AT Humboldt
Team Description 2007

Ralf Berger and Hans-Dieter Burkhard

Department of Informatics, AI Group
Humboldt University Berlin, Unter den Linden 6, 10099 Berlin (Germany)
simteam@informatik.hu-berlin.de,
http://www.robocup.de/AT-Humboldt

Abstract. *AT Humboldt*[1] is the RoboCup Soccer Simulation team of Humboldt University Berlin. For more than ten years now we are successfully using our soccer agents as a research testbed for different fields of artificial intelligence and robotics – especially long-term deliberation and realtime reasoning, cooperation and coordination in multi agent systems, case based reasoning aided decision making and machine learning. In this report we will briefly present some interesting aspects of the *AT Humboldt* agent implementation. Further we will describe our main areas of research as well as their outcomes for the RoboCup competitions to be held in Atlanta 2007.

1 Introduction

The Soccer Simulation team *AT Humboldt*[ATH] was founded in 1997, as part of the Artificial Intelligence Lab at the Humboldt-University Berlin and participated at each of the 10 previous RoboCup world cups. Our Lab is also home of our sister projects *Aibo Team Humboldt* and *Humanoid Team Humboldt* and looks back on a long and successful history in RoboCup competitions (World Champion 1997, Vice World Champion 1998 and 2004 in the Simulation League, World Champion in the Four-legged League in 2004 and 2005 as part of the GermanTeam).

Our group's general research focus encompasses agent-oriented techniques, behavior modeling, intelligent robotics, cognitive science, case-based reasoning, machine learning and neuro-dynamics to name but a few.

The main objective of our simulation group is the development of an universal behavior architecture based on mental models that is applicable to a variety of platforms. As a second focus we investigate the usability of case-based reasoning and reinforcement learning techniques to complex multi-agent problems.

In addition to participating in RoboCup competitions, we are successfully using our agent system as a great teaching platform in practical courses on AI (machine learning, multi-agent systems) and cognitive robotics.

[1] Our work is granted by the German Research Foundation (DFG) in the main research program 1125 "Cooperating teams of mobile robots in dynamic environments".

We are members of the DFG (German research foundation) program “Cooperating teams of mobile robots in dynamic environments” [DFG].
In this report we will give a brief overview of the most important concepts and some interesting work that has been done till now.

2 Research Topics

2.1 Behavior Control Architectures for (Multi)Agent Systems

Many behavioral abilities of agents in complex multiagent systems (MAS) are crucially dependent on what is mostly denoted as the agent architecture. Even if it is possible to get quite sophisticated behavior by stimulus-response mechanisms, complex and cooperative long-term maneuvers are at least very difficult to introduce and to control. If knowledge of the world is incomplete or unavailable, it is necessary to explicitly model some kind of goals for the involved agents - in such cases reactive behavior control will often fail to achieve cooperation. On the other hand pure deliberative control shows its weaknesses in fast changing environments and under realtime conditions.
It is now widely accepted that a combination of both approaches is essential for achieving complex behavior. Most of these hybrid architectures have in common that high-level layers gain control in larger time intervals than lower layers. This behavior and the mostly stack-oriented runtime organization leads to several problems in highly dynamic environments:
In changing situations reactions are delayed for the time of re-deliberation on higher levels, although fast reaction is desirable or even essential. Furthermore invoking the different layers asynchronously can cause inter-layer inconsistencies and oscillating behavior. We give a more detailed analysis of the problems and limitations (namely context-problem, failing upwards, least commitment) of existing approaches in [Ber06] and [Bur01].
As a conclusion we have developed a new architecture for plan-based control of autonomous agents (*double pass architecture* (**DPA**)) that overcomes these problems.
The DPA is based on the principles of Bratmans Belief-Desire-Intention Theory (BDI) [Bra87]. Thereby the DPA is more faithful in realization of the fundamentals of this theory than many other BDI-architectures:
An introducing description of the architecture is given in [Bur05,Bur02,Bur01]. A detailed specification and a comprehensive discussion of the *double pass architecture* can be found in a diploma thesis [Ber06].
We will now enumerate the key-concepts of our approach:

Decoupling of Behavior Control and Behavior Model
The *double pass architecture* abolishes the strong coupling between behavior control and behavior model that is typical for common layered architectures:
Long-term deliberation and planning are no longer restricted to higher behavior

layers but cover the whole behavior. Reactivity and realtime capabilities are not only part of the lowest layers but are also features of all intentions. Therefore the structural behavior model inside the DPA is independent from the dynamic processes of behavior control. Deliberation and reaction are independent processes with a separate runtime organization that encompass the entire behavior.

