KickOffTUG - Team Description Paper 2007

Stephan Gspandl, David Monichi, Michael Reip, Gerald Steinbauer, Máté Wolfram, Christoph Zehentner *

Institute for Software Technology, Graz University of Technology, Inffeldgasse 16B/II
8010 Graz, Austria,
stephang@sbox.tugraz.at,
http://kickofftug.tugraz.at

Abstract. This paper presents the RoboCup Simulation League Team KickOffTUG, in particular its aims and research topics. As RoboCup in Austria is young we work on the development of the Austrian RoboCup community.
Our research focus comprises agent architectures, decision making and behaviour modeling. Therefore, we present a general architecture for robotic control and its concrete implementation on our agents. Moreover, the topics of decision-making and automatic execution analysis in dynamic and uncertain environments are discussed. We propose a decision-making algorithm adapting the concepts of teleo-reactive programs and an analysis approach based on automaton-encoded meta-knowledge.
We conclude with a description of the proposed architecture as an education and research platform for various levels and give an overview over our future research activities.

1 Introduction

The KickOffTUG-team was founded in 2004 and participates in the 2D RoboCup Simulation League (RSL). In the initial phase the team comprised three bachelor students doing their theses. Meanwhile it provides a field for master and further bachelor theses, practicals and a wide range of research activities.
Our general research focus comprises agent architectures, decision making, planning, learning, behaviour modeling, i.e. the formalization of soccer knowledge, and several issues of low-level robotic control.
Therefore, we have developed a two-level architecture comprising a general decision-making level and a RoboCup-specific execution level. Thereby, we separate high-level behaviour modeling along with decision derivation, planning and learning from low-level robotic issues, extending the RSL to an education and research platform as each academic point-of-entry can be regarded independently without the need for understanding the internals of other components.
As robust decision-making is another important field of research we have developed a special decision-making algorithm which adapts the TR-formalism (proposed in [1]) and operates on a transformation of TR-programs (regarding

* Authors are listed in alphabetical order

STRIPS instances as proposed in [2]) to trees.
During the development of our project we have always been working on the problem of execution analysis and formal verification in the dynamic soccer server environment and thus suggest the use of a automaton-based meta-description of actions. This allows us to detect faults and to measure the performance (in terms of success and failure distinguishing various types) of each implementation in contrast to other teams.
In the following we will use the term 'decision-making' to describe the retrieval of the best possible, executable action, which will in this context also be called 'decision', whereas the 'execution of a decision' means the execution of the implementation associated with a retrieved action.

2 RoboCup in Austria

RoboCup is relatively young in Austria as the first teams were created in 2002. Therefore, one of our major concerns is the promotion of the RSL in Austria. Due to the high abstractness of the simulated game this league has a major disadvantage, which is a virtue as well. On the one hand its academic ambitions and research activities are especially hard to convey to the public, but at the same time the RoboCup Simulation League provides an ideal starting point for engagement in RoboCup. In order to foster the development of RoboCup in Austria we founded the Austrian National Committee together with two other Austrian teams in 2006.
The participation in joined public events as well as organizing own RSL specific activities provides another great opportunity for the team to address and attract a growing number of professionals, researchers, students and even a large general audience.
Furthermore we plan to promote RoboCup (esp. the RSL) in schools, thus ensuring future interest in competitions and robotics in Austria.

3 A two-level architecture

3.1 Overview

We have developed a general Java-based architecture with two separate layers: a state-of-the-art low-level base on the execution-level for RoboCup specific tasks as well as a high-level control system on the decision-making-level which uses Prolog as logic back end for derivation of decisions.
Figure 1 shows a general system overview with focus on the two-level separation.

3.2 Execution Level

The Execution Level (EL) comprises the three main components WorldModel, Execution and Conditions along with further packages containing functionality

