# Implementation

For implementation we have use the JADE framework with which we have built a multi-agent system: Context Model Administering Agent (CMMA), Context Interpreting Agent (CIA), GUI Agent and Reinforcement Learning Agent (RLA). The RLA parses a world file description containing the xml description of the available sensors and ads that information to the context ontology representation, while the uses a pooling mechanism to gather sensor data at specific time intervals.

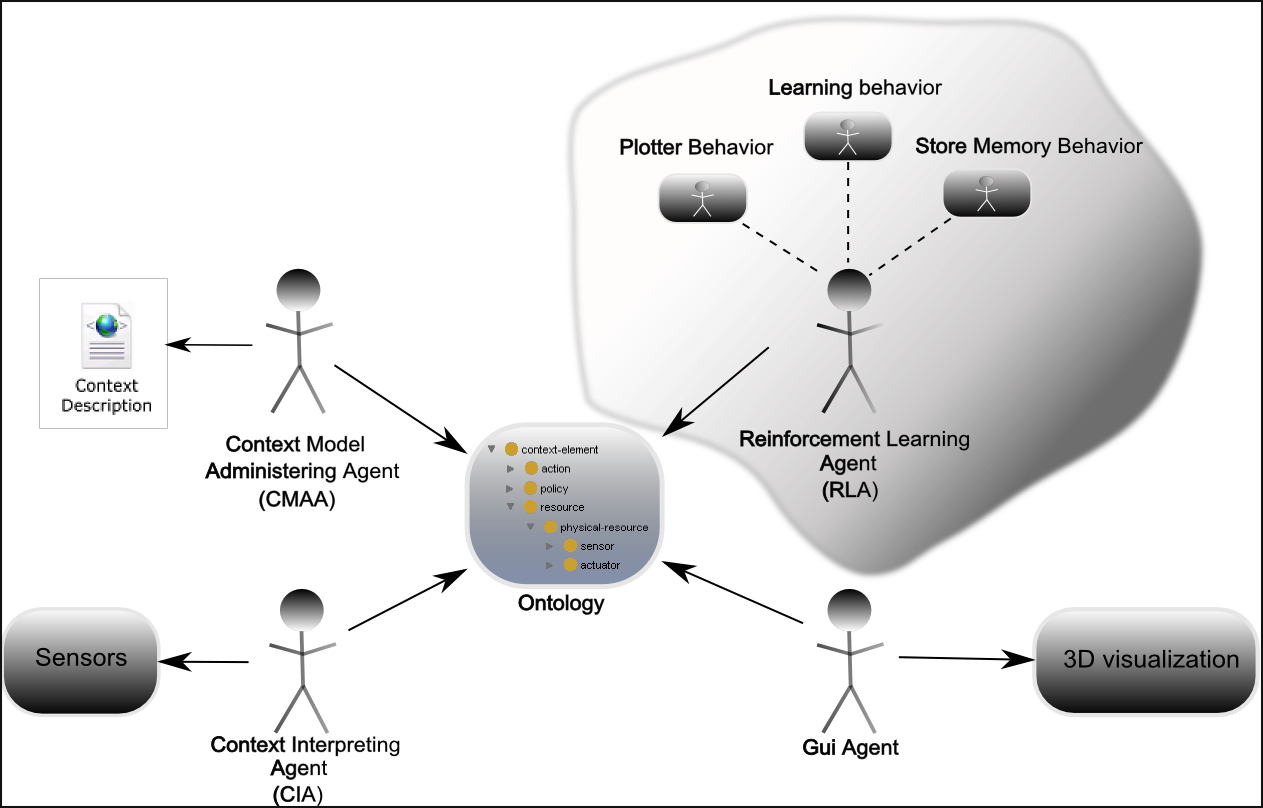


Figure 1 : Action Selection Framework Architecture

The actual action selection algorithm is implemented in the Learning Behavior attached to the RLA. The other two behaviors perform additional tasks like running time plot (Plotter Behavior) and file storage of what has been learned so far (Store Memory Behavior).

For test scenario we have chosen an environment having a computer, a camera with face recognition, a light source and an alarm. Each environment component has attached a sensor for monitoring its state. Also a humidity, temperature and room person count sensors have been added. There are three policies which our algorithm has to enforce: *light policy,* *face* *recognition policy* and *temperature and humidity* policy. The light policy specifies that the light should be on only if the room is not empty. The face recognition one specifies that if a professor is in the room the computer state must be on and if there is someone unknown in the room the alarm should go off. Last, the temperature and humidity policy enforces that the temperature should be between 18 and 23 degrees and the humidity between 20 and 30 %.

