A framework to model norm dynamics in Answer Set Programming

Sofia Panagiotidi and Juan Carlos Nieves and Javier Vázquez-Salceda
Knowledge Engineering and Machine Learning Group
Universitat Politecnica de Catalunya, Spain
{panagiotidi,jcnieves,jvazquez}@lsi.upc.edu

***Abstract*—This paper presents ongoing work in the formal definition and implementation of a normative framework in Answer Set Programming. The framework uses as basis an existing action language and enriches it with a formal definition of norms (implementing standard deontic operators such as obligations and permissions). Properties of a norm's lifecycle such as active, inactive, violated are specified. We argue that such properties can serve as a reasoning basis for the agent's cycle. A partial implementation of the framework is then presented. An example is used in order to illustrate the application of the principles presented.**

I. Introduction

The field of normative systems is an active area where researchers try to find formalisations and mechanisms to model and reason about normative statements. There is a lot of formal work on the formalisation of norms, typically based in variants of monadic or dyadic Deontic Logic. In this way, normative systems predefine the desired behaviour in terms of deontic concepts (obligations, prohibitions, permissions), which typically are extended with other concepts such as deadlines, violations and sanctions. But there is few work on how to bring these theories into practice.

One of the main barriers for the use of deontic-like formalisations is the lack of operational semantics [1]. Without clear operational semantics an agent cannot clearly reason about the influence of these norms in its practical reasoning mechanism. Some attempts have been made to reduce deontic formalisations of norms into operational formalisations based on, e.g. dynamic Logic [2] or Linear Temporal Logic [3], but often such formalisations cannot really be used at execution time.

Other attempts focus on extending existing operational formalisms with some normative concepts. These attempts mainly concern the design of action languages based on the effects of axioms and drawing inferences from the axioms and concentrate on ways of reasoning about actions. [4] presents an action description language capable of expressing causal laws which describe effects of actions as well as statements about values of fluents in possible states of the world. Work of the same authors covers some aspects of reasoning over dynamic domains [5] and representation and reasoning over properties of actions [6]. In [7] the authors intent to provide a logic based language in order to represent authorizations and obligation policies within dynamic environments as well as methods for checking compliance of performed actions with the defined policies. Still, the work lacks support for time. Insight into intelligent decision and possible prediction of desired or non-desired behaviours in advance seems to be absent too.

We are aware of the work in [8] in which the authors present a language for implementing multi-agent systems consisting of individual agents that interact with a computational organization specified in terms of roles, norms and sanctions. Still, this piece of work adopts an organisation-centered view more on the flexible role enactment within a BDI cycle while ours assumes a static role enactment and focuses on a decision making mechanism inferring possible future action taking and violated states.

In the last two decades, one of the most successful logic programming approach has been Answer Set Programming (ASP) [9]. ASP is the realization of much theoretical work on Non-monotonic Reasoning and Artificial Intelligence applications. It represents a new paradigm for logic programming that allows, using the concept of *negation as failure*, to handle problems with default knowledge and produce non-monotonic reasoning. Since its inception, ASP has been regarded as the computational embodiment of Nonmonotonic Reasoning and a primary candidate for an effective knowledge representation tool. This view has been boosted by the emergence of highly efficient solvers for ASP [10], [11], [12]. The efficiency of the answer set solvers has allowed to increase the list of ASP's practical applications, *e.g.*, planning [13], Bioinformatics [14], [15], argumentation theory [16], *etc*. Answer Set Programming does not represent a conventional logic programming approach as PROLOG. Usually an answer set program P can be regarded as a specification of a given problem where each answer set (a model) of P represents a possible solutions of the given problem.

In this paper (Figure 1) we propose, based on a previous normative formalisation [17], a methodology to represent such normative frameworks in Answer Set Programming. This representation can be used as a basis for decision making procedures as well as planning methodologies. We use as example the obligations between two actors (a Insurance Company and a Repair Company) defined in a electronic contract. This contract is composed by deontic clauses which will become norms of the expected behaviour of the contractual parties. Therefore each instantiated contract becomes a normative context at execution time.