Hierarchical Organized Option-Based Behavior Model

We argue that extensive individual or cooperative actions can be modeled by composing of more elementary actions, whereby composition can mean either choice of alternatives or subsequent concatenation. All these sub-behaviors can further be described as a combination of other behaviors. The result spans a tree of options with abstract long-term options near the root, like 'play soccer', and basic actuator commands, like 'kickToPos', at the leaves (figure 1) .

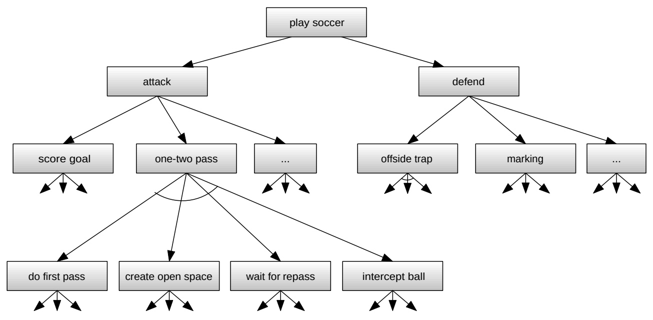


Fig. 1: Example extract of a very simple option-tree. Sequence options are denoted with an arc.

Instead of a fixed number of layers, an arbitrary number of layers is permitted in the tree, whereby the different layers itself do not have a predefined abstraction model but every node reflects a certain context (scope of admissibility for deliberation and execution). This context can be determined e.g. by a class of situations, a social goal, a role assignment or the agent's mental state. Evaluating only the possible actions within the context of one node instead of comparing all possibilities of a certain abstraction layer reduces the complexity of evaluation (local evaluation vs. global evaluation methods) and allows more specific decision making.

Independence of Planning and Acting

Like Bratman we see planning as an interactive process of guiding the (reactive) execution. Hence our approach is based on a structural separation of long-term deliberation and reactive execution (both considered on all levels). For this reason the double pass architecture uses two independent top-down passes:

Deliberator-Pass

The deliberator performs long-term planning[2] to prepare and monitor behavior

[2] not meant as the classical term of planning or means-end reasoning, but the preparation of a set of intentions, which can generate a long-term behavior

according to individual and social goals and persistent strategies. The deliberators main task is to choose the goals and to prepare all the context of the nodes which is necessary to decide how to realize these goals according to the current situation. 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. This plan is continuously updated and completed as time goes on. Additionally, the deliberator provides alternative options/plans that are instantly available if an exception occurs at execution time. The deliberator is independent from the actual run-time demands; it has to be ensured however that at any time enough information is prepared for execution of sensible actions.

Executor-Pass
The executor generates the reactive actuator commands that will fulfill the goals selected by the deliberator. The main tasks are checking the options consistency and transition conditions and resolving of symbolical data based on the most recent sensory information (least commitment). The executor is called whenever a timer component decides that it is necessary to perform an action. Based on the preparatory work of the deliberator, the executor has to perform only a minimum of computational work (the data that has been left open for least commitment) and thus can be delayed to get the latest possible perception.

An overview of the architecture components and its interactions is shown in figure 2.

