RoboCup Soccer Team using Jason

Vahid Farhadi, Mahsa Layeghi, Haozhi Liao, Luke Newton
Systems and Computer Engineering Department
Carleton University, Ottawa, Canada
{vahidfarahdi, mahsalayeghi, haozhiliao,
lukenewton}@cmail.carleton.ca

***Abstract*—This report presents the authors' RoboCup team implemented in Java using the Jason framework. Our solution contains three different types of agents - a goalie, forwards, and defenders - which mimic the different positions on a soccer team. Instructions are included for running the program, as well as an overview of the project architecture and agent behaviors.**

***Index Terms*—Software Agents, RoboCup, Jason, BDI.**

I. Introduction

RoboCup is an annual competition between autonomous soccer players with the eventual goal of producing a soccer team capable of defeating a human world cup champion team [1]. Numerous leagues of RoboCup exist, ranging from human-sized physical robots to fully simulated games. The focus of this project is on implementing a RoboCup team for the 2D-simulation league. In this league, teams produce five autonomous soccer players that connect to an external soccer server program. As communication with the server is done over a network, a team's implementation can use any programming language or software tool.

Our implemented RoboCup team uses Java with the Jason Belief-Desire-Intention (BDI) framework to produce a set of software agents. RoboCup poses a challenge for software agents as it provides an inaccessible, non-deterministic, continuous, and dynamic environment. BDI agents are well suited for this task, as their desires and intentions allow for planning and contingencies in an environment with such characteristics as RoboCup. Jason implements a variation of AgentSpeak [2], which provides a clear syntax for specifying BDI agent plans, and ties cleanly into the Java code to communicate with the RoboCup soccer server.

The rest of the report is outlined as follows. Section II presents the overall architecture of the software solution. Section III, IV, and V describe our offense, defense, and goalie agents - the three different player types on our team. Section VI provides instructions for packaging and running the code, and section VII concludes the report.

II. Overall Architecture

Figure 1 shows the overall structure of the program. The main elements of the program are the Player and RoboCupGame classes, the RoboCup server executable, the Jason framework, and the three Jason asl files.

An instance of the Player class represents one soccer player. A Player acts as a client that sends action messages to the RoboCup server and receives messages containing visual or auditory information. Every Player instance has a role - either forward, defender, or goalie - that determines which asl file is used for the Player instance's behavior. Visual and auditory information received from the server is encapsulated in VisualInfo or MessageInfo classes respectively. Each Player's auditory and visual information is converted into percepts by the Perceptor and RoboCupGame classes, the latter of which is a Jason environment object. The Jason framework uses the percepts of a particular Player, combined with the behaviors specified in the asl file that matches the Player's role, to determine what action the Player should take. This action is forwarded through RoboCupGame to the Player instance, which sends a message to the server to perform that action.

The soccer game itself is made up of several rounds of communication between Player clients and the soccer server. A typical round of this communication for one Player is as follows:

1) The RoboCupGame updates the percepts of a Player based on the contents of that Player's VisualInfo and MessageInfo objects.
2) The Jason framework uses these percepts in conjunction with the plans specified in the relevant asl file to determine which action the Player should take.
3) The decision by Jason is forwarded to the RoboCupGame, which forwards the action to the Player.
4) The Player converts this action into a message to send to the soccer server.
5) RoboCupGame sleeps for a short time until the next simulation step.

In parallel with this process is another continuous loop in the Player which simply listens for incoming messages from the soccer server so the Player can update its VisualInfo and MessageInfo objects as needed.

III. Forward Role

Forward players are the team's offense. They try to reach the ball and move towards the opponent side of the field, and then kick the ball towards the net to score a goal. When forward players connect to the soccer server before the game starts, they are placed randomly on the half of their side of the field closest to the ball.

Initially, a forward player starts the game by looking for the ball. Since there are multiple forward players in the game, they all head to the ball to reach it. Therefore a mechanism of message passing between forward players is needed. When a forward player reaches the ball, it sends out a message to nearby players with a "say" action containing the "gotBall" message. Now, other players with a suitable distance would be able to hear this message. There are multiple messages coming both from referees and players. Messages are also being heard by all other opponent players as well. Therefore, forwards need to check both the message's contents and sender to perform a suitable action.

