In Fourth International ICSC Symposium on Engineering of Intelligent Systems (EIS 2004)

Modeling natural motivations into hybrid artificial agents

F.Andriamasinoro

IREMIA, University of La Reunion
15, Avenue René Cassin
97715 Saint-Denis
La Réunion-FRANCE
E-Mail: fenintsoa.andriamasinoro@univ-reunion.fr

Abstract

Beforehand, the concept of natural motivations (i.e.motivations related to the satisfaction of natural needs) has been generally integrated into reactive agents, and particularly to animats. In this paper, we present and discuss a generic model which introduces such notions into hybrid agents. The basis of our model is the Abraham Maslow's pyramid of needs.

Keywords: artificial agent, natural motivations, hybridism, behavior

1 Introduction

In hybrid context, the interplay between reactivity and deliberation is one of the issues of paramount importance for intelligent[1] agent design. (Malec, 2000). But even if this interplay is important, it mainly deals with the *manner* the goal should be achieved. Only few things are told about the notion of motivations (the *reason*), which are nevertheless the basic source of this goal. When we here say "motivations", it is about the particular case of "natural motivations" i.e. motivations which are related to the satisfaction of natural needs (hunger, sexual impulse, etc.) coming from instinct or homeostasis, and mainly found in artificial agents[2].

In fact, some works already started to consider natural motivations in hybrid context. We can mention obstacle or fire avoidance behavior which are actually related to the natural preserving instinct (Au and al. 98, Mavromichalis and Vouros, 2000), or the running away from enemy (Tambe, 1996). But these works are taken case by case, according to the study. There is no real generic specifications for natural motivations in hybrid models unlike in animat ones (Guillot and Meyer, 1998), in which natural motivations completely determine the behavior of agents (Gershenson et al. 2000). However, as both humans and animals are firstly natural creatures and their behavior essentially starts from motivations (Andriamasinoro and Courdier, 2002), all artificial agents (either reactive or cognitive ones) should also permanently integrate them. The psychologist Daco (1965) who wanted to stress the importance of instinct in living being stipulated that a person (i.e. containing cognitive concepts) who lives with uniquely its reason is only a semi-person.

The aim of this paper is then to report on progress towards our effort to introduce this theory of basic motivations in hybrid agent paradigm, and at a generic level. Key issue towards this aim is the fact that we have to cope with the motivation selection problem while taking into account that the model should work for both reactive and cognitive agents. Precisely, this issue is about finding a set of generic criteria which determine the most important motivations an agent has to dynamically take into account its behavior. As a result, this generic approach will reduce the task of the user when using the system. Note that in this work, we cope with only individual agents. The social interaction is not treated here yet.

To make the paper objective clearer, we organize it as follows: Section 2 presents the model we propose in this work and Section 3 describes our experimentation. Section 4 next discusses the work before we conclude the paper in Section 5.

2 Description of the agent model

Our agent model is named MASLOW. It is based on three components: a pyramid of needs / motivations[3] (noted Π), a network of actions (noted Ω) and a component

[1] The term "intelligence" in this paper is defined, for an entity, as the ability to use all information it has to achieve its goal (Newell, 1982).

[2] This work is focused only on Artificial Life domain. Then, all notions we develop here follow this hypothesis.

[3] Here, *needs* and *motivations* may be confusing. Actually, since the basic motivation of an agent is the satisfaction of its basic needs, these two terms can be alternatively used.

called NIM (for Need Importance Manager) which "pilots" Π and Ω and particularly manages the generic agent behavior. The difference between these components is that users can act on Π and Ω while it cannot do so on the NIM. As we will see, the latter only concerns the agent proactivity.

The following sections successively explain these components. Note that Ω, as well as the interaction of components to each others are first reported in this paper.

2.1 The pyramid P

This component is inspired from the pyramid of needs designed by the American psychologist Abraham Maslow (Maslow, 1954) which stipulates that *all actions led by the living being's behavior are motivated by at least one of the following five hierarchical needs mentioned from the bottom (the most important) to the top (the less important) of the pyramid*: the physiological needs, the need for security, the need for love, the need for esteem and the need for self-realization.

