A BDI agent architecture for the GAMA modeling and simulation platform

Patrick Taillandier, Mathieu Bourgais, Philippe Caillou, Carole Adam, Benoit Gaudou

▶ **To cite this version:**

Patrick Taillandier, Mathieu Bourgais, Philippe Caillou, Carole Adam, Benoit Gaudou. A BDI agent architecture for the GAMA modeling and simulation platform. MABS 2016 Multi-Agent-Based Simulation, May 2016, Singapore, Singapore. 10 p., 10.3233/978-1-61499-254-7-395 . hal-01391002

HAL Id: hal-01391002
https://inria.hal.science/hal-01391002v1

Submitted on 2 Nov 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A BDI agent architecture for the GAMA modeling and simulation platform

Patrick Taillandier[1], Mathieu Bourgais[1,2], Philippe Caillou[3], Carole Adam[4], and Benoit Gaudou[5]

[1] UMR 6266 IDEES, University of Rouen, Rouen, France
patrick.taillandier@gmail.com
[2] EA 4108 LITIS, INSA of Rouen, Rouen, France
mathieu.bourgais@insa-rouen.fr
[3] UMR 8623 LRI, University of Paris Sud, Paris, France
caillou@lri.fr
[4] UMR 5217 LIG, University of Grenoble, Grenoble, France
carole.adam@imag.fr
[5] UMR 5505 IRIT, University of Toulouse, Toulouse, France
benoit.gaudou@ut-capitole.fr

Abstract With the increase of computing power and the development of user-friendly multi-agent simulation frameworks, social simulations have become increasingly realistic. However, most agent architectures in these simulations use simple reactive models. Indeed, cognitive agent architectures face two main obstacles: their complexity for the field-expert modeler, and their computational cost. In this paper, we propose a new cognitive agent architecture based on the BDI (Belief-Desire-Intention) paradigm integrated into the GAMA modeling platform and its GAML modeling language. This architecture was designed to be simple-to-use for modelers, flexible enough to manage complex behaviors, and with low computational cost. An experiment carried out with different profiles of end-users shows that the architecture is actually usable even by modelers who have little knowledge in programming and in Artificial Intelligence.

Keywords: agent-based simulation, BDI architecture, GAMA platform

1 Introduction

Agent-based simulations are widely used to study complex systems. When the goal is to understand or even predict the behaviour of such complex systems, the simulation requires an accurate agent architecture to lay valid results. However, designing the agents is still an open issue, especially for models tackling social issues, where some of the agents represent human beings. In fact, designing complex agents able to act in a believable way is a difficult task, in particular when their behaviour is led by many conflicting needs and desires.

A classic paradigm to formalize the internal architecture of such complex agents in Agent-Oriented Software Engineering is the BDI (Belief-Desire-Intention)

paradigm. This paradigm, based on the philosophy of action [6], allows to design expressive and realistic agents, yet it is still rarely used in social simulations, for two main reasons. First, most agent architectures based on the BDI paradigm are too complex to be understood and used by non-computer-scientists. Second, they are often very time-consuming in terms of computation and therefore not adapted to simulations with thousands of agents.

In a previous paper [7], we proposed a first architecture to overcome these obstacles. This architecture is now fully integrated into the GAMA platform, an open-source modeling and simulation platform for building spatially explicit agent-based simulations [10,9,2]. GAMA's Modeling Language (GAML) and the integrated development environment support the definition of large scale models (up to millions of agents) and are usable even with low-level programming skills. Our BDI architecture was implemented as a new GAMA plug-in, and enables the straightforward definition of BDI agents through the GAML language. Thus, modelers have a wide range of ways to define their agents behaviors, by mixing classic GAMA primitives with BDI concepts. Indeed, we state that there is no unique way of defining the behavior of agents that can fit all the possible application contexts, types of agents to define (level of cognition), and modelers' backgrounds (programming skills, modeling habits...).

This paper describes some major improvements for the architecture, as well as a usability study carried out with modelers. The paper is structured as follows: Section 2 proposes a state of the art of BDI architectures and their use in simulations. Section 3 is dedicated to the presentation of our architecture, and more particularly the novelties from the previous work. In Section 4, we present an experiment in which we asked modelers with different profiles to test the architecture. Finally, Section 5 provides a conclusion and some perspectives.

2 State of the art

2.1 Agent architectures for social simulations

Balke and Gilbert [3] cite Sun as remarking that "social simulation researchers frequently only focus on agent models custom-tailored to the task at hand. He calls this situation unsatisfying and emphasises that it limits realism and the applicability of social simulation. He argues that to overcome these short-comings, it is necessary to include cognition as an integral part of an agent architecture." There are several options to endow agents with complex cognitive capabilities: cognitive architectures based on cognitive sciences, such as SOAR or ACT-R, or BDI architectures based on the philosophy of action.

