Real Time Control for Autonomous Mobile Robots

Hans-Dieter Burkhard
Institute of Informatics
Humboldt University
Berlin, Germany
hdb@informatik.hu-berlin.de

September 26, 2008

Abstract

Control of autonomous mobile robots in dynamical environments is interesting from a cognitive point of view as well as under application view points. Stimulus response controls are often favored for fast reactions to the environment. On the other hand, goals and plans are needed to guide long term behavior by deliberative control. Layered architectures are used for combination, but they inherit advantages as well as problems from both approaches. The paper proposes the Double Pass Architecture with hierarchically structured options/behaviors providing fast reactions on all levels.

1 Introduction

Control of autonomous robots in dynamical environments is interesting from a cognitive point of view as well as under application view points. Technical requirements are established to construct intelligent autonomous systems in virtual worlds like the internet as well as in the real world. But still there is an ongoing debate about the best way to control intelligent behavior. Examples from nature include

1. Immediate reactions to inputs from the real world [Maes90], [Brooks91]: This approach can lead to surprisingly complex behavior if the stimulus-response actions are well tuned. The basic idea behind this approach is to use the complexity of the environment for control: The best model of the world is the world itself, complex behavior *emerges* from the interaction with the world. But note that virtual reality worlds must simulate the physical relations including substitutes for body sensors to a very detailed level to allow efficient stimulus-response behaviors.

2. Actions following long term plans [Bratman87], [Rao/Georgeff91]:
   The control uses complex internal models which are analyzed for reachable goals. Plans are developed to achieve the goals. It needs a lot of efforts to make appropriate models even for simple behaviors like following a path while avoiding obstacles in a dynamically changing environment. But complex behavior (e.g. playing chess or constructing an air plan) needs a lot of appropriate proactivity.

3. Swarm intelligence [Parunak97]: Complex behavior emerges from the interaction of large groups of simple agents. This approach can be seen as an extension of the first one using cooperation. Cooperation emerges by similar reactions of the agents to similar sensory inputs. Moreover, the results of the activities in the world are used as stimuli of other agents (e.g. the use of chemical substances as markers by insects). Flexibility and adaptation are realized by a certain randomness of reactions.

The paper is concerned with individual agents, i.e. with the first two approaches. Layered architectures are used for combination [Arkin1998, Murphy00]: Lower layers implement fast reactions using "behaviors", higher layers implement the guidance of behaviors by plans. The higher layers are called with lower frequencies and have longer reaction times.

Different behavior needs different triggers. Simple stimulus-response behavior is triggered by recent sensory data only. But often the environment does not provide appropriate sensory data for triggering the achievement of a long term goal (e.g. running to intercept a ball coming from behind - this point will be discussed in more detail below). This means the agent needs some knowledge about the situation in the outside world behind the sensory data and some knowledge about its goals and plans.

More abstractly spoken, the agent possesses "persistent" (mental) states to memorize world models and goals, respectively. We call them "persistent" states to emphasize their *persistence over longer time intervals*. This is necessary to make a clear distinction concerning certain software aspects: During complex computation processes we have "intermediate" states. Often used methods for action selection are decision trees, state machines, rule bases etc. These methods may go through different "intermediate" states while performing the selection process. But these intermediate states are forgotten when the process is finished. But if the result of the process needs to be stored (like a goal to be achieved in the future), then this is realized using a persistent state. It is used as an internal trigger for forthcoming decision processes.

Commonly used notions are "reactive" and "deliberative" behavior, respectively. Reactive behavior is mostly understood as simple behavior, without (persistent) commitment to goals and plans. Deliberative behavior is identified with complex decisions. There are different aspects mixed in these notions as

- complexity of the decision process,

- ability to anticipate possible future developments,

- planning capabilities,

- persistent states concerning the past (persistent worldmodel),

- persistent states concerning the future (persistent commitments).

As an example we may consider a chess program: It can anticipate future situations considering the possible moves of both players starting with the recent situation. It can evaluate reachable future "goals" using complicated evaluation procedures. Finally it comes up with simply the next move, and all intermediate results are forgotten. After the opponent's move, the same process starts again for the new situation without any reference to the previous computations. Is it a reactive behavior? We cannot solve these terminological problems in this paper, but we will discuss some of its aspects and the reasons behind.

The practical problems concern rapid reactions to fast changes in the environment. Reactive behaviors are considered appropriate for rapid reactions, while deliberative ones are concerned with long term planning. If long term planning is used in dynamically changing environments, both approaches are needed, but then they are in conflict concerning their synchronization. Layered architectures combine deliberative "higher" layers with reactive "lower" layers. Different synchronization strategies are in use, but usually the higher layers have some delay because they are computationally expensive. Hence only lower reactive layers react dynamically. Thereby they act according to the long term goals defined by the higher deliberative layers. But this goals remain the old ones as long as there is no redeliberation regarding the new requirements. In fact, there is a real time control problem concerning fast redeliberations.

To allow some kind of short term redeliberation in complex environments, it is inevitably to restrict the search space for rapid decisions. This corresponds to concepts of bounded rationality, where a special "screen of admissibility" ([Bratman87]) is introduced for the restriction of deliberation processes. The proposal in this paper is a new architecture with two separated passes through all levels of control as an attempt to combine complex long term decisions with short term behaviors under real time conditions.

The paper is organized as follows: General aspects of robot controls in dynamical environments are considered in Section 2 using the scenario of soccer playing robots (RoboCup). Control architectures are discussed in Section 3. Section 4 continues the discussion of control problems, the impacts of these problems to the design of control architectures are investigated. This analysis leads to the proposal of an hierarchically structured control architecture in Section 5. It allows for long term and short term decisions on all levels of the hierarchy. In contrast to other layered architectures, just-in-time decisions are possible on the higher levels, too.

2 Robot Control in Dynamic Environments

Dynamic environments are characterized by fast changes, such that plans may become invalid by unpredictable events. The robot football (European "football", i.e. "soccer") scenario promoted by the RoboCup initiative [RoboCup] [Kitano-et-al-97] is best suited as an illustrative example. It provides a dynamic environment for the football/soccer playing robots. Special characteristics are the presence of adversaries and the availability of only incomplete, imprecise data. One may theoretically think about a plan to play the ball via several players from the goal-kick to the opponents goal, but nobody would expect that plan to work. Note that there is a great difference to a chess program: It is easy to write a program for finding the ultimate best moves, it is "only" a question of complexity to run this program. But nobody is able to write a similar program for football/soccer playing robots.

