AT-Humboldt 2003 – Team Description

Ralf Berger, Michael Gollin, and Hans-Dieter Burkhard

Humboldt University Berlin, Department of Computer Science,
berger|gollin|hdb@informatik.hu-berlin.de,
http://www.ki.informatik.hu-berlin.de/RoboCup

Abstract. In this report we want to introduce the new aspects of our soccer simulation team AT-Humboldt [1] compared with former approaches [2–4]. More than 1.5 years ago we began a complete reimplementation of our team with major innovations in software design and the agents architecture. The main goal was to implement and to evaluate the double pass architecture (DPA) [5] in a highly dynamic environment. The result of this work is AT Humboldt 2003, that is now an usable testbed for long-term deliberation and real-time reasoning, cooperative behavior in MAS, CBR-aided decision making and scalable behavior control mechanisms. In addition to participating in RoboCup competitions, we are successfully using our agent system in education for various aspects of MA-issues.

1 Introduction

Our group's general research focus encompasses agent-oriented techniques, case-based reasoning, knowledge management, intelligent robotics, cognitive science and socionics. We investigate agent architectures and deliberation concepts that allow to build rational, scalable, dynamic and cooperative multi agent systems, with applications in e-commerce, medicine and industry [6],[7],[8].
We have found RoboCup to be an interesting and challenging domain for the development of new techniques and we participate in two leagues: the soccer simulation league and the Sony four-legged robots league.
We are members of the DFG (German research foundation) programme "Cooperating teams of mobile robots in dynamic environments"[9], with a particular focus on agent architectures. A lot of our work in RoboCup is strongly linked to current work in the fields of our other research topics, for instance: Case Retrieval Nets as a mean for efficient and flexible retrieval in Case Based Reasoning, Social modelling with multi agent systems or Belief-Desire-Intention models (BDI [10]) to go beyond emergent and allow cooperative behavior.
In return ideas from soccer team development have often been fruitful for projects outside RoboCup, for example in cognitive robotics. Furthermore our work in RoboCup is a great teaching platform for practical exercises in our AI and robotics courses.

2 Development of AT Humboldt

Our AI lab is participating in RoboCup since 1997, when we became world champion with 'AT Humboldt 97'. Since then we made a lot of experiences in the fields of Agent oriented techniques (AOT), behavior control and managing a large, evolutionary software project. Without these experiences we had not been able to write this years agent. We called our team an evolutional software project, meaning that the software itsself but also the concepts and even the programmers are subject to permanent change and further development. In spite of constant care, coding conventions and accompanying documentation, software tends to become slow, complex, redundant, inflexible und bad maintainable over the years. Same here with AT Humboldt, what basically was a constant extension of 'ATH 98'.
RoboCup 2001 in Seattle was crucial to do a break. Our performance was far away from the top teams. The software design as well as the behavior architecture had come to their limits. End of 2001 we began to rewrite our team from scratch. To increase the life time of the code and to have a stable and durable platform for Robocup competition, education and research, development was done under some strict paradigms:

- every functional piece of code has to be checkable for correctness
- documentation is done in parallel to code writing
- following a consequent object oriented design methodology
- all major agent tasks should be modular and data interchange has to occur only via clear interfaces
- all approaches should be easily scalable
- software performance has to be ensured by accompanying profiling
- administrative code is written by automatic code generation

All the code is completely self developed, which means that we could take care of subsequent needs very early in the design process. Just before RoboCup 2002 we have had an early version of the new agent with the first implementation of the double pass architecture [5] and the bulk of the codebase for 2003. As the overlying behavior was very rudimental and could not been tested in time, we decided not to use the new agent in the 2002 competition. So the new AT-Humboldt agent competed the first time in the 2003 events.

