An architecture for normative reactive agents

An Architecture for Normative Reactive Agents

Guido Boella and Rossana Damiano

No Institute Given

Abstract. We present a reactive agent architecture which incorporates decision-theoretic notions to drive the deliberation and meta-deliberation process. ...

1 Introduction

The amount of research devoted to norms in Artificial Intelligence has evidenced their role in guaranteeing a general advantage for a society at a reasonable cost for the individuals, both in cooperative and non-cooperative context. The existence of implicit or explicit norms is now recognized as one of the distinctive features of social systems, including multi-agent systems.

While norms can be embodied in a normative model in the form of hard-wired constraints on individual behavior ([Moses and Tennenholtz, 1995], [Tennenholtz, 1999]), this solution drastically limits the ability of the model to mirror real-world situations in which explicit normative reasoning is required. Explicit normative reasoning, in fact, must account for the possibility that an agent decides to deliberately violate a norm, as a consequence of conflicts between norms, or between norms and individual goals. Further, the attempts to integrate normative models in agent models (see [Conte et al., 1999], [Boersen et al., 2001] among others) have pointed out that normative reasoning cannot be disjoint from the feature of reactivity which characterizes the BDI model: as norms are instantiated as a consequence of the environment dynamically changing, a norm-aware agent must actively monitor for the contextual instantiation of norms and react accordingly.

But how can the respect of norms be reconciled with the activities that an agent is currently bringing about? In this paper, we propose a model of normative reasoning that allows an agent to react to norms in dynamically changing social contexts by modifying its intentions based on utility considerations.
Consider, for example, a toy domain where a robot accomplishes simple tasks like taking object from an office to another: in order to take some mail to the office of the boss, the robot has devised a plan for getting to the boss' office, and putting the mail on the desk. However, suppose that the robot's supervisor issues a prohibition to open a certain door, by invalidating the robot's plan to get to the boss' office. Should this order - or, more precisely, the obligation it sets on the robot - affect the robot's commitment to its higher level goal to deliver mail? Of course, the higher level goal should be not affected by the prohibition. Instead, the robot should replan by keeping an eye on the respect of the obligation. And,

should the option of opening the door turn out to be the only viable one, it should even consider violating the prohibition. The model relies on the use of a *reactive agent architecture*, that provides the agent with the ability to react to the exogenous goals - and, in particular, to the goals posed by norms - by modifying its current intentions.
The reactive agent architecture is integrated with *an interactional framework*; in this framework, the utility of a norm-compliant behavior is evaluated with respect to the social environment in which the agent is situated: the agent decides whether a norm is worth respecting or not by comparing the utility it may gain from respecting it with the utility it may gain from non respecting it.

The deliberative component of the agent architecture is based on the planning paradigm proposed by [Haddawy and Hanks, 1998], which incorporates decision-theoretic notions. Given a goal and a state of the world, the planner is invoked on a partial plan (i.e. a plan in which some actions are abstract) and iteratively refines it by returning one or more plans which maximize the expected utility according to the agent's preferences, modelled by a multi-attribute utility function.

When a norm is instantiated which is relevant for the agent, the agent tries to modify its current intentions in order to devise a norm-compliant behavior. However, instead of planning from the scratch, by re-starting the planning process in order to account for the normative goal, the agent tries to perform replanning on its current plan. In the same way as the planning process, the replanning process is driven by utility considerations.

2 The agent architecture

The architecture is composed of a *deliberation module*, an *execution module*, and a *sensing module*, and relies on a *meta-deliberation* module to evaluate the need for re-deliberation, following [Wooldridge and Parsons, 1999]. The internal state of the agent is defined by its beliefs about the current world, its goals, and the intentions (plans) it has formed in order to achieve a subset of these goals. The agent's deliberation and redeliberation are based on decision-theoretic notions: the agent is driven by the overall goal of maximizing its utility based on a set of preferences which are encoded in a utility function.

