Multi-Agent Programming Contest Java-Agents Description (2012 Edition)

http://www.multiagentcontest.org/2012/

Tristan Behrens Jürgen Dix Jomi Hübner
Michael Köster Federico Schlesinger

April 17, 2012

1 Overview

Again, we provide a set of very simple agents that function as a proof-of-concept and as dummy-agents for testing. On top of that we also show how to use the environment-interface EISMASSim (see the EISMASSim description for further details).

This document will show you two things: 1. how configure and execute our dummy agents, and 2. how to create new agents. Note, however, that we do not encourage you to create your own agent team from scratch using the agent-infrastructure we are providing. Our agent-infrastructure is rather limited and we are not optimistic that we will have enough time to add new features, if the need for such features arises. Nevertheless, if you accept this, feel free to use the agent-infrastructure.

2 Running Our Dummy-Agents

In the software package we have included a single agent-configuration (see below). It sets up two teams `A` and `B`. Each team has 10 agents.

In order to run the dummy agents, navigate to the `javaagents/scripts` directory and execute

```
./startAgents.sh
```

You will then be asked to select a configuration. At the time of the release you can only select a single configuration. If you add new ones you can select them as well. After selecting a configuration, the environment-interface will immediately establish the connections to the *MASSim*-server, which must be already up and running, as specified in the environment-interface configuration-file, and execute the agents.

3 Changing the Configuration

If you desire another configuration that is different from ours, please perform these steps:

1. create a new subfolder at `javaagents/scripts/conf`, you can define an arbitrary name, which will be made selectable by the `startAgents.sh` shell-script,
2. copy the two XML-files from `javaagents/scripts/conf/dummyteam/` to the directory you have just created,
3. if necessary adapt the `eismassimconfig.xml`-file (see the EISMASSim description for instructions), and
4. adapt the `javaagentsconfig.xml`-file, according to the following instructions.

Figure 1 shows an exemplary agents configuration-file. Here, the `<agent>`-tag is the most relevant one. Its attributes are:

- `name` is the agent's name, which is required for communication and registering to the environment-interface,
- `entity` is the name of the connector/vehicle the agent is supposed to control, as specified in the configuration file of the environment-interface,
- `class` is the class-name of the agent, which will be dynamically loaded via Java-reflection, and
- `team` is the team-name of the agent, which is required for communication[1].

The `class`-tag is the most relevant one, because this is the place to plug in different agents. There are a couple of unmentioned agents in the package `massim.javaagents.agents2011` (see the accompanying javadoc for their descriptions and the code of their implementations). On top of that you can also specify your own agents here.

[1]An agent can only receive messages from and send messages to its team-members.

```
<?xml version="1.0" encoding="UTF-8"?>
<javaAgentsConfig>
  <agents>
    <agent name="A1" entity="cA1" class="massim.javaagents.agents2011.SimpletonBuyerAgent" team="A"/>
    <agent name="A2" entity="cA2" class="massim.javaagents.agents2011.RechargingFighterAgent" team="A"/>
    <agent name="A3" entity="cA3" class="massim.javaagents.agents2011.RechargingFighterAgent" team="A"/>
    <agent name="A4" entity="cA4" class="massim.javaagents.agents2011.RechargingFullExplorerAgent" team="A"/>
    <agent name="A5" entity="cA5" class="massim.javaagents.agents2011.RechargingFullExplorerAgent" team="A"/>
    <agent name="A6" entity="cA6" class="massim.javaagents.agents2011.RechargingFullExplorerAgent" team="A"/>
    <agent name="A7" entity="cA7" class="massim.javaagents.agents2011.RechargingFullExplorerAgent" team="A"/>
    <agent name="A8" entity="cA8" class="massim.javaagents.agents2011.RechargingFullExplorerAgent" team="A"/>
    <agent name="A9" entity="cA9" class="massim.javaagents.agents2011.RechargingFullExplorerAgent" team="A"/>
    <agent name="B1" entity="cB1" class="massim.javaagents.agents2011.SimpletonBuyerAgent" team="B"/>
    <agent name="B2" entity="cB2" class="massim.javaagents.agents2011.RechargingFighterAgent" team="B"/>
    <agent name="B3" entity="cB3" class="massim.javaagents.agents2011.RechargingFighterAgent" team="B"/>
    <agent name="B4" entity="cB4" class="massim.javaagents.agents2011.RechargingFullExplorerAgent" team="B"/>
    <agent name="B5" entity="cB5" class="massim.javaagents.agents2011.RechargingFullExplorerAgent" team="B"/>
    <agent name="B6" entity="cB6" class="massim.javaagents.agents2011.RechargingFullExplorerAgent" team="B"/>
    <agent name="B7" entity="cB7" class="massim.javaagents.agents2011.RechargingFullExplorerAgent" team="B"/>
    <agent name="B8" entity="cB8" class="massim.javaagents.agents2011.RechargingFullExplorerAgent" team="B"/>
    <agent name="B9" entity="cB9" class="massim.javaagents.agents2011.RechargingFullExplorerAgent" team="B"/>
  </agents>
</javaAgentsConfig>
```

