Norms and Time in Agent-Based Systems

Tiberiu Stratulat, Françoise Clérin-Debart, Patrice Enjalbert
GREYC, CNRS UPRESA 6072
Université de Caen, 14032 Caen, France
tistratu, debart, patrice@info.unicaen.fr

ABSTRACT

We propose a first-order model as a possible formal basis for normative agent systems (NAS). The model allows us to describe the execution of actions in time and the use of dynamic norms. We present its application to the detection of the violation cases and to optimal scheduling.

1. INTRODUCTION

In agent-based systems (MAS), the agents need to coordinate their activities in order to satisfy their goals. A possible solution to the coordination problem, that was proposed in the last years concerns the use of norms or social laws. A norm specifies what an agent should do or not, under certain conditions. Because of the many concepts used in the construction of the norms (obligations, social structure, authority, time, action, etc.) the problems related to their application in MAS are various and have partial solutions. For instance Castelfranchi [4] shows the importance and the psycho-cognitive aspect of the obligations in the organizations and social structures. In [22] the authors study the general utility of the social laws and in [21] Shoham proposes AOP, an agent oriented language where the obligations are treated locally at the agent level, ignoring their social aspect. Barbuceanu et al. [3] use the dynamic approach of the obligations introduced by Meyer [15] and propose a communication language for the coordination with obligations, showing that their use is rather a constraint programming problem. An informal description of NAS and of the elements necessary to their implementation is given in [24]. Recently Dignum et al. [7] proposed an extension of the BDI model [17] with normative elements.

In this article we present a first-order model as the formal basis for NAS. The model allows us to describe the execution of actions in time and the use of dynamic norms (norms with lifetime). The norms are constructed with deontic predicates (such as obligations, permissions or interdictions) that characterize the execution of an action by an agent in an interval of time.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICAIL-2001 St. Louis, Missouri USA
Copyright 2001 ACM 1-58113-368-5/01/0005 $5.00.

In MAS, the norms are used as means (i) to influence and (ii) to monitor the behavior of the agents. For the former it means that in order to assure a behavior which is compliant to the norms, we will inject the normative information in the planning and scheduling modules of each agent. While for the latter, based on the normative descriptions and the actual (past and present) behaviors of the agents, the system should detect the deviating behavior. Both cases ask for operational elements in the formal model. Therefore we propose an extension with model checking mechanisms that will be used in computing the violation cases and in optimal scheduling.

The rest of the article is organized as follows: in section 2 we present NAS and the motivations behind their use. In section 3 we describe the general model and in section 4 we give its syntax and semantics followed by the definitions of the main components. The operational aspect is presented in section 5, where we show its applications and give some examples. We conclude in section 6.

2. MOTIVATION

Autonomy is the most attractive feature of MAS. It allows an agent to freely perceive and act in an environment, without the explicit intervention of an external designer. This is a very important requirement, because in practice agents are used to designate heterogenous components, built by different persons, using different methods, and therefore very difficult to modify to be adapted to new situations. On the other hand, we need robust systems, where the key concept is that of control. For instance we control the program execution, the memory allocation, the access to resources, etc. Applied to an agent it means that we control what the agent perceives, and the way it acts. If for the former we can find some mechanisms to filter the information received by an agent, for the later we should propose a trade-off solution that combines autonomy and control.

A possible candidate to solve this problem is the model of Normative Agent Systems (NAS). In such a system, instead of modifying the internal construction of the agents to control their behaviors, we "influence" them. We simply tell to an agent what it should do or not. The information received by an agent has a double aspect: descriptive and normative. It is descriptive because it provides details about the content of a rule of behavior, a plan or a protocol. It is normative because it describes what is considered to be a normal (or standard) behavior, from the point of view (or w.r.t. the

goal) of an authority. The authority could be the system's designer, the system itself or a part of it (a group of agents or just an agent). The general term used to name this piece of information is that of *norm*. So, a norm is just a syntactic description of a specific normal behavior, which is sent to each agent involved in coordinated interaction. The way a norm is then interpreted (accepted or ignored) depends on the agent. From an utilitarian point of view, there are at least two reasons for which an agent follows a norm:

1. when the benefit obtained by adhering to norms, as a whole, is higher than that obtained when ignoring them
2. when the violation of a norm has a price to be paid.

