Tiro Quick Start

for Tiro version 2.0b and later

Markus Bergkvist
markus.bergkvist@bth.se

October 12, 2003

1 Introduction

This document gives a brief description on what Tiro is, and how you can start developing your own RoboCup player[1] with Tiro as the foundation.

Tiro is dependent on the `CRaPI` and the `SituationDeviser` libraries. CRaPI provides the communication with the game server, a world model based on the visual input, and some basic commands and actions the agent can perform as request to the server. This document is mostly concerned with the SituationDeviser, but read *Simulated RoboCup - Creating a generic API* [2], for more details on CRaPI.

The Tiro project is merely a way of starting your reasoning agents, hence Tiro contains no logic on how to play RoboCup soccer. Default component used by Tiro to enable playing RoboCup soccer is the SituationDeviser, but it can easily be replaced with the component of your choice.

2 SituationDeviser

The SituationDeviser contains *Situation rules*, *Formations*, *Roles*, *Zones* and *Behaviours*.

Each formation is composed of 11 roles and is linked to a situation rule. When a rule fires, the linked formation is said to be the interim formation. The SituationDeviser then selects the role, in the interim formation, the agent has based on the agents *role id*.

Linked to the role is a set of behaviours and a zone. The zone is used to make the agents concentrate their play in certain areas of the field in each situation. This avoids “kiddie-soccer”, and makes the game smoother as the agents can change the home location dependent on the current situation.

When the SituationDeviser decides what action to take each cycle, the behaviours in the current role is evaluated, and the behaviour with the highest priority (of the behaviours where the pre-condition is met) is then run.

[1]A player for the RoboCup simulation league (`http://sserver.sourceforge.net/`)
[2]`http://crapi.sourceforge.net/articles/`

3 Database overview

The `SituationDeviserSettings` database contains the *Situation rules*, *Formations*, *Roles*, *Zones* and *Behaviours* the SituationDeviser reads when it initializes. Any changes in the database will therefore affect the performance of the agent.

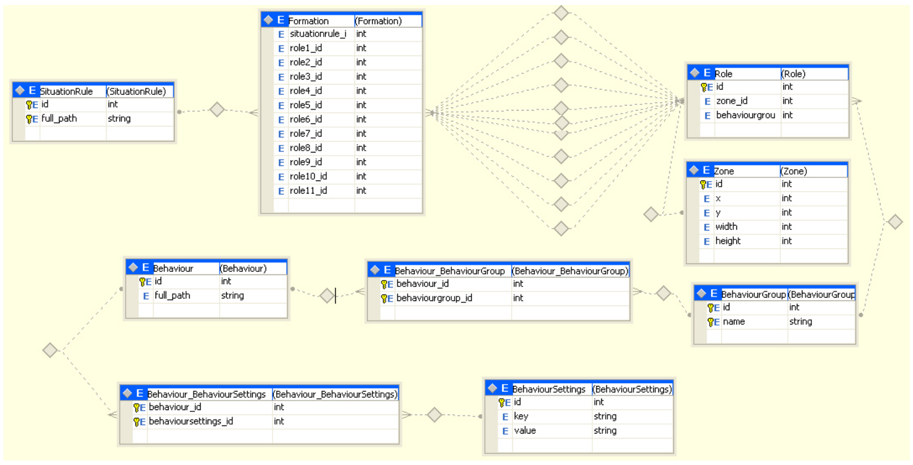
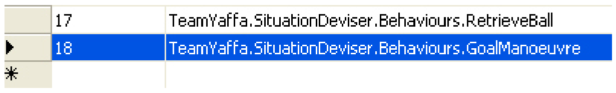


Figure 1: ER-model of SituationDeviserSettings

4 Example: Extend the database

The default database (created by running `Tiro.exe -m`) is rather minimal and needs to be extended in most of the tables to get a full team working. This example describes what must be added to the database for the agent to handle the `GoalOur`-situation. It is assumed that there exist a `GoalManoeuvre` behaviour which makes the agents do their "goal dance".

The first thing that must be done is to add an entry in the database that tells the full path to the behaviour. Do this by entering a new line in the `Behaviour` table with the next unique id and the full path (see Fig. 2).

