The magmaOffenburg 2026 RoboCup 3D Simulation Team

Mike Benz, Hannes Braun, Klaus Dorer, Stefan Glaser, Madhurshalini Mahalingam, Sven Oehler, Simon Palewicz, Simon Rösch, Adrian Schade, Sascha Schrempp, Artem Vlasiuk[1]

Hochschule Offenburg, Institute for Machine Learning and Analytics IMLA, Germany

Abstract. In this year's team description paper we describe the design changes that have been necessary to switch from the NAO robot simulated in simspark to the T1 robot simulated in the new Mujoco simulator. It is an example on how good design pays off for adding new requirements. Our team can play both simulators with the same binary just by specifying the robot type as program argument.

1 Communication Layer

Sending sensor information from the server to the robots in rcssservermj has mainly been implemented to be compatible with the simspark protocol. There are two main differences dealt with in the MujocoMessageParser class (see Figure 1): the game state perceptor sends the name of both teams and there are global position and orientation perceptors.

```
(GS (t 231.52) (pm PlayOn) (tl magma) (tr other) (sl 2) (sr 1))
(pos (n <name>) (pos <x> <y> <z>))
(quat (n <name>) (q <qw> <qx> <qy> <qz>))
```

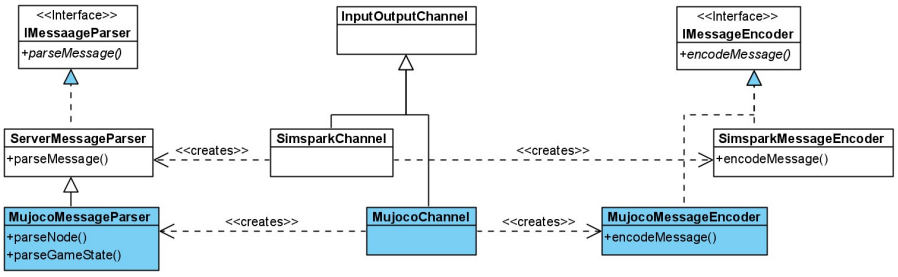

Fig. 1. Design changes to the communication layer.

Sending motor commands from the robots to the server has changed to include the desired angular position of the joint, the desired speed of the motor, the gain for the position difference, the gain for the speed difference and an absolute torque tau. This is handled by the MujocoMessageEncoder class.

```
(<name> <q> <dq> <kp> <kd> <tau>)
```

The creation of the proper parser and encoder objects is done in channel classes. This is handled by the MujocoChannel in case of a Mujoco server, which is created by an abstract factory described later.

2 Model Layer

The model layer is concerned with storing all information the robot has for the world around it and information it has about its own state. The former is stored in the WorldModel classes the later in the AgentModel classes. Class RoboCupWorldMetaModel (see Figure 2) stores prior information of the world around the robot like the definition of different field sizes and the geometry of fixed visible objects that does not change over time. The specifics of the humanoid field used in rcssservermj are modeled in class MujocoAdultSizeField.

Information about the robot itself that does not change over time is modeled in the RoboCupAgentMetaModel class and subclasses. Class T1AgentMetaModel stores the robot model of the T1 robot like the body parts sizes and positions and the motors connecting them. It also stores the position and orientation of the available sensors.

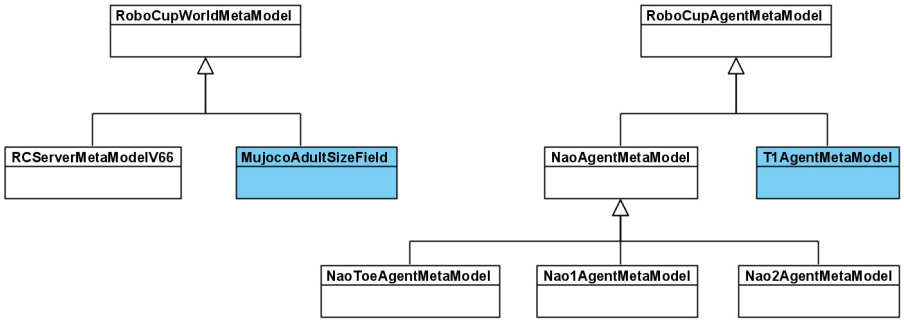

Fig. 2. Design changes to the model layer.

