TOWARDS MORE FLEXIBLE BDI AGENTS

Saadi Adel, Maamri Ramdane and Zaïdi Sahnoun
LIRE Laboratory, Mentouri University, Constantine, Algeria

Keywords: BDI agents, Practical reasoning, Decision making, Goal’s attributes.

Abstract: BDI agents are among the most popular models for the development of intelligent agents. The practical reasoning within the most of BDI models and architectures rely, in the best case, on three kinds of attributes: The utility associated with a goal, the cost of a plan and the uncertainty associated with the action’s effects. Based on a richer set of practical reasoning’s attributes, we propose a BDI architecture which aims to provide a step towards more flexible BDI agents.

1 INTRODUCTION

In a number of applications and fields, the software is required to be flexible and autonomous. This need brought about the "intelligent agent" paradigm. Because of the importance of this paradigm, the artificial intelligence is sometimes defined as a computer science subfield which aims to construct agents behaving intelligently (Wooldridge and Jennings, 1995). The interest in the agent technology gives rise to a range of models. The Belief-Desire-Intention (BDI) models are among the best known approaches to design intelligent agents. The BDI formalism lies on the Bratman's philosophical theory (Bratman et al., 1988) which argues the importance of intention in resource-bounded practical reasoning (Rao and Georgeff, 1991). Indeed, an intention constrains and supervises the future decisions (Wooldridge, 1999). Moreover, commitment embodies a trade-off between the reactivity and goal-directedness of an agent-oriented system (Rao and Georgeff, 1995). This trade-off is important for an agent situated in a dynamic environment with time constraints.

The practical reasoning within the most of BDI models and architectures (Bratman et al., 1988; Rao and Georgeff, 1991, 1995; Schut et al., 2004; Casali et al., 2009; Rahwan and Amgoud, 2006) rely, in the best case, on three kinds of attributes: The utility associated with a goal, the cost of a plan and the uncertainty associated with the action’s effects. Based on a richer set of practical reasoning’s attributes, we detail the generic BDI architecture described in (Wooldridge, 1999). The resultant architecture aims to provide a step towards more flexible BDI agents. In this paper, we define the flexibility of an agent as the ability, of the agent, to change its behaviour according to the situation. This definition was inspired from the definition of the adjective “flexible”, taken from “Cambridge Avanced Learner’s Dictionary”, which means to be able to change or to be changed easily according to the situation.

From the BDI agents’ literature, it is worth noticing that there is no consensus about the definition of the concept of goal, and its relation with desire and intention. In our BDI architecture, we adopt the same point of view about goal as in (Morreale et al., 2007):

> Desires and intentions are mental attitudes towards goals, which are in turn considered as descriptions of objectives. Thus, “pursing the goal g” is only a desire if the agent is not yet committed to it, due to some reason. On the other hand, “pursuing the goal g” becomes an intention when the agent is committed to it and work to achieve it. (p. 336)

We find the same point of view concerning the relation between goal, desire, and intention, in (Braubach et al., 2004). In this last work, each goal has a *life cycle* which is composed of some *states*. Each state expresses a different *agent’s attitude toward the goal*. The agent can see the goal as merely desired (There is a desire toward the goal, i.e., the goal has the state “option”) because it believes that the goal is not possible. On the other hand, the agent considers that pursuing the goal is an intention (There is an intention toward the goal, i.e., the goal has the state "active") when it is currently trying to achieve it. In our architecture, the agent's decision making process is based on the management of the state transitions of goals. The state of the goal is among the important attributes of the practical reasoning. For the other attributes, we inspired them from the work of Beaudoin (1994). This later provides a detailed analysis of the concept of goal, which suggests a rich set of goal's attributes. The processes that operate on goals were also presented. Nevertheless, as Beaudoin affirms in (Beaudoin, 1994), the proposed architecture is broad and shallow (i.e., which includes a large and rich set of functions and capabilities but which are not sufficiently detailed).

The next section of this paper gives an overview about the generic architecture of BDI agents. Section 3 presents the proposed architecture. The last section ends with conclusion and perspectives.

2 THE GENERIC BDI ARCHITECTURE

The generic architecture of BDI agents contains the following components (Wooldridge, 1999):

- A set of current beliefs.
- A belief revision function (Brf): It is a function which, on the basis of perceptual input and the agent's current beliefs, produces a new set of beliefs.
- An option generation function: It determines the options (desires) available to the agent, on the basis of its beliefs and intentions.
- A set of current options: It represents possible courses of actions available to the agent.
- A filter function (filter): On the basis of current beliefs, desires, and intentions, this function determines the agent's intentions.
- A set of current intentions: This set represents the current focus of the agent.
- An action selection function: It determines an action to perform on the basis of current intentions.

In the next section, based on a rich set of practical reasoning's attributes, we are going to detail the generic BDI architecture.

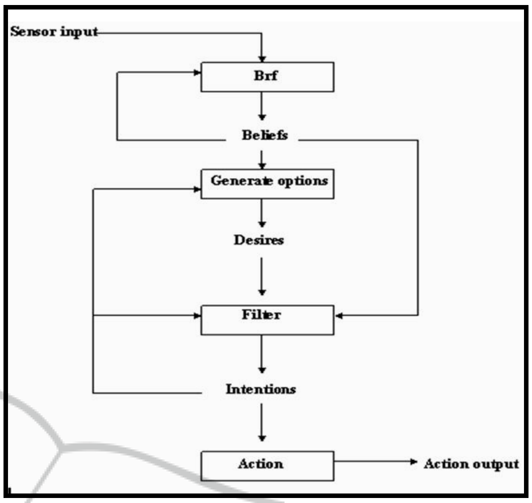

Figure 1: A generic BDI architecture (From (Wooldridge, 1999)).

