Temporal Reasoning: An Application to Normative Systems

Tiberiu Stratulat, Françoise Clérin-Debart, and Patrice Enjalbert
GREYC, CNRS UPRESA 6072
Université de Caen, 14032 Caen, France
{tistratu, clerin, 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 also present its operational aspect which is based on the reduction of the deontic constraints to temporal ones. In order to show the applicability of the model we briefly describe a logic programming implementation and we present two types of applications: the detection of the violation cases and the prototype of a scheduler with deontic and temporal constraints.

1 Introduction

In the area of MAS the problem of coordinating activities among agents is a very important one. A solution to this problem, that has been proposed in the last years concerns the use of norms or social laws ([21], [4], [3], [7]). A norm is mainly a syntactical description of an ideal behavior saying what an agent should do or not. The main users of the norms are the ordinary agents which try to have a norm-compliant behavior, and the control elements of the system which check if the agents obey the norms. Therefore a norm is just a piece of information sent to each agent involved in coordinated interaction and which is interpreted (accepted or ignored) at the agent level based on some decision-making internal mechanism.

When constructing norms the architecture of a normative system requires the use of many concepts such as social structure, authority, roles, obligations, action, time, etc. In this paper we study only a subset of them, and in particular the relationship between the deontic[1] characterization and the temporal execution of actions. We show that the use of norms could be reduced to the use of temporal information. This method only applies to a certain class of normative problems, notably those that describe ideal behaviors in time and where actions have a duration of execution. As shown before, the normative description is mainly used in two situations: to constrain a behavior and to verify if the actual behavior is conform to its ideal description. For the former case, by translating deontic constraints into temporal ones we gain when scheduling future courses of actions from the use of the constraint solvers that already exist for temporal models. For the latter case, we simply need some methods to verify if the interval of the act execution is consistent to the interval prescribed by the norm.

In order to clearly show how to manipulate the temporal information in such systems, we propose in the next section a logical model that helps to describe the execution of actions in time. We give ontological definitions for the various concepts used in the model such as acts, events, actions, event types, obligations and violations, and we present their properties. The main applications of the model, the deontic scheduling and the detection of violations are described in section 3. We present some related works in section 4 and give the conclusions in section 5.

2 Formal Model

The model behind a normative agent system requires for an explicit representation of the world and the way this world changes. We need a model that allows us to represent the execution of actions in time and which is able to treat at the same level the different notions of agency, time, events and obligations. Therefore, following the classical trend of AI ([15, 1, 20]) we propose a model, where the main ingredients are fluent predicates $p(t)$ that characterize the state of the world at a certain point in time, and event occurrences $occurs(e, i)$ which capture the changes that take place. We use a linear model of time.

We start by briefly presenting the syntactical elements of a many-sorted first-order language, provided a typed-signature Σ and the main sorts $Agent$, $Event$, $EventType$, $Time$ (instants represented by integers or rationals), $Interval$ (for time intervals) and $Boolean$. In this model we separate the use of events from that of acts (and

[1] by deontic we mean what is obligatory, permitted or forbidden

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.

2.1 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]:

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

- 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 defined as $in(t,i) \equiv [t,t] \sqsubset i$; $\varnothing$ the empty interval; $\cap$ the 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 $]\leftarrow, \rightarrow[=]\leftarrow, t] \cup [t, \rightarrow[$.

2.2 Temporal predicates

In this model we use the so-called temporal predicates, which are predicates with one or more temporal arguments (i.e. fluent predicates and occurrences of events). Their properties over an interval, however could be different. In order to show this difference we use the notion of homogeneity. A predicate is *homogeneous* if and only if when it holds over an interval i, it also holds over any of its subintervals. 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 fluent predicate $p(i, ...)$ we 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.[2] Since in any subinterval of $[9:00, 12:00]$ the library was open, we 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 contrast to homogeneous predicates, non-homogeneous temporal predicates describe a relation only w.r.t. a specific interval. For instance, the displayed time-schedule of a library, which opens from 9:00 to 12:00 has such property. Therefore, in the case of a fluent predicate it is possible to have many temporal arguments. The fact that last year the schedule of the library was from 9:00 to 12:00 is written as:

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

where the predicate $schedule/3$ is fluent and homogeneous over the first interval, but non-homogeneous over the second.