The agent is situated in a dynamic environment, i.e. the world can change independently from the agent's actions, and actions can have non-deterministic effects, i.e., an action can result in a set of alternative effects. Moreover, there is no perfect correspondence between the environment actual state and the agent's representation of it.

In this architecture, intentions are dynamic, and can be modified as a result of re-deliberation: if the agent detects a significant mismatch between the initially expected and the currently expected utility brought about by a plan, the agent revises its intentions by performing re-deliberation. As a result, the agent is likely to become committed to different plans along time, each constituted of a different sequence of actions. However, while the intention to execute a certain

Fig. 1. The structure of the agent architecture. Dashed lines represent data flow, solid lines represent control flow. The grey components determine the agent's state.

plan remains the same until it is dropped or satisfied, the commitment to execute single actions evolves continuously as a consequence of both execution and re-deliberation.

In order to represent dynamic intentions, separate structures for representing plan-level commitment and action-level commitment have been introduced in the architecture. So, intentions are stored in two kind of structures: *plans*, representing goal-level commitment, and *action-executions*, representing action-level commitment.
With respect to the commitment of an agent to achieve a goal, new instances of the *plan* structure follow one another in time as a consequence of the agent's re-deliberation; at each moment, however, the agent is committed to only one plan, which constitutes its *current-plan* and remains the same from the moment it becomes the current plan to the moment is dropped.
On the contrary, the action-level commitment of an agent is recorded in a unitary instance of the *action-execution* structure, called *execution record*, whose temporal extent coincides with the duration of the agent's commitment to a goal; the *execution record* is incrementally updated at every cycle, and eventually encompasses all the actions the agent has executed in obedience to a goal, and their relation to a plan. The *execution record* is internally dynamic, as it is updated every time the agent executes an action, or revises its plan-level intentions (i.e. each time the current plan changes).

The behavior of the agent is controlled by an execution-sensing loop with a meta-level deliberation step (see figure 1). When this loop is first entered, the deliberation module is invoked on the initial goal; the goal is matched against the plan schemata contained in the library, and when a plan schema is found, it is passed to the planner for refinement. This plan becomes the agent's current intention, and the agent starts executing it. After executing each action in the plan, the sensing module monitors the effects of the action execution, and updates the

agent's representation of the world. Then, the meta-deliberation module evaluates the updated representation by means of an execution-monitoring function: if the world meets the agent's expectations, there is no need for re-deliberation, and the execution is resumed; otherwise, if the agent's intentions are not adequate anymore to the new environment, then the deliberation module is assigned the task of modifying them.

Due to the agent's uncertainty about the outcome of the plan, the initial plan is associated to an expected utility interval, but this interval may vary as the execution of the plan proceeds. More specifically, after the execution of a non-deterministic action (or a conditional action, if the agent did not know at deliberation time what conditional effect would apply), the new expected utility interval is either the same as the one that preceded the execution, or a different one. If it is different, the new upper bound of the expected utility can be the same as the previous one, or it can be higher or lower - that is, an effect which is more or less advantageous than expected has taken place.

The execution-monitoring function, which constitutes the core of the meta-deliberation module, relies on the agent's subjective expectations about the utility of a certain plan: this function computes the expected utility of the course of action constituted by the remaining plan steps in the updated representation of the world. The new expected utility is compared to the previously expected one, and the difference is calculated: replanning is performed only if the higher bound (the maximal utility) of the new expected utility interval is lower than the higher bound of the previous expected utility interval and the difference is above a certain (arbitrary) threshold.

If new deliberation is not necessary, the meta-deliberation module simply updates the execution record and releases the control to the execution module, which executes the next action. On the contrary, if new deliberation is necessary, the deliberation module is given the control and invokes its *replanning component* on the current plan with the task of finding a better plan; the functioning of the replanning component is inspired to the notion of persistence of intentions ([Bratman et al., 1988]), in that it tries to perform the most local replanning which allows the expected utility to be brought back to an acceptable difference with the previously expected one.