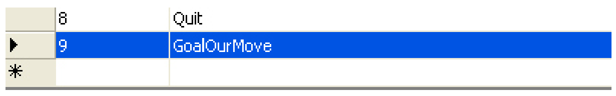


	17	TeamYaffa.SituationDeviser.Behaviours.RetrieveBall
▶	18	TeamYaffa.SituationDeviser.Behaviours.GoalManoeuvre
*		

Figure 2: Adding the path to the GoalManoeuvre behaviour to the database.

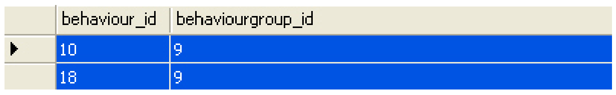
Next thing to do is to decide what set of behaviours that will run in the situation. In `GoalOur` it is aptly to use `LineUp` and `GoalManoeuvre`. Add a new entry in the `BehaviourGroup` table (see Fig. 3) and link the behaviours with the behaviour group in the `Behaviour_BehaviourGroup` table (see Fig. 4).

Before a new formation can be added to the database, the roles in that formation must exist.



8	Quit
9	GoalOurMove

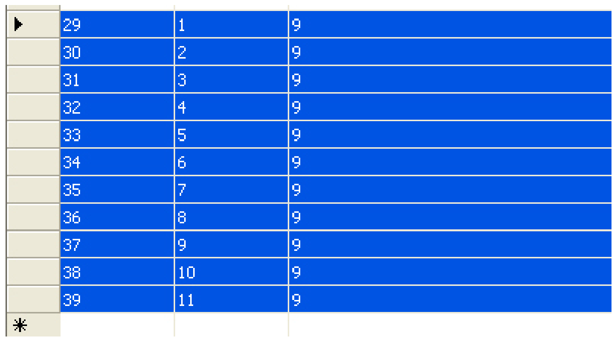
Figure 3: Adding the behaviour group GoalOurMove to the database.



behaviour_id	behaviourgroup_id
10	9
18	9

Figure 4: Linking the behaviour group GoalOurMove with the behaviours LineUp and GoalManoeuvre.

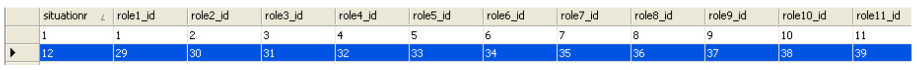
It is wise to have the agents "dance" at the same location they have when it is kick off, therefore the zones from the `BeforeKickOff` formation is reused (zone id 1-11), and all the agents will have the GoalOurMove behaviour group. The `role` table will therefore look like Fig. 5 after the roles are added.



29	1	9
30	2	9
31	3	9
32	4	9
33	5	9
34	6	9
35	7	9
36	8	9
37	9	9
38	10	9
39	11	9

Figure 5: The Role table after the roles for the GoalOur formation are added.

Finally the Formation is created by linking the situation rule (rule id 12) with the roles (role id 29-39), see Fig. 6.



situationr	role1_id	role2_id	role3_id	role4_id	role5_id	role6_id	role7_id	role8_id	role9_id	role10_id	role11_id
1	1	2	3	4	5	6	7	8	9	10	11
12	29	30	31	32	33	34	35	36	37	38	39

Figure 6: The formation for the GoalOur situation.

5 Example: Create your own Deviser

When creating a new Deviser the abstract methods `Evaluate()` and `Act()` must be implemented. `Evaluate` is called by Tiro at the beginning of a new cycle, directly followed by a call to `Act`.

To make Tiro use the new Deviser you need to alter the `createDeviser()` method in Tiro so an instance of the new Deviser is created.

By extending the SituationDeviser (see Fig. 7) the new Deviser can use the SituationDeviser to evaluate the current situation. Based on the current situation it can then do its own reasoning about the next action to perform.

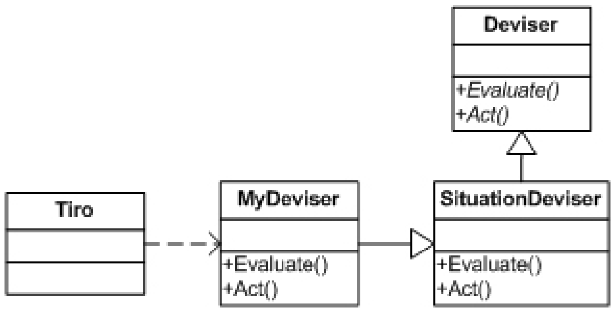


Figure 7: An example on how to use another Deviser in Tiro, while preserving the capabilities of the SituationDeviser.