It is important to realize that the robots have to work autonomously without any outside control. Moreover, there is no global control in our scenario: Each robot has to decide for its own with restricted knowledge about the environment and about other robots. Some communication is provided, but not enough to exchange detailed information about the situation and about decisions (the amount of data is restricted).

Control structures for intelligent robots/agents include

- sensors and perception unit to get inputs from the environment,

- behavior control (with different complexities ranging from simple stimulus response behavior to long term deliberative behavior as discussed in this paper),

- actors and basic action control to act in the environment (sometimes using direct feed back with sensors),

- operating system for synchronizing the different activities (using parallel processing if possible).

Communication capabilities are included in the sensors and actors, respectively. There exist a lot of different approaches for controls of intelligent agents and intelligent robots (cf. e.g. [Arkin1998, Murphy00, Weiß99]).

2.1 Basic Skills

The football/soccer scenario provides a lot of different situations to illustrate the needs of agent architectures for dynamic environments. They range from basic skills up to complex cooperative behavior.

- The raw input information provided by sensors is processed to yield a perception. The resulting data structure models the environment including the robot itself (especially positions and movements of the ball and of the players). It is called the "worldmodel".

4

- The information provided by sensors in a single moment is incomplete (the ball may be covered by other players) and imprecise (due to noisy data). It is possible to build a more complete worldmodel using information from the past together with the new perception. For example, the movement of a ball which is covered by other players can be anticipated using information from the old worldmodel.

  The important aspect of such a worldmodel is its *persistence* with respect to a time scale regarding sensor inputs and effector outputs. The agent maintains such a worldmodel as a *persistent state* with updates according to new sensory information. Since it is oriented to information from the past, it is called past-oriented mental state.

- Interception of a moving ball illustrates simple problems of the dynamic environment: A very simple "stimulus-response player" would run straight line to the place where he sees the ball. As the ball is moving he has to adjust its direction every time he looks for the ball, and he will perform a curved path as the result. A more skillful player could anticipate the optimal point for interception and run directly to this point.

- Now we discuss such a procedure for the anticipation of the optimal point for interception . It calculates the speed vector v for the optimal run to the ball depending on the recent position p and the speed u of the ball (relative to the player). It may use additional parameters according to opponents, whether conditions, noise etc.

  The calculation may explicitly exploit physical laws (including e.g. the expected delay of the ball). It may use simulation (forward model) for possible speed vectors v of the player. If an inverse model is available, the optimal speed vector v may be calculated directly. Calculations of v may use a neural network which has been trained by real or simulated data. (Which of these methods should be called "reactive"?)

- We still consider the optimal interception of a moving ball using calculations of the speed vector v. The calculation can be repeated whenever new sensor information is available. Therewith it always can regard newest information and hopefully obtain the best speed vector v. Alternatively, the player may keep moving according to v for a longer time. Therefore he needs another kind of *persistent state* to memorize this goal. Since this state is oriented to information concerning the future, it may be called future-oriented mental state. It can save computation time, and it is useful to keep stable behavior (see below).

  If the ball is not observable for some time (e.g., if it is covered by another player), then the persistent goal is used as the trigger to keep running. Alternatively, simulating the ball in the worldmodel can also be a trigger to continue the interception process.

- Problems with the reliability of the computed speed vector v arise due to noise in the sensory data (and may be due to imprecise calculations them-

selves). Repeated calculations may hence result in oscillations and sub-optimal behavior (as reported e.g. in [Müller-Gugenberger/Wendler1998]). It may be better to follow the old speed v_t as long as the difference to the new speed v_{t+1} is not too large. Keeping v_t in a future oriented mental state provides the necessary means. Exploiting the inertia of the robot provides another way using the physical world directly. A complete analysis of the problems behind stability and flexibility go behind the scope of this paper (cf. e.g. [Bratman87],[Burkhard00]).

The discussion shows a lot of different approaches and implementations for the simple behavior "follow a moving object". In most cases there is a lot of redundancies which can be exploited for efficient and more reliable controls in different ways. It is a typical observation in robot control that the same behavior can be realized in different ways yielding different trade offs. Since single methods are often of restricted reliability, the appropriate combination (regarding the overall system) is a challenging design problem.

To summarize: Two concepts of persistent ("mental") states have been introduced. It is commonly accepted that some form of persistent state is essential even for primitive beings. The worldmodel as a persistent state concerning the past compensates missing sensory information from the outside world. The persistent state concerning the future may be not really necessary at the level of basic skills.

2.2 Coordination

More complex problems of dynamic environments are illustrated by coordination. The decision processes become more and more complex (and subject to stability problems) as the time horizon is enlarged. Even in the recent Simulation League of RoboCup (the competitions in a virtual environment which do not suffer from the physical robot problems), a coordinated behavior like a double pass emerges only sometimes by chance, not by planned activities. Here are some examples of decision processes in the RoboCup scenario:

- A player decides if he can intercept the ball, i.e. if the ball is reachable during its move on the playground. The decision process can use the procedures for computing v from above to calculate the interception point and time.

- A player decides if he can intercept the ball before any other player. Therefore he has to compare his own chances with the interception times of other players (e.g. using the methods to calculate v from the view point of other players).

- A player decides not to intercept the ball even if he is the first to reach the ball. The reason may be a team mate in a better position for continuation.

Next we had to discuss the optimal behavior for all the players which are not in a position to control the ball directly. Their optimal behavior is determined by

long term strategic items, and it is important for the success of a robot team. Humans often use predefined behavior patterns for coordination, like change of wings, double pass etc. in football/soccer. The reader is invited to think about the related problems as discussed above for interception: maintenance of information from the past, anticipation of future chances, managing of stability and optimality, – all under the conditions of dynamic changes and incomplete imprecise information. There is a growing value of global (symbolic) descriptions of situations and behaviors in order to guide short term behavior by long term goals. Goals and plans are memorized by future oriented persistent states for at least two reasons, namely efficiency (repeated computations should be avoided) and stability (needed for cooperation of team mates).

3 Architecture Models

3.1 A Simple State Model

The notions of persistent states are discussed somewhat more formally in this section. A discrete control of the agent is considered for the sake of simplicity. There is some freedom for choosing the time steps $t = 0, 1, 2, \ldots$. There are good reasons to identify the time steps with the arrival of sensory data ("input") at the control unit (e.g. we have some kind of event driven control). Note that persistence depends on this definition: We call a state a persistent state if and only if it keeps information from one step to the next. The chess program considered in Section 1 does not have persistent states. It needs no persistent worldmodel if all board positions are used as input, and it needs no memorizing of goals if evaluation of possible moves starts from scratch for the new situation.