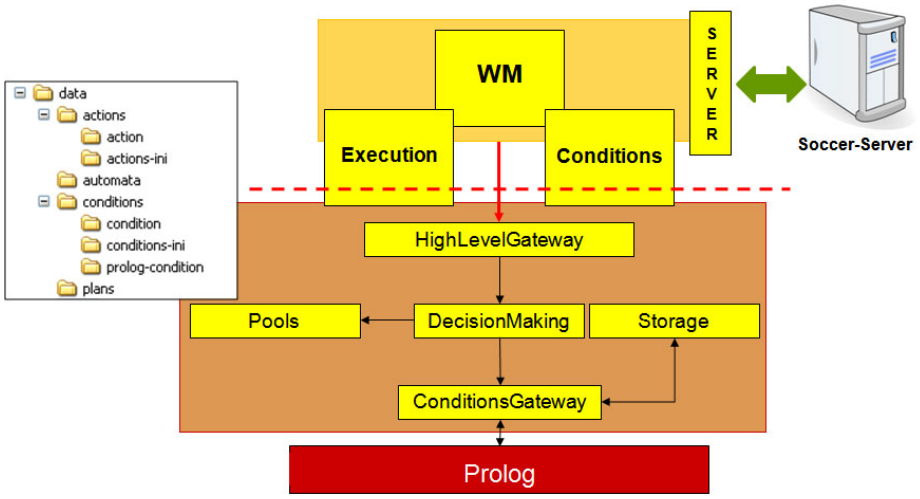

Fig. 1. System overview illustrating the separation in execution-level (top) and decision-making-level (bottom)

for server connection, algebraic calculations and some tactics modules to name a few. The tactics modules create commands like 'pointto' or 'turnneck' which are auxiliary and may be sent additionally to the main commands retrieved by the decision-making level.
The central structure in the EL is the world model which is responsible for providing accurate information about field objects (including self). In order to get the best possible quality of our sensed data we developed several filtering techniques like Kalman filters for object tracking (as proposed by [3] and [4]) and particle filters for self-localization (see [5]).
The quality of the results in the conditions and execution packages depends on this accuracy. Whereas various conditions are used to evaluate the actual state given by the world model, the execution breaks high-order actions, like dribbling, intercepting and scoring goals, down to server-protocol commands and ensures their dispatching via world model and server modules.
Most of the concepts applied in the EL are an extension based on the UvA-source [6] regarding the recently released source base of DAI [7].

3.3 Decision-Making Level

The proposed Decision-Making Level (DML) is a general control framework which allows for the implementation of arbitrary, versatile applicable decision-making algorithms as well as the monitoring and analysis of the execution. Furthermore it defines general decision-making entities and their syntax and semantics with aid of uniform XML-languages.
Thus, the DML uses plans to define the agents' behaviour and extract a decision

what action to perform next. Each plan is represented by a hierarchy of actions and each action consists in turn of a set of preconditions (to check feasibility), a set of postconditions (to check success), the EL-action to be executed and a set of parameters to this action. General attributes for actions and conditions are assigned in associated ini-files.
Pre- and postcondition checks are performed by the Prolog backend operating on basic clauses which are evaluated in the EL.

Features The DML described has following fundamental features:

Generality The DML framework is independent from any RoboCup issues and can be employed in different even non-robotic systems. It is able to take full responsibility for decision making and execution, planning, learning and/or logic reasoning by providing simple, adaptive interfaces making low demands on whatever execution-level it should operate on. Some example uses could be its application on Middle Size League robots, service robots or also economic process-oriented systems.

Decision Making Any possible decision making algorithm (operating on either static or dynamically created plans) can be implemented fast according to the simple interface and event-handling scheme provided by the DML. Whenever the EL requires the DML to derive a decision, all registered decision making implementations are consulted.

Monitoring and Analysis of Execution During execution the complete decision process can be stored and analysed to have the decisions made in the dynamic and uncertain environment under best possible control.
Moreover concepts of automaton theory are applied to verify the execution (i.e. implemented actions) by checking executed decisions with the aid of logged world information against automaton-encoded meta-descriptions of each action implementation.

Uniform Language All items on which decision making is based are represented by XML-files according to several XML schemes. When defining syntax and semantics for plans, actions and conditions the most important properties were generality and convenience. Thus, our languages enable the user to encode many different formalisms, like for example STRIPS instances [2], TR-programs [1] or a version of TDL [8] providing a basis for the according decision-making, learning and planning instances.

Design The combination of several elements is necessary to implement the decision-making level's desired behaviour (cp. Figure 1):

- *Decision-Making Interfaces and Algorithms*
  to traverse the plans and retrieve best-possible actions.

– *Pools*
  containing all plans, actions and conditions, mapping from the string representation in XML-files to concrete objects.
– *Storage*
  keeping the decision process, world states, formalizations and arbitrary debug info used for evaluation of the system's performance.
– *Prolog implementation*
  used for evaluating pre- and postconditions of all actions (our team is using the fine Prolog-implementation tuProlog [9]).
– *Gateway*
  as a general interface for the EL over which all requests operate.