As soon as a forward player receives a "gotBall" message from another forward on its own team, our strategy is to stop the forward from looking for a ball, instead sending him towards the opponent's goal and to stop somewhere close to the net.

The forward with the ball dribbles the ball and heads toward the goal. If there are no opponents or teammates visible in their field of view, the player continues dribbling and then kicks the ball towards the goal.

The forward with the ball also continuously checks for the existence of a team player and an opponent player. If there is an opponent close to the forward with the ball, or there is another forward player that is closer to the goal, the forward with the ball will try to pass the ball. To do this, the forward with the ball scans its left and right views and then passes the ball once turning towards a teammate.

IV. Defender Role

Defender soccer agents play the role of defense in a soccer game. Considering "!start." as an initial desire of the agent, which refers to the start of the game, the event "+!start" is added in the queue of events to be handled by the agent. There are three relevant plans for this event depending on the connection to the server. If the agent has not connected to the server, it should wait until connection. Otherwise, the agent is placed at an initial defender position in the

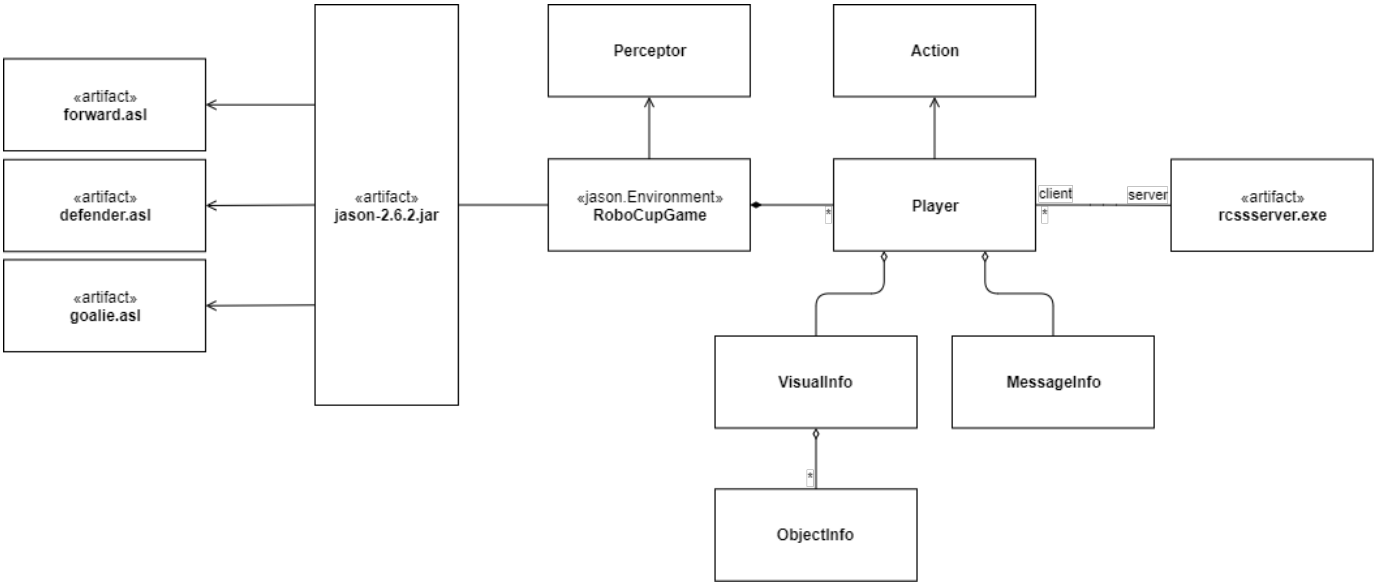

Fig. 1. Software Structure

soccer field, provided that the game has not started. The player should not be moved once connection to the server is established if the game has already started once the agent gets connected. Once the player is connected and optionally placed, a new intention of defending the ball is created. Then, a new event "+!waitDefendBall" is added. As for the context of this event, two cases are considered. If the ball is not visible to the agent, it turns 40 degrees and keeps "!waitDefendBall" as its intention. If the agent sees the ball in a far distance and in a different direction, it turns to be in the same direction and it will still have the same intention as before. If the ball gets close enough to the agent, the agent will dash towards the ball and add the "+!reachBall" event.

