Multi-Agent Programming Contest
MASSim Server Manual
(2013 Edition)

http://www.multiagentcontest.org/2013/

Jürgen Dix Michael Köster Federico Schlesinger

June 6, 2013

New in 2013: Differences between the last year and 2013 are now marked with boxes.

Contents

1 Starting the Programs

For Microsoft Windows we suggest that you install MSYS[1] or Cygwin[2] in order to run the *MASSim*-software taking advantage of the shell scripts.

1.1 *MASSim* Server

You can start the *MASSim* server by invoking this:

```
$ ./startServer.sh
```

New in 2013: We mention the files explicitly

You will then be prompted to choose a simulation. The server generates XML files, statistics etc. Please have a look at the folders (`output` and `backup` that were generated during a run.

1.2 *MASSim* Monitor

In parallel you can also start the monitor which will allow you to observe the current simulation. The monitor can be invoked like this:

```
$ ./startMarsMonitor.sh
```

The monitor has currently two different options for the visualization. Please have a look at the scenario description for details. Also note, this monitor provides you with complete information. Your agents on the other hand do not have access to complete information.

The monitor also stores the match on hard disk. You can view these files by invoking:

```
$ ./startMarsFileViewer.sh /path/where/the/files/are
```

1.3 *MASSim* Web Server

New in 2013: We have a working install script

For the tournament we always provide a web server that is running on an Apache using *Apache Tomcat* and RMI as well as XML and XSLT. This server is not needed for the development of your multi-agent system, however, for the sake of completeness we provide some information how to install it. A install script that explains the procedure is placed in `scripts/tools/`. If you have any questions please contact the organizers.

[1] http://www.mingw.org/wiki/MSYS
[2] http://www.cygwin.com/

2 Configuring *MASSim*

When starting *MASSim*, you must provide a configuration file to the server. Configuration files are XML-based, and a set of configuration files is already available in the `scripts/conf` sub-folder of your *MASSim* installation. A detailed explanation of the configuration file is given next.

2.1 General Configuration

The general structure of the configuration file is depicted in Fig. 1. Note, however, that we use some additional XML features that allows us to use more than one file and reuse parts of the XML in different parts of the configuration. Therefore, you have to look into the main file as well as into the corresponding `config.dtd` file.

New in 2013: You can set a starting time with `time` now. Also, a `debug-level` was introduced. Additionally, we describe some parameters in more detail.

```
<?xml version="1.0" encoding="UTF-8"?>
<conf   backuppath="backup"
        launch-sync-type="key"
        reportpath="./backup/"
        time="14:05"
        time-to-launch="10000"
        tournamentmode="0"
        tournamentname="Mars2013 "
        debug-level="normal" >
    <simulation-server>
        <network-agent backlog="10" port="12300"/>
    </simulation-server>
    <match>
        <simulation ...> ... </simulation>
        ...
        <simulation ...> ... </simulation>
    </match>
    ...
    <match>
        ...
    </match>
    <accounts>
        ...
    </accounts>
</conf>
```

Figure 1: General structure of the *MASSim* configuration-file.

Tag: `conf`. The attributes of the `conf` tag are the following:

- `backuppath` - The path where important information of each simulation step is stored.
- `launch-sync-type` - Determines whether the server is started by pressing ENTER or after a certain time defined in `time-to-launch` or at time point (defined by `time`). The value can be `key`, `timer` or `time`.

New in 2013: Time introduced.

- `reportpath` - The path where the overall tournament results are stored.
- `time` - The time point for the option `time`.

New in 2013:

- `time-to-launch` - The time for the option `timer`.
- `tournamentmode` - Defines the structure of the tournament. `0` sets it to a round robin tournament. `1` is used when only one team should play against all others. Finally, `2` allows one to set up all matches manually. You have to add some code similar to the one in Fig. 2 after `</match>` to make it work.

New in 2013: Mode 2 added

```
<manual-mode>
  <match team1="A" team2="B"/>
  <match team1="A" team2="C"/>
  <match team1="B" team2="D"/>
</manual-mode>
```

Figure 2: Manual mode.

- `tournamentname` - Sets the tournament name to this value.
- `debug-level` - Changes the verbosity of the output on shell. Allowed values: `debug`, `normal`, `critical`, `error`.

