Team Description RoboLog Koblenz 3D [⋆]

Heni Ben Amor, Joschka Boedecker, Anita Maas, Jan Murray,
Oliver Obst, Achim Rettinger, and Christoph Ringelstein

University Koblenz-Landau, AI Research Group, D-56070 Koblenz
{amor,jboedeck,amaas,murray,fruit,achim,cringel}@uni-koblenz.de

Abstract. For this years RoboCup Simulation League 3D competition we plan to significantly enhance our team. Building up on the low level skills and world-model as well as the *Zeitgeist* class object framework of our last years team we will add a graphical user interface based on UML-State-Charts. Besides manual specification of agents a planing component will be designed to improve agent behavior. Finally we propose a module that can predict the feasibility and utility of actions. Those criteria will be learned incrementally, both offline and online.

1 Introduction

This Team Description Paper outlines our ongoing development of RoboLog, a RoboCup 3D-Simulation-League team from the University of Koblenz. After successfully participating with a completely new 3D-team at the RoboCup 2004, this years enhancements are logical steps accompanying our research agenda. We are aiming at creating methods to implement physical multiagent systems that on the one hand can carry out mid and long term goals while staying reactive on the other hand.

For this purpose we plan to use our recently developed UML-State-Chart Editor for intuitive manual specification of agent behavior. This will possibly be complemented by a planning component that can contribute automatically generated strategies (see Section 3).

Another core component that we plan to integrate into the RoboLog framework is intended to improve decision making capabilities by assigning feasibility and utility to available actions. Using Machine Learning to come up with reasonable feasibility and utility criteria seems most promising to us. A big challenge in this context will be to make those criteria adaptable to specific opponent teams during a game (see Section 4).

2 Reusing the Simulator Architecture for the Team

Zeitgeist is the class object framework that was developed by us for the 3D simulator [1]. Here, class objects are basically class factories that allow to create instances of classes during run time based on a string interface. Additionally, with *zeitgeist* we can request type and interface information from object instances, so that scripting languages can be attached easily.

[⋆] This research has been partially supported the special research program DFG-SPP 1125 funded by the German Research Council (DFG).

Another part of *zeitgeist* is a kind of registry for class objects. *Zeitgeist* provides a tree-like structure to manage class objects; these can be accessed via the path name through the tree. This kind of design can also be found in the resource manager architecture of the QNX operating system [2].

By implementing all classes of the RoboLog3D kernel in the zeitgeist framework, it is possible to define interfaces for basic player modules in a generic way. Later on, it is possible to exchange single modules without breaking any existing code, as long as the defined entry points and interfaces are maintained. In the context of a soccer agent, modules are for example the world model holding the data the agents perceive or single behaviors.

3 Specifying Agent Behaviors

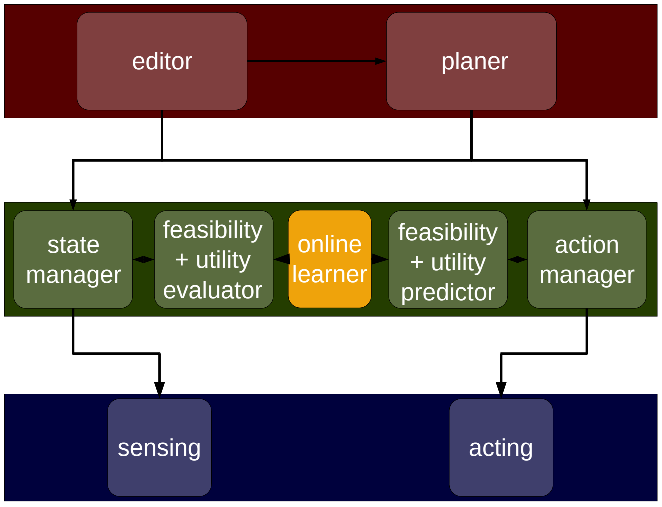

Fig. 1. RoboLog Framework

All components of an agent are controlled by an editor based on UML-State-Charts. In addition to that the planer can contribute automatically generated strategies. Those two high level components (see Figure 1; red area) have access to the state- and action manager (green area).