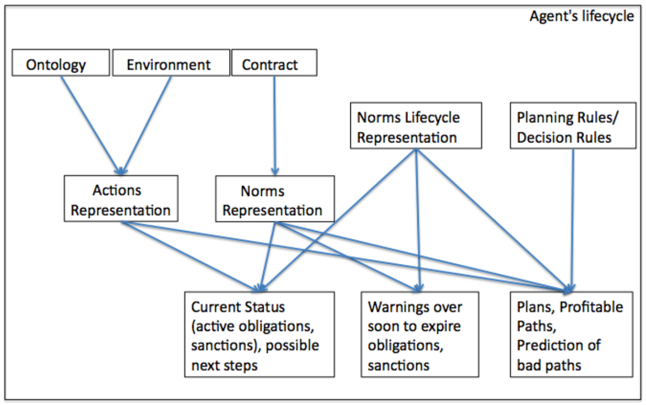

Agent's lifecycle
Ontology
Environment
Contract
Norms Lifecycle Representation
Planning Rules/ Decision Rules
Actions Representation
Norms Representation
Current Status (active obligations, sanctions), possible next steps
Warnings over soon to expire obligations, sanctions
Plans, Profitable Paths, Prediction of bad paths

Fig. 1. Architecture of the normative framework implementation

The work is at a preliminary state. Still, integrated within an agent's operating cycle, such a framework can operate as the brain of the agent [18], providing the necessary information over the current state of the world such as pending norms and currently violated obligations. Additional layers of planning rules and preferred choices might lead to the desired intelligence and flexibility an agent needs to perform in a normative environment. We define several layers of abstraction (Figure 1). An agent (possibly interacting with an ontology) can interpret the environment it operates in through an action language. The deontic restrictions (which in this paper come from a contract, but in other scenario could come from a Institutional specification) provide guidelines over the agent's behaviour and are formalised through a deontic framework.

Generic rules over the interpretation of norms, the interpretation of the action language and planning rules defined with respect to the action language might be added on top of the previous in order to be able to extract information over the agent's state, know the possible actions to take, predict possible violations and foresee future paths. Our interest and focus can be summarised in the following four:

- An action language representation
- Contract (viewed as a set of norms) representation
- Generic rules over the norm lifecycle
- Generic planning and choice rules

Currently, the three first of the above are implemented while the fourth and equally interesting issue is under research. In this paper we are aiming to provide the functionality of the aforementioned as well as some preliminary planning rules. We argue that the implemented layers leave further space for future research over the agent's planning capabilities and behavioural choices.

The rest of the paper is structured as follows. Section II defines the stable model semantics which is the basis for Answer Set Programming. In section III we present a simple motivating example which supports our work while in section IV we detail the action and normative language used and define formal properties of the framework with respect to the aforementioned. A partial implementation of the action and normative language in Answer Set Programming is provided in sections V and VI and parts of the motivating example presented in the beginning are cited in section VIII. We conclude in section IX and summarise future work.

II. Background

In this section, we present some basic definitions *w.r.t.* normal programs and the stable model semantics.

A. Syntax

A signature $\mathcal{L}$ is a finite set of elements that we call atoms. A *literal* is either an atom a, called *positive literal*; or the negation of an atom $not\ a$, called *negative literal*. Given a set of atoms $\{a_1, ..., a_n\}$, we write $not\ \{a_1, ..., a_n\}$ to denote the set of atoms $\{not\ a_1, ..., \ not\ a_n\}$. A *normal* clause, C, is a clause of the form

$$a \leftarrow b_1, \ldots, b_n,\ not\ b_{n+1}, \ldots,\ not\ b_{n+m}$$

where a and each of the b_i are atoms for $1 \leq i \leq n+m$. In a slight abuse of notation we will denote such a clause by the formula $a \leftarrow \mathcal{B}^+ \cup\ not\ \mathcal{B}^-$ where the set $\{b_1, \ldots, b_n\}$ will be denoted by $\mathcal{B}^+$, and the set $\{b_{n+1}, \ldots, b_{n+m}\}$ will be denoted by $\mathcal{B}^-$. We define a *normal program* P, as a finite set of normal clauses.

