The 10th Workshop on Humanoid Soccer Robots
15th IEEE-RAS International Conference on Humanoid Robots, Seoul, Korea, November 2015

GOL - A Language to Define Tactics in Robot Soccer

Matthias Hofmann and Florian Gürster*

Abstract— **Humanoid soccer serves as a suitable benchmark problem for research in artificial intelligence and robotics, and has seen impressive progress in recent years. Thus, many teams participating in the Standard Platform League (SPL) of the RoboCup competition[1] were focusing on improving basic skills for many years. But flexible and fast behavior engineering incorporating team tactics still remains as a challenge, albeit the application of sophisticated team play, consisting of tactics and strategies are still evolving, especially on the robot hardware. In this paper, we introduce the novel language GOL (Group Organizing Language) to design tactical instructions in robot soccer. In addition, we show how the visual editor TOR (Tactic Organizer for Robots) facilitates behavior engineering. Finally, we evaluate the developed components by suitable experiments.**

I. INTRODUCTION AND MOTIVATION

Humanoid soccer made impressive progress on the hardware level and software level during the last decade. This includes basic skills in motion (e.g. kicking the ball), and cognition (e.g. localization). One of the next steps in the evolution of robot soccer is tactical team play. It becomes more and more important, and is no longer restricted to simulation.

The development of sophisticated team behavior usually requires major changes in the behavior code. Behavior engineers must model game situations, and proper actions to be taken in these situations. This is a complex, cumbersome, and error-prone process that requires a lot of testing and tuning. To facilitate these tasks, we introduce the language GOL. It enables users to easily define, and model tactical instructions within minutes. There are similarities between GOL, and soccer coaches in real soccer where tactics are explained to the players visually by using a board.

The remainder of the paper is structured as follows: Related work is briefly described in section II. The tactic description language GOL, including TOR, is discussed in section III. The evaluation in section IV consists of a use case analysis, and exemplary shows how GOL can be applied to compare different implementations of behaviors. Section V consists of a conclusion, and future work.

II. RELATED WORK

In robotics, there is a plethora of domain-specific languages to serve different requirements from particular application areas. Since this work is related to design tactics and behaviors in the context of robot soccer, we briefly summarize a selection of suitable programming languages.

Robotics Research Institute, TU Dortmund University, , Germany {forename.surname}@tu-dortmund.de

[1] http://www.robocup.org

In 1997, the programming language GOLOG was introduced [1]. In this language, concepts of logic programming similar to PROLOG were integrated. An important aim of developing GOLOG was the application of the situation calculus that increases the level of abstraction in robot programming. However, this approach showed limited applicability, mainly because the transition of the real world into a logical model is still one of the big challenges in artificial intelligence [2].

With the introduction of coach agents into the RoboCup simulation league, COACH UNILANG was developed by a group called *FC Portugal*, participating in the RoboCup 3D Simulation League[2] [3]. COACH UNILANG is a standard language for coaching robot soccer teams. It enables high-level and low-level coaching including tactics, formations used in each situation, and giving instructions. Moreover, the coach is able to send game-related information to the players such as statistics, and a model of the opponent team. The main difference between GOL and COACH UNILANG is that GOL is designed to be league-independent. GOL comes with visual editing tools as well.

In robot soccer behavior, another important language called XABSL was introduced by the GermanTeam of the former 4-legged Legaue in RoboCup [4],[5]. XABSL defines behavior by using hierarchical finite state machines. The core idea of programming with XALBSL is the option graph. Starting from the root option, several other options are called. Within each option, a finite state machine is defined, and each state consists of an action to be taken. For decisions, so called symbols are used. These variables must be implemented by the behavior engineer before they can be used in XABSL. It requires the compilation of source code by utilizing the *XABSLEngine*. This way, different XABSL behaviors can be exchanged separately without recompiling the source code of other components. XABSL is a popular and well-established solution for behavior programming, especially in the RoboCup community.