3 Worldmodel

Since the worldmodel of this agents predecessor didn't leave much room for further improvement, only small changes have been made to the main algorithms. The most important one was a new algorithm used for merging seen players into an existing situation. The old approach was based on a set of simple rules. Unfortunately these rules did not cover all relevant cases and thus led to faulty situations. The new algorithm transforms the problem into finding the optimal matching in weighted bipartit graphs. The nodes of the first partition represent

the existing players while the nodes in the second partition represent the seen players. To reduce the complexity of the graph, edges are only inserted if the distance between existing and seen player is less then the distance the existing player could move maximal. The distance between both players is used as the edge's weight. The operation starts with an empty matching and is incrementally increasing it by finding an augmenting path with minimum weight, which will be reversed. It is based on the Dijkstra algorithm for finding shortest pathes in a graph. For the implementation we used Fibonacci heaps which significant reduced computational costs.
Another improvement was made with regard to the ball tracking. The method is based on ideas by UvA and was extended by taking care of a few more special cases. In addition we used some heuristic functions for calculating the balls position and velocity based on simulated and seen data. As a result we observed a significant smoother ball trajectory which had a major impact on our intercept behavior, namely the number of faulty intercepts.

4 Architecture

As we mentioned above, our most important interest was to implement the double pass architecture (DPA) and having a testbed for exploring various aspects of deliberative decision making. First we will give reasons for estabilishing a new control architecture and explain the special demands in RoboCup.
The domain of robotic soccer allows the study of the requirements for control architectures with respect to incomplete, possibly incorrect environmental information, limited communication and restricted computational resources under realtime conditions. However, these constraints, as demanding as they are, do not automatically foster the development of architectures that go beyond well-tuned, reactive behavior. Even in the simulation league, which tends to spawn the most sophisticated control architectures in the RoboCup domain, most successful approaches work according to principles that can be classified as reactive or are related to classical hybrid control architectures known for example from unmanned ground vehicles (UGV). If we consider actual soccer games, we will find that human teams go far beyond reactive behavior and follow actual and non trivial long-term strategies, of course. Even if it is possible to set up reactive agent behavior in such a way as to achieve emergent maneuvers such as occasional pass chains or wing changes, complex strategies are very difficult to introduce and control intentionally. As classical hybrid architectures are organized as layered architectures with low-level reflex behaviors and high-level planning layers, some restrictions follow also from that structure: Realtime requirements apply only to the basic behaviors not to the higher levels, and the time horizon and the number of behaviors are not scalable.
Because we believe in the benefit and the need of deliberation in RoboCup and other robot tasks in dynamic environments we introduced a new control architecture called double pass architecture. This architecture is able to fulfill the following demands:

- time independent long-term deliberation of complex behavior with free time horizon
- least commitment of all possible data to execution time
- scalability in number and complexity of behaviors
- realtime capability on all control layers, even realtime changing/replanning of high level goals
- natural persistency of goals instead of artificial stability constraints
- control of coordinated behavior involving more than one player

We will now describe the main ideas and some implementation details.
Just like in former approaches our architecture is based on the notions of belief, desire and intention (BDI [11, 12]). Main idea is a strict separation of a static structure that is a symbolic description of the agents abilities and a dynamic control procedure. This control mechanism is further splitted into two separated top-down passes for deliberation and execution, hence we call it "*double pass architecture*" (DPA).
Flow of data and control compared to classical 2–pass layered architectures is shown in figure 1.

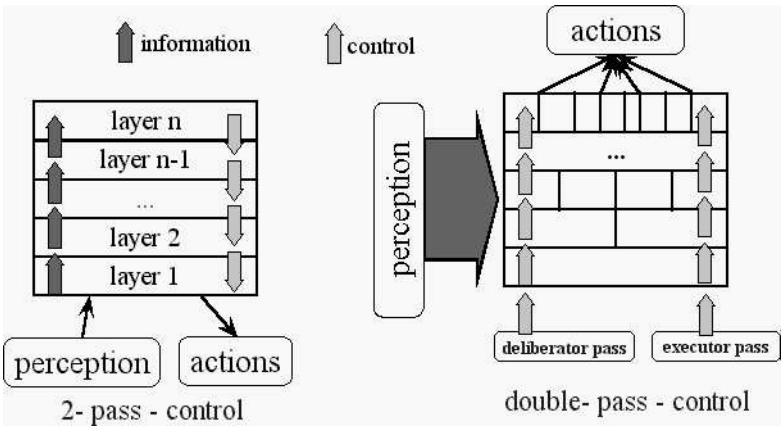

Fig. 1.