Computed goals and plans of a football/soccer robot are not persistent as long as they cannot be used by the decision process in the next time step. (Our implementation of the control architecture in [Burkhard *et al.*98] was based on the notions of belief, desire and intentions to describe intermediate results. Actually, the concepts did not stand for persistent states since the decision process was started from the beginning in each time step. But then certain stability problems occurred like oscillating directions while intercepting the ball. They were solved by references to old intentions later.)

A generic agent-oriented control architecture with a simple cyclic process ("sense-think-act"-cycle) is widely used. The cycle is performed at each time step t.

1. Input (sense): Data have been collected by the agent. They may come from outside (sensors), via communication and by body information (proprioceptive sensors). The data is preprocessed yielding some internal representation of the environment which is called "worldmodel". (Here the notion "worldmodel" may stand for non-persistent data, too.)

2. Commit (think): The control unit analyses the worldmodel. It may evaluate possible courses of actions and possible future situations. It commits

for actions to be performed immediately and perhaps for long term behavior.

3. Output (act): The control unit outputs the advice for actions to be performed by the agent immediately. The actions are performed (may be after some further processing) by effectors, and by communicators.

The most simple architecture is an architecture without any persistent state as in Figure 1. Only the most recent input can be used for the control. This causes no problems if the input is complete and reliable as far as necessary for commitments.

```
for t=0,1,2,...  do
        worldmodel_t := perceive(input_t);
        commitment_t := deliberate(worldmodel_t);
        output_t := execute(commitment_t);
```

Figure 1: Stimulus-response Architecture without Persistent World model

The `deliberate`-function can be a simple table, a neural network or a complicated decision process using goals and plans – remember the chess program. But the commitments are used only for the recent output, then they are completely forgotten in the case of stimulus-response architectures.

Next the stimulus-response architecture with persistent world model is considered as in Figure 2. A persistent worldmodel allows to regard former inputs. The agent can try to maintain a complete worldmodel even if the most recent sensor information is incomplete. The new input is integrated into the existing worldmodel, missing facts can be simulated to some extend. This means that the agent has the ability to anticipate world states. It is common understanding that the persistent worldmodel is used only as a "past-oriented" state which memorizes information concerning the situation of the outside world: It serves as a substitute for a complete and precise sensory information by the input. Again the `deliberate`-function can be very simple or complex, respectively. Commitments are used only for the recent output.

As discussed above, efficiency and stability are reasons to memorize previous commitments to guide further decisions and actions. The `deliberate`-function can use complicated processes to evaluate possible future situations, it can make plans to guide the behavior for a longer time regarding coordination with other robots. It may be useful or even necessary (for stability) to consider the same commitment over several time steps. This means to have additional persistent states related to the future. This is considered by the architecture with persistent states for worldmodel and commitment as given in Figure 3.

The essential difference to the stimulus response architectures is the treatment of commitment as a persistent mental state. It can be split further e.g.

```
for t=0,1,2,...  do
        worldmodel_t := update(worldmodel_{t-1}, input_t);
        commitment_t := deliberate(worldmodel_t);
        output_t := execute(commitment_t);
```

Figure 2: Stimulus-response architecture with Persistent World Model

```
for t=0,1,2,...  do
        worldmodel_t := update(worldmodel_{t-1}, input_t);
        commitment_t := deliberate(commitment_{t-1} , worldmodel_t);
        output_t := execute(commitment_t);
```

Figure 3: Architecture with Persistent States for Worldmodel and Commitment

into desires, intentions, plans (BDI-architecture) with a lot of variants in the literature (cf. [Wooldridge99], [Burkhard00]). It serves for efficiency as well as for stability.

3.2   Layered Architectures

The concept of persistent states works fine as long as there is enough time for the calculations in a single interval between two time points t and $t + 1$. But real time architectures in dynamical environments usually allow for fast action control (by **execute**) in short intervals, while sensor integration and update of the worldmodel as well as commitment need more time. There are severe synchronization problems.

A common used model is a hierarchical architecture where **execute** performs "low level" behavior with short time horizon and fast specification time, e.g. for collision avoidance. Each such low level behavior is realized by simple methods, e.g. predefined scripts or in the form of stimulus response behavior. The aim of the **deliberate**-function on the higher level is the choice of such a script, the computation of a plan etc. Following the necessities of "bounded rationality", the **deliberate** process constitutes a reduced "screen of admissibility" [Bratman87]:

If the environment is complex then longer computation time is necessary to analyze the global situation (e.g. for image processing and interpretation, modeling of other players, calculation of the utilities of different strategies etc.). But not all aspects of the global situation are subject to fast changes (e.g. the

9

ball possession may change very rapidly, but the positions of players do not). Hence there is the possibility of shared work: Complex analysis is performed by the "global" `deliberate` calculations leading to search space reduction for "local" short time decisions of `execute`.

Classical layered two pass architectures have a control flow bottom up from lower layers to higher layers and then back again to the lowest layer. To act in time, the higher layers are used only if needed, or with lower frequency. In the first approach, the lower layers must decide if higher layers have to be involved. This can yield context problems as discussed in Section 4.2. In the second approach, higher layers have delays.

Layered one pass architectures have only one control flow through the layers. To act in time, the higher layers are called with lower frequencies. Implementations of one pass top-down architectures can use stack-oriented programming paradigms. Actions are pushed onto a stack. The action a from the top is executed if a is low level, otherwise it is replaced by lower level actions $a_1, \ldots, a_n$, respectively (cf. e.g. [dMARS]). Subroutine calls as used in (procedural) programming implement the same principle (using the run time stack). The computed commitment activates a subroutine which performs the necessary actions for the achievement of the committed goal. The control turns back to the higher level deliberation process for a new commitment when the subroutine has finished.

Such layered models have different time scales on their layers. This means that the synchronization between `deliberate` and `execute` is somewhat different to the description by Figure 3. The commitment can be understood as a (may be conditional) plan or script computed on "higher levels" (by `deliberate`) at time t such that

   `commitment`$_t$ = `step`$_t$, `step`$_{t+1}$,..., `step`$_{t+k}$.

The "low level" `execute`-function computes the outputs for $i = t, \ldots, t+k$ according to that script:

   `output`$_i$:= `execute`(`step`$_i$, `input`$_i$) .