3 PRESENTATION OF THE ARCHITECTURE

The proposed architecture is structured in 6 data structures constituting the agent's internal state **S** and 9 modules (noted in italic) defining the set **C** of modules (See Figure 2):

$$\textbf{Agent} = \textbf{S} \cup \textbf{C} \qquad (1)$$

with: **S={M, B, LP, SP, G, I_M_Q}** and **C={*MU*, *BR*, *GG*, *FS*, *F*, *AE*, *DTUM, APU,GM* }**.

3.1 The Agent's Internal State

The internal state of the agent, noted by **S**, contains the following 6 data structures:

- **M** is the **motives set**. Motives can be viewed as higher-level non-derivative components that characterize the agent and from which goals are generated (Munroe et al, 2003; Norman and Long, 1995a, 1995b). In our architecture, each motive **m∈M** is viewed as a record **m=<Pr, Alt, I>** with **m.Pr ∈[0, 1]** is the priority of the motive m, **m.Alt** is the set of all alternative goals that can be generated from m, and **m.I ∈[0, 1]** is the intensity of m, which is updated as the environment changes.
- **B** is the **beliefs set**. It contains what the agent knows about the world.
- **LP** is the **plan library** of the agent. We assume that a plan **p** is composed from a goal attribute **gl** (the goal for which the plan will be executed), a pre-condition **p-c** (The condition

that must be true for p to begin execution), an in-condition **i-c** (The condition that must remains true during the execution of p), the body **b** (The actions composing p), a post-condition **e** or plan's effects, **c**$\in$**[0, 1]** the cost of executing p, and **r** the probability of achieving g if we apply p. We note p as a record **p=<gl, p-c, i-c, b, e, c, r>** (This structure of a plan was inspired from (Casali et al., 2005; Thangarajah et al., 2003)). A plan p, which is *not in execution*, is said to be an *applicable plan*, in the current situation, for a goal g, iff **gl="g"** and the pre-condition **p-c** of p is satisfied in the current situation. In the same manner, we say that a plan p, which is *actually trying to achieve g*, is an applicable plan, in the current situation, iff **gl="g"** and the in-condition **i-c** of p is satisfied in the current situation.

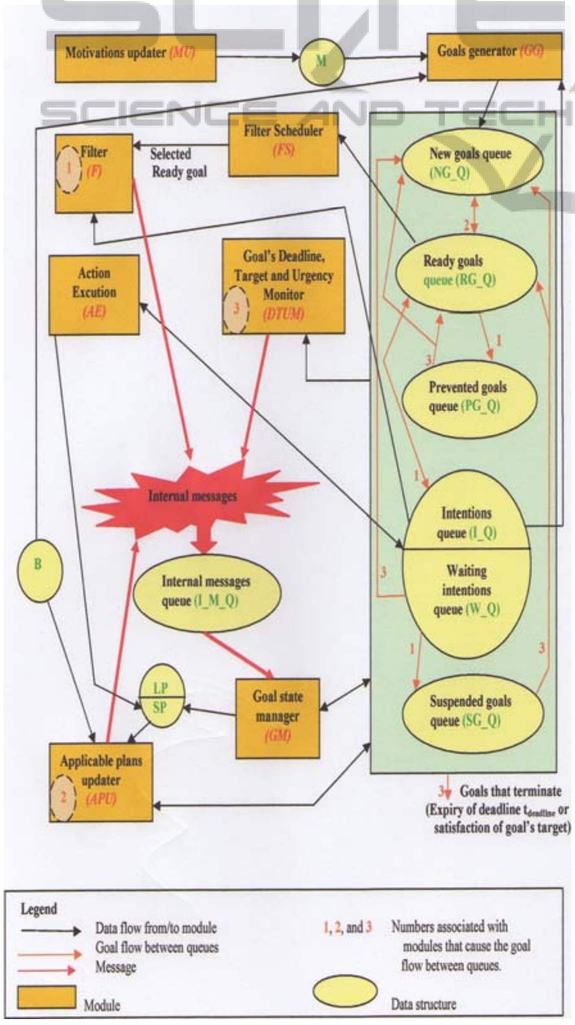
Motivations updater (MU)
M
Goals generator (GG)
Filter Scheduler (FS)
Filter (F)
1
Selected Ready goal
New goals queue (NG_Q)
2
Ready goals queue (RG_Q)
Action Excution (AE)
Goal's Deadline, Target and Urgency Monitor (DTUM)
3
1
Prevented goals queue (PG_Q)
Internal messages
Intentions queue (I_Q)
Waiting intentions queue (W_Q)
B
Internal messages queue (I_M_Q)
LP
SP
Goal state manager (GM)
Suspended goals queue (SG_Q)
Applicable plans updater (APU)
Goals that terminate (Expiry of deadline t_deadline or satisfaction of goal's target)
Legend
Data flow from/to module
Goal flow between queues
Message
Module
1, 2, and 3
Numbers associated with modules that cause the goal flow between queues.
Data structure

Figure 2: The proposed architecture (To avoid overload in the figure, we have omitted the *BR* module).

- **SP** is the set of plans whose execution was suspended for some reason (For example because the in-condition is actually not satisfied).
- **G** is the **goals set**. In our work the goal g is viewed as a record of the following properties :

$$\mathbf{g = <Target, State, U, Motiv, t_{begin}, t_{deadline}, t_{deadline\text{-}AE}, t_{Urg0}, t_{Urg1}, Urg, Interrupted, Prevented, Waited\_Int, App\text{-}Plans, A\text{-}Plan>} \qquad (2)$$

where:

- g.Target represents the world state the agent wants to bring about.