Main element of the DPA is the option tree that includes the principal modes of behavior as well as all information for the run-time control. The nodes are called options and are our basic data structure. These options represent specific behaviors on different levels of abstractness. The idea of complex behavior as composition of simpler ones gives us a natural hierarchical tree structure, whith abstract long-term options near the root and more specific short-term options near the leafs.
We distinguish between three different kinds of options:

- *basic options* – directly address skills that are implemented in the agent and generate soccerserver commands.
- *choice options* are complex behaviors that can be realized through one of their suboptions. Planning and commitments means evaluating alternatives and bind strategic parameters. Because of the hierarchical structure decision making can be performed localy in the given context.
- *sequence options* are complex behaviors that are realized through the sequencial execution of their suboptions. The transition criteria between the suboptions is time critical and by that a least commitment task.

Suboptions of choice- and sequence-options are again any of the three option types. As mentioned above we have two control processes that independently run through all levels of the option hierarchy. The main reason for this distinction is complex, time consuming deliberation can not be realized in real time (at least if we leave the 100ms steady–state–world of the soccerserver and move towards event-based simulation or high-resolution time modelling) and hence has to be performed in an asynchronous, independent process.

The **Delibarator** performs all the time-consuming analysis and long-term planning. Deliberation starts in the rootoption and evaluates all available suboptions by analyzing the current or later game-situation to determine their associated utility. This selection is based on the concept of bounded rationality [11]. The result is a pre-arranged partial plan – a set of evaluated options in the tree, that corresponds to desires and intentions in the BDI-methodology. Following the least commitment idea, the plan is continuously updated and completed as time goes on. If this is not necessary the deliberator provides alternative options/plans that are instantly available if an exception occurs at execution time. To allow for extensive analysis, the deliberator is independent from the actual run-time demands of the architecture and implemented asynchronously so it can be interrupted by other components of the agent that have to work in real-time. It is not necessary to call the deliberator between all execution steps, however, it has to be ensured that at least one valid option is available for execution at all times, to make sure the agent does not come to a halt.

The **Executor** is called whenever a timer component decides that it is necessary to perform an action. From the root node it traverses the activation path laid out by the deliberator to find an executable basic-option. Along this way the executor checks all the pre-, post- and break-conditions and resolves abstract parameters and targets to actual values. Based on the preparatory work of the deliberator, the executor has to perform only a minimum of computational work which needs the most recent sensory information. By this means the real-time contraint of generating actions can be fullfilled – the average executor runtime during RoboCup 2003 was lower than 1 millisecond.

Both processes interact only via manipulating the state of the tree. Additionaly the executor can force redeliberation because of special exceptions.

Having both passes browsing through all levels of the hierarchy avoids the necessity to define all conditions and evaluators at the level of the elementary options. This way, options can be modular and reused at different positions of the option tree.
An overview of architecture and the course of control is shown in figure 2.