The most recent input `input`$_i$ (or the worldmodel if available in time) is used for adaptation. A temporary stimulus response behavior is realized if identical steps `step`$_i$ are used. As an example we may think of the commitment to run to a certain position, where the `execute`-function has to realize the necessary movements over a longer time.

While `execute` is active at each time step t, the higher level commitments may remain unchanged over longer time intervals in the layered architectures. In fact using the subroutine-paradigm, the higher level processes are inactive until the lower level processes are finished. A problem of these approaches is the difficulty of fast reactions on the higher levels to unexpected changes in the environment. The problem cannot be overcome by concurrent computations on the different levels as far as the complete analysis of a global situation (including time consuming sensor processing and integration for the worldmodel, and future simulations, evaluations and means-ends-analysis for the commitment, respectively) consume more time than only a single interval between two time

points t and $t + 1$. We will discuss this matter in Section 4, and the proposal of the "Double Pass Architecture" in Section 5 is an attempt to overcome the problem. The name "Double Pass Architecture" refers to a difference to the one pass and two pass architectures, such that two independent passes are performed top-down for the `deliberate`- and `execute`-functions, respectively.

3.3 Classifications of Architectures

The consideration of persistent states can help to understand the differences between different agent/robot architectures. Especially the notion of "reactive" agents seems to be unclear as long as it is understood as "simple behavior". Choice of actions by a set of condition-action rules [Russell/Norvig95] may count as an example of simple behavior (but: is any number of rules and any kind of condition matching simple?). Do Neural Networks implement simple behavior? [Russell/Norvig95] avoid the name "reactive" and consider a "simple reflex agent" without any persistent state and a "reflex agent with internal state", where the state is considered as a persistent worldmodel. In both cases, reflexes are considered as "simple behavior", implemented by condition-action rules which are matched against the world model. More complicated reasoning is described by agents which are "goal-based" (trying to achieve goals) and "utility-based" (trying to maximize utilities).

There are different dimensions to classify agent architectures like complexity (concerning perception, worldmodel, commitment, ...) and persistence of states (concerning worldmodel and/or commitments: goals, plans,...). A "Stimulus response architecture" denotes an architecture without persistent commitments (future oriented mental state). It may or may not have a persistent worldmodel (past oriented mental state). A "deliberative architecture" denotes an architecture with both persistent commitment and persistent worldmodel. Nothing is said about simplicity/complexity of the procedures `update`/`perceive`, `commitment`, `execute`. A more formal consideration is planned for a forthcoming paper.

Some discussions in the community can be interpreted as asking how far a persistent commitment state is really needed. As discussed for the player running for a moving ball, the player can commit for a speed vector v according to an optimal interception point. He may keep v as a persistent commitment for a longer time. But he could also repeat the computation of v for each new worldmodel. The updates of the worldmodel may be performed using new inputs (if available), or simulations of the expected progress in the outside world. In any case, the worldmodel can be used as a trigger for the behavior instead of a persistent commitment to some extend. (Note that we do not allow a mixture between the contents of the states: We suppose that the data structures of the worldmodel are restricted to information concerning the outside world only, goals and plans are not parts of the worldmodel.)

Nevertheless, former commitments could be inferred from a history of the worldmodel: Knowing the old worldmodels, the old commitments could be recomputed by the `deliberate` function and then a recombination to the recent

11

commitment is possible (at least theoretically). The actual concern of the discussion is the possibility to let complex coordinated behavior "emerge" from simpler decisions. But the examples of complex coordinated play in section 2 illustrate the difficulties of approaches without persistent commitment.

It is also interesting to note that the algorithms used for the update of the worldmodel can be used to anticipate hypothetical future situations `h_worldmodel` using hypothetical inputs `h_input`$_{t+1}$, ... `h_input`$_{t+k}$:

```
worldmodel_{t+k}
    := update( ...(update(worldmodel_t, h_input_{t+1}), ...),h_input_{t+k}).
```

That means: An agent managing a persistent worldmodel has already the necessary means to simulate expectations of the future. He can compute the effects of possible actions, evaluate the expected results and then decide what to do. It means, proactivity is potentially possible whenever a worldmodel can be manipulated: The step from a stimulus response behavior with persistent worldmodels to proactivity is not too far.

4 Problems of Control

This section discusses some details of the problems concerning efficient controls. Efficiency means optimal behavior with respect to given constraints, especially complexity constraints ("bounded rationality").

4.1 Trade-offs

As discussed for layered architectures, worldmodel update and deliberation can be time consuming processes. An accurate analysis of the situation (i.e. by complex picture processing algorithms) is worthless if it comes too late. There is a **time-trade-off** between

- Precise decisions based on a sufficient analysis of the situation: It needs time for computing the perception from sensory data, its integration into the worldmodel, the calculation of possible outcomes of available activities, generation of appropriate plans etc.
  versus

- Immediate fast responses to the most recent sensory data: It leaves no time for complex deliberation.

Layered architectures distinguish between long term decisions (deliberations – which may need more time), and short term decisions (executions) guided by the long term ones. The guidance by the long term ones means a smaller scope of possible choices for the short term decisions and hence shorter computation times.

A problem of these architectures are delayed reconsiderations of long term plans: In the case of unexpected events, the adaptation of the long term commitments may come too late. Therefore collision avoidance is usually part of the low level behavior. In football/soccer, the ball handling can be considered as low level behavior, too. But for the other players, their low level behavior (e.g. running) is not related directly to the ball. Nevertheless, they should react quickly e.g. in cases when a team mate looses the ball.

The **stability-trade-off** concerns the consideration and optimal handling of (unexpected) changes in the environment. It is a trade-off between

- Fast adaptation to new situations in order to act according to the most recent data, and according to the most promising alternatives, respectively, versus

- Stabile following of old plans in order to pursue an intention. Stabile behavior is important for resolved acting and for cooperation (to ensure trustiness).

Both alternatives have their drawbacks: Stability has the danger of fanaticism, i.e. pursuing of unachievable goals, like keep on running for a pass when the ball is already controlled by the opponents. Adaptation may lead to permanent changes of behavior like oscillations, e.g. changing directions while running to an object.

Adaptation may be very inefficient if the costs for adaptation itself are high over time (think of an undecided goalie which permanently revises the place of the ball for a goal kick). Therefore, the costs of adaptation have to be considered, too. The appreciation of adaptation costs and consequences are a matter of the time trade-off.