All DML-functions can be accessed via a single gateway. This includes storage and decision interfaces. Whenever a decision query event in the EL is raised to indicate that a decision is necessary it is forwarded to each registered decision-making algorithm which will execute an appropriate action.
Concerning decision-making all XML-encoded elements are built at startup: Conditions are translated to Prolog code, action code is created and compiled dynamically and plans are constructed using the action pool.

User Interfaces Two graphical user interfaces provide easy access to behaviour modeling and analysis making it easy to teach or test various aspects of intelligent systems on different levels (from non-professional up to professional applications).

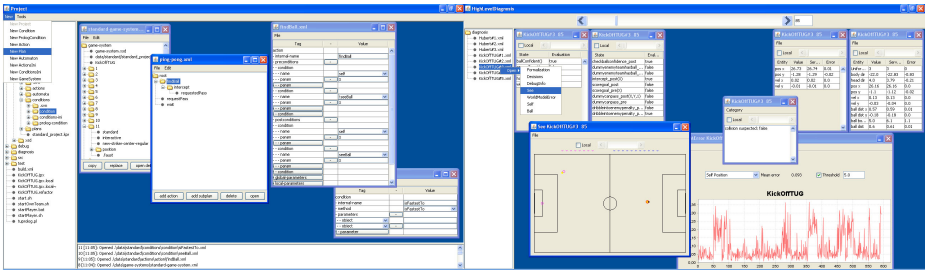

Fig. 2. Project and Diagnosis GUI

Project GUI The Project GUI aids the user in defining each agent's behaviour by means of generating basic decision-making-objects (conditions, actions and plans). EL-specific extensions can be included easily as demonstrated by game-systems which link each agent to its behaviour, set-up position and nickname. This way many alternative models can be created and tested against each other.

Diagnosis GUI The agents are capable of recording diagnosis information during the game - all decisions made, each time step's world state, the value of all conditions applied on the world model and user-specific debug info (organized by categories). This information can be displayed and evaluated using the Diagnosis GUI. Just like the Project GUI this user interface can be extended EL specifically, too. In addition to the capability of displaying the mentioned diagnosis information, it is hence possible for the RSL to compare the agent's world model against the exact server logs. This can be done via lists, graphical representation of statistics or even graphical reproduction of the game marking exact and logged positions.

3.4 Interaction

EL and DML interact over the DML's gateway. The EL is constantly collecting the server messages and incorporating them into the world model and requires a decision each cycle. It therefore raises a decision query event in respect to the present play mode.
In the next step the decision making algorithm (or several as more than one may be registered) is informed and retrieves an action considering the given situation by evaluating conditions. This evaluation is achieved via Prolog which further links to the EL's conditions-interface operating on the world model. By means of such out-sourced inference general, highly abstract conditions can be implemented and easily enhanced and optimized in Prolog source. The action is then executed in the EL, again consulting the world model to collect all information necessary to generate an appropriate command.
The use of Prolog to evaluate pre- and postconditions proves to be beneficial concerning the principles of flexibility, extensibility, functionality and convenience without having to reinvent the wheel. It serves as a robust logic backend for various applications, and thus suits the claim for generality as well. As our condition-implementations in the EL do not unify variables for reasons of simplicity and straightforwardness but instead accept constants as parameters, the DML has two options on how decisions are derived. Currently each condition clause (i.e., an individual pre- or postcondition of an action) is evaluated independently in the EL by grounding symbols over facts and calling the condition afterwards. This works perfectly well if the condition space should be reduced, e.g. for performance reasons. Full inference may be achieved by formalizing and consulting the complete condition space before retrieving a decision.
Figure 3 illustrates the decision-making cycle. We explain the decision-making process in more details using the following example. Assume that a new cycle has just begun and the EL requested the DML to derive a decision. The decision-making algorithm of player 1 is currently inspecting an action 'pass' and needs to check its preconditions, for example if the agent is in ball possession and whether there is a team-mate to whom he can pass.
As mentioned above those preconditions have been translated to Prolog syntax at startup:

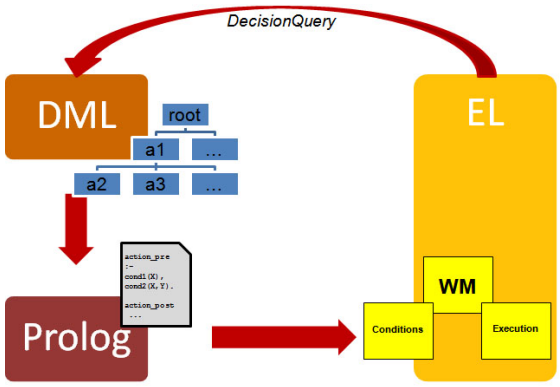

Fig. 3. The interaction between DML and EL over Prolog describing the decision-making cycle.

```
pass_pre :- self(X), hasBall(X), ownPlayer(Y), canPassTo(X,Y).
(This line has been simplified for reasons of convenience.)
```

Thus, Prolog is queried with *pass_pre* and unifies X with 1 (the agent's uniform number) using the fact *self(1)* and evaluates *hasBall(1)* next by calling the EL. If the implementation of *hasBall* returns true Prolog unifies Y with all possible uniform numbers (2-11) over predefined predicates until either the clause *canPassTo* returns true or no further unification is possible. A concrete own_player Y can be passed to the implementation of 'pass' to communicate the pass-partner.

4 Decision Making

Our 'Decision-Making Algorithm for Dynamic Environments' is based on the teleo-reactive program formalism as proposed by [1]. As suggested in this paper an agent's behaviour (in case of the RSL depending on its position and role) is defined by plans with each plan represented by a combination of actions, each action having conditions associated to them.
We adapted this notion (in regard to the STRIPS-notation [2]) describing an action to consist of

1. a set of preconditions C_{pre},
2. a set of postconditions C_{post},
3. the name of a low level action A (with A' as corresponding implementation) to be executed and
4. a set of parameters P_A which are passed when calling A'.

In contrast to TR-programs (which are basically encoding linear action sequences from goal to basic action) all action-sequences are encoded in a single tree, where the root depicts the start, siblings alternative actions and each leaf a possible

goal. This means, each path in a tree corresponds to a TR-program. Figure 4 illustrates how several TR-programs are transformed to a tree. The order in which the programs are added as paths is essential as it defines the order in which the actions are selected.

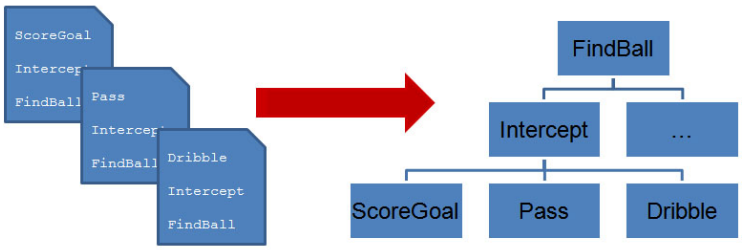

Fig. 4. TR-programs and their transformation to a simple exemplary tree.

The decision making algorithm starts at the root and traverses the tree in depth-first manner (for each state/time step).
Following pseudo-code describes the algorithm in detail:

```
traverse(current_root):

FOR allChildrenOfCurrentRoot
  retrieve action from actual_child;
  IF checkPostconditions(action)
    IF isLeaf(current_root)
      do nothing and return; // a goal has been reached
    ELSE
      traverse(actual_child); //continue on next level
  ELSE
    IF checkPreconditions(action)
      run action; //execute the action
```

Thus, if C_{pre} of A is satisfied, A' can be executed, whereas if C_{post} is satisfied the state which the execution of A' is to accomplish has already been reached. Applied to the very simple exemplary tree in Figure 4 this means that if

- 'FindBall' has C_{pre}={!seeBall(X)} and C_{post}={seeBall(X)}
- 'Intercept' has C_{pre}={seeBall(X),!hasBall(X)} and C_{post}={hasBall(X)}
- the world state denotes that an agent sees the ball but is not in ball possession

the decision making algorithm would thus execute 'Intercept' after

1. checking C_{post} of 'FindBall' (true)
2. checking C_{post} of 'Intercept' (false)

3. checking C_{pre} of 'Intercept' (true)

ensuring that the ball is already in sight, the agent is not in ball possession, but is able to intercept it. If the interception was successful, the next level's actions are examined. 'Pass' would be performed next according to the same principle provided that 'ScoreGoal' was not feasible because the goal was well covered for example.
Whenever a goal is reached or no sibling is left to be inspected no decision is executed. The first case means that no better possibility could be executed and a (for the moment) sufficient state has been reached. As one important property for our trees is completeness (in terms of [1]), the second case is not to happen because there is always an executable action.
The benefits of this approach are that while the advantages of TR-programs are preserved, the algorithm is able to produce high quality decisions faster and at the same time easier formalization is possible due to a more natural and bundled way of plan description.