When the plan "+!reachBall" starts to be executed, there are some possibilities. If the ball becomes no longer visible, the agent turns 40 degrees and still intends to reach the ball. If the ball is visible and accessible, the agent turns and goes to reach it. If the ball gets too far away through the actions of some other player, the ball is considered inaccessible and a new intention "!runBackAtOwnPosition" is introduced which tells the agent to go back to its defender position - an arc with radius 30 around their team's goal. Finally, if the agent gets access to the ball, it's time to defend. In this case, a set of plans are defined. The agent looks for the opponent's goal ("+!findOpponentGoal"), tries to find another player in that field of vision ("+!findCenterPlayer"), or on its right-hand side ("+!findRightPlayer") or on its left-hand side ("+!findLeftPlayer"), and pass the ball provided that the found player is a teammate. If it can't find any teammates, a plan "+!findOpponentGoaltoKick" is set and the agent kicks the ball towards the opponent's goal. Whether plan "+!passBall" or "+!kickBall" is executed, the next applicable plan is "+!runBackAtOwnPosition". It means that the agent goes back to the defender position and waits to defend the ball again. This is how a reasoning cycle loop is executed for a defender agent.

Figure 2 shows a defender having reached the ball on their side of the field. This defender will kick the ball towards one of the other teammates on the field, and then run back to the defender position shown in red as an arc around the goal.

V. Goalie Role

The client player must append "(goalie)" to its initial connection message to the RoboCup soccer server to inform the server that the client player is a goalie. This allows the goalie to use the "catch" command in addition to the actions that other players can perform.

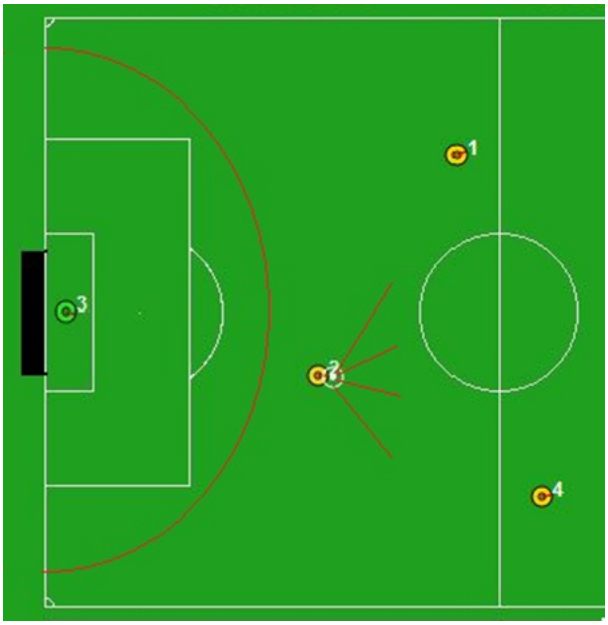

Fig. 2. Defender Map Showing Defense Position Arc

The goalie starts at the assigned position right in front of their team's goal (-50, 0). The goalie has a "saving range" defined as a radius of 22 in front of the assigned position which covers the majority of the penalty box. This is shown in Figure 3. Once the game starts, the goalie stays at the assigned position and keeps focus to the moving ball by turning to the ball direction if the ball is not within the saving range.

Once the ball is moved close within the 22 radius, the goalie starts to chase the ball in order to perform the catch action. When the goalie starts to chase the ball in order to catch, the ball may be moved out from the saving range by another player. To make sure the goalie does not leave the penalty box, it checks the following three flags' distance on the field:

1) flag c t

2) flag c
3) flag c b

Once the goalie's position is less than a distance of 36 to 'flag c' or 40 to 'flag c t' or 'flag c b', it returns to the assigned position.