2.2 BDI agents in AOSE frameworks

The BDI approach has been proposed in Artificial Intelligence [6] to represent the way agents can do complex reasoning. It has first been formalized using Modal Logic [8] in order to disambiguate the BDI concepts (Belief, Desire and Intention; detailed in Section 3.1) and the logical relationships between them.

In parallel, the Agent-Oriented Software Engineering (AOSE) field designed BDI operational architectures to help the development of Multi-Agent Systems embedding BDI agents. Some of these BDI architectures are included in frameworks allowing to directly use them in different applications. A classic framework is the Procedural Reasoning System (PRS) [13], that has been used as a base for many other frameworks (commercial, *e.g.* JACK [11], or open-source, *e.g.* JADE [5] with its add-on Jadex [14] offering an explicit representation of goals).

2.3 BDI agents in ABMS platforms

BDI architectures have been introduced in several agent-based modelling and simulation (ABMS) platforms. For example, Sakellariou et al.[16] have proposed an education-oriented extension to Netlogo [19] to allow modellers to manipulate BDI concepts in a simple language. This very simple architecture is inspired by PRS: agents have a set of beliefs (information obtained by perception or communication), a set of intentions (what they want to execute), and ways to manage these two sets.

Singh and Padgham [17] went one step further by proposing a framework acting like a middleware to connect components such as an ABMS platform and a BDI framework (*e.g.* JACK [11] or Jadex [14]). They demonstrated their framework on an application coupling the Matsim platform [4] and the GORITE BDI framework [15] for a bushfire simulation; but it aims at being generic and can be extended to couple any kind of ABMS platforms and BDI frameworks, only by implementing interfaces to plug each new component to the middleware. This approach is very powerful but remains computer-scientist-oriented, as it requires high programming skills to develop bridges for each component and to implement agents behaviours without a dedicated modelling language.

2.4 Previous work: BDI agents in GAMA

First attempts already exist to integrate BDI agents into the GAMA platform [10]. Taillandier et al. [18] proposed a BDI architecture where the choice of plans is formalized as a multi-criteria decision-making process: desires are represented as criteria, and decisions are made by evaluating each plan *w.r.t.* each criterion according to the agent's beliefs. However, this architecture was tightly linked to its application context (farmer's decision making), did not offer any formalism to model the agent's beliefs, and was rather limited *w.r.t.* how plans were carried out (*e.g.* no possibility to have complex plans with sub-objectives).

Le et al.[12] proposed another architecture dedicated to simulation with a formalized description of beliefs and plans and their execution. However, the desires and plans have to be written in a very specific and complex way that can be difficult to achieve for some application contexts, in particular for non-computer scientists. In addition, this architecture has a scalability problem: it does not allow to simulate thousands of agents.

Finally, we proposed in [7] a first simple architecture. This generic architecture proved its interest through an application to simulate agricultural parcels dynamics in Vietnam. However, the use of this architecture required to write many lines of code, in particular to manage the perceptions of agents and the inference of desires. Moreover, it did not allow to manage plans with different temporal scales. All these limitations made this architecture not usable for some applications and more complex to use by non-computer scientists. The next section therefore presents the improvements that we propose for this architecture.

3 Presentation of the Architecture

3.1 Overview

Our architecture is defined as a GAMA species architecture. The modeler is only required to define *simpleBDI* as the agent's architecture, and to define at least one plan and one initial desire to make it operational. After that the modeler mostly defines plans (what the agent can do), desires (what the agent wants) and (usually one) perception(s) (how the agent perceives its environment). Many keywords and functions are defined to help the user updating and managing Beliefs, Desires and Intentions bases, and creating and managing predicates.

The architecture and the vocabulary can be summarized with this simple Gold Miner example: the Miner agent has a general *desire* to find gold. As it is the only thing it wants at the beginning, it is its initial *intention* (what it is currently doing). To find gold, it wanders around (its *plan* is to wander). When it *perceives* some gold nuggets, it stores this information (it has a new *belief* about the existence and location of this gold nugget), and it adopts a new *desire* (it wants to extract the gold). When it perceives a gold nugget, the *intention* to find gold is put *on hold* and a new *intention* is selected (to extract gold). To achieve this *intention*, the *plan* has two steps, i.e. two new *(sub)intentions*: to choose a gold nugget to extract (among its known gold nuggets) and to go and take it. And so on.

This vocabulary is explained in more details in the next section.

3.2 Vocabulary

Knowledge. Beliefs, Desires and Intentions are described using **predicates**. A predicate has a name, may also have values (a set of name-value pairs), and can be true (by default) or false. Predicates can be easily defined through the GAML language. For example, in Figure 1, we define a new predicate *location_gold*, that represents the fact that a gold nugget is present at location (20,50). Note that when comparing two predicates, if one of them has no value for a specific key, the values corresponding to this key are not compared. For instance, to test if the predicate ("location_gold") is present in the belief base will return true if the predicate ("location_gold", ["location_value"::20,50]) is in the base.