- g.State$\in${"New", "Ready", "Prevented", "Active", "Suspended"} is the state of the goal:

g.State="New" means that g is not actually pursued by the agent and that g has no applicable plans in the current situation. This state is considered as a "waiting" state for g, where g waits for the availability of applicable plan.

g.State="Ready" means that g has at least one applicable plan but is not actually pursued by the agent. This state is considered as a waiting state for g, where g waits for the activation.

g.State="Active" means that the agent is actually trying to achieve the goal g (In this case, g is called an active goal, i.e., a goal towards which the agent has an intention). In this case, the agent is either executing a plan in order to achieve g or either waiting for an *applicable*, *free-conflict* plan for g. We say that a plan **p** is in *conflict* with another plan **p'** if the plans' effects **p.e** and **p'.e** are *inconsistent*. In this paper, we say that a plan p is a *free-conflict* plan if it doesn't conflict with plans currently achieving active goals.

g.State="Prevented" means that the goal g is waiting for the termination of active goals g' which are in conflict with g or is waiting for the urgency event (i.e., when g.Urg=1). g was considered for the activation but was prevented from the activation, by active goals g' (g was *prevented from the activation*, because g is in *conflict* with active goals g' and g is *not urgent*. See section 3.2, especially the paragraph concerning the filter module for more details). When the active goals g' terminate or when the urgency event g.Urg=1 appears, the goal g is moved from the "prevented" state to "Ready" or "New" state (according to the availability of applicable plans).

g.State="Suspended" means that the goal g is waiting for the termination of an active conflicting goal g'. This later caused the interruption and the suspension of the execution of g because g' is *urgent* and *more important* for the agent than g. When the active goal g’ terminate, the goal g is moved from the “suspended” state to “Ready” or “New” state (according to the availability of applicable plans).

It is worth noticing that in the case where **g.State ∈{“New”, “Ready”, “Prevented”, “Suspended”}**, pursuing the goal g is only a *desire*. Whereas in the case **g.State=“Active”**, pursuing g becomes an *intention* (See the introduction of the paper for more details about our point of view about the relation between goal, desire, and intention). In the remainder of this paper, we will use the previous goal’s states as adjectives of the world “goal”. For example, when we say “Prevented goal”, we mean that we have a goal with the state “Prevented”.

- **g.U** $\in$ **[0, 1]** is the **utility** value associated with the goal g. The calculation details of g.U are outside the scope of this paper.

- g.Motiv is the motive that gives rise to the goal g.

- The interval **[g.t$_{begin}$, g.t$_{deadline}$]** represents the period during which the goal g must be achieved. If there is no imposed beginning time to g, we take **g.t$_{begin}$=0**. Similarly, in the case where there is no imposed deadline time we write **g.t$_{deadline}$** = +∞.

- g. t$_{deadline-AE}$ is a deadline time that is attributed to an active goal g if the plan that is actually executed to achieve g is not applicable in the current situation and if **g.t$_{deadline}$** = +∞ (See section 3.2, precisely the paragraph concerning the “action execution” module for more details). If there is an imposed deadline **g.t$_{deadline}$** , **g. t$_{deadline-AE}$** will take the value +∞.

- The interval **[g.t$_{Urg0}$, g.t$_{Urg1}$]** with g.t$_{Urg0}$ ≥g.t$_{begin}$ and g.t$_{Urg1}$<g.t$_{deadline}$ represents the critical period during which the agent must begin the realization of g. Otherwise, it is greatly probable that the execution time of g will exceed the deadline time g.t$_{deadline}$, and thus it will fail (In this paper, we do not consider the case of partial achievement of a goal). In the case where **g.t$_{deadline}$** = +∞ there is no urgency for the agent to begin the realization of the goal. We express this situation by taking **g.t$_{Urg0}$ = g.t$_{Urg1}$**= +∞.

- g.App-Plans is the set of applicable plans available in the current situation, for the goal g. The two plans set LP and SP are taken into account when calculating the set of applicable plans.

- g.Urg is the urgency function. It is a time-varying function (In the following formula of Urg, t represents the time) and is derived basically from the interval [g.t$_{Urg0}$, g.t$_{Urg1}$]:

If the urgency function of g take 1 then the agent must immediately begin the realization of g (If there is no active goals conflicting with g and which are more important than g). The calculation of urgency function is inspired by the “alarm function” used to calculate the motivation intensity (Norman and

$$\mathbf{g.Urg(t)=}\begin{cases} \mathbf{1,\ \underline{If}\ (((g.State=\text{“Ready”})\ or} \\ \mathbf{(g.State\in\{\text{“Prevented”, “Suspended”}\}} \\ \mathbf{and\ g.App\text{-}Plans\neq\emptyset))\ and\ \ (g.t_{deadline}} \\ \mathbf{\neq +\infty\ )\ and\ (t\in[g.t_{Urg0},\ g.t_{Urg1}])\ \ and} \\ \mathbf{(\nexists g2\in\ (g.Motiv).Alt:} \\ \qquad\qquad \mathbf{g2.State=\text{“Active”}))} \qquad (3) \\ \qquad \textit{\{This later existential condition} \\ \qquad \textit{means that it doesn't exist an} \\ \qquad \textit{alternative goal to g that is} \\ \qquad \textit{actively pursued by the agent\}} \\ \mathbf{\ \ or\quad g.Urg(t\text{-}1)=1} \\ \mathbf{0,\quad Otherwise} \end{cases}$$