Since the autonomy is expressed in terms of freedom of choice, we let the agents to ignore and violate the norms. Both scenarios require monitoring and reasoning capabilities inside the system that should be able to detect the non-correspondence between the behavior of the agents and the behavior prescribed by a norm. We call control element, the entity in the system which is responsible for the detection of the deviating behaviors and the computation of their effects. Note that the agents themselves could play such a role.

NAS architectures (see [24]) impose the use of a certain set of elements when constructing norms. In the following we give the main components of a norm:

- type: obligations or rights
- object: an action or a state (property)
- condition on object: an interval of time
- subject: a non-empty set of roles or agents
- authority
- author
- interval of validity
- context of application: a logical condition
- cost of violation
- exceptions to context of application
- exceptions to subject

The norms could be conceived either in a centralized manner by a single "person", or decentralized, directly by the agents themselves. For the former case the authority and the author could be the same entity, while for the latter case, when the norms are created by special utterances (speech acts), it is essential to differentiate between the author of the norm and the authority in the name of whom the norm has been issued.

Finally, it is important to note that there are at least two final users of the norms: ordinary agents that will inject the norms in their planning and scheduling modules, and control components. In this paper we will concentrate only on a subset of elements that compose a norm. We claim that the way we introduce them is motivated by their main use and impact to planning, scheduling, and monitoring. It is the same motivation that led us to propose a model where we used a quantitative representation of time in the definitions of the predicates.

3. INFORMAL DESCRIPTION

The model behind a normative agent system requires for an explicit representation of the world and the way this world changes. Following the classical trend of AI ([14], [1], [20]) we propose a model, where the main ingredients are fluent predicates that characterize the state of the world at a certain point in time, and event occurrences which capture the changes that take place. In this framework we use a linear model of time.

In general a fluent predicate of the form $p(t)$ represents the fact that p is true at t. In our model we will use an equivalent notation $holds(t, p)$. Besides the fluent predicates we can have other predicates that have time as argument, but with different temporal meanings and properties. We will refer to them as non-fluent predicates and we will see later what properties they have.

3.1 Events, event types, acts and actions

We introduce the notion of *event* as a means to classify how the world may change. Each event has a duration. It starts and ends at certain moments of time (that could be different or equal). In order to represent the occurrence of an event we use two non-fluent predicates $starts(e, t)$ and $finishes(e, t)$, showing that the event e started/finished at the moment of time t. Their use could be replaced by an equivalent definition $occurs(e, i)$ with i an interval of time. For instance, the fact that it rained in Normandy between 8:00 and 10:00 is written as:

$$occurs(Rain\_in\_Normandy, [8:00, 10:00])$$

Depending on the moment of observation (now), they have different uses in the definitions of other fluent predicates such as those used to describe the execution of an action or the violation of a norm (see section 4).

In general the events describe changes that have no explicit actors. When an event has an actor, we call it *act*. For instance if Mr. X opens the window at 10:00, the event is $occurs(Open\_door, [10:00, 10:00])$ and that Mr. X is its actor, $agent\_of(X, Open\_door)$. Or much shorter, in the same logical language:

$$does(X, Open\_door, [10:00, 10:00])$$

We also introduce the notion of class of events, or *event type*, as an abstraction of a certain collection of events. This is useful when the same occurrence of an event is interpreted as meaning different things (it is the same instance of more classes). For instance when someone signs a check, it could be interpreted as writing on a piece of paper, as an identification or as making a payment. Similarly to events, we introduce for acts the notion of *action* as being a class of acts or events with actors. We resume: an event is seen as the instance of an event type, and an act as the instance of an action.

Events and acts could be basic or complex. To describe complex actions we use the traditional action operators for

sequential ";" and parallel "||" execution and the non-deterministic choice "?".

3.2 Deontic status of an action

In this framework we use deontic concepts and notions such as agency, action and time. By deontic we mean what is obligatory, permitted or forbidden. It helps to describe ideal states or properties (*to-be* type) and ideal behaviors or actions (*to-do* type). Since an agent is by definition someone who acts, we are more interested in working with the to-do type.

