The Transformation Tool Soccer Worldcup
The TTC 2014 Live Contest Case

Tassilo Horn
horn@uni-koblenz.de

Louis Rose
louis.rose@york.ac.uk

Christian Krause
me@ckrause.org

July 18, 2014

Abstract

This paper describes the TTC 2014's Live Contest Case. The task is to implement a soccer client using your favorite transformation tool. The game is network-based. A provided soccer server awaits two clients and then starts the turn-based match. In each turn, the clients receive an EMF model representing the current soccer pitch with all players and the ball. They then need to analyze that model, and send back to the server an EMF update model describing the actions of their players.

The TTC soccer world champion will be evaluated by letting the submitted clients play soccer against each other. There is also an Expert Judges' Choice award as perceived by the TTC audience with respect to quality properties such as understandability, expressiveness of the tool, conciseness, etc., and there is the overall Live Contest winner where the audience has to balance the match results with the perceived quality of the solution.

1 Introduction

The Soccer Worldcup in Brazil is over, but we're still hungry for more matches, so what could be more obvious than having our own little Transformation Tool Soccer Worldcup? Participants should use their favorite transformation tool to implement a soccer client that analyzes a soccer pitch model to emit an update model representing their team's actions.

The TTC Soccer World Champion will be determined by letting the participants' clients play against each other. Next to that, we give an award to the most elegant solution as perceived by the TTC audience. Here, quality properties such as understandability, expressiveness and adequacy of the used tool, conciseness, extensibility, etc. should be taken into account. Finally, there will be an award for the Overall Live Contest Winner where the success in the championship has to be balanced with the perceived quality properties.

In contrast to prior TTC Live Contest cases, this year's case is very open. There is no strict right and wrong, but there are many different possible strategies that might or might not be successful in the competition.

2 Case Description

In this section, the Live Contest case is discussed in detail. The task for participating teams is to implement a *soccer client* using their favorite transformation tool. A corresponding *soccer server* and also a reference soccer client for testing purposes are provided (see Section 3.1).

The game is network-based where communication is performed over plain sockets. The simple protocol is described in Section 2.4. The soccer server waits for two clients to connect. The first client to connect will control the blue soccer team, the second client will control the red soccer team. A coin toss decides whose team's goal keeper gets the ball for kickoff initially.

Then, the match starts in a turn-based manner. First the team currently owning the ball may make its moves, then the other team. Therefore, the server sends a SoccerPitch model to the client which describes the current state of the soccer pitch (see Section 2.1). The client needs to analyze this model, and then send back to the server an Update model containing the actions of its team's players (see Section 2.2). The server validates the Update model and updates its soccer pitch model accordingly. The updated pitch is then sent to the other client so that it can plan its actions.

If a team shoots a goal, all players are reset to their initial positions and the goal keeper of the non-scoring team receives the ball.

A match consists of 200 turns[1]. If scores are even after this number of turns, the team scoring the next goal wins (*golden goal*).

2.1 The SoccerPitch Metamodel

The Ecore metamodel for the soccer pitch is shown in Figure 1.

A SoccerPitch consists of Fields, Players, and a Ball. The fields are layed out in a grid of width 45 and height 25. Each field has an xPos and a yPos determining its location in the grid. Via the references north, south, west and east one can navigate to the field above, below, left of, and right of. A special kind of fields are GoalFields which belong to either the BLUE or the RED team. The blue team has 7 adjacent goal fields y-centered at xPos = 0, and the red team has 7 adjacent goal fields y-centered at xPos = 44.

All players belong to either the BLUE or the RED team and have a number. In each team, there is a GoalKeeper with number = 1 and eight FieldPlayers with numbers 2 to 9.

Each player references the field he is standing on using the field reference. There may be multiple players located at the same field. If the player currently has the Ball, he references it using the ball reference and the field references of the player and the ball target the same field. It is also possible that currently no player owns the ball.

