Università degli Studi di Torino

Dottorato di Ricerca in Informatica
(*XIII ciclo*)

Ph.D. Thesis

THE ROLE OF NORMS IN INTELLIGENT REACTIVE AGENTS

Rossana Damiano

Advisor: Prof. Leonardo Lesmo

Dipartimento di Informatica — Università degli Studi di Torino
C.so Svizzera, 185 — I-10149 Torino (Italy)
http://www.di.unito.it/

Contents

Chapter 1

Introduction

1.1 The Role of Norms in Social Contexts

The necessity of constraining the freedom of action of individual agents in interactional contexts has received increasing attention, mainly as a consequence of the development of multi-agent systems.

In *cooperative* systems, interferences between the actions of the individual agents can have a negative effect on the performance of the system, thus motivating the emergence of social conventions to constrain individual activity ([Moses and Tennenholtz, 1995], [Shoham and Tennenholtz, 1997]); in *non-cooperative* contexts, the widely accepted rationale for social constraints is the aim of guaranteeing a general advantage at a reasonable individual cost ([Tennenholtz, 1999]).

As far as cooperation is concerned, theories of group activity have highlighted that, as a consequence of being part of a group, individuals are not only constrained to playing their part in the joint activity of the group, but should actively cooperate with the others by adopting their goals when it is useful for the group ([Grosz and Kraus, 1998], [Castelfranchi, 1998]). However, even in the absence of cooperative motivations, agents may still be constrained to a certain line of behavior: for example, as a result of stipulating a contract where each party commits to a certain goal or to a specific course of action.

Finally, much interest has been devoted to social aspects in conversational systems and computational models of dialogue: conversation, being a social activity by definition, is not exempt from different types of social constraints, ranging from conversational obligations ([Traum and Allen, 1994], [Poesio and Traum, 1998]) to politeness-related phenomena ([Allwood, 1994], [Bunt, 1994]).

1.2 The Role of Norms in Intelligent Reactive Agents

The common element shared by the situations depicted above is the fact of being, in a way or another, social contexts, where an agent is faced with a number of limitations prescribed by the existence of implicit or explicit social norms.

Deontic logic, the branch of modal logic which investigates the characterization of norms, has introduced different logical systems to represent norms and to reason about their properties and interactions ([von Wright, 1951], [Kanger, 1971], [Chellas, 1980], [van der Torre, 1994]). Yet, logical accounts of norms, though valuable for their insights into the nature of norms and obligations, cannot be directly integrated into agent architectures.

A variety of approaches have been proposed to integrate normative elements in agent architectures ([Dignum, 1996], [Boman, 1999], [Castelfranchi et al., 2000], [Boersen et al., 2001]); these approaches rest on the existence of a 'normative' attitude - corresponding to a deontic operator - to relate an agent's behavior to norms.

1.2.1 The proposal

Here, we propose to use a framework for social agents for modelling normative reasoning as a *rational*, *utility-driven* activity in dynamically changing social contexts. To this aim, we present a model in which an 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 model relies on the use of *a reactive agent architecture*: the agent is able to react to exogenous goals by modifying its current intentions. The use of a reactive agent architecture provides the agent with the ability to react to the goals posed by norms by devising a norm-compliant behavior.
The reactive agent architecture is integrated with *an interactional framework* for social agents; in this framework, the evaluation of the utility of a norm-compliant behavior is carried out with respect to the social environment in which the agent is situated.

From the perspective of the agent which is subject to a norm (the *bearer* of the norm), the utility of complying with the norm depends on the possibility of receiving a *sanction* or a *reward*. This sanction (or reward) is imposed on the bearer as the result of the activity of the *normative authority*, i.e. the agent who is in charge of enforcing the respect of the norm.

The activity of the normative authority is in turn modelled as a rational, utility-driven behavior: the bearer's decision about whether to comply with the norm or not is not based on the immediate effects of the norm; instead, it takes

into account the behavior of the normative authority by reasoning about its reaction to the choice of the bearer (*anticipatory reasoning*).

In practice, when a norm becomes relevant, the agent who is the bearer of the norm trades off the utility of complying with the norm with the utility of continuing with its previous activity, at risk of being sanctioned.
However, evaluating the utility of complying with a norm requires in first place that the agent devises a *norm-compliant line of behavior* and integrates it within its current intentions, i.e., with the activity it is currently committed to as a means to obtain its goals. In most cases, this integration requires a modification of the agent's intentions: depending on the content of the norm, the agent may have to reconsider the line of behavior it is committed to in order to account for new constraints imposed by the norm, or it may have to add new elements to it in order to achieve the state of affairs prescribed by the norm.

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.

1.3 The Structure of This Thesis

This thesis is organized in the following way. In Chapter 2, an overview is given of the theories which are relevant for the model we propose. Theoretical characterizations of norms and normative systems are introduced, along with architectures for deliberative normative agents. The main contributions on reactive agents, redeliberation and replanning are also briefly reviewed.

Chapter 3 contains the description of an architecture for reactive agents which incorporates decision-theoretic notions; it constitutes the core of the thesis, as the overall model builds on this architecture. The agent model and the architecture for reactive agents are described, especially focusing on the process of redeliberation: structures for supporting redeliberation are introduced, together with a new approach to replanning, illustrated by some examples.

Chapter 4 presents the interactional framework. This framework constitutes the "connective tissue" which unifies the components of the model. The anticipatory reasoning of an agent is modelled by means of a look-ahead step that

accounts for the reaction of the partner, in a decision-theoretic perspective. In order to illustrate how the interactional framework works, and to demonstrate its explanatory insights, the framework is applied to different social settings.

Chapter 5 describes a characterization of the normative knowledge which centers on the role of the normative authority. Then, it shows how this model of normative knowledge and the reactive agent model can be integrated in the interactional framework introduce in Chapter 4. Some examples are presented to demonstrate how the model supports the steps of the normative reasoning introduced in the previous section.

Finally, Chapter 6 contains the conclusion and some considerations about the future work.

The description of the domain used for examples and experiments along the chapters can be found in the Appendix A and B.

Chapter 2

Related Work

2.1 Introduction

The reflection about norms has been carried out in a variety of research areas in Artificial Intelligence, ranging from modal logics to agent theories and multi-agent distributed systems.

Multi-agent systems are mainly concerned with the characterization of normative systems and the constraints they pose for the individuals, while the generation of the normative behavior at agent level is indissolubly linked to the ability of individuals to react to norms.

In the perspective of a dynamically changing environment, an agent must be endowed with the ability to react to norms which become contextually relevant. In order to build agents who react to norms, a model of how an agent realizes that a norm has become relevant is needed, along with a model of how it adapts its behavior to norms, possibly violating them under certain circumstances.

In the following, an overview will be given of the main contributions on normative reasoning and reactive agents. This presentation, however, is not assumed to be exhaustive in any way, since it is functional to illustrating the theoretical background and the motivations for the proposal contained in this thesis.

2.2 Reasoning about Norms

Traditionally, the reflection about norms has been carried out within the realm of Deontic Logic. Deontic Logic is a branch of Modal Logic which uses modal operators to formalize notions like obligation, permission and prohibition.

Initially, the aim of this discipline was to provide formal bases and sound foundations to ethical and legal concepts. Later on, when the interest grew in AI about the notion of agency, deontic logics were exploited for modelling the normative behavior of agents in a social context. At the same time, they turned

out to be relevant to several aspects of AI and Computer Science, within the more general framework of an agent-based view of distributed, complex systems.

2.2.1 Deontic Logics

Deontic Logics constitute an instrument for formalizing the content of norms, and making inferences about the obligations they pose for the individuals. However, deontic logics are difficultly exploited as such in the direct generation of the behavior of an agent; for example, deontic systems give rise to a number of paradoxes, which are difficulty reconciled with the constraints posed by real world applications. So, instead of being directly incorporated in agent architectures, deontic logics play the fundamental role of providing the specifications for modelling the normative behavior of an agent.

In addition, according to [Horty, 1996], "Although the study of deontic logics has lead to the clarification of a number of problems involved in normative reasoning, the topic is often viewed with indifference by researchers interested in ethical theory more generally. Part of the reason for this (...) is the impression that these logics are able to model only very crude normative theories - theories that can do no more than classify situations, simply, as ideal or non-ideal". However, as shown by recent work by [Horty, 1996] and [Boersen et al., 2001] - among others, deontic theories are evolving towards more fine-grained, flexible characterizations of normative concepts.

2.2.1.1 The Standard System of Deontic Logic

The first definition of a deontic logic dates back to the work by G. Henrik Von Wright ([von Wright, 1951]). Von Wright used a primitive operator P (permission) to define the notion of obligation (O, i.e. Obliged) and prohibition (F, i.e. Forbidden):

- $Oq \equiv \neg P \neg q$
- $Fq \equiv \neg Pq$

Oq reads "it is obligatory that q is executed", while Fq reads "it is forbidden that q is executed" and q is an action.

Von Wright's work has evolved into a standard system for which a Kripke-style semantics can be used. The Standard System of Deontic Logic (SDL), as described by [Meyer and Weiringa, 1991], is a KD propositional system where a primitive operator O represents the necessity, while possibility (P) and prohibition (F) are defined by means of O. Op reads "it is obligatory that p", Pp reads "it is permissible that p" and Fp reads 'it is forbidden that p.
SDL is axiomatized in the following way:

- all tautologies of Propositional Calculus
- $O(p \rightarrow q) \rightarrow (Op \rightarrow Oq)$
  (K-axiom)
- $Op \rightarrow Pq$
  (Obligatory implies Permitted, D-axiom)
- $Pp \equiv \neg O \neg q$
  (Permission is the dual of obligation)
- $Fp \equiv \neg Pp$
  (Forbidden is not permitted)
- $p \vdash Op$
  (O-necessitation)

However, there are two main problems with using SDL to reason about the normative behavior of an agent. First of all, SDL does not allow obligations to be *contradictory* (it is a theorem of SDL that $\neg(Op \wedge O\neg p)$): while, in principle, obligations should not be contradictory, agents often are faced in real world with contradictory obligations, and so need the instruments to reason about them. In order to deal with conflicting norms, defeasible deontic logics have been proposed (see for example [van der Torre, 1994]).

In addition, the *violation* of a norm in SDL leads to an inconsistent state, which is of no use for reasoning about the violation of norms, as underlined by [Shoham, 1993a]. In order to overcome this shortcoming, formal languages have been proposed to reason about norm violation (for example, [Dignum, 1996]).

2.2.1.2 A Decision-theoretic Logical Characterization of Obligations

In this section, I will sketch a brief overview of the work by John Horty ([Horty, 1996]), which centers on the relationship between agency and obligations in a decision-theoretic framework. The purpose of this work is to define a new deontic operator for representing what an agent *ought to do* based on a preference ordering of actions adapted from decision theory.

In previous work, Horty and Belnap ([Horty and Belnap, 1995]) explored the interactions between the standard deontic operator O (*it ought to be that*) and the *stit* operator representing agency, by defining a complex deontic operator of the form *it ought to be that an agent sees to it that*. Their analysis was based on the assumption that what an agent ought to do can be identified with what it ought to be.

In [Horty, 1996], however, Horty argues that this assumption leads to incorrect results and proposes a new analysis based on a parallel between a theory of

action in indeterministic time and the theory of choice under uncertainty. The decision-theoretic notion of *preference ordering* is characterized within the logical framework of the latter: an agent *ought to see to it that A* whenever the agent has available some action which guarantees the truth of A, and is not dominated in terms of expected utility by another action which does not guarantee the truth of A.

Formally, Horty's work is based on Belnap and Perloff's modal analysis of agency ([Belnap and Perloff, 1988]), an approach which is usually described as $STIT$ semantics because it focuses on constructions of the form *an agent sees to it that*. This approach builds on the logical framework of indeterminism ([Prior, 1991]): *moments* m are ordered into tree-like structures (collection of *histories* h), with forward branching to represented the indeterminacy of the future and the absence of backward branching to represent the determinacy of the past.[1]

Given the fact that an agent performs an action at a moment/history pair m/h_m, the meaning of the $STIT$ operator is that the truth of a proposition A is guaranteed by a choice of A, where choosing is thought of as constraining the course of events to lie within some definite subset H of the available possible stories and is represented by a choice function.[2]
Formally:

$M, m/h \models [\alpha cstit : A]$ iff $M, m/h' \models A$ for all $h' \in Choice^m_\alpha(h)$

The meaning of this definition is that an agent $[\alpha cstit : A]$ at the index constituted by the moment/history pair m/h just in case α performs an action at m/h that guarantees the truth of A. Note that a statement A holds at m/h just in case α *performs* an action at m/h that guarantees the truth of A: $[\alpha cstit : A]$ does not hold if A *happens* to hold at the m/h.

The notion of *expected utility* is introduced in the framework to induce a preference ordering among actions ([Luce and Raiffa, 1957]; the notion of *weak dominance* is chosen to define the dominance relation between two actions. Each history, instead of being assigned as ideal/non-ideal, is assigned a particular *value* representing its desirability. The theory associates with each history passing through a moment a real number representing the utility of the history (*utilitarian stit frame*); so OA, given m/h, means that A is true along some history through m and then true also at every history through m of equal or greater value.[3]
In this way, what an agent ought to do at a particular moment m is determined by the way in which the histories of different value filter through the agent's $Choice^m_\alpha$ partition.

[1] $m \in h$ means that m occurs in h, or that the history h passes through the moment m.

[2] Notice that Horty follows Chellas definition of a future-oriented $STIT$ operator, *cstit* ([Chellas, 1980]).

[3] As Horty points out, this characterization of OA is a conservative generalization of the standard deontic framework.

Thanks to the introduction of utilitarian elements in the framework, Horty can now define a new complex deontic operator relating the operators *ought* and *STIT*: the new operator $\bigodot[\alpha cstit : A]$ differs from the previous one in that it refers to what an individual agent ought to see to it that, instead of adopting a universal perspective. The evaluation rule for the $\bigodot[\alpha cstit : A]$ operator is the following:

$M, m/h \models \bigodot[\alpha cstit : A]$ if and only if there is a history $h' \in H_m$ such that:

1. $M, m/h' \models [\alpha cstit : A]$
2. $Choice^m_\alpha(h'') \leq Choice^m_\alpha(h')$ for each history $h'' \in H_m$ such that $M, m/h'' \models [\alpha cstit : A]$

A consequence of this definition is that, if there is no dominance between two actions, it is impossible to say what an agent ought to see to it that.

The work by Horty constitutes an attempt to clarify the relations between theories of rational agency and deontic theories. By defining the deontic notion of *what it ought to be that* for an agent in terms of what it is rational that the agent does in an utilitarian perspective, Horty obtains a flexible but rigorous model in which obligations are relativized to individual utility, providing at the same time theoretical foundations to utility-based approaches to normative reasoning.

2.2.2 Social Norms as Protocols

In multi-agent distributed systems, the achievement of the system's goals can be endangered by the lack of coordination between the agents who compose the system. A way to solve coordination problems in multi-agent systems consists of imposing norms on the activity of the individuals, thus restricting their behavior to *socially acceptable actions* ([Moses and Tennenholtz, 1995]). The most immediate technique for enforcing the respect of social norms in a system consists of embodying them as built-in constraints on the actions that the individuals can perform.

As far as the application of normative concepts in multi-agent system is concerned, one of the first proposal is Shoham's language for Agent Oriented Programming (AOP [Shoham, 1993b]), where obligations are accounted for the agent level. However, AOP ignores the social aspect of norms, and the problems involved in normative design; these issues have been discussed into depth by [Moses and Tennenholtz, 1995], [Shoham and Tennenholtz, 1997] and [Tennenholtz, 1999].

2.2.2.1 Artificial Social Systems

[Moses and Tennenholtz, 1995] investigate the notion of *artificial social system*, and highlight the main aspects involved in designing systems of this type. The

key point of their definition of an artificial social system centers on the possibility for an agent to devise a plan to attain its goals: if it is guaranteed that such a plan exists, no matter what the others will do, the system is defined to be *social*. Then, the role of social laws is precisely to guarantee that a system is social, by mutually constraining the actions of agents in way that allows each of them to attain its goals by executing an efficient plan.

However, the task of identifying the social laws for a given system is not an easy one: if the laws are too restrictive, they will achieve the goal of making the system social only at the cost of imposing heavy constraints on the freedom of action of the individuals. For example, after the application of social laws, the plans which are still available to an agent (the *legal strategies* in [Moses and Tennenholtz, 1995]'s terminology) may be inefficient, and some goals may not be reachable anymore. Finally, the cost of computing legal strategies may be too high for certain sets of social laws.

In general, the system designer has the goal of restricting the set of achievable goals (the so-called *social goals*)[4] to a reasonable set, by excluding only the goals which are not compatible with the requirement of sociality. Once this set has been identified, the designer has to design the laws in such a way that they restrict the achievable goals to this set. The designer's task is termed the "golden mean problem", and is shown to be NP-complete.

Finally, [Moses and Tennenholtz, 1995] present a logical framework for reasoning about social systems; within this framework, they give a semantic definition of the notion of artificial social system, that provides the conceptual tools for reasoning about the properties of such systems. In particular, the framework allows expressing the notion of *social necessity*: given the current state of the world, and a set of laws, agents can reason about what they and the other agents need to do.

2.2.2.2 Designing Social Constraints

The work by [Tennenholtz, 1999] investigates more deeply the design of social constraints for rational agents in multi-agent settings, analyzed in a game-theoretic framework. Tennenholtz presents a set of algorithms for identifying a set of social constraints which share the requirement of guaranteeing to each agent participating in the game a reasonable payoff from its actions.

The algorithm proposed for zero-sum games works on the game tree representing the agents' behavior and tries to identify a subtree that constitutes the result of constraining their behavior in a useful way. A *useful* constraint is a constraint that identifies a subtree whose leaves have an acceptable payoff for the players, given an arbitrary minimum. This problem is termed UCSLP (Useful Constrained Social Law Problem); Tennenholtz presents a polynomial algorithm

[4]The use of the term *social* can be somehow misleading in this context, as it does not refer to the content of goals, but to their property of being compatible with the social system.

to solve the UCSLP problem. Then, an algorithm is proposed for the case of general games; this algorithm identifies social constraints by associating certain actions with a sanction, i.e. a negative payoff, and has the same complexity as the previous one.

Up to now, constraints are not expressed as general laws which prohibit the execution of certain action types. However, the attempt to generalize contextual prohibitions to laws which pose general constraints on actions (Useful Action-constrained Social Law Problem, or UACSLP) turns out to be NP-complete for zero-sum games.

Finally, Tennenholtz presents a technique for enforcing social laws by constraining the perceptual activities of the agents, instead of constraining their actions: this problem is modelled as a game of incomplete information, and a polynomial algorithm is presented for this technique.

The work by [Moses and Tennenholtz, 1995] and [Tennenholtz, 1999], as it focuses on the properties of social systems, it is complementary to the line of research that tries to characterize normative reasoning at agent-level (see section 2.2.3). In an integrated perspective, the two approaches should not conflict: rather, system-level design of norms establishes the parameters for evaluating the properties of agent-level characterizations of normative reasoning, while the latter provide insights on cognitive motivations underlying the design of norms.

2.2.2.3 Spontaneously Emerging Social Conventions

The recent work by [Alterman and Garland, 2000] investigates the role of conventions in joint activity and how they arise as a consequence of cooperative practices.

They show how the emergence of behavioral conventions over time in joint activity can be modelled on the basis of the memory of pragmatic actions (see also [Shoham and Tennenholtz, 1997]). In the model, the individual memory of actions is organized around *points of coordination*, which allow the agents who take part in the joint activity to converge on an ideal line of behaviors.

Several experiments are shown to demonstrate the validity of this model of emergence of conventions as a result of an historically-based predisposition: as behavioral conventions begin to emerge, the performance of the system improves, and this improvement is compared to the performance of a base-line system where classical learning techniques are employed.

This model shows how the emergence of conventions significantly reduces the amount of time and communication needed for coordinating individual behaviors - without requiring individuals to have identical mental structures - thus providing a rationale for the existence of norms.

2.2.3 Norms in Agent Architectures

Apart from the models of normative systems and of their properties, studies on normative behavior have investigated also the relations between agents and norms, and the role that norms should play in determining the behavior of agents. The advantage of explicitly incorporating normative notions in agents, instead of hard-wiring them at system or agent level, is the obtainment of a more easily configured, flexible behavior.

The current standard in agent models is represented by the belief-desire-intention cognitive model (BDI) , first introduced at the beginning of the 90s by [Cohen and Levesque, 1990a] and [Rao and Georgeff, 1991]. According to the BDI model, a rational agent is characterized by its beliefs, desires (or goals) and intentions. The interplay between these attitudes can be described in a modal logic by defining the intentions of an agent in terms of its goals and beliefs.

The logical description of a BDI agent has inspired a number of different architectures, which basically share the following intention formation mechanism: given a set of goals and its current beliefs, the agent forms a set of intentions to achieve them - on the basis of its knowledge about acting - to which it becomes committed. The commitment to a set of intentions leads the agent to the execution of the actions involved in these intentions; in other words, it constitutes the bridge between deliberation and action.

The reflection on intentionality and commitment has been strongly influenced by M. Bratman 's work ([Bratman et al., 1988], [Bratman, 1990]). In particular, Bratman has investigated the notion of intention, and has related this notion with the notion of deliberation in resource-bounded agents. Intentions are future-oriented, and have the properties of being *persistent* and *under-determined*: they are detailed out only in the proximity of the execution time. Intentions constrain the agent's future deliberation, by filtering out options which conflict with the agent's established intentions.

Given the prominence of the BDI model, several scholars have investigated how this model relates to norms and how BDI architectures can be integrated with models of normative reasoning. In the following section (2.2.3.1), we introduce a cognitive account of the relations between agents and society; in section 2.2.3.2, a brief overview is given of some of the main proposals.

2.2.3.1 Cognitive and Social Action

The work by [Conte and Castelfranchi, 1995] investigates the link between the micro-level of individual agents and the macro level constituted by social structures. The thesis of these authors is that social action is a multi-level phenomenon which emerges from the interaction of cognitive agents, i.e., agents who are characterized by being endowed with the representation of goals and the ability to pursue them. So, *cognitive social action* is an emergent phenomenon, resulting

from the relations between each agent and the external social world.

However, the individual would not be able to participate in social action if it were not endowed with a feature that connects the individual mind with the society around it. This necessary 'bridge' is provided by the process of *goal adoption*. Goal adoption consists of an agent forming the goal g that another agent obtains the goal g'; in order to prevent accidental adoption, the definition also prescribes that the adopting agent believes that the adopted goal actually is a goal of the other agent.

The authors also introduce a *goal adoption rule*, which prescribes that an agent adopts a goal only if it believes that it will achieve some other goal by doing it. The goal adoption rule states a principle of instrumentality in adoption which is the basis for the principle of normative autonomy formulated in [Conte et al., 1999]. This work describes in detail the process of formation of *normative goals and intentions* from normative beliefs, in contrast with the approach to norms as constraints.

Initially, the agent forms a normative belief, which is submitted to the recognition of the authority from which the norm emanates and of the validity of the motivation inspiring the norm. Then, once the normative belief has formed, the agent evaluates whether or not the norm is relevant to its situation: if this evaluation leads to a negative answer, no normative goal is formed.

The subsequent step consists of the formation of the normative goal itself; this is where the *principle of normative autonomy* comes on the scene: the acceptance of a norm is always instrumental to the achievement of some other goal, that the norm implies. The agent can accept a norm as a means for obtaining an (internal or external) reward, or avoiding an (external or internal) punishment, or in order to achieve some other goal which is implied by the achievement of the normative goal: this category is suitable to model a value-oriented approach to norms, according to which the acceptance of a norm obtains a consequent, natural benefit.

The formal definition of the principle of normative autonomy is the following:

$$\mathrm{BEL}_X(\mathrm{O}_{yX}(\mathrm{q}) \wedge \mathrm{INSTR}(\mathrm{OBT}_X(\mathrm{q}),\mathrm{p}) \wedge \mathrm{GOAL}_X(\mathrm{p}|\mathrm{r})) >$$

$$\mathrm{N\text{-}GOAL}_X(\mathrm{OBT}_X(\mathrm{q})|\mathrm{GOAL}_X(\mathrm{p}|\mathrm{r})\wedge\mathrm{r})$$

In short, if an agent believes that it is subject to the obligation q, and q is a means for obtaining p, and in addition it has the goal that p holds, the agent will form the normative goal of obtaining q given the goal that p and r hold.

The advantage of defining cognitive social action as an emergent phenomenon is that it does not posit absolute constraints on the behavior of agents. On the contrary, individual behavior is a consequence of the interplay between the individual mind and the social context, in a flexible way. In this model, the formation of a normative goal does not necessarily translate into the compliance to the norm it is related to. Even if the agent has formed a normative goal, the

corresponding normative intention is adopted only if a set of conditions hold: the norm is not in conflict with other normative goals or with more important goals of the agent, and the agent is in the position to achieve the normative goal.

2.2.3.2 Integrating Norms in Agent Architectures

The work by Frank Dignum ([Dignum, 1996]) constitutes an attempt to define the notions which are necessary to model autonomous agents in an environment characterized by the existence of social norms, and the relations between these notions with reference to a BDI agent architecture.

According to ([Dignum, 1996]), norms can be divided over three levels, consisting of a *private* level, a *contract* level, and a *convention* level. The private level is the level of the individual commitment, the contract level is the level at which an agent stipulates contracts with other agents, and the convention level is the level of ethical norms. Norms, and the obligations they pose on agents, are described by resorting to the deontic notions of obligation, authorization and responsibility, with a special focus on the intermediate level, the contract level.

A direct *obligation* is the obligation for an agent i towards agent j that a state of affairs holds or that an action is performed. Obligations of this kind usually originate from the stipulation of contracts between agents, which determine a situation of power relations, authorizations, and responsibilities between the contractors. The notion of obligation implies the conditional authorization for agent j to sanction agent i if the obligation is not fulfilled.
The notion of *authorization* corresponds to permission in deontic logics and is related to roles and power relations. An authorization gives the authorized agent i the permission to perform a certain act, for example, a directive speech act addressed to another agent j. By means of authorizations, it is possible to model the notions of role and authority in agent societies; they can be enforced by encoding them as built-in features of agents, or as a consequence of role-assignment in a group.
The notion of *responsibility* is invoked in case of failures: if the system experiences a failure, this involves the obligation, for the agent who was responsible for the action which caused the failure, to repair the situation.

The deontic notions described so far are exploited to define norms at the contract level; however, [Dignum, 1996] also describes the relations among normative levels (private, contract, and convention) and how they contribute to the formation of the commitment in the reference agent architecture.
At the convention level, *prima facie* norms determine the ethical background against which the interactions and negotiations between agents take place. Prima facie norms concern for example cooperation, moral laws, etc, and can be described by using the deontic operators for prohibition, permission and obligations. No indication is given about how these norms should be enforced; however, it is suggested that instead of leading to an inconsistent state, the violation of norms

should lead to a sanction, in contrast with the limitations imposed by the use of a standard deontic logic.
The private level is the level at which an agent forms its commitment, given a set of goals which characterize it. In order to account for the role of norms in the generation of commitment, norms translate into preferences, which determine a goal ordering. Goals are ordered according to their partial fulfillment of preferences; this ordering can be encoded in a utility function. Then, when the agent selects a goal for achievement, it generates a plan for achieving this goal, and the plan is added to an agenda. The plans contained in the agenda constitute the set of intentions the agent is committed to; they can be further ordered according to their relations with norms.

The model by Dignum, although it especially focuses its attention on the obligations deriving from contracts between agents, shows how a set of basic deontic notions can be exploited to generate the behavior of an agent at different levels, ranging from ethical, private level to contractual level, and how these levels can be integrated into an overall architecture by using a language that allows manipulating deontic concepts.

[Castelfranchi et al., 2000] describe an agent architecture which supports an intelligent and flexible behavior with respect to norms, inspired to [Conte and Castelfranchi, 1995]. Here, the focus is on the norms of a society, rather than on the obligations which are established between individuals by the stipulation of contracts.

Flexibility is obtained by allowing the agent to reason about whether to obey to a norm or not. The first step to the obedience to a norm is the formation of a normative belief (for example, as a consequence of a communicative act by some authority); then, if the agent recognizes the norm as concerning itself, it forms a normative goal which influences its behavior as a *meta-level* goal.

Beside constituting a source of goals, norms also influence the selection of existing goals. The process of reasoning about the obedience to a norm is accomplished by a Norm Management component situated at a meta-level in the agent architecture. The task of the Norm Manager is to determine which norms are to be adopted and how they will influence the agent's behavior (whether they will be respected or rejected); the way normative meta-goals influence the agent's behavior, then, is regulated by the existence of *strategies*, which determine the generation and the selection of goals by the Goal Management component and the plan formation by the Plan Management component.
In this way, norms not only determine the formation of normative meta-goals, but also enter the goal generation and selection phases. The two phases are kept distinct, so that an agent can form a normative goal, as a consequence of recognizing that a norm concerns it, and subsequently reject it by assigning a very low priority to the norm-related goal during the goal selection phase; this is also the level at which conflicts between normative goals are resolved.

As far as the planning activity is concerned, the existence of a normative meta-goal can enter both the plan generation and the plan selection process.

The architecture proposed by [Castelfranchi et al., 2000] supports flexibility in the generation of normative behavior, by filtering the obedience to norms through the agent's goals and intentions.
With respect to [Conte and Castelfranchi, 1995], they introduce the notion of strategy as a further source of individual variability in the generation of the response to norms.

2.2.3.3 Other Approaches

The approach by [Boman, 1999] is similar to [Moses and Tennenholtz, 1995] in that norms are conceived of as constraints. However, Boman focuses on how to implement the constraining role of norms in the rational activity of agents, and proposes a method for enforcing the respect of norms within an agent architecture. The behavior of agents is driven by *supersoft decision theory*, i.e., a variant of classical decision theory where assessments are represented by vague and imprecise statements.
In the model proposed by [Boman, 1999], agents receive normative advice by a decision module. The functioning of this decision module can be constrained at three different levels of abstraction. At the lowest level, utilities are manipulated to induce a positive or negative assessment about some consequences. At the intermediate level, actions are filtered according to the agent's risk profile; this level is the most appropriate for enforcing coordination policies in a group. The highest level is the level where social norms are accepted; this is obtained by disqualifying the actions which have a negative impact on the utility of a group.

A compromise between norms as a means for controlling the agents' behavior and a dynamic view where norms can be violated is constituted by the work by [T. Statulat and Enjalbert, 2001]. They propose a first-order model for normative agent systems in which norms are temporally dynamic, in the sense that they have a lifetime and the prescription they contain is also constrained to a time interval. The violation of norms can be computed in the model given the current and past execution of the system and the temporally dynamic description of norms.

Finally, [Boersen et al., 2001] present an approach where normative obligations are represented as a separate modality. They introduce a Belief Obligation Intention Desire (BOID) architecture that accounts for the resolution of conflicts between these attitudes. Thanks to the introduction of a feedback loop in the intention formation process, the agent can consider the effects of its actions before committing to them, detecting the conflicts which may arise between its beliefs, obligations, desires and intentions.
In this model, the priority between conflict types is determined by the deriva-

tion order of the extensions of the agent attitudes. Different derivation orders correspond to agents characterized by different sociality levels. For example, if obligations are derived before desires, a super-social agent is obtained.

Beside the normative models of multi-agent systems, it is worth mentioning the notion of *social responsibility* as an alternative means for improving the group performance in group cooperation ([Hogg and Jennings, 2000a]): according to [Hogg and Jennings, 2000b], this notion can be modelled as a component of the utility function of agents, giving rise to a trade-off between the benefit for the individual and the benefit for the group.

2.3 Reactive Agent Architectures

2.3.1 Reconsidering Intentions

The work by [Wooldridge and Parsons, 1999] has pointed out a major limitation of the BDI agent model; as it is, the model does not contain any prescription concerning when the agent should act and when it should deliberate. In order to overcome this limitation, Wooldrigde and Parsons propose a variant of the BDI model where the agent is equipped with a meta-level function which allows it to choose between action and deliberation.

Formally, the agent is a 5-tuple (M, D, A, N, l_i), where M is a *meta-level control function*, D is a *deliberation function*, A is an *action function*, and N is a *next-state function*. The next state function works as a belief-revision procedure which updates the agent's beliefs as a result of the perception of the environment (E).

Given a local state of the agent l_i (defined by its beliefs and intentions) the meta-level control function determines whether it is necessary to act or deliberate: in the first case, the agent resorts to its action function, which selects the most appropriate action; in the second case, the deliberation function is applied, and the new intentions of the agent are determined.

Some important assumptions concern the environment, which is assumed to be static, and deterministic. The first assumption means that the environment changes only as an effect of the agent's action, while the second means that there is no uncertainty about the effect of performing a certain action.

Given the definition of agent and environment, the authors define the notion of *run*, i.e., the sequence of agent states induced by the agent meta-level function given an initial environment. Each run is associated with a payoff, that expresses the utility it yields to the agent. The association of a utility measure with a run provides the basis for the definition of the optimality of an agent: it is shown that the agent's optimality with respect to an environment implies the mutual optimality of its components.

The authors focus their attention on real-time environment, where the utility is strictly related to the trade-off between the amount of time spent deliberating and the amount of time spent acting. On the one side, the agent should not act in obedience to intentions which are not appropriate to the environment anymore; on the other side, it should not waste time deliberating when it is not necessary. For real time environment, it is demonstrated that the optimality of an agent implies the soundness and completeness of its meta-level function with respect to its deliberation component.[5]

The agent architecture presented in [Parsons et al., 1999] and [Parsons et al., 2000] constitutes an application to robotics of Wooldrige and Parson's work on meta-level deliberation. Here, the action component (with respect to the agent definition as a 5-tuple illustrated above) is provided by the Thinking Cap (TC, [Saffiotti, 1998]), a system for autonomous robot navigation based on fuzzy logic. Given the map of the environment, the TC takes as input a navigation goal, and its planning component devises a plan (called B-plan) that achieves the navigation goal; a monitoring component periodically evaluates the appropriateness of the current plan. The adequacy of the navigation plan, and the achievement of the navigation goal consist of continuous values in the interval [0,1] representing, respectively, the degree of goal satisfaction and the degree of plan adequacy.

The goal satisfaction and plan adequacy degrees constitute a key point of the integration with the higher-level BDI component: based on these values, the meta-level control function decides when to deliberate, and the deliberative component uses these values to deliberate. The responsibilities are partitioned between the BDI component and the fuzzy navigation component: the former deliberates about the navigation plan, while the latter deliberates about the navigation goal, given a set of hierarchically organized high-level intentions.

In short, if the degree of adequacy of the navigation plan drops below a given threshold, showing that the plan is not viable anymore, or the goal satisfaction value shows that the (fuzzy) navigation goal has been reached, the meta-level function gives the control to the deliberative component to generate a new set of intentions. Each time the deliberative component outputs a new goal, the TC navigation planner devises a new plan to achieve this goal. The effectiveness of this architecture is demonstrated by a set of experiments in an industrial setting.

Beside the reactive agent model by [Parsons et al., 2000] summarized above, other architectures for reactive agents have been proposed.