If the body of a normal clause is empty, then the clause is known as a *fact* and can be denoted just by $a \leftarrow$. or simply a. We write $\mathcal{L}_P$, to denote the set of atoms that appear in the clauses of P. We denote by $HEAD(P)$ the set $\{a | a \leftarrow \mathcal{B}^+,\ not\ \mathcal{B}^- \in P\}$.

B. Stable model semantics.

The stable model semantics was defined in terms of the so called *Gelfond-Lifschitz reduction* [19] and it is usually studied in the context of syntax dependent transformations on programs. The following definition of a stable model for normal programs was presented in [19]:

Let P be any normal program. For any set $S \subseteq \mathcal{L}_P$, let P^S be the definite program obtained from P by deleting

(i) each rule that has a formula $not\ l$ in its body with $l \in S$, and then

(ii) all formulæ of the form $not\ l$ in the bodies of the remaining rules.

Clearly P^S does not contain not. Hence S is a stable model of P if and only if S is a minimal model of P^S.

In order to illustrate this definition let us consider the following example:

Let $S = \{b\}$ and P be the following logic program:

$b \leftarrow\ not\ a.$ $\quad b \leftarrow \top.$

$c \leftarrow\ not\ b.$ $\quad c \leftarrow a.$

We can see that P^S is:

$b \leftarrow \top.$ $\quad c \leftarrow a.$

Notice that P^S has three models: $\{b\}$, $\{b, c\}$ and $\{a, b, c\}$. Since the minimal model amongst these models is $\{b\}$, we can say that S is a stable model of P.

III. Example

The focus of interest of this example is to show how a contract and a set of instantiated norms over the repair of a car operate within the domain and normative environment implemented in ASP. Such a case is useful to demonstrate how reasoning and planning over the norms defined in a contract can be done throughout its execution.

Our scenario is as follows: Two actors, a Repair Company and a Client create a contract ($RepairContract$) over the repair of a broken car. Initially, the Client takes the car to the Repair Company. Once the car is there, the Repair Company has to repair it. Whenever it is repaired, the Client has to pay a previously agreed price. Once the car is repaired, the client has the right to make a complaint.

Figure 2 depicts a pseudo-diagram including the interesting states that the process will go into.

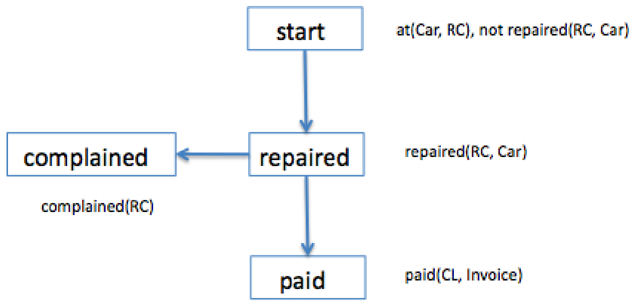


Fig. 2. Pseudo states diagram

The focus of interest of this example is to show how the norms of an electronic contract (explicitly expressed as clauses in the contract) can be mapped into ASP rules and, in a combination with an appropriate action language, to reason over, plan future actions and detect violations within the agent's scope.

IV. Language used

A. Actions

The action description in our approach is done as in most action description languages. In this work we use the formalisation of [6] but adapt several rules to fit our requirements.

Thus the alphabet of the action language $\mathcal{A}$ consists of two nonempty disjoint sets of symbols F and A. They are called the set of fluents and the set of actions. Fluents express the property of an object in a state of the world. A fluent or *fluent literal* (e.g. car_at_shop) is a *positive literal* or a *strongly negated literal* (a fluent preceded by $\neg$). A *state* σ is a collection of fluents. We say a fluent f holds in a state σ if $f \in \sigma$. We say a fluent literal $\neg f$ holds in σ if $f \notin \sigma$.

