason why it is reasonable to classify these actions as basic skills. Nevertheless, more complex actions will require (deductive) reasoning. That is the contents of the next layer (see Sect. 3).

In our system, the following basic skills (among others) are implemented (see also [18] that also describes special skills of the goalkeeper):

- The agents can search for the ball, taking into account their knowledge about the last time the ball was seen.
- Dashing and kicking to a certain position, regarding the agent's condition and avoiding obstacles is possible and (based upon these skills) also dribbling.
- Extrapolating the ball trajectory to a given time in the future enables the agents to intercept opponent passes and block shots.

3 Qualitative Spatial Reasoning (Layer 2)

During a match, a human soccer player will enter a lot of different situations, in which he has to decide what to do. In most of the cases, he will decide regarding former experience, i.e. comparing his situation to situations he already handled before. Hence, if we want to build a client, we have to provide the client with some situations and connected actions. We decided to model situations with the help of qualitative relations for two main reasons.

- The agent's situation will almost never fit exactly into a stored situation pattern (identified by its set of preconditions), so we have to parametrize and abstract the patterns. A basic set of qualities can be very easily abstracted from the visual data sent by the SoccerServer (see below). Thus the step from describing situations by quantities with tolerances to using qualitative data is easily taken.
- We think that *qualitative* spatial reasoning reflects the thoughts of a human player more clearly than the use of *quantitative* data. Consider a human soccer player who tries reaching the ball. He will think something like: the ball is *close enough*, or: a team-mate is *nearer* to the ball. Based on these *qualitative* perceptions he decides whether to run towards the ball or stay where he is. He will not calculate the trajectory of the ball and determine a set of coordinates at which he can intersect it.

What we need in order to identify situations is the abstraction of quantitative data onto a qualitative level. Therefore, we have another class of predicates—in addition to the classes mentioned in Sect. 2—, namely QUALITIES q. Qualitative predicates are defined upon the quantitative perceptions via logical rules and constraints, e.g. the *in-front-of* relation (1) (see below). But it may also be the case that there are qualitative predicates or relations based on each other. In the latter case we speak of purely qualitative predicates or reasoning, e.g. the relation *left_and_in-front-of* can be reduced to

the qualitative predicates *left* and *in-front-of* (2).

$$in\text{-}front\text{-}of \leftarrow Dist > 0. \tag{1}$$
$$left \leftarrow Dir < 0.$$
$$left\_in\text{-}front\text{-}of \leftarrow left \wedge in\text{-}front\text{-}of. \tag{2}$$

For example, concerning the distance of an agent to the ball in the RoboCup scenario only a few (qualitative) aspects are interesting. Thus, in RoboLog we only distinguish few distances: *close* (the ball is in the kickable area), *near* (the agent is able to detect much detail by its sensors), *short* (maximal shooting distance), *far away* (sensor data become unreliable from this distance), *remote* (out of reach). Quantitative distance intervals can be mapped to qualities. Concerning the other direction, chosen plan schemes must be instantiated with quantitative data for the actual execution. A related work is presented in [8]. There, reasoning on the qualitative level (alone) is provided. Fig. 2 illustrates the correspondence between quantitative and qualitative distances.

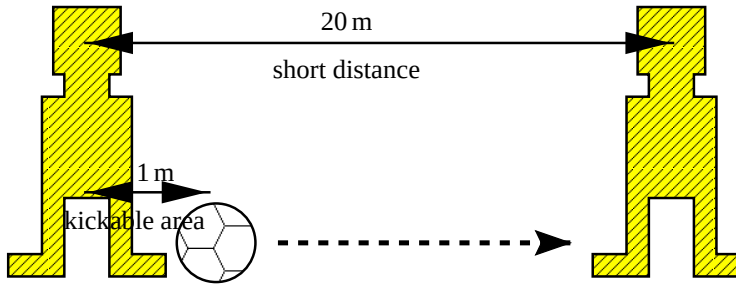

Fig. 2. Distances – quantitative and qualitative.

3.1 Constraint Reasoning

In the literature, many approaches for qualitative spatial reasoning are proposed. Most of them rely on the Region Connection Calculus (RCC), see e.g. [3, 23]. On the one hand, the advantage of qualitative information certainly is, that seemingly complex situations can be reduced to a few patterns of situations, and concentration on the relevant portion of information is possible. On the other hand, a qualitative description may be a too rough approximation of the reality, such that reasoning on a purely qualitative level may become too vague. So the question remains, how can we make use of both quantitative and qualitative information.