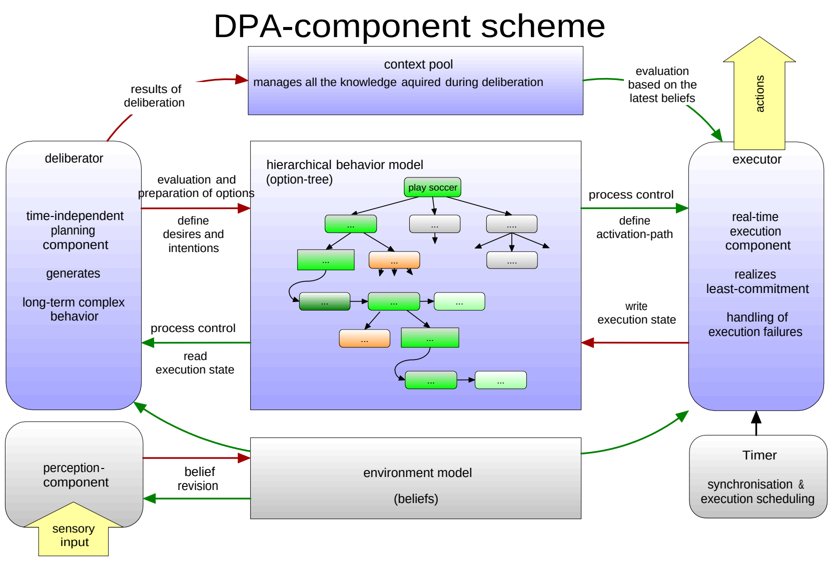

DPA-component scheme
context pool
manages all the knowledge aquired during deliberation
results of deliberation
evaluation based on the latest beliefs
actions
deliberator
time-independent planning component
generates
long-term complex behavior
evaluation and preparation of options
define desires and intentions
hierarchical behavior model (option-tree)
play soccer
process control
define activation-path
executor
real-time execution component
realizes least-commitment
handling of execution failures
write execution state
process control
read execution state
perception-component
sensory input
belief revision
environment model
(beliefs)
Timer
synchronisation & execution scheduling

Fig. 2: basic interactions between the components of the architecture

2.2 Case-Based Reasoning (CBR) for Decision Support

Selecting and initiating an appropriate (possibly cooperative) behavior in a given context is one of the most important tasks for soccer agents. The use of CBR techniques in this field was pioneered by our team already in 1997[Wen98] and amongst others continued with a case-based decision support module for the goal keeper[Ber]. We have continuously extended our work and have now developed a CBR-framework[LÖ6] that perfectly fits into the double pass architecture and that is able to select and initiate complex game plays for several players by using experience from previous situations. The system uses several additional techniques for optimizing the case base and the retrieval step in order to be efficient enough to use it in a realtime environment like robotic soccer. We have already integrated this CBR-system in our team with a first kind of game play, namely a wall pass[Lux05] – more game-plays will follow. The system was able to outperform the former hand-coded behavior selection for this particular tactics right from the start.
The most recent recent work is presented in [Ber07]. Here we will only enumerate some interesting parts:

Case Acquisition
The Simulation League provides the exceptional opportunity to access a huge repository of logfiles of already played games. We exploit this pool of experience by building up our initial case-base from these matches. Our goal is, that after the primary case acquisition, the agents start to advance and extend their case-bases by own experience. For this purpose we've developed a method for automatically analysing logfiles and extracting sequences where such game-plays actually occurred as well as 'potential'[3] *wall pass* situations.

Case Optimization
The raw case-base tends to be not optimal in in many aspects: The cases are very specific, possibly redundant, they hold unessential information and the distribution of information among the case-base is very unbalanced. Therefore we optimize the case-base in a twofold process:

First we have to extract only the significant pieces of information from each case. Based on the spatial relations between the ball and the relevant players we define up to three areas of interest. Players within these areas are seen to be relevant. Their positions are transformed, quantized and stored in the case. The deletion of the non-essential information speeds up the retrieval and leads to more general cases.

The second optimization task is to delete the redundant cases. To determine whether a case is redundant (it can be deleted without decreasing the competence of the case-base), we use the concepts of *coverage* and *reachability*. We

[3] The definition of a "potential" *wall pass* situation was followed the former hand coded selection criteria for this behavior.

experimented with several models, whereby the individual competence contribution model[Smy99] turned out to be the best. The deletion of the redundant cases shrinks the case-base significantly (from 1010 to 378 cases). Furthermore the information density of the case-base decreases and the dispersion of the information becomes more homogeneous which again speeds up the retrieval.