Figure 2 illustrates an excerpt of a SoccerPitch model (excluding the SoccerPitch root element).

The top-left field has the coordinates $(0, 0)$, the bottom-right field has the coordinates $(44, 24)$. The blue goal fields have the coordinates $\{(0, y) \mid y \in [9, 15]\}$, the red goal fields

[1]The number of turns in a match may be adapted depending on client speed and number of participants

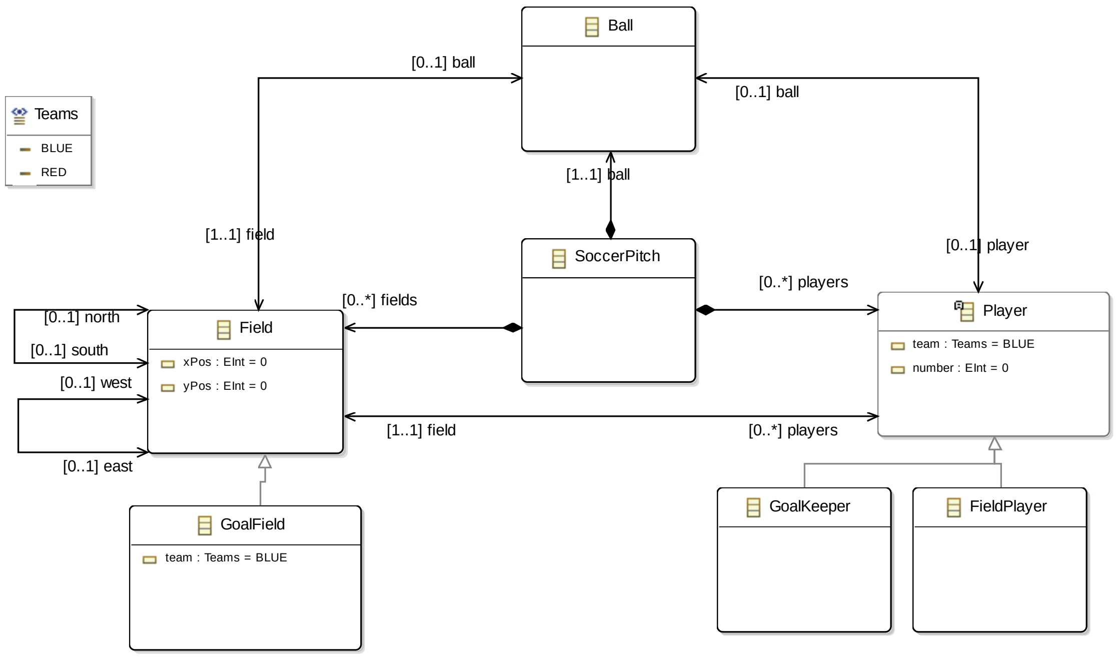

Figure 1: The SoccerPitch metamodel

have the coordinates $\{(44, y) \mid y \in [9, 15]\}$. So the blue team plays from left to right and the red team in the other direction.

In the illustration, the blue goal keeper stands on the goal field $(0, 11)$, and the red field player number 4 stands on the field $(22, 12)$ and owns the ball.

Figure 3 shows as screenshot of the soccer pitch as rendered by the soccer server.

All players are at their initial kickoff positions, and the ball hasn't been created and assigned to some player so far. After each goal, the players are switched back to those exact positions, and the goal keeper of the non-scoring team receives the ball.

2.2 The Update Metamodel

Figure 4 shows the Update metamodel whose instances are sent from the two clients to the server. These update models describe the moves of a team's players for the current turn.

The root Update element contains a list of Actions. Each action is concerned with exactly one player identified by the playerNumber. There are two concrete types of actions: MovePlayer and ShootBall.