In most cases, if sensor data is available, it is a good idea to make use of the quantitative data by just abstracting it to a qualitative level. Only in some cases, when no more precise quantitative information is available, purely qualitative reasoning is necessary. More precise knowledge should be preferred. So, we combine real-time quantitative

reasoning with qualitative spatial reasoning, that can be implemented as a constraint system (in the formal sense) and integrated in a more general deductive framework for constraint logic programming (CLP).

The process of spatial reasoning has to be seen in the context of its purpose, that is laying the basis for what action should be performed next. There are (at least) two decision problems in this context:

- If there are different sources of information (e.g. numerical sensor data, derived qualitative knowledge or conclusions thereof), there must be some control mechanism for deciding how the requested information should be obtained. In our current implementation, quantitative data is preferred: it is simply converted into a qualitative presentation. There are only very rare cases where purely qualitative reasoning is performed. This could mean applying the transitivity rule to topological relations such as *between*.
- In addition, it may be difficult to decide what should be done next in a situation where we have several options (e.g. dribbling, passing, kicking). In the current implementation, we simply make use of the backtracking facilities of Prolog for this purpose. However, it might be a good idea to employ defeasible reasoning in this decision process [11].

3.2 An Axiomatic Approach

We are also investigating the problem of modelling certain situations as patterns by means of logic programs and the full first-order theorem proving system Protein [2]. For example, passing the ball is possible in a situation where one player has the ball, another player can be reached and there is no player (of the opposite team) in between. We modelled these situations on top of the logical relations *left*, *right* and *between*. Since we use logic, the properties of the qualities have to be axiomatized. Two possibilities come into mind: we can model *between* on top of general geometric axioms [5], or use *collinear* as basic concept [12]. We believe that it is more natural to use (an ordered version of) *between* as base relation, since we can assume that the sensor data provides information about order anyway. In addition, the order information may be required for planning certain actions in detail.

However, for axiomatic approaches in general, there is one problem: how can the negative information be deduced, e.g. if we want to know that there is no opponent in between. With Prolog alone this is not possible: the built-in negation as failure sometimes causes problems if used in complex queries. So we were led to use full first-order logic with the Protein theorem prover [2]. As example for this, let us consider the problem of determining whether passing is possible. This could be checked by the following logical rule with negation in the rule body:

$$Passing \leftarrow \neg\exists Opp : Between(Me, Opp, Partner)$$

The intended meaning of this rule is as follows: passing is possible, if there is no opponent between the agent and one of its partners. The question is: how should negation ($\neg$) and existential quantification ($\exists$) be interpreted? Protein provides classical negation

as usual in first-order theorem proving. Existential quantification causes problems, if treated by Skolemization, i.e. replacing existentially quantified variables by new constant or function symbols, because then we have potentially infinitely many players.

Since we need real-time behavior, we just considered the finite domain of players visible for the agent in our implementation. This is closed world or constraint domain reasoning. By this, we get a complete and terminating system. Possibly, more sophisticated kinds of non-monotonic negation can be used here in this context of decision-finding. Note that, currently, this component is not yet integrated into the actual RoboLog Koblenz implementation, but has been used for axiomatizing situations (see [6]).

4 Higher Abilities (Layer 3)

Many tasks require deeper reasoning, which can be expressed within a BDI agent architecture [24]. In our context, a BELIEF b is a qualitative predicate q, its negation $\neg q$ or a conjunction of beliefs $b_1 \wedge b_2$. A GOAL g is either an *achievement* goal $!q$ or a *test* goal $?q$, where q is a qualitative predicate. A DESIRE (or event) d is a goal or an action. Now we can build rules for a certain SITUATION in form of scripts, written $d : b - i$, where d is a desire, b is a belief (identifying the precondition of the situation), and i is the INTENTION (or, strictly speaking, the intended plan).

4.1 Intended Plans

The intended plan is a tree of desires. Edges outgoing from test goals are labeled with *yes* or *no* and possibly a time-out delay. They realize alternatives in the plan. Depending on the truth value the agent follows different paths. Edges labeled with a time-out serve to delay the predicate. The agent only follows the labeled edge, if the respective truth value holds at a time within the time-out interval. An achievement goal has to be performed actively by the actor. The actual execution of an intended plan sometimes makes it necessary to leave the abstract level of qualitative reasoning and operate on quantitative data.