Long, 1995b). If the time variables t$_{begin}$, t$_{deadline}$, t$_{Urg0}$, t$_{Urg1}$ are unknown or cannot be derived for the goal g, then the urgency function g.Urg is derived basically from the intensity of the motive g.Motiv that leads to g. In this case, it is urgent to begin the execution of g if the intensity **(g.Motiv).I** exceeds some *urgency threshold*.

$$\mathbf{g.Urg(t)=}\begin{cases} \mathbf{1,\ \ \underline{If}\ \ (((g.State=\text{“Ready”})\ or} \\ \mathbf{(g.State\in\{\text{“Prevented”, “Suspended”}\}} \\ \mathbf{and\ g.App\text{-}Plans\neq\emptyset))\ and} \\ \mathbf{(g.Motiv).I\geq urgency\text{-}threshold)} \qquad (4) \\ \mathbf{and\ \ (\ \nexists g2\in\ (g.Motiv).Alt:} \\ \qquad\qquad \mathbf{g2.State=\text{“Active”}))} \\ \mathbf{or\ \ g.Urg(t\text{-}1)=1} \\ \mathbf{0,\ \ Otherwise} \end{cases}$$

- **g.Interrupted** is a variable that takes 1 if the execution of g was interrupted by another conflicting goal. It takes 0 when g resumes its execution or when g was not interrupted until now.

- **g.Prevented** is a variable that takes 1 if g was considered for the activation but was prevented from the activation, by another conflicting active goal. It takes 0 when g was not yet considered for the activation or when g is considered for the activation but g becomes active.

- **g.Waited_Int** contains the active goals that caused the interruption or the prevention of g.

- g.A-Plan is the plan that is actually executed by the agent, to achieve the goal g.

After giving the structure of a goal, we will give in what follows the structure of the goals set G.

The **goals set G** is structured in 6 queues (See Figure 2):

- NG_Q (New Goals Queue): It contains “New” goals.

- RG_Q (Ready Goals Queue): It contains “Ready” goals.

- PG_Q(Prevented Goals Queue): It contains "Prevented" goals.
- I_Q (Intentions Queue) and W_Q (Waiting intentions Queue): I_Q contains active goals that are actually achieved via plans. **W_Q** contains active goals that are waiting for an applicable, free-conflict plan.
- SG_Q (Suspended Goals Queue): This queue contains "Suspended" goals.
We conclude the presentation of goals set G by defining the notion of "*goals conflict*" (In this paper, we focus on the conflict between a ready goal$\in$RG_Q and an active goal$\in$ I_Q$\cup$W_Q).
A ready goal g is said to be **in conflict with an active goal g'$\in$I_Q,** iff all the applicable plans of g (i.e., the plans of g.App-Plans) are in conflict with g'.A-Plan. If g has an applicable plan that doesn't conflict with g'.A-Plan then g and g' are not in conflict. On the other hand, a ready goal g is said to be **in conflict with an active goal g'$\in$W_Q** iff the goal targets of g and g' (i.e., g.Target and g'.Target) are inconsistent. In this paper we assume that the targets of two alternative goals g, g' (i.e., g.Motiv =g'.Motiv) are inconsistent. This assumption leads to the following property:

$$\forall \mathbf{g}\in\mathbf{R\_G},\ \forall \mathbf{g'}\in \mathbf{I\_Q}\cup\mathbf{W\_Q}\text{:}\ \mathbf{g.Motiv} = \mathbf{g'.Motiv} \Rightarrow \mathbf{g}\ \textbf{is in conflict with}\ \mathbf{g'} \quad (5)$$

After presenting the set G, we give, in what follows, the last data structure included in the agent's internal state S.

- **I_M_Q (Internal Messages Queue):** This queue receives all messages addressed to the module *GM* (Goals state Manager) from other agent's modules (For more details about the module *"GM"* see the subsection 3.2).

3.2 The Modules of the Architecture

The proposed architecture contains the 9 following modules working in parallel: ***MU, BR, GG, FS, F, AE, DTUM, APU,*** **GM** (These modules define the set C). The ***MU*** module **(Motivations Updater)** is responsible of updating the agent's motives set **(M)**. On the basis of beliefs set **(B)**, **(M)**, and active goals in **I_Q** queue, the ***GG*** module **(Goals Generator)** generates and updates the set of new goals (the NG_Q queue). The ***GM*** module **(Goal state Manager)** on the basis of messages transmitted by the other modules, updates the states of the goals. The ***FS*** module **(Filter Scheduler)** selects one ready goal, from the set of ready goals, i.e., the RG_Q queue (The *FS* module sorts the ready goals g on the basis of the motive priority **(g.Motiv).Pr**, the utility **g.U**, the urgency measure **g.Urg**, the flags **g.Interrupted** and **g.Prevented**). The chosen ready goal g is transmitted to the ***F*** module **(Filter)**, which will decide about its activation (i.e., adding it to the I_Q queue). If *F* decides to activate g, then the active goals g'$\in$I_Q$\cup$W_Q that are in conflict with g will be moved to the "Suspended" state, i.e., added to SG_Q queue (If g conflict only with an active goal g' which is an alternative to g, then g' will be moved to NG_Q or RG_Q queue, according to the availability of applicable plan). In the case *F* decides to not activate g, then g will be moved to the "Prevented" state. The "prevented" and the "suspended" goals g will be moved to the "Ready" or "New" state (According to the availability of applicable plans) when the goals g' that conflict with g (i.e., causing the suspension/prevention) terminate. The prevented goals g are also moved to the "Ready" or "New" state when the urgency event appears, i.e., g.Urg= 1 (The termination of a goal and the urgency of a prevented goal are signalled to the "Goal state Manager", by the ***DTUM*** module **(Goal's Deadline, Target, and Urgency Monitor)**). The ***AE*** module **(Action Execution)** is responsible of the achievement of active goals via plans. If this module finds that the executed plan of an active goal g (i.e., g.A-Plan) is not applicable in the current situation (the in-condition of g.A-Plan is not satisfied) then g will wait in the active state (in the W_Q queue) until the availability of an applicable and free-conflict plan for g. The waiting time of g in W_Q queue should not exceed the deadline $g.t_{deadline}$. If g has no deadline then the *AE* module will associate to g a deadline $\mathbf{t_{deadline\text{-}AE}}$.