A MovePlayer action tells the server to update the position of the team's player with number == playerNumber. The target field is specified as xDist and yDist distances relative to the player's current field's xPos and yPos coordinates. Thus, a MovePlayer action with xDist = 3 and yDist = -2 moves a player standing on field $(7, 14)$ to field $(10, 12)$, e.g., in

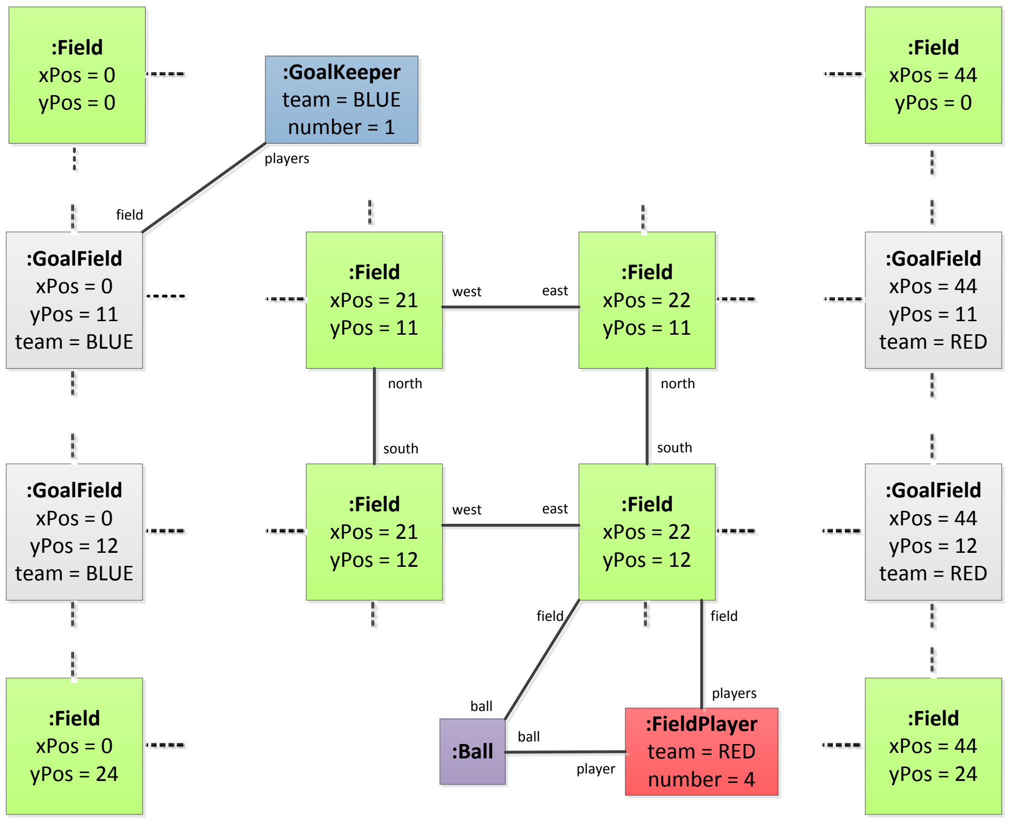

Figure 2: An sketch of an instance SoccerPitch model

top-right direction.

A ShootBall action tells the server to let the currently ball-owning player with number == playerNumber shoot the ball in the direction specified by xDist and yDist.

There are several constraints on the Update model like maximum move and shoot distances. Those are discussed in the next section which also depicts how the soccer server evaluates the Update model's actions.

2.3 The Soccer Game Rules

In this section, the constraints on the Update model checked by the server and the gameplay rules implemented by it are discussed.

2.3.1 Constraints on the Update Model

Next to the structural constraints implied by the metamodel, there are several more constraints concerning the Update model that are checked by the soccer server and avenged strictly when disobeyed.