5 Automatic Action Analysis

One of the most challenging tasks for an autonomous agent is the execution of actions in dynamic and uncertain environments. In RSL we have been facing the problem of verifying whether an action performs its desired behaviour in order to maintain robust decision execution. Therefore, we introduced a formalism of automaton-encoded meta-descriptions of actions (compare [10] and [11]). Such automata define a sequence of valid states along with transitions to error states to derive what went wrong. By checking associated conditions and invariants the processing of each automaton ensures the correct execution on the one hand as well as facilitates the detection of faults. In this context faults are either implementation bugs or caused by exogenous unforeseen events.
In the first place action analysis is applied offline on logged data to verify and optimize the execution and statistically monitor and illustrate the performance of individual actions but will be further extended operating online during plan-traversal, -execution, planning and plan adaption.

6 RSL as Education and Research Platform

The design described in this paper proposes an easy and powerful architecture and extends RoboCup Simulation League's fundamental idea to a new level of research and education by providing multiple points of entry for a number of educational and research activities.
According to the following classification the proposed system can be used to introduce the RoboCup Simulation League to a general non-professional public, teach artificial intelligence and/or logic programming at various levels in schools and at universities, use arbitrary RSL aspects in tutorials or courses and a number of other fields of application.

From the big picture towards detail (with increasing degree of difficulty) it is thus possible to deal with

- high-level behaviour modeling
  - by combining existing plans and defining new game-systems.
  - by creating new plans with existing conditions and actions.
  - by coding directly in/adapting generated Prolog clauses and relations.
- multi-level behaviour modeling (i.e. defining new EL actions and conditions along with high-level modeling).
- different decision making, planning and learning algorithms.
- mere low-level issues (which of course can be subdivided further).

7 Future Work and Research

As our DML easily enables the implementation of other decision making, and also planning and learning techniques, some future work will be the application of machine learning algorithms for decision-making to compare different methods, as well as teleo-reactive planning and learning (as suggested in [12]) adapted to the needs of our version as a useful alternative to solely static plans.
Furthermore we will continue developing our automatic action analysis based on automata which we believe will be useful for plan adaption, plan repair and dynamic plans in general.

References

1. Nilsson, N.J.: Teleo-reactive programs for agent control. Journal of Artificial Intelligence Research **1** (1994) 139–158
2. Nilsson, N.J., Fikes, R.E.: Strips: A new approach to the application of theorem proving to problem solving. Artifical Intelligence **2**(3/4) (1971) 189–208
3. Kálmán, R.E.: A new approach to linear filtering and prediction problems. ASME-Journal of Basic Engineering **82D** (1960) 35–45
4. Dietl, M., Gutmann, J.S., Nebel, B.: Cooperative sensing in dynamic environments. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'01). (2001)
5. Fox, D., Burgard, W., Dellaert, F., Thrun, S.: Monte carlo localization: Efficient position estimation for mobile robots. In: Proc. of the National Conference on Artificial Intelligence. (1999)
6. de Boer, R., Kok, J.: The incremental development of a synthetic multi-agent system: The uva trilearn 2001 robotic soccer simulation team. Master's thesis, University of Amsterdam, The Netherlands (February 2002)
7. Endert, H., et al.: The Dainamite Agent Framework. DAI Labor, TU Berlin. (November 2006)
8. Simmons, R., Apfelbaum, D.: A task description language for robot control. In: Proceedings Conference on Intelligent Robotics and Systems. (October 1998)
9. Denti, E., Omicini, A., Ricci, A.: tuprolog: A light-weight prolog for internet applications and infrastructures. In: Practical Aspects of Declarative Languages: Third International Symposium, PADL 2001, Berlin, Springer (March 2001) 184–199

10. Beetz, M.: Plan-Based Control of Robotic Agents: Improving the Capabilities of Autonomous Robots. Springer (2002)
11. Ferrein, A., Fritz, C., Lakemeyer, G.: Online decision-theoretic golog for unpredictable domains. In: Proceedings of 4th International Cognitive Robotics Workshop. (2004)
12. Benson, S., Nilsson, N.J.: Reacting, planning, and learning in an autonomous agent. In Furukawa, K., Michie, D., Muggleton, S., eds.: Machine Intelligence 14 - Applied Machine Intelligence, Oxford University Press (1995) 29–64