Case Retrieval

There are two key issues for the retrieval mechanism. Firstly, it should find

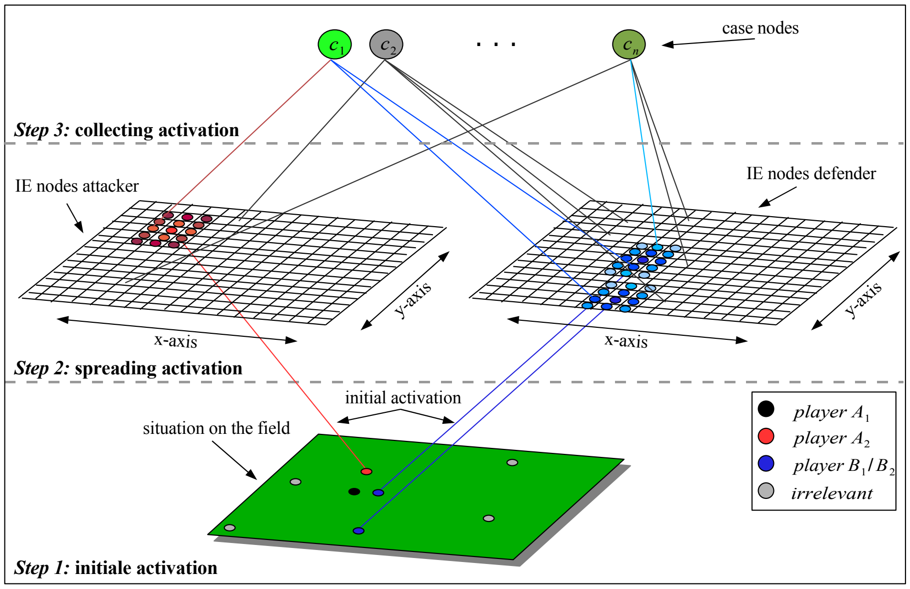


Fig. 3: Exemplary view of the Case Retrieval Net

the most similar case for a given query situation. Secondly, the retrieval needs to be fast. Since in our approach similarity is defined by degree of match of the quantified spatial features, we decided to use a Case Retrieval Net (CRN)[Len96]. CRNs provide an efficient and flexible way of retrieving a relatively small number of relevant cases out of a possibly huge case base. An exemplary view how the CRN looks like and how it works, is shown in figure 3. In *Step 1*, the IE nodes will get their initial activation, according to the relevant players in the section of the field. In *Step 2*, the initial activated IE nodes propagates their activation according to the similarity function. And finally, in *Step 3* all activated IE nodes propagates their activation to case nodes according to the relevance function. This activation will be collected in the case nodes and the case node with the highest activation will be returned.

Maintenance
Maintenance in the context of CBR usually denotes the enduring adaptation, refinement and optimization of the system (mainly the case-base), as well as remedying deficiencies, in order to ensure or improve the usability and applicability of the CBR-system. Therefore we have developed a framework for offline maintenance including an application with a graphical user interface, that supports the process of (manual) maintenance. The key features of this application are:

- Visualization of the cases
- Manual reclassification / deletion of cases
- Extraction of new cases from log fils
- Updating of the statistics (e.g. success rate or activation frequency)
- Statistical optimization of the case-base (e.g. automatic deletion of redundant and needless cases)
- Playback of actual game-plays

2.3 Integration of Soccer Simulation and Humanoid League

It has been stated several times that in order to achieve the ultimate RoboCup goal it is essential that the different leagues have to move much closer to one another. Especially in case of simulation league and humanoid league there already have been proposals for a road-map to a mid-term integration. Against this background we have started developing a physical simulation environment [Hei07a] for (humanoid) robots under consideration of the specific needs and aspects of both leagues. The simulation is based on the Open Dynamics Engine library (ODE) and simulates a simplified model of the 19-DoF *Bioloid* robot, consisting of 59 body parts and 19 servo motor joints. Several isolated motor characteristic experiments were accomplished, in order to adequately simulate the servo motors torque and friction (see fig. 4). Finally, as a weak validation of the simulation, several real robot motions were transfered to the simulated one and could reproduce similar behavior (see fig. 5).
Our main objective was the analysis of different model-free approaches for generating and controlling motion patterns on a humanoid robot – most of all biped locomotion. We found using neural oscillators for generating central oscillation patterns within a joint controlling neural network to be an interesting method for this purpose. Figure 6 illustrates the neural topology of the controller's network. Given this model, the crucial task is to identify and optimize appropriate weight parameter sets in order to generate the desired motions. For this task we have used evolutionary algorithms. We could show, that our approach is applicable in generating fast and robust walking patterns for the simulated biped robot (see fig. 7 and [Hei07b]). Our system is also used by our robotics group as well as for teaching purposes in the field of cognitive robotics and machine learning.