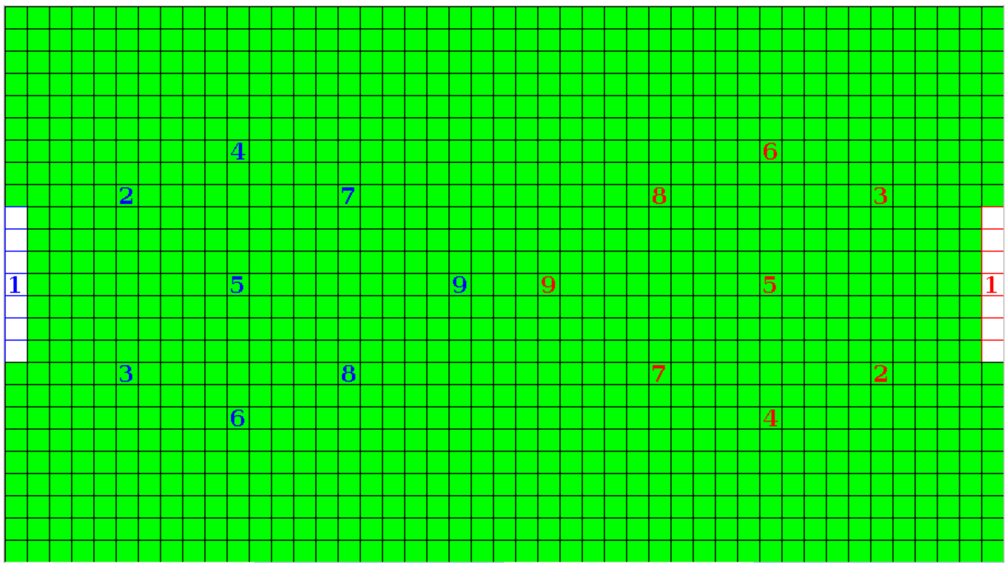

Figure 3: Rendering of the SoccerPitch model by the server

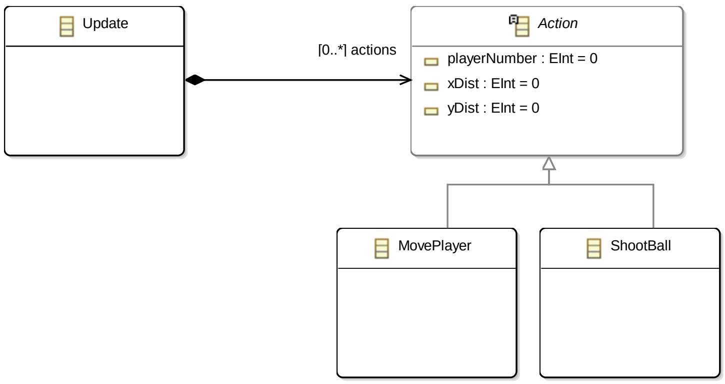

Figure 4: The Update metamodel

1. There may be at most one action per player and per turn. If the second action of a player is encountered, that action is skipped and the player gets a red card meaning the player is removed from the soccer pitch.

2. The player owning the ball can move at most 2 fields in both x and y direction. Violation of this rule is avenged with a red card.

3. A player without ball can move at most 3 fields in both x and y direction. Violation of this rule is avenged with a red card.

4. A goal keeper can only move on his own team's goal fields. If he tries moving out of his goal, he receives a red card.

5. Moving a player off the soccer pitch is avenged with a red card.
6. A shot can move the ball at most 7 fields in both x and y direction. Violation of this rule is avenged with a red card.
7. There may be at most 4 shoot ball actions in an Update model. The players trying to perform the fifth and later shots receive a red card. Since four shots are allowed, complex multi-passes are still possible, but it's not possible to cross the complete pitch in a single turn.
8. A shoot ball action for a player not owning the ball is simply skipped without penalty. The reason is that when performing a multi-pass, e.g., player 1 shoots the ball to player 2, and that player immediately shoots to player 3, the client cannot know that the first pass actually arrives at player 2. There is a chance that some opponent player intercepts the pass (see the gameplay rules below).
9. Shooting the ball off the soccer pitch is avenged with a red card for the shooting player.