In the following, we give details of the different modules:

- ***MU*** **(Motivations Updater):** This module is responsible of updating motives set. The details of updating motivations process are outside the scope of this paper.
- ***BR*** **(Belief Revision and update)**: This module updates and revises the set B on the basis of new perceptions. The details of this module are outside the scope of this paper.
- ***GG*** **(Goals Generator):** This module generates and updates the set of new goals on the basis of motives set M, beliefs set B, and I_Q queue. The produced goals are initially in "New" state, and then the "Goal state manager" updates their states in response of incoming events (see the paragraph about the "Goal state manager" for details). The details of this module are outside the scope of this paper.

- ***FS* (Filter Scheduler):** The task of this module is to select from the "ready goals queue" (RG_Q) one ready goal that will be considered for the activation by the filter module *F* (See the next paragraph concerning the filter). The ready goals g are sorted by *FS* on the basis of the motive priority **(g.Motiv).Pr**, the utility **g.U**, the urgency measure **g.Urg**, the flags **g.Interrupted** and **g.Prevented**.

  A ready goal g pertains to one of the 4 followings categories:

- The category RED1: Contains ready goals g with **g.Urg=1**. The goals of this category are placed by *FS* in the 1st places of RG_Q.

- The category RED2: Contains ready goals g with **g.Interrupted=1** and **g.Urg=0**. The goals of this category are placed in RG_Q, after goals of RED1 category.

- The category RED3: Contains ready goals g with **g.Prevented=1** and **g.Interrupted=g.Urg=0**. The goals of this category are placed in RG_Q, after goals of RED2 category.

- The category RED4: This category contains ready goals g with **g.Urg=g.Interrupted= g.Prevented=0**. The goals of this category are placed in RG_Q, after goals from RED3 category.

Assume that g1, g2, g3, and g4 are ready goals, such that: $g1 \in RED1$, $g2 \in RE2$, $g3 \in RED3$, and $g4 \in RED4$, then we have the following propriety:

$$\mathbf{g1 >_{RG} g2 >_{RG} g3 >_{RG} g4} \qquad (6)$$

where $>_{RG}$ is the precedence relation over ready goals ($g >_{RG} g'$ means that g precedes and is placed before g' in RG_Q).

Inside any category RED1, RED2, RED3, and RED4, the ready goals g are sorted according to the motive priority (g.Motiv).Pr. If two ready goals inside the category have the same motive priority then they will be sorted according to the utility g.U. Assume that we have two ready goals g and g' from the same category RED_i with $i \in \{1,2,3,4\}$, then we obtain the two following proprieties:

$$\forall g, g' \in RED_i: (g.Motiv).Pr > (g'.Motiv).Pr \Rightarrow g >_{RG} g' \qquad (7)$$

$$\forall g, g' \in RED_i: (g.Motiv).Pr = (g'.Motiv).Pr \text{ and } g.U > g'.U \Rightarrow g >_{RG} g' \qquad (8)$$

The ready goal that is transmitted by the module *FS* to the module *F* is the goal placed in the head of RG_Q.

- ***F* (Filter):** This module decides about the inclusion of ready goal g, chosen by the "filter scheduler", into the "Intentions queue".

If g has an applicable and free-conflict plan (i.e., g doesn't conflict with active goals of I_Q) and is not in conflict with active goals of W_Q, then the filter adds g to the "Intentions queue". In the case of conflict (i.e., there is no applicable and free-conflict plans for g or g conflicts with goals of W_Q), if the filter decides to include g in I_Q (The filter takes this decision, if g conflicts only with an active goal which is an alternative or g is an urgent goal and g is more important than the active goals conflicting with it) then all active goals conflicting with g are moved to the "Suspended goals queue" (If g conflict only with an active goal g' which is an alternative to g, then g' will be moved to RG_Q or NG_Q queue, according to the availability of applicable plan). In the case *F* decide to not activate g, then g will be moved to the "Prevented" state.

Before seeing the filter algorithm, we give some functions that will be used in it (Some functions are used by the other modules):

- Conflict-free(g∈G): This function returns the free-conflict plans of the goal g.

- Net-utility (g∈G, p∈LP∪SP): Assuming that p is a plan for the goal g (i.e., p.gl="g"), this function is calculated by the formula:

$$\mathbf{Net\text{-}utility\ (g, p) = p.r\ (g.U+(1-p.c))\ /2} \qquad (9)$$

with **p.r** is the probability of attaining g if we apply the plan p and **p.c** is the cost of executing the plan p (This Net-utility function was proposed in (Casali, 2005) to calculate the intention degree). The Net-utility function considers 3 parameters: the utility value of the goal g, the cost of the plan p achieving g, and the probability of achieving g if we apply p.