An action domain description D within the action language $\mathcal{A}$ consists of effect propositions of the following form:

$$a \quad \textbf{causes} \quad f \quad \textbf{if} \quad p_1, ..., p_n, \neg q_1, ..., \neg q_r \qquad (4.1)$$

where a is an action in A, f is a fluent literal in F (also called $postcondition$ of the action a) and $p_1, ..., p_n, q_1, ..., q_r$ are fluents (also called $preconditions$ of the action a) in F.

We assume a set of initial observations O that consists of value propositions of the following form:

$$f \quad \textbf{after} \quad a_1, ..., a_m \qquad (4.2)$$

where f is a fluent literal and $a_1, ..., a_m$ are actions. Intuitively that means that f will hold after the execution of $a_1, ..., a_m$.
When $a_1, ..., a_m$ is an empty sequence then we write:

$$\textbf{initially} \quad f \qquad (4.3)$$

We also condense a set of value propositions of the form $\{$**initially** f_1, ..., **initially** $f_n\}$ by

$$\textbf{initially} \quad f_1, \; ..., \; f_n \qquad (4.4)$$

The role of effect propositions is to define a transition function from states and actions to states. Given a domain description D, such a transition function Φ should satisfy the following properties. For all actions α, fluents g, and states σ:

- if D includes an effect proposition of the form (4.1) where f is the fluent g and $p_1, ..., p_n, \neg q_1, ..., \neg q_r$ hold in σ then $g \in \Phi(\alpha, \sigma)$;
- if D includes an effect proposition of the form (4.1) where f is a negative fluent literal $\neg g$ and $p_1, ..., p_n, \neg q_1, ..., \neg q_r$ hold in σ then $g \notin \Phi(\alpha, \sigma)$;
- if D does not include such effect propositions then $g \in \Phi(\alpha, \sigma)$ iff $g \in \sigma$;

If such a transition function exists, then we say that D is consistent, and refer to its transition function by Φ_D. Given a consistent domain description D the set of observations O is used to determine the states corresponding to the initial situation, referred to as *the initial states* and denoted by σ_0. While D determines a unique transition function, an O may not always lead to a unique initial state.

We say σ_0 is an initial state corresponding to a consistent domain description D and a set of observations O, if for all observations of the form (4.2) in O, the fluent literal f holds in the state $\Phi(\alpha_m, \Phi(\alpha_{m-1}, ...\Phi(\alpha_1, \sigma_0)...))$ (We will denote this state by $[\alpha_m, ..., \alpha_1]\sigma_0$). We then say that (σ_0, Φ_D) *satisfies* O.

Given a consistent domain description D and a set of observations O, we refer to the pair (Φ_D, σ_0) where Φ_D is the transition function of D and σ_0 is an initial state corresponding to (D, O) as a *model* of (D, O). We say (D, O) is *consistent* if it has a model and say it is *complete* if it has a unique model.

We say a consistent domain description D in the presence of a set of observations O entails a query Q of the form (4.2) if for all initial states σ corresponding to (D, O), the fluent literal f holds in the state $[\alpha_m, ..., \alpha_1]\sigma_0]$. We denote this as $D \models_O Q$.

Example 1: With respect to our example, let's assume that the set of actions $A = \{repair(Actor),\ pay(Actor),\ complain(Actor)\}$, the set of fluents

$F = \{car\_at\_shop,\ repaired(Actor),$
$paid(Actor),\ complained(Actor)\}$

and that the domain description consists of the following:

$repair(Actor)$ **causes** $repaired(Actor)$ **if** car_at_shop

$pay(Actor)$ **causes** $paid(Actor)$

$complain(Actor)$ **causes** $complained(Actor)$

Also suppose we have a set of observations

$O = \{$**initially** $car\_at\_shop,\ \neg repaired(rc),$
$\neg\ paid(cl),\ \neg\ complained(cl)\}$. □

A set of observations O is said to be initial state complete, if O only consists of propositions of the form (4.3) and for all fluents, either **initially** f is in O, or **initially** $\neg\ f$ is in O, but not both.