The deontic status of an action describes what is considered as being an ideal behavior. This notion is represented in our model by the following predicates: $O(agent, \alpha, i)$, $P(agent, \alpha, i)$ and $F(agent, \alpha, i)$. They put in relation an *agent*, an *action* and an *interval* of time, meaning that the agent is *obliged/permitted/forbidden* to execute some acts of type α in the interval i. The presence of a time interval as argument of deontic predicates is motivated by the way the agents and the control elements interpret the norms. For instance, let's suppose that a norm obliges an agent to execute α without making explicit the moment of its execution. We also suppose that doing an action takes time. In this case, the agent has three alternatives: to immediately execute the action, to postpone the execution until it will be available (according to planning and scheduling processes) or to never execute it. By leaving unspecified the interval of execution it will introduce a temporal non-determinism. Therefore, the problem is with the control element which does not have enough elements to determine when the agent violated the norm.

Given the deontic status of an action we can construct a norm. In order to keep the presentation as much as simple, we will consider only some elements from the general set that composes a norm. Therefore, in this model the form of a norm is of the following conditional type:

$$OPI \leftarrow Condition$$

where OPI is one of the deontic predicates introduced above, and $Condition$ represents the context of norm's application, being any logical formula of the language the we will present below.

As we already said before, the use of norms implies the use of violations. We compute the violations starting from the description of an ideal behavior and the observations on the past and current execution of the system. We employ model checking techniques, considering a violation as being the non-execution of an obligatory action or the execution of a forbidden action.

Besides the interval a norm refers to, there is another temporal dimension that characterizes its life in time. Note that the deontic predicates are fluent, that is, their truth values change in time. By considering them homogeneous over an interval of time it allows us to introduce the concepts of life time for norms, and persistence for obligations. This is quite natural, because in the real world we have laws that are voted at a certain moment and then abrogated. The life time could be different from the time interval the law refers to. For instance, the obligation for Mr. X to pay the taxes in January, is written as:

$$O(X, Pay\_taxes, [01/01, 31/01])$$

However this is not enough. We should make it explicit for which period of time the obligation is valid. Let's suppose that there is a more general tax law from which we derive this obligation and which was voted last year and is still in force. Therefore the complete form to express this obligation is:

$$holds([01/01/2000, \rightarrow[), O(X, Pay\_taxes, [01/01, 31/01]))$$

If the law is abrogated or modified in 2001, the updated form is:

$$holds([01/01/2000, 01/01/2001], \\ O(X, Pay\_taxes, [01/01, 01/31]))$$

Another example that shows the importance of both manners to treat the time within norms is related to retroactive laws. In this case, the period of time a norm refers to could contain subintervals that are in the past of the moment of its creation. As a consequence, while looking to some current behavior, it is possible to have completely legal acts at the moment of their execution, and which generate violations when the new retroactive law is approved.

In the next section we present a first order language, followed by the definitions of the main concepts introduced above, showing that the time is a major component.

4. SYNTAX AND SEMANTICS

We start by briefly presenting the syntactical elements of a multi-sorted first-order language. We consider a typed-signature Σ (see [23], [28]) and the main sorts $Agent$, $Event$, $EventType$, $Time$ (instants represented by integers or rationals), $Interval$ (for time intervals) and $Boolean$. In our model we separate the use of events from that of acts (and similarly for actions and types of events as their classes) and introduce two subsorts $Act \subset Event$ and $Action \subset EventType$. Variables are denoted by strings starting with lower case letters, and constants by strings starting with upper case letters. Free variables are assumed to be universally quantified. We use typed predicates, each with fixed arity.

We consider the usual definitions for terms and well-formed formulas. The semantics of the language is based on a Σ-signature with classical definitions for variable assignment, satisfaction and validity of a formula.

In the following we introduce the main predicates and functions used in this framework:

4.1 Time points and intervals

- time point predicates and functions: $=, <, \leq, +$.