3 Decision Layer

The decision layer of magma contains the high level decision making classes as well as all complex and simple behaviors. The description of the changes are

done with respect to the state of the team for the Brazil Open in October 2025. At that competition, we played 3 vs 3 T1 robots.

3.1 Decision

All major decisions are done in class SoccerDecisionMaker (see Figure 3). It uses a template method pattern so any single decision can simply be overridden in its subclasses. This is, for example, the case for the getup decision, that is handled slightly different to what the NAO robot did. Class MujocoThreeVSThreeDecisionMaker therefore overrides the getup() method to change the getup decision making. Another example is the decision to move to a position, that at the Brazil Open had not been done using our roles framework, but with a simple role logic directly implemented in the decision maker class.

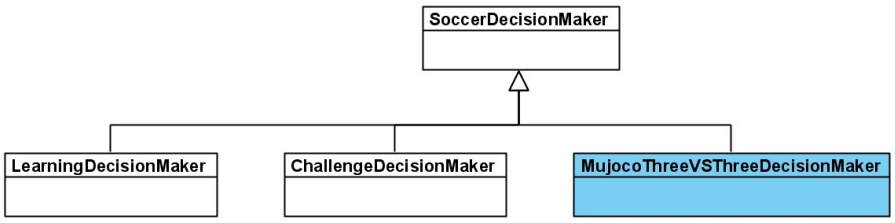

Fig. 3. Example of a design change to the decision layer.

3.2 Behaviors

All behaviors for the T1 robot have basically to be created from scratch. However, the integration into our existing design took advantage from various existing abstractions as exemplified with the walk behavior (see Figure 4).

The walk behavior used at the Brazil Open has been a behavior trained with Deep Reinforcement Learning for the real T1 robot used by our humanoid adult size league team Sweaty. Our framework already supported DeepBehaviors, but it required some adaptations especially to fit to the IBaseWalk interface done in class T1DeepWalk. This made sure, that we could use the same walk and obstacle avoidance done in classes Walk and WalkToPosition as we use for the NAO robots. Each DeepBehavior requires an IObservationSpace that knows how to transform the current state and sensor values into the input signal for the DRL policy network. This is done by class T1WalkObservation, which had to exactly match its Python version used for the real robot. Also, each DeepBehavior requires an IActionExecutor which encapsulates the knowledge of how to translate the output of the policy network into motor action. Due to the torque

based control of robots in rcssservermj, a new class TorqueActionExecutor had to be introduced that can be used for any DeepBehavior targeting rcssservermj.

Other DeepBehaviors used at the Brazil Open were the GetupFromFront and GetupFromBack behaviors that also required a new IObservationSpace class and a getup complex behavior controlling the phases of lying flat, deep getup and deep walk to get stable. Both getup behaviors have been trained using the rcssservermj simulation, while the walk has been trained in Isaac Sim.