2.3.2 Gameplay Rules

The gameplay rules define how the soccer server evaluates the move player and shoot ball actions sent by the soccer clients.

MovePlayer When the soccer server evaluates a move player action, it first checks the constraints 2 to 5 listed above and acts accordingly. If the player owns the ball, he is moved to the target field specified by xDist and yDist. If there is an opponent player within a distance less than one from the line of the move, there is a 30% chance that this player intercepts the ball. If not, the ball is placed at the target field and the original player still owns it. If the target field is one of the opponent's goal fields, the CheckGoal rule is performed.

If the player of a move action doesn't own the ball, the constraints are checked in the same way and the player is placed at the target field. If the target field contains the ball but no other players, the current player captures the ball. If the target field contains the ball-owning opponent player and that is not the goal keeper, the opponent is attacked and there is a 50% chance that the current player captures the ball.

ShootBall When the soccer server evaluates a shoot ball action, the constraints 6 to 9 are checked first. Then the target location of the shot is computed as specified with the xDist and yDist attributes. If there is an opponent player within a distance less than one from the line of the shot, there is a 30% chance that this player intercepts the ball. If not, the ball is placed on the target field. If the target field is one of the opponent's goal fields, the CheckGoal rule is performed. If the target is no goal field but other players are located there, a random choice decides which of them receives the ball.

CheckGoal If the ball finds its way to an opponent goal field as a result of a move or a shot and neither the opponent's goal keeper nor one of the opponent's field players is located there, the action's player scores a goal for his team. If the goal keeper is on the target field, he has a 70% chance of catching the ball. Opponent field players on the target field have a 20% chance of capturing the ball each.

2.4 The Soccer Protocol

The protocol between client and server is socket-based and very simplistic. It is illustrated in Figure 5.

```
  Server                                     Client
  ======                                     ======
{listening}                                    |
      |                                        |
      |<- - - - - - -{connect} - - - - - - - - |
      |                                        |
      |-<TEAM>-------------------------------->|
      |<--------------------------------<NAME>-|
      |                                        |
      |            {repeatedly}                |
   +---------------------------------------------+
   |  |-<SR><SoccerPitch><ER>----------------->| |
   |  |<----------------------<SR><Update><ER>-| |
   +---------------------------------------------+
      |                                        |
      |- - - - - - - -{close} - - - - - - - - >|
```

Figure 5: The Soccer Protocol

The server listens on some port awaiting two clients. When a client connects, the server sends it the TEAM that client will control. The first client's team is BLUE, the second client's team is RED.

As response, the client sends its NAME to the server. Preferably, solutions should use the name of the transformation tool they are using as the name of their client.

As soon as two clients have connected and TEAM and NAME have been exchanged, the actual match starts. The ball-owning team's client receives a SoccerPitch model and sends an Update model as a response. Then the other team's client receives the updated SoccerPitch model and sends its Update model. This cycle repeats until the match winner stands firm, i.e., after 200 or more turns depending if a golden goal is needed to determine the winner.

The models are send as EMF XMIResources and wrapped in a start marker SR and an end marker ER whose values are `#START_RESOURCE#` and `#END_RESOURCE#`. Section 3.3 provides sample Java methods that can be used to send and receive the models as required by the protocol.

After the match finished, the server simply closes the client sockets.

3 Additional Information

3.1 Artifacts

You can download the two Ecore metamodels and the soccer server from the following `http://www.transformation-tool-contest.eu/livecontest.html`.

There is also a soccer client implemented with FunnyQT[2] which you can compete with while developing your own solution.

3.2 Discussions about the Case

When you have questions concerning the case, you can ask me (Tassilo Horn) personally. I'll be around the STAF conference the complete week.

There is also a mailing list (Google Group) for posting questions about the case or discussing it. The list address is ttc14-live-contest@googlegroups.com, and it can also be used via the web interface at `https://groups.google.com/forum/#!forum/ttc14-live-contest`.