Based on this pyramid, our approach in this work is the following: we consider first that each level of Π is abstract. Then, for each level, we adopt two *types* of needs:

- the Low-Need (LN) which groups all natural needs. They are *common* and *permanent* and exist in all agents, independently of the application. It is handled either by instinct or homeostasis. In other words, the concept of LN corresponds to generic basic motivations.
- the High-Need (HN) which includes all *individual* and *temporary* needs which depend on the application. A HN may be for instance assimilated to the notion of *desire* found in many BDI models. What is important is that the satisfaction of a HN is *always* motivated by at least one LN.

Both types are conceptually grouped in a common abstract type named PN (for Pyramidal Need). Thus, Π={PN}={LN}∪{HN}. We only remind here the formalization of a PN, previously detailed in Andriamasinoro and Courdier (2002):

`PN ={lib, level, rank, state_lib, list_actions}` in which:

- `lib` is the unique identifier of PN
- `level/rank` correspond to the "physical" position of PN in Π: `level` is that of Abraham Maslow while `rank` differs the PN situated at a same level. Actually, a HN motivated by a LN means that this HN has the same `level` and `rank` than the LN. In the formalization of a HN, the two parameters are directly replaced by the associated LN as follow: `HN={lib, LN, state_lib, list_actions}`.
- `state_lib` formalizes the list of states in which PN may be. It may take one of the following values: `insufficient, limit_low, sufficient, limit_high, excessive`.
- `list_actions`={action[insufficient], …, action[excessive]} contains the set of actions corresponding to each state (except `sufficient` which has no action). The current state of PN is named `currentstate`. Note however that all states are not always represented in a PN. It depends on the semantic of the PN at application level.

A state can be presented as a set of *intervals* (SI) or *points* (SP). A specific case of the latter is the representation known as *boolean* (SB) where the associated PN can be only in two states: `insufficient` (= `false`) and `sufficient` (= `true`). In a general way, `state_lib` can be described as:
`state_lib={representation_lib, description}` in which, for each presentation, `description` is written as follow:

- `SI`: <interval (`state`$_1$), …, interval (`state`$_n$)>
- `SP`: <val (`state`$_1$), …, val (`state`$_k$)>
- `SB`: <proposition> which is a proposition returning `true` or `false`.

The `representation_lib` parameter may take the following value: "`interval`," `point` or "`Boolean`".

2.2 The network W of actions

Preamble: the concept of actions

We have two kinds of actions:

- a *primitive* (PR): it corresponds to the fine-grained action. It is uninterruptible during its execution.
- a *composed* action (AC): it is a combination either of PR or of other sub-AC.

The set of actions is noted Γ. In sum, Γ={PR}∪{AC}.

Each action `act` ∈ Γ is formalized as follow:
`act={pn_satisf, precond, list_conflicts}` in which :

- `pn_satisf` is the need to be satisfied via `act`, i.e. `pn.action[state]=act` ⇔ `pn=act.pn_satisf`,
- `precond` is a PN that must be satisfied before `act` can be executed. If `precond` is not set, it means that the action is always executable. The difference between `precond` and `pn_satisf` is that the former is not manipulated by the NIM during the motivation selection process (Section 2.3) while the latter does.
- `list_conflicts` contains the list of actions that the agent cannot simultaneously execute with `act`.

During the initialization of an action, last parameters which are non-initialized may be omitted.

The network W

Ω is a network formalized as Ω=(Δ, C) in which

- Δ are the nodes, constructed of *actions* (Δ ⊆ Γ),
- C are the arcs, composed by a set of 4 *connectors*: `then`, `imp`, `xor` and `and`. Precisely, C={then}∪{imp}∪{xor}∪{and}

Let `a1, a2, a3` ∈ Δ, and let the predicate `isConnector(a1, a2)` which is true when `a1` is effectively connected to `a2` via `connector`, we have:

- `then(a1, a2)` means that `a2` will be executed after `a1`. This connector is set when:
  `a2.precond == a1.pn_satisf`,
- `imp(a1, a2)` is possible only if `a1` ∈ `{AC}`. After this connection, `a2` is hence one of subactions of `a1`. For information, the proposition
  `imp(a1, a2)` ∧ `imp(a1, a3)`
  does not automatically involve `isThen(a2, a3)`. In fact, `a1` ignores the relation between its subactions.
- `xor(a1, a2)` means that `a1` and `a2` cannot be simultaneously executed. This connector is set when `a1` ∈ `a2.list_conflicts`. If `a2` is a AC and `a1` is in conflict with `a2`, then `a1` is automatically in conflict with all subactions of `a2` even if no explicit connector is set. This is the *law of conflict*. Formally, if `isXor(a1, a2)` ∧ `isImp(a2, a3)` ⇒ `isXor(a1, a3)`.
- `and(a1, a2)` is the connector by default if no connector is set between actions and the law of conflict does not hold. This connector means that agent can simultaneously execute `a1` and `a2`.