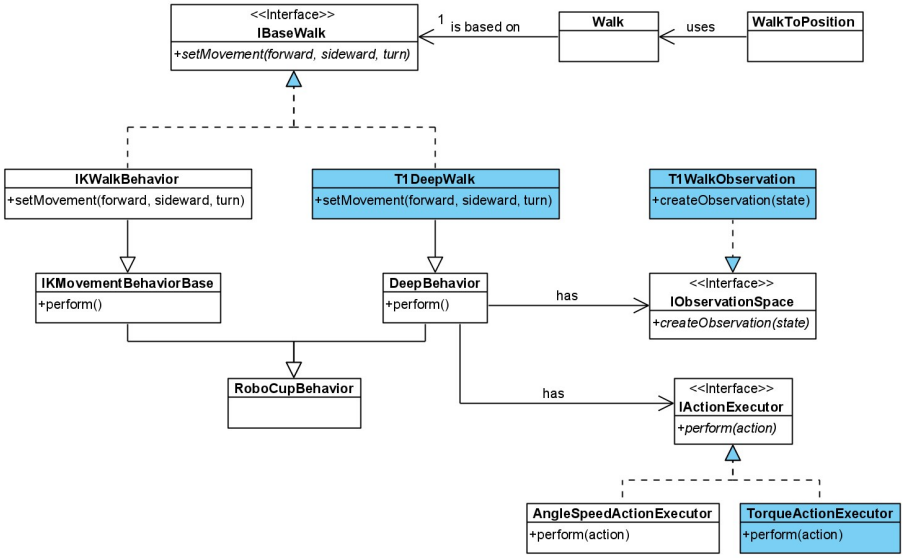

Fig. 4. Integration of a deep walk behavior from Sweaty.

4 Runtime Layer

The RoboCupAgentRuntime contains the logic for initializing all components at startup and also contains the main update loop. This main class did not require any change for the integration into rcssservermj.

This worked, since the abstract fabric ComponentFactory used to create all components is injected into the runtime. It contains factory methods for all robot specific components to be created like the decision maker, the meta model and model classes, and the channel classes for communication to only mention some. It is also responsible to create instances of all behaviors required for the robot. The class T1ComponentFactory does this for the specific components required for T1 (see Figure 5).

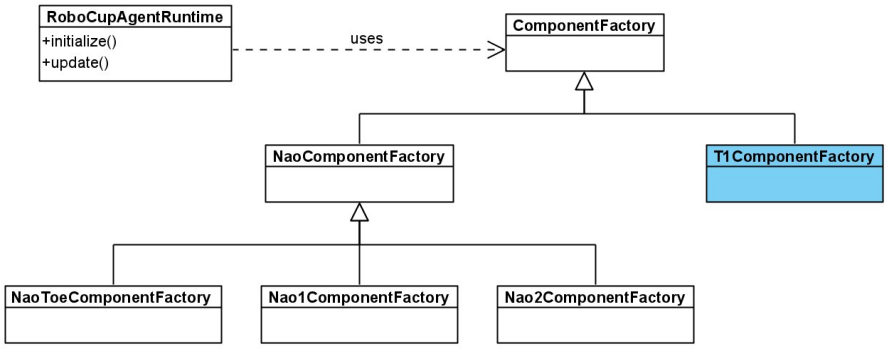

Fig. 5. Component creation layer extension.

5 Tools

A couple of tools have been developed over the years. Two of them are mentioned here, the MagmaDeveloper and the MagmaDeepLearning framework.

5.1 MagmaDeveloper

The MagmaDeveloper is the main tool for debugging magma robots. It provides a couple of views and dialogs for this.

The run dialog (see Figure 6) allows to start any specific number of robots with selectable robot factory and decision maker. It is up to the user to specify the proper ports of either simspark or Mujoco and to make sure to specify robot types that are supported be the specified server. No changes were necessary here to work with rcssservermj. The drop down boxes are filled with the available robot types and decision makers from the factory classes.

The 3D GL Self View shows information about the robot itself, like the current angle of joints and the position of the body parts as box model. Overlays are available for the center of mass, the current stability hull or the past foot trajectory. No changes were necessary here to work with rcssservermj. The tool is using the AgentMetaModel definitions.

The 2D and 3D GL Field views provide information about the world model beliefs the robot has. This includes information about the ball, other players, or the visible lines and flags. Overlays are available for showing known obstacles, player numbers, future ball positions, desired walk position of all own players, pass position evaluations and some more. No changes were necessary here to work with rcssservermj. The tool is using the WorldMetaModel definitions.

The MagmaSimulator, however, is not yet available to debug the decision logic of robots in rcssservermj. It is currently hard coded to the simspark field sizes and the soccer decision maker used by NAO robots. This is still future work to generalize this.

