An Architecture for Normative Reactive Agents

Guido Boella and Rossana Damiano

Dipartimento di Informatica – Universita' di Torino, Cso Svizzera 185 Torino Italy,
{guido,rossana}@di.unito.it

Abstract. We present a reactive agent architecture which incorporates decision-theoretic notions to drive the deliberation and meta-deliberation process, and illustrate how this architecture can be exploited to model an agent who reacts to contextually instantiated norms by monitoring for norm instantiation and replanning its current intentions.

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 ([10]), 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, normative reasoning cannot be disjoint from reactivity: as norms are instantiated as a consequence of the changes in the environment, a norm-aware agent must actively monitor for the contextual instantiation of norms and react to them accordingly.

Consider, for example, a domain where a robot accomplishes simple tasks like taking objects 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 mail box and picking up the mail. However, suppose that the robot's supervisor issues a prohibition to go through a certain door, by invalidating the robot's plan to get to the mail box by taking the shortest path. Should this prohibition – or, more precisely, the obligation it sets on the robot – affect the robot's commitment to its higher level goal to deliver mail? The obvious answer is that the commitment to 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 prohibition, as disobedience would expose it to the risk of a sanction. Moreover, we argue that, should the option of opening the door turn out to be the only viable one, it should even consider violating the prohibition.

K. Kuwabara and J. Lee (Eds.): PRIMA 2002, LNAI 2413, pp. 1–17, 2002.

Notice that the situation depicted in the example is similar to a replanning problem in a dynamic environment: in that it can be compared with a situation in which the robot finds the door locked on its way to the boss' office and is forced to replan.

But how can the compliance to 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 forming norm-related intentions based on utility considerations. 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 possibly modifying its current intentions.

In the example, this corresponds to realizing that the prohibition to open the door has been instantiated, forming the goal to respect this prohibition, and eventually considering this goal for satisfaction.

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 (thus avoiding a sanction) with the utility it may gain from non respecting it.

In practice, the robot should not automatically opt for complying with the prohibition: in order to exhibit an intelligent, flexible behavior, it should first devise a line of behavior that complies with the prohibition to go through the door, and then trade it off against its original line of behavior, in the light of the utility associated with each option.

This paper is organized as follows: in the section 2, we present the reactive agent architecture the overall model relies on; in section 3 and 4, the utility-driven deliberative component is described. Then, in section 5, we introduce the model of normative reasoning that, in conjunction with the reactive agent architecture described in previous sections, yields the reactivity to norms described and exemplified in section 6.

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 [11]. 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 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, new obligations may arise, 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 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.

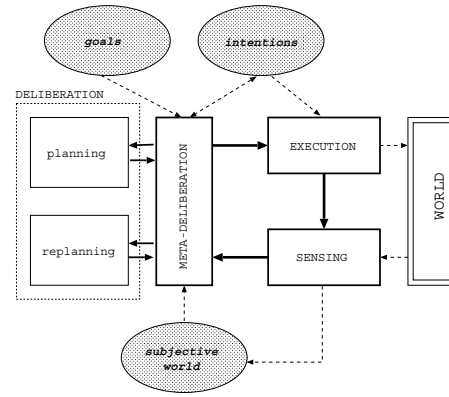

Fig. 1. The structure of the agent architecture. Dashed lines represent data flow, solid lines represent control flow. The grey components determine the agent's state.

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. New instances of the *plan* structure follow one another in time as a consequence of the agent's re-deliberation; 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 agent's commitment to a goal and which is updated at every cycle.

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. The best 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. If the agent realizes that the world has changed, the meta-deliberation module evaluates the updated representation of the world 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.

As discussed in the next section, 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. 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 there is a significant difference.

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 ([3]), 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 organized 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.

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.

Before refining a partial plan, the agent does not know which plan (or plans) – among those subsumed by that partial plan in the plan space – 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.

The planning process starts from the topmost action in the hierarchy which achieves the given goal. If there is no time bound, it proceeds re ning the current plan(s) by substituting complex actions with the associated decomposition and abstract actions with all the more speci c 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, i.e., a plan that subsumes a smaller set of alternatives in the plan space: at the beginning this plan coincides with the topmost action which achieves the goal, in the subsequent re nement 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 re ning any partial plan, no matter how it has been generated.