For example, [Haigh and Veloso, 1996] present an architecture for dealing with asynchronous user request. Their architecture incorporates a planner that is able to create plans for multiple interacting goals introduced by the user requests. Given a set of goals, which include the requested user goals, and the

[5]Notice that the agent's meta-level function is an example of first-order reasoning, while the reasoning about the meta-level function can be captured as a second-order reasoning.

current plan, a backward-chaining algorithm tries to add operators to the current plan to achieve the pending goals, and to satisfy the preconditions of non enabled operators.
The goal selection and operator selection are driven by a set of control rules which are based on the prioritization of goals. The planning process for a task can be suspended to give precedence to another task, and then can be resumed; in this way, the system can react to exogenous events.

2.3.2 Evaluating New Options

The architecture for reactive agents presented above are mainly concerned with deliberation as a reaction to changing beliefs about the environment - the agent's high level goals remaining the same. However, agent architectures should be able to account also for the possibility that new options (i.e., new goals) arise and are evaluated by the agent in the context of its actual intentions.

The importance of evaluating new options against the background of current intentions has been underlined by [Horty and Pollack, 2001]. Horty and Pollack have developed a model where the utility of new options is not evaluated by itself, but *within the context of current intentions*. The motivation for doing so is that a new option may positively or negative interfere with the agent's current intentions in a way which is not immediately apparent.

In order to know the actual cost of a new option, the model prescribes that the agent performs an evaluation step of this option within the context of the intentions it is currently committed to. This process corresponds to the property of intentions of acting as a filter of admissibility, discouraging the consideration of options which are incompatible with existing intentions, in line with Bratman's work.

The new option and the current intentions are both modelled as plans. The actual cost of the new option is its *marginal cost*, i.e., the difference between the cost of carrying out the current plan along with the new plan and the cost of the current plan alone. The marginal cost of the new option may differ from its cost in isolation as an effect of the interferences mentioned above: so, the contextual cost of an option is computed after performing a merging process of the new option with the current plan: the more steps can be merged, the lower the marginal cost of the option.

In order to perform the contextual evaluation of a plan, Pollack and Horty present an algorithm that executes the reasoning sketched above. This algorithm is an any-time algorithm and is based on the progressive refinement of the cost esteem for the execution of the current plan and the execution of the new plan along with the current plan, given the possible ways of merging the two plans.

2.4 Replanning and Plan Adaptation

Beside redeliberation, another way to adapt current intentions to a changing context is constituted by replanning and plan-repair. The research in these two fields has resulted in a variety of approaches; however, these approaches are hardly generalized in a comprehensive theory due to the fact that they are strongly influenced by the planning technique they assume as the starting point.
Here, I will mention some approaches that are relevant for this thesis, and which range from proper replanning ([Wilkins et al., 1994], [van der Krogt et al., 2000]) to plan adaptation ([Hanks and Weld, 1995]).

2.4.1 Replanning

The work by [Wilkins et al., 1994] concerns replanning in the framework of the Cypress system, a system for creating agent-based applications. Cypress agents are reactive agents designed to operate in dynamic and uncertain environments: when a plan fails, its execution does not abort in favor of a completely new plan, but is modified so that the execution of the plan threads unaffected by the failure can continue, following a *transformational* approach. "As events render some current activities obsolete, the agents should be able to modify their plans while continuing activities unaffected by those events" (p.1). However, it is important to underline that in this framework agents do not reconsider their high-level intentions.

The replanner receives as input the current plan, the execution front, i.e., the list of last successfully executed nodes, the goal and the updated world state. The replanner is provided by SIPE-2 partial-order generative planning system ([Wilkins, 1992]). It simulates the execution of the remaining plan portion from the execution front, then compares the expected world state with the actual state: the formulae which are not expected to be true are collected and the planner determines how they affect the failed plan. Finally, the plan is modified by satisfying new goals and eliminating unnecessary subparts.
As the plan is composed of parallel execution threads, the execution of those threads which are not affected by replanning can continue while the plan is being repaired. In this way, execution and repair proceed in an asynchronous way, with a gain in efficiency.

[van der Krogt et al., 2000] present a method for replanning an agent's actions in a dynamic environment; this method adopts a *plan adaptation* approach. The plan is a represented by using a formalism based on skills and resources and the replanning process focuses on the availability of resources. Instead of using action operators, the definition of a plan centers on the notion of *skill*: a skill is a representation of an agent's capability to do an action that consumes a set of resources (its preconditions in the standard terminology) and produces a set of

resources (or effects). The plan is a Direct Acyclic Graph consisting of two set of nodes; one set represents the skill nodes, the other represents the resource nodes.

The are various reasons why a plan could become obsolete. For example, a goal may be dropped (*goal removal*); this could happen for external reasons, or, in a group cooperation, because another agent has taken care of it. Alternatively, the agent may receive a request to satisfy a new goal and adopt it (*goal addition*).

The replanning algorithm proposed by [van der Krogt et al., 2000] can account for both goal removal and addition. Goal removal is accomplished by eliminating unnecessary skills, while goal addition is accomplished by making use of two specialized algorithms: the *Howto* algorithm returns the plan fragment (or *superskill*) that is necessary to create a missing resource in a planning graph, the *Overlap* algorithm is concerned with inserting a superskill in a planning graph.
The Howto algorithm retrieves a superskill from a library, and possibly extends it (by making all the resources it needs available) to adapt it to the current plan. Then, the Overlap algorithm examines the existing plan graph and returns the set of skills that can be eliminated from it as an effect of the merge with the superskill.

Normally, several "fixes" are possible as the algorithm proceeds towards the final solution: to select one of them, the algorithm uses a priority queue. The priority of a solution consists of an estimate of its cost, given by the weighted sum of three elements: skills to be added, changes to be made to the current plan, and missing resources.
The algorithm is a greedy one: each time a resource must be produced, the first partial solution in the queue is selected and an extended solution is computed and added to the queue; at this point, the process restarts by selecting the queue front element.

The advantage of this replanning technique lies in its flexibility: the same approach used here to merge the plan fragment with the plan to be repaired can be generalized to other situations, like for example multi-agent planning (see [Weerdt et al., 2000]).

2.4.2 Plan Adaptation

The work by [Hanks and Weld, 1995], although it is not specifically intended for replanning, is similar in spirit. *Plan adaptation* ([Hammond, 1987], [Alterman, 1988]) is proposed as a way to modify a plan to adapt it to a new situation, no matter if it is employed as a planning technique or as a repair technique. They propose a plan modification algorithm, Systematic Plan Adapter (SPA), that is based on a search in the space of partial plans, and partly relies on the use of the SNLP partial-order, constraint posting, least-commitment generative planner. It is guaranteed to be complete and non-redundant, i.e., it finds a solution to the adaptation problem, if it exists, and does it by searching the plan space in a systematic way.

The SNLP generative planner takes as input a null plan and gradually refines it by choosing a flaw and undertaking an action (adding new steps, causal links or constraints) to fix it. When a plan is generated, it is annotated with a *reason* data structure that records how and why that plan was refined.

The plan adaptation process is twofold: first, a plan is retrieved from a plan library, and *adjusted* to match the current problem; then, planning constraints are retracted from the plan, and the adaptive planner is invoked on it. Given the tree of partial plans, adaptive planning may begin at the root of the tree, at one of its internal nodes, or at a solution, depending on the retrieved plan.

Basically, the process of search in the plan space works in the following way: given a node in the plan tree, if the solution is located in the subtree rooted in it, it is found by means of plan refinement. However, if the solution is to be found in a different subtree, it is necessary to apply a *plan retraction* procedure.
The plan retraction algorithm selects a plan on the plan tree horizon and removes choices made by the generation process when the plan was generated (i.e., it removes a causal link, or a set of constraints), using the reason structure associated with it. The removal of elements is based on the *reconsideration* of decisions that are not valid anymore in the new situation. The retracted plan is replaced on the horizon with the parent plan, together with its siblings, which constitute the alternative refinements of the father plan.

The adaptation loop works on the *plan frontier*, a set of plans tagged for refinement, retraction, or both. Initially, the frontier consists of the initial plan. At each iteration, the algorithm picks up a plan from the frontier, deletes it from the frontier and either refines or retracts it depending on its tag. Refinement produces new plans, that are added to the frontier and tagged for further refinement. Retraction results in the plan siblings being added to the frontier and tagged for refinement, and in the parent plan being added to the frontier and assigned a retraction tag.

Chapter 3

A Reactive Agent Architecture

3.1 Introduction

This chapter introduces a reactive agent architecture based on a BDI agent model integrated with a meta-level deliberation function ([Wooldridge and Parsons, 1999] and [Parsons et al., 1999]).

According to the BDI model, an agent is characterized by a set of beliefs about the world, a set of intentions, and a set of goals (or desires): in brief, what the agent does, given a goal, is forming a commitment to certain intentions, and acting accordingly ([Cohen and Levesque, 1990a], [Rao and Georgeff, 1991]). The agent's main activities consist in *deliberating*, i.e., choosing what intentions to commit to, and *acting*: deliberation necessarily precedes action, which depends on it, but the result of deliberation can be subsequently revised as the agent's beliefs evolve, with an obvious impact on action.

At every moment, the agent is faced with a meta-level choice between two options: it can deliberate by reconsidering its current intentions, or can go on with its current activity according to the content of these intentions. In order to allow the agent to choose between these options on a rational basis, [Wooldridge and Parsons, 1999] propose to integrate the BDI model with a meta-deliberation function, that, given the agent's information state (its beliefs about the world) and its intention, decides whether it is more appropriate to deliberate or to act. If we consider an agent with limited resources, this problem is not trivial: reconsidering one's intentions has a cost and does not advance one's activity, but acting on the basis of inappropriate intentions results in a waste of time and resources.

The architecture presented here 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. 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) is 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.

We don't consider the problem of goal formation and selection; rather, we assume that a subset of the agent's goals have already been selected on the basis of the agent's preferences, or that some external goals have been imposed to the agent's consideration, or both. Rather, we focus on how the agent forms the intentions to achieve the set of input goals and execute the actions prescribed by these intentions, by interleaving execution and deliberation phases to react to a dynamic environment. This process continues until the input goals are achieved or the dropping conditions hold (according to the definition given by [Cohen and Levesque, 1990a]).

In section 5.4.1, we will examine some circumstances in which new goals are formed and evaluated within the context of existing intentions.

3.2 The Agent Model

The agent model is constituted by the representation of the agent's internal state (i.e., its beliefs, goals and intentions) and by a set of modules that determine the agent's behavior by manipulating this state: the deliberation module, the sensing module, and the execution module, as sketched above. In the following, we describe in detail how the epistemic attitudes are represented in the internal state which characterizes the agent.

3.2.1 Knowledge about the World

The representation of the world in the agent model is maintained updated by the *sensing module* and constitutes the agent's beliefs about the external world; the latter is the world where actions are executed,[1] and is given a separate representation that the agent can access only by a sensing act. A subjective, internal representation and an objective, external representation have been introduced in the architecture in order to perform experiments about the consequences of non-determinism and uncertainty on the agent's behavior.

Differently from [Wooldridge and Parsons, 1999], the agent's environment is not assumed to be static or deterministic. 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

[1] In the current implementation, actions are executed in the external world by the *execution module* by sending messages to a simulator which is external to the agent model. Although the external world is constituted by an arbitrary representation of the world maintained by the simulator, the agent architecture definition is independent from the choice of using a simulator instead of actuators which execute actions in the real world.

of alternative effects. Not only the environment is dynamic and actions are non-deterministic, but there is no perfect correspondence between the environment actual state and the agent's representation of it. Even assuming that the world is static and actions are deterministic, the agent's beliefs about the environment can still be erroneous or uncertain: in the first case, the agent has wrong beliefs about the world, while in the second case, the agent knows that the current world state is one in a set of world states, without knowing exactly which one it is. Uncertainty poses problems for the goal-directed behavior of the agent, setting the demand for redeliberation from time to time. If the agent's monitoring reveals a mismatch between the agent's beliefs and the environment, the agent may modify its current intentions or even drop its goals.

In general, the representation of a world is constituted by a set of attribute-value pairs. In the objective representation of the world, all attributes have a certain value, while the subjective representation of the world is more complex. In the subjective representation of the world, an attribute can have a probability distribution on its values. At the notational level, this is represented by listing all the alternative attribute-value pairs and attaching a probability value to each pair. If an attribute has a unique value, its attribute-value pair list is composed by only one attribute-value pair, with a probability of 1 attached; otherwise, for each attribute, the probabilities associated to the attribute-value pairs sum to 1.

Associating attribute-value pairs with a probabilistic value is a way to represent a set of alternative worlds in a compact fashion: a probability distribution on the value of one or more attributes determines a probability distribution on a set of worlds. An assumption of independence of the attributes from each other is implicit in this representation: the set of alternative worlds is obtained by combining in all possible ways the alternative values of the attribute Then, for each world in this set, the overall probability of the world can be obtained by multiplying the probability of the attribute-value pairs which compose it.

In order to represent interdependent attributes, the subjective world representation can be constituted by a probability distribution on alternative worlds, where interdependent attributes have alternative patterns of related values. For each world, then, the afore-mentioned independency assumption between attributes holds, so that each world may in turn correspond to a set of alternative worlds, depending on the probability distribution on the value of its attributes.

Finally, the agent can be totally uncertain about the value of an attribute: to represent this, the value of that attribute is expressed as an interval, without any probabilistic information attached to the single values in the interval, apart from the overall probability of the attribute-value pair. In the extreme, the interval of values coincides with the range of values that an attribute can take. If the uncertainty cannot be represented by a range (the agent is uncertain between two specific values or two value intervals), the uncertainty can be expressed by a probability distribution on worlds: in each world, the attribute takes one of

```
(objective-world
   :name         o-world
   :attributes    ( at-X = room-4)
                  ( at-Y = room-3)
                  ( has-load-Y = F)
                  ( has-load-X = F)
                  ( time = 0)
                  ( fuel = (100))
                  ( door = open)
                  ( at-mail = room-2)
)

(subjective-world
   :name         s-world
   :attributes   (( at-X = room-4) prob = 1)
                 (( at-Y = room-4) prob = 1)
                 (( has-load-X = F) prob = 1)
                 (( has-load-Y = F) prob = 1)
                 (( time = 0) prob = 1)
                 (( fuel = 80 - 100) prob = 1)
                 ((( door = closed) prob = 2/5)
                  (( door = open) prob = 3/5))
                 (( at-mail = room-2) prob = 1)
   :world-prob   1
)
```

Figure 3.1: The objective representation of the world in the office domain (above) and its subjective counterpart (below)

the alternative values, and the probabilistic value associated to the world is constituted by the interval of values from 0 to 1, corresponding to the complete uncertainty about the probability that the world is the objective world.

For instance, consider the figure 3.1, representing two world definitions in the office micro-world, a toy domain that will be used in the examples in the following exposition. In this domain, two robots, X and Y are situated in an office constituted by four rooms, and accomplish simple tasks, like taking the mail from one room to another (see the graphical representation of the domain in figure 3.2). The world is defined by attributes indicating the position of the two robots (`at-X` and `at-Y`) and whether they have a load or not (`has-load-Y` and `has-load-X`), the state of resources (`time` and `fuel`), and finally the state of the environment, namely whether the door between room 2 and 4 is open or not (`door`) and in what room the mail is situated (`at-mail`).

The first world in figure 3.1, `o-world`, constitutes the objective representation of a world; the execution of actions takes place in this world. The second world, `s-world`, may constitute an agent's subjective view of the same world, that the

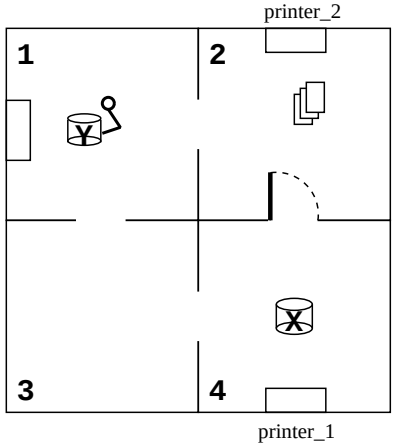

Figure 3.2: A graphical representation of the office micro-world.

agent refers to for deliberation. Notice that `s-world` contains an attribute (`door`) whose value is constituted by a probability distribution on alternative values: the door is closed with a probability of 40 % and open with a probability of 60 %. In other words, this world is a compact representation for two worlds, one, (`o-world'`), with a probability of 40 % associated, where the door is closed, and one, (`o-world''`) with a probability of 60% associated, where the door is open. Notice also that the value of the attribute `fuel` in `o-world` is constituted by a range, meaning that the agent in both `o-world'` and `o-world''` does not know the exact value of the attribute in that range.

3.2.2 Knowledge about Actions

The agent's knowledge about actions is constituted by a library of plan schemata, or recipes, each constituted by a hierarchically organized set of actions types (see figure 3.3 for an example of plan schema). Each hierarchy is a tree, in which nodes are constituted by elementary and non-elementary action types, and links represent abstraction and decomposition relations between actions.[2] Elementary action types are the leaves of the tree, and can be directly executed[3], while non elementary action types can be *abstract* or *decomposable* (or *complex*): abstract action types can be specialized into a set of alternative instantiations, decompos-

[2]Notice, however, that even if a plan schemata are trees, that does not prevent action types from appearing in more than one plan schema.

[3]The characterization of primitive actions as directly executable actions is somehow misleading: here, we are not concerned with how the actual execution is carried out by the agent and the execution module is an abstraction for real actuators. In a layered architecture, primitive actions could merely represent the lowest abstraction level for planning, without being directly executable. A reactive execution model could then be in charge of carrying out the actual execution of primitive actions in the most appropriate way.

able actions can be decomposed into a sequence of steps.
The abstraction hierarchy is obtained by *inheritance abstraction*, a technique for grouping together conditional probabilistic action operators in abstract classes based on their outcomes. ([Haddawy and Suwandi, 1994]).
The decomposition hierarchy is obtained by sequential abstraction, i.e., by gathering stereotypical sequences of action types into complex action types, in the tradition of hierarchical planning [Sacerdoti, 1977].

The actions which constitute the decomposition of a complex action are normally related to each other by enablement relations; an action *enables* another action when the execution of the first sets the conditions for the execution of the latter, by making its preconditions true. In practice, enabling the preconditions of another action constitutes in most cases the reason why an action is in the sequence. However, the use of sequential abstraction for representing actions does not explicitly encode this information, differently from other action representation techniques (see section (3.4) for a discussion of the interdependencies between actions). In order to bridge this gap, the representation of action types in this model contains the explicit information about the intended effect of an action, that we name "action goal": the intended effect is the effect an action has been included in the recipe for, differently from the the side-effects.

In the action representation, the decomposition and abstraction hierarchies are conflated into one abstraction/dcomposition hierarchy, where action types are represented in a uniform way. The representation of an action consists of:

- the **action name**;
- the set of **relevant attributes**, i.e., the attributes which appear in the definition of the action effects (see the definition of conditional effects below);
- the set of **conditional effects**.
  Conditional effects are constituted by a set of conditions, each corresponding to a set of effects. Conditions are logical expressions concerning the state of the world the action applies to; effects determine the state of the world after the execution of the action, and are expressed as attribute-value pairs. Effects can be non-deterministic, i.e., an action can have alternative effects, with a probability distribution on them.
- the list of sub-actions, which is constituted by a list of **more specific actions**, if the action is an abstract action, or by a list of actions which constitute its **decomposition**, if the action is a complex action. If the action is a primitive action, it does not have any sub-actions.
- the **action-goal**, i.e. the action intended effect (one in the list of the effects of the action).

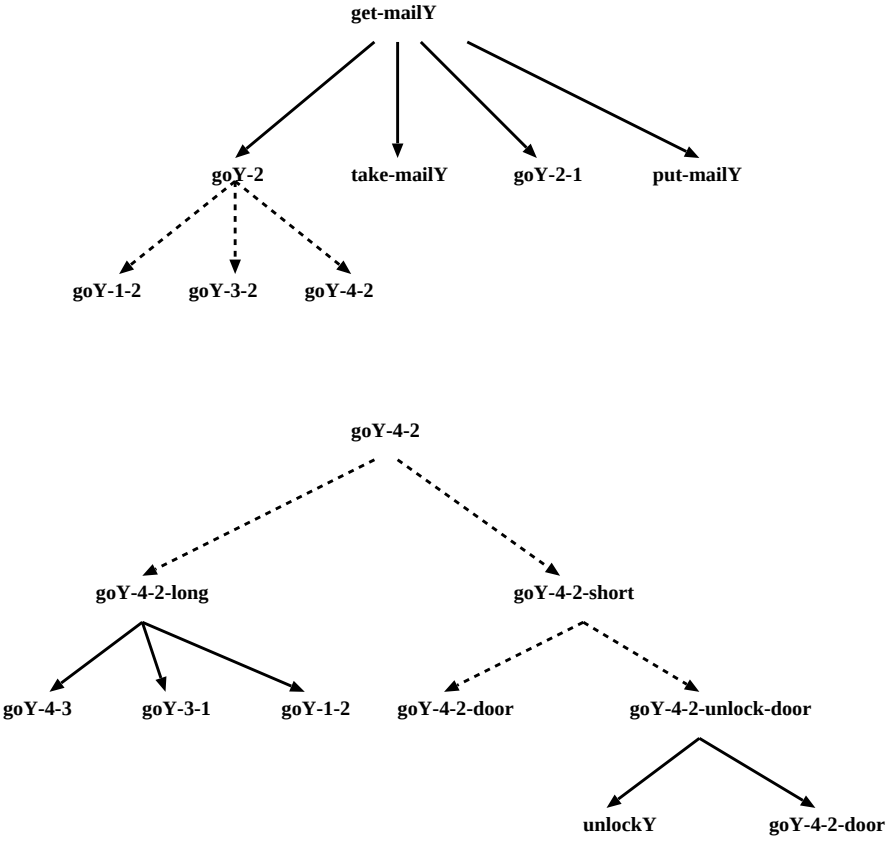

Figure 3.3: A portion of the library of plan schemata for the office micro-world domain representing the plan schema for getting the mail (above) and the plan schema for going from room 4 to room 2 (below); dashed lines represent the abstraction hierarchy, solid lines represent the decomposition hierarchy

3.2.3 Intentions and Plans

The agent architecture maintains a representation of the agent's intentions, which are constituted by the commitment to execute a plan by executing all plan steps, according to [Pollack, 1990]'s definition of *having a plan*. In an ideal setting, where intentions are static, once the agent has become committed to a plan, it executes it from the first to the last step. In order to know what is the next step to execute, the agent only needs to integrate its mental representation of the plan with the information about the last executed action as long as the execution proceeds.

In the architecture we propose, 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 goal achievement and 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,

```
(action

  :name           go-X-4-to-2-door

  :attributes     (at-X time fuel open-door)

  :effects        :cond-eff  :cond  ((and (at-X = 4)(door = open))
                             :eff   ((at-X = 2) prob = 1
                                     (time = time + 15) prob = 1
                                     (fuel = fuel - 0.5) prob = 1))
                             :prob  1

                  :cond-eff  :cond  ((or (not (at-X = 4))(door = closed))
                             :eff   ((time = time + 15) 1
                                     (fuel = fuel - 0.5) 1))
                             :prob  1

  :goal           (at-X = 2)

  :instantiation  nil

  :decomposition  nil
)
```

Figure 3.4: The definition of the elementary action *go-X-from-room-4-to-2* in the office domain.

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 (the intention to execute an action is dropped after the action has been executed) and re-deliberation (if the intention to execute a plan or a part of a plan is dropped, the intention to execute its steps is dropped as well).

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 plan-level commitment, and *action-executions*, representing action-level commitment.

Beside *plan* and *action-execution* structures, the architecture includes the representation of *actions instances*. While the agent knowledge about actions contains hierarchies of action types, the representations generated by the deliberative component refer to instances of action types. An *action instance* is characterized by a unique identifier, the information about the action type of which it constitutes an instance, and the information about the plan instance it belongs to.

The *plan* structure corresponds to the mental representation of a plan that

an agent builds as a consequence of the deliberation or re-deliberation process, while the *action-execution* structure keeps track of the agent's commitment to execute individual actions, similarly to the Plan Execution Situation construct proposed by [Traum and Allen, 1994]. Both *plan* and *action-execution* structures are needed for a deliberative reactive agent to be able to reason on its intentions and modify them in a dynamical fashion to respond to uncertainty and change. Since the agent is likely to re-deliberate, it needs a representation where executed actions and actions to execute are related to its evolving plan-related intentions. At the same time, it needs a representation of the plan it is currently committed to.

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 one plan, which constitutes its *current-plan*. Even if the *current plan* changes along time in a dynamical fashion, *plan* instances are internally static, as they represent the agent's commitment to a plan as a whole and are unrelated from its actual execution: a *plan* instance remains the same from the moment it becomes the current plan to the moment is dropped, in favor of a new, modified plan, which becomes the current plan. Each time a new plan becomes the current plan, it is added to an *intention history*, which tracks the evolution of the agent's intentions.

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 the achievement of 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. So, the *execution record* is internally dynamic, as it is updated every time the agent executes an action, and each time the agent revises its plan-level intentions (i.e. each time the current plan changes).

The *intention history* and the *execution record* provide the agent, at the end of every cycle, with complete information about how its plan-level intentions and its commitment to single actions have evolved along time. The *intention history* tracks the evolution of the the agent's plan-level intentions up to the current moment, while the *execution record* tracks how executed actions have been intended by the agent as steps of different plans, in accordance with the evolution of its plan-level intentions; when the commitment to a plan is dropped, the actions constituting the plan steps which have not been executed yet are discarded.

Notice that the two structures, separately from each other, would not suffice: recording only plan-level intentions would not allow the agent to "remember" all executed actions; recording only the execution-level information would leave the agent with a unmotivated sequence of executed actions, unrelated to any plan. [4]

[4]At best, the sequence of executed actions could be related to a static plan - if a perfect

With respect to Pollack's distinction between *knowing a plan*, and *having a plan*, the former is represented by the agent's knowledge about plans, while the latter is roughly encompassed by the *plan* and *action-execution* structures. These structures do not fit precisely in the theoretical framework proposed by Pollack, as their function is different; rather than focusing on plan inferencing, here we focus on reactive deliberation in a decision-theoretic perspective.

A **plan** structure contains the following elements:

- **Goal**: The goal the plan has been devised for;
- **Steps**: A sequence of action instances, constituting the plan steps;
- **Root**: The plan root, i.e. the action type that subsumes all plan steps in the decomposition hierarchy;
- **Expected-Utility**: The expected utility interval of the plan, i.e., the utility values ranging from the lower bound of expected utility to the upper bound of expected utility.

The agent loop generates a sequence of action executions which are incrementally stored in the execution record. The execution record is an instance of the **execution** structure, which contains the following elements:

- **Plan**: a link to the plan instance which constitutes the current plan
- **Goal**: the plan goal;[5]
- **Executed-actions**: the list of all actions which have been executed up to the current moment; the most recently executed action constitutes the **Last-executed-action**;
- **Actions-to-execute**: the list of actions the agent is committed to execute as remaining steps of the current plan; the first action to execute constitutes the **next-action**;

The *execution* structure is divided into a backward-looking component, Executed-actions, and a future-oriented component, Actions-to-execute. Executed-actions and Actions-to-execute are composed of action instances; since action instances contain a link to the plan they belong to, each action is linked to the plan of

match existed between the action sequence and one of the plans in the plan space defined by the action hierarchy.

[5]Notice that this information is redundant, for it is stored in the plan instance structure linked by the Plan element of the structure; however, it has been inserted for making the information about the goal directly available when the agent reasons about the execution.

which it constitutes a step. The actions in the Actions-to-execute list constitute the steps of the current plan that the agent has not executed yet and is committed to execute, while the actions in the Executed-actions list belong to the current plan or to any other plan in the same intention history (the sequence of plan-level intentions generated by re-deliberation with reference to the same goal).
Notice that if the execution of the initial plan is straightforward, actions are simply moved, at each agent cycle, from the Actions-to-execute list to the Executed-actions list, until the plan execution ends.

3.3 The Agent Architecture

The behavior of the agent is controlled by an execution-sensing loop with a meta-level deliberation step (see figure 3.5). When this loop is first entered, the deliberation module is invoked on the initial goal; the deliberation module devises a plan, by means of its planning component: 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, i.e., the agent forms a commitment to execute it, and 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.

3.3.1 The Planning Component

The planning component of the deliberation module builds on the DRIPS system, a decision-theoretic refinement planner. The decision-theoretic planning paradigm extends the classical goal satisfaction paradigm by allowing partial goal satisfaction and the trade-off of goal satisfaction against resource consumption.
Decision Theoretic Planning, being based on the conceptual framework constituted by Decision Theory, can account for uncertainty and non-determinism, which provide the conceptual instruments for dealing with uncertain world knowledge and actions having non-deterministic effects. These features make decision-theoretic planning, and the DRIPS system in particular, especially suitable for modelling agents who are situated in dynamically changing, non deterministic environments, and have incomplete knowledge about the environment (see [Blythe, 1999]).

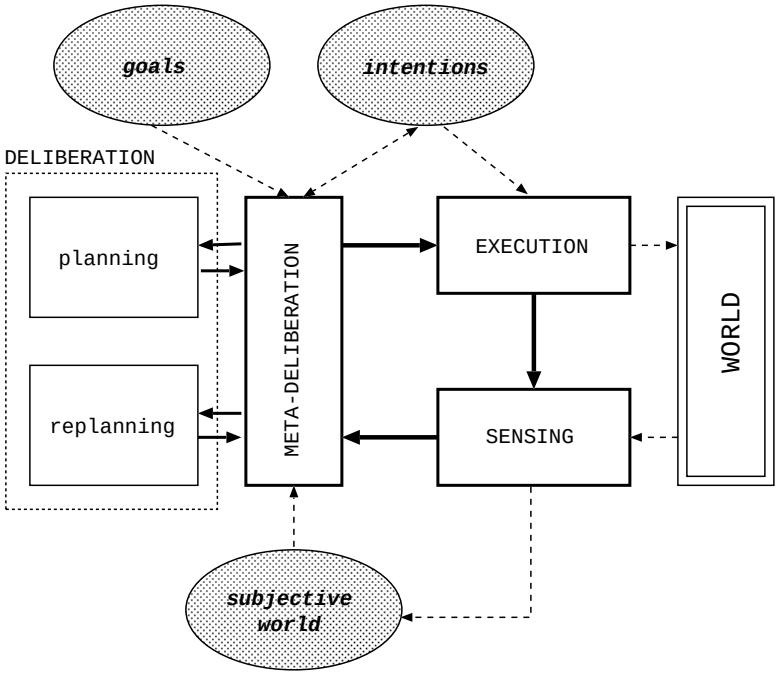

Figure 3.5: The structure of the basic agent architecture. Dashed lines represent data flow, solid lines represent control flow, and the grey components determine the agent's state

3.3.1.1 Decision-Theoretic Planning

Decision Theory ([Luce and Raiffa, 1957]) is the theory of rational decision, and characterizes the choice between alternative courses of actions in terms of the *utility* they yield. The utility of a course of action is computed on its outcome; however, due to uncertainty in the initial world or non-deterministic action effects, a course of action can result in a set of outcomes, with a probability distribution attached. A utility function computes the *expected utility* of a course of action by combining the inherent utility of its outcomes with their probability distribution: "Given a probability distribution over the possible outcomes of an action in any initial state, and a reasonable preference function over outcomes, we can define a utility function on outcomes such that, whenever the agent would prefer one plan over another, the preferred plan has the highest expected utility" ([Blythe, 1999], pag. 38). Finally, a multi-attribute utility function can account for a cost/benefit evaluation of the expected utility of a course of action.

By adopting the theoretical framework of Decision-Theory, Decision Theoretic Planning releases the "all or nothing" goal satisfaction paradigm. Goal achievement, being expressed by an utility value, is no longer reduced to the satisfaction of a logical formula. Rather than equating the notion of success with the crude achievement of the goal, *Decision-Theoretic Planning* allows an agent to make a

key distinction between the achievement of the goal and the utility associated to the state in which the goal is achieved: in other words, goal achievement is not evaluated separately from other parameters that are relevant from the agent's point of view, like costs. Goal achievement can be weighted against the costs of achieving them, and goals can be partially satisfied. What all these situations have in common is the fact that the states in which the goal is achieved, obtained by executing different plans, are ranked according to a certain preference order.

As surveyed by [Blythe, 1999], most Decision-theoretic planners rely on standard planning techniques to generate a set of plans such that each of them satisfies the goal, and focus on the choice among them, which is driven by a set of *preferences* embodied in a *multi-attribute utility function*. Besides a set of goals constituting the input to the planning process, the agent has a set of preferences which express the overall desirability of states. States are represented as sets of attribute-value pairs and the individual preferences of an agent are expressed as functions which map the value of single attributes to real numbers. In the DRIPS System, the overall utility-function of an agent is constituted of the weighted sum of the single preferences.

The preferences of an agent don't constitute the input to the planning process, i.e., the planning process is driven by the satisfaction of a goal as in the standard planning paradigm, no matter which specific planning technique is adopted. Preferences play a different role with respect to goals: they allow the planner to choose among alternative plans based on the desirability of the final state of each plan. A detailed analysis of how the utility model of an agent determines its behavior can be found in [Haddawy and Hanks, 1998].

3.3.1.2 The Planning Architecture

The DRIPS system searches the plan space for optimal plans using an abstraction/decomposition tree that describes the space of possible plans. Given a goal, the planner is invoked on a non refined, partial plan (i.e. a plan in which some actions are abstract), and outputs one or more plans which maximize the expected utility.[6] In theory, in order to find the plan with the maximum expected utility, the planner should generate all plans and then compute the expected utility of each of them.

The peculiarity of DRIPS is that it needs not generate all plans to select the best one, thanks to *inheritance abstraction*, that allows the planner to compare partial plans: the refinement process compares the partial plans which have been generated in the refinement step and chooses the most promising one, possibly pruning the plans which are suboptimal. Since a partial plan encompasses all its

[6] Although goals can be conjunctive, the planner is not able to devise a plan for achieving a conjunction of goals by combining the plans for achieving the individual goals; i.e., the planner can account for a complex goal, formed by a conjunction of simple goals, only if there is a recipe whose effect includes the conjunction of goals.

possible completions, the expected utility of a plan is represented by an interval, which includes the expected utilities of all possible instantiations of that abstract plan. If the lowest value of the expected utility range of a partial plan exceeds the highest value of the expected utility range of another plan, the latter can be pruned without expanding it.

Since the initial world can contain uncertainty, and actions can have non deterministic effects, a plan can result in a probability distribution over a set of alternative states. In order to evaluate the expected utility of a plan given a context (i.e., a world), DRIPS performs the *projection* of the plan, by generating a set of chronicles (sequences of states) representing all the possible outcomes of the plan, with a probability distribution. This anticipatory process relies on a Markovian assumption of independence of non deterministic action effects, that makes it possible to compute the probability distribution over the set of chronicles which constitute the outcome of a plan.