The both trade-offs are directly connected with persistent commitments: Time can be saved by memorizing commitments. The distinction between short and long term decisions helps to react faster. Stability needs the consideration of former decisions which can be memorized by persistent commitments. (But there exist other possibilities, e.g. the exploitation of physical properties like inertia.)

There are much more alternatives concerning the construction of robot controls. If the robot performs exact movements then the efforts of motion control can be reduced. Vice versa, inexact movements may be compensated by extensive motion control to some extend. This is a another trade off with consequences to deliberation and execution procedures.

4.2   Context

The context problem is best illustrated by the behavior of a player which does not control the ball. All he can do is changing his position using simple behaviors like run/walk/stay. Good positions are essential for the success of the team. The player has a lot of different alternatives related to many different goals. More than for the ball controlling agent, the optimal behavior depends on the global

situation, i.e. of the context (like defensive or offensive play, distance to the ball / to other players / to the goals, actual score etc.).

In our first RoboCup implementations, position changing was handled on a global level. A unique procedure had to compute utilities for all situations. The calculation of useful utilities becomes more and more difficult with growing numbers of contexts. Thus, our results were rather raw.

A reasonable principle of agent architectures are hierarchical structures: Complex skills use simpler skills, complex options are structured by simpler options. Different positioning options can be embedded into larger options like goal defense, double pass etc. This makes deliberation easier: First the agent decides for a double pass, then he decides for the appropriate positioning actions in this context.

In the classical, sequential, stack-oriented software architectures this can be implemented by successively called subroutines: The "play soccer method" calls the "offensive play method" which calls the "double pass method" which calls the "positioning method" at the right time. Only the most recently called method/procedure is active (here: "positioning method"), the callers wait for the termination of this method. Each subroutine in the stack can be considered as a "level" of hierarchically ordered commitments. The number of levels is not restricted. In contrast, classical layered architectures have a very limited number of layers (e.g. two or three) which implement different reasoning methods and which are called with different frequencies.

In the case of unexpected events, the successively called subroutines can react only on the lowest level, i.e. by the active subroutine. Classical layered architectures have related problems, they react only on the lowest layer. This is sufficient as long as the higher layers need no fast changes according to unexpected events. The long term goal of an unmanned ground vehicle usually does not change because of an unexpected obstacle. After drawing aside it will continue its way to the former goal. If there are still serious problems, it can stop for deliberation. Hence the concept of fast low level reactions works well for such scenarios.

But, if fast changes are needed on higher levels, then the considerations of unexpected events could be done best in the appropriate context. For example, the loss of the ball by the second player during a double pass should be handled by the "play soccer method". It should lead to termination of the "offensive play method" and its successively called methods ("double pass method", "positioning method"). At the same time, the "play soccer method" should activate the "defensive play method" with appropriate submethods, e.g. for attacking the opponents.

If only the "change position method" is active (e.g. as the active subroutine), all necessary computations (analysis of the situation, test of conditions, termination of higher level routines up to the "offensive play method") must performed by this method. This leads to a very complex and inefficient "change position method". Moreover, since the "change position method" is used in many other contexts, this method becomes overwhelming complex. Alternatively, different "change position method" could be implementing for different
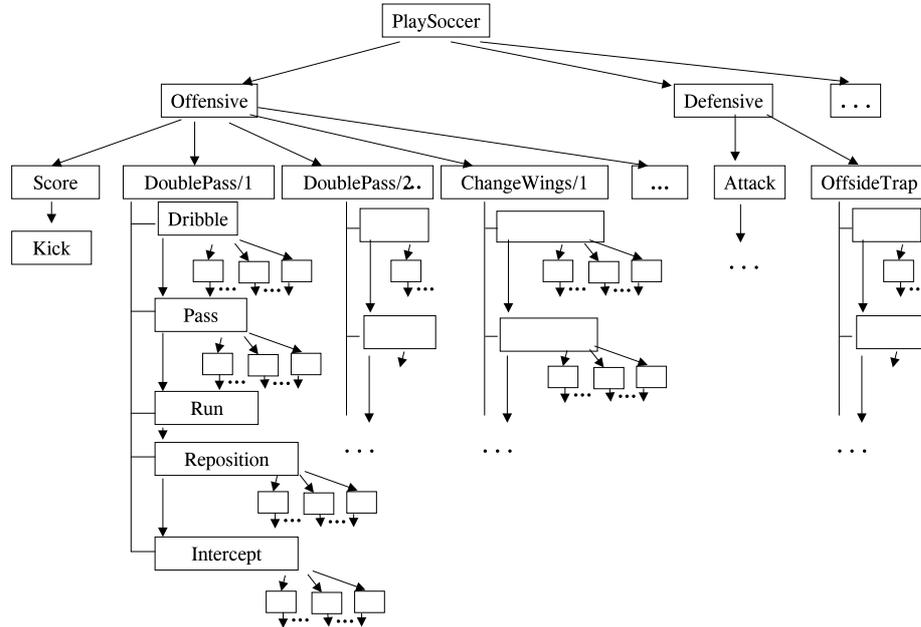
Figure 4: Example of an Option Tree

contexts. But this would lead to multiple copies of code.

The problem can be solved if the `execute` procedure has access to all levels, and if the decisions to be performed can be restricted. A solution is proposed in the following Section 5. The Double Pass architecture is intended to permit real time redeliberation on all levels using principles of bounded rationality.

4.3 Least Commitment

Long term commitments should be not too specific, since the future situations cannot be anticipated down to the very details in dynamic environments. In the beginning of a double pass, the first player knows about the sequence of dribble - pass - run -reposition - intercept, but he does not know the exact parameters of his pass (the final parameters will depend on the movements of his team mate and of the opponents). He does not exactly know to which position he will run afterwards and where he will intercept the ball. All this parameters are specified as late as possible. It can be considered as a kind of adaptation.

Layered architectures leave the specification to the low level behaviors which work in the scope of the decisions from higher levels.

5 The Double Pass Architecture

This section proposes an architecture which deals with the above problems in a reasonable way. It uses persistent mental states for the past (worldmodel) and for the future (commitment). It can implement goal-directed approaches, e.g. the BDI-approach [Bratman87]. It uses a hierarchical structure and a least commitment strategy. The hierarchical structure provides options on different levels. It is traversed by two independently running passes.