- Remaining (p∈LP): This function returns the actions of p that were not yet executed.

- I-Conf-G (g∈RG_Q): This function returns the set of active goals whose executed plans are in conflict with the applicable plans of goal g.

- W-Conf-G (g∈RG_Q): This function returns the set of active goals in W_Q queue, in conflict with goal g.

- I-Conf-PL (p∈ LP∪SP): This function returns the set of active goals whose executed plans are in conflict with the plan p.

- Net-utility2(g, p) is calculated by the formula:

$$\mathbf{Net\text{-}utility2(g, p) = (Net\text{-}utility(g, p) + (1\ /\ card(I\text{-}Conf\text{-}PL(p))))\ /2} \qquad (10)$$

with $\mathbf{I\text{-}Conf\text{-}PL(p) \neq \emptyset}$ and **card(S)** gives the number of elements in the set S. The Net-utility2 function considers 4 parameters: the utility value of the goal g, the cost of the plan p achieving g, the probability of achieving g if we apply p, and the number of active goals whose executed plans conflict with p. When the value of Net-utility increases and card(I-Conf-PL) decreases, the value of Net-utility2 increases

- BEST-PLAN-1(g∈G, s⊂ LP∪SP) is a function that retrieves which plan p'∈s for g maximizes the function Net-Utility(g, p). Its algorithm is described (in an abstract manner) by:

```
BEST-PLAN-1(g∈G, s⊂ LP∪SP): LP∪SP
Begin
  Find  p'∈s which is defined by:
Net-utility(g,p')=Max_{p∈s}Net-utility(g,p)
  Return p'
End
```

- **BEST-PLAN-2(g∈G, s⊂ LP∪SP)** is similar to the function **BEST-PLAN-1**. The only difference is that BEST-PLAN-2 uses **Net-utility2** instead of Net-utility

- In what follows, in the Filter algorithm, the procedure calls: ADD-I-Q, BEST-ALT, BEST-T-GS, PREVENTED are addressed to the goal state manager (See the last paragraph in this section, for more details about the *GM* module).

```
Filter Algorithm (g∈RG_Q)
Begin
   If g.Prevented=1 Then g.Prevented←0
   App-free-plans ← g.App-Plans ∩
                     Conflict-free (g)
   If (App-free-plans ≠ ∅)
   {There is at least an applicable and
          free-conflict plan for g}
       and
    (W-Conf-G(g)=∅)
    {There is no active goal in W_Q
    queue which conflicts with g}
   Then
   g.A-Plan←BEST-PLAN-1(g,
                          App-free-plans)
   {The plan g.A-plan is the one that
     maximises the function Net-utility}

      ADD-I-Q(g) {Move the ready goal g
                  to the I_Q queue}
   Else {g is in conflict with active
        goals∈I_Q∪W_Q}

   If g.Urg = 0 Then {g is not
                       urgent}
     If g is only in conflict with an
     Alternative goal g'∈ I_Q
     Then
        interesting-plans ← ∅
          For each p∈ g.App-Plans Do
           If Net-utility(g, p) >
Net-utility(g', Remaining(g'.A-Plan))
{The goal g taken with the plan p is
more interesting than the alternative
goal g'}  Then
           interesting-plans ←
                interesting-plans ∪ {p}
          End For
        If interesting-plans ≠ ∅  Then
          g.A-Plan←BEST-PLAN-1(g,
                        interesting-plans)

          BEST-ALT (g, g')
        {Replace the active goal g' by
         the alternative g}
        End If
      Else
        If g is only in conflict with
        an alternative goal g'∈ W_Q
       Then
          g.A-Plan←BEST-PLAN-1(g,
                              g.App-Plans)
          BEST-ALT (g, g')
       Else {the no urgent goal g
           conflicts with one or several
           active goals g'}
           PREVENTED(g){make  g in the
                     "Prevented" state}
   Else {g.Urg=1, i.e. g is an urgent
     goal which conflicts with one or
     several active goals g'}
     priority←False
      interesting-plans ← ∅
      If App-free-plans= ∅ Then
        For each p∈g.App-Plans Do
         If Net-utility(g, p)>
          $\sum_{g' \in I\text{-}Conf\text{-}PL(p)}$ Net-utility (g', g'.A-Plan))
     {i.e. the goal g taken with the
   plan p is more important than the
   set I-Conf-PL(p)}
          Then   interesting-plans ←
                 interesting-plans ∪ {p}
        End For

      If W-Conf-G(g)≠∅  Then
       If ∀g'∈W-Conf-G(g):
       (g'.Motiv).Pr<(g.Motiv).Pr
       Then priority ←True

      If(interesting-plans ≠ ∅) and
         (W-Conf-G(g)= ∅)
       Then
    g.A-Plan←BEST-PLAN-2(g,
                        interesting-plans)
   {The plan g.A-plan is the one that
   maximises Net-utility2}
      BEST-T-GS(g, I-Conf-PL(g.A-Plan))
      {replace active goals  g' by g}
      End If
      If (priority=True)
          and (App-free-plans ≠∅)
      Then   g.A-Plan← BEST-PLAN-1(g,
                        App-free-plans)
     {The plan g.A-plan is the one that
                maximises Net-utility}
           BEST-T-GS(g, W-Conf-G(g))
      End If
     If interesting-plans ≠ ∅)
         and (priority=True)
     Then  g.A-Plan←BEST-PLAN-2(g,
                     interesting-plans)
      BEST-T-GS (g, I-Conf-PL(g.A-Plan)
                          ∪W-Conf-G(g))
     End If
    End If   End
```