There is another XABSL implementation called CABSL[6] which is based on C++ macros. This way, behavior code is directly integrated into the software that controls the robot. An important advantage of CABSL is that variables and functions of other components can be used directly instead of creating a variety of symbols. Tactics in XABSL and CABSL can be defined by introducing symbols that provide tactical information. However, this is a time-consuming, and error-prone process.

[2] http://simspark.sourceforge.net/wiki/index.php/Main_Page

III. GOL - A SOCCER TACTICS DESCRIPTION LANGUAGE

In this section, we introduce the soccer tactics description language *GOL*. The technological basis for the language is JSON[3]. JSON is a lightweight data interchange format that is both, human-readable, and machine-readable. GOL defines a data structure which is called *tactic map*, and each tactic map consists of a set of *tactical instructions*. The entire tactic map is stored in one file. The JSON schema that defines the tactic map is publicly available on GitHub[4].

A. Tactical instructions

Tactical instructions define the particular behavior of a team of agents as a reaction on specific game situations, or game conditions. Tactical instructions consist of the following objects:

- A *set of preconditions* that decide whether the tactical instruction becomes active, i.e. which team behavior is currently executed. Subsection III-B describes the set of preconditions in more detail.
- A *representation of the ball*. This consists of the current ball position, and the prospective ball position. In this case, prospective means the desired position of the ball after executing the tactical instruction.
- A *representation of the own team players* and their current, and prospective positions on the field.
- A *representation of the opponent players*. There is no prospective value as the tactical instruction must not control the opponent team.
- A *group of team players*. Each group has exactly two elements, the group leader, and the follower. This way, the user may construct a formation of players. Furthermore, the distances in vertical, and horizontal direction between the leader and the follower are defined.

B. Preconditions and activation of tactical instructions

The decision whether a tactical instruction becomes active is defined by the preconditions that can be categorized as follows:

- *Organizational preconditions*: The user gives the name of the precondition, and the field type (e.g. Humanoids, SPL). He is able to add remarks on the specific tactical instruction.
- *Player-dependent preconditions*: This type consists of the number of own, and opponent players (number of own / opponent players, goalie). This way, it is possible to design a tactic map that will be activated when one team has less player on the field than the other.
- *Score-dependent preconditions*: The user is able to specify the goal difference. For example, if the own team takes the lead, it could play more defensive.
- *Time-dependent preconditions*: This category consists of the minute of play, the remaining time, and the half time can be specified.

Two fields in the precondition structure are mandatory: The name of the tactic map, and the name of the competition (e.g. the Standard Platform League, SPL in RoboCup).

C. Integration into the behavior control system

GOL is designed to easily integrate into existing robotic frameworks. Hence, developers need to implement the interface between GOL and their particular framework. This means in particular that all data that is required by GOL, needs to be provided properly. For instance, the positions of the robots on the field along with the ball position must be present, and continuously updated. A more detailed specification of all variables needed by GOL is part of the scheme file (see section III). The performance of GOL depends on the quality of data that is being fed into the system. Moreover, GOL can be extended by the developer according to the specific needs of the team, and requirements.

In summary, there are two important tasks to be implemented: Firstly, a *selection mechanism for the tactic instruction* that is going to be executed, has to be provided. We use the following policy: For each element present on the tactical instruction board, the corresponding data structure that is used and updated in the robotic framework (e.g. player, ball) has to be identified. The difference between the desired position specified by GOL, and the real position calculated by the robotic framework, is compared by utilizing the Euclidean. The tactical instruction board with the smallest deviation from the current game situation is selected and becomes active. Since not all possible situations are usually covered by the tactic map, a backup behavior is required. In this case, we use the standard RoboCup behavior.

Secondly, a *robot selection mechanism* must be implemented. Each robot is assigned a specific position where it has to move according to the team instruction. Note that the agents do always execute underlying behavior that is implemented (e.g. dribbling, or kicking the ball when the robot is in an appropriate position). For instance, the ball can be blocked by a robot when it rolls in its direction. For our experiments, we use the framework of team *NaoDevils* that is based on [6]. Both, the position selection mechanism, and the tactic map selection mechanism, offer a high degree of freedom. Thus, behavior engineers are free to specify behavior that is going to be undertaken by the individual robot.

