# The magmaOffenburg 2015 RoboCup 3D Simulation Team

Klaus Dorer, Jens Fischer, Stefan Glaser, David Weiler[1]

Hochschule Offenburg, Elektrotechnik-Informationstechnik, Germany

**Abstract.** This paper describes the magmaOffenburg 3D simulation team qualified as 3rd in Brazil for RoboCup 2015. While last year's TDP focused on how we statistically evaluate new features, this year we describe our team software released after Brazil 2014.

## 1 Introduction

One important aspect of RoboCup is its strong community. This community is intended to be an open community. In order to simplify the entry of new teams into 3D soccer simulation, we believe it is necessary to provide high quality base code with which teams can enter the league more easily. This and other releases of agent code or libraries can be found at [1]. We believe that a release is also beneficial to us, because it means we have to keep the software in a clean state and are forced to document it. Part of this documentation is this TDP.

Section 2 explains the main architecture of the runtime of our software. In section 3 we present the components in more detail.

## 2 Architecture

Our runtime code is organized as a three layer architecture (see Fig. 1). Each layer may only depend on the next lower layer, i.e. the decision layer, for example, only depends on the model layer and not on the channel layer, while the channel layer does not depend on any other layers. It is therefore possible to only use the channel layer of our software, or just use the channel and model layer without the decision layer.

Inside each layer, our code uses a component based architecture. Each component consist of two parts: the interface part and one or possibly more implementation parts. One component may only depend on the interface part of another component, but not on the implementation part. In Fig. 1 the selected component Decision, for example, has allowed dependencies to Belief and Behavior (green arrows and components shown in green), but not to their implementations.

Folder srcAgent contains the components for the runtime in package magma.agent and magma.robots. Both contain the packages shown in Fig. 1 with the component interfaces in the package itself and the implementation(s) in subpackag(es).

magma.agent contains all robot model independent code. magma.robots has a subfolder for each supported robot type. Each type follows again the same folder structure as magma.agent. It is worth to note that also our real robot sweaty is run by this software inside this architecture by providing sweaty specific parts in another (not released) subfolder agent.robots.sweaty.

The three components flags, general and meta that are not part of Fig. 1 will be explained in the next section.