3 The planning algorithm

The action library is organised along two *abstraction* hierarchies. The *sequential abstraction* hierarchy is a task decomposition hierarchy: an action type in this hierarchy is a macro-operator which the planner can substitute with a sequence of (primitive or non-primitive) action types. The *specification hierarchy* is composed of abstract action types which subsume more specific ones.
In the following, for simplicity, we will refer to *sequentially abstract* actions as *complex* actions and to actions in the specification hierarchy as *abstract* actions.

Before refining a partial plan, the agent does not know which plan (or plans) - among those subsumed by that partial plan - is the most advantageous according

to its preferences. Hence, the expected utility of the abstract action is *uncertain*: it is expressed as an interval having as upper and lower bounds the expected utility of the best and the worst outcomes produced by substituting in the plan the abstract action with all the more specific actions it subsumes. This property is a key one for the planning process as it makes it possible to compare partial plans which contain abstract actions.

A plan (see section 2) is a sequence of action instances and has associated the goal the plan has been planned to achieve. A plan can be partial both in the sense that some steps are complex actions and in the sense that some are abstract actions. Each plan is associated with the derivation tree (including both abstract and complex actions) which has been built during the planning process and that will be used for driving the replanning phase.

The planning process starts from the topmost action in the hierarchy which achieves the given goal. If there is no time bound, it proceeds refining the current plan(s) by substituting complex actions with the associated decomposition and abstract actions with all the more specific actions they subsume, until it obtains a set of plans which are composed of primitive actions.

At each cycle the planning algorithm re-starts from a less partial plan: at the beginning this plan coincides with the topmost action which achieves the goal, in the subsequent refinement phases it is constituted by a sequence of actions; this feature is relevant for replanning, as it make it possible to use the planner for refining any partial plan, no matter how it has been generated.

At each refinement step, the expected utility of each plan is computed by projecting it from the current world state. Then, a *pruning* heuristic is applied by discarding the plans identified as suboptimal, i.e., plans whose expected utility upper bound is lower than the lower bound of some other plan p. The suboptimality of a plan p' with respect to p means that all possible refinements of p have an expected utility which dominates the utility of p', and, as a consequence, dominates the utility of all refinements of p': consequently, suboptimal plans can be discarded without further refining them. On the contrary, plans which have overlapping utilities need further refinement before the agent makes any choice.

4 The replanning algorithm

If a replanning phase is entered, then it means that the current plan does not reach the agent's goal, or that it reaches it with a very low utility compared with the initial expectations. But it is possible that the current plan is 'close' to a similar feasible solution, where closeness is represented by the fact that both the current solution and a new feasible one are subsumed by a common partial plan at some level of the action abstraction hierarchy.

The key idea of the replanning algorithm is then to make the current plan more partial by traversing the abstraction hierarchies in a upsidedown manner, until a more promising abstract plan is found; at that point, the planning process is restarted from the the current partial plan. The abstraction and the decomposition hierarchy play complementary roles in the algorithm: the abstraction

hierarchy determines the alternatives for substituting the actions in the plan, while the decomposition hierarchy is exploited to focus the substitution process on a portion of the plan.
A partial plan can be identified as promising based on its expected utility interval, since this interval includes not only the utility of the (unfeasible) current plan but also the utility of the new solution. So, during the replanning process, it is possible to use this estimate in order to compare the new plan with the expected utility of the more specific plan from which it has been obtained: if it is not promising it is discarded.