At each re nement step, the expected utility of each plan is computed by projecting it from the current world state; notice that, as observed above, it is possible to compute the expected utility of a partial plan, which encompasses the expected utilities of the alternatives plans it subsumes. Now, a *pruning* heuristic can be applied, by discarding the plans identi ed as suboptimal, i.e., plans whose expected utility upper bound is lower than the lower bound of some other plan p. The sub-optimality of a plan p' with respect to p means that all possible re nements of p have an expected utility which dominates the utility of p', and, as a consequence, dominates the utility of all re nements of p': consequently, suboptimal plans can be discarded without further re ning them. On the contrary, plans which have overlapping utilities need further re nement before the agent makes any choice.

4 The Replanning Algorithm

If a replanning phase is entered, 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. In the norm instantiation example introduced in the Introduction, for instance, the possibility for the robot to be sanctioned if it violates the prohibition to go through the door may decrease the utility of the current plan although it satis es the robot s original goal to deliver mail.

However, even if the utility of the current plan drops, 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 abstraction in the plan space de ned by the action hierarchy.

The key idea of the replanning algorithm is then to make the current plan more partial, until a more promising partial plan is found: at each partialization step, the current plan is replaced by a more partial plan which subsumes it in the plan space by traversing the abstraction hierarchies in a upsidedown manner, and the planning process is restarted from the the new partial plan. The abstraction and the decomposition hierarchy play complementary roles in the algorithm: the abstraction hierarchy determines the alternatives for substituting

```
procedure plan replan(plan p, world w){
/* 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)){
/* look for a partial plan with better utility */
  while (not (promising(p', w, p))
         and has-father(a)){
    p' := partialize(p');
    project(p',w); } //evaluate the action in w
/* restart planning on the partial plan */
 p'' := refine(p',w);}
 return p'';}
```

Fig. 2. The main procedure of the replanning algorithm, *replan*

the actions in the plan, while the decomposition hierarchy is exploited to focus the substitution process on a portion of the plan.

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 [7]'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 actions has been augmented with the information about the action intended effect, which makes it possible to track the motivations why it has been included int he plan, and 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 3).

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.

[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.

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

Fig. 3. The procedure for making a plan more abstract, *partialize*.

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 4):

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 derivation tree and replaced in the plan by the complex sequential action, which becomes the current FA (see Figure 4).[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 abstract action includes the outcomes of all the actions it subsumes.

[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).

```
function action find-sibling(a,p){
/* get the next action  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))){
/* if it can be partialized */
   if (not complex(father(a1))){
     unmark(a); //change FA
     mark(a1)
     return a1;}
/* move to next action */
   a0 := right-sibling(a0,p);
   a1 := leftmost(descendant(a0,p));}
/* 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))){
   if (not complex(father(a1))){
     unmark(a);
     mark(a1)
     return a1;}
 action a1 := left-sibling(a,p);}
```

Fig. 4. The procedure for finding the new focused action.

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 the domain of the office world t illustrated in fig. 5. This domain consists of four interconnected rooms, where a robot accomplishes simple tasks like moving objects and delivering mail.

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.

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 (represented in the first box of figure 6):

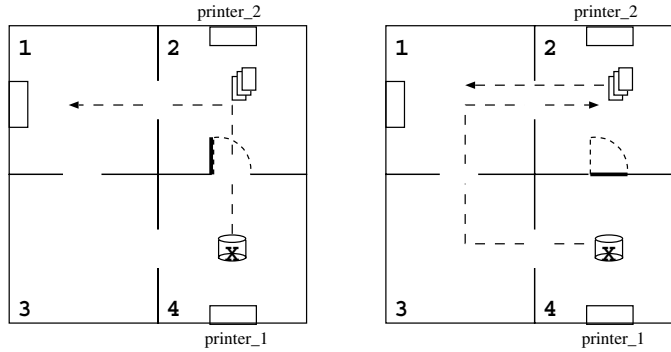

Fig. 5. The plan of the agent X before replanning (left) and after replanning (right).