Relation between W and P

This relation is set by the fact that Δ is composed by the set of `PN.action[state]` issued from all PN in Π. Inversely, if Δ=`{ai}` then Π=`{pn_satisf{ai}}`.

Besides, the existence of AC involves us to introduce the notion of *decomposition level* (noted `dl`) in Ω. This parameter situates the place of each action (and then their associate `pn_satisf`) in the network. Independently of the application, the entry-point of Ω is a AC generically named *net_entry* and having a `dl` as 0.

If dl(`a1`)=`k` and `isImp`(`a1`, `a2`) then dl(`a2`)=`k+1`.

2.3 The NIM

Generalities

As we have said, the initialization of the system by the user is made via Π and Ω. Once this user level task is performed, the agent drives these two components in a generic way via the NIM. For that, the NIM cyclically performs an algorithm called algorithm of proactivity whose main role is to select the most important needs (i.e. those which have to be treated first) in Π. This selection of motivations is followed by that of *primitives* that the agent should next simultaneously perform.

The algorithm of NIM is based on two functions:

- `isImportantBetween(PN, PN')` which determines the most important need between two PN. In this paper, the notion of importance is noted by '>'.
  Note that this function acts only on Π, i.e. it works independently of the studied Ω.
- `algoNIM(AC, dl)`, firstly introduced in this paper, which manipulates all needs and actions in the system.

The first function

It selects the most important need by applying the successive following generic criteria:

- `type`: the rule is that LN > HN
- `level`: if the two PN have the same type, the NIM detects their level according to the specification of Abraham Maslow: a lower need is more important.
- `rank`: the NIM detects the rank if the above criteria cannot determine the most important need.
- then, the NIM sees the *states*, based on the rule: insufficient/excessive>limit_low/limit_high>sufficient.

The above general steps are actually more dynamic. For example, even if LN>HN and LN is in `sufficient` state while HN is in `insufficient` one, then HN>LN.

The second function

It is more complicated than the first one. Let:

- `ListPN` the list from which needs are to be selected,
- `finalPrim` the list which will take the final primitives.

then, `algoNIM(AC, dl)` acts as follows (note first that the initial value of AC is `net_entry` and `dl` is 0):

1. for each `act` ∈ Δ / `isImp`(AC, `act`) do
   `listPN=listPN + {pn_satisf(act)}`
2. removing from `listPN` all PN/ `currentstate==sufficient`.
3. sorting `listPN` from the most important to the less important PN. The function `isImportantBetween` is used when comparing two elements.
4. generating `listAct`. It is composed by the set of `PN.action[currentstate]` of each PN in `listPN`,

5. removing act ∈ listAct / act.precond is not sufficient (i.e. not verified). This step makes the selection between actions connected by then

6. for each act ∈ listAct, successively do:
   - if act ∈ {PR} ⇒finalPrim=finalPrim+{act}
   - if act ∈ {AC}
   ⇒ finalPrim=finalPrim+{algoNIM(act, dl+1)}.
   Thus, each time this step is performed, finalPrim is progressively filled by primitives.

The above six steps are finished when at any *dl* where algoNim is called during the selection, the sixth step does contain no more act ∈ listAct / act ∈ {AC}.

At the end, we have a given value of the list finalPrim. We then apply the xor criteria among primitives as follow: ∀ pr_i, pr_j ∈ finalPrim, ∀ i, j their respective position in finalPrim (with i < j), if isXor(pr_i, pr_j), then remove pr_j from finalPrim. We remove pr_j instead of pr_i because pr_j comes from a less important need resulting from the sorting in step 3.

The remaining primitives in finalPrim will be those simultaneously executed by the agent. These primitives are implicitly connected by the and connector.