The construction of a plan is carried out by the planner in a stepwise fashion: Suppose that P^x is a partial plan for achieving the goal α, which contains a step β_m^x. First of all, the planner has to find the best instantiation for β_m^x (let's say P^y, with steps $\gamma_{m,1}^y$, $\gamma_{m,2}^y$, ..., γ_{m,n_m}^y); then, if necessary, the planner can start refining $\gamma_{m,1}^y$. The planner expands β_m^x in all possible ways (i.e. applying to the current state S all existing alternative instantiations of the step); then, it proceeds onward and expands the new partial plans obtained. The search goes on in parallel, but the search tree is pruned using the utility function (applied to the states resulting from the potential execution of the instantiations): so, the utility function acts as a heuristic able to exclude some possible ways to execute an action.

The plans generated by the planner are transformed into instances of the *plan* structure (see section 3.2). It is worth observing that these plans are slightly different from the plans generated by the original DRIPS system, since DRIPS does not enforce the distinction between action types and action instances: while DRIPS outputs a set of plans each constituted by a sequence of actions types, the planner used here outputs plan instances, namely sequences of action instances.

Notice also that the planner may output a pool of plans, if none is sub-optimal; in this case, the agent randomly selects one of the plan in this pool, and becomes committed to it. These plans constitute an intermediate level between the agent's plans knowledge and the agent's commitment: they are conceptually distinct from the agent's general knowledge about plans, as they are contextually generated, but the agent has formed no commitment to any of them yet.

3.3.2 The Execution Module

The agent's deliberation is based on its subjective representation of the world, maintained by the sensing component. The agent's activity, instead, takes place in the world in which the agent is situated, which is external to the agent. The

agent can access the external world and update its internal representation of it only by a sensing act; sensing is incomplete (and fallible) (see section 3.3.3) and, as a result, the agent's subjective representation of the world and the objective world can have some discrepancies.

For the motivations expressed above, namely accounting for uncertainty in the agent's knowledge about the world and for actions having non-deterministic effects, the representation of the environment need to be kept distinct from the representation available to the architecture of the agent. In order to enforce this distinction, the agent's actions are executed in a separate representation of the environment, where their actual effects take place; this representation is managed by a *simulator*.

When the agent deliberates about its intentions, in the phase preceding execution, it decides what course of action to commit to, or, in other words, what course of action will become its current intention. In order to know what course of action is the most advantageous according to its subjective preferences, it evaluates the expected utility of the available courses of action by simulating their execution in the current world.
However, a course of action can contain non-deterministic actions, or the agent's representation of the world can be uncertain, making it impossible for the agent to determine the final outcome with certainty; instead, the agent knows that a course of action is likely to result in a set of alternative outcomes, according to a certain probability distribution. Given a set of outcomes, the expected utility will be constituted by an interval, ranging from the outcome which is the less advantageous for the agent, to the outcome which is the most advantageous.
During the execution phase, the agent is committed to a course of action, with a certain interval of expected utility associated to it. The execution of each step is likely to modify the expected utility interval associated to this course of action: when a step is executed, only one of the possible effects takes place, thus reducing the agent's uncertainty about the final outcome.

The execution module communicates with the simulator which updates the objective world representation according to the messages it receives from the execution module. This simulator sees to it that, when an action has non deterministic or conditional effects, only one effect actually takes place, as it would happen in the real world. When an action has non-deterministic effects, the simulation module generates a world where one of the effects takes place. Similarly, when an action has conditional effects, but the agent has uncertain knowledge about the world and does not know the truth of the conditions in the world, the simulator simulates the execution of the action in the external world, where only one of the conditions holds, and only the corresponding effect applies.

As far as uncertainty about the current world state is concerned, keeping the subjective and the objective representations of the world separated allows the subjective representation of the world to be potentially uncertain, while the

objective world does not contain any uncertainty. When execution begins, the representation of the objective world in the simulation module consists of a world where each attribute has a certain value, representing the actual world, while the agent's initial representation of the world is constituted by a probability distribution over worlds.[7]
Regarding non-deterministic effects, the situation is slightly more complex: the simulation module generates n worlds corresponding to the n alternative effects of the action being executed, then picks up a world in a way that reflects the probability distribution. In practice, the simulation module converts the probability of each world into an interval (all intervals will sum up to 1), then draws a number randomly, and selects the world whose probability interval includes the number: in this way, over a sufficiently large number of draws, the distribution of action effects generated by the simulation module tends to mirror the probability distribution encoded in the agent's knowledge base.[8]

Moreover, the world can change dynamically, so that, at a certain moment, the agent's subjective representation may not match the objective world. Given the agent loop introduced before, the representation of the world contained in the simulation module can be altered between two subsequent sensing acts, by reproducing a multiplicity of real-world situations. By modifying the objective representation of the world between the agent's deliberation and execution, it is possible to model a situation where the world changes unexpectedly immediately after agent's deliberation, so that the execution of the selected course of action takes place in a different world than expected. Or, by modifying the objective representation of the world between the execution of a step and the subsequent monitoring, it is possible to reproduce what happens when the world changes immediately after the agent has executed an action, possibly altering the action effects.

If, for one of the reasons expressed above (a world change immediately before or immediately after the execution of the action, or a fault in the agent knowledge about the action) the execution of an action does not generate the effect it has been planned for, the agent incurs a *failure*.[9]
From the agent's point of view, all these situations are blurred: if the last executed action has had different effects than expected, the agent cannot tell whether the

[7] In order to represent the situation in which the agent's knowledge about the world is not uncertain, the probability distribution over initial worlds can be constituted by a probability distribution where there is just one simple world with a probability = 1.

[8] Alternatively, in order to model an agent's misconception about the probability distribution on the effects of an action, the simulator could be given a different probability distribution on the effects of that action with respect to the probability distribution that is attached to the effects of that action in the agent's knowledge.

[9] Since we lack a precise model of how the execution is carried out by the execution module, we don't consider, here, the execution failures produced by an error in the execution itself, as proposed by [Traum and Allen, 1994] in their taxonomy of execution failures for plan recognition purposes.

world was different than expected when the action was executed, or the world has changed unexpectedly after the action execution, or else if its knowledge about the last executed action was faulty.

Finally, changes can intervene after the agent's meta-deliberation, possibly making the result of the agent's meta-deliberation inadequate to the modified objective world.

3.3.3 Sensing

After the execution of each plan step, the agent senses the environment to verify if the expected action effects hold and updates its own internal representation of the world. Then, the *meta-deliberation* module is invoked on the updated agent's representation of the world to direct the choice between sticking to the current intentions, by continuing the execution of the current plan, or deliberating on them, by modifying the current plan. As anticipated in the previous section, the agent's monitoring allows the agent to ascertain if the expected effects of the last executed action hold in the current world; at the same time, it progressively reduces the agent's uncertainty about the outcome of the course of action the agent is currently committed to.

In order to avoid the expensive process of the agent sensing all the attributes defining its world in search for changes, sensing is restricted to the attributes affected by the execution of the last step, as encoded in the agent's knowledge about actions. This limited sensing activity, besides satisfying the general requirement of saving resources whenever possible, corresponds to the phenomenon of an agent *focusing* its attention only on a relevant portion of the world at a time, as it has been first evidenced by cognitive accounts of attention and studies on focusing in discourse and dialogue (see [Grosz, 1981] and [Grosz and Sidner, 1986]). In this case, the identification of the relevant portion of the world is driven by the agent's expectations about the effects of its activity: the agent senses only the values of the attributes that are affected by its last executed action, according to the action definition.
Several models have been proposed, where plan execution monitoring is accomplished by accounting for resource-boundedness and cognitive limitations in a more accurate way (see for example [Zilberstein, 1996], [Boutilier, 2000] and [Pollack and McCarthy, 1999]). By restricting sensing to the action effects, this model of sensing accounts for theories of plan monitoring, although in a general way.

Since sensing is selective, the agent's internal representation of the world is updated only with respect to the relevant attributes of the last executed action: the remaining attributes are not affected by the update. The obvious consequence is that, if the agent is uncertain about one or more attributes that are not in the set of currently relevant attributes, the uncertainty persists, and the same holds for wrong beliefs.

3.3.4 Meta-Deliberation and Replanning

After the agent's internal representation of the world has been updated, in the sensing phase, with the effects of the last executed action, the agent must verify if its current plan is still feasible given the updated representation.

Initially, the agent is committed to a plan; this plan is associated, as described before (see 3.3.1), to an expected utility interval, due to the agent's uncertainty about the outcome of the plan: the actual utility of the plan will be a value in the range established by this interval. As the execution of the plan proceeds the utility interval may vary. 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. For example, the execution of an action can result in a disadvantageous effect which was improbable given the probability distribution on the action effects known to the agent. If this is the case, it may be appropriate for the agent to reconsider its intentions, i.e. to modify the plan it is committed to in order to bring back the maximal utility to the previous value.

These remarks concern the cases in which the execution of a plan is not affected by any external changes in the world; however, we assume that the world is dynamic, and can change independently form the agent's activity. The world may change by affecting the action execution or the action effects in an unpredictable way: actions may fail, or result in unpredictable or improbable effects, by making the plan maximal utility drastically drop; at worst, given the current plan, the agent's goal may be not reachable anymore.

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 (the execution monitoring function returns "replan"); that is, the agent's best expectations have worsened significantly.

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.

If the *replanning component* outputs a plan, the execution record and the intention history are updated and the execution resumed. If, on the contrary, the replanning module is not able to devise a plan that has an acceptable cost, a failure is encountered that affects the commitment to the current goal. This notion of failure complies with the conditions in which it is rational to drop a goal according to ([Cohen and Levesque, 1990a]): the impossibility to find an alternative plan at all means that the current goal is not reachable anymore, while the impossibility to find a plan which achieves the goal at an acceptable cost corresponds to the goal becoming irrelevant.

3.3.5 The agent algorithm

In summary, the agent algorithm is the following:

```
procedure agent (goal, subjective-world)
    /* initial planning phase */
    plan := DELIBERATE (goal, subjective-world);
    execution := INITIALIZE-EXECUTION (plan);
    /* the agent loop begins here */
    loop
      /* execute next action */
      objective-world := EXECUTE (next action);
      /* sensing */
      subjective-world := MONITOR (next-action);
      /* check if goal has been achieved */
      if execution.actions-to-execute = empty
          and ACHIEVED-GOAL (subjective-world, goal) = T
       then return SUCCESS;
       * The meta-deliberation phase is entered: */
       else if MONITOR-EXECUTION (subjective-world) = "replan"
         /* the agent tries to revise its intentions */
         then
           begin
             /* redeliberation is attempted */
             new-plan := RE-DELIBERATE (execution, subjective-world, goal);
             if new-plan
               /* new feasible plan found, update intentions */
               then plan := new-plan; UPDATE-INTENTIONS (plan, execution)
               /* no new feasible plan */
               else return FAILURE
```

```
            end if
          end
        /* no revision, the agent updates the action execution */
        else SET-NEXT-ACTION (execution)
        /* the first action in the list of non-executed actions is selected */
      end if
    end if
  end loop
end procedure
```

3.4 Reactive Deliberation

Reactivity provides the agent with the ability to deal with non-determinism and uncertainty by modifying its intentions in a dynamic way, in response to external changes and failed expectations. With respect to the agent loop introduced before (section 3.3.5), re-deliberation is performed if the monitoring reveals that the current plan is *unfeasible*, i.e. it does not achieve the goal anymore or its expected utility has decreased significantly. However, the agent does not try to devise a new plan from the scratch: instead, it tries to save time and resources by "recycling" the unfeasible plan, if possible.

Before examining the techniques exploited by the replanning component in more detail, some considerations must be devoted to the notion of failure in decision-theoretic planning (see section 3.3.1.1).
From the point of view of decision-theoretic planning, if an action fails, i.e. it doesn't reach the effects it has been planned for and disappoints the agent's expectations, it can affect the achievement of the goal in two ways: by compromising the achievement of the plan goal, or by affecting the expected utility, without threatening the achievement itself (as reflected by the definition of unfeasibility of a plan given above).
Recall that, in decision-theoretic planning, the achievement of a goal is evaluated in association with a set of individual *preferences*, which are embodied in an agent's utility function. In other words, the expected outcome of a plan is not evaluated only with respect to the crude achievement of the goal, but also based on the partial satisfaction of the goal, and on the value of a set of relevant parameters. The agent's individual utility function can model situations where, given a set of plans for achieving a goal, the notion of goal achievement is not clear-cut: if a goal can be partially satisfied, the agent may have a preference for a higher degree of satisfaction of the goal - given an acceptable minimum; or, if a set of attributes is related to the goal satisfaction (for instance, attributes representing the resource consumption), the agent may have a preference for these attributes having a certain value in the final state (see previous section 3.3.1.1).

Decision-theoretic planners like DRIPS rest on traditional planning techniques, like refinement planning, to generate plans, then order them according to their utility in order to find the plan that maximizes the expected utility.[10]: plan generation and utility-driven choice are kept distinct, and the representation of actions encodes their local effects rather than explicitly coding the way they contribute to the plan expected utility: the plan expected utility is contextually determined by anticipating the possible plan outcomes and applying the utility function to them.
So, the information about the way a local failure affects the utility of the entire plan, which is needed in order to repair it, is not directly available, as it is not inherent in the plan structure. [Haddawy and Hanks, 1998] depict this problem of the "preference model" in contraposition to the "simple goal model", in the following way, : "There is no guidance regarding how to repair and improve a plan, which is precisely the information supplied in the simple goal model by the goal formula and plan operators" (p. 394). In order to fill this gap, [Haddawy and Hanks, 1998] propose the multi-attribute utility model (which the DRIPS system is based on), where the overall expected utility is computed from the satisfaction of a set of preferences. However, as utility is a plan-level notion, it is not clear how the contribution of local action effects to global plan utility could by computed, but it is reasonable to assume that it could involve some kind of complex regression from the plan outcome.

In our model, we distinguish *two ways in which a plan can fail*, depending on whether the plan failure is caused by the failure of an action or not.
In the first case, the achievement of the goal is compromised by the the failure of an action which constitutes a step of the plan(see section 3.3.2). This failure can be easily detected by sensing the action effects, thanks to the explicit encoding of the action goal in action definitions.
In the second case, the execution of an action, although it does not *fail* with respect to the action intended effect, produces different side-effects than expected, by affecting the utility of the plan in a negative way. The detection of this kind of failures is not straightforward, since the action definition does not contain any information about how the action effects affect the plan utility - this relation being encoded in the utility function: in order to verify that the action execution has not compromised the expected utility of the plan, the agent must compute the expected utility of the remaining part of the plan in the updated world (see section 3.3.3).

The replanning module gives precedence to *predicted failures*; when the anticipation process in the meta-deliberation phase reveals that a plan is not feasible

[10]Some systems have been developed, like Buridan [Kushmerick et al., 1994], which find a plan whose probability of success is above a certain threshold, instead of finding the plan with the highest expected utility. However, the problem of integrating the two approaches has not been investigated yet.

anymore, the replanning algorithm module tries to modify the plan by working on its non executed steps, instead of attempting first to repair executed steps based on a diagnosis of the failure. The rationale for doing so is twofold: on the one side, due to the environment's property of being dynamic, reasoning on the causes of the failure of an executed action, in order to repair it, is a complex task, that requires a causal model of the domain[11]; on the other side, it is preferable, as discussed later in section 3.4.1.2, to try to reuse executed steps, modifying one's own future intentions instead.
However, even if a diagnosis of the failure is not attempted, the replanning module tries to focus the repair process based on the information about predicted failures, as it will be made clear in the description of the submodules which compose the replanning module.

The **replanning module** relies on two separate submodules to devise a new plan starting from the unfeasible plan. First, it invokes a **plan repair** module, which tries to substitute existing plan actions with alternative actions, then, if the plan repair module is not able to devise a new feasible plan, it invokes a **plan expansion module**[12], which tries to add new actions to the failed plan.

In order to overcome the limitation of the preference model depicted above, the process of repairing a plan is driven by an heuristic (see next section, 3.4.1). The plan repair algorithm, following the assumption that *interdependent actions tend to be temporally and spatially close*, tries to alter the plan portion which is local to the failed action, gradually increasing the scope of modifications. The "local" portion of plan with respect to an action is defined by the decomposition hierarchy, which encodes the interdependencies between sequential actions (see the discussion of locality of action interdependencies in section 3.4.1.1).
This strategy obeys to the principle of *conservative repair*: it does not guarantee that the new plan is optimal from the point of view of the expected utility, since the search strategy may examine only a portion of the failed plan before finding a new feasible plan with an acceptable utility, while, in order to find an optimal plan, it would be necessary to reconsider all plan actions. However, reconsidering all plan actions would be very expensive in terms of time and cognitive resources, while this strategy is in line with the constrains an agent is subject to.

The modification process performed by the Plan Repair algorithm on the initial plan exploits the fact that action types are organized along a specialization hierarchy (see the dashed lines in figure 3.3): starting from the lowest abstraction level (directly executable actions), it looks for an alternative instantiation of one or more plan steps that yields a new feasible plan. When looking for alternatives, the algorithm tries to minimize the number of steps involved, by obeying, at the same time, to the principle of locality mentioned above. If no feasible alternative

[11]However, replanning based on the diagnosis of the plan failure certainly constitutes a promising line of research.

[12]Not to be confused with the technique of plan refinement.

is found (i.e. a new plan with a reasonable expected utility), the next level of abstraction is considered, until the root of the hierarchy is reached; eventually, the replanning module outputs either an updated plan, or a failure. Notice that operating on the hierarchy root means building a completely new plan.

If the plan repair submodule does not return a feasible plan, the replanning module calls the Plan Expansion submodule. The functioning of the Plan Expansion module is inspired to the style of reasoning of partial order planning ([Penberthy and Weld, 1992] and [Weld, 1994]): it selectively looks for steps whose preconditions are false, and tries to expand the current plan by inserting new actions in order to restore the failed preconditions. The Plan Expansion Algorithm is based on the assumption that those steps which are note enabled (i.e. whose preconditions do not hold) are probably responsible for the plan failure, and good candidates for guiding the replanning process.
For each false precondition, the Plan Expansion algorithm looks for elementary actions which restore that precondition, searching the action type definitions contained in the library of plan schemata as a library of plan fragments. Being inspired to a different reasoning style, the Plan Expansion algorithm does not exploit the plan space defined by the hierarchal relations encoded in agent's knowledge about actions. For this reason, this algorithm is applied only as a second attempt when the Plan Repair algorithm fails.

3.4.1 The Plan Repair Algorithm

3.4.1.1 Theoretical Framework

The Plan Repair algorithm rests on a set of assumptions;

- **Stability of Intentions**. In our model, the agent is committed to the current plan, and will stick to it as much as possible.
  According to [Bratman, 1990]'s analysis of commitment, intentions are stable and resist reconsideration. Persistence of intentions has been analyzed in detail by [Cohen and Levesque, 1990b], who define the conditions at which it is rational to maintain an intention: an agent does not abandon a goal unless it has been reached, is not reachable anymore or an escape condition holds.
  At the same time, intentions tend to be partial, and under-specified, since an agent normally postpones deliberation about the details of how to execute a plan until the execution time comes reasonably close. However, there is a trade-off between stability and under-specification: if the agent's intentions are fully specified and become inadequate to the current world, the algorithm tries to maintain them at an higher, abstract level, by questioning only their lower-level specific instantiation.
  The goal of the repair algorithm is to perform first a "partialization" op-

eration, as local as possible, in order to go back to an under-specified plan which is then refined anew taking into account the new situation.

- **Limited Resources**. The agent is resource-bounded, so it should avoid rethinking the entire plan, if not strictly necessary.
  Although a new, completely different plan may be the optimal one with respect to the changed context, replanning has a cost in terms of time and cognitive resources. In particular, if a plan fails during its execution, this fact further limits the extent of deliberation allowed to the agent in terms of time and resources.
  In line with these limitations, the principle of *conservative repair* (see []), establishes a preference for re-using executed steps; this preference constrains the space of possible plans, by focussing the replanning process on plans which re-use executed steps.

- **Locality Principle**. The agent model proposed here makes use of hierarchically organized libraries of actions, which correspond to recipes, or plan schemas. This approach is cognitively motivated since it relieves the agent from the task of planning from first principles and makes the planning process easier and faster, though less flexible and adaptable. Recipes constitute standard ways to solve a planning problem, represented in a compact way, and can be seen as the result of the "compilation" of an agent's planning activity.
  Recipes are obtained in a bottom-up fashion, by abstracting recurrent sequences of simpler actions into complex actions (see [Fikes and Nilsson, 1971], [Sacerdoti, 1977], and [Tenenberg, 1991]), and omitting the information about the enablement links that relate them. An action enables another action when the execution of an action sets the conditions for the execution of another, subsequent action.
  Sequential abstraction is based on the assumption that actions which are related to each other by the existence of an enablement link tend to be executed contiguously, trying to limit the interposition of intermediate steps; thus, it is established a principle of temporal locality of interdependencies. Given this locality principle, if a step in a sequence is modified, modifications normally affect steps which are temporally closer, i.e. immediately preceding and following steps, and leave the other steps unaffected.

  As noticed by [Nau et al., 2001], total order requirement on the steps in hierarchical planning is too restrictive for some domains. However, in the absence of this requirement, complex information about dependencies between actions in a domain must be included in the knowledge base in order to constrain the ordering of steps. So, the trade-off between releasing the total order requirement and explicitly coding interdependencies in the knowledge base must be carefully evaluated in a domain-dependent way.

> Even if we have not made any accurate evaluation of the domain we use for experiments, this constitutes an issue for future research.

In the following, we give some useful definitions for illustrating the functioning of the Plan Repair Algorithm .

Remember from section 3.2 that the agent's knowledge about plans is constituted by abstraction/decomposition hierarchies of action types. Each hierarchy is a tree, where nodes represent action types, and links represent decomposition or abstraction relations.

Given an abstraction-decomposition hierarchy, the **plan space** is constituted by all plans (sequences of action types) that can be generated by following the abstraction-decomposition links in the hierarchy. Here, we consider as plans also the sequences which contain non-elementary action types; we call these plans **partially refined plans**, and the plans constituted only by elementary action types **fully refined plans**.

A **plan instance**[13] is an instance of one of the plans in the space of possible plans and is composed of an ordered sequence of steps. Steps are constituted by **action instances**, i.e. instances of action types; if the plan is intended for execution, its steps are directly executable, i.e., they are instances of elementary action types. A **sub-plan** is a plan whose steps are a proper subset of contiguous steps of another plan; we say that a sub-plan is included in another plan. Finally, corresponding to fully refined and partially refined plans, there are also fully refined and partially refined plan instances.

With respect to an action node, we call **plan subspace** the space of possible plans described by the collection of nodes dominated by that action node in the hierarchy of actions; since the hierarchy of actions is a tree, this collection is also a tree. If the action node is an elementary action node, the plan subspace is constituted by that node, and it contains only one plan, that has that node as the only step. Notice that each plan in a plan subspace is a sub-plan of some other plan in the overall plan space described by the overall action hierarchy.[14] For instance, consider the plan subspace represented in figure 3.6, determined by action node D. In this plan subspace it is possible to identify eight plans, (D), (G), (H), (M), (N), $(G\ H)$, $(G\ M)$ and $(G\ N)$; two plans, $(G\ M)$ and $(G\ N)$, are fully refined plans, while the remaining plans are partially refined plans. Notice also that the plan constituted by the sequence of steps G, N is a sub-plan of the plan constituted by the sequence of steps G, N, L.

Given a plan-space (or a plan space subset), we say that a plan (or a plan instance) is a **complete plan** if it is not included in any other plan (or plan instance) in that plan space (or plan space subset). Of course, the notion of complete plan is relative to a plan subspace, so that a complete plan in subspace

[13] For the sake of brevity, in the following we will refer to plan instances as plans

[14] Unless the action node is the root of a decomposition hierarchy.

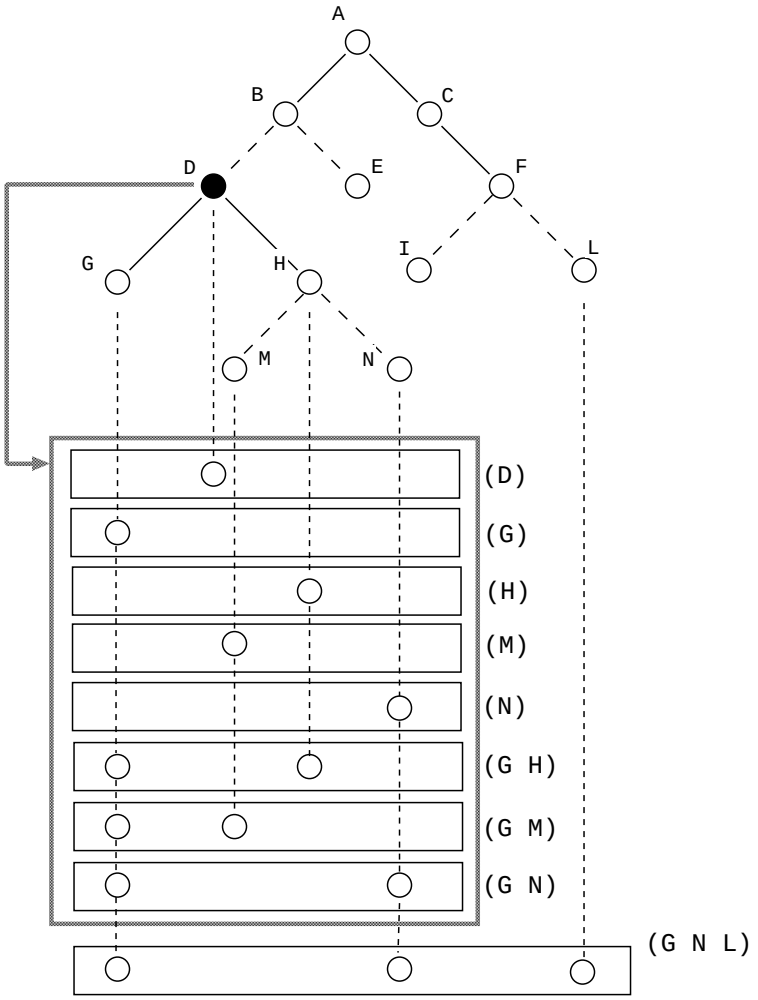

Figure 3.6: The representation of a hierarchy of generic action types. Solid lines represent decomposition relations, dashed line represent abstraction relations. The action node in black determines the plan subspace constituted by the set of plans represented in the bottom part of the figure (each plan is contained in a separate box); an extra plan is represented for exemplification purposes (see section 3.4.1.1).

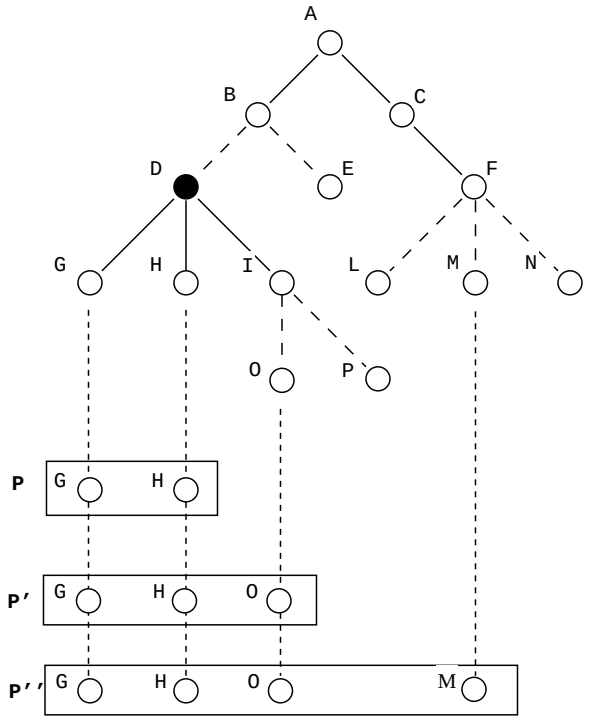

Figure 3.7: A figure representing three plans, (P, P', and P''), and the action hierarchy used to generate them. With respect to the plan subspace identified by the action node D, P is not a complete plan, while P' is a complete plan; both P and P' are sub-plans of P''.

S' can be a sub-plan of some plan in a larger sub-space S'' : for example, in figure 3.7, plan P' is a sub-plan of P'' and a complete plan, while P is a sub-plan of P but it is not a complete plan.

Given a sub-plan (no matter if it is complete or not) the portion of the abstraction/decomposition hierarchy which is **relevant** to that sub-plan is the set of action nodes dominated by the complex action which constitutes the *lowest-level common ancestor* of all actions in the plan. Since the action hierarchy is a tree, the relevant portion of this hierarchy with respect to a plan is also a tree, and it describes a subset of the overall plan space (according to the above definition of plan space subset); we name the action node that constitutes its root **local plan root**[15].

Since the Plan Repair Algorithm works on the decomposition/abstraction hierarchy, we must define some properties of plans which are related to this hierarchy. Recall from section 3.2 that a partial plan contains one or more steps which consist of abstract actions: since abstract actions subsume a set of alternative actions in the abstraction hierarchy, partial plans subsume a set of alternative plans. We say that plan Pp is **more partial** than plan Pp' if, with respect to the abstraction/decomposition hierarchy, they share the same local plan root and Pp' can be obtained by refining Pp along abstraction links.

At the same time, we say that a plan is under-determined if it contains non-primitive, decomposable actions. A plan Pu is **more under-determined** than Pu' if they have the same local plan root, and Pu' can be obtained by refining Pu along decomposition links.

3.4.1.2 The Algorithm

With reference to the agent algorithm introduced in section 3.3, the Plan Repair Algorithm is invoked during the meta-deliberation phase, after the execution phase and the subsequent sensing phase. At this point, the current action has been executed, and it has been moved from the list of the actions to execute to the list of executed actions, becoming the new last executed action; the subsequent action in the list of the actions to execute has become the new next action, i.e. the first action in the list of the actions to execute.

The Plan Repair Algorithm recursively calls itself on increasingly more under-determined plan instances, until it eventually reaches the root of the action hierarchy, or finds a feasible plan.
The algorithm performs on the input plan a **revision** process - the core of the replanning mechanism - aimed at modifying a *subpart* of this plan. The revision process is based on the notion of partiality defined above: roughly speaking, starting from the input plan, a more partial plan is built (partialization), i.e. a

[15]Notice that this definition allows finding a relevant subtree of the action hierarchy and a local plan root for any arbitrary sequence of contiguous actions

plan where a step has been replaced by a more abstract action. Then, the planner is invoked on this partial plan, called **revision plan**, and a new plan is obtained. The new plan differs from the initial plan since one or more of its steps constitute a different instantiation, with respect to the initial plan, of some abstract actions situated at some level of the abstraction hierarchy.
In practice, the initial plan and the new plan are generated by refining the revision plan at different times, with different initial worlds. In the worst case, the new plan coincides with the initial plan, (no alternative refinement is possible, even in the changed context), while, in the best case, it is different and it yields an acceptably higher utility than the input plan: if this is the case, the replanning ends and the new plan is returned; otherwise, the revision process is repeated until a new feasible plan is found or the current under-determinacy level has been completely explored.

Initially, the Plan Repair Algorithm takes as input the most recent plan instance in the intention history, i.e., the directly executable plan the agent is currently committed to execute.

1. The algorithm selects a step of this plan as the focused step, by calling an apposite selection function, Set-Focused-Steps.
   Set-Focused-Step tries to focus the search for alternatives on a step which will certainly fail when executed. In order to do so, it examines the preconditions of the actions which constitute the remaining plan steps: if the preconditions of an action are not predicted to be satisfied when the action is executed, that action is not *enabled* and its execution will fail, i.e., it will not achieve its intended effect (this process will be described in detail in section 3.4.3). If there are more than one non-enabled steps, the first one is selected, as the failure of the subsequent ones is supposed to depend on it; if no such step exists, the focused step is set to the next step.
   On the contrary, when the Plan Repair algorithm recursively calls itself on a more under-determined plan, the selection of focused step is determined by the previous plan repair cycle, as it will be made clear later.

2. Then, based on the focused step, the Plan Repair Algorithm selects a sub-plan of the input plan that will constitute the scope of revision (**candidate sub-plan**) and calls the Revision algorithm on it.
   The candidate sub-plan to be passed to the Revision Algorithm is selected by calling the Find-Candidate-Sub-plan function, which performs the following steps: given the action hierarchy, it selects the action node which constitutes the lowest ancestor of the focussed step according to the decomposition hierarchy. This node is called **lowest decomposition ancestor** (**LDA**) and is the local plan root (see definition of local plan root above in this section) of a sub-plan of the input plan, which becomes the candidate

sub-plan; its steps will constitute the **candidate steps** for the revision process.

3. Given the focused step, the search for alternative plans by the subsequent Revision process is limited to a portion of input plan, namely to the candidate sub-plan. The steps of the candidate sub-plan constitute the decomposition of the LDA. The rationale for this choice is to ensure that only the steps of the plan which are most directly related to the focused step are potentially modified by the revision process.
   Only if a solution to the repair problem is not found by modifying this sub-plan, the scope of revision is enlarged by collapsing the candidate sub-plan on the LDA.

   The Revision Algorithm is given as input the plan, the candidate sub-plan, and the focussed step.

   - First of all, the algorithm calls Set-LDA, that sets the LDA to the local-plan-root of the candidate sub-plan.
   - The Order-Candidates function establishes the order in which the candidate steps will be examined for partialization: given the focused step and the LDA, the focused step is examined first; then, the steps on its right (the ones that temporally follow it) are examined, if any, and finally, the steps on its left (the ones that precede it, if any) are examined.
     The revision process is incremental: if the revision does not produce a new feasible plan, the next revision takes the partial revision plan as input. This guarantees that, if the only solution involves modifying all steps at a time in the candidate plan, the algorithm will find it before collapsing to the next higher level.
   - Then, the Revision algorithm starts the partialization process on the steps of the candidate plan, i.e., on the candidate steps.
     (a) The Revision Algorithm examines the path on the abstraction / decomposition hierarchy which connects that candidate step to the LDA, in search for an abstract action (Find-Revision-Step); if there are more than one abstract actions, the lowest-level one is selected, while, if there are no abstract actions, the candidate step is discarded.
     (b) The selected abstract action becomes the **revision step**, and it is substituted for the candidate step in the input plan (Substitute-Step), by obtaining the **revision plan**.
     (c) The revision plan is refined by the planner, obtaining a **new plan**. As noted before, the revision plan is more partial than the input

plan, as it subsumes a set of alternatives plans which includes both the input plan and the new plan.

(d) The new plan obtained by refining the Revision Plan is then evaluated in order to assess whether it is a feasible plan or not. This test is accomplished by the Is-Feasible-Plan function by simulating the execution of the new plan in the current world, obtaining a set of outcomes.
First, Is-Feasible-Plan verifies if the goal is reached in at least one of the alternative outcomes; if so, it computes the expected utility range of the new-plan, and verifies if the difference with the expected utility range of the initial plan is below the given threshold. If the plan does reach the goal and its expected utility range is acceptable, it is deemed a feasible plan: the revision process stops and the Revision algorithm returns the new plan.
Otherwise, the partialization process is attempted on the plan obtained in step 2 with the next candidate step.