The state manager is some kind of high level worldmodel that observes the current state of the simulator and the agents. It also has a history of actions previously performed. It can evaluate the success (and utility) of those actions with the help of the feasibility + utility evaluator. Accordingly, the action manager hosts a list of all actions available to an agent. It can also predict the feasibility and utility of every action with the help of the feasibility + utility predictor.

These mid level components can access (sensing) and controll (acting) all low level skills (see Figure 1; blue area).

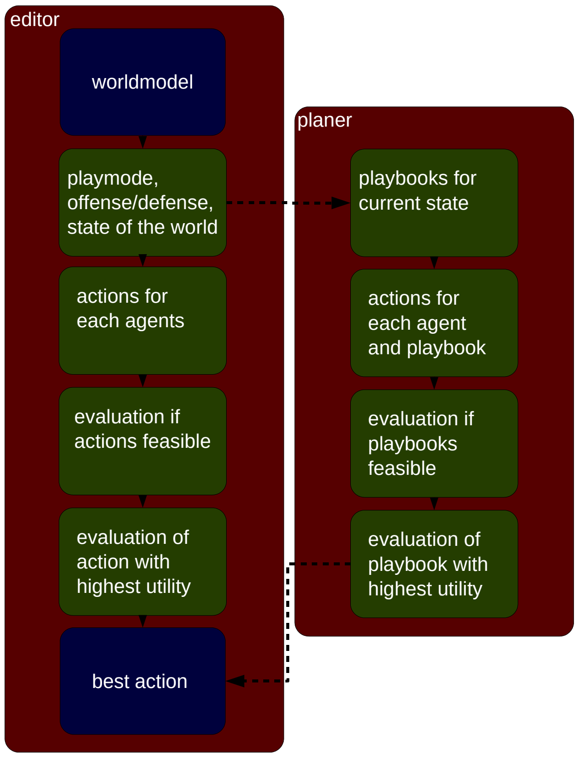
editor
worldmodel
playmode, offense/defense, state of the world
actions for each agents
evaluation if actions feasible
evaluation of action with highest utility
best action
planer
playbooks for current state
actions for each agent and playbook
evaluation if playbooks feasible
evaluation of playbook with highest utility

Fig. 2. Steps of Decisions-Making Process

The sequence of steps taken to make a decision is exemplary presented in Figure 2. According to the current state of the world (obtained from the worldmodel or state manager, respectively) the editor considers a set of actions specified by the user and hosted by the action manager. The feasibility and utility of each action is predicted with the help of the feasibility + utility predictor and finally the best action is executed. Under certain conditions the planer can also make contributions to this process (see right column in Figure 2).

4 Evaluating Feasibility and Utility of Actions

To come up with reasonable criteria for the feasibility + utility predictor an online machine learning technique will be used (see Figure 1; yellow rectangle) . The training data is obtained during a game by the feasibility + utility evaluator. Then a hypothesis is generated by the online learner and provided to the feasibility + utility predictor. Thus, the team can adapt to actions of the opponent team online. An a priori hypothesis can be obtained offline before the game by playing training games and storing the resulting hypothesis. In this manner the prediction accuracy of the feasibility + utility predictor can incrementally be improved.

References

1. Marco Kögler and Oliver Obst. Simulation league: The next generation. In Daniel Polani, Andrea Bonarini, Brett Browning, and Kazuo Yoshida, editors, *Proceedings of the RoboCup Symposium 2003*, July 2003. An LNAI version of the proceedings will appear later.
2. QNX Software Systems Ltd. QNX Neutrino Realtime Operating System: System architecture. http://www.qnx.org/developer/docs/momentics_nc_docs/neutrino/sys_arch/about.html, 2003.