Multi-Agent Programming Contest Protocol Description (2009 Edition)

Tristan Behrens   Mehdi Dastani   Jürgen Dix
Michael Köster   Peter Novák

June 18, 2009

1   Scenario: Cows and Herders

In this year's agent contest the participants have to compete in an environment for cows. Each team controls a set of herders (up to ten agents) in order to direct the cows into their own corral. The team with the most cows in the corral at the end wins the match.

2   General Agent-2-Server Communication Principles

The agents from each participating team will be executed locally (on the participant's hardware) while the simulated environment, in which all agents from competing teams perform actions, runs on the remote contest simulation server.

Agents communicate with the contest server using standard TCP/IP stack with socket session interface. The Internet coordinates (IP address and port) of the contest server (and a dedicated test server) will be announced later via the official Contest mailing list.

Agents communicate with the server by exchanging XML messages. Messages are well-formed XML documents, described later in this document. We recommend using standard XML parsers available for many programming languages for generation and processing of these XML messages.

3   Communication Protocol Overview

Logically, the tournament consists of a number of matches. A match is a sequel of simulations during which two teams of agents compete in several different settings of the environment. However, from agent's point of view, *the tournament consists of a number of simulations in different environment settings and against different opponents.*

   The tournament is divided into three phases:

1. the initial phase,

2. the simulation phase, and

3. the final phase.

   During the initial phase, agents connect to the simulation server and identify themselves by username and password (AUTH-REQUEST message). Credentials for each agent will be distributed in advance via e-mail. As a response, agents receive the result of their authentication request (AUTH-RESPONSE message) which can either succeed, or fail. After successful authentication, agents should wait until the first simulation of the tournament starts.

   Here is a picture of the initial phase (UML-like notation):
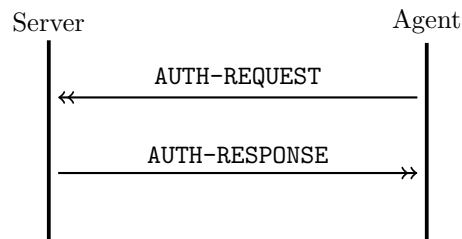
   At the beginning of each simulation, agents of the two participating teams are notified (SIM-START message) and receive simulation specific information:
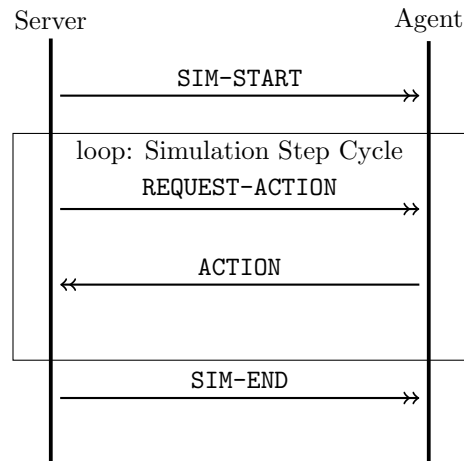
- simulation ID,

- opponent's ID,

- grid size,

- corral position and size,

- line of sight, and

- number of steps the simulation will last.

In each simulation step each agent receives a perception about its environment (`REQUEST-ACTION` message) and should respond by performing an action (`ACTION` message). Each `REQUEST-ACTION` message contains

- information about the cells in the visibility range of the agent (including the one agent stands on),

- the agent's absolute position in the grid,

- the current simulation step number,

- the current number of cows in the team's corral, and

- the deadline for responding.

The agent has to deliver its response within the given deadline. The action message has to contain the identifier of the action, the agent wants to perform, and action parameters, if required.

Here is a picture of the simulation phase:

When the simulation is finished, participating agents receive a notification about its end (`SIM-END` message) which includes

- the information about the number of caught cows, and

- the information about the result of the simulation (whether the team has won or lost the simulation).

All agents which currently do not participate in a simulation should wait until the simulation server notifies them about either 1) the start of a simulation, they are going to participate in, or 2) the end of the tournament.

At the end of the tournament, all agents receive a notification (`BYE` message). Subsequently the simulation server will terminate the connections to the agents.

Here is a picture of the final phase:

Server   Agent

|———————————————— BYE ————————————————»|
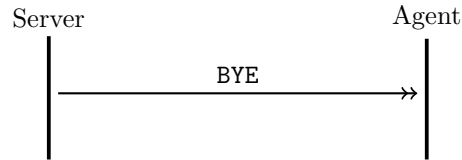
3.1   Reconnection

When an agent loses connection to the simulation server, the tournament proceeds without disruption, only all the actions of the disconnected agent are considered to be empty (*skip*). Agents are themselves responsible for maintaining the connection to the simulation server and in a case of connection disruption, they are allowed to reconnect.