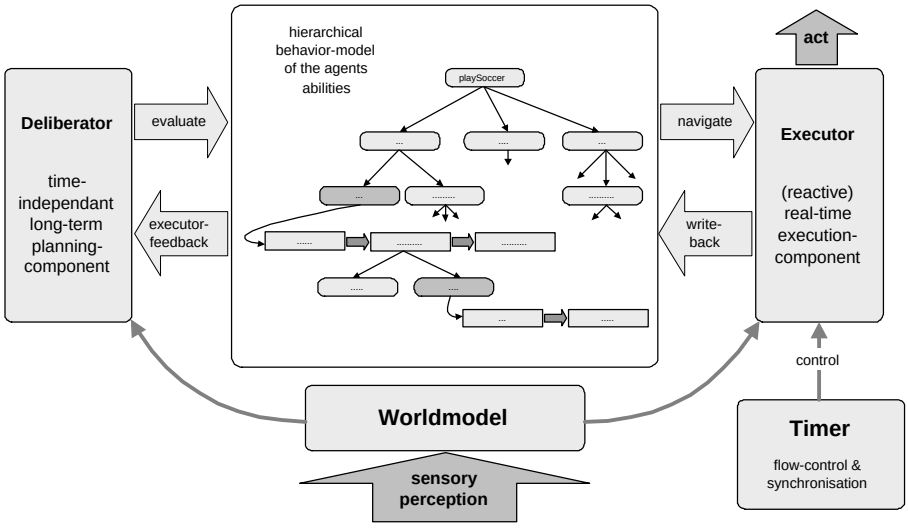

Fig. 2.

5 Behavior Description

We found that standard programming languages are not ideally suited for the definition or description of complex behavior nets and plan structures, because the resulting code is typically not very instructive, inflexible, vulnerable to bugs and not transferable between different architectures. The double pass architecture gives us the possibility to have a symbolic representation of all possible behavior modes so we should exploit this feature to have a human readable definition, where even non-developers can easily understand and change the defined behavior. Therefore, we have implemented a new description language based on XML [13], that fits very well with our architecture.
We use XML because it is an open standard that is available on practically all computing platforms, a lot of tools and libraries are available and it has a powerful background of xml-related technologies like xsl, DOM, schema and many

more. XML documents provide a natural hierarchy of all elements which make them perfect for describing tree-structures.
The description of a player's role (our root element) spans a tree of options that includes the principal modes of behavior (basic-, choice- and sequence-options) as well as all the criteria for the control processes and interaction policies. Every option, according to its type, can be further specified by a variety of other elements, for example:

- targets define abstract symbolic values, such as positions on the field or interaction partners, that are resolved to actual values at the time of execution
- conditions (pre-, post- and break-conditions) specify, under which circumstances options are started, executed or aborted. Conditions can be defined on different levels of worldmodel aggregation and combined with boolean functions.
- free parameters to use general elements in different contexts
- analyzer and evaluators are used to select possible options or sort them by preference, depending on utility-functions, cost or success probability
- selection constraints influence the behavior in a given context according to the intentions of the developer, for instance by restricting the maximal utility of an option.
- policies that control the communication-, attention- or neck-behavior

To follow ideas of object oriented software design we made it possible to provide arbitrary descriptions of subtrees or single options in own libraries, where they can be referenced everywhere and further specified and even be reused in other own provided subtrees. This made the actual role-description much more compact and led to shorter development time and less errors. Using very simple styleshets it is possible to visualize the option tree in different resolutions to get an overview of the agents behavior possibilities.
The roles we used in RoboCup 2003 contained more than 200 different options and about 2000 control elements for every player.

```
...
<choice-option name="protect ball"
               max-utility="1.0"
               evaluate-pre-conds="true"
               control-mode="RETAIN"
               say-strategy="BALL_AND_CALLER"
               min-activation="0.1"
               activation-bonus="0.05">
  <neck behavior="TO_BALL" align="OSCILLATE" width="NARROW"/>
  <analyzer name="CLEAR_BALL" caller-pass="DELIBERATOR"/>
  <pre-condition>
    <or>
      <basic-condition type="kickPossible">
        <parameter name="type" value="MYSELF"/>
      </basic-condition>
```

```
      <basic-condition type="tackle">
        <parameter name="minProb" value="0.3"/>
      </basic-condition>
    </or>
  </pre-condition>

  <option-ref ref="tackling-defender" max-utility="1.0"/>

  <basic-option type="kick" name="ClearBall" max-utility="0.1">
  ...
```

Fig.3. sample behavior code fragment of a defender-role

We also use an xml-based language to specify what we call formation. This includes all kinds of team positioning data in standard or running game situations, preferred passes and run-directions for every player-role. Also the tasks of every player within the team are set as well as special tasks for example player associations for defense, goal attacks or covering. This strategic data has a significant influence on the teams gameplay, so finetuning can do much improvement on the teams performance. Because of the huge amount of this numerical data, this is hard and fault susceptible work. Therefore we developed a set of xsl-stylesheets that transform the raw xml-data directly to an interactive graphical visualisation of the soccer-field and the corresponding impact of the data to the team or single players (uses svg and ecma-script). This makes it possible for us to make strategic changes to the team-play in seconds and thus to adapt to the opponents performance.