BDI agents have 3 databases:

```
new_predicate("location_gold",["location_value"::{20,50}]);
```

Figure 1. predicate definition

- **belief_base** (what it knows): the internal knowledge the agent has about the world or about its internal state, updated during the simulation. A belief can concern any type of information (a quantity, a location, a boolean value, etc) and is described by a predicate. For example the predicate *gold_location* (Figure 1) is added when the agent perceives a gold nugget at position (20,50).
- **desire_base** (what it wants): objectives that the agent would like to accomplish, also updated during the simulation. Desires can have hierarchical links (**sub/super desires**) when a desire is created as an intermediary objective Desires have a dynamic **priority** value used to select a new intention among the desires when necessary.
- **intention_base** (what it is doing): what the agent has chosen to do. The **current intention** will determine the selected plan. Intentions can be put **on hold** (for example when they require a sub-intention to be achieved). For this reason, there is a stack of intentions: the top one is the current intention, all others are on hold.

Behavior In addition to the basic behavior structures available for all GAMA agents, our *simpleBDI* architecture has three available types of behavior structures (which are new compared to [7]):

- **Perception:** A perception is a function executed at each iteration to update the agent's Belief base, to know the changes in its environment (the world, the other agents and itself). The agent can perceive other agents up to a fixed distance or inside a specific geometry. For example, a miner can perceive all the gold nuggets at a distance of 10 meters and for each of them add a new belief concerning the gold's location (see Figure 2). Like for many other GAMA statements, several facets (most of them optional) can be used to tune the perceptions:
  - **target** (mandatory): the list of agents (or species of agents) to perceive.
  - **when** (optional): perception condition - can be a dynamic expression.
  - **in** (optional): can be either a radius (float) or a geometry (for instance, definition of a cone of perception or something more complex).
- **Rule:** A rule is a function executed at each iteration to infer new desires or beliefs from the agent's current beliefs and desires, i.e. a new desire or belief can emerge from the existing ones. For example, when the miner has a belief about some gold nuggets at a specific location, the rule creates the desire to extract gold (see Figure 3). Of course, if the new desire (or belief) is already in the base, nothing is added. Rules facets are the following:
  - **belief** (optional): required belief to activate the rule.
  - **desire** (optional): required desire to activate the rule.
  - **when** (optional): required condition to activate the rule.

  - **new_belief** (optional): the new belief to be added to the agent's base.
  - **new_desire** (optional): the new desire to be added to the agent's base.
- **Plan:** The agent has a set of **plans**, which are behaviors defined to accomplish specific intentions. Plans can be instantaneous and/or persistent, and may have a **priority** value (that can be dynamic), used to select a plan when several possible plans are available to accomplish the same intention. For example, when the miner has the intention to $find\_gold$, it can activate its $letsWander$ plan (see Figure 4) that makes it randomly move. Concerning the facets of plans:
  - **intention** (optional): intention the plan try to fulfill.
  - **when** (optional): activation condition.
  - **finished_when** (optional): termination condition.
  - **priority** (optional): priority of the plan, 1 by default.
  - **instantaneous** (optional): if $false$, no other plan can be executed afterwards during the current simulation step; otherwise, (at least) one other plan will be executed during the same step. By default $false$.

```
perceive target:gold in:10 {
   focus var:location;
}
```

Figure 2. perception definition: when the agent perceives a gold nugget at a distance lower than 10 meters, a new belief is added concerning the location of this gold nugget.

```
rule belief: new_predicate("location_gold") new_desire: get_gold ;
```

Figure 3. rule definition: when the agent has a belief about a gold nugget somewhere, a desire is added to extract gold

```
plan letsWander intention:find_gold {
   do wander;
}
```

Figure 4. plan definition: if the agent has the intention to $find\_gold$, it can use the $letsWander$ plan, which makes the agent moves randomly (uses of the $wander$ built-in action of GAMA)

3.3 Thinking process

At each step, the agent applies the process described in Figure 5. Roughly, the agent first perceives the environment, then i) **continues its current plan** if it is not finished, or ii) if the plan is finished and its current intention is not fulfilled, it **selects a plan**, or iii) if its current intention is fulfilled, it **selects a new desire** to add to its intention stack.

This process is similar to the one proposed in [7], however it introduces the perception and rule steps. In addition, it allows to apply several plans in the same time step in case of instantaneous plans. More precisely:

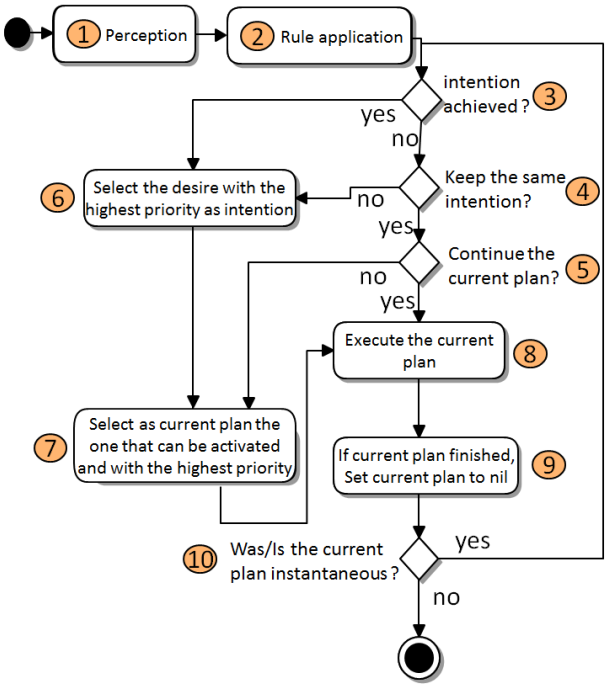

Figure 5. Activity diagram

1 - Perceive: Perceptions are executed.

2 - Rule: Rules are executed.

3 - Is one of my intentions achieved?: If one of my intentions is achieved, sets the current plan to nil and removes the intention and all its sub-desires from the desire and intention bases. If the achieved intention's super-desire is on hold, it is reactivated (its sub-desire just got completed).

4 - Do I keep the current intention?: To take into account the environment instability, an intention-persistence coefficient *ip* is applied: with the

probability *ip*, the current intention is removed from the intention stack. More details about this coefficient are given in section 3.4.

5 - Do I have a current plan?: If I have a current plan, just execute it. As for the current intention stability, the aim is both persistence (I stick to the plan I have chosen) and efficiency (I don't choose at each step). Similarly to intentions, a plan-persistence coefficient *pp* is defined: with a probability *pp*, the current plan is just dropped.

6 - Choose a desire as new current intention: If the current intention is on hold (or the intention base is empty), choose a desire as new current intention. The new selected intention is the desire with higher priority among those not already present in the intention base.

7 - Choose a plan as a new current plan: The new current plan is selected among the plans compatible with the current intention (and if their activation condition is checked) and with the highest priority.

8 - Execute the plan: The current plan is executed.

9 - Is my plan finished?: To allow persistent plans, a plan may have a termination condition. If it is not reached, the same plan will be kept for the next iteration.

10 - Was my plan instantaneous?: Most multi-agent based simulation frameworks (GAMA included) are synchronous frameworks using steps. One consequence is that it may be useful to apply several plans during one single step. For example, if a step represents a day or a year, it would be unrealistic for an agent to spend one step to apply a plan like "choose a destination". This kind of plans (mostly reasoning plans) can be defined as **instantaneous**: in this case a new thinking loop is applied during the same agent step.

3.4 Properties

Persistence and priority In our architecture, the persistence of plans and desires, as well as their priority can be dynamically defined (i.e. modified by the agents during the simulation and/or computed from a function).

Flexibility The architecture aims at being simple-to-use and as flexible as possible for the modeler. More advanced modelers can use its full potential (for example dynamic coefficients as presented before), while others can rely on the default values of most parameters to easily specify agents using only some of the available features. In addition, in order to manipulate the different elements of the architecture, we provide modelers with built-in actions directly usable in their models, among which (desire related actions are not listed here):

- **add_belief**: add a new belief to the belief base.
- **remove_belief**: remove a belief from the belief base.
- **has_belief**: test if a belief is in the belief base.
- **replace_belief**: replace a belief by a new one in the belief base.
- **get_belief**: return the first belief corresponding to a given predicate.
- **get_beliefs**: return all beliefs corresponding to a given predicate.

- **remove_all_beliefs**: remove all beliefs corresponding to a given predicate.
- **clear_beliefs**: remove all beliefs.
- **get_current_intention**: return the current intention.
- **clear_intentions**: remove all intentions.
- **add_subintentions**: add a sub-intentions to a predicate
- **current_intention_on_hold**: put the current intention on hold and add as its activation condition all its sub-intentions.

4 Experiments

4.1 Modelers feedback analysis

Context of the experiment In order to validate our claim that our architecture is simple to use and is adapted to several types of users, we carried out an experiment with six modelers with different backgrounds: 3 computer scientists, and 3 geographers, all of whom knew GAMA (at least the basic concepts). Two of them have a low level in programming (2 geographers) and four a higher level (1 geographer and 3 computer scientists). All of them have at least a low (2) or medium (3) level in GAMA (they know at least the basic concepts), the last one being an expert. Only the 3 computer scientists know the BDI paradigm.

This experiment consisted in a short lesson (45 minutes) about the *simpleBDI* architecture, followed by an exercise which required to use this architecture (2 hours). At the end of the exercise, each participant answered a short survey to assess the architecture (with both open questions and closed scaled assessments). All these documents (introduction course, exercise, models developed by the different modelers, and their answers to the questionnaire) are available on the project website [1].