D. TOR - A VISUAL TACTICS EDITOR

Although a manual definition of the tactic map is feasible, we introduce the visual tactic editor *Tactic organizer for robots, TOR*. This WYSIWIG[5] editor enables the user to define a tactic map within minutes. The tactic editor was written in C++, and can be obtained from GitHub [6].

Figure shows the situation editor, and figure the graphical user interface of TOR. The user is able to define the tactic map, the tactic instructions, and their entire properties (see

[3] `http://json.org/`

[4] `https://github.com/NaoDevils/GOL`

[5] what you see is what you get

[6] `https://github.com/NaoDevils/GOL/tree/master/TOR`

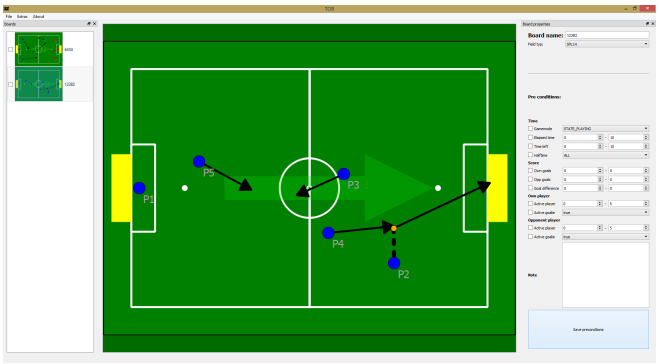

Fig. 1: The graphical user interface to edit tactical instructions.

subsections III-A, and III-B). The user-friendly editor features drag-and-drop, and consists of context-sensitive menus for editing objects on the field, and the field itself.

Fig. 2: The graphical user interface to edit field properties.

IV. EVALUATION

In order to show the functionality of tactic maps written in GOL, we conduct a series of test games. Note that the focus of the evaluation is the demonstration of the functionality of GOL as a language, and its applicability in game situations, and not on the performance of one particular tactic map that has been designed by a behavior engineer. Behavior engineers that would like to compare the performance of different tactics should play many more games to get resilient results. Therefore, the series of test games give a first impression of the potential, and usefulness of GOL. Moreover, the series of test games provide a basis to discuss how individual tactic maps can be scrutinized to define design guidelines and identify potential problems. In order to get an impression about the influence of the different tactic maps on the game, we provide movies of a selection of test games. They can be obtained from GitHub [7].

A. Experimental Setup

We use *SimRobot*[7] in the version 2014 for our experiments. The test games follow a league system where each behavior is playing against each other two times with alternating kickoff and a total game length of ten minutes. All test games are conducted with five versus five players, including one goalie for each team. A team gets three points for a win, one point for a draw, and zero points for a lose. Four different teams were used for the tests:

1) *NaoDevils (T1)*: This serves as the reference behavior. It has been successfully used in RoboCup 2014 by team *NaoDevils*. The team reached the quarterfinals in the competition.
2) *One defensive instruction (T2)*: This very simple tactic map consists of four team players that take very defensive positions around the own goal. The player nearest to the ball shoots it towards the center circle. There is no offensive action being undertaken. The naive tactic map is shown in Figure 3.
3) *Few instructions (T3)*: This tactic maps consist of four instructions. The first one is to define the initial positions before kickoff. The second instruction defines a kickoff behavior where the ball is moved to the outer part of the field. The robot behind the player conducting the kickoff is moving to the position where the ball is intended to be. A third instruction defines the defensive behavior. Here, the goal is to clear the ball as soon as possible (see Figure 4). The last instruction defines the offensive play of the team (see Figure 5). While two robots are defending to counter distance shots, two robots are close to the opponents' goal trying to obtain a goal.
4) *Six instructions (T4)*: This tactic map is the most complex one. The total time to create this map was about 10 minutes. It consists of more individual situations, e.g. if the opponent goal is blocked, or a critical situation. Figure 6 shows an offensive situation that is going to be executed.