- interval predicates, of type $Interval \times Interval \rightarrow Boolean$. Their use and definition are those developed by Allen in [1], [2]:

$starts(i,j)$	
$finishes(i,j)$	
$meets(i,j)$	$i : j$
$before(i,j)$	$i \prec j$
$during(i,j)$	$i \sqsubset j$
$before(i,j) \vee meets(i,j)$	$i \prec: j$
$during(i,j) \vee i = j$	$i \sqsubseteq j$
$disjoint(i,j)$	$i \bowtie j \equiv i \prec: j \vee j \prec: i$

- mixed point-interval predicates and functions: $[t_1, t_2]$ the interval function, it generates an interval given two instants; $in(t,i)$ is true if t is inside i, which formally is $in(t,i) \equiv during([t,t],i)$; $\varnothing$ the empty interval; $\cap$ - interval intersection function; $min(i)$ and $max(i)$, functions that compute the minimum and the maximum value of an interval.
- the open-ended intervals $]\leftarrow, t] = \{x | x \leq t\}$, $[t, \rightarrow[= \{x | x \geq t\}$, and $]\leftarrow, \rightarrow[=]\leftarrow, t] \cup [t, \rightarrow[$.

As shown above, in this model we use the so-called temporal predicates, which are predicates with one or more temporal arguments. Two examples of temporal predicates are the occurrences of events and the fluent predicates. However their behavior over an interval is different. In order to capture this difference we will use the notion of homogeneity. A predicate is *homogeneous* if when it holds over an interval i, it also holds over any subinterval. For instance a fluent predicate is homogeneous, and an event is non-homogeneous. In order to avoid the explicit introduction of the homogeneity axiom for each homogeneous predicate $p(i,...)$ we will use the following equivalent form: $holds(i, p(...))$. As an example, let's consider that the library was open between 9:00 and 12:00. Since in any subinterval of $[9:00, 12:00]$ the library was open, we will write:

$$holds([9:00, 12:00], open(Library))$$

The definition of *holds* in a time-point is given by:

$$holds(t,p) \equiv holds([t,t],p)$$

In comparison to homogeneous predicates, when we want to say that a predicate defines a relation that is true only w.r.t. a specific interval of time we will consider just simple temporal predicates. For instance, the fact that the (displayed) time-schedule of the library is from 9:00 to 12:00 has this property. This is written as:

$$schedule(Library, [9:00, 12:00])$$

Both ways of representing different properties over intervals of time could coexist for temporal predicates with many temporal arguments. For instance, the fact that last summer the schedule of the library was from 9:00 to 12:00 is written as:

$$holds([06/01/2000, 08/31/2000], schedule(Library, [9:00, 12:00]))$$

where the predicate *schedule* is homogeneous over the first interval and non-homogeneous over the second.

Another syntactic sugar is the use of relative intervals. In general an interval of time is constructed by applying the function $[,]$ to two absolute time points. Time is considered to be absolute w.r.t. a time origin t_0. Therefore, given a third time point t we can construct a relative interval of time by using the construction $[t+t_1, t+t_2]$. If we want to use relative time, we should introduce for each predicate of the form $p(...,t)$ or $p(..,i)$, relative predicates defined as:

$$\forall t, t', ... \; P_R(t, ..., t') \equiv P(..., t+t') \text{ and}$$
$$\forall t, t_1, t_2, ... \; P_R(t, ..., [t_1, t_2]) \equiv P(..., [t+t_1, t+t_2])$$

All predicates will use absolute time in what follows except when explicitly noted.

4.2 Events

In this model the events could be basic or complex. In order to make this distinction among events we will provide the model with an algebra of events with operations for sequential ";", parallel "||" and non-deterministic choice "?" compositions. They are overloaded to work on both $Event$ and $EventType$ objects.

- *starts(e, t)* and *finishes(e,t)*, predicates of type $Event \times Time \rightarrow Boolean$. Each event has unique moments of time to start and to finish. This is described by:

$$\forall e, t, t' \neq t \; starts(e,t) \Rightarrow \neg starts(e,t')$$
$$\forall e, t, t' \neq t \; finishes(e,t) \Rightarrow \neg finishes(e,t')$$