The presentation of the *simpleBDI* architecture used a simple gold miner model to present the underlying concepts. Then the exercise theme was the evacuation of the city of Rouen (France) (see Figure 6). A technological hazard is simulated in one of the buildings of the city center. Drivers can perceive the hazard at a distance of 400 meters. Those who know that there is a hazard try to reach one of the evacuation sites (shelters). A driver who sees (in a radius of 10 meters) another driver trying to evacuate has a small chance to understand that a hazard is happening.

The participants were given a first basic model of that situation, containing four species of agents: road, hazard, evacuation site and driver. The behavior of the driver agents is defined by a single reflex executed at each simulation step: the agent moves towards a random target (any point on the road network), and if it reaches its destination, it chooses a new random target. A weighted graph is used for the movement of drivers: they first compute the shortest path between their location and their target, then use this path to move. The weights of the edges of the graph (roads) are updated every 10 simulation steps to take into account the number of drivers on each road. The agent speed on each road is a function of the number of drivers on this road and its maximum capacity. If a driver already has a computed path, it will not recompute it even if the weights of that path

change. At the initialization of the model, the roads (154), evacuation sites (7) and the hazard (1) agents are created and initialized using shapefiles; then 500 driver agents are created and randomly placed on the roads. The GAML code for the driver species is given Figure 7. In this first basic model, drivers do **not** perceive the hazard and do **not** try to reach evacuation sites; they just keep moving randomly.

The exercise was composed of two steps of increasing difficulty:

- **Step 1**: modification of the behaviors of the driver agents, in order to:
  - make them aware of the hazard: via its direct perception (with a probability of 0.2 if they are in a radius of 400m) or indirectly when they see other driver agents trying to reach an evacuation site (with a probability of 0.01 in a radius of 10m);
  - make them try to reach the closest shelter (euclidean distance) when they know that there is a hazard.
- **Step 2**: modification of the behaviors of the driver agents and of the evacuation site, so that:
  - if a driver agent trying to reach a shelter thinks that its road is blocked (speed coefficient lower than 0.5), then it should test a new path (recomputation of the path according to the current weights of the graph);
  - drivers take into account the maximum capacity of the evacuation sites (50 drivers each). To know that an evacuation site is full, drivers have to be less than 20 meters away from it.

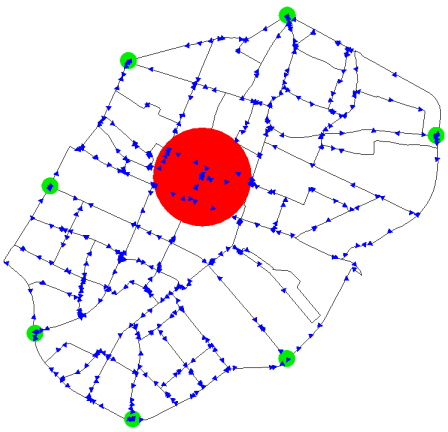

Figure 6. Snapshot of the city evacuation model: the red circle is the hazard perception area, the green circles the evacuation sites and the blue triangles the drivers.

```
species driver skills: [moving]{
    point target;
    float speed <- 30 #km/#h;
    rgb color <- #blue;

    reflex move {
        if (target = nil) {
            target <- any_location_in(one_of(road));
        }
        do goto target: target on: road_network
            move_weights: current_weights recompute_path: false;
        if (location = target) {
            target <- nil;
        }
    }

    aspect default {
        draw triangle(10) rotate: heading + 90 color: color;
    }
}
```

if I do not have any target, I choose as target a random point in one of the road

I move toward my target on the road graph

If I arrived at destination, I set my target to nil

Figure 7. Initial GAML code for the driver species using reflex architecture, provided to the modelers.

Possible solution of the exercise We present here a possible solution. We first add the *simpleBDI* architecture to the driver species and 3 new variables (Figure 8):

- **escape_mode**: boolean to indicate if the agent is evacuating or not
- **recompute_path**: boolean to indicate if the agent has to recompute its path or not.
- **current_shelter**: the current evacuation site to reach.

as well as 4 predicates that will be used as the content of mental attitudes:

- **at_target**: the agent is at its target used as the desire to go to a specific target.
- **in_shelter**: the agent is in an evacuation site used as the desire to go to an evacuation site.
- **has_target**: the agent has a defined target used as the desire to define a target.
- **has_shelter**: the agent has selected an evacuation site used as the desire to choose an evacuation site.

We define 4 perceptions (Figure 9):

- **hazard**: perceives the hazard in a radius of 400m with a probability of 0.2.
- **driver**: perceives the drivers in escape mode in a radius of 10m with a probability of 0.01.
- **road**: perceives the blocked roads in a radius of 1.
- **shelter**: perceives the full evacuation sites in a radius of 20m.