The hierarchy allows to describe behaviors and plans in a unique way, ranging from single actions on the lowest level up to long term plans on the highest levels . The lower level behaviors are combined to higher level plans. The passes perform different tasks:

The Deliberator performs time consuming processes regarding all aspects of the recent situation like choice of goals and long term planning. It sets up a partial hierarchical plan. Following the least commitment idea, the plan is refined as time goes on. Normally the deliberator does not have time problems since he works with sufficient forerun. Time critical decisions are left to the executor.

The Executor performs the contemporary decisions. Based on the preparatory work of the deliberator, its search space is restricted to a minimum of decisions using the most recent sensory information. In contrast to classical layered architectures, the executor considers all levels in real time.

Both lines of operation are independently running *passes* through all (!) levels of the hierarchy: Thus we have a *"Double Pass"* run time structure. This is in contrast to runtime organization in layered architectures (where short time decisions only affect the lowest level) and in programming languages (where only the procedure on the top of the stack is active).

5.1 Options

The data structure from which goals (or desires and intentions) are chosenfrom are the options. The set of options can be considered as a (virtual) tree structure with long term options near the root and specific short term actions near the leaves. An example from the football/soccer domain is given in Figure 4. The numbers (e.g. in DoublePass/1) denote the role (first player) in a cooperative behavior.

An option is performed by appropriate suboptions as defined by the tree. There are two kinds of connections between options and suboptions:

- Choice-Options can be performed by different, alternative suboptions (e.g. a pass can be performed by a forward-kick, a sideward-kick etc.), cf. Figure 5 for a Petri Net description of the alternatives of an offensive option. Transition firing depends on side conditions. "MaxUtility" means temporal priority for the transition with highest utility according to the recent situation. Other conditions are boolean valued.
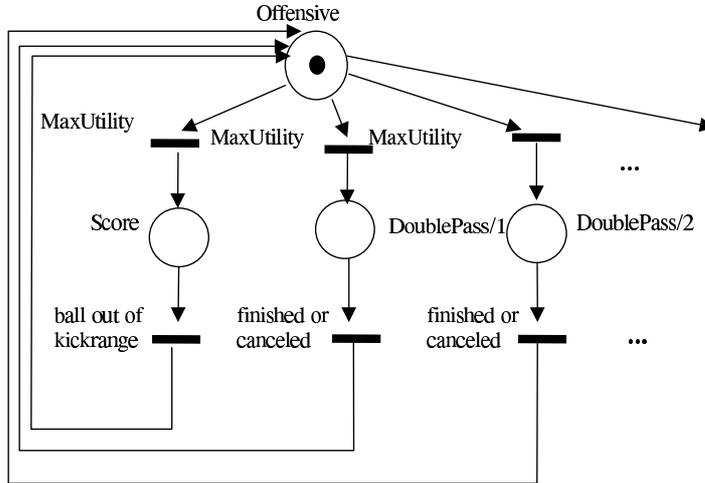
Figure 5: Example of a Choice-Option

- Sequencing-Options are performed by a sequence of suboptions (e.g. the suboptions of a double pass as described above), cf. Figure 6 for a Petri Net depicting the suboptions of the double pass option from the perspective of the player with the role DoublePass/1.

For clarity, the both kinds of connections are not mixed. This is similar to Prolog concepts: alternative suboptions correspond to different clauses of a predicate, sequenced suboptions correspond to the subgoals in a clause.

Choice-options describe the different possibilities in the context of that option. Deliberator activities consist of choices from the alternative suboptions (e.g. using utilities), calculating appropriate parameters (e.g. the player to cooperate with in a double pass) and decisions concerning the termination (or cancellation) of intended activities. Alternative plans can be provided if a plan is canceled. The hierarchical structure allows for local decisions. Redeliberation (if needed) is performed in a given context.

Sequencing options describe the steps (suboptions) needed to perform a higher level option. There have to be well-defined criteria for the transitions from one suboption to the next one. The evaluation of these criteria is time critical because they are performed by the executor when acting in response to the newest sensory data.

According to deliberation and execution, options can be in different states. The deliberator chooses options/suboptions to be executed as *intentions/subintentions*, their state is then called "intended". They build a subtree of the option-tree as shown for a double pass in Figure 7. The complete intention subtree must

Dribble Pass Run Reposition Intercept

Teammate free Pass finished Teammate passes Teammate finished Pass
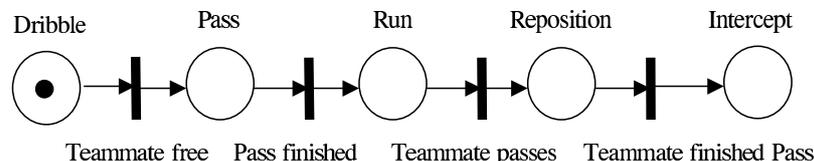
Figure 6: Example of a Sequencing Option

contain one subintention for each choice-option starting in the root down to some leaves, and all subintentions for each sequencing option. Using the least commitment principle, the intention tree has the form of a hierarchical partial plan. Subintentions describe the plan parts on different levels.

At any concrete time point, there exists a unique path in the intention subtree from the root to a leave consisting of the *active* options. This path is called *activation path*. At the time when the first player passes to the second one, the activation path in Figure 8 consists of "PlaySoccer"–"Offensive"– "DoublePass/1"–"Pass"– ... down to a concrete action (e.g. a kick-command with specified power and direction).

The executor performs the transition (as soon as the related condition is satisfied) from an active option to the subsequent option (as provided by the plan in the form of a sequencing option), and then the subsequent becomes active. For example, after the pass is finished, the player starts running for a new position (cf. Figure 6). Transitions are checked (and performed if conditions are fulfilled) on all levels following the activation path.

Besides intentions, the deliberator can also prepare desires as candidates for forthcoming or alternative intentions. Desires build a subtree similar to intentions. The deliberator may choose between different desires when he has to decide for an intention. Desires can be used as fast available alternatives for the executor when he has to stop a plan according to unexpected situations. As an example we might think about the fast switch to scoring a goal (because the situation allows it) instead of continuing the double pass (a related transition can be added to the Petri Net in Figure 5).