If an action or achievement goal fails or an external interruption occurs (e.g. a referee message in the RoboCup scenario), the agent has to return to a *default plan*, which must be applicable without precondition.

4.2 Example 1: The Goalie Runs Home

Let us now consider an example for such an agent script. When the ball is in the opponent half of the field, the goalkeeper of RoboLog Koblenz moves to his home position and waits there in order to regain stamina. This means, if the goalie *believes* that the ball is in the opponent half, his *desire* is to be at his home position. So he executes the *intended plan* to run there. Figure 3 (a) shows the respective script. In order to execute this script, the agent has to further decompose the desire $\mathit{Run\_to}(\mathit{home})$ as shown in Figure 3 (b).

Let us now take a deeper look at the three desires of the second intended plan in Fig. 3 (b). Each of them shows a different aspect of the language.

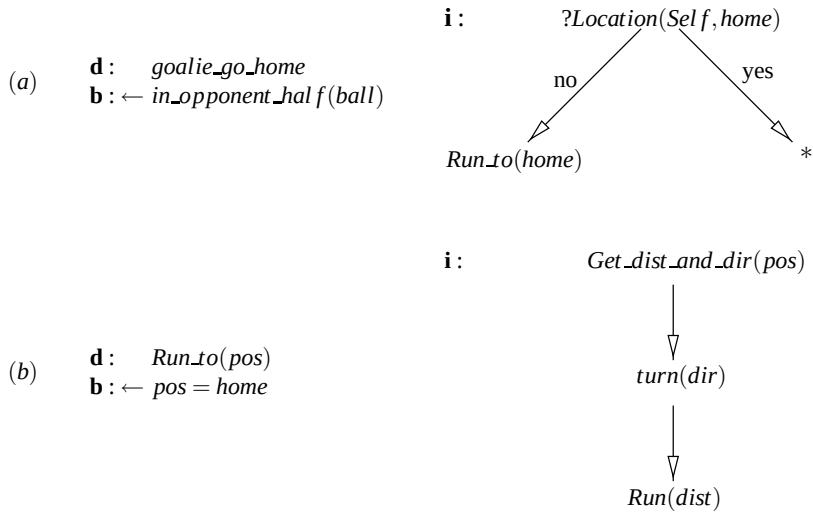

Fig. 3. Scripts for the goalkeeper.

- *Get_dist_and_dir*(*home*): The satisfaction of this desire realizes the transition from the *qualitative* level to the *quantitative*. It takes a quality (*home*) as input and returns quantitative values, namely the relative distance and direction of the home position from the agent. The other desires operate on these quantities.
- *turn(dir)*: This action belongs to the lowest level of our architecture. It is atomic in the sense that it can be sent to the SoccerServer directly.
- *Run*(*dist*): This, finally, is a complex action. From the point of view of our agent language, it is assumed to be *atomic*, too. But for actual execution, it has to be decomposed into a series of *dash* commands.

5 Cooperative Behavior (Layer 4)

The description language introduced in Sect. 4 is only suitable for modelling single-agent plans. But as we want to describe situations in which several agents have to cooperate, we will now extend the language to allow for the description of collective actions and multi-agent plans.

In this context a DESIRE *d* is a goal or an action, *indexed by a list of agents*—the actors—, which must satisfy the desire by performing some actions. Now the intended plan *i* becomes an acyclic graph of desires with a designated start node. Its edges are labeled with actors which must be a subset of the actors in *d*. Consider now all possible subgraphs wrt. edges for a certain actor. It is required that this still is a tree with the start node as root, where binary branching is only allowed after test nodes. These subgraphs represent the ROLE for the respective actor. Achievement goals are performed by the indexed actors, while non-actors wait for the achievement until a certain time limit. So for the latter such an achievement goal automatically becomes a test goal, normally labeled with a certain time-limit.