We define two rules to infer the desire to go toward an evacuation site (if the agent perceived the hazard directly or through the observation of escaping drivers) and an action to switch to escape mode (Figure 10):

Finally, we define four plans for the agents (Figures 11 and 12):

```
species driver skills: [moving] control: simple_bdi{
    point target;
    float speed <- 30 #km/#h;
    rgb color <- #blue;
    bool escape_mode <- false;
    predicate at_target <- new_predicate("at_target") with_priority 1;
    predicate in_shelter <- new_predicate("shelter") with_priority 5;
    predicate has_target <- new_predicate("has target") with_priority 2;
    predicate has_shelter <- new_predicate("has shelter") with_priority 10;
    bool recompute_path <- false;
    shelter current_shelter;
}
```

If true, it means that I am trying to reach an evacuation site

If true, it means that I will try to recompute my path

Current evacuation site that I want to reach

Figure 8. Possible solution GAML code for the driver species and variables

- **normal_move**: the agent chooses a target if it has none and moves towards it.
- **choose_normal_target**: the agent chooses a random target
- **evacuation**: the agent chooses an evacuation site if it has none and moves towards it.
- **choose_shelter**: the agent chooses an evacuation site.

Results of the experiment After the exercise, each participant answered a short survey about the BDI architecture. The survey was composed of 7 closed questions using a 1-5 scale, 2 yes/no questions and 9 open questions/commentary sections. The first three questions were used to assess the participant background and the others to assess the *simpleBDI* architecture and the exercise performance. Due to the low number of participants, the survey results are used as a qualitative evaluation, and not as a statistically significant quantitative assessment.

A first analysis of the results shows that all the participants find the proposed architecture clear (answer values of 4 or more for that question). Furthermore, the three participants that have a background in BDI architectures find that our architecture translates the BDI paradigm well (3+ answers). Concerning the simplicity of use of the architecture inside GAMA, three of the participants find it good (4) or very good (5), and two pretty good (3).

Five out of the six participants succeeded in implementing Step 1, but none of them achieved Step 2. A tentative explanation is that two hours was a too short time to understand what was asked in the exercise, formalize the behavior of the agents using the *simpleBDI* architecture, write the GAML code, and test it. In addition, only one of the participants knew GAMA very well, so as mentioned in the survey, most of the others lost a lot of time searching for specific GAMA operators (not linked with the *simpleBDI* architecture). Nevertheless, four of the participants were very close to succeed in the second step of the exercise (the three computer scientists and one of the geographers).

Concerning the comparison to other architectures, one of the modelers (the BDI expert) preferred *simpleBDI* to the others, while two of them found *simpleBDI* to be complementary to the existing ones, and mentioned that it allows

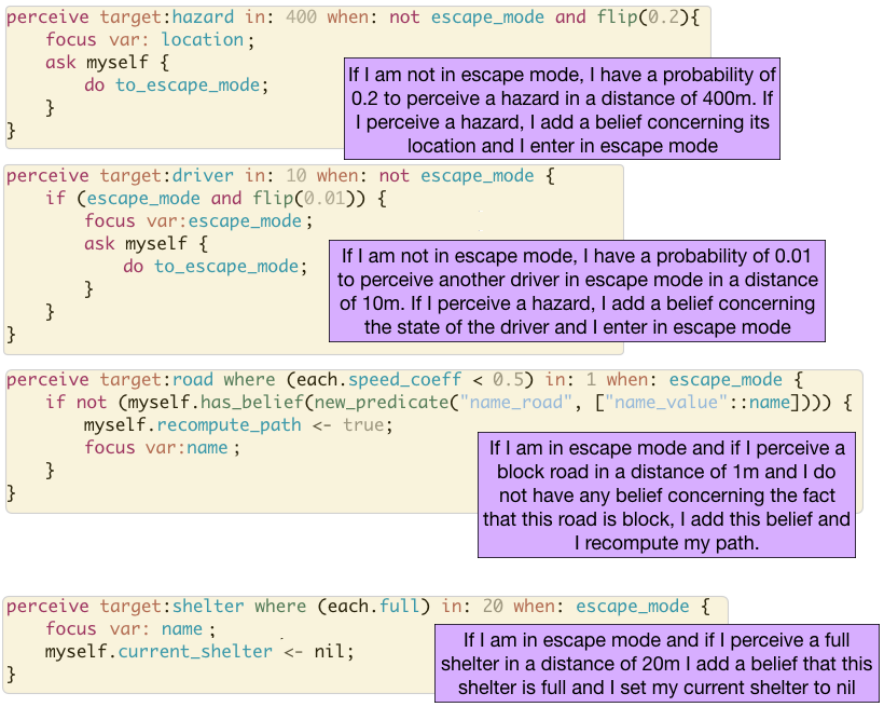

Figure 9. Possible solution GAML code for the driver perceptions

to define the behavior in a simpler way, avoiding to write many complex reflexes. Only one modeler mentioned that they preferred the reflex architecture as they were more used to it. The last two participants did not answer this question.

An interesting remark is that some of the participants mixed the *simpleBDI* and *reflex* architectures, using the BDI architecture to define perceptions and objectives (especially the agents target), and reflexes for the repetitive operational behaviors (moving).