3.3 Sending and Receiving Models

As discussed in Section 2.4, the SoccerPitch model and the Update model are sent and received as XMIResources, i.e, in their XMI representation, via the connection socket's input and output streams. They are wrapped in some start and end marker strings as given in Listing 1.

```java
public static final String START_MARKER = "#START_RESOURCE#";
public static final String END_MARKER = "#END_RESOURCE#";
```

Listing 1: Markers for wrapping resources

The method for sending an XMIResource is shown in Listing 2.

```java
private void sendResource(XMIResource r) {
	// socket is the Socket connected to the Server
	// out is new PrintWriter(new OutputStreamWriter(socket.getOutputStream()));
	out.println(START_MARKER);
	try {
		Map<Object, Object> opts = r.getDefaultSaveOptions();
		opts.put(XMLResource.OPTION_FORMATTED, false);
		r.save(out, opts);
	} catch (IOException e) {
		e.printStackTrace();
		throw new RuntimeException(e);
	}
	out.println(END_MARKER);
	out.flush();
}
```

Listing 2: Sending an XMIResource

[2]FunnyQT homepage: `http://jgralab.github.io/funnyqt/`

The method for receiving an XMIResource is shown in Listing 3.

```java
private XMIResource receiveResource() {
	// socket is the Socket connected to the Server
	// in is new BufferedReader(new InputStreamReader(socket.getInputStream()));
	try {
		String s = in.readLine();
		if (s == null) {
			throw new RuntimeException("Connection lost.");
		}
		if (!s.equals(START_MARKER)) {
			throw new RuntimeException("Bad message from server.");
		}
		StringBuilder sb = new StringBuilder();
		while (!(s = in.readLine()).equals(END_MARKER)) {
			sb.append(s);
		}
		s = sb.toString();
		XMIResource r = new XMIResourceImpl();
		r.load(new ByteArrayInputStream(s.getBytes()), r.getDefaultLoadOptions());
		return r;
	} catch (IOException e) {
		e.printStackTrace();
		throw new RuntimeException(e);
	}
}
```

Listing 3: Receiving an XMIResource

3.4 Running the Soccer Server

The soccer server is distributed as a runnable JAR (see Section 3.1 above). Running it without arguments prints its synopsis:

```
% java -jar SoccerServer.jar
Usage:

  java -jar SoccerServer.jar <PORT> <TURNS> <SLEEP_TIME>

<PORT>          The port the server listens on for clients.
<TURNS>         The number of turns in a match.
<SLEEP_TIME>    The pause time (in ms) between player actions.
```

When the soccer server is started, an new soccer pitch with all players at their initial positions is created and displayed with a simple GUI. The server then waits for two clients to connect by listening on the port specified by the argument PORT.

When two clients have connected, a coint toss determines whose team's goal keeper receives the ball and kicks off. Then the game starts for at least as many turns as specified by the TURNS argument.

The SLEEP_TIME argument slows down the game so that each player's action in a turn can be observed better in the server's GUI.

3.5 Running the Reference Client

Like the soccer server, the reference soccer client is distributed as a runnable JAR file (see Section 3.1 above). Executing it without arguments prints its synopsis:

```
% java -jar ttc14-soccer-client.jar
Loading soccerpitch model...
Loading update model...
Usage:
  java -jar ttc14-soccer-client.jar <host> <port>
```

The host argument specifies the host name or IP address of the machine running the soccer server. The port argument specifies the port the soccer server is listening on.

3.6 Solution Submission

The soccer clients should be submitted **until Thursday, July 24th, 12pm (i.e., noon, time for lunch)**. To submit your client, you should commit and push it to the Git repository at `https://github.com/TransformationToolContest/ttc14-live-contest-solutions`. Please follow the instructions given at that site. Most importantly, the solution must be runnable in order to participate in the TTC Soccer Worldcup, and it must be documented how to do so.