3 Experimentation

To evaluate our model, we use for the first time the ADK platform developed by Calderoni (2002). Initially, ADK was mainly designed to simulate reactive agents. ADK had no cognitive structure at all.

The architecture of an ADK agent is based on three components: sensors, effectors and the control architecture.

Our concern is about the improving of this controller architecture. Particularly, we deal with the way the behavior rules are managed there. Our case study is RDK, a specialization of ADK to the robot foraging problem The scenario simulated in RDK is the following: in a given physical 2D environment, there is a situated robot, a set of blue and red pucks, one blue base, one red base and some obstacles. The robot first has to explore (action explore) the environment to find pucks, then it comes up to the detected puck region and acquires the closest puck (action acquireClosestPuck). Then, it delivers this acquired puck to the *base* having the same color color as it (action deliver[Color]Puck). In its moving, the robot has to avoid obstacles. Due to the size of the robot's gripper, it can deliver only one puck at a time. Furthermore, in case of reactive robots, each base respectively utters a signal, to help these robots to find where to deliver the pucks.

The objective of the experiment is twofold:

- to evaluate how our approach lets the robot to better reach by itself its goal (i.e. delivering the pucks to their respective base) with a less user intervention;
- to show the possible application of our generic model based on LN, in hybrid context.

3.1 The three experimented scenarios

Scenario (a): preprogrammed rules

This corresponds to the initial version of RDK where the behavior rules were handcoded at application level. They are only made by an automatic successive repetition of <explore, acquire[blue/red]puck, deliver[blue/red]puck> until all pucks are delivered to their respective base.

Scenario (b): generic rules with a reactive robot

In this scenario, we introduce into the controller our generic architecture based on natural motivations. The robot behavior will hence be managed by the algorithm of the NIM. However, a user first has to introduce all information needed by the applicative scenario. During that initialization phase, users are free to introduce motivations according to its own semantic interpretation of a given instance of motivations. Indeed, such an interpretation may possibly differ from one user to another (e.g. from a sociologist to a psychologist).

Unlike the preprogrammed scenario, we add the fact that the robot may be hunger or tired, in order to show that those needs can be introduced without a coding process.

Scenario (c): generic rules with a cognitive robot

In (b), we deal with a reactive robot, that is, a robot which acts only according to what it perceives from its sensori-motor (Ferber, 1997). In order to valid the model in cognitive domain (and then in hybrid one), we add Scenario (c) in which the robot has a partial representation of the environment, particularly the last location where a puck has been found, and the respective location of the two bases. Thus, we here remove the signals.

3.2 User initialization

As in Scenario (a), the initialization corresponds to a direct preprogramming of the behavior, we immediately present in this section the user initialization according to the generic MASLOW model.

Initialization of all pn_satisf

no_hunger=("eat", 1, 1, state_hunger, {eat, eat}} with

state_hunger=(interval, <[0…2[(insufficient), [2…5] (limite_b),]5…8] (sufficient)>}

fatigue_away=("fatigue", 1, 1, state_fatigue, {pause, sleep}} with
state_fatigue=(interval, <[0…3] (sufficient),]3…4,5] (limite_h), [4,5…7] (excessive)>}

obstacle_away=("obstacle_away", 2, 1, state_obstacle, {avoid}} with
state_obstacle=(boolean, <is_obstacle_away>}

loved=("being_loved", 3, 1, state_love, {search_agent}} with
state_love =(boolean, <is_loved>}

all_puck_delivered=("all_pucks_delivered", no_hunger, state_apd, {deliver_all_pucks}} with
state_apd=(interval, <[0 ⟶ 3] (insufficient), [3 ⟶ 5] (sufficient)>}.

We assume here that the HN all_puck_delivered is motivated by the need no_hunger, that is, the conveying is the work of the robot in its general life and the resulting wage will be used to buy foods. If the pucks to be conveyed is for example used to reinforce the security of the bases, we would first have created a LN like "to_be_secure" having a level=2 and rank=2, and would have associated the above HN to this LN.