Eventually, the Revision Algorithm returns a new, feasible plan, or a failure.

4. If the Revision Algorithm fails, this means that no alternative instantiation of the steps in the candidate sub-plan solves the plan repair problem; so the scope of revision is enlarged, by recursively calling the Plan Repair Algorithm on a plan where the candidate sub-plan has been replaced by the LDA (remember that the Lowest Decomposition Ancestor is the local plan root of the candidate sub-plan). In order to do so, the Collapse-Plan procedure "collapses" the candidate-plan on its local plan root; the LDA becomes the new *focused step* that drives the search for modifications in the subsequent revision process.

   Notice that, each time the Plan Repair Algorithm calls itself recursively, the input plan is more under-determined. While the revision step increases the partiality of the plan within the boundaries imposed by the decomposition hierarchy, the collapsing step increases its under-determinacy, by allowing for enlarged search boundaries in the following revision step. However, the Plan Repair Algorithm - when it succeeds - always outputs a fully-fledged, executable plan, no matter how under-determined is the input plan.

It is worth noting that, even if the Plan Repair Algorithm does not find the optimal plan, it is complete, as it will eventually explore all the overall plan space before returning a failure. Of course, this algorithm is advantageous only if a 'local' solution can be found with respect to the focused elementary step, while it is rather disadvantageous if the solution can be found only at a high level of the abstraction hierarchy, as, in that case, the algorithm will perform several revision cycles moving upward in the decomposition hierarchy.

Although the planning paradigm adopted here is different, the re-refinement and the partialization steps resemble the plan refinement and plan retraction movements introduced by [Hanks and Weld, 1995] (see Chapter 2 for a detailed description).

In summary, the Plan Repair Algorithm is the following; it takes as input a plan under execution, and a world:

```
function PLAN-REPAIR (plan, world, goal, EU )
    focused-step := SET-FOCUSED-STEP (plan);
    candidate := FIND-CANDIDATE-SUB-PLAN (plan, focused-step);
    new-plan := REVISION (plan, candidate, focused-step, world, goal, EU );
    if new-plan then return new-plan
      else if plan = plan-root
        then return nil
        else
          begin
            large-plan := COLLAPSE-PLAN (plan, candidate-sub-plan);
            /* calls itself recursively on a "collapsed" plan */
            PLAN-REPAIR (larger-plan, world, goal, EU)
          end
      end if
  end if
end function

function REVISION (plan, focused-step, sub-plan, world, goal, old-EU)
    LSR := SET-LDA (sub-plan);
    ordered-candidates := ORDER-CANDIDATES (sub-plan);
    for each candidate in ordered-candidates do
      revision-step := FIND-REVISION-STEP (LSR, candidate);
      revision-plan := SUBSTITUTE-STEP (sub-plan, revision-step);
      new-plan := REFINE (revision-plan, world)
      if IS-FEASIBLE-PLAN (new-plan, world, goal, old-EU)
        then return new-plan
        else sub-plan := revision-plan
      end if
    end for each
end function
```

For example, consider the repair process that the Plan Repair algorithm would perform given the generic library represented in figure 3.8 and the plan represented in figure 3.9(1), which is composed of the steps B' - D - E'.

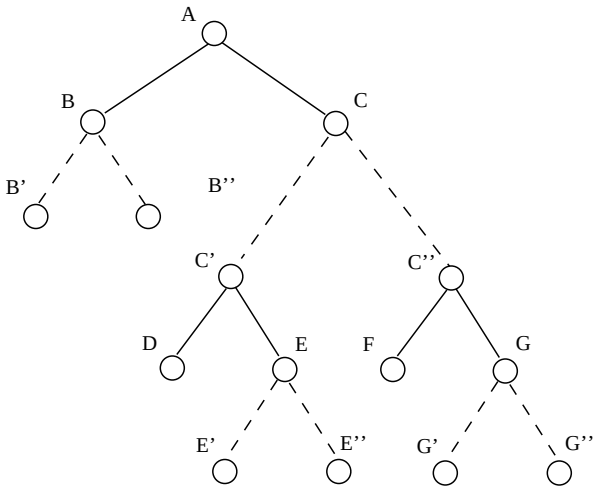

Figure 3.8: The representation of a generic hierarchy of action types. Solid lines represent decomposition links, dashed lines represent abstraction links.

1. At the beginning of the repair process, assuming that the focused step is action D, the LDA is set to C'. The candidate plan is thus constituted by the steps D and E'. D is examined first, but a revision node cannot be found (as it has no parent in the abstraction hierarchy); a revision node (E) is found for its (unique) right sibling, the candidate step E' (2).

2. The planner is given as input the partial plan B' - D - E. Assuming that a feasible plan is not found (i.e., B' - D - E'', the only possible alternative to the original plan, does not work), the plan repair process is started again after collapsing the candidate sub-plan on its local sub-plan root, the LDA node C'.

3. Given the new input plan B' - C', the new focused step now being the former LDA, C', the algorithm selects A as the new LDA. The new candidate plan now is constituted by the steps B' and C'. The focused step is examined first, and the revision node C is found, so the planner is invoked on the partial revision plan constituted by B' and C (3).

4. Again, assuming that a new feasible plan has not been found, the repair process would continue by examining B' in search for a revision node (4). Before the candidate plan is collapsed on its root, the repair process gives the planner as input the revision plan obtained by substituting the revision node, B for the candidate step being examined, B', in the previous revision plan (B' - C).

5. Finally, if the refinement of the revision plan B - C does not yield a feasible plan, the candidate plan is collapsed on the plan root A. If a feasible plan

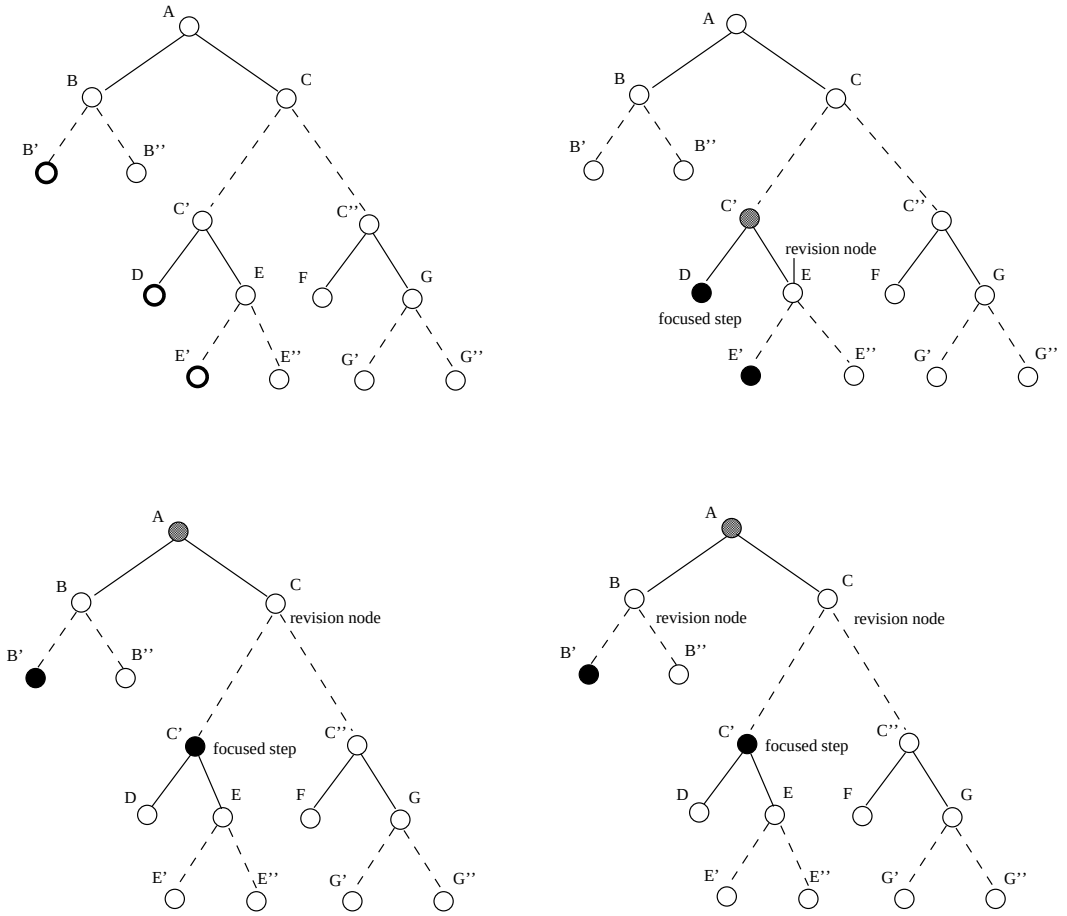

Figure 3.9: A graphical representation of the plan repair process on a generic action hierarchy. (1) represent the initial plan; (2)-(3)-(4) represent the subsequent phases of the plan-repair process. Black nodes represent the actions constituting the candidate steps in the candidate sub-plan, while the grey nodes represent the Lowest Decomposition Ancestor.

is not found by refining the plan constituted by the root alone, the plan repair algorithm fails.

With respect to the SPA plan modification algorithm presented by [Hanks and Weld, 1995] (see section 2.4.1 in Chapter 2), the solution proposed here contains some similarities and differences that are worth mentioning.

As far as the purpose of the algorithm is concerned, here the focus is on repairing a failed plan, rather than adapting an existing plan to a new planning problem: so, beside the efficiency reasons mentioned by [Hanks and Weld, 1995] in favor of the plan modification approach for solving a planning problem, our approach to plan repair is supported by the specific properties of intentions mentioned in section 3.4.1.1 (stability of intentions, limited resources, and the locality principle).

For what concerns the technique employed for modifying an existing plan, the two algorithms presuppose two different planning techniques: the SPA algorithm is based on a partial order planner, while the Plan Repair algorithm rests on the use of a utility-driven hierarchical planner. Although the paradigm of hierarchical planning does not lend itself to causal reasoning on the failure of a plan, its use is motivated by its higher cognitive plausibility.

The search on the plan space performed by SPA is based on two complementary procedures: the refinement procedure searches the portion of the tree of partial plans dominated by a node, while the reconsideration procedure climbs the tree of partial plans by retracting previous refinements.
Although the use of a refinement planner on partial plans we adopt is similar in spirit to the refinement step in the SPA algorithm, our solution benefits from the properties of the utility-driven planner on which it is based: given the current node in the tree of partial plans, the search is directed towards the most promising plan, i.e., the plan which has the highest expected utility.
Then, Plan Repair algorithm accounts for the advancement of the execution of the original plan, which contributes to determine the portion of it to be revised: given the ordering of the plan steps and the execution record, the portion of the original plan being revised is determined by the locality principle and by the preference for altering the non-executed portion of the plan, inspired to resource-boundedness considerations.

The retraction procedure in SPA can be compared with the Collapse-Plan procedure in our algorithm; however, the retraction procedure in SPA makes use of the structural knowledge encoded in plans, while the Plan Repair algorithm does not take the structure of the current plan into account, being based on enablement relation compiled in plan schemata in the form of decomposition relations. However, as it will be discussed in section 3.5, the interactions between the steps of a plan are reflected in the expected utility: in this way, structural information is still available to the Plan Repair algorithm, though in an indirect form.

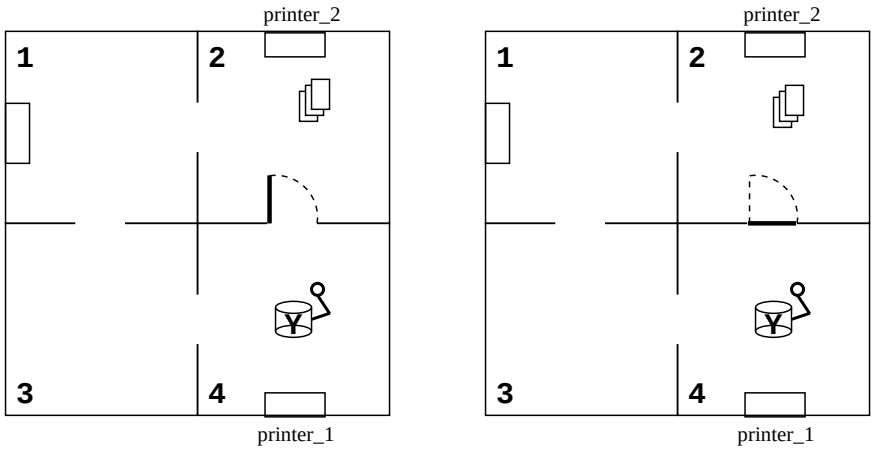

Figure 3.10: A representation of the world from Y's subjective point of view and from an objective point of view. Notice that Y wrongly believes that the door between room 4 and 2 is open

3.4.2 Plan Repair Examples

For example, consider the micro-world depicted in figure 3.10, constituted by the office domain inhabited by two agents, X and Y. The abilities of X and Y are different: X is not able to open doors, while Y, by having a mechanical arm, is able to open doors. Figure 3.11 represents a portion of the library describing the recipe for Y getting the mail from room 2 to room 1.

In the following example, Y has the goal of getting the mail from room 2 to room 1, and to this end it has devised the following plan:
GO-Y-4-2-door TAKE-MAIL-Y GO-Y-2-1 PUT-MAIL-Y

The first action (GO-Y-4-2-door) consists in going from room 4 to room 2 through the (open) door; then, the agent picks up the mail (TAKE-MAIL-Y), which is supposed to be situated in room 2, and goes to room 1 (GO-Y-2-1), where it puts the collected mail on the boss' desk (PUT-MAIL-Y). However, the plan is unfeasible, since it rests on Y's wrong belief that the door between room 4 and room 2 is open (see figure 3.10).

After executing the first step of the plan, GO-Y-4-2-door, Y performs a sensing action concerning the effects of the last step, and it realizes that the remaining plan is not feasible: the effects of GO-Y-4-2 being that Y is still in room 4 (as Y executed the action in a different context than it believed, it bumped on the closed door and remained in room 4), the following action, TAKE-MAIL-Y, would fail if executed - as the mail is not in room 4 - and so would do the subsequent steps of the original plan, GO-Y-2-1 and PUT-MAIL-Y. So, when Y anticipates the execution of the remaining steps in the updated representation of the world (TAKE-MAIL-Y GO-Y-2-1 PUT-MAIL-Y), it realizes that the remaining plan is

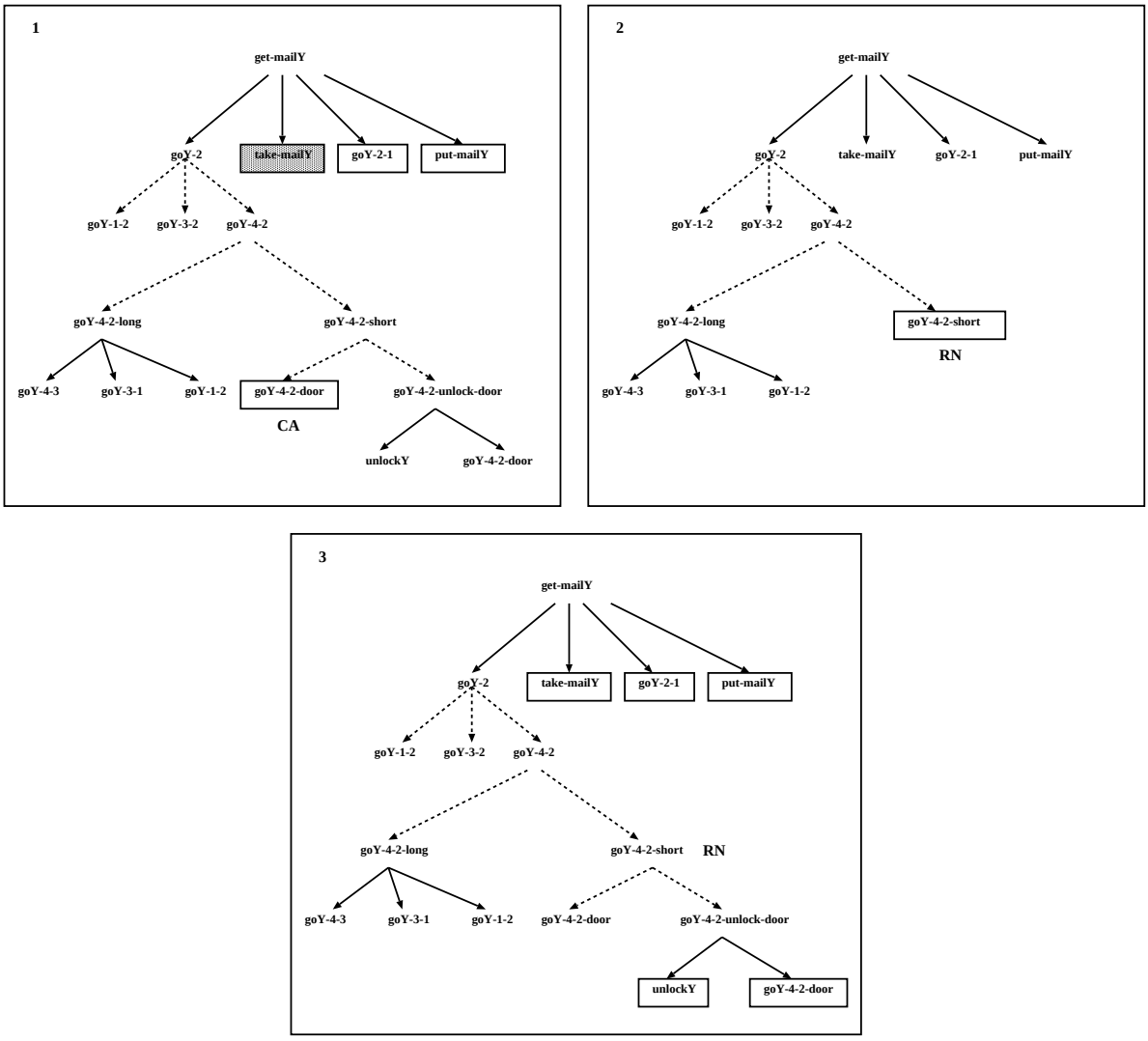

Figure 3.11: A representation of the steps performed by the repair algorithm on the action hierarchy. The original plan (1); a node is selected for revision (RN); a different instantiation of the RN has been chosen (3).

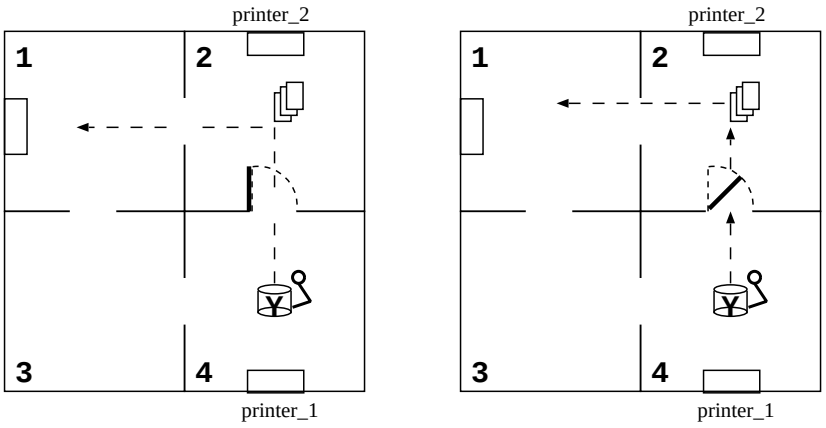

Figure 3.12: The plan of the agent Y before replanning (left) and after replanning (right).

unfeasible, and enters a replanning phase.[16]

The algorithm first selects the next action as the focused action, TAKE-MAIL-Y. Then, given the portion of plan hierarchy reported in the first box of figure 3.11, the algorithm climbs the action hierarchy and selects the node GET-MAIL-Y as the lowest decomposition ancestor (LDA) of the focused step. The GET-MAIL-Y action is the root of the sub-plan constituted by the actions GO-Y-4-2-door TAKE-MAIL-Y GO-Y-2-1 PUT-MAIL-Y, which becomes the candidate sub-plan.[17] Once the candidate sub-plan has been identified, the algorithm examines the candidate steps, starting from the current step, TAKE-MAIL-Y, and its right siblings. For each candidate step, it looks for a revision nodes, i.e. a more general action on the path connecting the candidate step to the LDA; none of the right siblings of TAKE-MAIL-Y satisfies this condition, as they all are direct child of the LDA.

After examining TAKE-MAIL-Y itself and all its right siblings, the algorithm examines the left siblings, starting from the most recent one: when inspecting the nodes on the path connecting GO-Y-4-2-door to the LDA, GET-MAIL-Y, it finds a node constituted by the abstract action GO-Y-4-2-short: this node becomes the revision node. The revision node, being an abstract action that has an alternative instantiation with respect to the one which appears in the plan, is a good starting point for finding an alternative plan (GO-Y-4-2-door becomes the RN, or revision node - see the second box in figure 3.11).

[16]Notice that Y does not specifically reason neither on the failure of executed actions, nor on its causes. Rather, the replanning algorithm simply looks for an alternative plan, by giving precedence to the non-executed steps of the plan.

[17]This example, which has been chosen for its simplicity, is a degenerated example, as the root of the local subtree coincides with the root of the hierarchy; however, it is easy to imagine examples where bringing the mail to a certain office is a small part of a much larger plan, which is not affected by the way the mail arrives there.

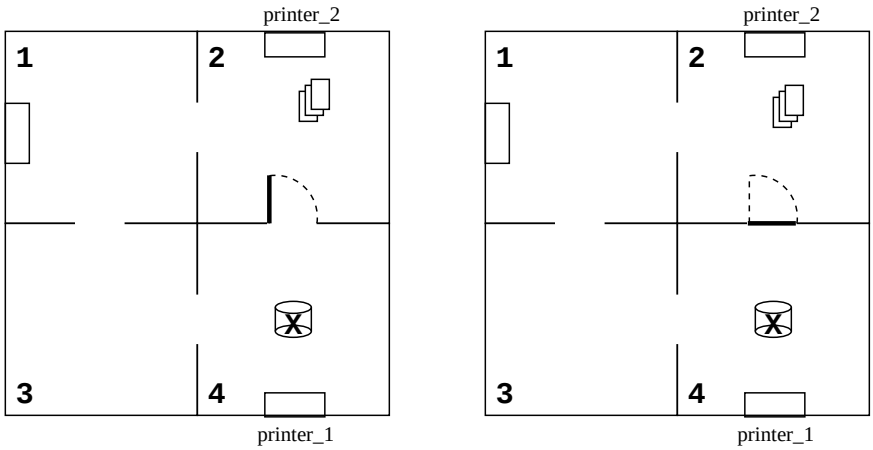

Figure 3.13: A representation of the world from X's subjective point of view and from an objective point of view. Notice that X wrongly believes that the door between room 4 and 2 is open

Consequently, the candidate step being examined (GO-Y-4-2-short) becomes the candidate for revision. The planner is called on the revision plan (GO-Y-4-2-short TAKE-MAIL-Y GO-Y-2-1 PUT-MAIL-Y), where the candidate step has been substituted for RN, the revision node. The planner outputs the alternative plan (UNLOCK-Y GO-Y-4-2 TAKE-MAIL-Y GO-Y-2-1 PUT-MAIL-Y) (see the third box in figure 3.11 and in figure 3.12), which is a feasible plan given the current world.[18]

Now consider the situation in which X has the same goal, getting the mail from room 2 to room 1, but the same misbelief as well: X 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 5.3). Remember that, differently from Y, X does not have the ability to open the door between 4 and 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, as represented in the first box of figure 5.4:
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-

[18]When refining the plan (GO-Y-4-2-short TAKE-MAIL-Y GO-Y-2-1 PUT-MAIL-Y), the planner compares the alternative plans (UNLOCK-Y GO-Y-4-2 TAKE-MAIL-Y GO-Y-2-1 PUT-MAIL-Y) and (GO-Y-4-2-door TAKE-MAIL-Y GO-Y-2-1 PUT-MAIL-Y), but the latter is discarded since its expected utility is now believed to be too low.

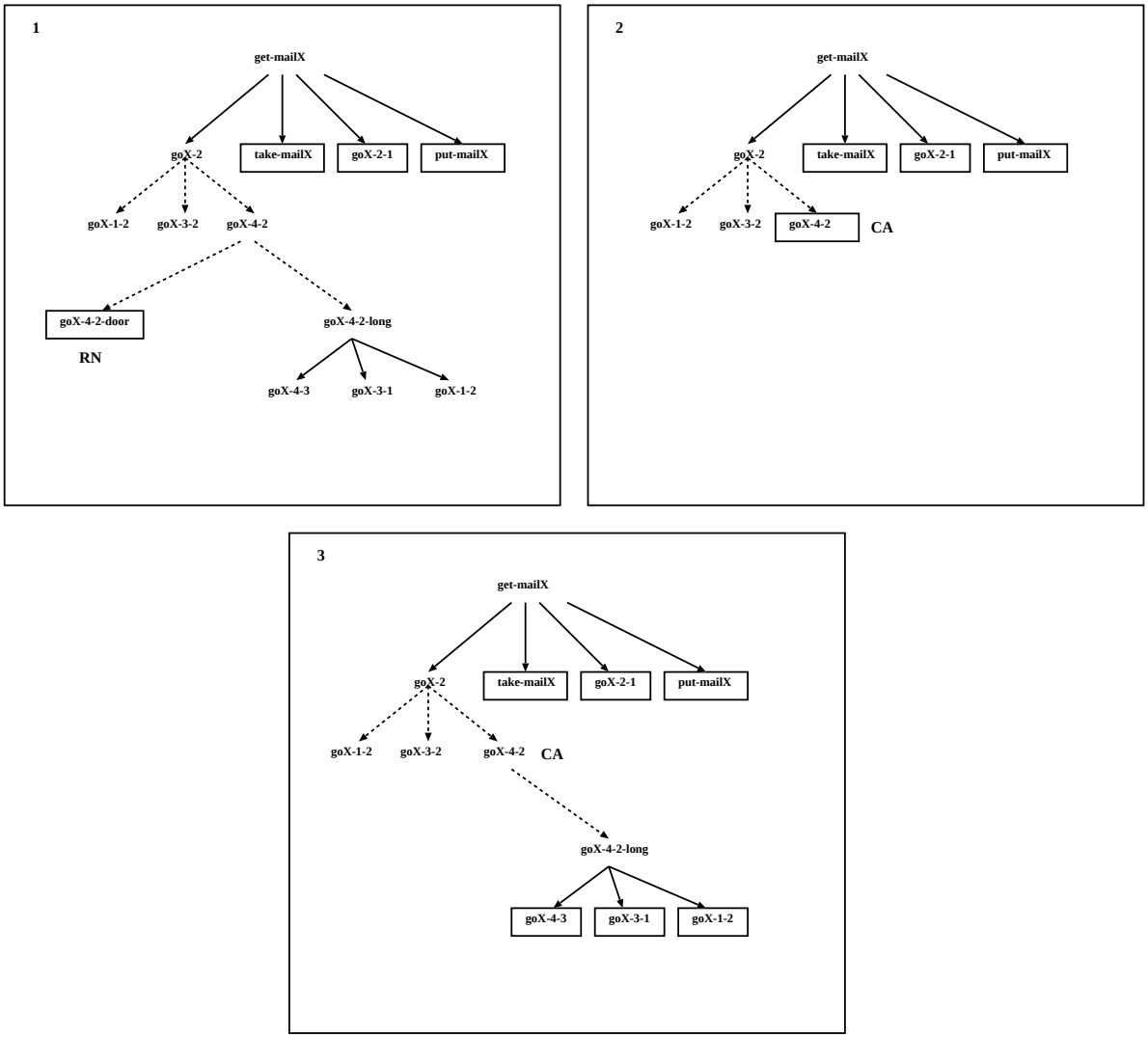

Figure 3.14: A representation of the steps performed by the repair algorithm on the action hierarchy given X's plan. The original plan (1); a node is selected for revision (RN) ; a different instantiation of the RN, the sequence of steps composing the action GOX-4-2-long, has been chosen (3).

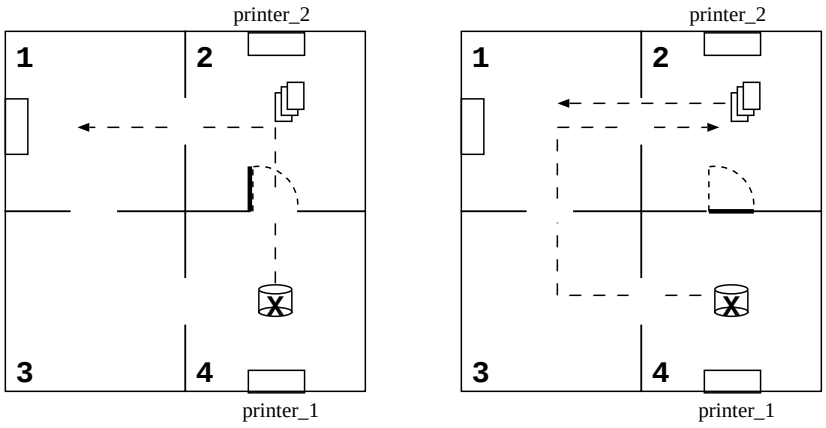

Figure 3.15: The plan of the agent X before replanning (left) and after replanning (right).

X-4-2-door (see the second box in figure 5.4). 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 5.4 and in figure 5.5):
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.

3.4.3 The Plan Expansion Algorithm

We now describe in more detail the *Plan Expansion sub-module*, that the replanning module invokes on the failed plan if the plan repair module has not been able to devise a new feasible plan. Though remaining within the boundaries imposed by the use of a refinement planner, this component expands the current plan by adding new actions to it, in the style of partial order planners, as proposed by [Haigh and Veloso, 1996] and [van der Krogt et al., 2000].

The Plan Expansion Algorithm first identifies a set of actions whose preconditions are not met, then tries to enable them by inserting new steps in the plan. However, the representation style of actions in the agent model does not match the notion of "preconditions" and "effects" in classical planning[19]; the representation of actions consists of an "objective" description of their conditional effects, which is required for evaluating the expected utility of plans, without referring to a specific notion of intended effect.

[19]The conjoined use of sequential abstraction and inheritance abstraction allows a compact representation of plan recipes, but, at the same time, they leave out the explicit representation of precondition enablement relations between actions.

In order to overcome this gap, and to be able to reason on the failure of an action with respect to its intended effect, the action representation contains the explicit information about the intended effect, called *action goal*, as illustrated in section 3.2.2. Given the set of condition-effect pairs which appear in the description of an action and the information about which effect is the intended effect, it is possible to infer what condition determines it, or, in other words, what condition is a precondition to that effect.

Given the intended effect of an action (action-goal), in order to identify the corresponding precondition (the condition whose satisfaction would lead to the achievement of the action goal), the agent reasons in the following way:

1. The agent anticipates the execution of the action in the given initial world;

2. If the action goal does not hold in the resulting world, it examines the initial world to find the action conditions (there may be several alternatives) that are not verified in that world.

3. For each non verified condition, the agent builds a world where it is verified, and performs the action again in that world.
   If the action goal holds, a "pre-condition" has been found.

The plan expansion algorithm works in the following way:

1. First, the algorithm identifies the steps whose preconditions are not enabled, by resorting to the strategy illustrated above.

2. Then, for each precondition, it searches the library of plan fragments for elementary actions whose effects match the precondition. The new actions are inserted in the plan as new plan steps, possibly generating a feasible plan. The algorithm core is the precondition-effects match, which resembles the basic mechanism of partial order planners like UCPOP [Penberthy and Weld, 1992] (see also [Weld, 1994]).

3. Action sequences which don't constitute a feasible plan are further modified by trying to restore the invalid causal links (see [Weld, 1994]) they may contain by performing a further step of plan expansion. However, the recursion is limited to a second modification cycle in order to reduce the impact of modifications on the original plan and the cognitive load imposed by this reasoning style.

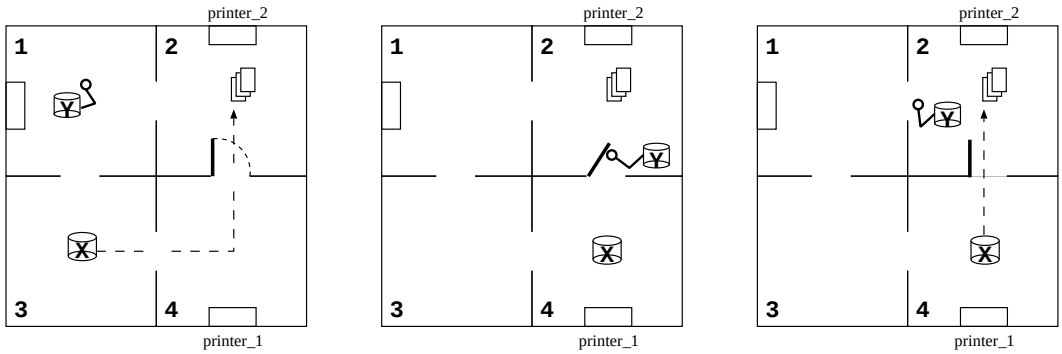

Figure 3.16: X's plan before knowing that the door between room 4 and 2 is closed (left), and X's plan after its expansion: the action of asking to Y to open the door has been added to the initial plan.

3.4.4 Plan Expansion Example

In order to illustrate how the Plan Expansion algorithm works, we resort again to an example in the office micro-world. Consider the simple situation where agent X has the goal of being in room 2, and currently is in room 3; to this aim, X has devised the plan of going form 3 to 4 and then from 4 to 2 (see figure 3.16, left). This time, X correctly believes that the door between 4 and 2 is open: however, after it executes the step of going from room 3 to room 4, X's plan becomes unfeasible, due to the fact that the door has been closed in the meantime. However, suppose that X does not know any alternative feasible plan, as it is temporarily subject to a prohibition to transit from room 3 to room 1, due to some work in progress in that area of the office.

Now, we will consider the kind of reasoning performed by Y following the Plan Expansion strategy. Remember from section 3.4 that, after sensing the world state that results from the first plan step (going from room 3 to 4), X has updated its subjective view of the world and has verified whether the remaining plan would succeed by anticipating its execution. At this point, Plan Repair has already been unsuccessfully attempted, so X resorts to Plan Expansion.

Having found that the action of going from room 4 to room 2 (the only one left for execution) would not succeed, X tries to find out why. The action of going from 4 to 2 has two alternative conditional effects: Y is in 2 after executing it only if the door was open, and is still in room 4 if the door was closed. The second condition is verified, while the first one is not: so X *tries to figure out* what it would happen if this condition were verified, following the simple algorithm described above. By doing do, X realizes that it would be able to reach room 4 if the door were open: the reason why it is not be able to go from room 4 to 2 is that the door is currently closed.

Having discovered what action would fail and why, X now tries to enable the action condition that the door is open: in order to do so, it searches the action

library for an action whose effect matches the false condition (the precondition for achieving the action goal), and finds the action of *asking Y to open the door*. This action is not part of the plan schema for X going from room 4 to room 2 (so it was not be considered neither in the planning nor in the subsequent replanning phase), and, what's more, is a quite expensive action, which only succeeds with a probability of 80%: however, it is the only suitable one in the present situation.