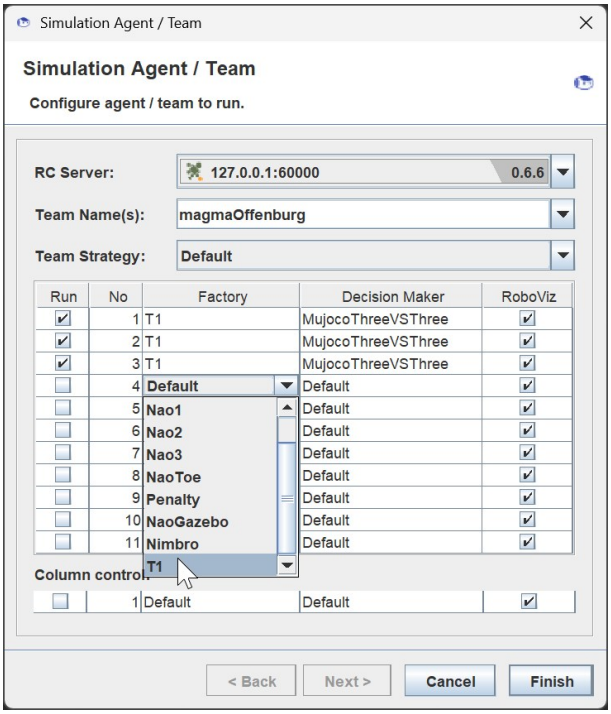

Fig. 6. MagmaDeveloper run agents dialog.

5.2 MagmaDeepLearning

The deep learning environment of team magma provides classes to run DRL learning tasks. The ServerController is a class used to automatically start and stop the simulation server in cases the server crashed or at startup. The class had to be subclassed by a SimsparkServerController and a MujocoServerController to handle, for example, the different name and command line arguments of the two servers.

The ServerCommander class encapsulates the commands that the learning component can send to the server via the monitor protocol. Commands include placing the ball or the robots. The class has been extended with a new move3DRotatePlayer() method that makes use of the newly available 3D position and orientation placement of robots in rcssservermj. It has been extremely useful for training the getup behaviors with initial positions already flat on the ground.

Finally, in order to train the getup behaviors, a MujocoGetupProblem class was newly created to specify the initial training setup, the reward functions and the decision maker for this learning task.

References

1. Bohlinger N, Dorer K (2023) A Deep Reinforcement Learning Library (Not Only) for RoboCup. Robot World Cup 2023, pages 228–239, Springer Nature.
2. Spitznagel M, Weiler D, Dorer K (2021) Deep Reinforcement Multi-Directional Kick-Learning of a Simulated Robot with Toes. In Proceedings of IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), IEEEXplore.
3. Klaus Dorer (2017) Learning to Use Toes in a Humanoid Robot. In Hidehisa Akiyama, Oliver Obst, Claude Sammut and Flavio Tonidandel, editors, RoboCup 2017: Robot World Cup XXI, Lecture Notes in Artificial Intelligence, pp. 168-179, Springer Verlag, Berlin.
4. Martin Baur, Klaus Dorer, Jens Fischer, Duy Nguyen, Carmen Schmider and David Weiler (2017) The magmaOffenburg 2017 RoboCup 3DSimulation Team. Team description paper RoboCup 2017.
5. Patrtick MacAlpine and Peter Stone (2017) Prioritized Role Assignment for Marking. In Sven Behnke, Daniel D. Lee, Sanem Sariel, and Raymond Sheh, editors, RoboCup 2016: Robot Soccer World Cup XX, Lecture Notes in Artificial Intelligence, pp. 306–18, Springer Verlag, Berlin.
6. João Cravo, Fernando Almeida, Pedro Henriques Abreu, Luís Paulo Reis, Nuno Lau, Luís Mota (2014) Strategy planner: Graphical definition of soccer set-plays. Data Knowledge Engineering 94, pp. 110-131.
7. Lau N, Lopes LS, Corrente G, Filipe N. Roles (2009) Positionings and set plays to coordinate a msl robot team. In: Proceedings of the 4th international workshop on intelligent robotics, IROBOT'09. Lecture notes in computer science, vol. 5816, Aveiro, Portugal, Springer, p. 323–37.