The starting point of the partialization process inside the plan is the first plan step whose *preconditions* do not hold, due to some event which changed the world or to some failure of the preceding actions.
In [Haddawy and Suwandi, 1994]'s planning framework the Strips-like precondition/effect relation is not accounted for: instead, an action is described as a set of conditional effects. The representation of an action includes both the action intended effects, which are obtained when its 'preconditions' hold, and the effects obtained when its 'preconditions' do not hold. For this reason, the notation of the action has been augmented with the information about the action intended effect, which makes it possible to identify its preconditions.[1]

The task of identifying the next action whose preconditions do not hold (the 'focused action') is accomplished by the *Find-focused-action* function (see the main procedure in Figure 2); *mark* is the function which sets the current focused action of the plan). Then, starting from the focused action (FA), the replanning algorithm partializes the plan, following the derivation tree associated with the plan (see the *partializes* function in Figure 2).

If the action type of the FA is directly subsumed by an abstract action type in the derivation tree, the focused action is deleted and the abstract action substitutes it in the tree frontier which constitutes the plan.
On the contrary, if FA appears in a decomposition (i.e., its father in the derivation tree is a sequentially abstract action) then two cases are possible (see the find-sibling function in 3):

1. There is some action in the plan which is a descendant of a sibling of FA in the decomposition and which has not been examined yet: this descendant of the sibling becomes the current FA. The order according to which siblings are considered reflects the assumption that it is better to replan non-executed actions, when possible: so, right siblings (from the focused action on) are given priority on left siblings.
2. All siblings in the decomposition have been already refined (i.e., no one has any descendant): all the siblings of FA and FA itself are removed from the

[1] Since it is possible that more than one condition-effect branch lead to the goal (maybe with different satisfaction degrees), different sets of preconditions can be identified by selecting the condition associated to successful effects.

```
procedure plan replan(plan p, world w)
begin
/* find the first action which will fail */
 action a := find-focused-action(p,w);
 mark a; //set a as the FA
 plan p' := p;
 plan p'' := p;
/* while a solution or the root are not found */
 while (not(achieve(p'',w, goal(p'')))
            and has-father(a))
 begin
/* look for a partial plan with better utility */
  while (not (promising(p', w, p))
         and has-father(a))
   begin
    p' := partialize(p');
    project(p',w); //evaluate the action in w
   end
/* restart planning on the partial plan */
 p'' := refine(p',w);
 end
 return p'';
end

function plan partialize(plan p)
begin
/* a is the FA of p */
action a := marked-action(p);
/* if it is subsumed by a partial action */
if (abstract(father(a)))
 begin
/* delete a from the tree */
   delete(a, p);
   return p;
 end
/* no more abstract parents: we are in a decomposition */
else
 if (complex(father(a))
  begin
   a1 := find-sibling(a,p);
   if (null(a1))
/* there is no FA in the decomposition */
     begin
      mark(father(a)) //set the FA
      //delete the decomposition
      delete(descendant(father(a)),p);
      return p;
     end
   else
    begin //change the current FA
     unmark(a);
     mark(a1);
  end
 end
end
```

Fig. 2. The main procedure of the replanning algorithm, *replan*, (above) and the procedure for making a plan more abstract, *partialize* (below).

```
function action find-sibling(a,p)
begin
/* get the next action in the plan to be refined (in the same decomposition as a) */
 action a0 := right-sibling(a,p);
 action a1 := leftmost(descendant(a0,p));
 while(not (null (a1)))
  begin
/* if it can be partialized */
   if (not complex(father(a1)))
    begin
     unmark(a); //change FA
     mark(a1)
     return a1;
   end
/* move to next action */
   a0 := right-sibling(a0,p);
   a1 := leftmost(descendant(a0,p));
 end
/* do the same on the left side of the plan */
 action a1 := left-sibling(a,p);
 action a1 := rightmost(descendant(a0,p));
 while(not (null (a1)))
  begin
   if (not complex(father(a1)))
    begin
     unmark(a);
     mark(a1)
     return a1;
   end
 action a1 := left-sibling(a,p);
end
```

Fig. 3. The procedure for finding the new focused action.

derivation tree and replaced in the plan by the complex sequential action, which becomes the current FA (see Figure 3).[2]

As discussed in the Introduction, the pruning process of the planner is applied in the refinement process executed during the replanning phase. In this way, the difficulty of finding a new solution from the current partial plan is alleviated by the fact that suboptimal alternatives are discarded before their refinement.