So, X expands the initial plan by inserting the asking action in the right position (the only one available is right before the step of going from room 2 to room 4, see figure (see figure 3.16, center)). Finally, X simulates the execution of the expanded plan (ask-X-to-open-door - go-Y-4-2); then, as the plan appears to be feasible, X starts the execution.[20]

3.5 The Plan Repair Algorithm Improved

In this section, we propose a way to improve the efficiency of the replanning algorithm described above. The Plan Repair algorithm, as it is, is based on the assumption that interdependencies between plan steps tend to be local, i.e. that they are circumscribed to the decomposition of the same decomposable action, which is normally situated at a low level of the action hierarchy.

However, the failure of a plan step sometimes affects one or more plan steps which are not local to the failed step with respect to the decomposition hierarchy: the action constituting the failed step and the actions constituting the affected steps still have a common ancestor at some level of the decomposition hierarchy, but not at a low level. If this is the case, the Plan Repair Algorithm will call itself a number of times before climbing the action decomposition hierarchy up to the appropriate level: each time, it will perform the partialization process and the subsequent refinement in vain, without any hope to find an alternative feasible refinement of the initial plan at that under-determinacy level.

In order make the algorithm more efficient, a step has been added to the Revision algorithm to avoid performing the revision process on the input plan until the appropriate level of under-determinacy has been reached for the input plan. This step is based on the properties of partial plans: in particular, the expected utility interval associated to a partial plan encompasses the expected utility intervals of all the plans which constitute its alternative instantiations i.e., all its possible refinements. Starting from the observation that, until the appropriate level in the decomposition hierarchy has been reached, there is no way to restore the expected utility interval to an acceptable level (no alternative refinement would restore the enablement link in any way, by bringing back the expected utility to an acceptable level) we use this information to assess if the

[20]Here, we are not concerned with the interleaving of action execution between two or more agents.

current working level is the appropriate one.

The following step has been inserted in the Revision function: after the Lowest Decomposition Ancestor of the input plan has been set, it is substituted for the plan steps it subsumes (the candidate sub-plan) and the resulting plan is projected to compute its expected utility. Recall that the Revision process, given a candidate sub-plan, tries to find an alternative instantiation of the sub-plan steps, i.e., it re-refines the initial plan by partializing increasing portions of the candidate sub-plan in search for a feasible plan. If the upper bound of the expected utility of the input plan is lower than the upper bound of the initial plan, this means that no alternative instantiation of any of the candidate steps (which constitute the decomposition of the LDA) would restore the enablement link, and so it is not worth trying.

The added projection step makes the replanning algorithm more efficient, much in the same way in which the pruning heuristic uses the information about the expected utility to make the planning algorithm more efficient; at the same time, it counterbalances the lack of explicit information about the interdependencies between actions in the action hierarchy.

```
function REVISION (plan, focused-step, sub-plan, world, goal, old-EU)
    LSR := SET-LDA (sub-plan);
    test-plan := Collapse-Plan (plan, sub-plan);
    if test-plan.EU > old-EU
     then
       begin
        ordered-candidates := ORDER-CANDIDATES (sub-plan);
        for each candidate in ordered-candidates do
          revision-step := FIND-REVISION-STEP (LSR, candidate);
          revision-plan := SUBSTITUTE-STEP (sub-plan, revision-step);
          new-plan := REFINE (revision-plan, world)
          if IS-FEASIBLE-PLAN (new-plan, world, goal, old-EU)
            then return new-plan
            else sub-plan := revision-plan
          end if
        end for each
       end
     else return test-plan
end function
```

For illustrating the gain in efficiency determined by this modification, we resort again to figure 3.9, that we here propose again here in figure 3.17. Suppose, for example, that the execution of the step B' fails, and that B' was supposed to enable the step constituted by action D; in order to make the plan feasible, B' must be replaced with B'', whose precondition are satisfied by the current

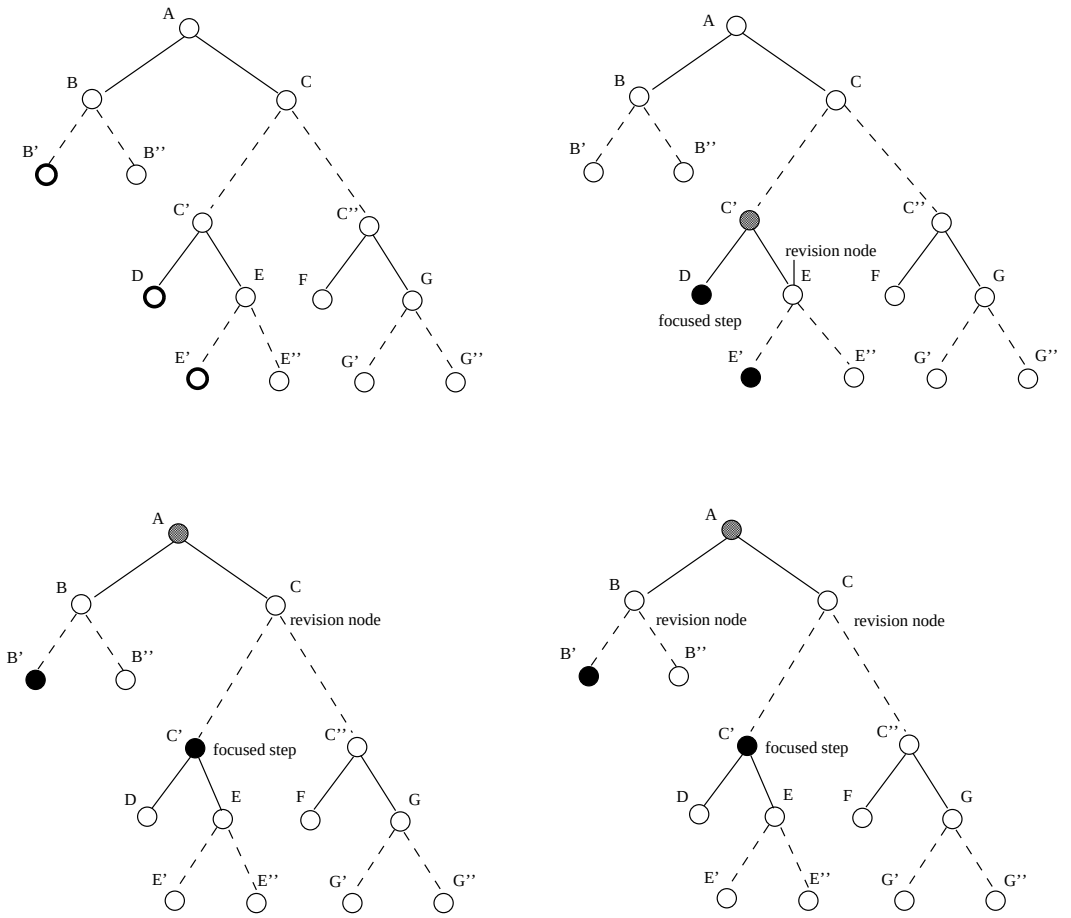

Figure 3.17: (*repetition of figure 3.9*) A graphical representation of the plan repair process on a generic action hierarchy. (1) represents the initial plan; (2)-(3)-(4) represent the subsequent phases of the plan-repair process. Black nodes represent the actions constituting the candidate steps in the candidate sub-plan, while the grey nodes represent the Lowest Decomposition Ancestor.

context, contrarily to B'. In order to perform this substitution, the replanning algorithm must climb the action hierarchy until it reaches the topmost action, A, because the enablement relation we are concerned with holds at the level of its decomposition.

Only when the A becomes the LDA (figure 3.17 (3)), the projection of the plan having A as the only step reveals that this plan subsumes an feasible alternative to the failed plan (the upper bound of the expected utility interval rises anew). Then, when the leftmost portion of the plan with respect to the focused step (C') is explored, an alternative instantiation for B is found, i.e., B''.

The situation would be the same if the plan could be improved by an alternative refinement of node C'' (for example, suppose that the failure of B' is

compatible with a new continuation constituted by the execution of steps F and G', or F and G''); again, the refinement would be attempted only when action node A becomes the LDA.

In this cases, the addition of the projection step allows the algorithm to skip the refinement when the LDA is action node C'. When the hierarchy is deep, the modification is more likely to be useful; however, if the hierarchy is shallow, and/or the locality principle holds, it requires only a small amount of additional work which easily compensated by the more difficult cases.

Chapter 4

Interactional framework

4.1 Introduction

In the previous chapter, we described an architecture for agents who react to a dynamic environment by alternating execution and re-deliberation phases in response to a dynamic environment. In this chapter, we focus on a specific aspect of the environment, constituted by the presence of other agents, and we introduce a framework for modelling the interaction with other agents.

This interaction is regulated by a set of rules - more or less formal, partly aimed at managing coordination and concurrency issues, and partly aimed at defining inter-agent relations. Here, we don't investigate the nature of these interactional rules: instead, we take their existence for granted, and focus on the reasoning that an agent performs in order to deal with them.

In order to model the interactional capabilities of an agent, we rely on a framework where decision-theoretic planning is integrated with the notions of *adoption* and *anticipatory-planning* ([Boella et al., 2000a], [Boella and Damiano, 2000]).

Adoption ([Castelfranchi, 1998]) is the link between the agent's internal state and the goals of other agents in social settings. We resort to the notion of adoption to allow an agent to take into account external goals, focusing on situations where an agent's behavior is influenced by constraints deriving from social norms.

The *anticipatory planning* process takes place during the deliberation phase and is constituted by a *look-ahead* step in which the agent computes the reaction of the partner to her own behavior, in a game-theoretic fashion: first, the outcome of each alternative course of action available to the agent is computed; then, for each outcome, the partner potential reaction is simulated, and the agent utility is evaluated on the set of states thus obtained. An similar form of anticipatory reasoning has been used by [Gmytrasiewicz and Durfee, 1995] and [Gmytrasiewicz and Durfee, 2000] to model coordination and communication in multi-agent systems.
In short, the *anticipatory planning* process is the following: the planner produces

a set of alternative plans; the outcomes that would result from the execution of these plans are computed and a look-ahead step is performed on them. Finally, the expected utility of each outcome is evaluated using a multi-attribute utility function which embodies the agent preferences. The outcome of plans is not assumed to be deterministic: the utility function accounts for the fact that each alternative plan is associated with a set of alternative outcomes, with a certain probability distribution.

The chapter is organized as follows: first, we introduce the interactional framework (section 4.2); then, in section 4.3, we show how it can be exploited for modelling the agent's reasoning about social aspects in an interaction, namely *social goals*; finally, in sections 4.3.2 and 4.3.3, we illustrate the functioning of the model by means of examples.

In the following chapter this framework will be integrated within the reactive agent architecture to model the reactivity to norms in general.

4.2 Social Aspects in the Intention Formation

When an agent is immersed in a social context, where she interacts with other agents, her deliberative capabilities face a more complex situation with respect to the situation presented in the previous chapter.
In the previous chapter, we focussed our attention on how an agent can account for external changes in a dynamic environment; in a social context, the situation is more complex, yet more predictable; while the environmental changes may be completely arbitrary, the way other agents may affect the goal-directed behavior of an agent can be at least partially computed, as it depends - to a certain degree - on the agent's behavior itself.
The reason for this circularity is that, since the agent's world is populated by other agents, her behavior may affect the subsequent behavior of other agents in some way. For this reason, in an interactional context, whenever an agent believes that a course of action is likely to give rise to a response of any kind by her fellow agents, she should try to figure out this response in order to evaluate the expected utility of that course of action.

A special occurrence of this situation arises when an agent has explicitly asked another agent for help by performing a communicative act of any kind. If this is the case, it is possible that the requestee becomes committed to the requesters's goal and forms the intention to execute a plan for satisfying it, even if that plan does not yield any direct utility to her; but it is also possible that she does not do anything to satisfy the partner's goal.
If the agent evaluated the utility of a plan for achieving a goal that has been put forth by another agent only on the basis of its immediate utility, she would never choose that plan (at least, not in non-cooperative settings): executing a plan for achieving another agent's goal is not advantageous by itself, as it would

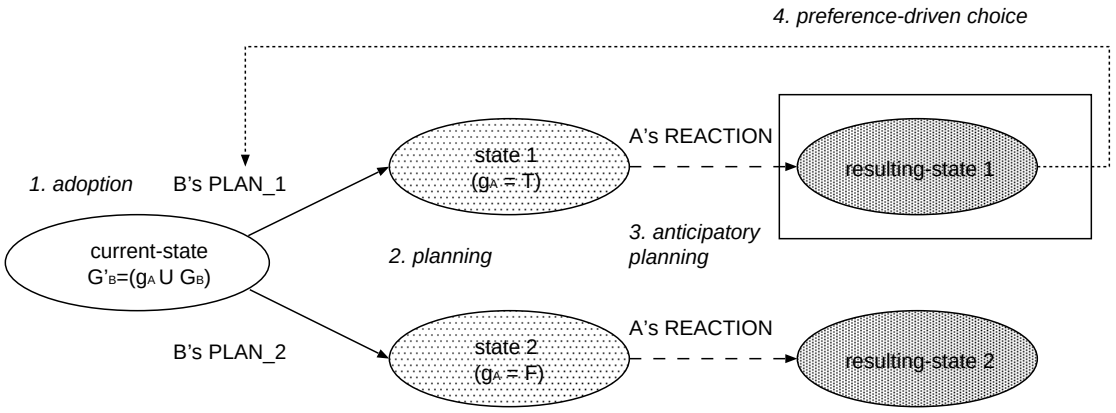

Figure 4.1: The intention formation process in interactions

only waste resources and time. The reason why the agent may adopt a goal of another agent - and possibly becomes committed to it later - is the fact that the satisfaction of that goal can have an indirect utility for her in the light of the other agent's reaction.
In order to evaluate how the partner's reaction to her decision affects her own preferences, the agent evaluates the utility of a plan by considering the world states resulting from the partner's reaction (one-level lookahead), both in case she has committed to the partner's goals, and in case she has decided that they are not worth satisfying.

The general *mechanism of intention formation* with adoption and anticipatory planning is composed of the following steps (see figure 4.1):

1. *adoption*: if A communicates to B a goal g_A which she wants B to achieve (or B becomes aware of a goal g_A), then the current set of B's goals, G_B, becomes G'_B, the union of $\{g_A\}$ and G_B.
2. *planning*: B builds the set of plans P_B aimed at achieving (all or some of) the goals in G'_B (in this way, the plans achieving also g_A are compared with those which do not). expand!!
3. *anticipation*: from the state resulting from each plan p_i in P_B, B considers the possible reaction of A: the world state resulting from the reaction becomes the new outcome of p_i.
4. *preference-driven choice*: B chooses the p_i in P_B whose outcome maximizes her utility.

The steps from 2 to 4 will be illustrated in more detail in section 4.2.2.

4.2.1 Goal Adoption

We employ the term adoption in a weak sense: by adoption of a goal, we refer to taking the goal into account as a new option, without implying the actual commit-

ment to the new option, similarly to what proposed by [Conte and Castelfranchi, 1995]. In general, it may happen that the agent who performs the adoption will never do anything to satisfy the adopted goal.

Here, the ability of an agent to predict the potential reactions of another agent is exploited to decide whether it is worth for her to commit to the satisfaction of the other agent's goal.

However, the fact that B takes into account g_A we say that B is *potentially-cooperating* with A. This notion is inspired to that of *potential-intention* of [Grosz and Kraus, 1996]: even if B is not properly cooperating with A, she is, in some form, cooperating: though it may happen that she will never do anything to satisfy A's goal, the agent allocates times and planning resources to consider the alternative of adopting the partner's goal.
In case B chooses a plan that leads to the satisfaction of the partner's goal, B is *cooperating* with A. In case that A is aware of the formation of this intention, then the agents are cooperating to a shared goal (see [Boella et al., 2000a]).

4.2.2 Anticipatory Planning

The planning component we refer to is the decision-theoretic planner presented in the previous chapter. This planner builds on the DRIPS system [Haddawy and Hanks, 1998], a hierarchical planner which merges decision theory with probabilistic planning techniques. Actions are organized along decomposition and abstraction hierarchies to form a library of plan schemata and can have non-deterministic effects, which induce a probability distributions over the outcomes of plans. The agent uncertainty about the world is also modelled, by introducing in the framework another source of non-determinism.
In order to evaluate how promising a given plan is, the planner exploits a multi-attribute utility-function, which computes the expected utility of each outcome of the plan by aggregating simpler utility functions associated with the preferences of the agent.

The implementation of the anticipatory planning has required two main extension of the agent model described in the previous chapter:

- With respect to the planner described in the previous chapter, some further modifications to the core planning mechanism of DRIPS have been necessary in order to implement the anticipatory planning, especially for what concerns the heuristic evaluation of partial plans. In particular, the pruning step has been moved to the end of the anticipation process, so that plans which may be advantageous only with respect to the inter-actant's reaction are not pruned prematurely.

- Since the planning agent has to predict the reactions of the partners, it must be endowed with the beliefs concerning her partner's beliefs and preferences. So, beside her own beliefs, goals, intentions and preferences, the

agent now has some beliefs concerning the partner's beliefs, goals, intentions and preferences.

In short, the anticipatory planning algorithm is the following:

- The planner (playing the role of A), expands the current state S by applying all alternative plans for $\beta_m^{x,A}$, thus producing the states S_1, S_2, ..., S_r (where r is the number of different plans for $\beta_m^{x,A}$).
- This set of states is transformed in the set of the same states as viewed by B, S'_1, S'_2, ..., S'_r.
- The planning process is started on each state S'_m ($1 \leq$ m $\leq$ r), from the perspective of her partner B (i.e. trying to solve her current task $\beta_h^{x,B}$).
- This produces a set of sets of states $SS' = \{\{S'_{1,1}, \ldots, S'_{1,n_1}\}, \{S'_{2,1}, \ldots, S'_{2,n_2}\}, \ldots, \{S'_{n_r,1}, \ldots, S'_{r,n_r}\}\}$.
- The utility function of the partner is applied to these states, and the best state of each subset is identified: SS'_{best}=$\{S'_{1,best(1)}, S'_{2,best(2)}, \ldots, S'_{n_r,best(n_r)}\}$. These states are the ones assumed to be reached by B's best plan, for each of the possible A initial moves.
- The utility function of the agent is applied to the states $S_{k,best(k)}$ ($1 \leq k \leq r$) from A's point of view. This models the perspective of A on what could happen next.
- The best one of these states is selected ($S_{max,best(max)}$). This corresponds to the selection of the best plan for $\beta_m^{x,A}$ of A (i.e. R^{max}).

The algorithm above is just a modification of a two-level min-max algorithm: actually, it is a max-max, since at both levels the best option is selected, although at the second level it is evaluated from B's perspective. As in min-max, A, when predicting B's behavior, assumes that her partner is a rational agent, i.e. that she will choose the plan that yields the highest utility for her.

The problem of simulating another agent's planning is very difficult. For instance, in some situations, B could not be aware of P^x effects. In our implementation, we adopted the simplification that the initial state is shared by the agents, while, during the planning phase, B's knowledge of a state is updated in A's beliefs by an action of A only with the effects which are explicitly mentioned as believed by B (e.g., the result of a communicative action having B as receiver). However, the treatment of the changes of the beliefs of the partner would deserve a more accurate model, as the one proposed for multi-agent systems by [Isozaki and Katsuno, 2000].

```
plan(A, Plan_A, B, Plan_B, world)
  begin
    /* refinement of Plan_A */
    refined-plans := refine-plan (plan_A, A, world);
    final-worlds := nil;
    /* for each possible outcome of each possible alternative */
    for-each plan in refined-plans
    begin
      /* outcomes of a plan of A from the initial worlds
      (their probability sums to one)*/
      for-each world in resulting-worlds(plan, initial-world)
      begin
        /* save the probability of the outcome of plan */
        prob := world.prob;
        /* simulate B planning from the outcome of plan */
        world.prob := 1;
        alternative-plans-B := plan(plan_B, B, world);
        /* select best plan from B's point of view
        by computing their expected-utility from B's point of view */
        chosen-plan-B := best-plan-EU(alternative-plans-B, B, world);
        resulting-worlds := resulting-worlds(chosen-plan-B, B, world);
        /* restore the probability of each outcomes w */
        for-each w in resulting-worlds
        begin w.prob := w.prob * prob; end
        /* the probability of worlds in final worlds will sum to one */
        final-worlds := final-worlds + resulting-worlds;
      end
      /* for each A's alternative compute its expected utility
      from A's perspective */
      plan.EU := compute-EU(plan, final-worlds, A);
    end
    /* eliminate plans that are not promising */
    return(eliminate-plans(refined-plans, A));
  end
```

Figure 4.2: The function of the anticipatory planner that, given a plan, performs a step of refinement and discards unpromising alternatives by simulating the partner's reaction.

Some more words must be devoted to the probability that an effect holds after the execution of a plan P. Note that if a plan P of A makes a proposition *Prop* true only with probability *p(Prop)* the simulation of B's planning phase must be carried on starting from both "possible" worlds resulting from the execution of P (i.e. one where *Prop* is true and one where *Prop* is false).
Therefore, what B would plan if *Prop* were true and if *Prop* were false are simulated separately; since also B's recipes may involve uncertain effects, we adopted the solution of multiplying the probability of the different outcomes of B's actions with the probability of B's initial states in order obtain the set of worlds representing the possible outcomes of B's reactions to the plan P.

4.3 Socially Aware Agents

4.3.1 Social Goals in Interactions

In order to show how the anticipatory planning framework proposed above can be exploited to model the interactional capabilities of an agent, we now introduce an explanation of some conversational phenomena that is based on the notion of *social goals* ([Boella et al., 2000b]). We believe that conversation constitutes an appropriate domain for testing the validity of a general model of interaction, as it is a well-studied domain both in linguistics and computational linguistics, by being at the same time the focal center of several applicative efforts in the field of human-computer interaction.

According to the pioneering analysis of interactional exchanges conducted by Goffman (([Goffman, 1967], [Goffman, 1981]), agents are aware of the fact that their actions have social effects, like conveying some information about their character and about their attitude towards the partners, i.e., their social *face*: "An act is taken to carry implications regarding the character of the actor and his evaluation of his listeners, as well as reflecting on the relationship between him and them" ([Goffman, 1981], p. 21). As a consequence, agents are very cautious in the use of the expressive means they have at disposal, namely verbal actions: besides monitoring the partner's reactions, they try to anticipate them with the aim of *not offending the partner*.

In general, the preference for not offending is motivated by the requestee's goal of displaying a good-tempered acceptance of the request itself: in [Goffman, 1981]'s terms, interactional exchanges are subject to a set of "constraints regarding how each individual ought to handle himself with respect to each others, so that she not discredit her own tacit claim to good-character or the tacit claim of the others that they are persons of social worth (...)" (p. 16).

A similar analysis is at the basis of the work by Brown and Levinson ([Brown and Levinson, 1987])on politeness in dialogue; in this work, politeness is explained with reference to the notion of *face*: for example, a request threat-

ens the hearer's wants of freedom of action; therefore, direct requests in imperative form should be avoided. The importance of politeness for obtaining naturalness in human-computer conversation has been remarked on recently by [Brennan and Ohaeri, 1999].

In the model we propose, the *social goal* of not offending the partner[1] corresponds to a *preference* of the agent: it doesn't constitute an input to the agent's planning, but, as it plays a role in the evaluation of the expected utility , it contributes to plan selection, by promoting the plans which do not have offending as a consequence. This is in line with [Goffman, 1967]'s claim that "Ordinarily, maintenance of face is a condition of interaction, not its objective" (p.12).

Social preferences influence behavior in an indirect way: in order to evaluate the effects of an action on her interlocutor, an agent attempts to do a prediction of her reaction by performing some kind of anticipation of the interlocutor's reaction, that we model by means of the *anticipatory planning*. This tentative prediction allows the agent to keep the partner's possible reaction into account when planning her behavior. So, social preferences don't trigger the deliberation process, like goals; they intervene during the action selection phase, by leading the planning agent to choose the actions which minimize the offence to the partner.

In order to illustrate how the interactional framework works, we will resort to the paradigmatic example constituted by request-response pairs.

Again, the model we propose is based on [Goffman, 1967]'s analysis of conversational exchanges. According to this analysis, any **request** contains a potential offence, as it presupposes a dominant position of the requester and constrains the requestee's freedom of action. According to [Brown and Levinson, 1987], the requester, being aware of this, tries to nullify this potential offence by using polite forms which don't threaten the face of the requestee and let her - at least superficially - an option to refuse.

At the same time, any **response** is likely to contain a *relief* that displays the requestee's acceptance of the requester's effort: not only requests for action are granted by the requestee "with a show of good spirit" ([Goffman, 1981]), but also, if they are to be turned-down, "a mollifying reason" is given as a *remedy* to the potential offence stemming form the refusal (see also [Levinson, 1983]). Any refusal constitutes in turn a potential offence to the requestee's face, and sets up the social need for the refusing agent to nullify this potential offence.

In general, the situation a requestee is faced with is constituted by the choice between the alternative of satisfying the requester's conversational or non conversational goals and the alternative of going on with her own activity. The fact that the partner becomes offended is not modelled as a direct effect of an action of the agent; instead, during the planning phase, the agent makes a tentative prediction of the partner's attitude in the state where she is faced with a refusal,

[1] In [Clark, 1996]'s terminology, being polite is an *interpersonal goal*.

in order to evaluate how this state complies with her preference for not offending: the partner is offended as a result of her reaction to the agent's refusal. The effect on the requester is evaluated by the planning agent by means of the anticipation of her reaction.

As noticed by [Airenti et al., 1993], a dialog participant, even when she does not commit to the domain goals of her partner (i.e., she doesn't cooperate *behaviorally*), typically continues to cooperate at the *conversational* level.
The analysis introduced so far does not explain why requests at behavioral level seem to pose less constraints on the addressee, if compared to requests at conversational level: provided that the interactants don't have shared goals, it is a matter of fact that it is easier to refuse a request for money. In particular, conversational goals often force the hearer to satisfy them: it is aggressive not to answer at all or to ignore the speaker.
The reason why paying attention to people, listening and understanding are not easily refused is that they are low cost actions, or "free goods" that no one can refuse without threatening the speaker's face and offending her. A refusal at the conversational level - ignoring a potential partner and not even responding to her verbal request - constitutes a menace to the face of the requester, so it is hardly justified. Up to this point no explicit obligation to act is established; rather, the social pressure determined by a request depends on the cost of the requested action: if the cost of the action is low (e.g., a conversational action), the refusal to execute it can be motivated in the requester's eyes only by a requestee's negative attitude towards her. So, the requester, as a result of her ability to infer the requestee's reasoning, will be offended by a refusal; the preference for not threatening the face of the partners and preserving one's own social face normally makes the utility of offences negative, thus leading requestees to avoiding refusals. At the same time, this analysis, by making explicit the underlying motivations for the preference for a certain type of response, accounts for the existence of preferred and dispreferred second turns in adjacency pairs.

The preference for not offending holds also in the circumstances where an agent is forced to refuse her cooperation by the *impossibility* of executing the appropriate action to achieve the partner's goal. However, if this is the case, the requestee has to cope with the additional fact that a simple, negative answer can be mistakenly taken to count as a refusal to cooperate at all, as exemplified by the following exchanges (compare the oddity of example [1] with the naturalness of examples [2] and [3]):

[1] A: *Have you got a cigarette?*
B: *No*

[2] A: *Do you have Marlboros?*
B: *Uh, no. We ran out*[2]

[2] [Merritt, 1976]

[3] A: *Can you tell me the time?*
B: *No. My watch is broken*[3]

4.3.2 A Detailed Dialogue Example

In this section, the anticipatory framework introduced so far will be used to model an example situation, in line with the interactional analysis presented in the previous section. First, a theoretical model of the example will be given, based on the notion of intention; then, the computational model for it will be presented.

4.3.2.1 Analysis

Consider the situation depicted in example [3] from B's point of view, where B is faced with A's indirect request:

[3] A: *Can you tell me the time?*
B: *No. My watch is broken.*

With reference to this conversational exchange, B can attribute to A two main goals, the goal that B tells her the time and the conversational goal of knowing if her request has been understood, and a set of subsidiary goals (see figure 4.3). We will not describe here how these goals are identified and kept together in a unified structure: we will refer to the work by [Ardissono et al., 2000], where it is shown how the recognition of the intentions stemming from the problem solving activity can constitute the required glue. It is worth noticing, however, that the recognition of domain goals depends on the recognition of the linguistic goals, i.e., on the success of the linguistic actions.

1. the *behavioral* goal that B tells the time to A (*tell-time*): since A cannot take B's cooperation for granted, this goal is related in turn to the goal of knowing whether B has committed to the perlocutionary intention[4] of telling the time to A (*know-if-intend*), by committing to A's goal (*intend-satisfy*); if this is the case, A also has the subsidiary goal of knowing whether she has completed the corresponding plan to tell the time (look at the watch, reading the time, ect.) to A (*knowif-completed*).

2. the *conversational* goal of knowing if the request has been understood by B and is now part of the common ground (*grounded*); this goal directly relates to the management of dialog: if A does not believe that the illocutionary effect of her question holds, she should repeat or reformulate the question.

[3] [Green and Carberry, 1999]

[4] In the terminology of speech acts (see [Austin, 1962] and [Searle, 1969], [Searle, 1975], [Searle, 1992]), the perlocutionary effect of a speech act is its effect contextual effect, which is not contextually determined.

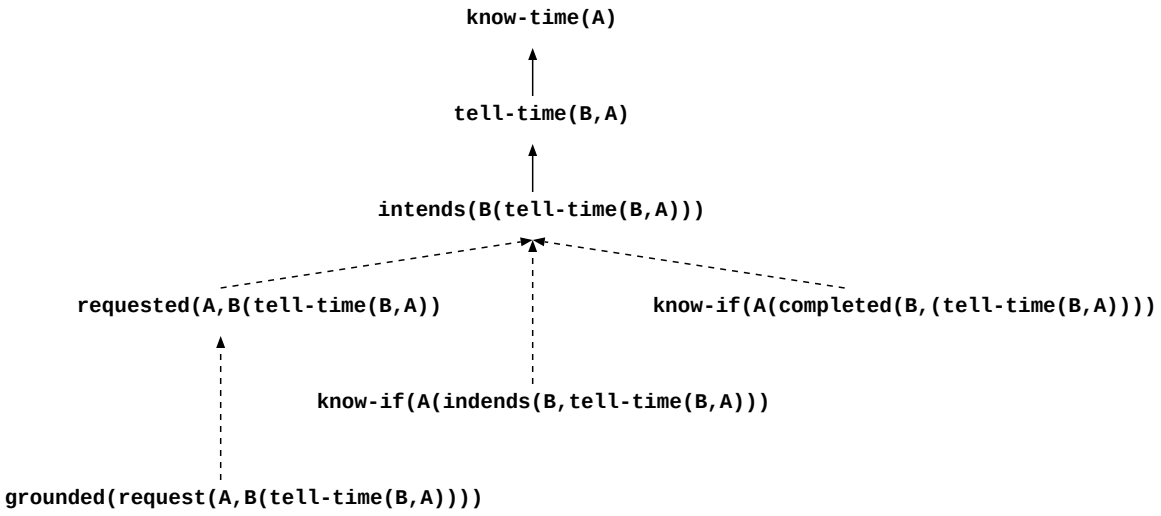

Figure 4.3: The intentions of A in example [1]

Note that, at both levels, subsidiary goals arise as part of the intentional behavior of an agent: for example, after performing an action for achieving some goal, it is rational to check whether this action has succeeded.

B considers if it is possible for her to commit to the higher-level goal (*tell-time*), to which the remaining recognized goals are subordinated: although B is inclined to satisfy this goal, B knows that one of the preconditions for executing the action of telling the time (*(is-working(B,watch))*) is not true.
At this point, besides the choice of not responding at all, the alternative courses of action available to B consist in committing to A's goal to know if B has committed to her (unachievable) `tell-time` goal (`know-if-intends`), and the subordinate goal to know if B has completed the plan to achieve it (`know-if-completed`), or to commit to A's goal - at conversational level - to have her illocutionary act grounded (`grounded`).[5] The choice between the alternative of not responding at all and any of the other alternatives is accomplished by considering the reaction of the partner to a refusal at the conversational level; this choice is enforced by the consideration that communicative actions are "free goods", so they cannot be refused without incurring in a state where the partner is offended.

Being committed to the satisfaction of the `know-if-completed` goal, B has to choose between different ways to communicate the impossibility to execute the plan. In this case, two plans can apply: the simple plan for refusing, or the elaborated plan for refusing which includes a justification for the refusal.
The first plan is less expensive, by being shorter and by not requiring a mental effort; however, it is not fully explicit about the motivations of the refusal, and

[5]Note that, when producing an illocutionary act to satisfy the `know-satisfied` or `know-if-completed` goal, B satisfies the `grounded` goal as well: by displaying the reaction to the perlocutionary effect, the uptake of the illocutionary effect is granted .

so it is potentially offensive in the partner's evaluation (A could think that B didn't want to tell her the time). On the contrary, the second plan, though more expensive, obeys to the preference for not offending, since it protects the refusing requestee from the accusation of not being cooperative.

The existence of complex refusal acts has been remarked on by [Green and Carberry, 1999]. In their mechanism for initiative in answer generation, the ambiguity of a negative answer to a pre-request between a literal answer and a refusal triggers the "Excuse-Indicated" rule, which generates the appropriate explanation.

4.3.2.2 The Model

Five different attributes have been introduced to depict the situation in example [3], where B is interrupted by A while she is executing a generic action *Act*; this action is aimed at reaching one of B's private goal.

[3] A: *Can you tell me the time?*
B: *No. My watch is broken*

The following **attributes** model the states involved in the example situation, and appear in the effects of the participants' actions[6].

- *time*: it models time as a limited resource; the utility decreases as a function of *time*;
- *grounded*: it models A's goal of knowing whether B has successfully interpreted the request;
- *res*: it models the consumption of resources, namely the locutionary and illocutionary effort in case of conversational acts;
- *refused*: it models the fact that the requested action has been refused, or granted; it is true if A believes that B has refused, without any justification, to commit to A's communicated goal;
- *offended*: it is a quantitative attribute, which models the degree of offence of A;

Other goals like knowing whether B has committed to the achievement of the goal or whether the achievement has been successful or higher-level domain goals are not included in this example for space reasons.

In order to model the alternatives available to B, we have introduced the following **actions** (see figures 4.4 and 4.5). Effects are represented as changes in the value of attributes: for example, (time = time + 2) means that after the execution of the *Notify-motivation* action, the value of the *time* attribute will be increased by 2.

[6]Note that the values 0 and 1 of the attributes *ground* and *satisfied-request* represent the truth-values of the corresponding propositions.