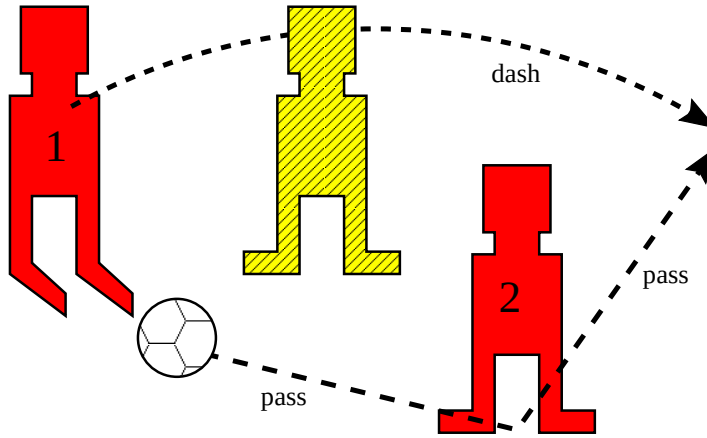

Fig. 4. Double Passing Situation.

5.1 Example 2: Double Passing

Let us now consider an example for a collective action of agents, namely double passing. There are two actors in this situation: actor 1 kicks the ball to actor 2, then actor 1 runs towards the goal, and expects a pass from actor 2. This is illustrated in Fig. 4. In order to initiate such an action, two agents simultaneously have to recognize their respective roles in the current situation in their belief state. The belief b for a double passing situation can be described as follows: 1 and 2 are nearest neighbors belonging to the same team, and player 1 has the ball. Player 2 must be clear, whereas an opponent is near to 1 such that 1 cannot dribble straight on. The intended plan i is then, that 1 passes the ball to 2 at first, then 1 runs towards the goal, and finally 2 passes the ball to 1. The respective rule can be expressed as shown in Fig. 5.

$\mathbf{d}: \quad DoublePassing_{1,2}$
$\mathbf{b}: \leftarrow NearestNeighbor(1,2),$
$\quad Close(1,ball),$
$\quad Clear(1,2),$
$\quad Between(1,Opp,goal)$

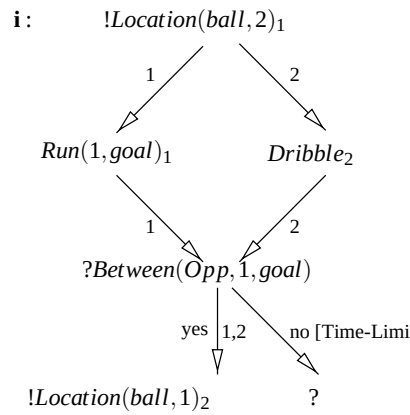

Fig. 5. Double Passing Script.

While experimenting with an implementation of double passing, we noticed that the main problem is that both actors simultaneously have to recognize their role, because one of the agents possibly does not see the other agent. In this context, communication (i.e. telling the other agent one's desire) helps a lot. A cooperating partner could tell its coordinates or even its whole own belief state. We made similar experiences with an even simpler kind of action, namely simple passing.

5.2 Communication

As we stated earlier, it makes sense to allow communication between agents. It helps them to recognize situations or their roles in them and thus reduces the complexity of the agents' reasoning and decision processes. But then another implementation decision has to be made, namely which communication language to use.

A general approach for the exchange of knowledge between agents is the Knowledge Query and Manipulation Language (KQML) [17]. However, if the domain of application is restricted, KQML may be too general. But it allows reliable communication between agents, even if their internal architecture is quite different or unknown for the other agent, by providing a common syntax. Instead, we communicate Prolog predicates directly. The advantage of this approach is, that no meta-logical interpretation of received information is necessary. A disadvantage is that for a successful communication the agents have to know each other's internal structure exactly. But this drawback can be overcome by specifying a subset of the available predicates together with their intended functionality as the communication language.

Thus, communication between the agents can be done by transmitting these predicates together with the action the recipient is expected to take on them, i.e. execute them as function. The goalkeeper, for example, could communicate his uniform-number to his teammates by saying $assert(goalie\_nr(1))$. The language is by its definition specific to the domain, thus enabling efficient communication while maintaining the flexibility of a more general language like KQML.

5.3 Translating Rules into CLP

We may distinguish several types of plans: basic plans with only one actor and complex plans where there are more than one actors. The former plans implement higher abilities (layer 3), while the latter realize teamwork (layer 4). Each BDI script can be translated into a CLP rule in a straightforward manner. For each achievement or test goal we introduce new symbols: $\underline{!P}$ and $\underline{?P}$. For each rule some default recipes are introduced:

$$P(x_1,\ldots,x_n) \leftarrow \underline{!P}(x_1,\ldots,x_n).$$
$$\underline{?P}(x_1,\ldots,x_n) \leftarrow P(x_1,\ldots,x_n).$$

The former and external events update predicates; this is the main difference to CLP. An approach that can handle external events and concurrency is *ConGolog* [10].