2.3 Events

We introduce the notion of *event* as a means to classify how the world may change. Each event has a duration and it starts and ends at unique moments of time (different or equal). In order to represent the occurrence of an event we use either the non-fluent predicates $starts(e,t)$ and $finishes(e,t)$ of type $Event \times Time \rightarrow Boolean$ and which have the properties:

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

or $occurs(e,i)$, an equivalent predicate. *Example:* there is a meeting between 8am and 10am: $occurs(Meeting, [8:00, 10:00])$.

The predicate $occurs(e,i)$, 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. In order to make this distinction among events we 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.

1. for basic events we have that:

[2] Note that in order to have more expressive examples we introduce user-defined intervals such as hours and dates.

$$occurs(e,[t_1,t_2]) \equiv t_1 \leq t_2 \wedge starts(e,t_1) \wedge finishes(e,t_2)$$

2. for composite events we have:

$$occurs(e_1?\, e_2,i) \equiv occurs(e_1,i) \vee occurs(e_2,i)$$

$$occurs(e_1 || e_2,i) \equiv \exists i'\, i' \sqsubseteq i \wedge (occurs(e_1,i) \wedge occurs(e_2,i') \vee occurs(e_1,i') \wedge occurs(e_2,i))$$

$$occurs(e_1;e_2,i) \equiv \exists i_1,i_2\, i_1 \prec: i_2 \wedge starts(i_1,i) \wedge finishes(i_2,i) \wedge occurs(e_1,i_1) \wedge occurs(e_2,i_2)$$

We continue with $occurs(e)$, a variant of $occurs(e,i)$ without mentioning the time interval:

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

In this model we consider a linear time, which permits 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*:

- $occurred(e)$ 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:

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

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

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

- $occurring(e)$ is a property which is true only when an event is occurring.

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

  Some might argue that the meaning of this predicate is different from Allen's one proposed in [1] and used in formalizing *processes*. Since we have not introduced an explicit separation between events and processes, our approach is much closer to the formalization and to the motivations behind the use of the "process-like" predicates (i.e. $CupFilling$) proposed in Allen and Ferguson's work [2].

2.4 Event types

We also introduce the notion of class of events, or *event type*, as an abstraction of a certain collection of events. The classes could have a hierarchical structure, for which we introduce $subclass(\varepsilon,\varepsilon')$ a non-fluent predicate of type *EventType* $\times$ *EventType* $\rightarrow$ *Boolean*. We also define $instance\_of(e,\varepsilon)$ that links event tokens to their classes. It is useful when the same occurrence of an event is interpreted as meaning different things (the same instance of more classes). For instance when someone signs on the check 4526, it could be interpreted as writing on a piece of paper, as an identification or as making a payment: $instance\_of(sign\_check4526,Pay)$.

Because we work with complex events and event types, we need axioms for this predicate that allows the composition. If "$\circ$" denotes one of the composition operators "?", ";" or "||", then the axioms are:

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

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

Similarly to events, we introduce for acts the notion of *action* as being a class of acts.

2.5 Acts

When an event has an actor, we call it *act*. 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. We start with $does(agent,act,i)$, which shows that $agent$ does act over interval i. It is the corresponding predicate to $occurs(e,i)$ for events:

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

The predicate $agent\_of(agent,act)$ describes who is responsible for doing act. In the case of a composite act we have:

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

Example The event of signing the check 4526, produced by Mr. X at 10 o'clock is written as:

$$does(X,sign\_check4526),[10:00,10:00])$$

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 in general is problematic. The solution is to adopt the "closed world assumption" and consider $\neg$ as the negation by failure [5] used 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.

2.6 Deontic properties

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 a norm is of the following conditional type:

$$OPI \leftarrow Condition$$

where OPI is one of the deontic predicates: $O(agent, \alpha, i)$, $P(agent, \alpha, i)$ or $I(agent, \alpha, i)$. These predicates show that an agent is *obliged/permitted/-forbidden* to execute some acts of type α in the interval i. In general the deontic notions help to describe ideal states (*to-be* type) or ideal behaviors (*to-do* type). Since an agent is by definition someone who acts, we are more interested in working with the to-do type. This is different from the main trend in the area of deontic logic, where the deontic operators are modal logical operators of type Op with p a logical formula (see [17] for an introductory presentation).