All above needs and actions are situated at dl = 1.

env_explored=("env_explored", no_hunger, state_eve, {explore}} with
state_eve =(boolean, <puck_visible or puck_in_gripper>}

red_acquired=("red_acquired", state_red_acq, {acquireRedPuck}} with
state_red_acq =(boolean, <red_puck_in_gripper or not red_puck_visible>}

red_delivered=("red_delivered", no_hunger, state_red_dlv, {deliverRedPuck}} with
state_red_dlv =(boolean, <at_red_base and not red_puck_in_gripper>}

blue_acquired=("blue_acquired", state_blue_acq, {acquireBluePuck}} with
state_blue_acq =(boolean, <blue_puck_in_gripper or not blue_puck_visible>}

blue_delivered=("blue_delivered", no_hunger, state_blue_dlv, {deliverBluePuck}} with
state_blue_dlv =(boolean, <at_blue_base and not blue_puck_in_gripper>}

Initializing all preconditions

prec_explore = ("prec_explore", no_hunger, state_precond_explore) with
state_precond_explore = (boolean, <not puck_visible and not puck_in_gripper>)

prec_red_acquired=env_explored

prec_red_deliver=red_acquired

prec_blue_acquired=env_explored

prec_blue_deliver=blue_acquired

explore.precond=prec_explore

acquireRedPuck.precond= prec_red_acquired

deliverRedPuck.precond= prec_red_deliver

acquireBluePuck.precond= prec_blue_acquired

deliverBluePuck.precond= prec_blue_deliver

Setting the action connections

The then connectors have been here implicitly created by the steps before. For instance, given that

- prec_blue_deliver=blue_acquired
- acquireBluePuck.pn_satisf= blue_acquired
- deliverBluePuck precond= prec_blue_deliver,

⇒ then(acquireBluePuck, deliverBluePuck) is implicitly created. The same principle is valid for all other needs and actions in this application.

The imp and the xor connectors are set as follow:

```
imp(deliverAllPucks, explore)
imp(deliverAllPucks, acquireBluePuck)
imp(deliverAllPucks, acquireRedPuck)
imp(deliverAllPucks, deliverBluePuck)
imp(deliverAllPucks, deliverRedPuck)

xor(acquireRedPuck, deliverBluePuck)
xor(deliverRedPuck, acquireBluePuck)
xor(sleep, deliverAllPucks)
xor(sleep, eat)
xor(sleep, avoidObstacle)
xor(pause, deliverAllPucks)
```

With the imp connector, actions such as explore, acquireBluePuck and their associate pn_satisf are automatically situated at dl=2.

3.3 Experiment results

Figure 1 shows the robot behavior related to each scenario. The numbers found in their trail are actions. This figure shows that the robot behavior in Scenario (b) is as *coherent* as that presented in Scenario (a), the human preprogrammed scenario while in both scenarios, the objective to delivering all pucks is progressively performed. The advantage of MASLOW is however that our intervention as a user, in behavior preprogramming is largely reduced. The robot is more capable by itself to reach its goal. And even after adding needs, this coherence in behavior is maintained.

Besides, as we can see, our model can also be used in hybrid scenarios. We agree that the robot behavior is