B. Results and Discussion

The following tables I, and II show the results of the individual games, and the final ranking. It is notable that tactic map three was the most successful while the reference behavior is third. As expected, the defensive behavior (T2) did not score at all. Although T3 consists of less instructions than T4, T3 performed better. This is due to the fact that T4 ran occasionally into hysteresis problems. Moreover, the simulation environment does not take the complete set of game rules into account. For instance, penalties resulting from pushing, are not considered.

[7]https://github.com/NaoDevils/GOL/tree/master/Games

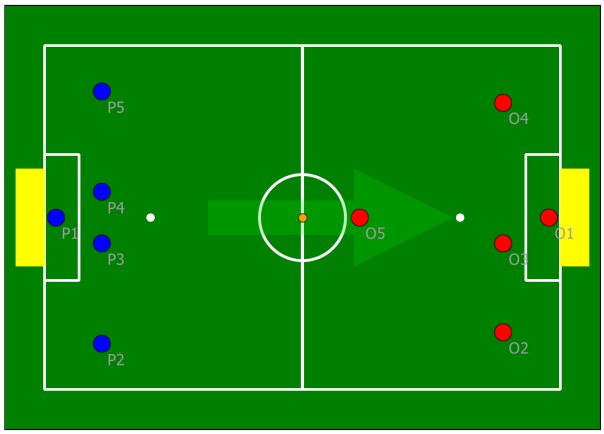

Fig. 3: The naive tactic map (T2).

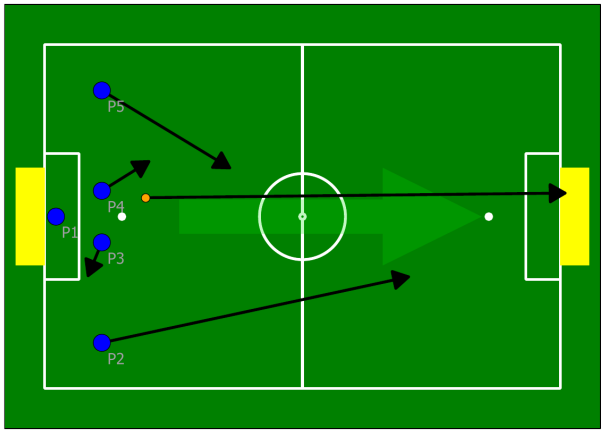

Fig. 4: Example of a defensive tactic instruction (T3).

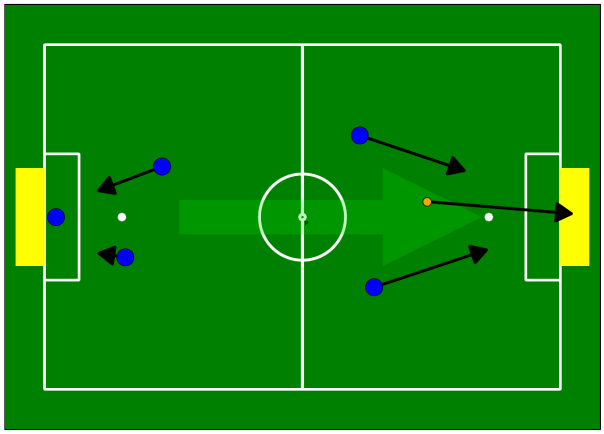

Fig. 5: Example of an offensive tactic instruction (T3).

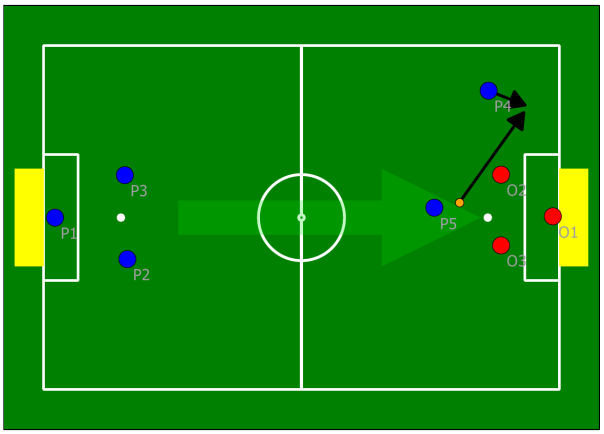

Fig. 6: Example of an offensive tactic instruction (T4).