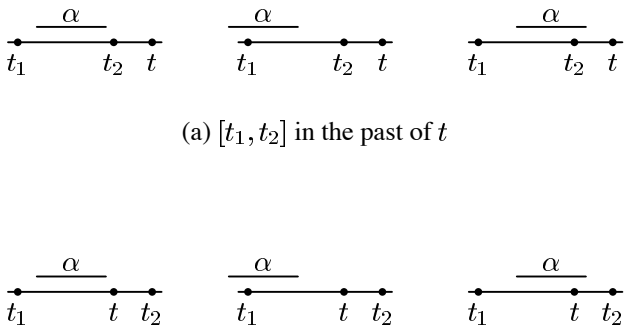
Note that the deontic predicates are fluent, that is, their truth values change in time. By considering them fluent, and therefore homogeneous, it allows us to introduce the concepts of lifetime for norms, and persistence for obligations. This is quite natural, because in the real world we have laws that are voted at a certain moment and then abrogated. Their lifetime could be different from the time interval they refer to. For instance, the obligation for Mr. X to pay the taxes in January, is written as $O(X, Pay\_taxes, [01/01, 31/01])$. If this obligation is derived from a more general tax law voted last year and still in force, the complete representation 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 becomes:

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

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.



(a) $[t_1, t_2]$ in the past of t

(b) t inside $[t_1, t_2]$

Figure 1. Violation cases for $I(agent, \alpha, [t_1, t_2])$ at t

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 are viewed as instances of actions. For example, when someone signs the check 4526, what is obligatory is not the act of effectively signing that check, but the class to which it belongs, in our case, the action Pay.

2.7 Violations

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

1. the norm is an obligation $O(agent, \alpha, i)$ with i in the past of t and there is no act of type α executed in i.
2. the norm is an interdiction $I(agent, \alpha, i)$ with i in the past of t and there is an act of type α executed into an interval that intersects i (see figure 1a).
3. the norm is an interdiction $I(agent, \alpha, i)$ with t contained in i and there is an act of type α in execution or executed into an interval started by i and finished by t (see figure 1b).

Formally, this is captured by the next axiom:

$holds(t, V(agent, \alpha, [t_1, t_2])) \equiv$

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

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

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

If we isolate the temporal relations in the above definition, the violation case could be defined only in terms of the intersection between the interval of the act and the interval given in the norm. There is a violation if this intersection is empty for obligations and non-empty for interdictions. We will use this equivalent definition when scheduling with norms (see next section).

3 Applications

The model proposed in this framework has been implemented in Prolog Eclipse [8] and has been used to build two applications: the prototype of 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 is essential to the implementation of our model as a logic program. We use the techniques of meta-programming to answer to questions of the type $holds(i, p)$, where the temporal argument is a variable. The constraint propagation module collects the temporal constraints and tries to find a solution which makes p true by instantiating i to a domain (if possible).

Violation checking 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. This scenario is described by the following propositions:

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

$occurs(sign\_check4526, [01/14/01, 01/14/01])$

$agent\_of(X, sign\_check4526)$

$instance\_of(sign\_check4526, Pay\_taxes)$

From the above description and by using the operational aspect of the implementation of $holds/2$, we infer the following propositions:

$holds([14/01/01, \rightarrow[, occurred(sign\_check4526))$

$holds([14/01/01, \rightarrow[,$
$occurred(sign\_check4526, [14/01/01, 14/01/01]))$

$holds([14/01/01, \rightarrow[, done(X, sign\_check4526))$

$holds([14/01/01, \rightarrow[,$
$done(X, sign\_check4526, [14/01/01, 14/01/01]))$

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 into a frame of time $\mathcal{F}$ 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 $\mathcal{L} = \{(\alpha_k, [s_{\alpha_k}, f_{\alpha_k}])\}_k$ given the estimated duration for each action $\mathcal{D} = \{duration(\alpha_k, d_k)\}_k$ and the deontic constraints that hold at the moment of scheduling:

$$\mathcal{O}_{\mathcal{F},t} = \{O(\alpha, i, cost) | holds(t, O(\alpha, i, cost) \wedge i \sqsubset \mathcal{F}\}$$
$$\mathcal{I}_{\mathcal{F},t} = \{I(\alpha, i, cost) | holds(t, I(\alpha, i, cost) \wedge i \cap \mathcal{F} \neq \varnothing\}$$