Beside allowing the pruning heuristic, however, the abstraction mechanism has another advantage. Remember that, by the definition of abstraction discussed in Section 2, it appears that, given a world state, the outcome of an

[2] Since an action type may occur in multiple decompositions[3], in order to understand which decomposition the action instance appears into, it is not sufficient to use the action type library, but it is necessary to use the derivation tree).

abstract action includes the outcomes of all the actions it subsumes.
Each time a plan p is partialized, the resulting plan p' has an expected utility interval that includes the utility interval of p. However p' subsumes also other plans whose outcomes are possibly different from the outcome of p. At this point, two cases are possible: either the other plans are better than p or not. In the first case, the utility of p' will have an higher higher bound with respect to p, since it includes all the outcomes of the subsumed plans. In the second case, the utility of p' will not have a higher upper bound than p. Hence, p' is not more promising than the less partial plan p.
The algorithm exploits this property (see the *promising* condition in the procedure *replan*) to decide when the iteration of the partialization step must be stopped: when a promising partial plan (i.e., a plan which subsumes better alternatives than the previous one) is reached, the partialization process ends and the refinement process is restarted on the current partial plan.

In order to illustrate how the replanning algorithm works, we will resort to a toy domain constituted by the office micro-world. In this domain, two robots, X and Y are situated in an office consisting of four rooms, and accomplish simple tasks, like taking the mail from one room to another. Differently from Y, X does not have the ability to open the door between 4 and 2.

Now consider the situation in which the robot X has the goal of getting the mail from room 2 to room 1, but wrongly believes that the door between 4 and 2 is open, and thinks that passing through it will suffice to get to room 2 (see figure 4).

Fig. 4. The plan of the agent X before replanning (left) and after replanning (right).

In order to satisfy the goal to get the mail from room 2 to room 1, X has devised a plan composed of the following steps, as represented in the first box of figure 5:

GO-X-4-2-door TAKE-MAIL-X GO-X-2-1 PUT-MAIL-X

However, during the meta-deliberation phase, after executing the step GO-Y-4-2-door, X realizes that something went wrong, and starts replanning.

Given the LDA, GET-MAIL-X, and the candidate sub-plan, the repair algorithm examines the right siblings of the focused action, without finding a revision node; then, it inspects executed actions, and finds a candidate step for substitution, GO-X-4-2-door (see the second box in figure 5).
The candidate step is replaced by the lowest abstract ancestor on the path to the LDA, GET-MAIL-X, and the revision plan thus obtained is passed to the planner. A new, alternative refinement is produced (graphically represented in the third box of figure 5 and in figure 4):

GOX-4-3 GOX-3-1 GOX-1-2 TAKE-MAILX GOX-2-1 PUT-MAILX

Finally, the execution is resumed, starting from the first action of the new plan.

[Hanks and Weld, 1995] has proposed a similar algorithm for an SNLP planner. The algorithm searches for a plan similar to known ones first by retracting refinements: i.e., actions, constraints and causal links. In order to remove the refinements in the right order, [Hanks and Weld, 1995] add to the plan an history of 'reasons' explaining why each new element has been inserted.
In a similar way, our algorithm adapts the failed plan to the new situation by retracting refinements, even if in the sense of more specific actions and decompositions. The same role played by 'reasons' is embodied in the derivation tree associated to the plan which explains the structure of the current plan and guides the partialization process.

Finally, it is worth mentioning that the replanning algorithm we propose is complete, in that it finds the solution if one exists, but it does not necessarily finds the optimal solution: the desirability of an optimal solution, in fact, is subordinated to the notions of resource-boundedness and to the persistence of intentions, which tend to privilege conservative options.