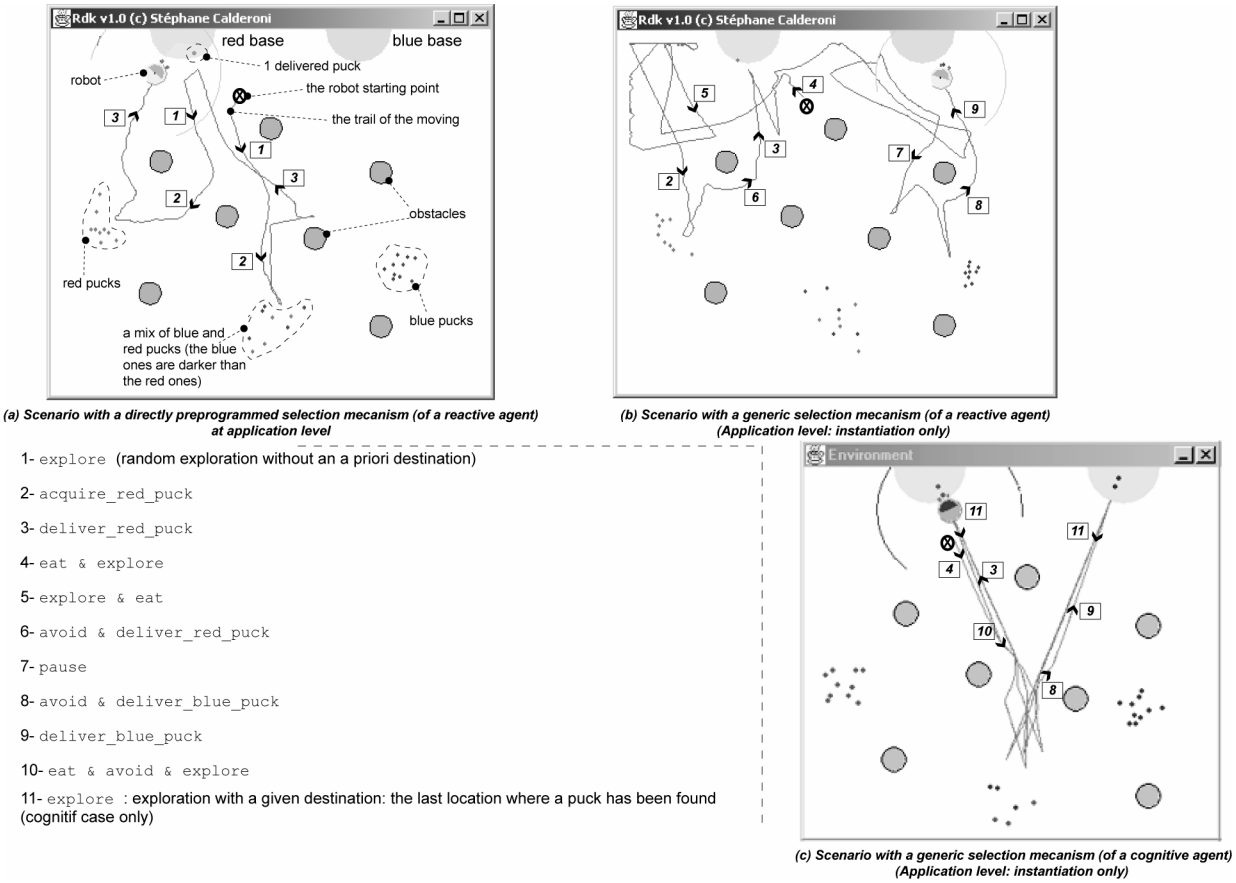

Figure 1: The robot behavior in the three experimented scenarios

different in (b) and (c). Indeed, unlike the reactive scenarios in which the robot always proceed to a new exploration, the agent in Scenario (c) comes back to the last location where it has previously found a puck. However, this behavior results from its (partial) knowledge of the environment and this difference can be explained at behavioral level, not at motivational one where our current work is situated.

4 Discussion

4.1 Natural motivations in agent models

As we have told in Introduction, the consideration for natural motivations in a generic way does not really exist in artificial agent design, except in animat models. Even the concept of *desire* in BDI ones, also called "motivational attitudes" by (Brazier and al. 1999) is actually an abstract representation of natural motivations at a higher level. However, the interest to extend this concept to hybrid agents is that since natural motivations permanently exist into artificial agents, they constantly influence the agent behavior whatever its type (reactive, cognitive, hybrid), its current goal, as well as its abilities to fulfill this goal (reasoning, planning, etc.).

Currently, a hybrid agent fulfills its plan while avoiding obstacles, running away from enemy, etc. (remind Introduction), but there is no general consideration for the fact that this cognitive plan may be interrupted because the agent is hunger or wants to sleep. The idea in this work is then to reduce this limitation by giving the user the possibility to introduce as many natural motivations as possible. And since it is impossible to list all of them, the idea is to set a generic specification of them. Afterwards, the application can instantiate them.

4.2 About the modeling of MASLOW

The network W

Ω can be compared to the ANA architecture initially developed by Maes (1991) in which the link between actions are emphasized via the successor links (similar to the `then` connector, and the conflict links (similar to the `xor` connector). The main difference is that in ANA, the degree of motivations for each action is

determined by only its activation level whilst in MASLOW, the selection of motivations are also guided by other generic criteria like `level`, `type`, etc. Remind that the level criterion is issued from a real-world study: the pyramid of Maslow.

A less user intervention

As we have said, the users' intervention in the agent behavior rules programming is reduced as we here prevent them to still hardcode. This reduction of the user intervention is useful because not only everybody is not specialized in computer programming. Additionally, it shows that our agent is now more able to reach its goal by itself.