New in 2013: Debugging functionality added.

Tag: `simulation-server`. The `simulation-server` tag has one child which has two attributes. `backlog` defines the time intervals (in milliseconds) for printing the debug messages to `stdout` or `stderr` respectively. The attribute `port` sets the port of the server.

Tag: `match`. A configuration can have one or more `match` tags, that will be instantiated depending on the `tournamentmode` attribute.

Tag: `accounts`. Finally, the `accounts` tag contains the details about the agents that are allowed to take part in the matches.

2.2 Simulation Configuration

The simulation tag is used to specify the scenario to be run, along with all the parameters that affect the simulation.

Tag: `simulation`. The attributes available for the `simulation` tag are the following:

- `id` - An identifier for the simulation. To distinguish among different instances of the simulation executed during a tournament, this identifier will be appended to the names of the teams taking part in that instance.
- `simulationclass` - The name of the main Java class implementing the scenario. For the 2013 Mars Scenario, the class that must be used here is `massim.competition2013.GraphSimulation`.
- `configurationclass` - The name of the Java class that will hold the configuration data specified in the `configuration` child tag. For 2013 Mars Scenario, the class to use is `massim.competition2013.GraphSimulationConfiguration`.
- `rmixmlobsserverhost` - The host to which the scenario monitor should connect.
- `rmixmlobsserverport` - The port to which the scenario monitor should connect.
- `rmixmlobserver` - The name of the Java class that will translate the current scenario state into XML data, and send it via RMI to the scenario monitor when connected. For the 2013 Mars Scenario, the class to use is `massim.competition2013.GraphSimulationRMIXMLDocumentObserver`.
- `xmlstatisticsobserver` - This is needed for the *Apache Tomcat* `Connection Status` and `Results` page. **New in 2013:** Added.
- `rmixmlobserverweb` - This is needed for the *Apache Tomcat* `Current Simulation` and `Results` page. **New in 2013:** Added.
- `visualisationobserver` - This defines the class for the visualization. **New in 2013:** Added.
- `visualisationobserver-outputpath` - This defines the output path for the visualization files. **New in 2013:** Added.
- `xmlobserver` - This allows one to store the results of a simulation as xml file. **New in 2013:** Added.
- `xmlobserverpath` - This is the path for the `xmlobserver`. **New in 2013:** Added.
- `statisticsobserver` - This generates a lot of useful statistics about a simulation. **New in 2013:** Added.
- `statisticsobserverpath` - This sets the path for the `statisticsobserver`. **New in 2013:** Added.

```
<simulation ...>
    <configuration ...>
        <actions>
            <action .../>
            <action .../>
            ...
        </actions>
        <roles>
            <role ...>
                <actions>
                    <action .../>
                    <action .../>
                    ...
                </actions>
                <actionsDisable>
                    <action .../>
                    <action .../>
                    ...
                </actionsDisable>
            </role>
        </roles>
        <achievements>
            <achievement .../>
        </achievements>
    </configuration>
    <agents>
        <agent ...>
            <configuration .../>
        </agent>
        <agent ...>
            <configuration .../>
        </agent>
        ...
    </agents>
</simulation>
```

Figure 3: Simulation XML structure

A skeleton XML for the `simulation` tag is shown in Fig. 3. It has two children: `configuration` and `agents`. The `configuration` part is scenario-specific, and must be in correspondence with the `configurationclass` specified in the `simulation` attributes. For the 2013 Mars scenario, the `configuration` attributes are the following:

- `maxNumberOfSteps` - The number of steps that the simulation must run until determining a winner.
- `numberOfAgents` - The total number of agents that take part in the simulation run.
- `numberOfTeams` - The number of teams that take part in the simulation run.
- `numberOfNodes` - The size of the randomly generated map, in terms of number of nodes (vertices).
- `gridWidth`, `gridHeight` - Affect the map-generation algorithm. Internally, nodes are created as being situated on a grid, and then edges are calculated according to this grid. `gridWidth*gridHeight` must be greater than `numberOfNodes`.
- `cellWidth` - This parameter is not used from *MASSim* itself, but is given to the monitor to facilitate the visualization. It stands for the distance (measured in `pts`) between two adjacent points in the grid.
- `minNodeWeight`, `maxNodeWeight` - The minimum and maximum possible value for the weights of the nodes (randomly assigned).
- `minEdgeCost`, `maxEdgeCost` - The minimum and maximum possible value for the costs of the edges (randomly assigned).
- `nodeWeighting` - The nodes' weight is computed out of a random and a gradient component. This parameter is the percentage of the random one (e.g. 100 would be a fully random assignment, whereas 0 would lead to a fully gradient generation).
- `randomSeed` - A long integer used as seed for the random map generation. If none is specified, the current system-time in milliseconds will be used instead. If you want to debug your multi-agent system, you can fix this seed to a constant to always use the same map.

  New in 2013: Very useful for debugging.
- `mapGenerator` - The type of generator that is to be used for building the map. Currently availabe are `GraphGeneratorTriangulation` as well as `GraphGeneratorTriangulationBalanced`, while the latter will generate symmetrical maps. Additionally, we added `GraphGeneratorTriangBalOpt` that generates a symmetric map with more than one so called center.

  New in 2013: Added.
- `randomWeight` - The weighting factor of random node weight generation.

  New in 2013: Added.

- `gradientWeight` - The weighting factor of gradient node weight generation.

New in 2013: Added.

- `optimaWeight` - The weighting factor of optima node weight generation.

New in 2013: Added.

- `blurIterations` - How many iterations of blurring shall happen (only relevant if `optimaWeight` > 0)

New in 2013: Added.

- `optimaPercentage` - The probability with which a node might be a local optimum (only relevant if `optimaWeight` > 0)

New in 2013: Added.

2.2.1 Actions

The `actions` section is used to specify the costs that actions may imply for the agents attempting to execute them. There must be one `action` tag for each action. Thus, the `name` attribute must be one of the following: `recharge`, `goto`, `attack`, `parry`, `probe`, `survey`, `inspect`, `repair` or `buy`.

In the general case, the rest of the attributes to be specified here represent the costs of attempting to execute that action in different situations. An action can cost `energy`, `health`, and `achievement points` (money). The costs can vary depending on the success or failure of the action, and also on whether the agent is in a `normal` or `disabled`[3] state, so attributes for all the combinations can be specified.[4] The names of these attributes are:

- `energyCost`
- `healthCost`
- `pointsCost`
- `energyCostFailed`
- ...
- `energyCostDisabled`
- ...
- `energyCostFailedDisabled`

Two special cases are the actions `recharge` and `goto`. For the `recharge` action, the values represent the percentage of the maximum `energy` and `health` that gets recovered. "Failure" in this particular case means that the agent has been attacked, and thus the health and energy recovering rates can be specified to be different.

The energy cost of a successful `goto` action is actually determined by the cost of the traversed edge. Therefore, the `energyCost` and `energyCostDisabled`

[3] An agent is considered to be in `disabled` state when its current `health` is 0

[4] Not all combinations make sense, and some of them may be just ignored by the server. Nevertheless, they are provided for notation consistency

specified here are considered as factors, that are multiplied by the edge cost. The cost for the `Failed` cases, on the other hand, are constants, as with the rest of the actions.

A thing to note here is that some costs can be specified to be negative values, e.g. if an agent should recover some energy when it was not able to perform a particular action.

2.2.2 Roles

The `roles` section defines the different roles that agents participating in the simulation will assume. A `role` encompasses all the internal characteristics of the agent and the set of actions that the agent is allowed to perform, both when in normal state and when disabled. The following attributes should be specified for each role:

- `name` - The name by which this role is referenced.
- `maxEnergy` - The initial upper limit for the `energy` of the agent.
- `maxBuyEnergy` - The upper limit for `maxEnergy` that can be reached when attempting to perform the `buy` action with `param="battery"`.
- `rateBuyEnergy` - The amount by which `maxEnergy` is increased when successfully performing the `buy` action with `param="battery"`.
- `maxEnergyDisabled` - The initial upper limit for the `energy` of the agent when `disabled`.
- `rateBuyEnergyDisabled` - The amount by which `maxEnergyDisabled` is increased when successfully performing the `buy` action with `param="battery"`.
- `maxHealth` - The initial upper limit for the `health` of the agent.
- `maxBuyHealth` - The upper limit for `maxHealth` that can be reached when attempting to perform the `buy` action with `param="shield"`.
- `rateBuyHealth` - The amount by which `maxHealth` is increased when successfully performing the `buy` action with `param="shield"`.
- `strength` - The initial strength of the agent.
- `maxBuyStrength` - The upper limit for `strength` that can be reached when attempting to perform the `buy` action with `param="sabotageDevice"`.
- `rateBuyStrength` - The amount by which `strength` is increased when successfully performing the `buy` action (with `param="sabotageDevice"`).
- `visRange` - The initial visibility range of the agent.
- `maxBuyVisRange` - The upper limit for `visRange` that can be reached when attempting to perform the `buy` action with `param="sensor"`.

- `rateBuyVisRange` - The amount by which `visRange` is increased when successfully performing the `buy` action (with `param="sensor"`).

The `actions` and `actionsDisable` sections of the role definition expect a list of `action` tags with only one attribute: the `name` of an action. The actions listed in these sections are the only actions that will be enabled for agents having this role when in `normal` or `disabled` state respectively.

2.2.3 Achievements

The achievements that will yield `achievement points` for the teams are defined here. Each achievement has four attributes: a (preferably unique) `name`, a `class` stating the type of achievement, a `quantity` needed to reach the achievement, and the number of `points` that the achievement yields. Six different classes of achievements are implemented:

- `probedVertices` - The `quantity` means the number of different nodes that a team needs to probe.
- `surveyedEdges` - The `quantity` means the number of different edges that a team needs to survey.
- `inspectedAgents` - The `quantity` means the number of different opponent agents that a team needs to inspect.
- `successfulAttacks` - The `quantity` means the number of successful attacks that a team needs to perform.
- `successfulParries` - The `quantity` means the number of successful parries that a team needs to perform (only counted when the parrying agent is actually attacked by an opponent).
- `areaValue` - The `quantity` means the score of a zone that a team needs to build.

2.2.4 Agents

The `agents` part of the simulation configuration is where it is defined how server-side teams are to be composed during the simulation. Agents defined here will be matched with agents defined in the `accounts` section to be controlled externally by the participants. This matching of agents varies in function of the `tournamentmode` parameter explained in 2.1.

The attributes for the `agent` tag are:

- `team` - The server-side name of the team.
- `agentclass` - The name of main Java class implementing the agents. For the 2013 Mars scenario, the class to use is `massim.competition2013.GraphSimulationAgent`.

- `agentcreationclass` - The name of the Java class that will hold the configuration parsed from the `configuration` child tag. For the 2013 Mars scenario, the class to use is `massim.competition2013.GraphSimulationAgentParameter`.

The `configuration` child tag for the 2013 Mars scenario only has one attribute: `roleName`, which refers to the name of one of the previously defined roles.

2.3 Accounts Configuration

In the `accounts` section of the configuration file, one can configure the developers' team that will participate in the tournament, and with which credentials each developer-side agent will connect to *MASSim* to control its server-side counterpart.

The `actionclassmap` has one attribute `name` and defines all available action classes for the agent accounts. Each `actionclass` has a `class` attribute and an `id`. An `account` is structured as follows:

- `actionclassmap` - Refers to the actionclassmap name that is used for this account.
- `auxtimeout` - Additional timeout for messages. The purpose of this parameter is to give the agents some additional time to allow the server to process the message.
- `defaultactionclass` - Sets the default action class.
- `maxpacketlength` - Defines the maximal length of on message.
- `password` - The password for the agent.
- `team` - The team name for the agent.
- `timeout` - The timout for messages.
- `username` - The user name of the agent.

3 Statistics Generation

MASSim now includes a new observer that generates statistics for each simulation and delivers them as images. It can be activated in the file `config.dtd` following this example:

```
<!ATTLIST simulation
  ...
  statisticsobserver CDATA "massim.competition2013.GraphSimulationStatisticsObserver"
  statisticsobserverpath CDATA "statistics"
>
```

This will activate the statistics-observer which will plot all the output to the specified folder. Please have a look at the scenario description for details.