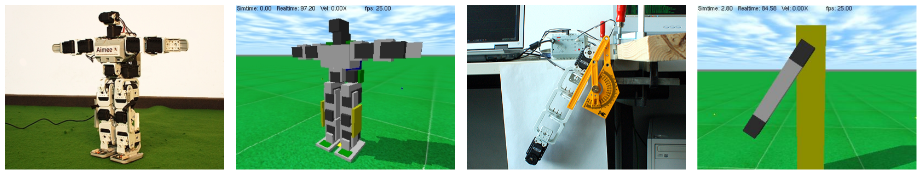


Fig. 4: Real and simulated world (left to right): Real Bioloid, Simulated Bioloid, Real servo motor torque and friction experiment setup, Simulation experiment setup.

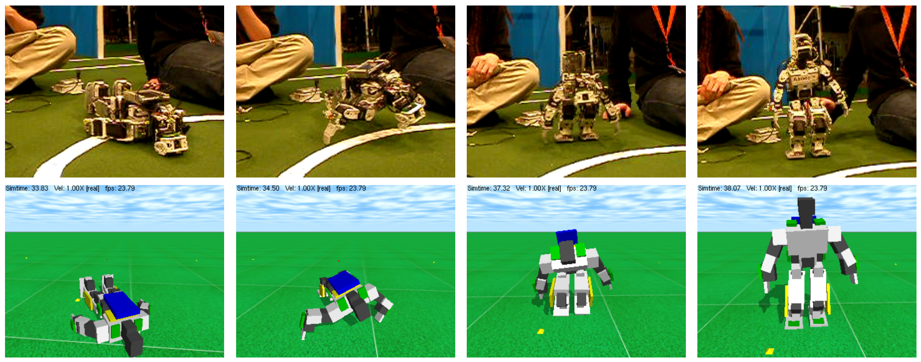


Fig. 5: A first validation of the simulation: The stand up motion was developed on the real Bioloid. The (raw) transfer to simulation shows similar behavior.

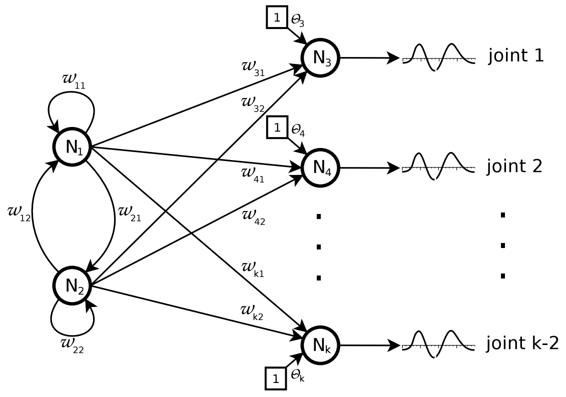


Fig. 6: Topology of the neural net controller. Each reference trajectory is controlled by a dedicated neuron, which derives its activation by the two oscillating neurons and a bias term.

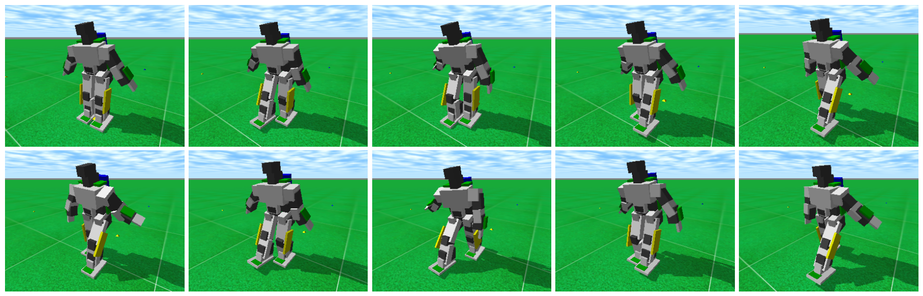


Fig. 7: Example of an evolved walking pattern. The displayed motion reaches a walking speed of about $0.45m/s$, which corresponds to a human walking speed of approx. $7km/h$.