- $occurs(e,i)$, a predicate of type $Event \times Interval \rightarrow Boolean$ meaning that an event e occurs at the interval i. This is the main predicate with which we build the most part of the predicates used in this framework. Its definition depends on the type of event, if it is basic or complex.
  1. for basic events we have that:
     $occurs(e, [t_1, t_2]) \equiv$
     $t_1 \leq t_2 \wedge starts(e, t_1) \wedge finishes(e, t_2)$
  2. for composite events we have:
     $occurs(e_1? \, e_2, i) \equiv$
     $occurs(e_1, i) \vee occurs(e_2, i)$
     $occurs(e_1 || \, e_2, i) \equiv \exists i' \; i' \sqsubseteq i \wedge$
     $(occurs(e_1, i) \wedge occurs(e_2, i') \vee$
     $occurs(e_1, i') \wedge occurs(e_2, i))$
     $occurs(e_1; e_2, i) \equiv \exists i_1, i_2 \; i_1 \prec: i_2 \wedge$
     $starts(i_1, i) \wedge finishes(i_2, i) \wedge$
     $occurs(e_1, i_1) \wedge occurs(e_2, i_2)$
- $occurs(e)$ is a version of $occurs(e,i)$ without giving the interval:

$$occurs(e) \equiv \exists i \; occurs(e,i)$$

- in this model we consider a linear time, which allows the use of the past relatively to an instant. In order to describe the past occurrence of an event, we introduce new predicates, which are fluent versions of *occurs*. We start with $occurred(e)$ which is true at t if e occurred in the past of t.

$$holds(t, occurred(e)) \equiv \exists i\, occurs(e, i) \wedge i \prec: [t, \infty)$$

- $occurred(e, i)$ is the version that indicates if e occurred in the interval i, where i is either its interval of occurrence or a larger interval that includes it. The two versions are:

  1. $holds(t, occurred^*(e, i)) \equiv occurs(e, i) \wedge i \prec: [t, \infty)$
  2. $holds(t, occurred(e, i)) \equiv \exists i'\, i' \sqsubseteq i \wedge occurs(e, i') \wedge i' \prec: [t, \infty)$

  In the definitions that follow, we will consider only the second version.

- $occurring(e)$ is true at t if e is currently occurring relative to t.

$$holds(t, occurring(e)) \equiv \exists i\, in(t, i) \wedge occurs(e, i)$$

- $subclass(\varepsilon, \varepsilon')$ a non-fluent predicate of type $EventType \times EventType \rightarrow Boolean$. Example: $subclass($ *Sign*, *Write*$)$.

- $instance\_of(e, \varepsilon)$ a non-fluent predicate that links event tokens to their classes. For instance the fact that the signature of the check 4526 is a payment is written as: $instance\_of(sign\_check(4526), Pay)$. Because we work with complex events and event types, we need axioms for this predicate that allows for composition. If "$\circ$" denotes one of the symbols for composition functions "?", ";" or "||", then the axioms are:

$$instance\_of(e_1 \circ e_2, \varepsilon_1 \circ \varepsilon_2) \equiv instance\_of(e_1, \varepsilon_1) \wedge instance\_of(e_2, \varepsilon_2)$$

$$instance\_of(e, \varepsilon_1) \wedge subclass(\varepsilon_1, \varepsilon_2) \Rightarrow instance\_of(e, \varepsilon_2)$$

4.3 Acts

Since we consider an act as an event produced by an actor, we give similar definitions for predicates on acts as we did for events:

- $does(agent, act, i)$, a predicate of type $Agent \times Act \times Interval \rightarrow Boolean$. It shows that *agent* does *act* over interval i. It is the corresponding predicate to $occurs(e, i)$ for events. Its definition is given by:

$$does(agent, act, i) \equiv occurs(act, i) \wedge agent\_of(agent, act)$$

- where $agent\_of(agent, act)$ is a predicate of type $Agent \times Act \rightarrow Boolean$, showing who is responsible for doing *act*. In the case of a composite act we have:

$$agent\_of(agent, act_1 \circ act_2) \equiv agent\_of(agent, act_1) \wedge agent\_of(agent, act_2)$$

- in a similar manner we consider the definitions for the rest of the corresponding fluent act predicates, respectively: $done($*agent, act*$)$, $done($*agent, act, i*$)$ and $doing($*agent, act*$)$.
  We introduce another fluent predicate $failed($*agent, act, i*$)$ to represent the fact that the agent failed to do *act* in i:

$$holds(t, failed(agent, act, i)) \equiv holds(t, \neg done(agent, act, i))$$

  The introduction of $failed$ predicate is motivated by the fact that we don't permit the use of the action negation, which is in general problematic. The solution is to adopt the "closed world assumption" and consider $\neg$ as the negation by failure [5] in logic programming. The failure of doing an act in an interval is given by the failure to derive if the act has been done in that interval.