Game	Blue	Red	Result
A1	T1	T2	0:0
A2	T3	T4	1:0
A3	T2	T1	0:1
A4	T4	T3	0:1
A5	T1	T3	1:0
A6	T2	T4	0:1
A7	T3	T1	1:1
A8	T4	T2	1:0
A9	T1	T4	1:1
A10	T2	T3	0:1
A11	T4	T1	1:0
A12	T3	T2	0:0

TABLE I: Game results.

It is recommendable that tactic maps are sufficiently different from each other to avoid oscillations in selecting the tactic map to become active. Thus, it is obvious that different situations for defensive situations, the midfield, and offensive play should be designed. Since tactic maps offer a high degree of freedom, behavior engineers are enabled for fast and easy changes in the tactical orders without changing the code. However, extensive testing is recommended to elaborate how tactic maps work together to increase the game performance.

It has to be noted that, with the current version of GOL, it is possible to give agents different instructions within one tactic map. This occurs when an instruction given for a group is different from the instruction given to individual agents. This has to be taken into account when analyzing and debugging the behavior.

Rank	Team	Goals	Diff.	Points
1	T3	4:2	2	11
2	T4	4:3	1	10
3	T1	4:3	1	9
4	T2	0:4	-4	2

TABLE II: Final Ranking.

V. CONCLUSION AND OUTLOOK

This paper introduced the tactics description language *GOL*, and the visual tactic map editor *TOR*. We have shown that GOL successfully integrates into an existing robotic framework. Moreover, we described a use case, conducted a series of test games to show the applicability of tactical instructions in robot soccer. Finally, we discussed design guidelines.

Future work will include a more detailed evaluation of GOL as language, and TOR as the complementing user interface. Possible setups for the performance evaluation are the upcoming RoboCup events. Users, mainly from the area of behavior engineering, will be asked to try GOL, and TOR, and to rate the language according to its usability, and usefulness.

For future work we will test the system on the physical robot. GOL is a promising approach to behavior engineering since once it has been integrated into the existing robotic framework, it rapidly increases the flexibility of the team behavior, and speeds up behavior design. Moreover, the language can be extended to be utilized by a coaching robot that observes the game from outside the field. The coach robot could select the tactical instruction within the tactic map, and communicate it to the active players on the field.

REFERENCES

[1] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl, "Golog: A logic programming language for dynamic domains," *The Journal of Logic Programming*, vol. 31, no. 1, pp. 59–83, 1997.

[2] F. Dylla, A. Ferrein, G. Lakemeyer, J. Murray, O. Obst, T. Röfer, F. Stolzenburg, U. Visser, and T. Wagner, "Towards a league-independent qualitative soccer theory for robocup," in *RoboCup 2004: Robot Soccer World Cup VIII*. Springer, 2005, pp. 611–618.

[3] L. P. Reis and N. Lau, "COACH UNILANG-a standard language for coaching a (robo) soccer team," in *RoboCup 2001: Robot Soccer World Cup V*. Springer, 2002, pp. 183–192.

[4] M. Lötzsch, J. Bach, H.-D. Burkhard, and M. Jüngel, "Designing agent behavior with the extensible agent behavior specification language xabsl," in *RoboCup 2003: Robot Soccer World Cup VII*. Springer, 2004, pp. 114–124.

[5] M. Loetzsch, M. Risler, and M. Jungel, "XABSL-a pragmatic approach to behavior engineering," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006, pp. 5124–5129.

[6] T. Röfer, T. Laue, J. Müller, M. Bartsch, M. J. Batram, A. Böckmann, M. Böschen, M. Kroker, F. Maaß, T. Münder, M. Steinbeck, A. Stolpmann, S. Taddiken, A. Tsogias, and F. Wenk, "B-human team report and code release 2013," Tech. Rep., 2013.

[7] T. Laue and T. Röfer, "Simrobot-development and applications," in *International Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*. Citeseer, 2008.