B. Deontic statements

Norms (clauses) express agreements between parties in the form of deontic statements. In order to express the clauses we have adopted a variation [20] [17] of the norm representation defined in [3].

Since norms may have normative force only in certain situations, they are associated with an activation condition. Norms are thus typically abstract, and are instantiated when the norm's activation condition holds. Once a norm has been instantiated, it remains active, irrespective of its activation condition, until a specific expiration condition holds. When the expiration condition occurs, the norm is assumed no longer to have normative force. Independent of these two conditions is the norm's normative goal, which is used to identify when the norm is violated (in the case of an obligation), or what the agent is actually allowed to do (in the case of a permission). Obligations and permissions are the two norm types on which our framework focuses. For every norm there is a maintenance condition which, if does not hold while the norm is active then a violation is considered to be taking place. Finally, for every norm there exists another one (violation norm) indicating what happens if it gets violated. Such properties of norms with respect to a normative environment are implemented in section VI.

We assume a set of norms DS. A norm N in DS is represented as a tuple of the form [17]:

$<NormType, DeonticStatement, ActivatingCondition,$
$MaintenanceCondition, DisactCondition, Actor>$ (4.5)

such that:

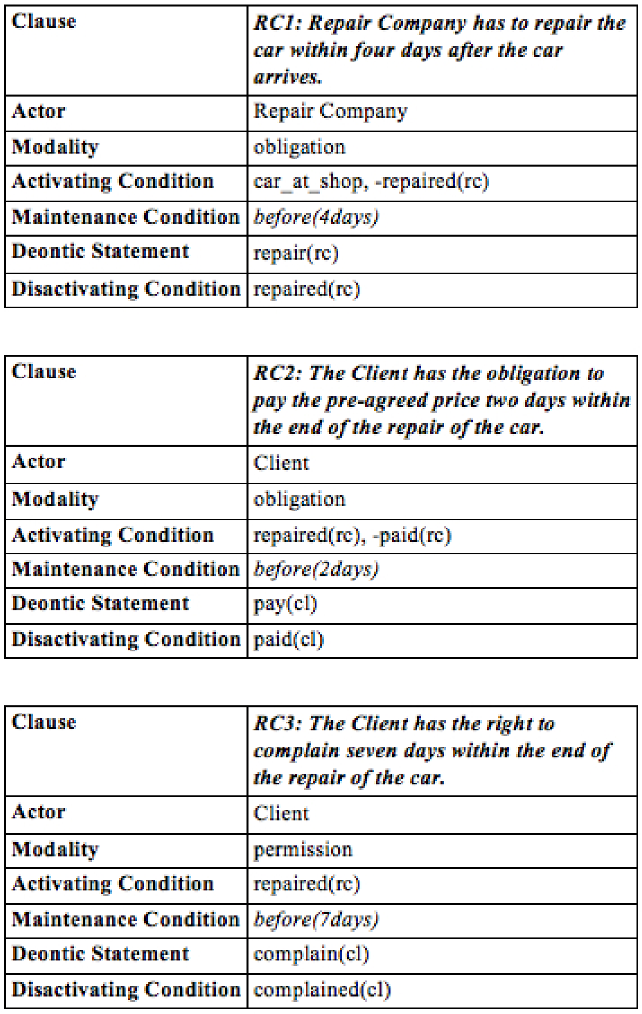
- $NormType \in$ {obligation, permission}
- $DeonticStatement$ is an action a in the set of actions A_{norm} subset of A
- an $ActivatingCondition$ is of the type: $k_1, ..., k_n, \neg l_1, ..., \neg l_n$, where $k_1, ..., k_n, l_1, ..., l_n$ are fluents in F
- a $DisactCondition$ is of the type: $u_1, ..., u_n, \neg v_1, ..., \neg v_n$, where $u_1, ..., u_n, v_1, ..., v_n$ are fluents in F
- a $MaintenanceCondition$ is of the type: $w_1, ..., w_n, \neg z_1, ..., \neg z_n$, where $w_1, ..., w_n, z_1, ..., z_n$ are fluents in F
- a $Actor$ is the actor responsible for the deontic statement

The definition and semantics of permissions are not trivial. For example, it is dubious whether their existence can be assumed within the context of an environment without obligations. In [21] the authors discuss the different types of permissions and their role in deontic logic. We assume that permissions are an explicit indication on what an actor can perform. As in standard deontic logic we assume further that whenever there is an obligation it is always implied that the same is a permission too ($O_a \rightarrow P_a$, axiom D of deontic logic).