4.4 Deontic properties

In this framework we treat at the same level the deontic concepts and the notions of agency, action and time. The deontic status of an action is captured by fluent deontic predicates, such as $O(agent, \alpha, i)$ where i is a relative or absolute interval of time. This is different from the classical trend of deontic logic, where the deontic operators are modal logical operators of the type Op with p a logical formula (see [16] for an introductory presentation).

In addition, we consider for the sake of simplicity only the case of unanalyzed deontic predicates, leaving for future work the provision with additional semantics or axioms that describe and allow the reasoning about the relationships between them (i.e. what relationship exists between obligation and interdiction, or how to check the consistency of the norms). They are important tools for the creator of the norm. However, at the agent level, we suppose that the only thing we need is an event-based mechanism that trigers the generation of a deontic state from normative descriptions.

It is important to note that the deontic predicates apply only to actions, and not to acts. An action is an abstract concept that is used in our model as a class. The acts represent real and concrete occurrences and hence they are viewed as instances of actions. For example, when someone signs the check 4526, what it is obligatory is not the act of effectively signing that check, but the class to which it belongs: in the example, the Pay action.

4.5 Violations

When we compute the violations we take into account the normative descriptions and the current and past execution of the system. A possible definition for the violation case is captured by the fluent predicate $V(agent, \alpha, i)$. It means that the agent *violated* at the moment t a norm w.r.t. the execution of an act of action type α over the interval i. Its definition is given by the next axiom:

$$holds(t, V(agent, \alpha, [t_1, t_2])) \equiv holds(t, O(agent, \alpha, [t_1, t_2])) \wedge [t_1, t_2] \prec [t, \infty) \wedge \forall act\, (instance\_of(act, \alpha) \Rightarrow holds(t, failed(agent, act, [t_1, t_2])))\vee$$

$holds(t, I(agent, \alpha, [t_1, t_2])) \wedge [t_1, t_2] \prec [t, \infty) \wedge$
$\exists act, instance\_of(act, \alpha) \wedge$
$(holds(t, done(agent, act, [t_1, t_2])) \vee$
$holds(t_1, doing(agent, act)) \vee$
$holds(t_2, doing(agent, act))) \vee$

$holds(t, I(agent, \alpha, [t_1, t_2])) \wedge in(t, [t_1, t_2]) \wedge$
$\exists act, instance\_of(act, \alpha) \wedge$
$(holds(t, done(agent, act, [t_1, t])) \vee$
$holds(t_1, doing(agent, act)) \vee$
$holds(t, doing(agent, act)))$

5. APPLICATIONS

The model proposed in this framework was implemented in Prolog Eclipse [8] and has been used to build two applications: the prototype for a control element and the scheduling module of an agent. Each predicate of the model has a direct corresponding Prolog predicate. We used for the definitions of the temporal predicates (fluents, execution of acts, violations, etc.) the library for finite domains of Eclipse, that provides a very powerful mechanism for constraint propagation.

The introduction of the $holds/2$ predicate was essential to the implementation of our model as a logic program. For questions of the type $holds(i, p)$, where the temporal argument is a variable, the constraint propagation module collects all the temporal constraints and tries to find a solution by instantiating i to a domain.

Example For the detection of the violations we are interested to have questions of the kind $holds(i, V(agent, \alpha, interval))$. Let's reconsider the example with Mr. X and his obligation to pay the taxes in January. In addition we have the fact that he paid the taxes on January 14th, by signing the check with the number 4526. All the information is described in our model by the following propositions:

$holds([01/01/2000, \rightarrow[,$
$O(X, Pay\_taxes, [01/01/01, 31/01/01]))$
$occurs(sign\_check(4526), [01/14/01, 01/14/01])$
$agent\_of(X, sign\_check(4526))$
$instance\_of(sign\_check(4526), Pay\_taxes)$

From these definitions and by using the operational aspect of the implementation of the $holds/2$ predicate we infer the following propositions:

$holds([14/01/01, \rightarrow[, occurred(sign\_check(4526)))$
$holds([14/01/01, \rightarrow[,$
$occurred(sign\_check(4526), [14/01/01, 14/01/01]))$
$holds([14/01/01, \rightarrow[, done(X, sign\_check(4526)))$
$holds([14/01/01, \rightarrow[,$
$done(X, sign\_check(4526), [14/01/01, 14/01/01]))$

Note that there is no violation detected. If we add the interdiction to make payments on weekends:

$holds(]\leftarrow, \rightarrow[, I(X, Pay, [13/01/01, 14/01/01]))$
$subclass(Pay\_taxes, Pay)$

we get the violation:

$holds([14/01/01, \rightarrow[, V(X, Pay, [13/01/01, 14/01/01]))$

Scheduling A second application of the model concerns the specification of a scheduling problem with temporal and deontic constraints. Let's suppose that an agent is under the influence of a set of norms, each norm having a cost to pay if violated. The agent should schedule its future acts so that it should pay the minimum cost. It is the role of the agent's scheduling module to propose the list of the future acts and their times of execution that satisfy the following constraints:

1. temporal constraints:
   (a) each act has a period of execution.
   (b) each act respects the execution order.
2. deontic constraints: the obligation and the interdiction to execute an act in an interval. Each deontic constraint specifies the cost to be paid if violated.

Starting from the Prolog implementation of the model, it was easy to implement an algorithm that generates for each obligation and interdiction on an action, a planned instance of that action, and which collects all the constraints that are generated afterwards (see [25] for a version with more details on scheduling). We used the the same library for finite domains and the predicate `minimize/2`. This predicate is implemented using the *branch and bound* method and offers an optimal solution.

6. CONCLUSIONS AND RELATED WORK

In this paper we proposed a first-order model for normative agent systems, starting from the idea that norms are means for influencing and controlling the agent's behavior. Inspired by the role they play in human societies and in legal domains, the use of norms in agent-based systems remains an open debate. However, we adopted the perspective of an explicit use and representation of norms. In our case the norms are dynamic in the sense that they have a lifetime (appear and disappear in time) and prescribe the execution of an action over a time-interval. We showed how to describe the execution of an act and how to compute the violation cases.

Another essential point in this framework is the distinction between acts (as concrete occurrences) and actions (as classes of acts). Since a norm describes an ideal behavior, we use this distinction to show that a norm characterizes a class of acts and not an act.

In order to show the applicability of this model we briefly described a logic programming implementation in Eclipse Prolog and we presented two types of applications: the detection of the violation cases and the prototype of a scheduler with deontic and temporal constraints.

By proposing a model in a first-order logic has some advantages. First, we benefit from the results of a long tradition in representing and reasoning about actions, events and time [2, 1, 14]. Second, there are some previous works that attempted to formalize the normative concepts in terms

of first-order predicate calculus (see below), the first seeming to be von Wright [27]. Third, the formal aspects of such frameworks could be easily translated into Prolog implementations. The existing tools for constraint satisfaction proposed by different Prolog systems [8] are nice and useful results.

As examples of works that have been proposed to represent normative concepts in first-order formalisms, we mention Sergot who described in [19] an application of logic programming for representing legal rules. We also note in his work the presence of predicates with temporal arguments. In a more recent paper [13] Lokhorst gives a complex account of reasoning about actions and obligations which generalizes the logic programming implementation of Ronald M. Lee's deontic expert system DX. See [18] for a presentation of defeasible deontic reasoning used in DX. Finally, a temporal treatment of legal norms in a first-order formalism is proposed by Hernández Marín and Sartor [10]. They present a model based on event-calculus [12] and treat various temporal aspects of legal rules, showing that there is a legitimate separation between the validity of a norm and its internal time.

Although we didn't follow the modal tradition to describe normative concepts, we acknowledge that there are other interesting approaches that belong to this stream of research. Many of them propose semantics that interpret the deontic concepts in terms of temporal ones. For instance in [9] the notion of obligation is considered as a sort of liveness condition: something will happen in the future. This view is arguable, since an obligation refers to something that should happen. Horty and Belnap [11] define the obligations in a branching time framework. In order to represent the obligations on actions, they use the *stit* operator and the possibility to choose among future courses of actions. Van Eck's approach [26] is based on the definition of temporal necessity and in [6] Dignum and Kuiper propose a treatment of the obligations with deadlines, based on Meyer's reduction of deontic logic to dynamic logic [15].