- Action *Tell-time*: it represents B's cooperation with A at the behavioral level (B executes the requested action of telling the time);
- Action *Ground*: it has the effect that A knows that the illocutionary effect of her request has been properly recognized by the partner (the *grounded* attribute is set to "true").
- Action *Notify-impossible*: it models B's notification that A's goal is impossible to achieve; it specializes into two sub-actions, *Notify-motivation* and *Notify-simple*. Both actions have a cost in terms of resources and time and set the *grounded* attribute to true, but the second one negatively affects the *refused* attribute, meaning that A considers it as a (possible) unjustified refusal.
- Action *Complex-Act*: it constitutes B's current plan when she is interrupted by A's request. It is constituted by two steps, Act_1 and Act_2, and it affects both the *grounded* and the *refused* attribute, by setting the latter to "false".
- Action *Refuse*: it represents B's act of communicating to A that she will not do what A requested, without providing any justification. Among its effects, there is the fact that B comes to know A's decision to refuse (*refused* and *grounded* attribute are set to "true").

Before B replies to A's request, the *grounded* attribute is set to false and the *refused* attribute is set to true. Note that - with the exception of *Act* - all actions affect the value of the *grounded* attribute, meaning that, after performing any of them, A's request results grounded anyway, since all these actions are coherent replies.

On A's side, we have introduced the action *React* (see Figure 4.7), that models the change of the *offended* parameter depending on B's choice. The key parameter affecting the level of offence is the cost of the requested actions: the less the cost of the requested action, the greater the offence [7]; this follows the principle that low-cost actions cannot be refused, and, if they are, requesters get offended. In particular, the lack of grounding is interpreted by A as B is not cooperating at the conversational level: since cooperating at the conversational level (interpreting the sentence, grounding it) has a low cost, it is offensive not to do it.

Now, let's consider in detail the current situation, i.e, the one where A has just asked to B to do something while B has just performed the first step of *Act*. In order to explore the different alternatives, the planner builds and evaluates some plans. These plans differ in that the actions for pursuing the partner's recognized goal can be included or omitted. From the result state of each alternative, the planner then tries to predict the reaction of A by simulating the execution of the *React* action by A (see figure 4.7), and commits to the plan whose resulting state after the predicted reaction yields the greater utility according to B's preferences (see Figure 4.8.

[7] In practice, (cost(action) = (res * 2) + time).

```
(action
   :name         Notify-motivation
   :attributes   (time resources grounded refused)
   :effects      :cond-eff   :cond   nil
                             :eff    (((time = time + 3) prob = 1)
                                      ((resources = resources - 3) prob = 1)
                                      ((grounded = 1) prob = 1)
                                      ((refused = 0) prob = 1))
                             :prob 1
:goal            (refused = 0)
:instantiation   nil
:decomposition   nil )

(action
   :name         Notify-simple
   :attributes   (time resources grounded refused)
   :effects      :cond-eff   :cond   nil
                             :eff    (((time = time + 1) prob = 1)
                                      ((resources = resources - 1) prob = 1)
                                      ((grounded = 1) prob = 1)
                                      ((refused = 1) prob = 1))
                             :prob 1
:goal            (grounded = 1)
:instantiation   :nil
:decomposition   :nil )

(action
   :name         Refuse
   :attributes   (time resources grounded refused)
   :effects      :cond-eff   :cond   nil
                             :eff    (((time = time + 2) prob = 1)
                                      ((resources = resources - 2) prob = 1)
                                      ((grounded = 1) prob = 1)
                                      ((refused = 1) prob = 1))
                             :prob 1
:goal            (refused = 1)
:instantiation   :nil
:decomposition   :nil )
```

Figure 4.4: The representation of some of the elementary actions that B can execute: Notify-motivation, Notify-simple, and Refuse.

```
(action
   :name         Ground
   :attributes   (time resources grounded)
   :effects      :cond-eff   :cond   nil
                             :eff    (((time = time + 1) prob = 1)
                                      ((resources = resources - 2) prob = 1)
                                      ((grounded = 1) prob = 1))
                             :prob 1
:goal            (grounded = 1)
:instantiation   :nil
:decomposition   :nil )

(action
   :name         Tell-time
   :attributes   (time resources grounded refused)
   :effects      :cond-eff   :cond   nil
                             :eff    (((time = time + 1) prob = 1)
                                      ((resources = resources - 2) prob = 1)
                                      ((grounded = 1) prob = 1)
                                      ((refused = 0) prob = 1))
                             :prob 1
:goal            (refused = 0)
:instantiation   :nil
:decomposition   :nil )

(action
   :name         Act_2
   :attributes   (time resources goal)
   :effects      :cond-eff   :cond   nil
                             :eff    (((time = time + 5) prob = 1)
                                      ((resources = resources - 5) prob = 1)
                                      ((goal = 1) prob = 1))
                             :prob 1
:goal            (goal = 1)
:instantiation   :nil
:decomposition   :nil )
```

Figure 4.5: The representation of some of the elementary actions that B can execute: Ground, Tell-Time, and Act.

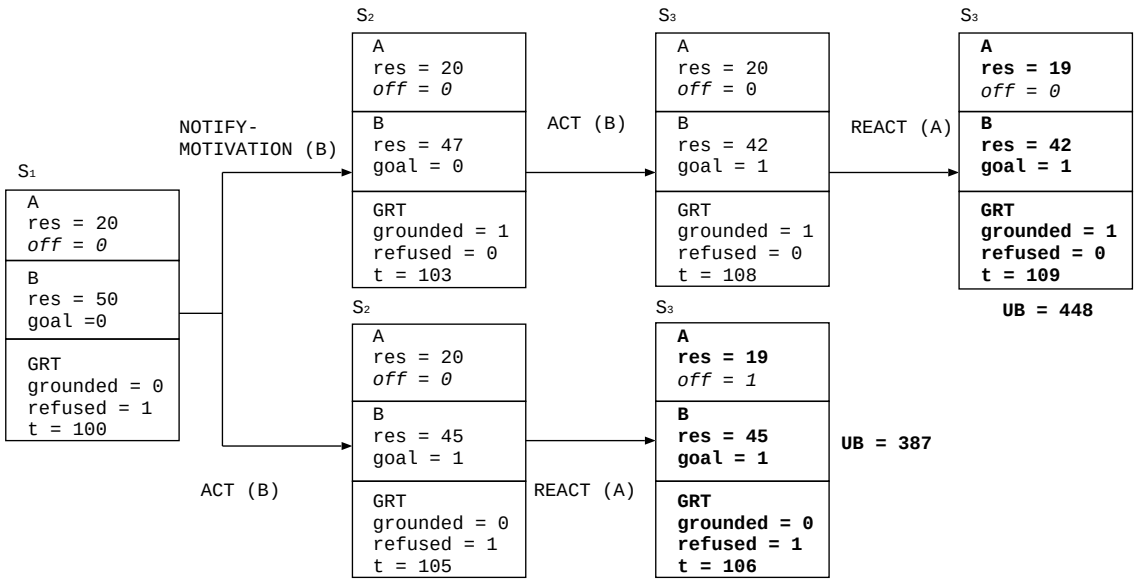

Figure 4.6: Two of B's alternative plans in response to A's request

As explained in chapter 2, an agent's utility function is a weighted sum of individual utility functions, which represent the preferences of the agent. The weights associated to the individual functions reflect the strength of each preference, by allowing for different trade-offs among preferences during the process of decision making.[8]
In figure 4.6, two alternative plans are represented, where the utility of B is calculated by using the utility function in figure 4.8. Assuming that the weights W_1, W_2, W_3, and W_4 are set to 10, 5, 8, and 100, respectively, B will choose the plan which includes *Notify-impossible* as the first step, and *Act* - the prosecution of B's previous activity - as the second step. This solution yields in fact a higher utility than the alternative of ignoring A's request at all and continuing one's own activity.

In order to validate the model by adapting it to different situations, we performed some experiments by modifying the utility function of the agent.

A change in the weights of the utility function of B affects her behavior, by determining a variation in the degree of cooperation: the stronger is the preference for not offending, the more cooperative is the agent. For example, if the utility function of B associates a greater utility to the achievement of B's private goal (by executing *Act*) than to the social preference for not offending, B will decide

[8] As [Traum, 1999] notices with reference to *social rules*, "when they directly conflict with the agent's personal goals, the agent may choose to violate them (and perhaps suffer the consequences of not meeting its obligations)." In our model, this roughly amounts to associating a greater utility to the achievement of the agent's own goals than to the preference for not offending.

```
(action React
     (time = time + 1)
     (res = res - 1)
     (offended = offended +
         (W1 * not(grounded)) +
         (W2 * refused/cost(action))))
```

Figure 4.7: The partner's reaction

```
UB= (resB * W1) +
    ((t - 100) * W2) +
    (offended * W3) +
    (g * W4)
```

Figure 4.8: The utility function of B

to disregard A's request, both at conversational and behavioral level.[9] On the contrary, if the utility function of B models a more balanced trade-off between the achievement of B's private goals and social preferences, B will decide to ground A's request, at least, or to be fully cooperative by satisfying A's request.

- By associating a greater utility to the completion of *Act* than to *offended*, and by keeping t_{end} tight (i.e., t_{end} coincides with the sum of the durations of Act_1 and Act_2), we obtain that B decided not to cooperate at the behavioral level. The *Ground* action was executed depending on how fast the utility of doing *Act* decreases as a function of time after the deadline, and on the utility of not offending the partner.
- By relaxing t_{end}, so that B has time enough to answer to A and complete *Act* as well within the time constraint, we obtain a fully cooperative behavior: the sequence $<Notify - motivation$, $Act_2>$ is the preferred plan (the $Notify - motivation$ action is not very expensive and, at the same time, it ensures that the partner's request is grounded and the sequence $<Adopt$, $Act_2>$: these two effects are both accounted for by the utility function that drives the planning process).
- By increasing the utility of the resource *res* with respect to *offended*, the agent does not commit to satisfying the interlocutor's communicated goal, or, at best, the agent grounds the request or explicitly refuses behavioral cooperation. the agent was more worried about not wasting her efforts than about preventing B from being offended.

Finally, notice that this solution does not rest on an explicit notion of obligation, even if some similarities can be found with [Traum and Allen, 1994]. The

[9] Tipically, this is the case in specific contexts when private goals of the addressee are very relevant and contrast with the satisfaction of the requester's goal; for example, B could missing the train.

advantage of not resorting to a primitive notion of obligation is to have a uniform source of motivations for explaining the behavior of agents in social settings.

With respect to approaches which stipulate a primitive notion of obligation, here, the same phenomena are accounted for without introducing further propositional attitudes. This explanation of the motivations leading to cooperation provides an explicative model that is uniform with the treatment of *deontic reasoning* in agent theories [Conte et al., 1998].

It is clear that, by reducing the number of propositional attitudes, the reasoning process becomes more complex, but our model is aimed at constituting an explanation, and it does not exclude the possibility of compiling the reasoning in more compact form: as [Brown and Levinson, 1987] notice, "there is a rational basis for conventions, too". Moreover, in case an agent decides to violate an obligation, she has to predict the consequences of her behavior: a primitive notion of obligation is not sufficient to reason about violations.

4.3.3 Social Anticipatory Reasoning in a Complex Scenario

4.3.3.1 The Scenario

In this section, I will present an example for illustrating the role of social goals in multi-party dialogue settings. The example is inspired by the Mission Rehearsal Exercise (MRE) project, a virtual reality application developed at the University of Southern California's Institute for Creative Technologies with the aim of improving decision-making skills of US military officers in complex, real time situations ([Rickel et al., 2001]). The Mission Rehearsal Exercise is an immersive application, where the human user interacts with virtual characters by communicating with them in natural language.

The MRE virtual characters consist of Steve agents ([Rickel and Johnson, 1999]) which are given dynamically animated body; virtual characters are situated in a dynamic environment including a message-passing event simulator and accurate graphic effects. Real-time, interactive aspects pose critical issues for the generation of the behavior of these characters, and for their dialogic and interactional behavior in particular. [Traum and Rickel, 2001] propose a multi-modal dialogue model for supporting the conversational and interactional capabilities required by the application.

The example scenario is situated in the virtual reconstruction of a small village in Bosnia. During a peace-keeping mission, a military vehicle of the US Army, while speeding through the village, crashes into a civilian car. The driver of the military vehicle is lightly injured, but one of the passengers of the car, a young boy, is seriously injured. In the meantime, a crowd of local inhabitants gathers, and starts showing hostile intentions to the US military people.
Having heard that a platoon is in trouble, a Liutentant rushes to help; on his way

```
LT:     Sergeant, what happened here?

SGT:    They just shot out from the side street sir
        The driver couldn't see'em coming.

LT:     How many people are hurt?

SGT:    The boy and one of our drivers.

LT:     Are the injuries serious?

MEDIC:  Driver's got a cracked rib but the kid's -
        Sir, we gotta get a Medevac in here ASAP

LT:     We'll get it

LT:     Platoon Sergeant, secure the area!
```

Figure 4.9: An excerpt of the conversation taking place in the Mission Rehearsal Exercise. Non-verbal events are not represented in the figure; for a complete representation, see [Traum and Rickel, 2001].

there, he notices the accident. After the platoon sergeant has briefed him on the situation, and the medic has assessed the boy's situation as critical, the lieutenant orders the sergeant to secure the area in order to get a Medical Evacuation (Medevac) for the boy. The conversational exchange between the Lieutenant, the Sergeant and the Medic is reported in figure 4.9.

With respect to the conversation reported in figure 4.9, we will focus on the following excerpt:

```
[1] LT     Are the injuries serious?

[2] SGT    [Makes eye contact with the medic and nods]

[3] MEDIC  Driver's got a cracked rib, but the kid...

[4]        Sir, we gotta get a Medevac ASAP
```

4.3.3.2 Analysis

In particular, we will use the anticipatory planning to model social aspects in the generation of the answer by the Medic (lines 3-4). The Medic, in fact, does not

answer to the Lieutenant's question (line [1]) literally, by explicitly expressing the fact that the boy's condition is critical; instead, he resorts to an non literal answer, strongly suggesting a medical evacuation (lines [3] and [4]). Notice that this answer, although it is not a literal answer, it is perfectly appropriate from a pragmatic point of view, as all the participants to the interaction perfectly know that the boy's condition being critical almost necessarily implies the choice of a Medevac. By answering not literally, and by using a technical expression, "Medevac", the Medic obtains the result of not being understood neither by the boy's mother, who's visibly upset, nor by the angry crowd, evidently hostile, which has been gathering around the accident area. At the same time, the answer is characterized by a high degree of initiative, as it contains a the explicit suggestion of a certain line of behavior, instead of the plain assessment which has been required by the interlocutor.

Provided that the goal of minimizing the risk for the boy's life drives the domain and linguistic choice of all interactants, the model should account for the following domain-specific factors which influence the decision-making activity of the Medic:

- The Medic's *social preference* for not arousing the mother's emotional reaction. An emotional reaction, in fact, would be disadvantageous as it may compromise the ability of the military people to keep the situation under control. In the model, this preference determines the choice of a non-literal answer.

- The Medic's personality trait constituted by *self-confidence*. In order to model the fact that taking the initiative by giving suggestions involves an assumption of responsibility for what has been suggested, we model the Medic's self-confidence as the factor which determines his attitude towards the assumption of responsibility. The more self-confident is the Medic, the more he tends to take the initiative, by assuming the responsibility for the suggested line of behavior. So, the Medic's confidence in his assessment affects the degree of initiative contained in the answer:

- The *resources* required by the suggested solution. In fact, given the goal of minimizing the risk for the boy, it would be irrational to suggest a line of behavior which has a higher cost than needed. Although the cost of the suggested line of behavior is not a direct effect of the Medic's communicative act, it has been inserted in the model as a local effect for the sake of simplicity.

As far as the available actions for the Medic are concerned (see figure 4.10), the highest level action consists of a plan for a two-step complex dialogue act of answering: as the Lieutenant's question concerns both the driver and the boy

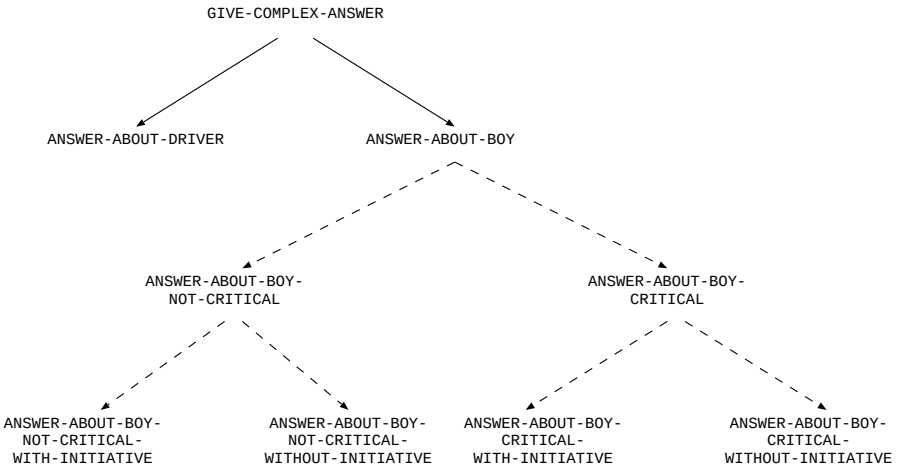

Figure 4.10: The plan schema used in the experiments to model the Medic's answer.

(the people involved in the car crash), the first part of this dialogue act is the answer to the question about the driver's condition, the second part is the answer to the question about the boy's.

The focal point of our interest, however, is constituted by the second step of the answering act, concerning the boy's condition. Provided, again, that his goal is minimizing the risk for the patient, the Medic has two main options, which consist of answering that the situation is critical or answering that the situation is not critical: the choice of the appropriate option depends on whether the boy's condition is critical or not in the Medic subjective view of the situation, and with what probability it is critical in case of uncertainty. The two options are associated with different costs: if the situation is critical, this equates to saying that a Medevac - a very expensive line of behavior - is necessary, while answering that the situation is not critical implies a less expensive course of action, like ambulance evacuation (see above for what concerns the consumption of resources as a local effect of the Medic's actions).

Both in case the Medic opts for answering that the boy's condition is critical and in case it is not, he has to choose between using a direct or indirect way to express his assessment. So, each option further specializes into two other options consisting of an explicit, literal answer, or an implicit, non-literal answer.

4.3.3.3 Description of the Example Domain

In summary, the initial world is modelled by the following attributes:

- **confidence**: this attribute models the Medic's confidence in his assessment. It has a boolean value, *true* of *false*.

- **critical**: it models the fact that the boy's condition is critical (*true*) or not (*false*). Notice that the value of this attribute is independent of the value of the confidence attribute, in order to model the Medic's individual self-confidence separately from his assessment of the situation.[10]

Notice that, in reality, none of these attributes is two-value; however, they have been reduced to boolean attributes for simplifying the model.

In addition, the Medic's actions affect the following attributes

- **responsibility**: answers which are characterized by initiative set this parameter to *true*, meaning that the Medic assumes the responsibility for the suggestion he makes by taking the initiative. Otherwise, the value of this attribute is set to *false*.

- **risk**: the range of this attribute corresponds to the integers between 1 (*very low*) and 4 (*very high*). Answering that the boy's condition is not critical (not matter if directly or indirectly) sets the the risk to very low if it is actually non critical, answering non critical when it is critical always sets the risk to very high (as it would lead to the most inappropriate course of actions); on the contrary, answering that the boy's condition is critical when the situation is actually not critical set the risk to the very low.

  In addition, the two specializations of the action of answering that the boy's condition is critical are differentiated from each others: a direct answer sets the risk to 3 (*high*), while an indirect answer sets the risk to 2 (*low*), thanks to the fact that it is more explicit about the appropriate behavior (the Medevac) - and it will presumably influence the Lieutenant's decision.

- **resources**: this attribute models the consumption of resources associated with the course of action associated with the type of answer. The initial value is 100; answering that the situation is not critical, no matter if the answer is direct or indirect, sets the consumption of resources to 10, while answering that the situation is critical sets the consumption of resources to 20. In reality, this is a way to account for the impact of the answer on the subsequent planning of the interlocutors who are in charge of taking the final decision. However, since here we are interested only on the generation of the Medic's answer, we attach the consumption of resources to the type of answer for the sake of brevity.
  This attribute is affected also by the first step of the complex answering action.

[10]While the confidence directly enters the utility evaluation, as shown later, the situation being critical influences the utility in a different way, by indirectly affecting the expected utility value in the presence of uncertainty about the boy's condition, thanks to the built-in evaluation mechanism of the DRIPS system (see also [Xuang and Lesser, 1999].

- **explicit**: this boolean attribute models whether the answer of the Medic is explicit or not about the boy's condition. It is crucial to the evaluation of the social preference for not upsetting the mother, as the mother's emotional reaction varies according to its value.

- **upset**: this attribute is set by the mother's reaction to *false* (the mother is not upset) or *true* (the mother is upset). Its value depends in turn on the value of the explicit attribute: the mother is upset if, as a result of the Medic's answer, it is explicit that the boy's condition is critical.

- **time**: it models the duration of each action.

Finally, the model includes a utility function which specifies the role of the goal achievement and of the social and domain preferences in the evaluation of the overall utility of a certain course of action (plan).
More specifically, the expected utility evaluation combines the satisfaction of the goal of minimizing the risk (modelled by the risk attribute) with the utility which derives from *the satisfaction of the preferences* (modelled by the responsibility, resources and upset attributes) by means of a weighted sum, where the goal satisfaction utility is assigned a weight equal to 1, while the preference utility is assigned a weight of 0.2.[11]. Note that all these values involved in the evaluation of the expected utility are arbitrary; for an account of how the utility function can be learned from the a preference order on course of action, see ref.

For what concerns the goal satisfaction utility, it is inversely proportional to the degree of risk: the lesser the risk, the higher the utility.[12] The utility which derives from the satisfaction of the preferences combines the advantage of *saving resources* with the advantage provided by the *initiative* in giving the answer.[13] The evaluation of the preference-dependent utility, however, varies according to the Medic being self-confident or not, in the following way:

- The Medic is self-confident: the utility of saving resources is increased by a factor of 0.5 if the responsibility attribute is set to true in the final state, and is decreased by the same factor if the responsibility attribute is set to false. By doing so, the plans which result in an assumption of responsibility are preferred, i.e., the communicative plan which contain initiative (i.e., the non-literal answer).

[11]EU = (W_1 * goal-utility + W_2 * preference-utility), where $W_1 = 1$ and $W_2 = 0.2$.

[12]If risk = 1, goal-utility = 1; if risk = 2, goal-utility = 0.7; if risk = 3, goal-utility = 0.5; finally, if risk = 4, goal-utility = 0.35.

[13]The utility associated with saving resources is computed in this way: if the consumption of resources is more than 30, the utility of saving resources is equal to the amount of resources left divided by 100; otherwise, it is equal to 1. Given the library of plan for this example, the first case applies to the plans which contain the specifications of the act of answering that the situation is critical, while the latter applies to the plans containing the specifications of the act of answering that the situation is not critical.

- The Medic is not self-confident: the utility of saving resources is increased by a certain value if the responsibility attribute is set to false in the final state, and is decreased by the same value if the responsibility attribute is set to true. By doing so, the plans which do not result in an assumption of responsibility are preferred, i.e., the communicative plans which contain no initiative (i.e., the literal answer).

Furthermore, the utility function accounts for the social preference not upset the mother. If the mother is upset, i.e., the upset attribute is set to true in the final outcome, the utility is decreased by a certain factor[14]; otherwise, it is increased by an (arbitrary) factor of 1.5.

In order to verify how this model accounts for the Medic's behavior in the example, we performed some experiments by varying the initial situation. First of all, the behavior of the Medic has been simulated in a set of initial situations without accounting for the social responsibility. So, a range of different behaviors is obtained, depending on the Medic's self-confidence and the seriousness of the boy's condition, which in the Medic's subjective representation of the world is characterized by different degrees of uncertainty. Then, the results of these experiments has been contrasted with a set of experiments where the Medic's behavior has been simulated by accounting for the social responsibility: the comparison between the two sets evidence the role of social awareness in the Medic's planning.

4.3.3.4 Experiments

Socially unaware behavior

1. The value of the confident attribute in the initial world is true (the Medic is confident in his own assessment), and the value of the critical attribute is set to true (the boy's condition is critical): this situation depicts the situation represented in the scenario. Given this situation, the agent Medic would opt for the following plan:

   `ANSWER-ABOUT-DRIVER - ANSWER-ABOUT-BOY-CRITICAL-WITH-INITIATIVE`

   (with an expected utility of .9400)

   i.e., the medic would opt for a plan which, although not intentionally, would not upset the mother, by giving the same answer as the example excerpt.

2. If the value of the confident attribute is set to false (the Medic is not confident about his assessment) and the situation is critical, the preferred plan is the following:

[14] Arbitrarily set to 0.5.

`ANSWER-ABOUT-DRIVER - ANSWER-ABOUT-BOY-CRITICAL-WITHOUT-INITIATIVE`

(with an expected utility of 0.8400: notice that the utility is lower, as the type of answer affects the value of the risk attribute - the indirect answer makes the risk lower than the direct answer)

3. If the Medic is confident about his assessment, but is uncertain about the seriousness of the boy's condition, we obtain a set of different behaviors. The uncertainty is modelled by a posing a probability distribution on the value of the critical attribute.

   (a) With a probability of 2/3 that the boy's condition is critical, the preferred plan is
   `ANSWER-ABOUT-DRIVER - ANSWER-ABOUT-BOY-CRITICAL-WITH-INITIATIVE`
   (with an expected utility of 1.0400);

   (b) With a probability of 1/3 that the boy's condition is critical, the preferred plan is
   `ANSWER-ABOUT-DRIVER - ANSWER-ABOUT-BOY-CRITICAL-WITH-INITIATIVE`
   (with an expected utility of 1.1400);

   (c) With a probability of 1/10 that the boy's condition is critical, the preferred plan is
   `ANSWER-ABOUT-DRIVER - ANSWER-ABOUT-BOY-NOT-CRITICAL-WITH-INITIATIVE`
   (with an expected utility of 1.2350).

   Notice also that the utility monotonically rises as the the seriousness of the situation - in the Medic subjective assessment - diminishes: it is lower when the situation is certainly critical, and higher when it is almost certainly critical (critical is set to true with a probability of 10%).

These experiments highlight that, in the absence of social awareness, the possibility that the Medic opts for a behavior that complies with the social preference for not upsetting the mother depends only on his confidence about his assessment and on the uncertainty about the seriousness of the situation (see result of the last experiment, where the Medic is almost certain that the situation is not critical).

Socially aware behavior

By introducing the social preference for the mother not being upset, we obtain a different, socially aware behavior that accounts for the mother's emotional reaction:

1. When the confident attribute is set to true, and the critical attribute is set to true, i.e., the boy's condition is critical and the Medic is confident about his assessment, the preferred plan is

   `ANSWER-ABOUT-DRIVER - ANSWER-ABOUT-BOY-CRITICAL-WITH-INITIATIVE`

   (with a utility of 1.0600).

   Notice that this plan would be preferred to the plan `ANSWER-ABOUT-DRIVER - ANSWER-ABOUT-BOY-CRITICAL-WITHOUT-INITIATIVE`, which would yield an expected utility of 0.6200: for this reason, this plan is pruned during the planning process.

2. When the critical attribute is set to true, but the confident attribute is set to false, the preferred plan is still the same:

   `ANSWER-ABOUT-DRIVER - ANSWER-ABOUT-BOY-CRITICAL-WITH-INITIATIVE`

   (with an expected utility of 0.7600)

   Notice that, differently from the examples 2.a, 2.b, and 2.c in the first series of experiments, the preferred plan is the plan which contains the initiative, independently of the Medic's self-confidence: here, even when the Medic is not confident, the social preference for not upsetting the mother promotes the option constituted by the non-literal answer.

   Here, the expected utility of the plan `ANSWER-ABOUT-DRIVER - ANSWER-ABOUT-BOY-CRITICAL-WITHOUT-INITIATIVE` would be 0.7200: the gap between the expected utility of the two options is lower in this experiment with respect to the previous one, due to the preference for not taking the responsibility (which derives from the initiative).

3. Finally, we considered the situation in which the Medic is confident, but is uncertain about the seriousness of the boy's condition (in parallel with the third experiment in the first series), with a probability of 2/3 that the attribute critical is set to true. In this case, the preferred plan was, again, the same:

   `ANSWER-ABOUT-DRIVER - ANSWER-ABOUT-BOY-CRITICAL-WITH-INITIATIVE`

   (with an expected utility of 0.8533)

This second series of experiments highlights the impact of the social preference for not upsetting the mother affects the behavior of the Medic in the example, by modelling the example in the interactional framework we propose.

Depending on how the social preference combines with the remaining preferences, however, different behaviors could be accounted for. If the Medic's utility function were modified, a different trade-off among the preference could be modelled: for example, the Medic could give more importance to saving resources

or to taking (or not taking) the responsibility, so that these preferences would override the social preference in a wider range of situations.

Finally, some words must be devoted to the role of the Medic's self-confidence in the model. As evidenced by [A. Ortony and Collins, 1988] and [Picard, 1997], *emotional factors* have a role in the rational decision process. In the account of emotional factors in decision-making given by [Gmytrasiewicz and Lisetti, 2000], these factors influence the decision-making process in an indirect way. For example, emotional factors can affect the agent's uncertainty about the values of the attributes which define the world in the agent's subjective representation.
Although we think that personality traits, like self-confidence, should influence the decision process in a similar, indirect way, the reactive agent architecture this model is based on does not currently allow us to account for the role of self-confidence in an indirect way. For instance, modelling self-confidence as affecting the agent's uncertainty about the world would require a more complex model of sensing, where a set of parameters can influence the results of the agent's monitoring. By modelling self-confidence as an external factor to the rational decision process which affects the Medic's uncertainty about the diagnosis of the boy's condition, the Medic's initiative would be affected only indirectly by his confidence in the decision process, as the probability of the boy's condition being critical reflects in the preference for an answer which contains initiative.

The model we proposed here explicitly encodes the Medic's self-confidence in an explicit way: however, we think that the validity of the model for what concerns the predictions it makes is not threatened, as it correctly predicts that self-confidence determines the assumption of responsibility, although it may be imprecise for what concerns the cognitive modelling.

Chapter 5

Reacting to Norms

5.1 Introduction

In a society, the behavior of an agent is constrained by a set of norms, more or less formalized, which form a continuum spanning from laws to social conventions; social preferences, discussed in the previous chapter, are situated at one extreme of this continuum, while the laws of a state are situated at the opposite extreme.

In this chapter, we will introduce a model of normative reasoning that exploits the interactional framework described in the previous chapter to characterize norms with respect to the notion of *sanction* and *normative authority* ([Boella and Lesmo, 2001]).

The schema of the chapter is the following: first, we illustrate a model for normative reasoning in BDI agents; then, in section 5.3, this model is integrated within the reactive agent architecture described in chapter 2. Section 5.4 describes the process of integrating normative goals with existing intentions, focusing on a plan merging algorithm (5.4.1). Finally, detailed examples (sections 5.5.1 and 5.5.2) are presented in order to illustrate the functioning of the norm-reactive agent architecture.

5.2 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. Al-

ternatively, the satisfaction of a norm can be associated with a reward, whose aim is to motivate agents to respect the norm.

An important advantage of this solutio is that it does not require the introduction of an ad hoc propositional attitude for modelling the agent's obligation to respect a norm. On the contrary, by modelling the compliance to norms as a rational decision, normative reasoning can be easily integrated in the process of intention formation.
Moreover, the cost/benefits reasoning sketched above complies with the *principle of normative autonomy* ([Conte and Castelfranchi, 1995]), according to which the decision to comply with a norm is always instrumental to the satisfaction of an independent goal of the agent, like receiving a reward or avoiding a punishment. Furthermore, it is in line with some recent developments in Deontic Logics ([Horty, 1996], [Lang et al., 2001]), where the notion of duty is characterized in terms of utility (see Chapter 1, Related Work).

Formally, the definition of a norm ([Boella and Lesmo, 2000]) is constituted by a quadruple which includes the agent who is the *bearer* of the obligation, the *normative authority*, who is in charge of the respect of the obligation, the *sanction*, the *content* of the obligation, and a *triggering condition*:

The **bearer** of the norm is the agent who is *obliged* to respect the norm. In this approach, norms are assumed to be personal, i.e., they concern individuals: an individual who is obliged to respect a norm must act so that the prescription contained in the norm, at a certain moment, is fulfilled.

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**) provides the rational motivation for respecting the norm. The bearer knows that, if she does not comply with the norm, she may incur (depending on the normative authority's behavior) the sanction prescribed by the norm definition.

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 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, as stated in previous chapter, 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 previous chapter. 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. In order to do so, the normative authority monitors the environment for violations, and, if one is detected, forms the intention to sanction the agent who has violated the norm (or to reward those who are compliant with it), and eventually executes a sanctioning (or rewarding) plan. As any other agent, the normative authority is characterized by a utility function, that models the advantage, for the normative authority, of sanctioning the agents who do not comply with the norm.[1] So, 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:

[1]Beside the goal of enforcing the respect of the norm, the normative authority, of course, may have other goals and preferences as well.

- 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.

5.3 Utility-based Compliance 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].

In Chapter 2, 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 chapter 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 presented in the previous chapter), i.e. in the light of the reaction of the normative agent ([Boella et al., 2000a], [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 the following, we report the norm-reactive agent algorithm:

```
procedure agent (goal, subj-world)
    /* initial planning phase */
    plan := DELIBERATE (goal, subjective-world);
    execution := INITIALIZE-EXECUTION (plan);
    /* the agent loop begins here */
    loop
      /* execute next action */
      objective-world := EXECUTE (next action);
      /* sensing */
      subj-world := MONITOR (next-action);
      /* check if goal has been achieved */
      if execution.actions-to-execute = empty
          and ACHIEVED-GOAL (subj-world, goal) = T
       then return SUCCESS;
       else
        begin
         /* The meta-deliberation phase is entered: */
         if MONITOR-EXECUTION (subj-world) = "replan"
         /* the agent tries to revise its intentions */
          then
           begin
            /* redeliberation is attempted */
            new-plan := RE-DELIBERATE (execution, subj-world, goal);
              if new-plan
              /* new feasible plan found, update intentions */
             then plan := new-plan; UPDATE-INTENTIONS (plan, execution)
               /* no new feasible plan */
```

```
                else return FAILURE
              end if
            end
          end if
          /* the agent monitors for norms */
          if not (MONITOR-NORMS = empty)
            /* norms triggered: normative reasoning */
          then plan := NORMATIVE-DELIBERATION(plan, norms, subj-world)
          end if
          /* set the next action to resume execution */
          SET-NEXT-ACTION (execution)
        end
      end if
    end loop
end procedure
```

5.4 Contextual Integration of Normative Goals

The definition of norms presented in section 5.2 does not provide any specification concerning the content of norms. Here, we introduce a classification of norms which is functional to the integration of normative goals into existing intentions. The classification of norms refers to their content and is based on 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, nromally 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; in extreme, the goal may be that the prescribed

course of action be executed, while the goal for which it is executed may be irrelevant.

- 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**. Similarly to the prescription to execute a certain course of action, the real goal may be irrelevant, provided the agent complies with the prohibition that the specified course of action is not executed.

In order to account for norms containing prescriptions, we introduce a *Plan Integration Algorithm*, which tries to modify the current plan in order to achieve the normative goal. If the prescription concerns a state of affair, the algorithm looks for plan fragments which achieve the normative goal, then produces a set of candidate plans by inserting each of the collected plan fragments in all available positions in the current plan. Candidate plans which are not feasible are further modified by performing a step of plan expansion (see section ? in chapter 2).