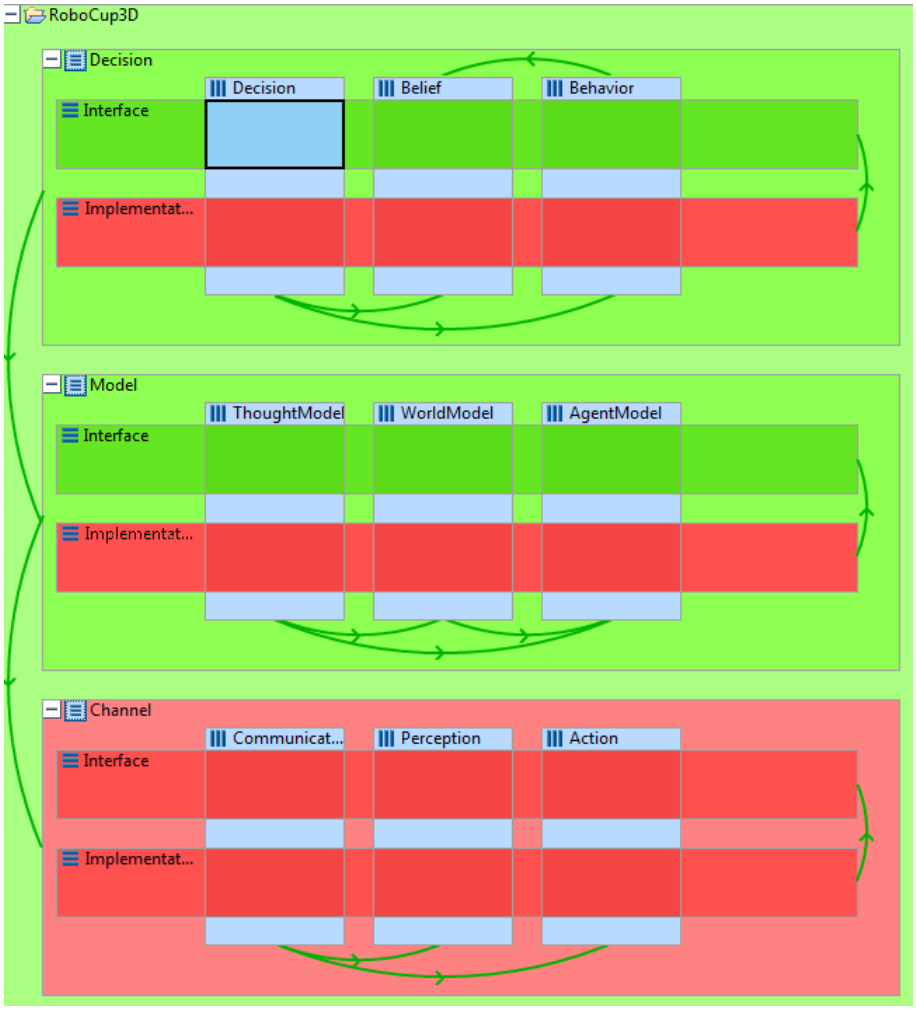


**Fig. 1.** Architecture of the magmaOffenburg Runtime.

## 3 Components

In this section we describe each of the components in more detail.

### 3.1 Core

The following components are outside the general runtime architecture. They contain, for example, the component factory and will therefore possibly depend on all other components.

**General** The core component, we call general, mainly contains the component factory and the agent runtime. The first follows the factory pattern and creates the desired component implementations. It is subclassed, for example, by different robot types or to provide special learning setups (not released). We decided not to use an off the shelf inversion of controll container like, for example, guice, but use our own explicit factories. All components use constructor injection to inject the dependencies. The agent runtime implements the main update loop for the components.

**Meta** The meta component contains two parts: server meta modeling and robot meta modeling. The server meta models contain all necessary information of field sizes and flag positions for all versions of the competition since 2009. The robot meta models contain all information of the robots body parts, sizes, joint positions and rotation axes and sensor positions. Each robot model that varies from the standard nao has to overwrite the meta model and provide the corresponding information.

### 3.2 Channel

The next components are part of the channel layer and ensure communication with the server.

**Perception** Perception contains interfaces and implementations for all visible and sensible information in the league. These perceptor classes are used to transport the information into the model layer. The Perception class itself is only a container for the perceptors. Main part of this component is the ServerMessageParser which parses the strings received from the server into perceptor objects.

**Action** Action contains interfaces and implementations for all actions that can be triggered. More specifically it knows how to encode those actions, represented as effectors, into a string representation for the server.

**Communication** This package implements the so called channels and the channel manager. Channels are input or output connections that can operate asynchronously to receive or send information. The channel manager manages the channels and triggers model updates. For the application on real robots this is quite handy. In 3D simulation all information is sent and received through one channel (SimsparkChannel) and therefore does not make use of this feature.

### 3.3 Model

The following components build the model layer and store all information the agent has from itself and the world around it. Like the channel layer also the model layer has been completely released.

**Agentmodel** The agent model contains all information the agent knows about its own state. It has three object trees: the sensed body model contains information we have received as perceptions, the expected body model, that is based on the sensed model but updated with the information we have sent to the server and expect to be true next sensing cycle and the future body model that is writable and contains the state we want to be in. The latter is used to create the commands to the server.

The agent model also contains a complete and efficient implementation of a reverse kinematics calculation for the Nao types.

**Worldmodel** The world model stores all information we have about the soccer field and game (GlobalMap) as well as our own position inside the game (ThisPlayer). It contains the localizer sub-component that has various implementations of 2D and 3D localization algorithms.

**Thoughtmodel** In the thought model we store information that is not directly perceived, but concluded from perceived data. The strategy sub-package contains the complete role assignment and team strategy classes. The IFOCalculator determines the indexical functional objects like who is the agent closest to the ball or the opponent closest to me. The KickDirectionProfiler determines which direction we should pass the ball to.

### 3.4 Decision

The decision layer contains the high level decisions the agent performs as well as the beliefs, theses decisions are based on and the behaviors that execute the decisions possibly by taking further mid or low level decisions.