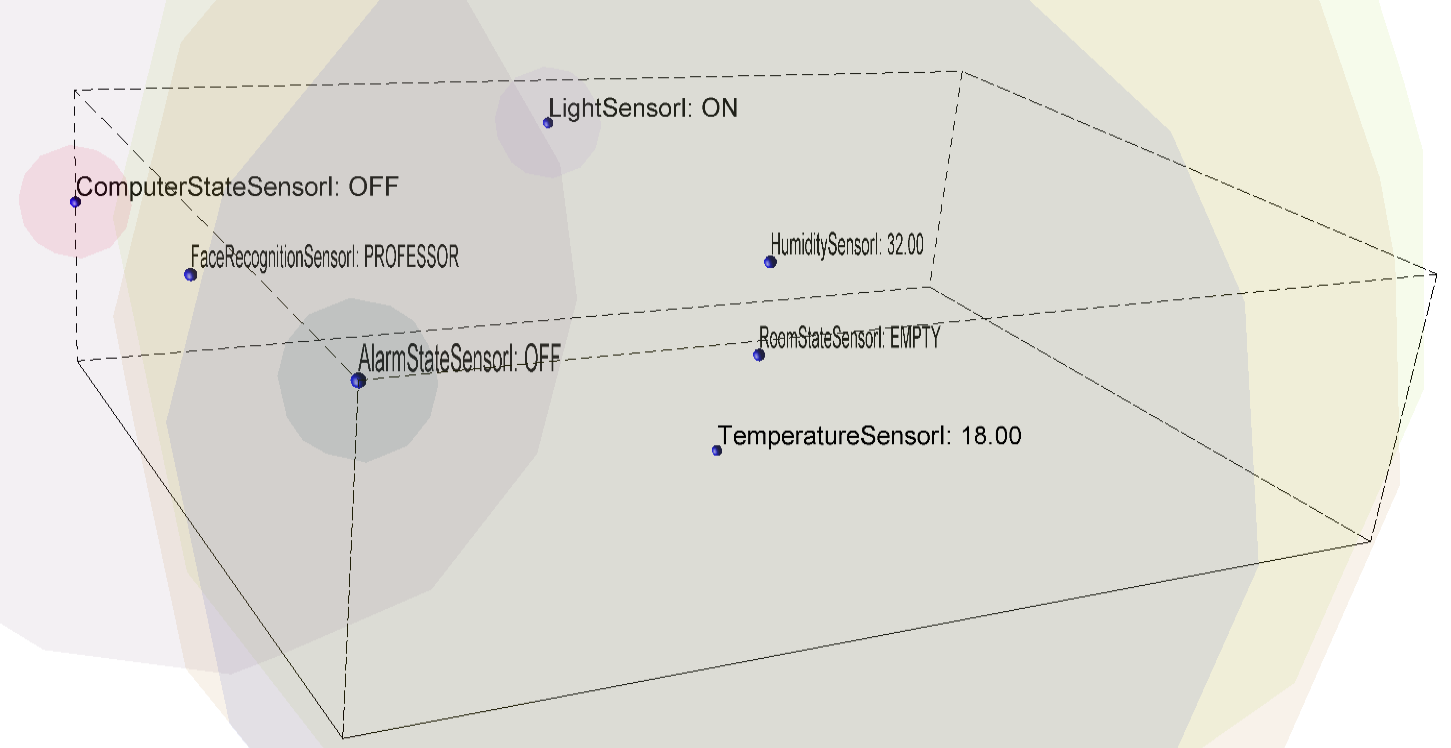


Figure 2: Test context

For better understanding we present a trace of the algorithm on the following case: a professor enters in the room, the computer and the light are off, breaking both the Alarm and Computer Policy and the Light Policy. The Reinforcement Learning Agent will see that the entropy is above the allowed threshold and call the reinforcement learning algorithm to fix the context. For our test scenario we consider the weights for policies and resources being equal to 1. The first inspected broken policy for the current context is the Alarm and Computer Policy. For each resource which doesn’t have acceptable value for the current policy, we try each action attached to the actuators of current resource and generate a new context. The first resource which is broken is the computer, on which we take an action which brings it to a state different from the current state, so the computer will be Turned On with the final state ON. The current context is added to the contexts queue, and reinforcement learning is called on current queue. After fixing the first policy, we continue with the Light Policy. For it we take the resource RoomNotEmpty, which has no attached actions and the resource Light for which we take the action which brings the resource to a state different than the current one: Turn On the light, with final state ON. After adding the new context to the contexts queue the algorithm is called again. When removing the context from the queue, the context entropy will be zero and the context will be returned. The sequence of actions attached to the current context will be executed, and the learned sequence of actions will be stored in the memory for further use.