After reaching the ball with distance less than one, the goalie tries to catch the ball with its direction. The catch outcome by the RoboCup server is probability based which means the catch will not always be successful. The goalie falls back to the assigned position if a catch fails. If the goalie successfully catches the ball, the game mode changes to 'free_kick_l' or 'free_kick_r' depending on the player's team side. During the free_kick mode, the goalie instantly moves to the assigned position. To continue the game, the goalie free kicks the ball to the first found team player. If there is no visible player, the goalie long kicks the ball to the opponent's goal direction.

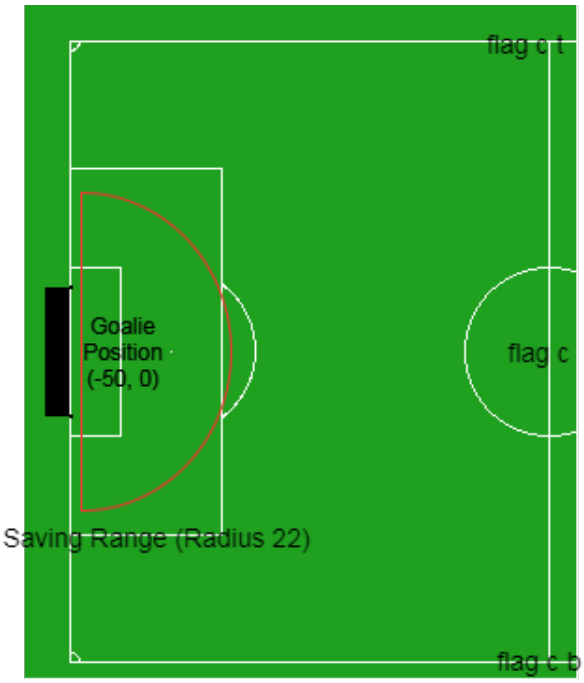

Fig. 3. Goalie Map Showing Saving Range and Flags Checked

VI. Compilation and Execution Instructions

The project source code is submitted alongside this report, and is available online on the authors' GitHub repository[1]. Jason uses Apache Ant to package the program and generates a build.xml for compile and jar Java sources. There are two ways to build an executable jar of the RoboCup project: either in the command line or using jEdit. In the command line, Apache Ant has to be installed first by following the installation manual [3], then run the command:

```
ant -f \$robocup\_project\_path/bin/build.xml
    jar
```

In the build.xml, the 'jasonJar' property has to be set to the path of the local Jason framework executable for the project jar to be successfully created. Set:

```
<property name="jasonJar" value="
    $path_to_jason\libs\jason-2.6.3.jar"/>
```

where $path_to_jason is the path to the folder containing the Jason framework.

[1]https://github.com/liaohaozhi/robocup_project

Using jEdit is an easier option because the Ant jar is included within the Jason library. Open the '$robocup_project_path/roboCupTeam.mas2j' in jEdit and from the menu select

```
Plugins -> Jason -> Create an executable jar
```

to create an executable jar.

The server port is hard coded to 6000 in the static field PORT_NUMBER of class RoboCupGame. Running the project with a server of different port other than 6000 needs to recompile and jar the project.

To run the project, simply execute the jar file by either double-clicking the file in the file explorer or through the command line with

```
java -jar roboCupTeam.jar
```

VII. Conclusion

In this report we presented a RoboCup team for the 2D simulation league which makes use of various features of the Jason BDI framework. We created three different types of agents - forward, defender, and goalie - which cooperate and coordinate their actions to be an effective team.

References

[1] M. Chen, K. Dorer, E. Foroughi, F. Heintz, Z. Huang, S. Kapetanakis, K. Kostadis, J. Kummeneje, J. Murray, I. Noda, and et al., "Robocup soccer server," Feb 2003. [Online]. Available: https://sourceforge.net/projects/sserver/files/rcssmanual/9-20030211/

[2] R. H. Bordini, J. F. Hübner, and M. Wooldridge, *Programming multi-agent systems in AgentSpeak using Jason*. John Wiley & Sons, 2007, vol. 8.

[3] "Installing apache ant." [Online]. Available: https://ant.apache.org/manual/install.html