5 A model of Normative Reasoning

In the approach proposed by [Boella and Lesmo, 2001], the normative knowledge of an agent encodes the representation of the behavior of the normative authority, who is in charge of enforcing the respect of norms by means of sanctions or rewards. The decision about whether to comply with the norm or not is reduced to a rational choice, given the expected behavior of the normative agent.
The agent reasons on the alternatives constituted by respecting or non respecting a norm in terms of the reaction of the normative agent: the norm-compliant behavior has a cost but avoids the risk of a sanction, while not respecting the norm allows the agent to save resources but exposes her/him to a sanction. Alternatively, the satisfaction of a norm can be associated with a reward, whose aim is to motivate agents to respect the norm.

Fig. 5. A representation of the steps performed by the repair algorithm on the action hierarchy given X's plan. The original plan (1); a node is selected for revision (RN) ; a different instantiation of the RN, the sequence of steps composing the action GOX-4-2-long, has been chosen (3).

The **triggering condition** of a norm describes the condition in which the norm becomes relevant for the bearer, by making her obliged to bring about the content of the norm.

The existence of a norm in the agent normative knowledge is independent of the obligation it establishes for the bearer, which is contextually determined. If the current situation matches the triggering condition of a norm stored in the knowledge base of an agent (i.e., a norm of which she is bearer), the norm is instantiated, and the agent becomes obliged to respect it. Every time an agent is obliged to a norm, she forms a **normative goal** with reference to that norm, i.e., she forms the goal to comply with the norm, or, more specifically, to bring about the prescription contained in the norm. This goal is an *exogenous* goal, deriving from the obligation to respect the norm which is pending on the agent as a consequence of the triggering of the norm; it becomes an agent's goal by means of *adoption*. Again, adoption is the bridge between the agent's commitment and its social environment.
However, the adoption of a goal does not necessarily imply that the agent becomes committed to the goal. When an agent adopts a normative goal, the normative goal enters the rational deliberation process, and the agent may become committed to it and form a normative intention as a result of this process.

During the normative deliberation, the agent who is subject to the obligation to respect the norm (the bearer of the norm, according to the definition above) evaluates the reaction of the normative authority by performing a *look-ahead* step. In practice, the bearer considers the possibility that the normative agent sanctions her for violating the norm, or rewards her for respecting the norm, as prescribed in the definition of the norm itself. This process - similar to game-theoretic approaches - is carried out by means of the *anticipatory planning* technique illustrated in (). The agent computes the plans for bringing the about the normative goal, and trade them off against her current intentions from an utilitarian point of view. However, the expected utility is not evaluated on the outcome of these plans, but *in the light of the normative authority's subsequent reaction*: the agent becomes committed to the normative goal only if the corresponding plans yield a higher utility in the agent preference model.

In this way, the sanction is not an external event, but the result of the activity of the normative authority, who is an intelligent reactive agent as well: the normative authority has the goal of enforcing the respect of the norm, by detecting the violations to the norm and sanctioning them accordingly. When the agent who is subject to an obligation reasons on the utility of complying with it, she must have a model of the normative authority, that she uses to predict the reaction of normative authority. In particular, she considers:

- the *probability* that the normative authority *detects* the non-compliance to the norm.
- the *probability* that the normative authority - provided the he detects the violation of the norm - actually *issues a sanction* (or a reward).

Under certain circumstances, in fact, the agent may decide that it is not worth complying with the norm because there is a low probability that the normative authority will detect the violation, or that he will issue a sanction. Besides, an agent may try to deceive the normative authority by inducing the normative to incorrectly believe that she complied with the norm part, or by preventing the normative authority from becoming aware of the violation. Finally, an agent may violate a norm by planning to avoid the effects of the sanction in some way.

Notice that the notion of obligation is not related to a specific propositional attitude in the agent model; rather, it is embedded in the knowledge about the normative authority's sanctioning (or rewarding) reaction, which is exploited in the look-ahead step. The role played by this knowledge in the intention formation process is to promote the respect of obligations as a consequence of a rational, utility driven choice. However, this reasoning style is not incompatible, in principle, with a different characterization of the notion of obligation: an agent may as well have in its own utility function the preference for respecting the norm, in association or not with a preference for not being sanctioned.