Both languages (for roles and formations) are specified with xml-schema [14], which enables us to do complex constraint checking/validation even during writing the behavior.

6 Case Based Reasoning

As mentioned above, AT-Humboldt 2003 also makes use of case-based-reasoning [15] techniques. [1] Apart from former approaches [16] it's the first time we applied it to the decision making process. As far as we know AT-Humboldt is the first simulation team that uses CBR for behavior control.
For applying cbr-control we looked for a special well defined player task, that in real soccer is known to be a matter of experience. The problem also should be quite simple in output for the first attempts. We decided to experiment with a problem in goal-defending:
When an opponent attacker that possesses the ball dribbles towards our goal, the

[1] Much of the conceptual work and most of the implementation was done by Christina Bell.

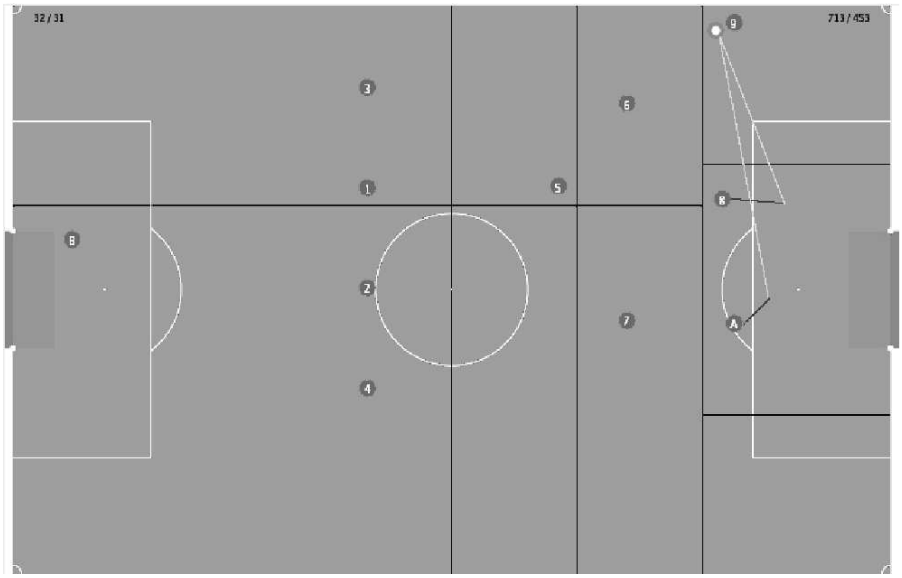

Fig. 4. screenshot of some visualized formation data

goalie has to decide whether to stay in front of the goal and defend the goal-line or to run towards the attacker to decrease the possible shooting angles to the goal. This decision depends on various parameters. If the Attacker is alone and no defender is able to support the goalie, it's quite obvious that the goalie has to run out of his goal. On the other hand he has to stay inside of the 5m-area if an defender can intervene in time. Most of the real game situations can't be classified so easily. Much is depending on the special abilities of goalie, attackers and defenders and often only minor changes in the situation require a different goalie-behavior. In real soccer the most important decision factor is the goal keeper's experience in such situations.
We will now explain our approach more detailled.
We used a set of logfiles from RoboCup German Open 2003 as initial data. These logfiles were converted with the rcg2xml-tool which is part of the rcsslogplayer and further transformed to allow a highly performant xml-processing. Next we wrote some xsl-stylesheets that automatically grep these logfiles for situations in which a goal attack is running and the goalie has to decide what to do. From each of such situations and the 100 following game steps a case was generated which basically contained the positions of the players and the ball, the balls velocity, the decision of the goalie regarding the discussed scenario and the success of this behavior. After rejecting very similar cases we had some hundred files for further working. With the same technique we used for visualizing formation data we can easily browse through this case base.
A challenging problem was to find a suitable similarity masure. After experi-