Example 2: Figure 3 depicts the formal representation of the deontic statements within the contract of the example presented in the third section. □

Below we give some definitions with respect to the action language and the deontic statements. Let a norm N be in the form of (4.5) and σ be a state.

- N is activated in σ iff $k_1, ..., k_n, \neg l_1, ..., \neg l_n$ hold in σ. We denote this as $D \models_{DS,\ O,\sigma}\ activated(N)$. In the contrary case, N is not activated. We denote this as $D \models_{DS,\ O,\sigma}\ \neg activated(N)$.
- N is disactivated in σ iff $u_1, ..., u_n, \neg v_1, ..., \neg v_n$ hold in σ. We denote this as $D \models_{DS,\ O,\sigma}\ disactivated(N)$. In the contrary case, N is not disactivated. We denote this as $D \models_{DS,\ O,\sigma}\ \neg disactivated(N)$.
- N is violated in σ iff at least one of $w_1, ..., w_n, \neg z_1, ..., \neg z_n$ does not hold in σ. We denote this as $D \models_{DS,\ O,\sigma}\ violated(N)$. In the contrary case, N is not violated. We denote this as $D \models_{DS,\ O,\sigma}\ \neg violated(N)$.



Clause	***RC1: Repair Company has to repair the car within four days after the car arrives.***
Actor	Repair Company
Modality	obligation
Activating Condition	car_at_shop, -repaired(rc)
Maintenance Condition	*before(4days)*
Deontic Statement	repair(rc)
Disactivating Condition	repaired(rc)

Clause	***RC2: The Client has the obligation to pay the pre-agreed price two days within the end of the repair of the car.***
Actor	Client
Modality	obligation
Activating Condition	repaired(rc), -paid(rc)
Maintenance Condition	*before(2days)*
Deontic Statement	pay(cl)
Disactivating Condition	paid(cl)

Clause	***RC3: The Client has the right to complain seven days within the end of the repair of the car.***
Actor	Client
Modality	permission
Activating Condition	repaired(rc)
Maintenance Condition	*before(7days)*
Deontic Statement	complain(cl)
Disactivating Condition	complained(cl)

Fig. 3. Formal representation of norms in the Repair Contract

Queries with respect to norms consist of value propositions of the form:

$$\begin{array}{l} activated(N) \quad \textbf{after} \quad a_1, ..., a_m \\ disactivated(N) \quad \textbf{after} \quad a_1, ..., a_m \\ violated(N) \quad \textbf{after} \quad a_1, ..., a_m \end{array} \qquad (4.6)$$

where N is a deontic statement of the form (4.5), and $a_1, ..., a_m$ are actions in A.

We say a consistent domain description D in the presence of a set of observations O and a set of norms DS entails a query Q of the form (4.6) if for all initial states σ_0 corresponding to (D,O) the proposition $activated(N), disactivated(N), violated(N)$ holds respectively in the state $[\alpha_m, ..., \alpha_1]\sigma_0$. We denote this as $D \models_{DS,\,O} Q$.

Example 3: Let's assume the set of actions A, the set of fluents F and the set of observations O in example 1 and the set of norms DS in example 2. O is an initial state complete. Then the initial state corresponding to (D, O) will be

$\sigma_0 = \{\textbf{initially}\ car\_at\_shop,\ \neg repaired(rc),$
$\neg\ paid(cl),\ \neg\ complained(cl)\}$.

Then:

$D \models_{DS,\,O} activated(N_{RC2})$ since $activated(N_{RC2})$ holds in the state $[repair(rc)]\sigma_0$.