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

After executing the step GO-X-4-2-door, X enters the meta-deliberation phase. At this point, X realizes that the plan has failed (it is still in room 4) because the door is locked, and starts replanning:

- First of all, the replanning algorithm identifies the focused action (FA): the focused action is the first action in the sequence of non-executed actions whose preconditions are not satisfied. In this example, TAKE-MAIL-X is marked as the Focused Action: as TAKE-MAIL-X requires the agent to be in the same room (room 2) as the mail in order to successfully perform the taking action, its preconditions are clearly not satisfied.
- Since the abstract action which subsumes the focused action (FA) in the action hierarchy, GET-MAIL-X, is a sequentially abstract action, the replanning algorithm examines the right siblings of GO-X-4-2-door, GO-X-2-1 and PUT-MAIL-X, in search of a non-elementary, abstract action to become the new FA.
  However, none of the right siblings of the currently focused action, TAKE-MAIL-X, can be marked as the new FA, as GO-X-2-1 and PUT-MAIL-X are both elementary action types.
- At this point, the only left-side sibling of the FA in the action hierarchy, GO-X-2, is examined: this action type has a descendant, GO-X-4-2-door, which is subsumed by an abstract action type (GO-X-4-2): GO-X-4-2-door becomes the new FA.
- Since GO-X-4-2-door is directly subsumed by a abstract action type, GO-X-4-2, the latter is substituted for GO-X-4-2-door in the plan frontier, and the new partial plan thus obtained is feeded to the planner for refinement.
  As it can be seen by looking at the action hierarchy depicted in figure 6, the partial plan GO-X-4-2 TAKE-MAILX GOX-2-1 PUT-MAILX is a promising one: as it subsumes an alternative refinement of the initial plan which is contextually more appropriate than the now inexecutable initial plan (GOX-4-2-long GOX-2-1 PUT-MAILX does not require the door to be open, as it

consists of taking a longer path through rooms 3 and 1), it has a higher upper utility bound.
- When the planner refines the new partial plan, it produces the following refinement of the partial plan (graphically represented in the second box of figure 6 and in figure 5): 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.

As it has been remarked on by ([9]), 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.

For what concerns the complexity issues, 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 a higher tree which 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 is true, 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.

[8] 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, [8] add to the plan an history of 'reasons' explaining why each new element has been inserted.

5 A Model of Normative Reasoning

In the approach proposed by [2], the normative knowledge of an agent, beside the content of the norm, 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-

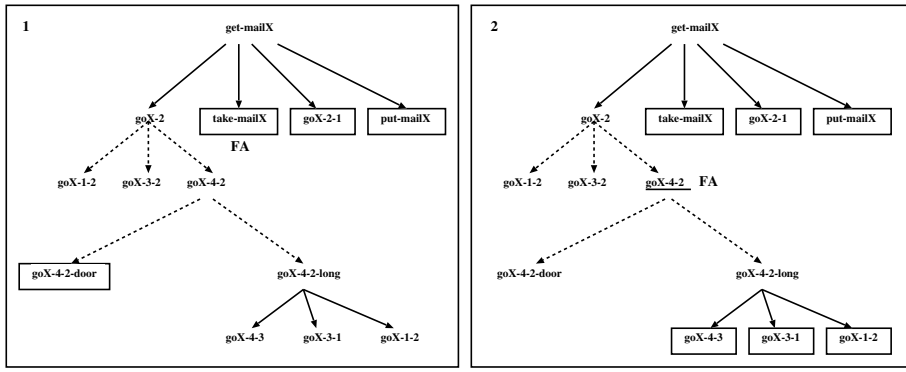

Fig. 6. A representation of the steps performed by the replanning algorithm on the action hierarchy given X's plan. The original plan (1); a new node is marked as the focused action (GO-X-4-2) and a different instantiation of the it is chosen (2), the sequence of steps composing the action GOX-4-2-long.

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 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.

The **bearer** of the norm is the agent who is *obliged* to respect the norm.
The **normative authority** is the agent who is in charge of enforcing the respect of the norm; in order to do so, he has the faculty of sanctioning or rewarding the bearer depending on her compliance to the norm.
The **sanction** (or **reward**) is an action of the normative authority, which provides the bearer with the rational motivation for respecting the norm.
The **content** of the norm is the prescription it contains; in other words, the norm establishes for the bearer the obligation to adhere to a certain behavior.
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 he is bearer), the norm is instantiated, and the agent becomes obliged to respect it. Every time an agent is obliged to a norm, he forms a **normative goal** with reference to that norm, i.e., he 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.

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 him for violating the norm, or rewards him 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 [1]. The agent computes the plans for bringing the about the normative goal, and trade them off against his 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, he must have a model of the normative authority, that he uses to predict the reaction of normative authority.

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 he 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.