menting we decided to use a combination of an inverse distance, implemented as a manhattan-distance on a special grid, and a relevance function that provides a rating of the estimated impact of a players position on the goalie decision. Because an intuitive search-retrieval through the database is far away from realtime computing, we had to to build up a special index over the distance grid to have high speed access to similar cases to a given game situation.
During a game if a certain trigger is activated the goalie looks in his case database for akin situations and the appropriate behavior and executes according to it.
This use of cbr-techniques for behavior control is still at an early stage and under study, but the approach seems to be promising especially because the whole procedure from getting the logfiles to building up the index and the runtime structure works fully automatic and lasts on a 1.4GHz maschine just a few hours.

7 Coach

Team AT-Humboldt uses the coach-client since RoboCup 2001. Nevertheless it offers only basis functionality and up to now makes no use of clang.
Implemented features include:

- Heterogeneous player types are evaluated and ranked using an heuristic function. Players are substituted according to the ranking, but only before the game. Substitutions during the game are also planned as part of stamina management and opponent adaptation, but not yet implemented.
- The types of heterogenous players used by the opponent are normally detected within the first few cycles (1–10). But because of a delay in coach–agent communication the team receives the message not before the 50th cycle.
- In standard situations such as *free_kick* the positions of players and ball are communicated without the communication delay.
- On start-up the team formation and individual player roles are set. This feature was added as a consequence of the changed regulations in RoboCup 2003, which forbade any human interaction during round robin games. To make this feature an usefull instrument, it is of course necessary for the coach to observe all other matches and to figure out the strengthes and weaknesses of other teams.

For the future we plan to make a more intense use of the coach for example by changing the teams formation during a game, selecting special trained maneuvers in standard situations or allocating the players for coordinated man to man covering.

8 Developement Tools

During the reimplementation of ATH-2003 two tools were created - ClientControl and ADT.

ClientControl is a graphical frontend for the SoccerServers rcssclient and was mainly used for developing the agents sensors (parser) and actors (commandformat, testing of commands). For the gui gtkmm was used in the first and gtkmm2 in the improved second version. Both versions are open source and can be obtained via our website.
The second tool called ADT (Agent Debug Tool) offers a variety of useful features and made a not negligible contribution to the agents development. A client-server architecture was used with ADT as server and the agents as clients. Communication is done via UDP. Originally the tool was intended to visualize worldmodel-data only, in order to make the agents decisions more transparent for the developer. Therefore an interface to the FrameView (official SoccerServer Monitor) was created that gives access to various functions, especially the drawing of geometric figures on the field. That makes it possible to draw additional 22 players and a ball on the field which show the positions believed by an agent. In addition all worldmodeldata can be presented in a table.
In normal game mode a new cycle starts every 100 ms, since this would be to fast to carefully analyze the presented data only sync mode is used with a small change in the server. Normally the server would wait for all clients to send a done-command to advance to the next cycle, but it is waiting no longer than a few seconds. That was changed so the server will wait forever or until all clients sent their command. When the ADT is used, all clients wait for the ADT to give permission (either by pressing a button or setting an auto-mode) to send the done-command. Thus a step by step game is possible.
Later the following features were added:

- By using an history-function past cycles can be reviewed to watch a scored goal again for example. The history currently includes 250 cycles, but can freely be extended. When watching past cycles all other ADT features are also present.
- SoccerServer, own agents, opponent agents and even teams can easily be started and stopped by the ADT.
- Players and ball can be moved on the field using drag and drop. The changed positions will automatically be sent to the server.
- Agents can send coloured geometric figures such as points, lines, circles, circle arcs, rectangles, plusses, crosses and arrows to the ADT. These will be drawn on the field and can be switched on and off for each agent individually.
  In addition they can send arbitrary strings to the ADT. These two features are useful when debugging new algorithms or heuristics.
- The option tree of each agent can be shown in a graphical explorer-like tree view. Therefore each agent sends its mental state every cycle after the deliberator- and the executorcall. In addition to the state of an option after deliberator and executor its activation and the states of pre-, post- and break-conditions can be monitored. This enables the developer to retrace when and why changes in the tree occured or certain behaviors are performed or not. For developing complex behavior this feature has proven its effectiveness by decreasing the debugging time to a fraction of the time needed before.