6 Reactivity to Norms

Being situated in a social environment, an agent must be able to react to norms which are contextually triggered: a norm can be triggered by the agent's behavior itself, by a change in the environment, or else as a consequence of the behavior of another agent. Here, we are concerned with *reactivity to norms*, i.e., with the situations in which the compliance to norm must be reconciled with existing intentions; for an account of how norms filter the agent's choices in the intention formation phase itself, see [Boella and Lesmo, 2001].

An agent does not devise and evaluate a norm-compliant behavior in isolation from its current intentions. In this model, norms are treated as an exogenous and asynchronous source of goals which are submitted to the agent for deliberation. So, the evaluation of the utility of complying with a norm takes place in the context of the agent's existing intentions, which in turn continuously adapt to a dynamically evolving environment.
The agent's current commitment constitutes the background against which the agent devises a line of behavior which complies with the norm: the agent reasons on its current intentions trying to modify them in order to devise a norm-compliant line of behavior. This line of behavior is then traded off with the option of not complying with the norm, in the light of the reaction of the normative authority.

In section ..., we described an architecture for reactive agents, focussing on how the agent modifies its current intentions depending on the changes of a dynamic environment; we were not concerned, however, to reactivity to new goals. Here, we want the agent to react to events which setting up new goals, like the instantiation of norms. In order to do so, we exploit the architecture presented in section 2 to provide the agent with the capability to monitor for new goals and to modify its current intentions in order to achieve them.

Norms are stored in the agent's **normative knowledge base**; as illustrated above, the definition of a norm includes a triggering condition, which, when instantiated, gives rise to a normative goal. After the deliberation phase in the reactive agent architecture presented in section 2, the agent *monitors for normative goals*, by checking if the conditions of the norms stored in her knowledge base are verified: if one or more norms are triggered, new normative goals arise, and are adopted by the agent.

After adopting a normative goal, the agent tries to integrate its current intentions with actions for satisfying the new goal; the integration process yields a set of new plans, but the agent's commitment is not affected so far. The expected utility of the original plan and of the new plans is evaluated after performing the look-ahead step, which is carried out by exploiting the *anticipatory planning* framework), i.e. in the light of the reaction of the normative agent ([Boella et al., 2000], [Boella and Lesmo, 2000]): as a result of the utility-based trade-off between the alternatives (*preference-driven choice*), the agent may commit to a plan which complies with the normative goal.

In short, the normative behavior of an agent is generated through the following steps:

1. **Reactivity**: when the agent becomes aware of the relevance of a norm (i.e., the norm is instantiated), it devises a norm-compliant behavior by modifying its current intentions.
2. **Utility-driven evaluation**: the agent evaluates the utility of complying with the norm or not in the light of the reaction of the normative authority, by performing a form of anticipatory reasoning.
3. **Normative deliberation**: the agent decides whether to comply with the norm or not on the basis of the results of the utility-driven evaluation.

Now consider the situation in which Y has a plan to go from room 4 to room 2 by passing through a door, with the final goal to take the mail from room 2 to room 1. Since the door is initially closed, the plan includes the step to open it:

```
(UNLOCK-Y   GO-Y-4-2   TAKE-MAIL-Y   GO-Y-2-1  PUT-MAIL)
```

Now, suppose that Y is faced with the obligation to keep the door closed, (DOOR = closed), the replanning algorithm produces the following plan (initially discarded since more expensive):

```
(go-Y-4-3 go-Y-3-1 go-Y-1-2 TAKE-MAIL-Y   GO-Y-2-1  PUT-MAIL-Y)
```

i.e., in order to comply with the obligation, Y should choose an alternative way of getting from room 4 to room 2 by going through room 3.