- ***AE*** **(Action Execution):** This module is responsible of the achievement of active goals via plans. If this module finds that the executed plan of an active goal g (i.e., g.A-Plan) is not applicable in the current situation (The in-condition of g.A-Plan is not satisfied) then g will wait in the active state (in the W_Q queue) until the availability of an applicable and free-conflict plan for g. The waiting time of g in W_Q queue should not exceed the deadline g.$t_{deadline}$. If g has no deadline then the *AE* module will associate to g a deadline $\mathbf{t_{deadline\text{-}AE}}$. We assume that $\mathbf{t_{deadline\text{-}AE}}$ is the same for all goals and will take the value $\mathbf{t_w}$.

```
Action execution Algorithm
Begin
    While   True   Do
      While (I_Q≠ ∅ )  Do

   Repeat Execute goals g of I_Q
        Until ((∃g1∈I_Q: g1.A-Plan is
        not applicable) or (∃g2∈W_Q:
g2.App-Plans∩Conflict-free(g2) ≠∅))
     For all g1∈I_Q: g1.A-Plan is not
      applicable Do
        ADD-TO-SP(Remaining(g1.A-Plan))
      {The ADD-TO-SP function is used to
     add a suspended plan to the set SP}
     If  g1.t_deadline =+∞
      Then  g1.t_deadline-AE ← t_w
     I_Q ← I_Q-{g1}
     W_Q ← W_Q ∪{g1}
     g1.A-Plan ←∅
     End for

     For all g2∈ W_Q:
    g2.App-Plans∩Conflict-free(g2)≠∅ Do
          If  g2.t_deadline =+∞
          Then  g2.t_deadline-AE ← 0
         g2.A-Plan← BEST-PLAN-1(g2,
           g2.App-Plans∩Conflict-free(g2))
          I_Q ← I_Q∪{g2}
          W_Q ← W_Q-{g2}
       End for
     End While
End While   End
```

- ***DTUM*** **(goal's Deadline, Target, and Urgency Monitor)**

This module monitors the expiry of goals deadlines $t_{deadline}$ / $t_{deadline\text{-}AE}$, the satisfaction of goals targets, and the appearance of urgent prevented goals (i.e., prevented goals with Urg attribute equal to 1). In the following we give its general algorithm (DELETE, NEW, READY are messages addressed to the goal state manager):

```
While   True   Do
   If  ∃g ∈G: g.t_deadline expires  Then
    DELETE(g){i.e.,Delete the failed
    goal g from G }
   If ∃g∈W_Q: g.t_deadline-AE expires Then
      g.t_deadline-AE ← 0
      NEW(g)
  {i.e., moving g to the New goal
queue. g was failed to continue its
execution and is given another chance
to restart but from the new state}
   End if
   If ∃g∈G: g.Target is satisfied in
    the current situation Then DELETE(g)
   If  ∃g ∈PG_Q: g.Urg=1 Then
      g.Waited_Int←∅
      If g.App-Plans≠∅ Then
      READY(g) {i.e., moving g to the
               Ready goal queue}
      Else   NEW(g)
   End If
End while
```

- ***APU*** **(Applicable Plans Updater):** This module updates for each goal g the set of applicable plans **g.App-Plans**. Also, it monitors the new and ready goals. If it notices for a new goal g that g.App-Plans≠∅, then it will send the message READY(g) to the goal state manager. If it notices for a ready goal g that g.App-Plans=∅, then it will send the message NEW(g) to the goal state manager.
- ***GM*** **(Goal State Manager):** All the events influencing the goal state are placed in the queue **I_M_Q**. This later is monitored by the module ***GM*** in order to manage and update the

goals states. The *GM* module associates for each type of message one procedure. This module contains the seven following procedures: ADD-I-Q, BEST-ALT, BEST-T-GS, PREVENTED, DELETE, NEW, and READY. In what follows we give the general algorithms of BEST-ALT, BEST-T-GS, and DELETE (The algorithms of ADD-I-Q, PREVENTED, NEW, and READY were omitted from the paper for a problem of space):

```
BEST-ALT (g∈RG_Q,  g′∈I_Q∪W_Q)
Begin
 Remove g′from the corresponding queue
   RG_Q ← RG_Q ∪ {g′}
   g′.State← “Ready”
   ADD-TO-SP(Remaining (g′.A-Plan))
   g′.A-Plan ← ∅
   RG_Q ← RG_Q - {g}
   I_Q ← I_Q ∪ {g}
   g.State← “Active”
   If g.Interrupted=1 Then
     g.Interrupted ←0
End
BEST-T-GS (g∈RG_Q,  gs⊂ I_Q∪W_Q)
Begin
       For each g′∈gs Do
           g′.Interrupted←1
          If g′∈I_Q Then
        ADD-TO-SP(Remaining(g′.A-Plan))
           g′.A-Plan ← ∅
          End If
         g′. Waited_Int←g
 Remove g′ from the corresponding queue
         SG_Q ← SG_Q ∪ g′
         g′.State← “Suspended”
    End for
       RG_Q ← RG_Q - {g}
       I_Q ← I_Q ∪ {g}
       g.State← “Active”
    If g.Interrupted=1 Then
       g.Interrupted←0
End
DELETE (g∈G)
Begin
   s←g.State
   If  g.Target is satisfied in the
    current situation
   Then
     For each g′∈G: g′.Motiv= g.Motiv Do
       DELETE(g′)
    Remove g from the queue associated
   with the g.State
     If s= “Active” Then
        If ∃g′∈ SG_Q∪ PG_Q:
            g∈g′.Waited_Int
           Then
         g′.Waited_Int← g′.Waited_Int-{g}
              If g′.Waited_Int = ∅ Then
                 If g′.App-Plans≠∅ Then
                     READY(g′)
                  Else  NEW(g′)
             End If

  End If   End
```