The integration of normative goals in existing plans is encompassed by the more general theory of rational choice: as [Horty and Pollack, 2001] claim, new options should be evaluated in the context of existing plans. According to [Horty and Pollack, 2001], the cost of a new option should not be evaluated in isolation; instead, a new option should be evaluated by merging the plan for achieving it into the existing plan: the differential cost of the merged plan with respect to the existing plan constitutes the contextual cost of the new option. Here, the new option is constituted by achieving the norm-related goal; in order to evaluate the new option in the context of the existing intentions, the agent must be able to merge the current plan with the plans for achieving the normative goal. However, the evaluation of the new option is based on different premises with respect to [Horty and Pollack, 2001], as the original plan and the merged plans are not evaluated as such, but in the light of the reaction of the normative authority.

On the contrary, norms which pose constraints on the agent's freedom of action, like prohibitions, require a different type of intention revision. Instead of adding actions for bringing about the normative goal, the agent should rather try to find a plan which is compatible with the norm by replanning with the additional constraints imposed by the norm. If an alternative plan exists which achieves the agent's current goals, and, at the same time, leads to the observance of the norm, such plan can then be traded-off with the original plan.

5.4.1 Merging Plans for Satisfying a New Goal

In this section, we present an algorithm for integrating normative goals within existing intention by means of plan merge. In short, the Plan Integration algorithm, performs the following steps:

1. It looks for suitable plan fragments by accessing the action library (RETRIEVAL); the retrieval is based on a match between the effect of plan fragments stored in the plan library and the goal.

2. Then, for each plan fragment it tries to merge the plan fragment it into the current plan (PLAN MERGE).

3. For each of the resulting plans, if it is not a feasible plan - in fact, there may be interferences among actions - it calls the Plan Expansion algorithm on it (EXPANSION).

4. Finally, it orders the feasible plans according to their expected utility.

As far as the merge is concerned, there are two cases: if the plan fragment is not a partial plan, the algorithm tries to insert the sequence of steps which compose it in the current plan, in all positions, and tests the resulting plan for feasibility, i.e., it checks whether the resulting plan achieves the new set of goals constituted by the initial goal and the new goal.

If the plan fragment to be merged into the current plan is a partial plan, it requires refinement as the agent can execute only elementary actions. However, since during the refinement process the planner orders plans on the basis of their expected utility and prunes sub-optimal ones, the partial plan must be refined in the appropriate context (**contextual planning**). The initial world for the planning process, that determines the plan expected utility is normally the current world in the standard planning process; in this case, the initial world for the contextual planning is the world that would result from the execution of the sequence of steps that precedes the position where the expanded plan fragment will be inserted (contextual insertion).
So, the Plan Merging algorithm first creates the appropriate world by simulating the execution of the steps which precede the insertion point of the plan fragment, then invokes the planner on the plan fragment in the appropriate world. The resulting plan is finally tested for feasibility, and if it turns out to be unfeasible, it is passed to the Plan Expansion algorithm.

This algorithm does currently not account for the fact that the candidate plan built in step 2 may contain duplicate steps, which should be merged for obtaining a rational solution in which resources are not wasted. Since the action representation style include the action type, this task could be quite easily accomplished; however, we leave intelligent plan merging for future work.

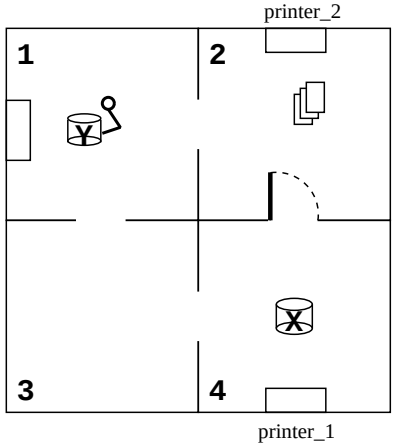

Figure 5.1: The office world

5.5 Examples

5.5.1 A Plan Integration Example

In order to illustrate the functioning of the Plan Merging Algorithm, we resort to an example in the office micro-world introduced in chapter 2, in which two robots, X and Y, perform simple tasks like going from one room to another to move small objects, taking the mail, etc. The office micro-world contains four rooms, and there is a door between room 4 and room 2 (see figure 5.1).

Due to funding limitations, only one of the robots, let's say Y, has the ability to open and close the door, while X can go through a door only if it is open; however, Y can open the door only if it is in room 4. In addition, the normative knowledge of Y contains the following norm, according to which, if Y is required to open the door, Y must open the door; otherwise, the normative authority (the office supervisor) may sanction Y by forcing it to a 10 minute stand-by.
So, the attribute-value pair (request-open = T) is the condition which triggers the norm:

(NORM
:BEARER (Y)
:TRIGGER (REQUEST-OPEN = T)
:CONTENT (DOOR = OPEN)
:SANCTION (STAND-BY (10'))
:NORMATIVE-AUTHORITY (SUPERVISOR)
)

Consider the situation where agent Y is committed to executing a plan for taking some papers from office 1 to office 2 and is currently situated in room 3. The current plan is the following:

GO-Y-3-1 TAKE-Y-papers GO-Y-1-2

In the meantime, his companion, X, finds a closed door on his way, and, being unable to open it, asks for Y's help by issuing a request of help; this request affects the world by setting the attribute request-open to T (request-open = T). This attribute triggers the norm stored in Y's normative knowledge, that prescribes that Y opens the door when requested by X. So, when Y is about to execute this plan, he realizes than X has issued a request that Y opens the door. Below, we report the trace of the Normative Monitoring and of the Plan Integration Algorithm, which is subsequently invoked:

```
I found an instance of the norm:

(NORM

    :TRIGGER ( REQUEST-OPEN = T )

    :CONTENT (DOOR = OPEN)

    ...

    )
```

The algorithm first tries to find a plan-fragment to achieve the normative goal (DOOR = OPEN) and retrieves from the library the plan composed of the elementary action (UNLOCK-Y).

```
Retrieving plan fragments to achieve the new goal: ( UNLOCK-Y )

The merging step generates the following plans:

Modify-plan:

  candidate plan: ( UNLOCK-Y GO-Y-3-1 TAKE-Y-papers GO-Y-1-2 )

Modify-plan:

  candidate plan:( GO-Y-3-1 UNLOCK-Y  TAKE-Y-papers GO-Y-1-2 )

Modify-plan:
```

```
  candidate plan : ( GO-Y-3-1 TAKE-Y-papers UNLOCK-Y GO-Y-1-2 )

Modify-plan:

  candidate plan : ( GO-Y-3-1 TAKE-Y-papers GO-Y-1-2  UNLOCK-Y )
```

However, none of the generated plans is, by itself, a feasible plan; so the algorithm tries to modify them by resorting to the Plan Expansion algorithm (see chapter 2). For each plan, the Plan Expansion algorithm looks for actions whose preconditions are not valid (see section 3.4.3) and tries to insert additional steps to enable these actions. The first plan (UNLOCK-Y GO-Y-3-1 TAKE-Y-papers GO-Y-1-2) turns out to be repairable; for the sake of brevity, the description of the expansion attempts performed on the remaining plans are omitted, and the repair process is illustrated for the only repairable plan.

The Plan Expansion algorithm tests the precondition of all actions constituting the plan steps, in search for actions which are not enabled, and finds out that the inserted action, UNLOCK-Y, is not enabled: in the initial world, the agent is in room 3, while it should be in room 4 in order to be able to open the door. Then, by reasoning as if the execution of the action of opening the door (UNLOCK-Y) were successful, the algorithm realizes that, in that case, the subsequent action of going from room 1 to room 2 would not be enabled: under the assumption that Y goes to room 4 to open the door, Y cannot subsequently perform the action of going from room 3 to 1 by being in room 4, which requires Y to be in room 3 again.

Having examined all plan actions, the algorithm tries to satisfy the preconditions which have been found unsatisfied by inserting an appropriate action right before each action which is not enabled: thus, it inserts a step constituted by the action of going from room 3 to room 4 before UNLOCK-Y step (GO-Y-3-4), and a step of going (back) from room 4 to room 3 (GO-Y-4-3) before the step of going from room 3 to room 1. The resulting plan is now feasible - given the agent's beliefs, and execution can start:

```
( GO-Y-3-4   UNLOCK-Y   GO-Y-4-3 GO-Y-3-1 TAKE-Y-papers  GO-Y-1-2 )
```

During the process of integrating current intentions with the actions which achieve the normative goal, utility is not directly taken into account. Even if the original goal is still achievable after the integration process, the utility of the new plan(s) is not the same as the utility of the original ones - with the exception of the case in which the original plan already satisfies the normative goal without any modifications. The differential utility between the original plan and the new

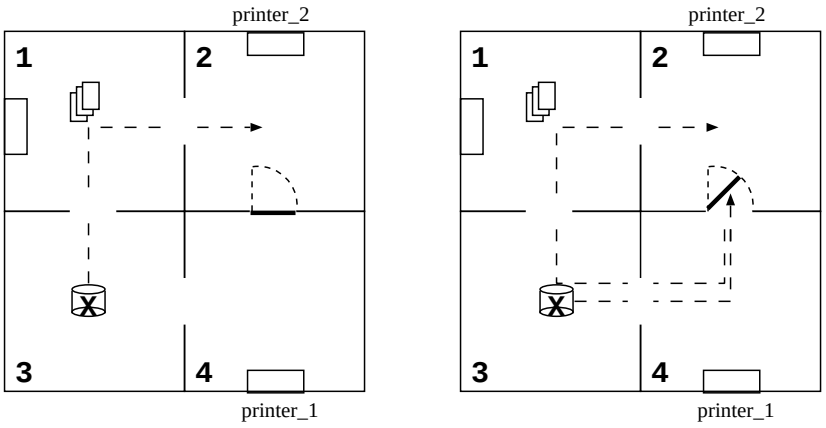

Figure 5.2: The original plan of agent Y for taking papers from room 3 to room 2 (left) and the new plan obtained by integrating the original plan with the plan for opening the door between room 4 and 2 (right).

plans which achieve the normative goal is motivated by the fact that the agent's utility function normally takes the consumption of resources, like time and fuel, into account.

At this point, although he has devised a norm-complying plan, Y is committed to its original plan, and so will remain until the expected utility of the original plan and of the new plans have been evaluated in the anticipatory planning step: only then an informed choice can been performed.
The output of the integration process is passed to the anticipatory planning module, according to the framework introduced in chapter 3, and is evaluated in the light of the reaction of the normative authority (the Supervisor), as encoded in the normative knowledge base of the agent.

In this example, Y can be sanctioned by the Supervisor if it does not comply with the obligation to open the door for his partner, X, when requested. In general, the decision of

- Y may eventually commit to the norm-compliant plan which includes the unlocking action - even if it is more costly in terms of time and fuel - since it decreases the probability of being sanctioned by the Supervisor.

- On the contrary, if the utility function of Y encodes a preference model according to which saving fuel and time is more advantageous than avoiding the sanction, this may lead Y to choose the original plan in which it ignores X's request.

In addition, Y's decision is influenced by the uncertainty related to the possibility that the Supervisor forms the intention to sanction Y and successfully executes the sanctioning plan: after all, the normative authority could be absent

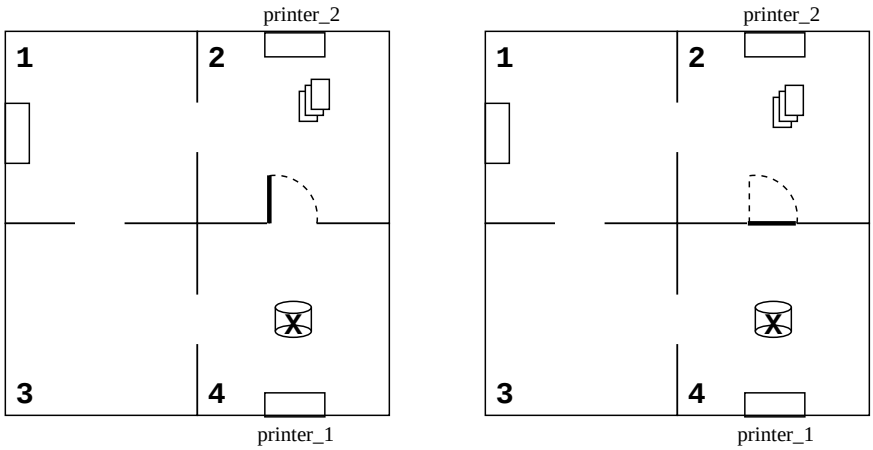

Figure 5.3: A representation of the world from X's subjective point of view and from an objective point of view. Notice that X wrongly believes that the door between room 4 and 2 is open

or could opt for not sanctioning Y in favor of other, more advantageous options according to his preference model.

5.5.2 Replanning Example

Now consider the situation in which X has the same goal, getting the mail from room 2 to room 1, but the same misbelief as well: X 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 5.3). Remember that, differently from Y, X does not have the ability to open the door between 4 and 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, as represented in the first box of figure 5.4:
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 5.4). 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 5.4 and in figure 5.5):
GOX-4-3 GOX-3-1 GOX-1-2 TAKE-MAILX GOX-2-1 PUT-MAILX

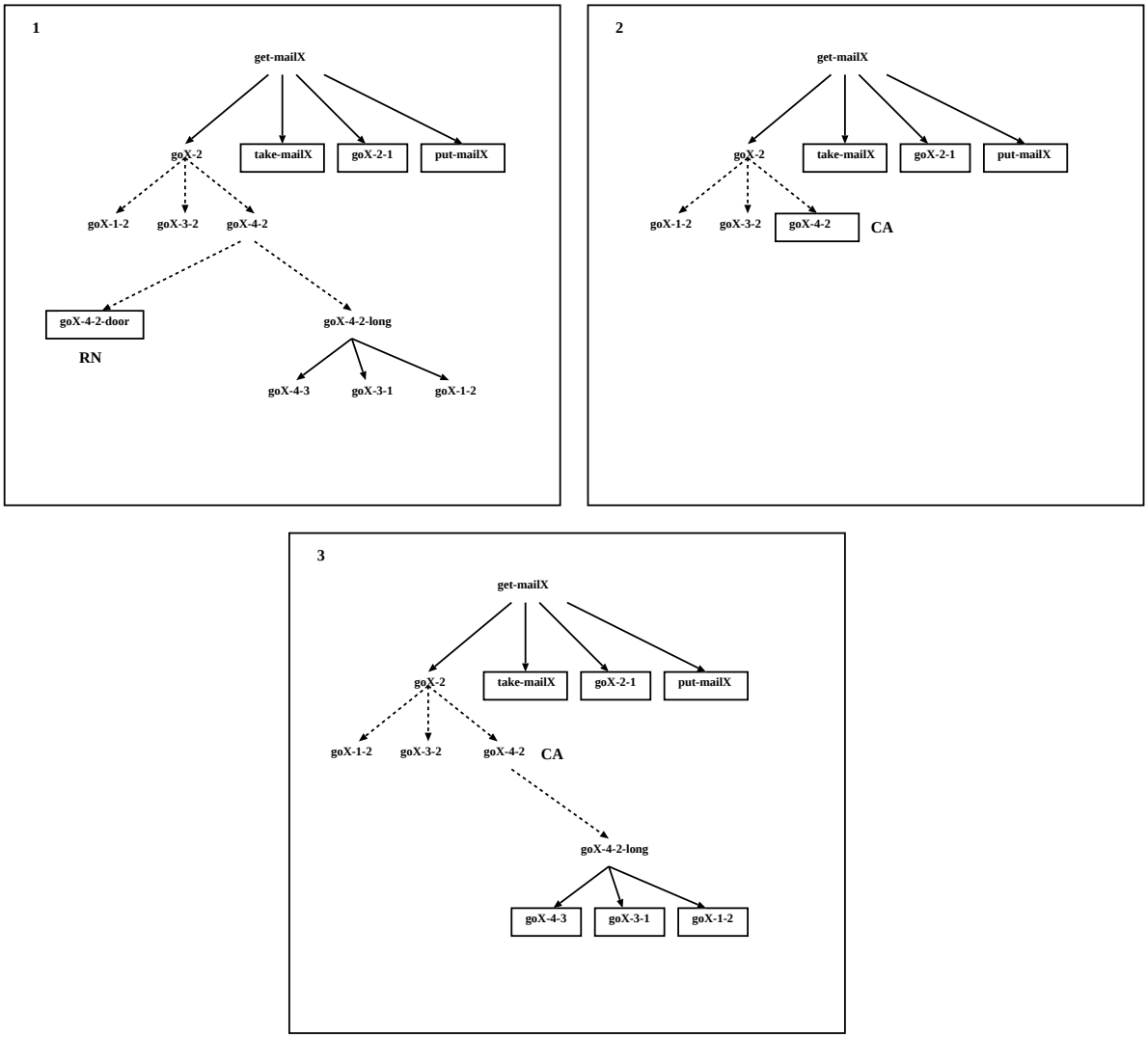

Figure 5.4: A representation of the steps performed by the repair algorithm on the action hierarchy given X's plan. The original plan (1); a node is selected for revision (RN) ; a different instantiation of the RN, the sequence of steps composing the action GOX-4-2-long, has been chosen (3).

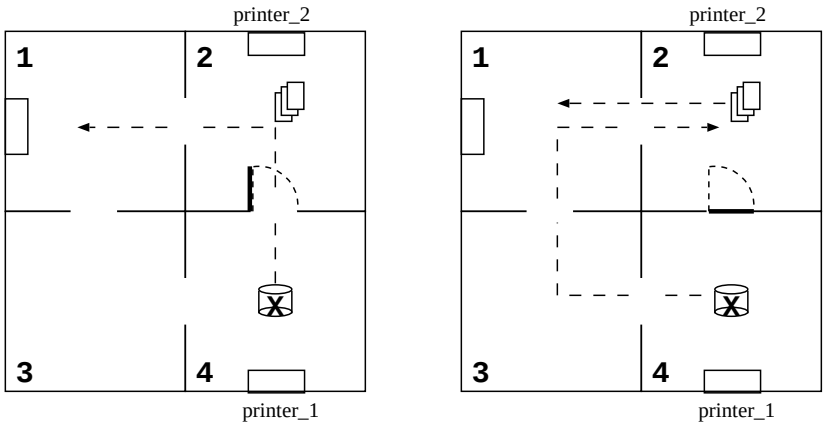

Figure 5.5: The plan of the agent X before replanning (left) and after replanning (right).

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

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.

However, like in the previous example, the decision about whether to commit to the modified plan or not is referred to the anticipation step, where Y evaluates the reaction of the normative authority. Again, after computing the world which would result from the reaction of the normative authority, if Y finds out that it would be sanctioned with a high probability, Y may decide to comply with the norm. So, Y would become committed to the norm-compliant plan, notwithstanding the higher cost associated with this plan: given the probability of Y being sanctioned, the norm-compliant plan now results the most advantageous.

On the contrary, if Y envisages that the probability of being sanctioned is low, and its preference for avoiding the sanction is weak if compared to the preference for saving resources, Y may decide to ignore the norm. In this case, Y would remain committed to the plan it was executing when the norm was triggered.

5.6 Conclusions

The model of normative reasoning and the norm-reactive architecture for intelligent agents we propose here 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 [Conte et al., 1999], 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, i.e., with the line of behavior the agent is currently committed to. This solution is in line with the utilitarian characterization of the deontic notion of duty proposed by [Horty, 1996].

Moreover, this evaluation is not accomplished in isolation from the social context; rather, it is mediated by the simulating the reaction of the normative authority, which indirectly determines the utility of complying with a norm. The anticipatory reasoning on the normative agent's reaction is not inserted in the model as an *ad hoc* solution, but is incorporated in the model as a consequence of the adoption of an interactional framework for social agents.

Chapter 6

Conclusions and Future Work

The main contribution of this thesis is a model of normative reasoning, which is conceived of as a rational, utility-driven process. This model consists of the integration of an interactional framework for social agents with a reactive agent architecture.

The interactional framework is based on a form of social anticipatory reasoning, by which an agent evaluates the reaction of its partners to its own behavior. The reactive agent architectures incorporates decision-theoretic notions, in that both the deliberation and redeliberation processes are driven by utility considerations.

By integrating the interactional framework and the reactive agent architecture, we obtain a model where the agent who is subject to a norm (the bearer) is not obliged, strictly speaking, to obey it: on the contrary, its normative deliberation is based on the probability of being sanctioned or rewarded by the authority who is in charge of the respect of the norm (the normative authority). The behavior of this authority is accounted for by the bearer by means of the anticipatory reasoning mentioned above.

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.

This model of normative reasoning allows for a *flexible*, utilitarian behavior:

the agent's normative deliberation is determined by a number of factors, including the *probability of being sanctioned* or rewarded and the *cost* of complying with the norm. Moreover, these factors are evaluated *contextually* with respect to the agent's current intentions.

6.1 Future Work

The reactive agent model introduced in this thesis rests on the assumption the only external source of goals is provided by norms; apart from norms, goals are assumed to be static. Clearly, this assumption is highly implausible, so it needs to be released by elaborating a more sophisticated notion of commitment which accounts for the role of exogenous goals in a general way. This model should also take sources of non-normative goals into account.

The reactive agent architecture itself is not fully detailed in all its components. In particular a more accurate model of monitoring is needed to account for resource-bounded, fallible sensing in a systematic manner.

Additional work is also needed to integrate the plan repair component and the plan expansion component in order to enable the interleaving between the plan repair and plan expansion phases.

Finally, the replanning module and the normative reasoning process need further integration as well. In particular, the use of plan schemata for planning and replanning must be reconciled with the merged plans generated in the normative reasoning phase.

In a more theoretical perspective, the interaction between normative reasoning and redeliberation has not been investigated yet, and constitutes an important issue for future work.

Future work should also address the characterization and the representation of normative knowledge. In particular, sources of norms should be investigated and related to the taxonomy sketched in Chapter 5. For what concerns the expression of norms in natural language, the relation between the linguistic expression of a norm and its formal representation needs further study.

6.2 Implementation

The work described in this thesis has been implemented in Allegro Common Lisp for Linux. It is constituted by a suite of programs which implement the agent architecture described in Chapter 3 and the reactivity to norms described in Chapter 5.
The integration between these components and the interactional framework described in Chapter 4 has not been completed so far.

Acknowledgements

Appendix A

Office Domain Description

A.1 X's Actions

```
(template-action go
    (x time f)
    (cond   ((x = y)
                (1  (time = time + 15) 1
                    (x = z) 1
                    (f = f - 0.5) 1))
            ((not (x = y))
                (1  (time =  time + 15) 1
                    (f = f - 0.5) 1)))
            ((x = z))
)


(template-action pass
            (x time f open2)
            (cond ((and (x = y)
                        (open2 = 1))
                                (1 (time = time + 15) 1
                                (x = z) 1
                                (f = f - 0.5) 1))
                        ((or (not (x = y))(not (open2 = 1)))
                                (1 (time = time + 15) 1
                                (f = f - 0.5) 1)))
            ((x = z))
)


(add-template 'goX-1-2 go '(1 2 atX))

(add-template 'goX-2-1 go '(2 1 atX))
```

```
(add-template 'goX-1-3 go '(1 3 atX))

(add-template 'goX-3-1 go '(3 1 atX))

(add-template 'goX-4-3 go '(4 3 atX))

(add-template 'goX-3-4 go '(3 4 atX))

(add-template 'goX-1-2 go '(1 2 atX))

(add-template 'goX-2-1 go '(2 1 atX))

(add-template 'goX-2-4-A pass '(2 4 atX))

(add-template 'goX-4-2-A pass '(4 2 atX))


(my-add-action goX-2-3-A
        (atX time f)
        (cond   ((atX = 2)
                    (1  (time = time + 30) 1
                        (atX = 3) 1
                        (f = f - 1) 1))
                ((not (atX = 2))
                    (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atX = 3)) ; goal
        ()
        (list goX-2-1 goX-1-3))


(my-add-action goX-2-3-B
        (atX time f)
        (cond   ((and (atX = 2)(open2 = 1))
                    (1  (time = time + 30) 1
                        (atX = 3) 1
                        (f = f - 1) 1))
                ((and (atX = 2)(open2 = 0))
                    (1  (time = time + 30) 1
                        (f = f - 1) 1))
                (t  (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atX = 3)) ; goal
        ()
        (list goX-2-4-A goX-4-3))


(my-add-action goX-2-3
        (atX time f open2)
        (cond   ((and (atX = 2)(open2 = 1))
                    (1  (time = time + 30) 1
```

```
                        (atX = 3) 1
                        (f = f - 1) 1))
                ((and (atX = 2)(open2 = 0))
                    ((0 1)  (time = time + 30) 1
                            (atX = 3) 1
                            (f = f - 1) 1)
                    ((0 1)  (time = time + 30) 1
                            (atX = 2) 1
                            (f = f - 1) 1))
                (t  (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atX = 3)) ; goal
        (list goX-2-3-A goX-2-3-B))


(my-add-action goX-3-2-A
        (atX time f)
        (cond   ((atX = 3)
                    (1  (time = time + 30) 1
                        (atX = 2) 1
                        (f = f - 1) 1))
                ((not (atX = 3))
                    (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atX = 2)) ; goal
        ()
        (list goX-3-1 goX-1-2))


(my-add-action goX-3-2-B
        (atX time f)
        (cond   ((and (atX = 3)(open2 = 1))
                    (1  (time = time + 30) 1
                        (atX = 2) 1
                        (f = f - 1) 1))
                ((and (atX = 3)(open2 = 0))
                    (1  (time = time + 30) 1
                        (f = f - 1) 1))
                (t  (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atX = 2)) ; goal
        ()
        (list goX-3-4 goX-4-2-A))


(my-add-action goX-3-2
        (atX time f open2)
        (cond   ((and (atX = 3)(open2 = 1))
                    (1  (time = time + 30) 1
                        (atX = 2) 1
                        (f = f - 1) 1))
```

```
                ((and (atX = 3)(open2 = 0))
                    ((0 1)  (time = time + 30) 1
                            (atX = 2) 1
                            (f = f - 1) 1)
                    ((0 1)  (time = time + 30) 1
                            (atX = 3) 1
                            (f = f - 1) 1))
                (t  (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atX = 2)) ; goal
        (list goX-3-2-A goX-3-2-B))


(my-add-action goX-1-4-A
        (atX time f)
        (cond   ((atX = 1)
                    (1  (time = time + 30) 1
                        (atX = 4) 1
                        (f = f - 1) 1))
                ((not (atX = 1))
                    (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atX = 4)) ; goal
        ()
        (list goX-1-3 goX-3-4))


(my-add-action goX-4-1-A
        (atX time f)
        (cond   ((atX = 4)
                    (1  (time = time + 30) 1
                        (atX = 1) 1
                        (f = f - 1) 1))
                ((not (atX = 4))
                    (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atX = 1)) ; goal
        ()
        (list goX-4-3 goX-3-1))


(my-add-action goX-4-2-B
        (atX time f)
        (cond   ((atX = 4)
                    (1  (time = time + 45) 1
                        (atX = 2) 1
                        (f = f - 1.5) 1))
                ((not (atx = 4))
                    (1  (time = time + 45) 1
                        (f = f - 1.5) 1)))
        ((atX = 2)) ; goal
```

```
        ()
        (list goX-4-3 goX-3-1 goX-1-2))


(my-add-action goX-4-2
        (atX time f open2)
        (cond   ((and (atX = 4)(open2 = 1))
                    (1  (time = time + (15 45)) 1
                        (atX = 2) 1
                        (f = f - (0.5 1.5)) 1))
                ((and (atX = 4)(open2 = 0))
                    ((0 1)  (time = time + (15 45)) 1
                            (atX = 4) 1
                            (f = f - (0.5 1.5)) 1)
                    ((0 1)  (time = time + (15 45)) 1
                            (atX = 2) 1
                            (f = f - (0.5 1.5)) 1))
                ((not (atX = 4))
                    (1  (time = time + (15 45)) 1
                        (f = f - (0.5 1.5)) 1)))
        ((atX = 2)) ; goal
        (list goX-4-2-A goX-4-2-B))


(my-add-action goX-1-4-B
         (atX time f)
            (cond ((and (atX = 1)(open2 = 1))
                                (1 (time = time + 30) 1
                                (atX = 4) 1
                                (f = f - 1) 1))
                  ((and (atX = 1)(open2 = 0))
                                (1 (time = time + 30) 1
                                (f = f - 1) 1))
                  (t (1 (time = time + 30) 1
                        (f = f - 1) 1)))
          ((atX = 4)) ; goal
          ()
          (list goX-1-2 goX-2-4-a))


(my-add-action goX-4-1-B
        (atX time f)
        (cond   ((and (atX = 4)(open2 = 1))
                    (1  (time = time + 30) 1
                        (atX = 1) 1
                        (f = f - 1) 1))
                ((and (atX = 4)(open2 = 0))
                    (1  (time = time + 30) 1
                        (f = f - 1) 1))
                (t  (1  (time = time + 30) 1
                        (f = f - 1) 1)))
```

```
        ((atX = 1)) ; goal
        ()
        (list goX-4-2-A goX-2-1))


(my-add-action goX-2-4-B
        (atX time f)
        (cond   ((atX = 2)
                    (1  (time = time + 45) 1
                        (atX = 4) 1
                        (f = f - 1) 1))
                ((not (atX = 2))
                    (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atX = 4)) ; goal
        ()
        (list goX-2-1 goX-1-3 goX-3-4))


(my-add-action goX-2-4
        (atX time f open2)
        (cond   ((and (atX = 2)(open2 = 1))
                    (1  (time = time + (15 45)) 1
                        (atX = 4) 1
                        (f = f - (0.5 1.5)) 1))
                ((and (atX = 2)(open2 = 0))
                    ((0 1)  (time = time + 30) 1
                            (atX = 4) 1
                            (f = f - 1) 1)
                    ((0 1)  (time = time + 30) 1
                            (atX = 2) 1
                            (f = f - 1) 1))
                (t  (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atX = 4)) ; goal
        (list goX-2-4-A goX-2-4-B))


(my-add-action goX-4-1
        (atX time f open2)
        (cond   ((and (atX = 4)(open2 = 1))
                    (1  (time = time + 30) 1
                        (atX = 1 ) 1
                        (f = f - 1) 1))
                ((and (atX = 4)(open2 = 0))
                    ((0 1)  (time = time + 30) 1
                            (atX = 1 ) 1
                            (f = f - 1) 1)
                    ((0 1)  (time = time + 30) 1
                            (atX = 4) 1
                            (f = f - 1) 1))
```

```
                (t  (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atX = 1)) ; goal
        (list goX-4-1-A goX-4-1-B))


(my-add-action goX-1-4
        (atX time f open2)
        (cond   ((and (atX = 1)(open2 = 1))
                    (1  (time = time + 30) 1
                        (atX = 4) 1
                        (f = f - 1) 1))
                ((and   (atX = 1)(open2 = 0))
                    (1 (time = time + 30) 1
                        (atX = 4) 1
                        (f = f - 1) 1)
                        (1 (time = time + 30) 1
                        (atX = 1) 1
                        (f = f - 1) 1))
                  (t (1 (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atX = 4)) ; goal
        (list goX-1-4-A goX-1-4-B))


(my-add-action stayX
        (atX time f open2)
        (cond   (T  (1 (time = time + 1) 1
                    (f = f - 1) 1))))


(my-add-action goX-4
        (atX time f open2)
        (cond   (T  ((0 1)  (time = time + (0 45)) 1
                            (atX = 4) 1
                            (f = f - (0 1.5)) 1)
                    ((0 1)  (time = time + (0 45)) 1
                            (atX = atX) 1
                            (f = f - (0 1.5)) 1) ))
        ((atX = 4)) ; goal
        (list goX-3-4 goX-1-4 goX-2-4 stayX))


(my-add-action goX-3
        (atX time f open2)
        (cond   (T  ((0 1)  (time = time + (0 45)) 1
                            (atX = 3) 1     ;; goal
                            (f = f - (0 1.5)) 1)
                    ((0 1)  (time = time + (0 45)) 1
                            (atX = atX) 1
                            (f = f - (0 1.5)) 1) ))
```

```
        ((atX = 3)) ; goal
        (list goX-4-3 goX-1-3 goX-2-3 stayX))


(my-add-action goX-2
        (atX time f open2)
        (cond  (T   (0 1)   (time = time + (0 45)) 1
                            (atX =  2) 1
                            (f = f - (0 1.5)) 1)
                    ((0 1)  (time = time + (0 45)) 1
                            (atX = atX ) 1
                            (f = f - (0 1.5)) 1)) )
        ((atX =  2)) ; goal
        (list goX-4-2 goX-1-2 goX-3-2 stayX))


(my-add-action goX-1
        (atX time f open2)
        (cond (T    ((0 1)  (time = time + (0 45)) 1
                            (atX =  1) 1
                            (f = f - (0 1.5)) 1)
                    ((0 1)  (time = time + (0 45)) 1
                            (atX = atX ) 1
                            (f = f - (0 1.5)) 1)) )
        ((atX =  1)) ; goal
        (list goX-4-1 goX-3-1 goX-2-1 stayX))


(my-add-action take-mailX
        (atX mail time f)
        (cond   ((and (atX = 2)(mail = 2))
                    (1  (time = time  + 1) 1
                        (mail = 5) 1
                        (f = f - 1) 1))
                ((not (atx = 2))
                    (1  (time = time + 1) 1
                        (f = f - 1) 1)))
        ((mail = 5)) ; goal
)


(my-add-action put-mailX
         (atX time f mail)
         (cond  ((mail = 5)
                    (1  (time = time  + 1) 1
                        (mail = atX) 1
                        (f = f - 0.5) 1))
                ((not (mail = 5))
                    (1  (time = time + 1) 1
                        (f = f - 0.5) 1)))
        ((mail = atX)) ; goal
```

```
)


(my-add-action get-mailX
        (atX time f mail)
        (cond   ((mail = 2)
                    ((0 1)  (time = time + (15 90)) 1
                            (mail = 1) 1
                            (atX = 1) 1
                            (f = f - (0.5 3)) 1)
                    ((0 1)  (time = time + (15 90)) 1
                            (mail = 2) 1
                            (atX = (atX 1)) 1
                            (f = f - (0.5 3)) 1))
                ((not (mail = 2))
                        (1  (time = time + (15 90)) 1
                            (f = f - (0.5 3)) 1)))
        ((mail = 1)) ; goal
        ()
        (list goX-2 take-mailX goX-2-1 put-mailX))
```