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 preference for the compliance to a contextually instantiated norm must be reconciled with existing intentions;

An agent does not devise and evaluate a norm-compliant behavior in isolation from its current intentions. The agent's current commitment constitutes the background ([3]) against which the agent devises a plan which complies with the norm: the agent reasons on its current intentions trying to modify them in order to devise a norm-compliant plan. 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 this model, norms are treated as an exogenous and asynchronous source of goals which are submitted to the agent for deliberation;

at the same time, norm instances modify the utility that the different alternatives have for the agent, depending on the application of the sanction.

In section 2, we described an architecture for reactive agents, focussing on how the agent modifies its current intentions depending on how the changes of a dynamic environment affect the utility provided by his current intentions. Here, we want the agent to react to events which set up new goals and modify the utility as a consequence of a possible sanction, 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 (see the reactive agent architecture presented in Chapter 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 ([1], [2]): 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.

As illustrated above, when the triggering condition included in the definition of a norm is instantiated, it 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 his 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 modify his current intentions in order to satisfy the new goal; the replanning process yields a set of new plans, but the agent's commitment is not affected so far (see figure 7).

Norms can be classified according to their content, given the distinction between **prescriptions** and **prohibitions**; given a plan which constitutes the agent's current intention, prescriptions normally require the bearer of the norm to *add new action to the current plan* in order to bring about the normative goal, while the prohibitions, by posing constraints to the viable courses of actions, normally require that the agent *modifies the current plan.* At the same time, both normative prescriptions and prohibitions can concern **states of affairs** of **courses of action**. From the point of view of the integration with current intentions, norms referred to state of affairs require more reasoning to the agent, as the relation with courses of action is not given in the norms.

In summary, the content of a norm can be constituted by:

- The *prescription* to bring about a certain **state of affair**; in this case, the agent forms a normative goal to achieve the prescribed state of affair, without being constrained to a specified course of action. In other words, the norm does not give any instruction about how the prescribed state of affair must be produced.
- The *prescription* to execute a certain **course of action**, in order to get a certain state of affair. In this case, the focus is on the execution of the prescribed course of action.
- The *prohibition* to bring about a certain **state of affairs**. In this case, the normative goal is to avoid achieving the prohibited state of affairs. Again, the norm does not pose any constraints to the courses of action the agent may be committed to execute.
- The *prohibition* to execute a certain **course of action**.

In this paper, we focus on norms which express *prohibitions*, i.e., they concern a state which holds and must not be made false by the agent's plan, or an action which must not occur in the agent's plan[4]. Prohibitions cause *maintenance* goals to arise: differently from achievement goals ([4]), which require the agent to do something to achieve them, the agent needs not insert new steps in the plan to satisfy a maintenance goal; on the contrary, he only has to assure that the plan achieving his intentions does not make the prescribed state false.[5]

However, the expected utility of the original plan and of the new plans should be 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 ([1], [2]): 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. Since the details of planning with anticipatory coordination are described elsewhere, here we will not discuss this issue: we simply assume that the utility of the plans which violates the norm is lowered as an effect of the sanction.

The normative behavior of an agent is generated through the following steps:

1. **Reactivity**: if the agent comes to believe that the triggering condition of a norm is true, he checks whether his currently intended plan violates the prohibition (i.e. some step makes the goal false); if this is not the case, he continues his activity. In case the prohibition is violated the agent has to consider whether to replan his current plan in order to find a more profitable plan which avoids the sanction. Since a violation can be equated to a failure, the replanning algorithm described in section 4 is used: the focused action is the step which makes the prohibited goal true.

[4] For an account of how *prescription* norms lead the agent to modify his plans see [6] and how norms filter the agent's choices in the intention formation phase itself, see [2].

[5] Unless the agent is explicitly required to watch for the goal to hold, thus actively acting to prevent it from being falsified by other agents.

```
procedure agent (goal, subj-world){  /* initial planning phase */
  plan := deliberate (goal, subjective-world);
  execution := initialize-execution (plan);
  loop { /* the agent loop begins here */
    /* execute next action */
    objective-world := execute (next action);
    subj-world := monitor (next-action); /* sensing */
    /* check if goal has been achieved */
    if (execution.actions-to-execute = empty
      and achieved-goal (subj-world, goal) = T) return success;
    else{ /* The meta-deliberation phase is entered: */
      if (monitor-execution (subj-world) =‘‘replan"){
      /* the agent tries to revise its intentions */
        /* redeliberation is attempted */
        new-plan := re-deliberate (execution, subj-world, goal);
        if (new-plan)
          {/* new feasible plan found, update intentions */
          plan := new-plan;
          update-intentions (plan, execution)}
        else return failure /* no new feasible plan */
        }}}}
   if (not (Monitor-Norms = empty))   /* the agent monitors for norms */
     /* norms triggered: normative reasoning */
     plan :=Normative-Deliberation(plan, norms, subj-world)
     /* set the next action to resume execution */
     set-next-action (execution)
   }}}}
```

Fig. 7. The procedure for finding the new focused action.