Agent reconnects by performing the same sequence of steps as at the beginning of the tournament. After establishing the connection to the simulation server, it sends `AUTH-REQUEST` message and receives `AUTH-RESPONSE`. After successful authentication, server sends `SIM-START` message to an agent. If an agent participates in a currently running simulation, the `SIM-START` message will be delivered immediately after `AUTH-RESPONSE`. Otherwise an agent will wait until a next simulation in which it participates starts. In the next subsequent step when the agent is picked to perform an action, it receives the standard `REQUEST-ACTION` message containing the perception of the agent at the current simulation step and simulation proceeds in a normal mode.

Here is a picture:

Server   Agent

«———————————————— AUTH-REQUEST ————————————————|

|———————————————— AUTH-RESPONSE ————————————————»

|———————————————— SIM-START ————————————————»
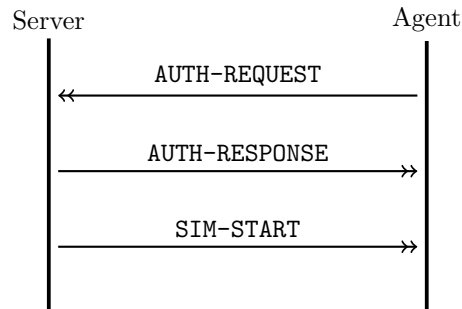
3.2 XML Messages Description

3.2.1 XML message structure

XML messages exchanged between server and agents are zero terminated UTF-8 strings. Each XML message exchanged between the simulation server and agent consists of three parts:

- Standard XML header: Contains the standard XML document header

  `<?xml version="1.0" encoding="UTF-8"?>`

- Message envelope: The root element of all XML messages is `<message>`. It has attributes the timestamp and a message type identifier.

- Message separator: Note that because each message is a UTF-8 zero terminated string, messages are separated by nullbyte.

The timestamp is a numeric string containing the status of the simulation server's global timer at the time of message creation. The unit of the global timer is milliseconds and it is the result of standard system call "time" on the simulation server (measuring number of milliseconds from January 1st, 1970 UTC). Message type identifier is one of the following values: `auth-request`, `auth-response`, `sim-start`, `sim-end`, `bye`, `request-action`, `action`, `ping`, `pong`.

Messages sent from the server to an agent contain all attributes of the root element. However, the timestamp attribute can be omitted in messages sent from an agent to the server. In the case it is included, server silently ignores it.

Example of a server-2-agent message:

```
<message timestamp="1138900997331" type="request-action">
  <!-- optional data -->
</message>
```

Example of an agent-2-server message:

```
<message type="auth-request">
  <!-- optional data -->
</message>
```

Optional simulation specific data According to the message type, the root element `<message>` can contain simulation specific data.

3.2.2 `AUTH-REQUEST` (agent-2-server)

When the agent connects to the server, it has to authenticate itself using the username and password provided by the contest organizers in advance. This way we prevent the unauthorized use of connection belonging to a contest participant. `AUTH-REQUEST` is the very first message an agent sends to the contest server.

The message envelope contains one element `<authentication>` without subelements. It has two attributes `username` and `password`.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message type="auth-request">
  <authentication username="xteam5" password="jabjar5"/>
</message>
```

3.2.3 `AUTH-RESPONSE` (server-2-agent)

Upon receiving `AUTH-REQUEST` message, the server verifies the provided credentials and responds by a message `AUTH-RESPONSE` indicating success, or failure of authentication. It has one attribute `timestamp` that represents the time when the message was sent.

The envelope contains one `<authentication>` element without subelements. It has one attribute `result` of type string and its value can be either `"ok"`, or `"fail"`. Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message timestamp="1204979083919" type="auth-response">
  <authentication result="ok"/>
</message>
```

3.2.4 `SIM-START` (server-2-agent)

At the beginning of each simulation the server picks two teams of agents to participate in the simulation. The simulation starts by notifying the corresponding agents about the details of the starting simulation. This notification is done by sending the `SIM-START` message.

The data about the starting simulation are contained in one `<simulation>` element with the following attributes:

- `id` - unique identifier of the simulation (string),

- `opponent` - unique identifier of the enemy team (string),

- `steps` - number of steps the simulation will perform (numeric),

- `gsizex` - horizontal size of the grid environment (west-east) (numeric),

- `gsizey` - vertical size of the environment (north-south) (numeric),

- `corralx0` - left border of the corral (numeric),

- `corralx1` - right border of the corral (numeric),

- `corraly0` - upper border of the corral (numeric),

- `corraly1` - lower border of the corral (numeric).