7 Related Work and Conclusions

As it has been remarked on by ([Nebel and Koehler, 1993]), reusing existing plans raises complexity issues. They show that modifying existing plans is advantageous only under some conditions: in particular, when, as in our proposal,

it is employed in a replanning context (instead of a general plan-reuse approach to planning) in which it is crucial to retain as many steps as possible of the plan the agent is committed to. Second, when the complexity of generating plans from the scratch is hard, as in the case of the decision-theoretic planner we adopt.

In particular, it must be noticed that the replanning algorithm works in a similar way as the *iterative deepening* algorithm. At each stage, the height of the tree of the state space examined increases. The difference with the standard search algorithm is that, instead of starting the search from the tree root and stopping at a certain depth, we start from a leaf of the plan space and, at each step, we select an higher tree rooted by one of the ancestors of the leaf.
In the worst case, the order of complexity of the replanning algorithm is the same as the standard planning algorithm. However, two facts that reduce the actual work performed by the replanning algorithm must be taken into account: first, if the assumption that a feasible solution is "close" to the current plan, then the height of the tree which includes both plans is lower than the height of root of the whole state space. Second, the pruning heuristics is used to prevent the refinement of some of the intermediate plans in the search space, reducing the number of refinement runs performed.

References

[Boella et al., 2000] Boella, G., Damiano, R., and Lesmo, L. (2000). Cooperation and group utility. In Jennings, N. and Lespérance, Y., editors, *Intelligent Agents VI — Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-99, Orlando FL)*, pages 319–333.

[Boella and Lesmo, 2000] Boella, G. and Lesmo, L. (2000). Normative reactive agents. In *Proc. of Autonomous Agents 2000 Workshop on Norms and Institutions.*, Barcelona.

[Boella and Lesmo, 2001] Boella, G. and Lesmo, L. (2001). Deliberate normative agents. In *Social Order in MAS*. Kluwer.

[Boersen et al., 2001] Boersen, J., Dastani, M., Hulstijn, J., Huang, Z., and van der Torre, L. (2001). The boid architecture. In *Proc. of AGENTS '01*, Montreal, Canada.

[Bratman et al., 1988] Bratman, M. E., Israel, D. J., and Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355.

[Conte et al., 1999] Conte, R., Castelfranchi, C., and Dignum, F. (1999). Autonomous norm acceptance. In Mueller, J., editor, *Proc. of the 5th International Workshop on Agent Theories, Architectures and Languages, Paris 1998*, LNAI, Berlin. Springer.

[Haddawy and Hanks, 1998] Haddawy, P. and Hanks, S. (1998). Utility models for goal-directed, decision-theoretic planners. *Computational Intelligence*, 14:392–429.

[Haddawy and Suwandi, 1994] Haddawy, P. and Suwandi, M. (1994). Decision-theoretic refinement planning using inheritance abstraction. In *Proc. of 2nd AIPS Int. Conf.*, pages 266–271, Menlo Park, CA.

[Hanks and Weld, 1995] Hanks, S. and Weld, D. S. (1995). A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, 2:319–360.

[Moses and Tennenholtz, 1995] Moses, Y. and Tennenholtz, M. (1995). Artificial social systems. *Computers and Artificial Intelligence*, 14(6:553–562).

[Nebel and Koehler, 1993] Nebel, B. and Koehler, J. (1993). Plan modification versus plan generation: A complexity-theoretic perspective. In *Proceedings of of the 13th International Joint Conference on Artificial Intelligence*, pages 1436–1441, Chambery, France.

[Tennenholtz, 1999] Tennenholtz, M. (1999). On social constraints for rational agents. *Computational Intelligence*, 15(4).

[Wooldridge and Parsons, 1999] Wooldridge, M. and Parsons, S. (1999). Intention reconsideration reconsidered. In Müller, J., Singh, M. P., and Rao, A. S., editors, *Proc. of ATAL-98)*, volume 1555, pages 63–80. Springer-Verlag.