A.2 Y's Actions

```
(add-template 'goY-1-2 go '(1 2 atY))

(add-template 'goY-2-1 go '(2 1 atY))

(add-template 'goY-1-3 go '(1 3 atY))

(add-template 'goY-3-1 go '(3 1 atY))

(add-template 'goY-4-3 go '(4 3 atY))

(add-template 'goY-3-4 go '(3 4 atY))

(add-template 'goY-1-2 go '(1 2 atY))

(add-template 'goY-2-1 go '(2 1 atY))

(add-template 'goY-2-4-A pass '(2 4 atY))

(add-template 'goY-4-2-A2 pass '(4 2 atY))


(my-add-action unlockY
        (atY open2 time f)
        (cond   ((atY = 4)
```

```
                    (1  (time = time  + 1) 1
                        (open2 = 1) 1
                        (f = f - 1) 1))
                ((not (atY = 4))
                    (1  (time = time + 1) 1
                        (f = f - 1) 1)))
        ((open2 = 1)) ; goal
)


(my-add-action goY-4-2-A1
        (atY time f)
        (cond   ((atY = 4)
                    (1  (time = time + 16) 1
                        (atY = 2) 1
                        (f = f - 1.5) 1))

                ((not   (atY = 4)) (1 (time = time + 16) 1
                        (f = f - 1.5) 1)))
        ((atY = 2)) ; goal
        ()
        (list unlockY goY-4-2-A2))


(my-add-action goY-4-2-a
        (atY time f open2)
        (cond   ((and (atY = 4)(open2 = 1))
                    (1  (time = time + (15 16)) 1
                        (atY = 2) 1
                        (f = f - (0.5 1.5)) 1))
                ((and (atY = 4)(open2 = 0))
                    ((0 1)  (time = time + (15 16)) 1
                            (atY =  2) 1
                            (f = f - (0.5 1.5)) 1)
                    ((0 1)  (time = time + (15 16)) 1
                            (atY =  4) 1
                            (f = f - (0.5 1.5)) 1))
                ((not (atY = 4)) (1 (time = time + (15 16)) 1
                            (f = f - (0.5 1.5)) 1)))
        ((atY = 2)) ; goal
        (list goY-4-2-A2 goY-4-2-a1))


(my-add-action goY-2-3-A
        (atY time f)
        (cond   ((atY = 2)
                    (1  (time = time + 30) 1
                        (atY = 3) 1
                        (f = f - 1) 1))
                ((not (atY = 2))
                    (1  (time = time + 30) 1
```

```
                    (f = f - 1) 1)))
        ((atY = 3)) ; goal
        ()
        (list goY-2-1 goY-1-3))


(my-add-action goY-2-3-B
        (atY time f)
        (cond   ((and (atY = 2)(open2 = 1))
                    (1  (time = time + 30) 1
                        (atY = 3) 1
                        (f = f - 1) 1))
                ((and (atY = 2)(open2 = 0))
                    (1  (time = time + 30) 1
                        (f = f - 1) 1))
                (t  (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atY = 3)) ; goal
        ()
        (list goY-2-4-A goY-4-3))


(my-add-action goY-2-3
        (atY time f open2)
        (cond   ((and (atY = 2)(open2 = 1))
                    (1  (time = time + 30) 1
                        (atY = 3) 1
                        (f = f - 1) 1))
                ((and (atY = 2)(open2 = 0))
                    ((0 1)  (time = time + 30) 1
                            (atY = 3) 1
                            (f = f - 1) 1)
                    ((0 1)  (time = time + 30) 1
                            (atY = 2) 1
                            (f = f - 1) 1))
                (t  (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atY = 3)) ; goal
        (list goY-2-3-A goY-2-3-B))


(my-add-action goY-3-2-A
        (atY time f)
        (cond   ((atY = 3)
                    (1  (time = time + 30) 1
                        (atY = 2) 1
                        (f = f - 1) 1))
                ((not (atY = 3))
                    (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atY = 2)) ; goal
```

```
        ()
        (list goY-3-1 goY-1-2))


(my-add-action goY-3-2-B
        (atY time f)
        (cond   ((atY = 3)
                    (1  (time = time + (30 31)) 1
                        (atY = 2) 1
                        (f = f - (0.5 2.5)) 1))
                ((not (atY = 3))
                    (1  (time = time + (30 31)) 1
                        (f = f - (0.5 2.5)) 1)))
        ((atY = 2)) ; goal
        ()
        (list goY-3-4 goY-4-2-A))


(my-add-action goY-3-2
        (atY time f open2)
        (cond   ((atY = 3)
                    (1  (time = time + (30 31)) 1
                        (atY = 2) 1
                        (f = f - (1 2.5)) 1))
                ((not (atY = 3))
                    ((0 1)  (time = time + (30 31)) 1
                            (atY = 2) 1
                            (f = f - (1 2.5)) 1)
                    ((0 1)  (time = time + (30 31)) 1
                            (atY = atY) 1
                            (f = f - (1 2.5)) 1))
                (t  (1  (time = time + (30 31)) 1
                        (f = f - (1 2.5)) 1)))
        ((atY = 2))
        (list goY-3-2-A goY-3-2-B))


(my-add-action goY-1-4-A
        (atY time f)
            (cond   ((atY = 1)
                        (1  (time = time + 30) 1
                            (atY = 4) 1
                            (f = f - 1) 1))
                    ((not (atY = 1))
                        (1  (time = time + 30) 1
                            (f = f - 1) 1)))
        ((atY = 4)) ; goal
        ()
        (list goY-1-3 goY-3-4))
```

```
(my-add-action goY-4-1-A
        (atY time f)
        (cond   ((atY = 4)
                    (1  (time = time + 30) 1
                        (atY = 1) 1
                        (f = f - 1) 1))
                ((not (atY = 4))
                    (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atY = 1)) ; goal
        ()
        (list goY-4-3 goY-3-1))


(my-add-action goY-4-2-B
        (atY time f)
        (cond   ((atY = 4)
                    (1  (time = time + 45) 1
                        (atY = 2) 1
                        (f = f - 1.5) 1))
                ((not (atx = 4))
                    (1  (time = time + 45) 1
                        (f = f - 1.5) 1)))
        ((atY = 2)) ; goal
        ()
        (list goY-4-3 goY-3-1 goY-1-2))


(my-add-action goY-4-2
        (atY time f open2)
        (cond   ((and (atY = 4)(open2 = 1))
                    (1  (time = time + (15 45)) 1
                        (atY = 2) 1
                        (f = f - (0.5 1.5)) 1))
                ((and (atY = 4)(open2 = 0))
                    ((0 1)  (time = time + (15 45)) 1
                            (atY = 4) 1
                            (f = f - (0.5 1.5)) 1)
                    ((0 1)  (time = time + (15 45)) 1
                            (atY = 2) 1
                            (f = f - (0.5 1.5)) 1))
                ((not (atY = 4))
                    (1  (time = time + (15 45)) 1
                        (f = f - (0.5 1.5)) 1)))
        ((atY = 2)) ; goal
        (list goY-4-2-A goY-4-2-B))


(my-add-action goY-1-4-B
        (atY time f)
        (cond   ((and (atY = 1) (open2 = 1))
```

```
                    (1  (time = time + 30) 1
                        (atY = 4) 1
                        (f = f - 1) 1))
                ((and (atY = 1)(open2 = 0))
                    (1  (time = time + 30) 1
                        (f = f - 1) 1))
                (t  (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atY = 4)) ; goal
        ()
        (list goY-1-2 unlockY goY-2-4-a))


(my-add-action goY-4-1-B
        (atY time f)
        (cond   ((and (atY = 4)(open2 = 1))
                    (1  (time = time + 30) 1
                        (atY = 1) 1
                        (f = f - 1) 1))
                ((and (atY = 4)(open2 = 0))
                    (1  (time = time + 30) 1
                        (f = f - 1) 1))
                (t  (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atY = 1)) ; goal
        ()
        (list goY-4-2-A goY-2-1))


(my-add-action goY-2-4-B
        (atY time f)
        (cond   ((atY = 2)
                    (1  (time = time + 45) 1
                        (atY = 4) 1
                        (f = f - 1) 1))
                ((not (atY = 2))
                    (1  (time = time + 30) 1
                        (f = f - 1) 1)))
        ((atY = 4)) ; goal
        ()
        (list goY-2-1 goY-1-3 goY-3-4))


(my-add-action goY-2-4
        (atY time f open2)
        (cond   ((and (atY = 2)(open2 = 1))
                    (1  (time = time + (30 45)) 1
                        (atY = 4) 1
                        (f = f - (1 2)) 1))
                ((and (atY = 2)(open2 = 0))
                    ((0 1)  (time = time + 30) 1
```

```
                            (atY = 4) 1
                            (f = f - 1) 1)
                            ((0 1) (time = time + 30) 1
                            (atY = 2) 1
                            (f = f - 1) 1))
                    (t  (1  (time = time + 30) 1
                            (f = f - 1) 1)))
        ((atY = 4)) ; goal
        (list goY-2-4-A goY-2-4-B))


(my-add-action goY-4-1
        (atY time f open2)
        (cond   ((and (atY = 4)(open2 = 1))
                    (1  (time = time + 30) 1
                        (atY = 1) 1
                        (f = f - 1) 1))
                ((and (atY = 4)(open2 = 0))
                    ((0 1)  (time = time + 30) 1
                            (atY = 1) 1
                            (f = f - 1) 1)
                    ((0 1)  (time = time + 30) 1
                            (atY = 4) 1
                            (f = f - 1) 1))
                    (t  (1  (time = time + 30) 1
                            (f = f - 1) 1)))
        ((atY = 1)) ; goal
        (list goY-4-1-A goY-4-1-B))


(my-add-action goY-1-4
        (atY time f open2)
        (cond   ((and (atY = 1)(open2 = 1))
                    (1  (time = time + 30) 1
                        (atY = 4) 1
                        (f = f - 1) 1))
                ((and (atY = 1)(open2 = 0))
                    ((0 1)  (time = time + 30) 1
                            (atY = 4) 1
                            (f = f - 1) 1)
                    ((0 1)  (time = time + 30) 1
                            (atY = 1) 1
                            (f = f - 1) 1))
                    (t  (1  (time = time + 30) 1
                            (f = f - 1) 1)))
        ((atY = 4)) ; goal
        (list goY-1-4-A goY-1-4-B))


(my-add-action stayY
        (atY time f open2)
```

```
        (cond   (T  (1 (time = time + 15) 1
                       (f = f - 1) 1))))


(my-add-action goY-4
        (atY time f open2)
        (cond   (T  ((0 1)  (time = time + (0 45)) 1
                            (atY = 4) 1     ;; goal
                            (f = f - (0 1.5)) 1)
                    ((0 1)  (time = time + (0 45)) 1
                            (atY = atY) 1
                            (f = f - (0 1.5)) 1)))
        ((atY = 4)) ; goal
        (list goY-3-4 goY-1-4 goY-2-4 stayY))


(my-add-action goY-3
        (atY time f open2)
        (cond   (T  ((0 1)  (time = time + (0 45)) 1
                            (atY = 3) 1     ;; goal
                            (f = f - (0 1.5)) 1)
                    ((0 1)  (time = time + (0 45)) 1
                            (atY = atY) 1
                            (f = f - (0 1.5)) 1)))
        ((atY = 3)) ; goal
        (list goY-4-3 goY-1-3 goY-2-3 stayY))


(my-add-action goY-2
        (atY time f open2)
        (cond (T    ((0 1)  (time = time + (0 45)) 1
                            (atY =  2) 1    ;; goal
                            (f = f - (0 2.5)) 1)
                    ((0 1)  (time = time + (0 45)) 1
                            (atY = atY) 1
                            (f = f - (0 2.5)) 1)) )
        ((atY =  2)) ; goal
        (list goY-4-2 goY-1-2 goY-3-2 stayY))


(my-add-action goY-1
        (atY time f open2)
        (cond   (T  ((0 1)  (time = time + (0 45)) 1
                            (atY = 1) 1     ;; goal
                            (f = f - (0 1.5)) 1)
                    ((0 1)  (time = time + (0 45)) 1
                            (atY = atY) 1
                            (f = f - (0 1.5)) 1)) )
        ((atY = 1)) ; goal
        (list goY-4-1 goY-3-1 goY-2-1 stayY))
```

```
(my-add-action openY
        (atY time f open2)
        (cond   (T  ((0 1)  (time = time + (15 90)) 1
                            (open2 = 1) 1
                            (atY = 2) 1
                            (f = f - (1 3)) 1)
                    ((0 1)  (time = time + (15 90)) 1
                            (open2 = 0) 1
                            (atY = atY) 1
                            (f = f - (1 3)) 1)))
        ((open2 = 1)) ; goal
        ()
        (list goY-4 unlockY))


(my-add-action lockY
        (atY open2 time f)
        (cond   ((atY = 2)
                    (1  (time = time  + 1) 1
                        (open2 = 0) 1   ;; goal
                        (f = f - 1) 1))
                ((not (atY = 2))
                    (1  (time = time + 1) 1
                        (f = f - 1) 1)))
        ((open2 = 0)) ; goal
)


(my-add-action  getY
        (aty time f)
        (cond   ((and (ObjAt2 > 0)(atY = 2)(loadY = 0))
                    (1  (time = time  + 1) 1
                        (loadY = 1) 1
                        (ObjAt2 = ObjAt2 - 1) 1
                        (f = f - 1) 1))
                ((not (or (loadY = 0)(atY = 2)(ObjAt2 > 0)))
                    (1  (time = time + 1) 1
                    (f = f - 1) 1)))
        ((ObjAt2 = ObjAt2 - 1)) ; goal
)


(my-add-action  putY
        (aty time f)
        (cond   ((and (atY = 3)(loadY = 1))
                    (1  (time = time  + 1) 1
                        (loadY = 0) 1
                        (ObjAt4 = ObjAt4 + 1) 1
                        (f = f - 1) 1))
                ((not (or (loadY = 1)(atY = 3)))
```

```
                    (1  (time = time + 1) 1
                        (loadY = 0) 1
                        (f = f - 1) 1)))
        ((ObjAt4 = ObjAt4 + 1)) ; goal
)


(my-add-action closeY
        (atY time f open2)
        (cond   (T  ((0 1)  (time = time + (15 90)) 1
                            (open2 =  0) 1
                            (atY = 2) 1
                            (f = f - (0.5 3)) 1)
                    ((0 1)  (time = time + (15 90)) 1
                            (open2 =  1) 1
                            (atY = atY) 1
                            (f = f - (0.5 3)) 1) ))
        ((open2 =  0)) ; goal
        ()
        (list goY-2 lockY))
```

Appendix B

MRE Experiment Description

```
(my-add-action say-not-critical-without-initiative
        (time risk critical reponsib res)
        (cond   ((critical = 0)
                    (1  (time = time + 50) 1
                        (risk = 1) 1
                        (responsib = 0) 1
                        (res = res - 10) 1))
                ((critical = 1)
                    (1  (time = time + 70) 1
                        (risk = 4) 1
                        (responsib = 0) 1
                        (res = res - 10) 1)))
        ((risk = 1))
)


(my-add-action say-not-critical-with-initiative
        (time risk reponsib critical res)
        (cond   ((critical = 0)
                    (1  (time = time + 50) 1
                        (risk = 1) 1
                        (responsib = 1) 1
                        (res = res - 10) 1))
                ((critical = 1)
                    (1  (time = time + 70) 1
                        (risk = 4) 1
                        (responsib = 1) 1
                        (res = res - 10) 1)))
        ((risk = 1))
)

(my-add-action say-not-critical
        (time risk reponsib critical res)
        (cond   ((critical = 0)
```

```
                    (1  (time = time + 50) 1
                        (risk = 1) 1
                        (responsib = (0 1)) 1
                        (res = res - 10) 1))
                ((critical = 1)
                        (1  (time = time + 70) 1
                            (risk = 4) 1
                            (responsib = (0 1)) 1
                            (res = res - 10) 1)))
        ((risk = 1))
        (list say-not-critical-without-initiative say-not-critical-with-initiative)
)


(my-add-action say-critical-without-initiative
        (time risk reponsib res critical explicit)
        (cond   ((critical = 0)
                    (1  (time = time + 50) 1
                        (risk = 1) 1
                        (responsib = 0) 1
                        (explicit = 1) 1
                        (res = res - 20) 1))
                ((critical = 1)
                    (1  (time = time + 50) 1
                        (risk = 3) 1
                        (responsib = 0) 1
                        (explicit = 1) 1
                        (res = res - 20) 1)))
                ((risk = 2))
)


(my-add-action say-critical-with-initiative
        (time risk reponsib res critical explicit)
        (cond   ((critical = 0)
                    (1  (time = time + 50) 1
                        (risk = 1) 1
                        (responsib = 1) 1
                        (explicit = 0) 1
                        (res = res - 20) 1))
                ((critical = 1)
                    (1  (time = time + 50) 1
                        (risk = 2) 1
                        (responsib = 1) 1
                        (explicit = 0) 1
                        (res = res - 20) 1)))
        ((risk = 2))
)


(my-add-action say-critical
```

```
        (time risk reponsib res critical explicit)
        (cond   ((critical = 0)
                    (1  (time = time + 50) 1
                        (risk = 1) 1
                        (responsib = (0 1)) 1
                        (explicit = 0) 1
                        (res = res - 20) 1))
                ((critical = 1)
                    (1  (time = time + 50) 1
                        (risk = (2 3)) 1
                        (responsib = (0 1)) 1
                        (explicit = 0) 1
                        (res = res - 20) 1)))
        ((risk <= 2))
        (list say-critical-without-initiative say-critical-with-initiative))


(my-add-action give-opinion-kid
        (time risk responsib res critical explicit)
        (cond   ((critical = 0)
                    (1  (time = time + 50) 1
                        (risk = 1) 1
                        (responsib = (0 1)) 1
                        (explicit = (0 1)) 1
                        (res = res - (10 20)) 1))
                ((critical = 1)
                    ((0 1)  (time = time + (50 70)) 1
                            (risk =  4) 1
                            (responsib = (0 1)) 1
                            (explicit = (0 1)) 1
                            (res = res - (10 20)) 1)
                    ((0 1)  (time = time + (50 70)) 1
                            (risk = (2 3)) 1
                            (responsib = (0 1)) 1
                            (explicit = (0 1)) 1
                            (res = res - (10 20)) 1)))
        ((risk <= 2))
        (list say-critical say-not-critical))


(my-add-action reaction
        (time upset)
        (cond   ((explicit = 0)
                (1  (time = time + 2) 1
                    (upset = 0) 1))
                    ((explicit = 1)
                (1  (time = time + 2) 1
                    (upset = 1) 1))))

(my-add-action give-opinion-driver
    (time res)
```

```
    (cond   (t  (1  (time = time + 10) 1
                    (res = res - 10) 1))))


(my-add-action give-complex-opinion
        (time risk responsib res critical explicit)
        (cond   (t  (1  (time = time + 60) 1
                        (risk = 1) 1
                        (responsib = (0 1)) 1
                        (explicit = (0 1)) 1
                        (res = res - (20 30)) 1))
                ((critical = 0)
                    ((0 1)  (time = time + (60 80)) 1
                            (risk =  4) 1
                            (responsib = (0 1)) 1
                            (explicit = (0 1)) 1
                            (res = res - (20 30)) 1)
                    ((0 1)  (time = time + (60 80)) 1
                            (risk = (2 3)) 1
                            (responsib = (0 1)) 1
                            (explicit = (0 1)) 1
                            (res = res - (20 30)) 1)))
        ((risk <= 2))
        ()
        (list give-opinion-driver give-opinion-kid))
```

List of Figures

Bibliography

[A. Ortony and Collins, 1988] A. Ortony, G. C. and Collins, A. (1988). *Cognitive Stucture of Emotions*. Cambrigde University Press.

[Airenti et al., 1993] Airenti, G., Bara, B., and Colombetti, M. (1993). Conversational and behavior games in the pragmatics of discourse. *Cognitive Science*, 17:197–256.

[Allwood, 1994] Allwood, J. (1994). Obligations and options in dialogue. *THINK*, 3.

[Alterman, 1988] Alterman, R. (1988). Adaptive planning. *Cognitive Science*, pages 393–421.

[Alterman and Garland, 2000] Alterman, R. and Garland, A. (2000). Convention in joint activity. *Cognitive Science*, (25, 611–657).

[Ardissono et al., 2000] Ardissono, L., Boella, G., and Lesmo, L. (2000). A plan based agent architecture for interpreting natural language dialogue. *to appear in International Journal of Human-Computer Studies*.

[Austin, 1962] Austin, J. L. (1962). *How to Do Things with Words*. Harvard University Press, Cambridge, Mass.

[Belnap and Perloff, 1988] Belnap, N. and Perloff, M. (1988). Seing to it that:a canonical form for agentives. *Theoria*, 51:175–199.

[Blythe, 1999] Blythe, J. (1999). Decision-theoretic planning. *AI Magazine*, 20(2).

[Boella and Damiano, 2000] Boella, G. and Damiano, R. (2000). Communication and cooperation among agents. In *Proc. of Gotalog 2000 Workshop on the Semantics and the Pragmatics of Dialogue.*, Gothenburg (Sweden).

[Boella et al., 2000a] Boella, G., Damiano, R., and Lesmo, L. (2000a). 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 et al., 2000b] Boella, G., Damiano, R., and Lesmo, L. (2000b). Social goals in conversational cooperation. In *Proc. of 1st Sigdial Workshop on Dialogue and Discourse.*, Hong Kong.

[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.

[Boman, 1999] Boman, M. (1999). Norms in artificial decision making. *Artificial Intelligence and Law*, 7(1):17–35.

[Boutilier, 2000] Boutilier, C. (2000). Approximately optimal monitoring of plan preconditions. In *Proc. of UAI2000*.

[Bratman, 1990] Bratman, M. E. (1990). What is intention? In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in communication*, pages 15–32. MIT Press.

[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.

[Brennan and Ohaeri, 1999] Brennan, S. E. and Ohaeri, J. O. (1999). Why do electronic conversations seem less polite? In *Proc. of the 12th International Joint Conference on Work Activities, Coordination and Cooperation*.

[Brown and Levinson, 1987] Brown, P. and Levinson, S. C. (1987). *Politeness: some universals on language usage*. Cambridge University Press, Cambridge.

[Bunt, 1994] Bunt, H. (1994). Context and dialogue control. *Think*, 3:19–31.

[Castelfranchi, 1998] Castelfranchi, C. (1998). Modeling social action for AI agents. *Artificial Intelligence*, 103:157–182.

[Castelfranchi et al., 2000] Castelfranchi, C., Dignum, F., Jonker, C. M., and Treur, J. (2000). Deliberate normative agents: Principles and architecture. In Jennings, N. and Lespérance, Y., editors, *Intelligent Agents VI — Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin.

[Chellas, 1980] Chellas, B. (1980). *Modal Logic: An Introduction*. Cambridge University Press.

[Clark, 1996] Clark, H. C. (1996). *Using Language*. Cambridge University Press.

[Cohen and Levesque, 1990a] Cohen, P. R. and Levesque, H. J. (1990a). Intention is choice with commitment. *Artificial Intelligence*, 42:213–261.

[Cohen and Levesque, 1990b] Cohen, P. R. and Levesque, H. J. (1990b). Rational interaction as the basis for communication. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in communication*, pages 221–255. MIT Press.

[Conte and Castelfranchi, 1995] Conte, R. and Castelfranchi, C. (1995). *Cognitive and Social Action*. UCL Press.

[Conte et al., 1998] Conte, R., Castelfranchi, C., and Dignum, F. (1998). Autonomous norm-acceptance. In Mueller, J. P., Singh, M., and Rao, A., editors, *Intelligent Agents V — Proc. of 5th Int. Workshop on Agent Theories, Architectures, and Languages (ATAL-98)*. Springer Verlag, Berlin.

[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.

[Dignum, 1996] Dignum, F. (1996). Autonomous agents and social norms. In *ICMAS'96 Workshop on Norms, Obligations and Conventions*.

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.

[Gmytrasiewicz and Durfee, 2000] Gmytrasiewicz, P. and Durfee, E. (2000). Rational coordination in multi-agent environments. *Autonomous Agents and Multi-Agent Systems*, 3(4):319–350.

[Gmytrasiewicz and Durfee, 1995] Gmytrasiewicz, P. J. and Durfee, E. H. (1995). Formalization of recursive modeling. In *Proc. of first ICMAS-95*.

[Gmytrasiewicz and Lisetti, 2000] Gmytrasiewicz, P. J. and Lisetti, C. L. (2000). Using decision theory to formalize emotions for multi-agent systems. In *Proceedings of 2nd ICMAS-2000 Workshop on Game Theoretic and Decision Theoretic Agents*, pages 207–222, Boston.

[Goffman, 1967] Goffman, E. (1967). *Interaction Ritual*. Penguin, Harmondsworth.

[Goffman, 1981] Goffman, E. (1981). *Forms of Talk*. University of Pennsylavania Press.

[Green and Carberry, 1999] Green, N. and Carberry, S. (1999). Interpreting and generating indirect answers. *Computational Linguistics*, 25(3):389–435.

[Grosz and Kraus, 1996] Grosz, B. and Kraus, S. (1996). Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357.

[Grosz and Kraus, 1998] Grosz, B. and Kraus, S. (1998). The evolution of Shared Plans. In Rao, A. and Wooldridge, M., editors, *Foundations and Theories of Rational Agencies*. to appear.

[Grosz, 1981] Grosz, B. J. (1981). Focusing and description in natural language dialogues. In Webber, B., Joshi, A., and Sag, I., editors, *Elements of Discourse Understanding*, pages 85–105. Cambridge University Press.

[Grosz and Sidner, 1986] Grosz, B. J. and Sidner, C. L. (1986). Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12:175–204.

[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 Int. Conference on Artificial Intelligence Planning Systems*, pages 266–271, Menlo Park, CA.

[Haigh and Veloso, 1996] Haigh, K. Z. and Veloso, M. (1996). Interleaving planning and robot execution for asynchronous user requests. In *Planning with Incomplete Information for Robot Problems: Papers from the 1996 AAAI Spring Symposium*, pages 35–44. AAAI Press, Menlo Park, California.

[Hammond, 1987] Hammond, K. (1987). Explaining and repairing plans that fail. In *In Proceeding of IJCAI-87*, pages 109–114. Morgan Kaufmann.

[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.

[Hogg and Jennings, 2000a] Hogg, L. and Jennings, N. R. (2000a). Variable sociability in agent-based decision making. In Jennings, N. and Lespérance, Y., editors, *Intelligent Agents VI — Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin.

[Hogg and Jennings, 2000b] Hogg, L. and Jennings, N. R. (2000b). Variable sociability in agent-based decision-making. In Lesperance, Y. and Jennings, N., editors, *Proc. 6th Int. Workshop on Agent Theories Architectures and Languages (ATAL 99)*, Berlin. Springer-Verlag.

[Horty and Belnap, 1995] Horty, J. and Belnap, N. (1995). Journal of philosphical logic. *The deliberative stit: a study of action, omission, ability and obligation*, 24:583–644.

[Horty, 1996] Horty, J. F. (1996). Agency and obligation. *Synthése*, 108:269–307.

[Horty and Pollack, 2001] Horty, J. F. and Pollack, M. E. (2001). Evaluating new options in the context of existing plans. *Artificial Intelligence*, 127(2):199–220.

[Isozaki and Katsuno, 2000] Isozaki, H. and Katsuno, H. (2000). Observability-based nested belief computation for multiagent systems and its formalization. In Jennings, N. and Lespérance, Y., editors, *Intelligent Agents VI — Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin.

[Kanger, 1971] Kanger, S. (1971). New foundations for ethical theory. In Hilpinen, R., editor, *Deontic Logic*. D. Reiderl, Dordrecht.

[Kushmerick et al., 1994] Kushmerick, N., Hanks, S., and Weld, D. (1994). An algorithm for probabilistic, least commitment planning. In *Proc. of the 12th National Conference on Artificial Intelligence*, pages 1073–1078.

[Lang et al., 2001] Lang, J., van der Torre, L., and Weydert, E. (2001). Utilitarian desires.

[Levinson, 1983] Levinson, S. C. (1983). *Pragmatics*. Cambridge University Press, Cambridge.

[Luce and Raiffa, 1957] Luce, D. and Raiffa, H. (1957). *Games and Decisions*. John Wiley and Sons, Inc.

[Merritt, 1976] Merritt, M. (1976). On question following question (in service encounters). *Language in Society*, V:315–357.

[Meyer and Weiringa, 1991] Meyer, J.-J. C. and Weiringa, R. (1991). Deontic logics: A coincise overview. In *Proc. DEON91*, pages 2–14.

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

[Nau et al., 2001] Nau, D., Munoz-Avila, H., Cao, Y., Lotem, A., and Mitchell, S. (2001). Total-order planning with partially ordered subtasks. In *Proc. IJCAI-01*, pages 425–430.

[Parsons et al., 1999] Parsons, S., Pettersson, O., Saffiotti, A., and Wooldridge, M. (1999). Robots with the best of intentions. In Wooldridge, M. and Veloso, M., editors, *Artificial Intelligence Today*, number 1600 in LNAI. Springer-Verlag, Berlin, DE.

[Parsons et al., 2000] Parsons, S., Pettersson, O., Saffiotti, A., and Wooldridge, M. (2000). Intention reconsideration in theory and practice. In *Proc. of ECAI 2000*, Berlin.

[Penberthy and Weld, 1992] Penberthy, J. and Weld, D. (1992). Ucpop: A sound, complete, partial-order planner for adl. In *Proc. Third International Conference of Principle of Knowledge Representation and Reasoning*, pages 103–114.

[Picard, 1997] Picard, R. (1997). *Affective Computing*. MIT Press.

[Poesio and Traum, 1998] Poesio, M. and Traum, D. (1998). Towards an axiomatization of dialogue acts. In Hulstijn, J. and Nijholt, A., editors, *Proceedings of the Twente Workshop on the Formal Semantics and Pragmatics of Dialogues*, pages 207–222, Enschede.

[Pollack, 1990] Pollack, M. E. (1990). Plans as complex mental attitudes. In Cohen, P. R., Morgan, J., and Pollack, M. E., editors, *Intentions in communication*, pages 77–103. MIT Press.

[Pollack and McCarthy, 1999] Pollack, M. E. and McCarthy, C. E. (1999). Towards focused plan monitoring: A technique and an application to mobile robots. In *Proc. of the IEEE Int. Symposium on Computational Intelligence in Robotics and Automation: CIRA99*, pages 144–149. Kluwer.

[Prior, 1991] Prior, A. (1991). *Past, Present and Future*. Clarendon Press, Oxford.

[Rao and Georgeff, 1991] Rao, A. and Georgeff, M. P. (1991). Modeling rational agents within a BDI-architecture. In *Proc. 2th Int. Conf. Principles of Knowledge Representation and Reasoning (KR:91)*, pages 473–484, Cambridge, MA.

[Rickel et al., 2001] Rickel, J., Gratch, J., Hill, R., Marsella, S., and Swartout, W. (2001). Steve goes to bosnia: Towards a new generation of virtual humans for interactive experiences. In *Proc. AAAI Spring Symposium on AI and Interactive Entertainment*.

[Rickel and Johnson, 1999] Rickel, J. and Johnson, W. (1999). Animated agents for procedural training in virtual reality: Perception, cognition and motor control. *Applied Artificial Intelligence*, pages 343–382.

[Sacerdoti, 1977] Sacerdoti, E. D. (1977). *A Structure for Plans and Behavior*. American Elsevier, New York.

[Saffiotti, 1998] Saffiotti, A. (1998). *Autonomous Robot Navigation*. Ph.d. thesis, Uninversité Libre de Bruxelles, Brussels, Belgium.

[Searle, 1969] Searle, J. R. (1969). *Speech Acts: an Essay in the Philosophy of Language*. Cambridge University Press, Cambridge, England.

[Searle, 1975] Searle, J. R. (1975). Indirect speech acts. In Cole, P. and Morgan, J., editors, *Syntax and Semantics: Speech Acts*, volume 3, pages 59–82. Academic Press, New York.

[Searle, 1992] Searle, J. R. (1992). *(On) Searle on Conversation*. Benjamins, Amsterdam.

[Shoham, 1993a] Shoham, Y. (1993a). Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92.

[Shoham, 1993b] Shoham, Y. (1993b). Agent-oriented programming. *Artificial Intelligence*, (60, 51–93).

[Shoham and Tennenholtz, 1997] Shoham, Y. and Tennenholtz, M. (1997). On the emergence of social conventions: Modeling, analysis and simulations. *Artificial Intelligence*, 94(1–2):139–166.

[T. Statulat and Enjalbert, 2001] T. Statulat, F. C.-D. and Enjalbert, P. (2001). Norms and time in agent-based systems. In *Proc. of the ICAIL 2001*, pages 178–185, Saint Louis, Missouri, USA.

[Tenenberg, 1991] Tenenberg, J. (1991). In *Reasoning about Plans*. Morgan-Kaufman, San Mateo, CA.

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

[Traum, 1999] Traum, D. (1999). Speech acts for dialogue agents. In Wooldridge, M. and Rao, A., editors, *Foundations and Theories of Rational Agents*, pages 169–201. Kluwer.

[Traum and Rickel, 2001] Traum, D. and Rickel, J. (2001). Embodied agents for multi-party dialogue in immersive virtual worlds. In *to be added*.

[Traum and Allen, 1994] Traum, D. R. and Allen, J. F. (1994). Discourse obligations in dialogue processing. In *Proc. 32nd Annual Meeting of ACL*, pages 1–8, Las Cruces, New Mexico.

[van der Krogt et al., 2000] van der Krogt, R., Bos, A., de Weerdt, M., and Witteveen, C. (2000). An algorithm for replanning. In van den Bosch, A. and Weigand, H., editors, *Proceedings of the Twelfth Belgium-Netherlands Artificial Intelligence Conference BNAIC '00*, pages 21–28, Tilburg.

[van der Torre, 1994] van der Torre, L. (1994). Violated obligations in a defeasible deontic logic. In *Proc. ECAI94*, pages 371–375.

[von Wright, 1951] von Wright, G. H. (1951). *An Essay in Modal Logic*. North-Holland, Amsterdam.

[Weerdt et al., 2000] Weerdt, M., Bos, A., Tonino, H., and Witteveen, C. (2000). A plan fusion algorithm for multi-agent systems. In Satoh, K. and Sadri, F., editors, *Proceedings of the Workshop on Computational Logic in Multi-Agent Systems (CLIMA-00)*, pages 56–65, Imperial College, London.

[Weld, 1994] Weld, D. S. (1994). An introduction to least commitment planning. *AI Magazine*, 15(4):27–61.

[Wilkins, 1992] Wilkins, D. (1992). *Using the SIPE planning system*. SRI Artificial Intelligence Center, Menlo Park, CA.

[Wilkins et al., 1994] Wilkins, D., Myers, K., Lowrance, J., and Wesley, L. (1994). Planning and reacting in uncertain and dynamic environment. *Journal of Experimental and Theoretical AI*, 6:197–227.

[Wooldridge and Parsons, 1999] Wooldridge, M. and Parsons, S. (1999). Intention reconsideration reconsidered. In Müller, J., Singh, M. P., and Rao, A. S., editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 63–80. Springer-Verlag.

[Xuang and Lesser, 1999] Xuang, P. and Lesser, V. R. (1999). Incorporating uncertainty in agent commitement. In Müller, J., Singh, M. P., and Rao, A. S., editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555. Springer-Verlag.

[Zilberstein, 1996] Zilberstein, S. (1996). Bounded sensing and planning in robotic systems. In *Planning with Incomplete Information for Robot Problems: Papers from the 1996 AAAI Spring Symposium*, page 141ff. AAAI Press, Menlo Park, California.