Remark: One step involves all agents acting at once. Therefore if a simulation has n steps, it means that each agent will receive `REQUEST-ACTION` messages exactly n times during the simulation (unless it loses the connection to the simulation server).

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message timestamp="1204979126544" type="sim-start">
  <simulation
    corralx0="0"
    corralx1="14"
    corraly0="55"
    corraly1="69"
    gsizex="70" gsizey="70"
    id="stampede"
    lineOfSight="8"
    opponent="xteam"
    steps="10"/>
</message>
```

3.2.5 SIM-END (server-2-agent)

Each simulation lasts a certain number of steps. At the end of each simulation the server notifies agents about its end and its result.

The message envelope contains one element `<sim-result>` with two attributes `score` and `result`. `score` attribute contains number of caught in the corral of the team the given agent belongs to, and `result` is a string value equal to one of `"win"`,`"lose"`,`"draw"`. The element `<sim-result>` does not contain subelements.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message timestamp="1204978760356" type="sim-end">
  <sim-result result="draw" score="9"/>
</message>
```

3.2.6 BYE (server-2-agent)

At the end of the tournament the server notifies each agent that the last simulation has finished and subsequently terminates the connections. There is no data within the message envelope of this message.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message timestamp="1204978760555" type="bye"/>
```

3.2.7 REQUEST-ACTION (server-2-agent)

In each simulation step the server asks the agents to perform an action and sends them the corresponding perceptions.

The message envelope of the REQUEST-ACTION message contains the element `<perception>` with six attributes:

- `step` - current simulation step (numeric),

- `posx` - column of current agent's position (numeric),

- `posy` - row of current agent's position (numeric),

- `score` - number of cows in the corral of the agent's team at the current simulation step (numeric),

- `deadline` - server global timer value until which the agent has to deliver a reaction in form of an ACTION message (the same format as timestamp),

- `id` - unique identifier of the REQUEST-ACTION message (string).

The element `<perception>` contains a number of subelements `<cell>` with two numeric attributes `x` and `y` that denote the cell's relative position to the agent.

If an agent stands near the grid border, or corner, only the perceptions for the existing cells are provided.

Each element `<cell>` contains a number of subelements indicating the content of the given cell. Each subelement is an empty element tag without further subelements:

- `<agent>` - there is an agent in the cell. The string attribute `type` indicates whether it is an agent of the enemy team, or ally. Allowed values for the attribute `type` are `"ally"` and `"enemy"`.

- `<obstacle>` - the cell contains an obstacle. This element has no associated attributes.

- `<cow>` - the cell contains a cow. The string attribute `ID` represents the cow's unique identifier.

- `<corral>` - the cell is a corral cell. The string attribute `type` indicates whether it belongs to the team's or the opponent's corral. Allowed values for the attribute `type` are `"ally"` and `"enemy"`.

- `<switch>` - the cell contains a switch.

- `<fence>` - the cell contains a segment of a fence. Allowed values for the attribute `open` are `"true"` and `"false"`.

- `<empty>` - the cell is empty.

- `<unknown>` - the content of a cell is not provided by the server because of information distortion.

The specific rules on allowed combinations of objects in a cell are provided in the scenario description.

Example (compare to Fig. 1):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<message timestamp="1243942024822" type="request-action">
<perception deadline="1243942032822" id="1" posx="16" posy="20" score="0" step="0">
<cell x="-8" y="-8">
<empty/>
</cell>
<cell x="-8" y="-7">
<empty/>
</cell>
<cell x="-8" y="-6">
<empty/>
</cell>
<cell x="-8" y="-5">
<obstacle/>
</cell>
<cell x="-8" y="-4">
<obstacle/>
</cell>
<cell x="-8" y="-3">
<obstacle/>
</cell>
<cell x="-8" y="-2">
<obstacle/>
</cell>
<cell x="-8" y="-1">
<obstacle/>
</cell>
<cell x="-8" y="0">
<obstacle/>
</cell>
```

```xml
<cell x="-8" y="1">
<obstacle/>
</cell>
<cell x="-8" y="2">
<obstacle/>
</cell>
<cell x="-8" y="3">
<obstacle/>
</cell>
...
<cell x="-7" y="-4">
<corral type="ally"/>
</cell>
<cell x="-7" y="-3">
<corral type="ally"/>
</cell>
<cell x="-7" y="-2">
<corral type="ally"/>
</cell>
<cell x="-7" y="-1">
<corral type="ally"/>
</cell>
<cell x="-7" y="0">
<corral type="ally"/>
</cell>
<cell x="-7" y="1">
<corral type="ally"/>
</cell>
<cell x="-7" y="2">
<corral type="ally"/>
</cell>
...
<cell x="-1" y="-4">
<fence open="false"/>
</cell>
<cell x="-1" y="-3">
<fence open="false"/>
</cell>
<cell x="-1" y="-2">
<fence open="false"/>
</cell>
<cell x="-1" y="-1">
<fence open="false"/>
</cell>
<cell x="-1" y="0">
<fence open="false"/>
```

```
</cell>
<cell x="-1" y="1">
<fence open="false"/>
</cell>
<cell x="-1" y="2">
<switch/>
</cell>
...
<cell x="2" y="-3">
<cow ID="0"/>
</cell>
<cell x="2" y="-2">
<empty/>
</cell>
...
<cell x="5" y="-2">
<agent type="enemy"/>
</cell>
...
<cell x="8" y="8">
<empty/>
</cell>
</perception>
</message>
```