5.2 Deliberator

The aim of the deliberator is the preparation of intentions as partial hierarchical plans (built from options) without any time stress (cf. Figure 7). It can prepare this plan (as a desire) while the executor is stillperforming an old intention. For example, the deliberatorevaluates the available plans after an intercept while the robot is still running for the ball. At the same time, other players can evaluate their contributions to the possible plans of their team mate. As in real football/soccer, planning from stretch is difficult because of the indetermination
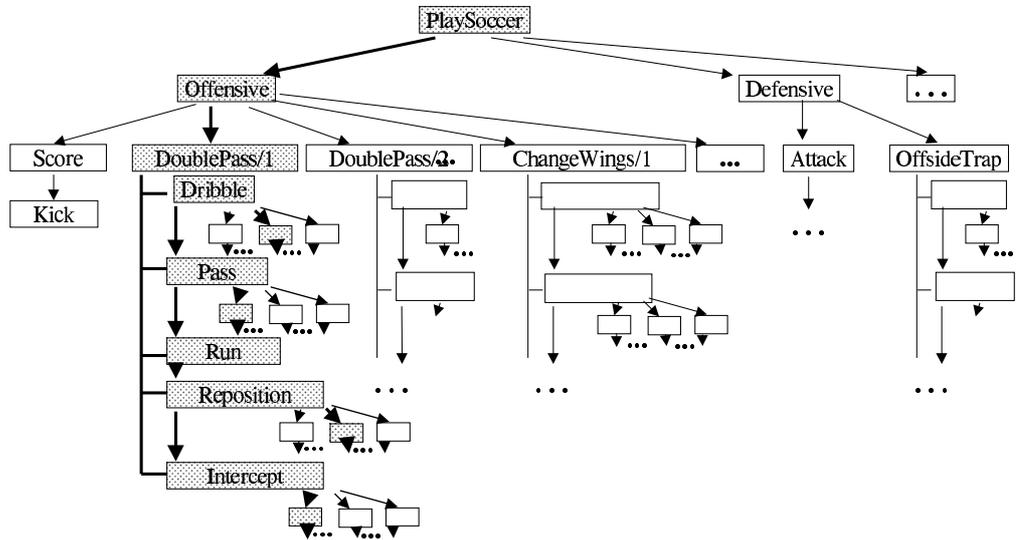
Figure 7: Intention Subtree as part of the Option Tree

of other player's behavior. Instead we can use so-called standard situations.

Standard situations provide generic cases of cooperative play. Using methods from Case Based Reasoning (CBR, cf. [Lenz-et-al98]), a concrete situation can be matched to the standard situation. For example, a triggering feature for the double pass is an opponent on the way of an offensive player controlling the ball. The standard situation (the "case") provides a standard scheme ("solution") for an intention. Using CBR methods for adaptation, a concrete intention can be specified. The option hierarchy serves as a structure for describing cases (cf. Section 6).

The deliberator computes long term decisions. It can be understood as the `deliberate`-function from Figure 3.

5.3 Executor

Short time behavior should rely on the newest available data: Hence there is no place for time consuming deliberations. The advances and the drawbacks of stimulus-response approaches and layered deliberative approaches have already been discussed. Stimulus-response architectures allow for fast reactions, but cannot handle complex long term behavior, while layered deliberative ar-
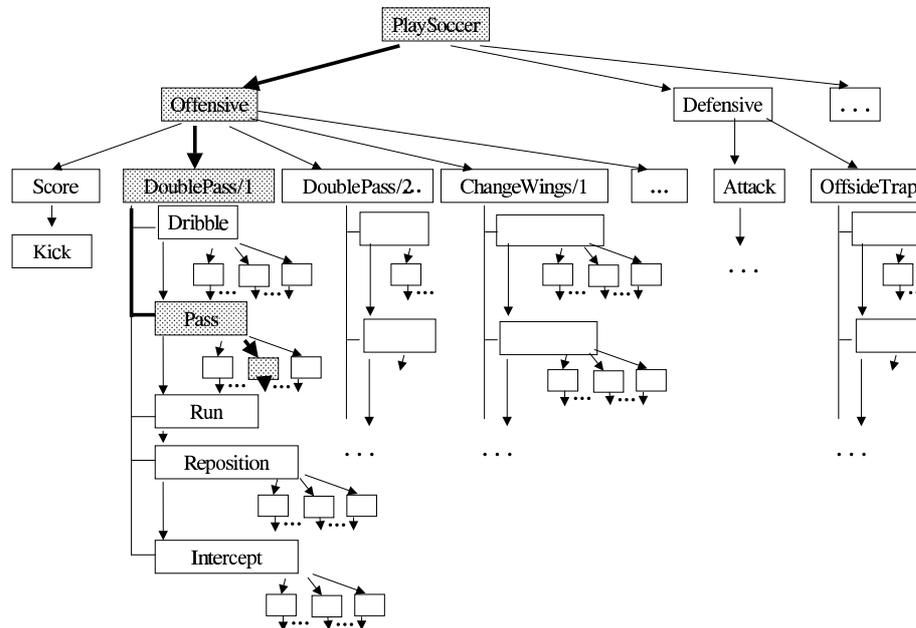
Figure 8: Activation Path in the Intention Subtree

chitectures can handle complex long term behavior, but have problems with dynamically changing situations.

The concept of the special executor pass through all layers is proposed as a solution. It works according to the recent activity path in the intention subtree. It starts from the root and proceeds level by level down to the leave which specifies the next output action to be executed by the robot. On each level it performs certain tests (e.g. if a subintention should terminate or stop), and it can calculate parameters according to the newest data (e.g. for performing an optimal kick). If a subintention is terminated, it performs the transition to the next subintention. It may also switch to a desire and make it the new intention.

It is essential that all tests and calculations of the executor can be performed in short time, and that they are performed on the appropriate level. All time consuming computations should be performed by the deliberator in time before. The structure of options must be designed for these purposes.

The executor works as soon as new actions are to be performed, and as late as the newest data relevant for these actions can be analyzed. This can be done concurrently to the work of the deliberator - which at the same time prepares and specifies later activities for the executor. In a strictly sequential approach, the executor must interrupt the interpreter. Concrete implementations are possible

in different ways, they are still in an experimental state.

The executor operates over the restricted search space of the intention tree provided by the deliberator. It can be understood as the implementation of the `execute`-function from Figure 3, but regarding all levels.

5.4 Main Features of the Double Pass Architecture

The option hierarchy allows for unique descriptions of behaviors and plans on different levels. All levels are treated the same way. An important feature of the Double Pass Architecture is the possibility of immediate reactions on all levels. It can be described as a "doubled one-pass-architecture": One-pass-architectures have a control flow which passes through each level only once. In our case, the control flow is directed bottom-down from the highest level to the lowest one. The difference consists in the fact, that there are two separated passes: One pass for the deliberator which prepares commitments (e.g. goal and plans), and another path for the executor which allows for real time reactions on all levels. The executor allows for a certain kind of stimulus-response behavior on all levels, where the stimulus-response behavior has been prepared by the deliberator. The executor realizes real-time behavior, while the deliberator acts without short time constraints.