To conclude, this first experiment showed very promising results: all the participants found the *simpleBDI* architecture clear and easy to use. After a short 45 minutes lesson, they were able to apply it to a real model previously unknown to them. To achieve a more complete and efficient use of the *simpleBDI* architecture, they would however have needed more time, better programming skills in GAMA, and/or a better knowledge of the BDI paradigm.

4.2 Architecture scalability

In order to test the architecture scalability, we used the two previous models (gold miners and city evacuation) with an increasing numbers of agents.

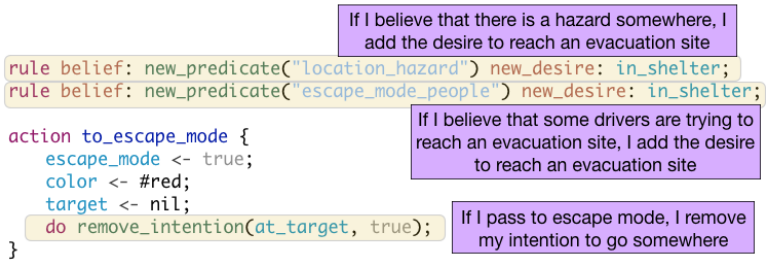

Figure 10. Possible solution GAML code for the driver rules and action

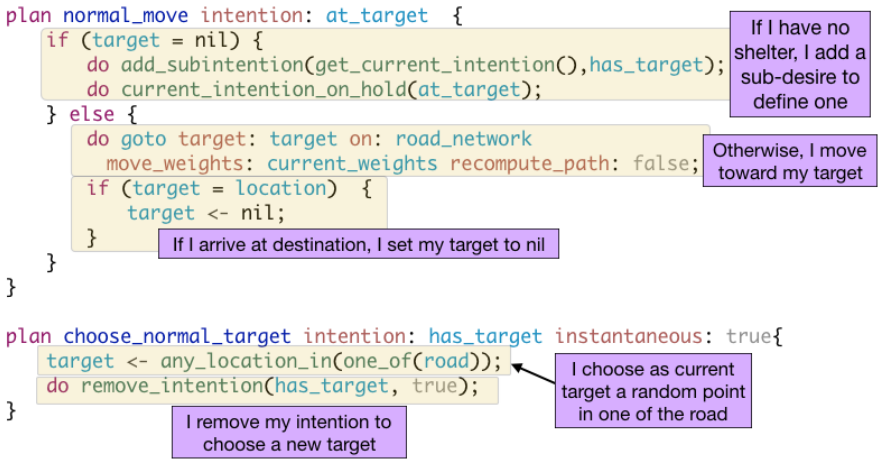

Figure 11. Possible solution GAML code for the *normal_move* and *choose_normal_target* driver plans

As GAMA is often used by modelers with old computers (for social simulation in developing countries), we chose to carry out the experiment with a 5-year old Macbook pro (2011) with an i7 processor and 4Go of RAM.

The gold miner model was tested with 10 000 miners, 1000 golds and a square environment of 10 x 10 kilometers. The simulation was stopped when all the gold nuggets had been returned to the base. The average duration of a simulation step (without any graphical display) was 140ms.

The evacuation model was tested with 1000 drivers (due to the road network used and how the capacity of roads was defined, it was not possible to test the model with more driver agents - all the road would have been blocked) and a capacity for each of the evacuation sites of 200 driver agents. We stopped the simulation when all the drivers reached an evacuation site. The average duration of a simulation step (with no graphical display) was 70ms.

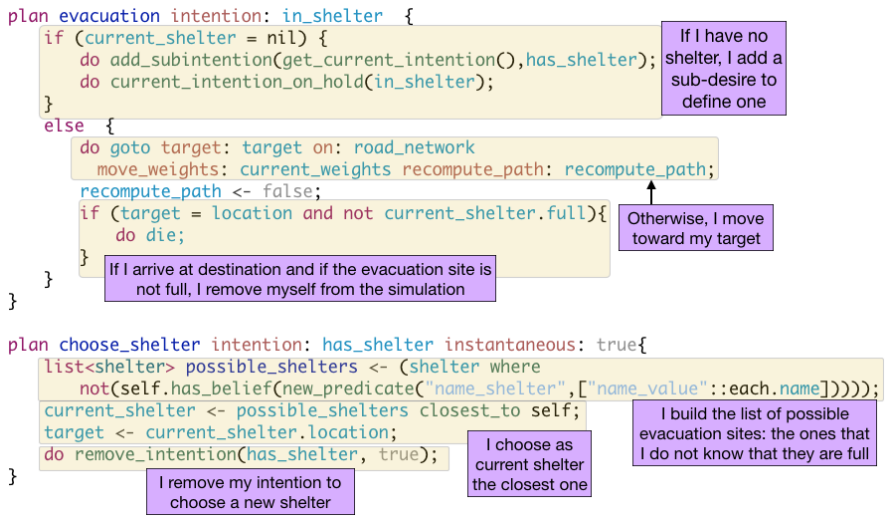

Figure 12. Possible solution GAML code for the *evacuation* and *choose_shelter* driver plans

The results obtained in terms of performance show that the architecture can already be used with medium-scale real world problems. However, the architecture will still be continuously optimized and we plan to compare the results with other GAMA architectures (especially the reflex one). We also plan in the future to test the architecture with more complex agents having many possible desires, sub-desires and plans.