We have decided not to release the complete decision layer in respect of other existing teams. It would not be fair, we believe, if new teams play on a top level by just using our release. But we added all beliefs and many behaviors and decision makers as examples of how to use the model layer.

**Belief** A belief encapsulates a high level (fuzzy) belief about the world. It was introduced to decouple decision making from the model layer in order to be flexible as to which decision architecture to use. We have an implemenation of extended behavior networks [9] (not released), and planned to have a BDI architecture, but ended up to use simple decision makers so far. That is why we now have decision, belief and behavior in one layer and allow decision to have access to the model layer. This means that beliefs are technically no longer necessary, but still help to encapsulate common beliefs into separate classes.

**Behavior** Behaviors are either simple behaviors performing exactly one action or they are complex behaviors that first decide which more basic behavior to perform. We support several possibilities how to specify movement behaviors:

**Movement Behaviors**
Each movement is subdivided into movement phases which are composed of single joint movements (package movement).

**Function Behaviors**
Static behaviors can be defined using functions that model the joint angle over time (package supportPoint). Examples of function behaviors can be found in folder config (behaviors.nao.FunctionBehaviors.UsedBehaviors)

**Motorfile Behaviors**
Motorfiles specify motor angles for each joint in a column of a csv file. Each row corresponds to a point in time. (package motor). Motorfiles could be seen as special case of function behaviors representing piecewise linear functions. Examples of motorfile behaviors can be found in folder config (behaviors.nao.BehaviorValues.UsedBehaviors)

A more detailed description can be found in our team description paper 2012 [2].

Complex behaviors (package complex) are composites of more basic behaviors and contain further decisions as to which more basic behavior to perform. A complex kick behavior, for example, may contain the decision to kick with the right or left leg and then will perform the corresponding basic behavior. The behaviors also encapsulate the decision, at which time a behavior switch may occur as well as morphing of behaviors in case of compatible behaviors.

The behaviors are not the latest behaviors available in our team as described above. The Getup used is an example of a motor behavior used in 2009. The walk is our walk from 2010 experiencing some timing problems with respect to the sensed and expected body model [5]. Also the kicks are not the newest ones. This is were your work starts really using our framework.

**Decision** Decision contains the high level decision making of the agent. There are different decision makers for the goalie and field players as well as for training situations in our release. We currently use a sequential architecture for our agents, so all processing steps have to be performed within the 20 ms cycle time.

## References


1. http://simspark.sourceforge.net/wiki/index.php/Agents
2. http://robocup.hs-offenburg.de/nc/downloads/
3. Glaser S and Dorer K (2013) Trunk Controlled Motion Framework. In Proceedings of the 8th Workshop on Humanoid Soccer Robots, IEEE-RAS International Conference on Humanoid Robots, Atlanta, 2013.
4. Hochberg U, Dietsche A and Dorer K (2013) Evaporative Cooling of Actuators for Humanoid Robots. In Proceedings of the 8th Workshop on Humanoid Soccer Robots, IEEE-RAS International Conference on Humanoid Robots, Atlanta, 2013
5. Schindler, I.: Laufen auf zwei Beinen in der simulierten RoboCup 3D-Umgebung. Bachelor thesis, Hochschule Offenburg, Germany (2009)
6. Dorer, K.: Modeling Human Decision Making using Extended Behavior Networks. J Baltes et al. (Eds.): RoboCup 2009, LNAI 5949, pp. 81–91. Springer (2010)
7. O. Obst and M. Rollmann, *SPARK A Generic Simulator for Physical Multiagent Simulations* Computer Systems Science and Engineering, 20(5), September 2005
8. Dorer, K.: Extended Behavior Networks for Behavior Selection in Dynamic and Continuous Domains. In: U. Visser, et al. (Eds.) Proceedings of the ECAI workshop Agents in dynamic domains, Valencia, Spain (2004)
9. Dorer, K.: Behavior Networks for Continuous Domains using Situation–Dependent Motivations. Proceedings of the Sixteenth International Conference of Artificial Intelligence (1999) 1233–1238