Note however that the network in our model contains a methodological problem at user level. It is about the connection of actions. Indeed, the determination of the actions in conflict (related to the `xor` connection) is not always obvious at a first glance, especially when the application complexity (i.e. the involved manipulated data number) is increasing. Currently, only the simulation can help to progressively detect such connections. We are currently analyzing this issue.

5 Conclusion

In this paper, we attempt to build a generic model based on natural motivations (i.e. motivations related to the satisfaction of natural needs) intended to hybrid agents, as it is currently found in animat models. To deal with this problem, we base our model to the Abraham Maslow's pyramid of needs (managing the motivations) and especially try to find the generic criteria for the motivation selection process of the agent.

As for the evolution of our agent model MASLOW itself, we plan to integrate the learning capacity in the model so that the motivation selection process, is further dynamic. In addition, the social part of the model, and especially the interaction, will be emphasized.

References

Andriamasinoro Fenintsoa, Courdier Rémy (2002). *The Basic Instinct of Autonomous Cognitive Agents*. In Proceedings of 1st International Congress on Autonomous Intelligent System (ICAIS'2002), February 12th-15th, Geelong, Australia, in CD-Rom, and Abstract in World Scientific and Engineering Academy and Society Press, (ISBN 3-906454-30-4), p. 61.

Au Sherlock, Liang Jiasen, Parameswaran N (1998). *Progressive plan execution in a dynamic world* in Dynamic and Uncertain Environments Workshop. In Artificial Intelligent Planning, Eds. *Ralph Bergmann, Alexdrader Kott,* Pittsburgh USA, AAAI, pp 136-143.

Brazier Frances, Dunin-Keplicz Barbara, Treur Jan, Verbrugge Rineke (1999). *Beliefs, Intentions and DESIRE*. Modeling internal Dynamic behavior of BDI agents. In JJ Meyer, PY Schobbens (eds.) Formal Models of Agents: ESPRIT project Modelage final workshop, Lecture Notes in AI, volume 1760, Springer, pp 36-56.

Calderoni S. (2002). *Ethologie Artificielle et Contrôle Auto-Adaptatif dans les Systèmes d'Agents Réactifs: de la Modélisation à la Simulation.* Thèse à l'Université de La Réunion, 155 pages.

Gershenson Carlos, González Pedro Pablo, Negrete Jose Martinez (2000). *Action Selection Properties in a Software Simulated Agent*., in Cairó et. al. (Eds.) MICAI 2000: Advances in Artificial Intelligence. Lecture Notes in Artificial Intelligence 1793, Springer-Verlag, pp. 634-648.

Guillot Agnès, Meyer Jean Arcady (1998). *Synthetic Animals in Synthetic Worlds.* In Kunii et Luciani (Eds.) *Cyber Worlds*. Tokyo: Springer Verlag, pp.111-123.

Maes Pattie (1991). *A Bottom-Up Mechanism for Action Selection in an Artificial Creature*. From Animals to Animats: Proceedings of the Adaptive Behavior Conference '91, edited by S. Wilson and J. Arcady-Meyer, MIT Press, February, pp. 238-246.

Malec Jacek (2000). *On Augmenting Reactivity with Deliberation in a Controlled Manner*. Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems at the 14th European Conference on Artificial Intelligence (ECAI), Berlin, Germany, Lecture Notes in AI, volume 2103, Springer, pp. 76-91.

Maslow Abraham (1954). *Toward a Psychology of Being*, 3rd Edition. Edited by Lowry Richard, J.Wiley & Sons, Inc, November 1998, 320 Pages.

Mavromichalis Vangelis Kourakos, Vouros George (2000). *ICAGENT: Balancing between Reactivity and Deliberation*. In Workshop on Balancing Reactivity and Social Deliberation in Multi-Agent Systems at the 14th European Conference on Artificial Intelligence (ECAI), Berlin, Germany, Lecture Notes in AI, volume 2103, Springer, pp. 53-75.

Newell Allen (1982). *The knowledge level*. In Artificial Intelligence. Volume 18, n°1, pp. 87-127.

Tambe Milind (1996). *Executing Team Plans in Dynamic Multi-agent environments*. In AAAI Fall Symposium on Plan Execution: Learning Complex Behaviors In Adaptive Intelligent Systems, November 9th-11th, Boston USA.