5 Conclusion

In this paper, we have presented a new BDI architecture dedicated to simulations. This architecture is integrated into the GAMA modeling and simulation platform and directly usable through the GAML language, making it easily usable even by non-computer scientists. We have presented a first experiment that was carried out with modelers with different profiles (geographers and computer scientists). This first experiment showed that our plug-in can be used even by modelers that have little knowledge in programming and Artificial Intelligence, and that it allows to simulate several thousands of agents.

If our architecture is already usable, some improvements are planned. First, we want to improve the inference capabilities of our architecture: when a new belief is added to the belief base, desire and intention bases should be updated in a efficient way as well. Second, we want to make it even more modular by adding more possibilities concerning the choice of plans and desires (beyond just choosing that with the highest priority): user-defined selection, multi-criteria decision process, etc. Finally, we want to add the possibility to use high performance computing (distribute the computation on a grid or cluster) to decrease computation time.

6 Acknowledgement

This work is part of the ACTEUR (Spatial Cognitive Agents for Urban Dynamics and Risk Studies) research project funded by the French Research Agency (ANR).

References

1. ACTEUR website: http://www.acteur-anr.fr (2015), http://www.acteur-anr.fr
2. GAMA website: http://gama-platform.org (2015), http://gama-platform.org
3. Balke, T., Gilbert, N.: How do agents make decisions? a survey. Journal of Artificial Societies and Social Simulation 17(4) (31 October 2014)
4. Balmer, M., Rieser, M., Meister, K., Charypar, D., Lefebvre, N., Nagel, K., Axhausen, K.: Matsim-t: Architecture and simulation times. Multi-Agent Systems for Traffic and Transportation Engineering pp. 57–78 (2009)
5. Bellifemine, F., Poggi, A., Rimassa, G.: JADE–A FIPA-compliant agent framework. In: Proceedings of PAAM. vol. 99, p. 33. London (1999)
6. Bratman, M.: Intentions, plans, and practical reason. Harvard Univ. Press, Cambridge (1987)
7. Caillou, P., Gaudou, B., Grignard, A., Truong, C.Q., Taillandier, P.: A Simple-to-use BDI architecture for Agent-based Modeling and Simulation. In: ESSA (2015)
8. Cohen, P.R., Levesque, H.J.: Intention is choice with commitment. Artificial Intelligence 42, 213–261 (1990)
9. Drogoul, A., Amouroux, E., Caillou, P., Gaudou, B., Grignard, A., Marilleau, N., Taillandier, P., Vavasseur, M., Vo, D.A., Zucker, J.D.: Gama: multi-level and complex environment for agent-based models and simulations. In: AAMAS. pp. 1361–1362 (2013)
10. Grignard, A., Taillandier, P., Gaudou, B., Vo, D., Huynh, N., Drogoul, A.: GAMA 1.6: Advancing the art of complex agent-based modeling and simulation. In: PRIMA 2013. LNCS, vol. 8291, pp. 117–131. Springer (2013)
11. Howden, N., Rönnquist, R., Hodgson, A., Lucas, A.: JACK intelligent agents-summary of an agent infrastructure. In: 5th AA (2001)
12. Le, V.M., Gaudou, B., Taillandier, P., Vo, D.A.: A new BDI architecture to formalize cognitive agent behaviors into simulations. In: KES-AMSTA. pp. 395–403 (2013)
13. Myers, K.L.: User guide for the procedural reasoning system. SRI International AI Center Technical Report. SRI International, Menlo Park, CA (1997)
14. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A BDI reasoning engine. In: Multi-agent programming, pp. 149–174. Springer (2005)
15. Rönnquist, R.: The goal oriented teams (gorite) framework. In: Programming Multi-Agent Systems, pp. 27–41. Springer (2008)
16. Sakellariou, I., Kefalas, P., Stamatopoulou, I.: Enhancing NetLogo to simulate BDI communicating agents. In: Artificial Intelligence: Theories, Models and Applications. pp. 263–275. Springer (2008)
17. Singh, D., Padgham, L.: OpenSim: A framework for integrating agent-based models and simulation components. In: Frontiers in Artificial Intelligence and Applications-Volume 263: ECAI 2014. pp. 837–842. IOS Press (2014)
18. Taillandier, P., Therond, O., Gaudou, B.: A new BDI agent architecture based on the belief theory. application to the modelling of cropping plan decision-making. In: iEMSs (2012)
19. Wilensky, U., Evanston, I.: Netlogo. center for connected learning and computer based modeling. Tech. rep., Northwestern University (1999)