2.4 Reinforcement Learning in Complex Domains

In 2004 we successfully started using methods of reinforcement learning (RL) for improving low-level and mid-level skills of our agents. Furthermore we developed an universal and comprehensive RL-library [RL++] as part of a diploma thesis [Gol05]. As one result the agent's dribble-skill is now about twice as fast and even more failsave than our former handcoded one. A particularly interesting aspect of the actual learning method is the use of evolutionary selection of suitable meta-actions and tile-coding settings for modeling the problem.
While trying to face more complex problems with RL methods, we increasingly felt the effects of the so-called course of dimensionality. In high-dimensional state spaces it becomes more and more difficult to find an appropriate state representation and a corresponding configuration of a value-function approximator. Therefore we started examining the use of adaptive methods for state and policy representation with help of statistical analysis of the learning process. The theoretical fundamentals of this approach are researched in a diploma thesis, that is currently in progress.

3 Team Details

In the following sections we want to briefly present some details of the team design that might be found interesting by other simulation teams.

3.1 Synchronization

In 2007 our team will use a completely redeveloped timing module. Its most important task is to synchronize the agent with the server. For this task we use an improved version of the flexible windowing method described in [dB02] which combines an adaptive internal timer (floating average of the sensed cycle lengths), external signals (body-Info) and a pretty sophisticated way of using the timing information of the see-Infos. Furthermore we are able to use specific view-modes (mainly `NARROW, LOW`) to control the desired incoming times of the server's see-messages. We did a lot of tests with the new timer and it turned out that is was much more exact in predicting the incoming times of body- and see-messages and therefore enabled us to use even late see-messages without risking actions holes.

3.2 Localization

Even if self-localization doesn't seem to be an interesting field of research in Simulation League any more, we will pick up this topic here anyway, because our algorithm differs significantly from common approaches like triangulation or particle filters. Our gradient descent algorithm makes use of the facts, that in Simulation League the vision error model is known and that a set of seen flags corresponds to a defined localization figure with an uniform probability distribution. Since we have already published our method before [Bac01], we will just give an idea here:

1. Estimate the current position from previous position and movement actions.
2. For all visible flags, compute the error limit.
3. For the current estimation, calculate the distances to all visible flags. Calculate the 'force vectors', which are given by the difference between seen distance and calculated distance.
4. If there is no force left, or the maximum number of iterations has reached, return the current estimate.
5. Otherwise, move the estimate by the longest force vector, continue with 3.

We could show that self-localization using the gradient descent method is not only outstandingly exact and very fast, but also yields much more consistency than other approaches.

3.3 Sensor Fusion

For the problem of merging seen players into an existing situation of the world-model we borrowed an algorithm from graph theory. We transforms the sensor fusion problem into the problem of finding the optimal matching in weighted bipartite graphs. The nodes of the first partition represent the existing players while the nodes in the second partition represent the seen players. 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 paths within a graph. For the implementation we used Fibonacci heaps which significantly speeds up the computation. We could show that this approach performs much better in a variety of situations compared to a rule-based approach.

3.4 Behavioral Complexity

As we mentioned earlier the DPA allows for very specific decision making and intuitive modelling complex behavior. With our old architecture we were able to model and handle only about 30 different behavior options. Since we implemented the DPA within our simulation team we could easily increase the complexity of the used behavior models from year to year. The option trees to be used this year will consist of about 500 different options and more than 6.000 control elements for every single agent. Nevertheless we were still able to understand the resulting agent behavior and could also ensure its execution in real-time ($< 1ms$).
Compared to classical SPA models the concurrent P-SA organization of the DPA can better handle varying computational resources with regard to the behavioral performance. Because the executor has to perform only a minimum of computational work and the delibarator is an any-time process by nature, shortages of available resources have only minor influence to the behavior. To proof this we did a series of test games to measure the influence of the architecture on the team performance in case of very expensive deliberation and limited resources (table 1).

UvA base	0.4 : 5.1	0.6 : 4.1	1.5 : 1.2
ATH-2005	0.3 : 1.3	0.3 : 1.0	1.3 : 0.0
	ATH-2006	ATH-2006 limited resources deliberation-execution- call-ratio: 1:3	ATH-2006 limited resources sequential SPA-flow (no DPA)

Table 1: Average results of 20 games ATH-2006 versus UvA base and ATH-2005. ATH-2006 was started with different configurations: a) normal, b) with limited computational resources that causes the deliberator to be called only every third agent cycle, c) same as b) but with sequential control flow (execution after deliberation).

Reducing the available computational power has almost no influence on the behavior performance if the concurrent control flow of the DPA is used. In contrast using a classical sense-think-act flow causes a major fall of the performance.