2. **Utility evaluation under anticipatory coordination**: during the replanning phase under a prohibition, the agent evaluates the utility of complying with the norm or not in the light of the reaction of the normative authority, by performing the anticipatory reasoning mentioned above: plans which violate the prohibition will receive a lower utility evaluation with respect to the plans which respect it.
3. **Normative deliberation**: at the end of the replanning phase, the agent is returned with the plan which provides the best individual utility, i.e., which optimizes the trade-off between the advantage of achieving the individual goals of the agent against the disadvantage of being sanctioned.

Now consider again the replanning problem illustrated in section 4. Suppose that the supervisor of the robot has issued a prohibition to open the door between rooms 4 and 2 (see figure 5): the utility of the robot's current plan drops, as it involves violating the prohibition. In order to find a new plan which complies with the prohibition, X enters a replanning phase on its current plan (GO-X-4-2-door TAKE-MAIL-X GO-X-2-1 PUT-MAIL-X).

In parallel to the replanning example, the replanning component outputs a plan where the step of going through the door has been replaced by a sequence of steps which don't require the robot to go through the door. However, differently to the previous example, in this case the utility of the initial plan for the robot does not decrease as a consequence of a mismatch between the subjective knowledge of the agent and the world (where the plan turned out to be inexecutable), but, rather, as a consequence of the preference for not being sanctioned for violating the prohibition to open the door. Notice that, in this case, the initially focused action (FA) is GO-X-4-2-door, so the replanning algorithm begins by replacing it with GO-X-4-2.

7 Related Work and Conclusions

The model of normative reasoning and the norm-reactive architecture which is implemented in our agent allows the generation of a flexible normative behavior, in which the compliance to norms is subordinated to a rational decision process based on individual utility. The evaluation of the utility of complying to norms is accomplished within the context of the agent's current intentions and accounts for the reaction of the normative authority, thanks to the use of anticipatory reasoning. At the same time, this solution does not exclude that the agent decides to comply with the norm as a result of an existing private goal.

The work presented here shares the advantage of generating a flexible behavior with the architecture proposed by [5], where norm-compliance is filtered through the agent's goals and intentions by means of a process which makes use of individual strategies. However, in our proposal, flexible norm-compliance is obtained by means of a utility-driven comparison of the norm-compliant line of behavior with the agent's current intentions.

References

1. G. Boella, R. Damiano, and L. Lesmo. Cooperation and group utility. In N.R. Jennings and Y. Lespérance, editors, *Intelligent Agents VI*, pages 319–333. Springer, 2000.
2. G. Boella and L. Lesmo. A game theoretic approach to norms. *Cognitive Science Quarterly*, 2002.
3. M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.
4. P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.
5. R. Conte, C. Castelfranchi, and F. Dignum. Autonomous norm acceptance. In J. Mueller, editor, *Proc. of the 5th International Workshop on Agent Theories, Architectures and Languages, Paris 1998*, LNAI, Berlin, 1999. Springer.
6. Rossana Damiano. *The Role of Norms in Intelligent Reactive Agents*. Ph.d. thesis, Universitá di Torino, Torino, Italy, 2002.
7. P. Haddawy and M. Suwandi. Decision-theoretic refinement planning using inheritance abstraction. In *Proc. of 2nd AIPS Int. Conf.*, pages 266–271, Menlo Park, CA, 1994.

8. Steve Hanks and Daniel S. Weld. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, 2:319–360, 1995.
9. B. Nebel and J. Koehler. 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, 1993.
10. M. Tennenholtz. On social constraints for rational agents. *Computational Intelligence*, 15(4), 1999.
11. Mike Wooldridge and Simon Parsons. Intention reconsideration reconsidered. In Jörg Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proc. of ATAL-98)*, volume 1555, pages 63–80. Springer-Verlag, 1999.