For each situation and for each role in it, a BDI script can be translated directly into a logic program rule, possibly with concurrent constraints (belief conditions):

$$d \leftarrow b \wedge i$$

The reader may have noticed that a situation with n roles corresponds to n CLP rules. These rules are identical wrt. their heads d. The preconditions b for the actions are also very similar; they only differ in their actor role. The last (but not least) part i is really different, because each actor plays a different role in the respective situation. For example, the instantiated plans for both actors of the double passing rule (see Fig. 5) are as follows:

Role 1	*Role 2*
$!Location(ball, 2)$	$?Location(ball, Self)$
$Run(Self, goal)$	$Dribble$
$?Between(Opp, Self, goal)$	$?Between(Opp, 1, goal)$
$?Location(ball, Self)$	$!Location(ball, 1)$

Recall that achievement goals are converted into test goals for non-actors. In addition, the control sequence for giving up after some time-limit is not shown here. Clearly, the translation into several CLP rules increases the time complexity for deciding which action or role therein is performed next. This problem can at least be partially overcome by communicating the next action directly to partners. In fact, we do this in our implementation by sending calls to Prolog predicates. But nevertheless, robustness of the whole system (of agents) has to be guaranteed in the case of failing actions or failing communication.

6 Conclusions

We presented a logical description language for multi-agent systems, following the lines of [24]. This language can be understood as a generalization of CLP. Both, quantitative and qualitative spatial reasoning can be built-in. With the script language proposed here, it is possible to express multi-agent plans. The RoboLog system provides a clean means for programming soccer agents declaratively. We conducted several test games with different scores on our local network—a 100 MBit Ethernet—and participated in RoboCup-99 (see also the team description *RoboLog Koblenz* in this volume).

6.1 Other Approaches with Logic Programming

Despite of the fact, that there are many logic-based approaches to agent programming in the literature, there are only few systems that are implemented with logic programming and that participated in the RoboCup. So, it seems that almost no team employs one of the well-known planning techniques in artificial intelligence (e.g. with the situation calculus [10]). *CS Freiburg*—the world champion of the middle-size league in 1997—makes use of path planning [13], but emphasizes the need of reliable basic skills. In this approach, path planning is restarted again every few milliseconds.

As mentioned in the introduction (Sect. 1), [24] proposes a framework that allows agent programs to be written and interpreted in a manner similar to that of Horn-clause logic programs. Nevertheless, only single-agent actions can be specified within this approach. The team described in [16] participated in RoboCup-98. The architecture of this system is layered (as ours) and hosts a behavior-based, a local planning, and a social planning layer. The system is implemented with *Oz*, a concurrent constraint logic programming language.

Another interesting approach is presented in [19]. There, an architecture for intelligent agents (with application to the RoboCup simulation league) is described, using the so-called organic programming language *Gaea*. It provides dynamic rearrangement of programming modules and multi-threading among other features. This, of course, is needed in a dynamic context as robotic soccer: when the system predicts or detects a change in the environment, it can swap some portion of its program accordingly.

6.2 Future Work

Further work should concentrate on the real-time requirements in exceptional situations and the concurrency of different mechanisms for information acquisition. The robustness of the decision process can be improved by means of defeasible reasoning [11] and/or organic programming [19]. Another area of research is how far logical mechanisms can be used within the lower levels of our approach. Deduction could be used to build a more complete view of the agent's world model. The application of these techniques to real robots is one of the next steps of our research activities. Finally, the specification of a flexible communication language should also be investigated.

References

[1] M. Asada and H. Kitano, editors. *RoboCup-98: Robot Soccer WorldCup II*. LNAI 1604. Springer, Berlin, Heidelberg, New York, 1999.

[2] P. Baumgartner and U. Furbach. PROTEIN: A PROver with a Theory Extension INterface. In A. Bundy, editor, *Proceedings of the 12th International Conference on Automated Deduction*, LNAI 814, pages 769–773, Nancy, 1994. Springer, Berlin, Heidelberg, New York.

[3] B. Bennet, A. G. Cohn, and A. Isli. Combining multiple representations in a spatial reasoning system. In *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97)*, pages 314–322, Newport Beach, CA, 1997.

[4] M. Betke and L. Gurvits. Mobile robot localization using landmarks. *IEEE Transactions on Robotics and Automation*, 13(2):251–263, Apr. 1997.