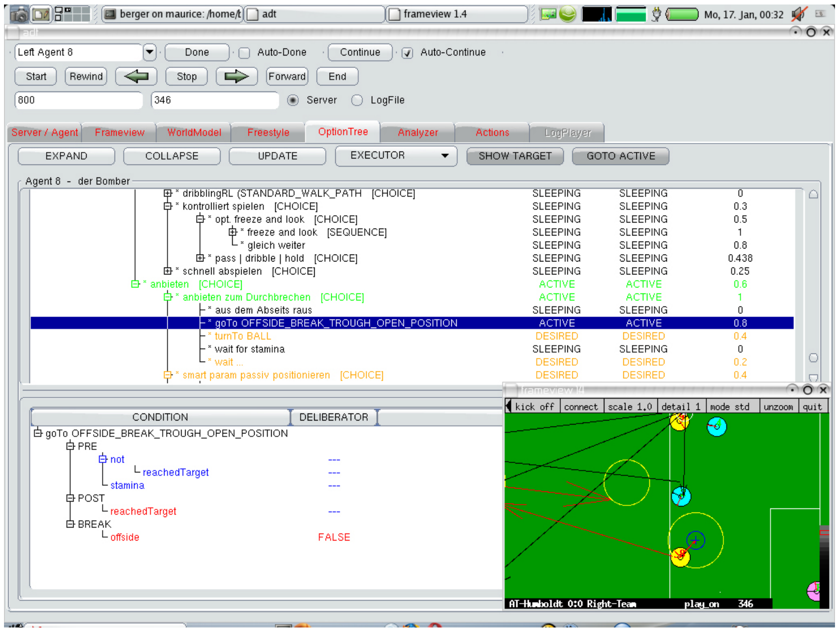
3.5 Agent Development Tool

Since 2003 we use a very powerful tool (ADT) for supporting the process of agent development. Originally the tool was intended to visualize worldmodel-data only, in order to make the agents' decisions more transparent. By adding more and more features it has now become an indispensable implement for designing and revising the agent's behavior. Key-features of our tool are:

- Playing real games step by step
- Generate arbitrary game situations on the filed
- Analyzing / visualizing all internal (mental) states of all agents

– Tracing the entire behavior control process (all decisions)
– Visualizing arbitrary data on the field

Figure 8 shows a screenshot of the ADT



```
* dribblingRL (STANDARD_WALK_PATH   [CHOICE]          SLEEPING   SLEEPING   0
* kontrolliert spielen   [CHOICE]                     SLEEPING   SLEEPING   0.3
    * opt. freeze and look   [CHOICE]                 SLEEPING   SLEEPING   0.5
        * freeze and look   [SEQUENCE]                SLEEPING   SLEEPING   1
        * gleich weiter                               SLEEPING   SLEEPING   0.8
    * pass | dribble | hold   [CHOICE]                SLEEPING   SLEEPING   0.438
* schnell abspielen   [CHOICE]                        SLEEPING   SLEEPING   0.25
* anbieten   [CHOICE]                                 ACTIVE     ACTIVE     0.6
* anbieten zum Durchbrechen   [CHOICE]                ACTIVE     ACTIVE     1
    * aus dem Abseits raus                            SLEEPING   SLEEPING   0
    * goTo OFFSIDE_BREAK_TROUGH_OPEN_POSITION         ACTIVE     ACTIVE     0.8
    * turnTo BALL                                     DESIRED    DESIRED    0.4
    * wait for stamina                                SLEEPING   SLEEPING   0
    * wait ...                                        DESIRED    DESIRED    0.2
* smart param passiv positionieren   [CHOICE]         DESIRED    DESIRED    0.4
```

```
goTo OFFSIDE_BREAK_TROUGH_OPEN_POSITION
    PRE
        not                        ---
            reachedTarget          ---
        stamina                    ---
    POST
        reachedTarget              ---
    BREAK
        offside                    FALSE
```

Fig. 8: Agent Development Tool (ADT) showing a live fragment of the behavior tree and the internal state of the double pass architecture

4 Conclusion and Outlook

For the last 10 years we successfully used our agent system *AT Humboldt* for interesting and fruitful research and as a great teaching platform resulting in many publications, diploma and PhD thesis'. While much work for 2007 is still in progress, a detailed description of new results will appear in a final team report.
Some still unmentioned issues we want to tackle for this years RoboCup are:

– Extending the use of CBR supported decision making to more game-plays and standard situations
– Extending the maintenance tools in a way that the agents share and integrate their experiences self-controlled after each played match
– Extending the use (and number) of local agent roles for specific tasks
– Using reinforcement learning for a broader range of agent tasks

References

Bac01. J. Bach and M. Gollin. *Self-Localization Revisited*. In *RoboCup 2001: Robot Soccer World Cup V*, vol. 2377 of *LNAI*, pp. 251–256. Springer: 2001.