Classical layered one- and two-pass architectures in complex dynamical environments have serious synchronization problems. Computations on higher (deliberative) layers are performed in longer time intervals, and rapid responses to changes in the environment are possible only at the lower (reactive) layers. The executor of the Double Pass Architecture works in short time intervals like the reactive components of classical layered architectures, but it passes through the higher levels, too. This is possible without synchronization problems since the deliberator prepares a restricted search space (the intention tree) for the executor.

The requirement to run through all levels by the executor needs a special runtime organization. Most runtime organization methods in programming are based on stacks, where a higher level method is called again only when the lower level has terminated. This holds for imperative languages as well as for descriptive ones, and it is used in agent architectures, too. The implementation of the new runtime strategy is still under work.

6 Conclusion, Further Work

The paper has discussed different aspects of basic approaches for robot/agent control. The notions of persistent states (concerning the past and the future, respectively) have been identified as characteristic concepts. Different approaches can be classified along these lines. They provide more clear differences than the classical notions of reactive and deliberative behavior. The classification helps to identify the problems of real time controls in dynamical environments when long term planning is involved. The Double Pass Architecture was proposed

to avoid the difficulties of layered architectures in dynamically changing environments. It allows for fast adaptations to new situations even on the higher levels.

Future plans include the use of the new architecture for robot learning. The project "Architectures and Learning on the Base of Mental Models" of the research program 1125 "Cooperating teams of mobile robots in dynamic and competitive environments" granted by the German Research Association (DFG) investigates the usage of CBR methods for control of robots. The cases correspond to generic behavior which can be specified and adapted according to the current situation. There are two main goals for using CBR: The first goal is the efficient control, while the second goal is learning from experience. Learning can be twofold: New cases can be acquired as templates for behavior, and the usage of existing cases can be improved by better analysis and adaptation methods.

There have already been some attempts to use CBR-methods for Robot Control, e.g. for opponent positioning models [Wendler/Lenz98], and for localization [Wendler et. al.00]. The investigations in opponent modeling have discovered a problem of dynamics when using CBR for low level behavior: As soon as the team tries to adapt to the opponents positions, the opponents did change to other positions. In the consequence, we need adaptation to higher level strategies. The option hierarchy serves as a structure for describing higher level cases. It gives room for off-line learning as well as on-line learning.

References

[Arkin1998] Arkin, R.C.: Behavior Based Robotics. MIT Press,1998.

[Bratman87] M.E. Bratman. *Intentions, Plans, and Practical Reason.* Harvard University Press, Massachusetts, 1987.

[Brooks91] Brooks, R.A.: Intelligence without reason. Proceedings IJCAI-91, 569-595.

[Burkhard00] H.D. Burkhard: Software-Agenten. In: Günther Görz, Claus-Rainer Rollinger, Josef Schneeberger: Einführung in die Künstliche Intelligenz. Oldenbourg 2000, 941-1015. (In German)

[Burkhard *et al.*98] Burkhard, H.D., Hannebauer, M. and Wendler, J.: Belief-Desire-Intention Deliberation in Artificial Soccer. *AI Magazine* 19(3): 87–93. 1998.

[dMARS] dMARS: Technical Overview - 25 JUN 1996.
www.aaii.oz.au/proj/dmars_tech_overview/dMARS-1.html

[Georgeff/Kinny97] Georgeff, M.P.; Kinny, D.N.: Modeling and Design of Multi-Agent Systems. In: M"uller, J.P.; Wooldridge, M.J.; Jennings, N.R. (Hrsg.): Intelligent Agents III, LNAI 1193, 1–20, Springer-Verlag, 1997

[Georgeff/Lansky87] M.P. Georgeff, A.L.Lansky: Reactive reasoning and planning. Proc. AAAI-87, 677–682, 1987.

[JACK] `www.agent-software.com.au`

[Kitano-et-al-97] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. RoboCup: A challenge problem for ai. *AI Magazine*, 18(1):73–85, 1997.

[Lenz-et-al98] Lenz, M., Bartsch-Spörl, B., Burkhard, H. D., Wess, S. (Eds.): Case Based Reasoning Technology. From Foundations to Applications. LNAI 1400, Springer 1998.

[Maes90] Maes, P. (Hrsg.): Designing Autonomous Agents. Theory and Practice from Biology to Engineering and Back. MIT Press 1990

[JPMüller96] M"uller, J.P.: The Design of Autonomous Agents – A Layered Approach. LNAI 1177, 1996.

[Müller-Gugenberger/Wendler1998] Müller-Gugenberger, P. and Wendler, J.: *AT Humboldt 98 — Design, Implementierung und Evaluierung eines Multi-agentensystems für den RoboCup-98 mittels einer BDI-Architektur*. Diploma Thesis. Humboldt University Berlin, 1998.

[Murphy00] Robin R. Murphy: Introduction to AI. MIT Press, 2000.

[Parunak97] Van Parunak: 'Go to the Ant': Engineering Principles from Natural Agent Systems. Annals of Operations Research, 1997.

[Rao/Georgeff91] A. S. Rao and M. P. Georgeff. Modeling agents within a BDI-architecture. In R. Fikes and E. Sandewall, editors, *Proc. of the 2rd Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'91)*, 1991.

[RoboCup] RoboCup. The Robot World Cup Initiative: `www.robocup.org` Annual Proceedings of the RoboCup Workshops/Symposia appear in the Springer LNAI-Series.

[Russell/Norvig95] Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice-Hall, 1995.

[Weiß99] Weiß, G. (Hrsg.): Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence, MIT Press 1999.

[Wendler et. al.00] J.Wendler, S.Brüggert, H.D.Burkhard, H.Myritz: Fault-tolerant Self Location by Case Based Reasoning. In P.Stone, T.Balch, G.Kraetzschmar (eds.): RoboCup 2000: Robot Soccer World Cup IV. Springer LNAI 2019, 259-268.

[Wendler/Lenz98] J.Wendler and M.Lenz: CBR for Dynamic Situation Assessment in an Agent-Oriented Setting. In D.Aha and J.Daniels (eds.): Proc. of the AAAI-98 Workshop on Case-Based Reasoning Integrations, 1998, Madison, USA.

[Wooldridge99] Wooldridge, M.: Intelligent Agents. In [Weiß99], pp. 27–78.