Figure 1: An exemplary agents configuration-file.

4 Creating Your Own Agents

In order to create and use your own agents you are required to perform these steps:

1. create a new agent-class that inherits from `massim.javaagents.Agent`,
2. implement a constructor and a couple of required methods,
3. incorporate your new agent-class into the `javaagentsconfig.xml`,
4. make sure that your new agent-class is in the class-path, and
5. execute.

Figure 2 shows a very rudimentary agent without any functionality. It is not necessary to extend the constructor, unless you have to add some useful things. You have to add code to the `handlePercept`-method if you intend to handle percepts-as-notifications. Please note two things: 1. handling percepts-as-notifications is optional, that is there are other means to retrieve percepts, and 2. you have to explicitly activate percepts-as-notifications in the environment-interface configuration file (see the EISMASSim description) if you intend to use them. The `step`-method is automatically called by the interpreter that executes all agents. It is supposed to return an action, which will then be executed automatically. Note that if this method returns `null`, no message is sent to the server. The `step`-method is the place where you are supposed to add your agent's intelligence.

Finally, we will introduce a couple of methods that might be useful (see the javadoc of `massim.javaagents.Agent` for the full overview):

- `Collection<LogicBelief> getBeliefBase()` yields the current belief-base (immutable),

```
public class YourAgent extends Agent {

        public YourAgent(String name, String team) {
                super(name, team);
                // TODO do something if necessary
        }

        @Override
        public void handlePercept(Percept p) {
                // TODO handle percepts if necessary
        }

        @Override
        public Action step() {
                // TODO deliberate and return an action
                return null;
        }

}
```

Figure 2: A new agent without any functionality.

- `Collection<LogicGoal> getGoalBase()` yields the current goal-base (immutable),
- `Collection<Percept> getAllPercepts()` yields all percepts that are currently available (the EISMASSim description contains an overview),
- `Collection<Message> getMessages()` yields all messages that have been sent to the agent,
- `void sendMessage(LogicBelief belief, String receiver)` sends a message to a recipient,
- `LinkedList<LogicBelief> getAllBeliefs(String predicate)` returns all beliefs that have a given predicate,
- `void removeBeliefs(String predicate)` removes all beliefs that have a given predicate,
- `void removeGoals(String predicate)` removes all goals that have a given predicate,
- `void addBelief(LogicBelief belief)` adds a belief to the belief-base,
- `void addGoal(LogicGoal goal)` adds a goal to the goal-base,

- `boolean containsBelief(LogicBelief belief)` returns true if the belief-base contains a given belief,
- `boolean containsGoal(LogicGoal goal)` returns true if the goal-base contains a given goal,
- `void clearBeliefs()` empties the belief-base, and
- `void clearGoals()` empties the goal-base.