In order to simulate a real context we have created a Context Disturbing Behavior (CDB) which assigns values to the context sensors with the purpose of breaking one or more policies and attached it to the RLA.

For the first test, a pattern of four broken contexts is used: the professor is in the room while the computer is off and light is off, the student is in the room while the alarm is off, the temperature and humidity are out of their admissible ranges and an unknown person is in the room and alarm is off. We suppose we have at most one person in the room.

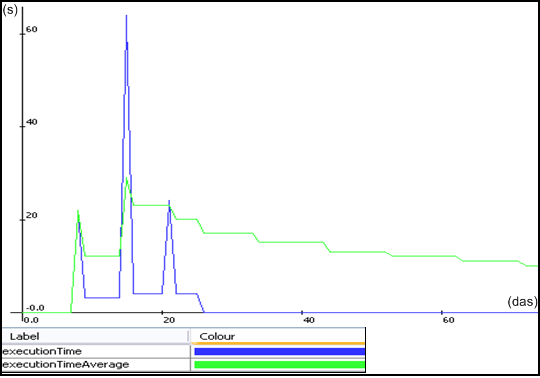


Figure 3: Context disturbed using a pattern

In the above graph we can see that the first four times that the reinforcement algorithm runs, we have running times of 20, 4, 61 and 22 seconds. When the disturbing pattern is repeated, we already have in the memory the broken context with its corresponding actions, and therefore the learning will take 0 seconds and the self-healing mechanism will apply the actions that it already learned.

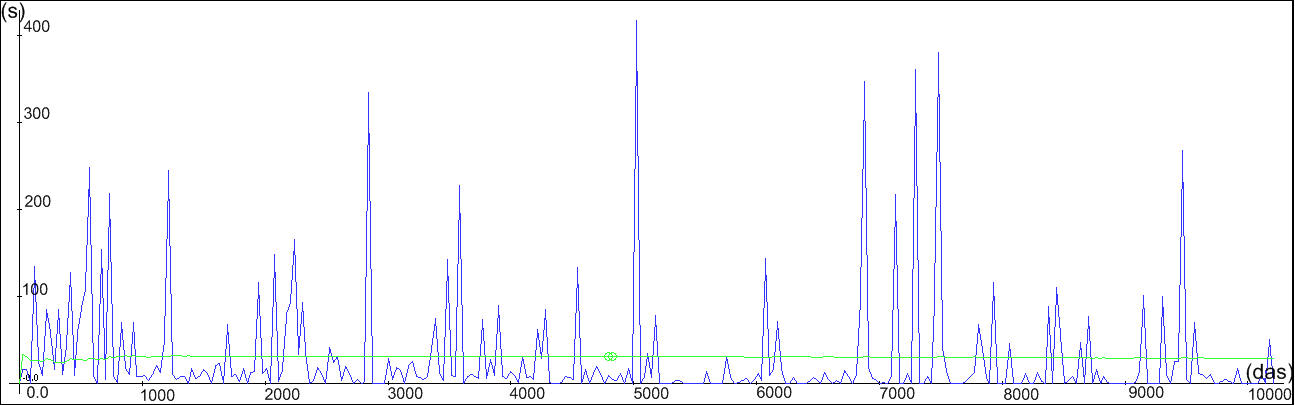


Figure 4: Context randomly disturbed

For the second test we left the program run for 28 hours. During this time the CDB sets random values to al sensors as following: for temperature from 15 to 25, for humidity from 15 to 35, 0 or 1(off, on) for light, room not empty, computer state and alarm sensors and 0, 1, 2 (professor, student, unknown) for the face recognition sensor. In the first 10000 seconds, almost all the running times of the reinforcement learning algorithm are larger than 10 seconds. After that, the self-healing mechanism begins to learn, achieving the performance of having only three running times greater than 50 seconds in the time interval [50000, 70000]. At each step of the reinforcement learning algorithm, it checks if it doesn’t already know the best sequence of actions for the context that it arrived in, and adds it to what it has discovered so far. Considering that the number of possible sensor combinations for random values of sensors is 22.481.940, the self-healing mechanism behaves quite well in rapidly finding and taking the needed actions for fixing the broken context.