– Analyzer results based on an agents worldmodel-data or accurate data can be presented in a table or graphically. This feature is at an early stage and still under development. By now we can show for example the score-goal probability distribution for arbitrary configurations on the field.

9 Outlook

Following our last years developments we still see much room for further improvements, for instance:

– more and advanced use of case-based-reasoning for deliberation tasks
– further experiments with different similarity- and relevance-measures for our CBR issues
– using an explicit time representation in long-term partial plans of the deliberator
– providing extended control structures for option-sequences (e.g. loops)
– much more sophisticated use of the coach because of the new regulations regarding human interaction
– evaluation of some learning techniques for low-level skills
– advanced weight-function in our situation-merge algorithm
– extend our development tool by additional features

References

1. Website of team AT-Humboldt. *http://www.ki.informatik.hu-berlin.de.*
2. H.-D. Burkhard, M. Hannebauer, and J. Wendler. AT Humboldt – Development, Practice and Theory. In *RoboCup–97: Robot Soccer World Cup I*, LNAI. Springer, 1998.
3. P. Gugenberger, J. Wendler, K. Schröter, and H.-D. Burkhard. AT Humboldt in RoboCup–98. In *RoboCup–98: Robot Soccer World Cup II*, volume 1604 of *LNAI*, 1999.
4. H.-D. Burkhard, J. Wendler, T. Meinert, H. Myritz, and G. Sander. AT Humboldt in RoboCup–99. In *RoboCup–99: Robot Soccer World Cup III*, volume 1856 of *LNAI*, 2000.
5. H.-D. Burkhard, J. Bach, R. Berger, B. Brunswiek, and M. Gollin. Mental models for robot control. In M. Beetz et al., editors, *Plan Based Control of Robotic Agents*, number 2466 in Lecture Notes in Artificial Intelligence. Springer, 2002.
6. A. Hübner, M. Lenz, R. Borch, and M. Posthoff. Last minute travel application. *AI Magazine*, 21(4):58–62, 2002.
7. M.Minor and M.Lenz. Textual cbr im e–commerce. *KI*, (4), 2000.
8. I. Münch and G. Lindemann. Charitime–concepts of analysis and design of an agent oriented system for appointment management. *Fundamenta Informaticae*, (43).
9. Website of DFG Programme 1125 "Cooperating teams of mobile robots in dynamic environments". *http://ais.gmd.de/dfg-robocup/.*
10. Anand S. Rao and Michael P. Georgeff. Modeling Agents Within a BDI-Architecture. In *Proc. of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR91)*, pages 473–484, Cambridge, Mass., apr 1991. Morgan Kaufmann.
11. Michael E. Bratman. *Intention, Plans, and Practical Reason.* Harvard University Press, Cambridge, Mass., 1987.
12. Anand S. Rao and Michael P. Georgeff. BDI Agents: From Theory to Practice. In Victor Lesser, editor, *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS)*, pages 312– 319, San Francisco, CA, 1995. MIT Press.
13. Website of the xml project of the W3C. *http://www.w3.org/XML/.*
14. Website of the xml-schema project of the W3C. *http://www.w3.org/XML/Schema.*
15. J. Kolodner. *Case-Based Reasoning.* Morgan Kaufmann, Cambridge, Mass., 1993.
16. Jan Wendler, Steffen Brüggert, Hans-Dieter Burkhard, and Helmut Myritz. Fault-tolerant self localization by case-based reasoning. In Tucker Balch, Peter Stone, and Gerhard Kraetzschmar, editors, *Proceedings of the fourth RoboCup Workshop*, pages 82–89, Melbourne, Australia, 2000.