[5] K. Borsuk and W. Szmielew. *Foundations of Geometry*. North-Holland, Amsterdam, 1960.

[6] B. Bremer. Erkennung von Paß- und Abseitssituationen mit räumlichem Schließen. Studienarbeit S 570, Fachbereich Informatik, Universität Koblenz, 1999.

[7] H.-D. Burkhard, M. Hannebauer, and J. Wendler. Belief–desire–intention – deliberation in artificial soccer. *AI Magazine*, pages 87–93, 1998.

[8] E. Clementini, P. Di Felice, and D. Hernández. Qualitative representation of positional information. *Artificial Intelligence*, 95(2):317–356, 1997.

[9] E. Corten, K. Dorer, F. Heintz, K. Kostiadis, J. Kummeneje, H. Myritz, I. Noda, J. Riekki, P. Riley, P. Stone, and T. Yeap. *Soccerserver Manual*, 5th edition, May 1999. For Soccerserver Version 5.00 and later.

[10] G. De Giacomo, Y. Lespérance, and H. J. Levesque. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In M. E. Pollack, editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1221–1226, Nagoya, Japan, 1997. Volume 2.

[11] J. Dix, F. Stolzenburg, G. R. Simari, and P. R. Fillottrani. Automating defeasible reasoning with logic programming (DeReLoP). In S. Jähnichen, editor, *Proceedings of the 2nd German-Argentinian Workshop on Information Technology*, pages 39–46, Königswinter, 1999. To appear.

[12] C. Eschenbach and L. Kulik. An axiomatic approach to the spatial relations underlying *Left-Right* and *in Front of–Behind*. In G. Görz and S. Hölldobler, editors, *KI-97: Advances in Artificial Intelligence — Proceedings of the 21st Annual German Conference on Artificial Intelligence*, LNAI 1303, pages 207–218, Freiburg, 1997. Springer, Berlin, Heidelberg, New York.

[13] J.-S. Gutmann, W. Hatzack, I. Herrmann, B. Nebel, F. Rittinger, A. Topor, T. Weigel, and B. Welsch. The CS Freiburg robotic soccer team: Reliable self-localization, multirobot sensor integration and basic soccer skills. In Asada and Kitano [1], pages 93–108.

[14] International Computers Limited and IC-Parc. *ECLiPSe User Manual / Extensions User Manual – Release 4.0*, 1998. Two volumes.

[15] J. Jaffar and M. J. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[16] C. G. Jung. Layered and resource-adapting agents in the RoboCup simulation. In Asada and Kitano [1], pages 207–220.

[17] Y. Labrou and T. Finin. A proposal for a new KQML specification. Technical Report TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD 21250, Feb. 1997.

[18] J. Murray. My goal is my castle — Die höheren Fähigkeiten eines RoboCup-Agenten am Beispiel des Torwarts. Studienarbeit S 564, Fachbereich Informatik, Universität Koblenz, 1999.

[19] H. Nakashima and I. Noda. Dynamic subsumption architecture for programming intelligent agents. In *Proceedings of the International Conference on Multi-Agent Systems*, pages 190–197. AAAI Press, 1998.

[20] I. Noda. Soccer server: a simulator for RoboCup. In *JSAI AI-Symposium*, 1995.

[21] O. Obst. *RoboLog – An ECLiPSe-Prolog SoccerServer interface: Users manual*, March 1998.

[22] O. Obst. RoboLog: Eine deduktive Schnittstelle zum RoboCup Soccer Server. Diplomarbeit D 488, Fachbereich Informatik, Universität Koblenz, 1999.

[23] D. A. Randel, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connections. In *Proceedings of the third Int. Conf. on Knowledge Representation and Reasoning*, pages 165–176, San Mateo, 1992. Morgan Kaufmann.

[24] A. S. Rao. AgentsSpeak(L): BDI agents speak out in a logical computable language. In W. van de Velde and J. W. Perrame, editors, *Agents Breaking Away – 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, LNAI 1038, pages 42–55, Berlin, Heidelberg, New York, 1996. Springer.

[25] P. Stone, M. Veloso, and P. Riley. The CMUnited-98 champion simulator team. In Asada and Kitano [1], pages 61–76.

[26] K. Zimmermann and C. Freksa. Qualitative spatial reasoning using orientation, distance, and path knowledge. *Applied Intelligence*, 6:49–58, 1996.