Note that the agent perceives an area that is a square with the size 17 with the agent in the center (see Fig. 1). Thus each agent is able to see 289 cells. We refrained from depicting all 289 cells in the above example and showed just some of the relevant cells instead. The three dots indicate the missing `<cell>` elements.
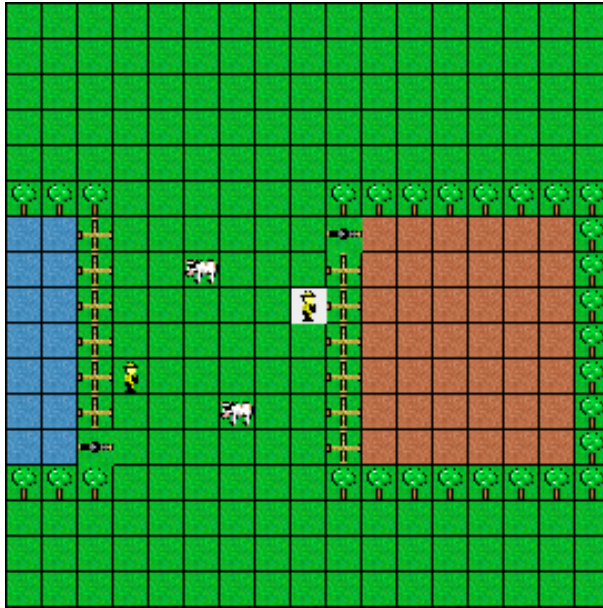
Figure 1: The view range of the agents. The agent in the center perceives all depicted cells.

3.2.8 `ACTION` (agent-2-server)

The agent should respond to the `REQUEST-ACTION` message by an action it chooses to perform.

The envelope of the `ACTION` message contains one element `<action>` with the attributes `type` and `id`. The attribute `type` indicates an action the agent wants to perform. It contains a string value which can be one of the following strings:

- `"skip"` – (the agent does nothing),

- `"north"` – (the agent moves one cell to the top) ,

- `"northeast"` – (the agent moves one cell to the top and one cell to the right),

- `"east"` – (the agent moves one cell to the right),

- `"southeast"` – (the agent moves one cell to the right and one cell to the bottom),

- `"south"` – (the agent moves one cell to the bottom),

- `"southwest"` – (the agent moves one cell to the bottom and one to the left),

- "west" – (the agent moves one cell to the left),

- "northwest" – (the agent moves one cell to the left and one to the top).

The attribute id is a string which should contain the REQUEST-ACTION message identifier. The agents must plainly copy the value of id attribute in REQUEST-ACTION message to the id attribute of ACTION message, otherwise the action message will be discarded.

Note that the corresponding ACTION message has to be delivered to the time indicated by the value of attribute deadline of the REQUEST-ACTION message. Agents should therefore send the ACTION message in advance before the indicated deadline is reached so that the server will receive it in time.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<message type="action">
  <action id="70" type="skip"/>
</message>
```

4 Remarks and notes

4.1 Ill-formed messages

Not well-formed XML messages received by the server from agents are discarded. This means, that if some obligatory information (element, or attribute) of a given message is missing the server silently ignores it. In the case that a message will contain additional not-required informations, only the first occurence is processed by the server. We strongly recommend to comply with the communicatison protocol described above.

Examples:

```
<?xml version="1.0" encoding="UTF-8"?>
<message type="auth-request">
  <authentication username="team1agent1" password="qwErTY"/>
  <authentication username="team1agent32" password="11111WWw"/>
  <some-element arbitrary="234TreE"/>
</message>
<message type="action">
  <authentication username="team1agent1" password="qwErTY"/>
  <authentication username="team1agent1" password="qwErTY"/>
  <some-element arbitrary="234TreE"/>
</message>
```

The message above will be processed as an AUTH-REQUEST message with the credentials team1agent1/qwErTY.