7. ACKNOWLEDGEMENTS

We would like to thank Gérard Becher for his interest in this work and the anonymous referees who carefully read and made helpful comments on the initial version of this paper.

8. REFERENCES

[1] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[2] J. F. Allen and G. Ferguson. Actions and events in interval temporal logic. Technical report, Computer Science Department, University of Rochester, 1994.

[3] M. Barbuceanu, T. Gray, and S. Mankovski. Coordinating with obligations. In *Proceedings of Autonomous Agents'98*, Minneapolis, MI, 1998.

[4] C. Castelfranchi. Commitements: From individual intentions to groups and organization. In *Proceedings of ICMAS'95*. AAAI Press, 1995.

[5] K. Clark. *Negation as Failure*. Logic and Databases. Plenum Press, New York, 1978.

[6] F. Dignum and R. Kuiper. Specifying deadlines with dense time using deontic and temporal logic. *International Journal of Electronic Commerce*, 3(2):67–86, Winter 1998-99.

[7] F. Dignum, D. Morley, L. Sonenberg, and L. Cavedon. Towards socially sophisticated BDI agents. In *Proceedings of ICMAS'2000*, Boston, USA, 2000.

[8] `http://www-icparc.doc.ic.ac.uk/eclipse/`.

[9] J. L. Fiadeiro and T. S. E. Maibaum. Temporal reasoning over deontic specifications. *Journal of Logic and Computation*, 1(3):357–395, 1991.

[10] R. Hernández Marín and G. Sartor. Time and norms: a formalisation in the event-calculus. In *Proceedings of ICAIL-99*, 1999.

[11] J. Horty and N. Belnap. The deliberative stit: A study of action, omission, ability, and obligation. *Journal of Philosophical Logic*, 24:583–644, 1995.

[12] R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[13] G.-J. C. Lokhorst. Reasoning about actions and obligations in first-order logic. *Studia Logica*, 57:221–237, 1996.

[14] J. M. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intellingence. *Machine Intelligence*, 4:463–502, 1969.

[15] J. J. C. Meyer. A different approach to deontic logic: Deontic logic viewed as a variant of dynamic logic. *Notre Dame Journal of Formal Logic*, 29:109–136, 1988.

[16] J. J. C. Meyer and R. J. Wieringa. Deontic logic: A concise overview. In J. J. C. Meyer and R. J. Wieringa, editors, *Deontic Logic in Computer Science: Normative System Specification*. John Wiley & Sons, 1993.

[17] A. S. Rao and M. P. Georgeff. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems, ICMAS'95*, San Francisco, 1995.

[18] Y. U. Ryu and R. M. Lee. Defeasible deontic reasoning: A logic programming model. In J. J. C. Meyer and R. J. Wieringa, editors, *Deontic Logic in Computer Science: Normative System Specification*. John Wiley & Sons, 1993.

[19] M. Sergot. Prospects for representing the law as logic programs. In K. L. Clark and S. A. Tärnlund, editors, *Logic Programming*. Academic Press, 1982.

[20] Y. Shoham. *Reasoning about change*. MIT Press, 1988.

[21] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

[22] Y. Shoham and M. Tennenholtz. On social laws for artificial agent societies: off-line design. *Artificial Intelligence*, 73:231–252, 1995.

[23] G. Smolka and H. Ait-Kaci. Inheritance hierarchies: Semantics and unification. *Journal of Symbolic Computation*, 7(3/4):343–370, 1989.

[24] T. Stratulat. Normative agent systems. In *Proceedings of POLICY'99*, Bristol, UK, 1999.

[25] T. Stratulat, F. Clérin-Debart, and P. Enjalbert. Temporal reasoning: An application to normative systems. Submitted to publication.

[26] J. van Eck. A system of temporally relative modal and deontic predicate logic and its philosophical applications. *Logique et Analyse*, 99,100, 1982.

[27] G. H. von Wright. Deontic logic. *Mind*, 60:1–15, 1951.

[28] C. Walther. Many-sorted unification. *JACM*, 35(1):1–17, 1988.