Ber. R. Berger, M. Gollin and H.-D. Burkhard. *AT Humboldt 2003 – Team Description*. In *RoboCup 2003 - Proceedings of the International Symposium*.

Ber06. R. Berger. *Die Doppelpass-Architektur – Verhaltenssteuerung autonomer Agenten in hochdynamischen Umgebungen*. diploma thesis, Institut für Informatik, Humboldt-Universität zu Berlin: 2006.

Ber07. R. Berger and G. Lämmel. *Exploiting Past Experience – Using Case-Based Techniques for Decision Support in Soccer Agents*. In *RoboCup 2007 - Proceedings of the International Symposium (submitted)*: 2007.

Bra87. M. E. Bratman. *Intentions, Plans, and Practical Reason*. Harvard University Press: 1987.

Bur01. H.-D. Burkhard, J. Bach, R. Berger, B. Brunswiek and M. Gollin. *Mental Models for Robot Control*. In M. B. et al. (ed.), *Advances in Plan-Based Control of Robotic Agents*, vol. 2466, pp. 71–88. Springer: 2001.

Bur02. H.-D. Burkhard. *Real Time Control for Autonomous Mobile Robots*. *Fundamenta Informaticae*, vol. 51:pp. 251–270: 2002.

Bur05. H.-D. Burkhard. *Programming Bounded Rationality*. In *Proceedings of the International Workshop on Monitoring, Security, and Rescue Techniques in Multiagent Systems (MSRAS 2004)*, pp. 347–362. Springer: 2005.

dB02. R. de Boer and J. R. Kok. *The incremental development of a synthetic multi-agent system: the UvA Trilearn 2001 robotic soccer simulation team*. Master's thesis, University of Amsterdam: 2002.

Gol05. M. Gollin. *Implementation einer Bibliothek für Reinforcement Learning und Anwendung in der RoboCup Simulationsliga*. Diploma Thesis, Institut für Informatik, Humboldt-Universität zu Berlin: 2005.

Hei07a. D. Hein. *Simloid – Evolution of Biped Walking Using Physical Simulation*. Diploma thesis, Institut für Informatik, Humboldt Universität zu Berlin: 2007.

Hei07b. D. Hein and R. Berger. *Evolution of Biped Walking Using Neural Oscillators and Physical Simulation*. In *RoboCup 2007 - Proceedings of the International Symposium (submitted)*: 2007.

LÖ6. G. Lämmel. *Fallbasierte Verhaltenssteuerung im RoboCup*. Diploma Thesis, Institut für Informatik, Humboldt Universität zu Berlin: 2006.

Len96. M. Lenz and H.-D. Burkhard. *Case Retrieval Nets: Basic Ideas and Extensions*. *Lecture Notes in Computer Science*, vol. 1137:pp. 227–239: 1996.

Lux05. J. A. Luxbacher. *Soccer-3rd Edition - Steps to Success*. Human Kinetics: 2005.

Smy99. B. Smyth and E. McKenna. *Building Compact Competent Case-Bases*. *Lecture Notes in Computer Science*, vol. 1650:p. 329: 1999.

Wen98. J. Wendler and M. Lenz. *CBR for Dynamic Situation Assessment in an Agent-Oriented Setting*. In *Proc. AAAI-98 Workshop on CaseBased Reasoning Integrations*: 1998.

ATH. *Website of AT Humboldt Soccer Simulation Team* [online]. Available from: http://www.robocup.de/AT-Humboldt.

RL++. *Website of RL++ – Open Source C++ library for Reinforcement Learning* [online]. Available from: http://sourceforge.net/projects/rl-pp/.

DFG. *Website of the DFG Main Research Program 1125* [online]. Available from: http://www.ais.fraunhofer.de/dfg-robocup/.