Temporal reasoning is a very good field of application for constraint reasoning (TCSP). In this type of application we separate the temporal information from the rest of the problem and treat it separately as a constraint domain for which we can use existing specialized reasoning tools. Therefore we reconsider the deontic constraints as temporal constraints and solve the scheduling problem as a TCSP problem. Figure 2 presents an algorithm for scheduling with deontic constraints. The algorithm generates a (possible null) planned instance for each obligatory action α_k, and then collects all the temporal constraints

procedure $schedule$
  input: $\mathcal{O}_{\mathcal{F},t}, \mathcal{I}_{\mathcal{F},t}, \mathcal{D}$
  output: $\mathcal{L}, Cost, \mathcal{C}$
  $\mathcal{L} = \{\}; Cost =_{\mathcal{C}} 0;$
  for $O(\alpha, i, cost) \in \mathcal{O}_{\mathcal{F},t}$
    $generate\_act((\alpha, i, cost), s_\alpha, f_\alpha, cost_\alpha);$
    $\mathcal{L} = \mathcal{L} \cup \{(\alpha, [s_\alpha, f_\alpha])\};$
    $Cost =_{\mathcal{C}} Cost + cost_\alpha;$
end

procedure $generate\_act$
  input: $(\alpha, i, cost)$
  output: $s_\alpha, f_\alpha, cost_\alpha$
  $generated_\alpha =_{\mathcal{C}} \{0, 1\};$
  if $generated_\alpha$
    $cost_\alpha =_{\mathcal{C}} 0;$
    $[s_\alpha, f_\alpha] \sqsubseteq_{\mathcal{C}} i;$
    $d_\alpha = duration(\alpha);$
    $f_\alpha =_{\mathcal{C}} s_\alpha + d_\alpha;$
    $\mathcal{I}_\alpha = \{(i_k, cost_k) | I(\alpha, i_k, cost_k) \in \mathcal{I}_{\mathcal{F},t}\};$
    for $(i_k, cost_k) \in \mathcal{I}_\alpha$
      if $i_k \cap [s_\alpha, f_\alpha] \neq \varnothing$
        /* interdiction violated */
        $cost_\alpha =_{\mathcal{C}} cost_\alpha + cost_k;$
  else
    /* obligation violated */
    $cost_\alpha =_{\mathcal{C}} cost;$
    $s_\alpha =_{\mathcal{C}} null; f_\alpha =_{\mathcal{C}} null;$
end

Figure 2. Algorithm for deontic scheduling

$\mathcal{C}$ that are hence created. The generation of each act and the propagation of constraints are made by using the *backtracking* method. We indexed the operations on time intervals with $\mathcal{C}$ to show that they are added to the set of constraints only if they preserve the overall consistency, otherwise they generate a backtracking step. The algorithm computes the list of the scheduled actions and the global cost for that solution. Since it collects only temporal constraints, the algorithm could be extended with other types of temporal restrictions (i.e. the execution order between actions). Starting from the Prolog implementation of the model, it was easy to implement this algorithm notably because Prolog already provides backtracking for solving the predicates. We used the same library for finite domains to implement the constraint propagation mechanism for $\mathcal{C}$. To obtain the solution with the minimum cost we used the predicate `minimize(+predicate,cost)`. This predicate is implemented using the *branch and bound* method. It tries to instantiate the variables of `predicate` and offers the solution that optimizes `cost`.

4 Related Work

The formalization of normative concepts such as obligation, right, permission, duty, power, etc., has a long tradition that has been developed in the area of Deontic Logic. One may say that Deontic Logic came into existence in 1951 with the publication of von Wright's paper *Deontic Logic* [24] that inspired, directly or indirectly, most of the works that followed it. Presented initially as a first-order formalism, Deontic Logic has been developed after as a branch of modal logic.

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 [14] 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 [23] 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 [16].

5 Conclusions

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. 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 our 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.

A possible future extension could be the use of obligations on relative intervals and repetitive actions. We study the possibility of integrating a language for describing user-defined periods as presented in [13] and used in [22].

Acknowledgements We would like to thank Gérard Becher for his interest in this work and the anonymous referees for their helpful comments.

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 organizations. 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] B. Leban, D. D. McDonald, and D. R. Forster. A representation for collections of temporal intervals. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 367–371, Philadelphia, 1986.

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

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

[16] 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.

[17] 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.

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

[22] P. Terenziani. Integrated temporal reasoning with periodic events. *Computational Intelligence*, 16(2):210–256, 2000.

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

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