4 CONCLUSIONS AND PERSPECTIVES

In this paper, we argue that the first step towards more flexible reasoning in the actual BDI models and architectures is that the agent should consider the different and varied attributes useful for the decision making process (See (Beaudoin, 1994) for an example of such attributes). In fact, the practical reasoning within the most of BDI models and architectures rely, in the best case, on three kinds of attributes: The utility associated with a goal, the cost of a plan and the uncertainty associated with the action's effects. We have presented in this paper a BDI architecture based on a richer set of attributes (Inspired from the work of Beaudoin (1994)) that concerns fundamentally the characteristics of a goal (We have used for example the two attributes concerning the urgency and the state of a goal). This set of attributes is far from being exhaustive but it constitutes a step towards a more complete one. This set of attributes permitted us to detail the generic model of BDI agents. In our architecture, the module *FS* taken with the module *F* correspond to the "filter" function of the generic BDI model. The modules *GG*, *BR*, *AE* correspond respectively to the generic BDI model's functions: "option generation" function, Brf function, and action selection function. The decision making process in our agent is based on the *management of the state transitions of goals.* In fact, the proposed architecture includes the module *GM* (Goal state Manager) which, on the basis of the messages transmitted from other modules, updates the states of the goals. The presented work aims to provide a step towards more flexible BDI agents. Nevertheless, some points should be addressed in future works:

- Incorporate other types of goals as "maintenance goals", i.e., maintain some world state (Braubach et al., 2004). In this paper we have used only one type of goals: "achievement goals" (achieve some world state).
- Treat the case of partial achievement of a goal.

- Detail the "motivations updater", "goals generator", and "Belief revision and update" modules. Besides, we plan to incorporate uncertainty in the architecture and to study its impact on the whole agent's reasoning.
- The actual version of the architecture considers only one type of conflict between goals: Conflict that takes into account the inconsistencies between goals' targets and inconsistencies between plans' post-conditions. We plan to consider other kinds of goals conflict as conflict caused by the incompatibility between the post-condition of a plan p and the pre-condition of another plan p' (i.e., the execution of the plan p will prevent the execution of the plan p') (Rahwan and Amgoud, 2006; Thangarajah et al., 2003).
- Experiment and evaluate the architecture in a simulated worlds and scenarios.

REFERENCES

Beaudoin, L., 1994. Goal processing in autonomous agants. PhD Thesis, University of Birmingham.

Bratman, M. E., Israel, D. J., Pollack, M. E., 1988. Plans and resource-bounded practical reasoning. Computational intelligence, 4(3), 49-355.

Braubach, L., Pokahr, A., Moltdt, D., Lamersdorf, W., 2004. Goal representation for BDI Agent Systems. In: Proceedings of ProMAS04.

Casali, A., Godo, L., Sierra, C., 2009. g-BDI: A Graded Intensional Agent Model for Practical Reasoning, In: Proceedings of MDAI 2009, 5–20.

Casali, A., Godo, L., Sierra, C., 2005. Graded BDI Models For Agent Architectures. In: Leite, J., Torroni, P. (Eds.), CLIMA V, Lecture Notes in Artificial Intelligence LNAI 3487, 126-143.

Morreale, V., Bonura, S., Francaviglia, G., Centineo, F., Puccio, M., Cossentino, M, 2007. Developing Intentional Systems with the PRACTIONIST Framework. In Proceedings of the 5th IEEE International conference on Industrial Informatics, 633-638.

Munroe, S. J., Luck, M., d'Inverno, M., 2003. Towards motivated-based decisions for worth goals. In: V. Marik, J., Mueller, Pechoucek (Eds.), Multi-agents systems and applications III, LNAI, 2691, 17-28.

Norman, T. N., Long, D., 1995. Goal creation in motivated agents. In: Wooldridge, M., Jennings, N.R.(Eds.), Intelligent Agents: Theories, Architecture, and Languages, LNAI, 890, 277-290.

Norman, T. N., Long, D., 1995. Alarms: Heuristics for the control of reasoning attention. In: Proceedings of the seventeenth annual conference of the cognitive science society.

Rahwan, A., Amgoud, L., 2006. An Argumentation-based Approach for Practical Reasoning. In: Proceedings of AAMAS'2006, 347-354.

Rao, A., Georgeff, M., 1995. BDI agents: From theory to practice. In: Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95), 312-319.

Rao, A., Georgeff, M., 1991. Deliberation and its role in the formation of intentions. In: Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence, 300-307.

Schut, M., Wooldridge, M., Parsons, S., 2004. The Theory and Practice of IntentionReconsideration. Journal of Experimental and Theoretical Artificial Intelligence, 16(4), 261-293.

Thangarajah, J., Padgham, L., Winikoff, M., 2003. Detecting and avoiding interference between goals in intelligent agents. In: Proceedings of AAMAS'03, 401-408.

Wooldridge, M., 1999. Intelligent Agents. In: Weiss, G. (Eds), Multiagent Systems. The MIT Press.

Wooldridge, M., Jennings, N. R., 1995. Intelligent Agents: Theory and Practice. Knowledge Engineering Review, 10(2), 115-152.