$D \models_{DS,\,O} activated(N_{RC3})$ since $activated(N_{RC3})$ holds in the state $[repair(rc)]\sigma_0$.

where N_{RC2} is the norm $RC2$ and N_{RC3} is the norm $RC3$. □

Proposition 1: Let D be a consistent domain description, O be an initial state complete set of observations and DS a set of deontic statements. If (D, O) is consistent then $\forall N \in DS$ and for any state σ reached by Φ_D:

- $D \models_{DS,O,\sigma} activated(N)$ or $D \models_{DS,O,\sigma} \neg activated(N)$
- $D \models_{DS,O,\sigma} disactivated(N)$ or $D \models_{DS,O,\sigma} \neg disactivated(N)$
- $D \models_{DS,O,\sigma} violated(N)$ or $D \models_{DS,O,\sigma} \neg violated(N)$

C. Temporal Projection

Planning: In the case of planning we are given a domain description D, a set of observations about the initial state O, a collection of deontic statements DS and a collection of fluent literals $G = \{g_1, ..., g_h\}$ which we will refer to as goal. We are required to find a sequence of actions $a_a, ..., a_n$ such that for all $1 \leq i \leq h, D \models_O g_i$ **after** $a_1, ..., a_n$ and for all states σ passed throughout the execution of the actions, $\nexists\sigma$ such that $D \models_{DS,O,\sigma}\ violated(N)$.

V. Representation of Actions

Given an action domain description D, a set of observations O and a set of deontic statements DS we construct a program that partially implements the above. The basic elements and definitions of types are (π_{basic}). The program putting into effect $(D,\ O)$ consists of three parts namely $\pi_{ef},\ \pi_{obs},\ \pi_{in}$ which represent the effect propositions, observations and inertia rules. In addition, π_{norm} and (π_{lc}) partly implement the norms specification and norm lifeycle. We denote the union of $\pi_{basic} \cup \pi_{ef} \cup \pi_{obs} \cup \pi_{in} \cup \pi_{norm} \cup \pi_{lc}$ as π.

In addition to the approach of [6], we assume that actions are durative. That is, they have a starting and ending time and a duration which is the interval between the two. For this, we use predicates like *started* and *teminated* to indicate the time that an action starts being executed and the time it terminates.

A. Basic elements

We introduce here some predicates (π_{basic}) which avoid the program to have weakly restricted rules.

(1) The predicate $fluent$. For every fluent $f \in F$ we add the following rule to the program.

$fluent(f) \leftarrow .$

(2) The predicate $action$. For every action $a \in A$ we add a rule:

$action(a) \leftarrow .$

(3) The predicate $time$. The predicate is used to represent number of distinct times which in their turn represent the time flow. Initially $time(T)$ where T is a constant.

B. Action rules

(1) Translating effect propositions (π_{ef}). The effect propositions in D are translated as follows. For every effect proposition of the form (4.1) if f is a fluent in F then the following rules are created:

$holds(f,\ T+1) \leftarrow \quad terminated(a,\ T),\ time(T).$

$ab(f,\ a,\ T) \leftarrow terminated(a,\ T),\ time(T).$

else if f is the negative fluent literal $\neg g$ then the following rules are created:

$\neg\ holds(g,\ T+1) \leftarrow terminated(a,\ T),\ time(T).$

$ab(g,\ a,\ T) \leftarrow terminated(a,\ T),\ time(T).$

Intuitively, predicate $holds$ indicates whether a fluent's value is true or false throughout time. The ab predicate represents changes of a fluent's value at some specific timestep.

(2) Translating observations (π_{obs}). We intentionally omit observations of the type (4.2) and assume that all observations are of the type (4.3). The value propositions in O then are translated as follows. For every value proposition of the form (4.3) if f is a positive fluent literal then the following rule is made:
$holds(f,1) \leftarrow .$

else if f is the negative fluent literal $\neg g$ then the following rule is created:

$\neg\ holds(g,1) \leftarrow .$

(3) Inertia rules (π_{in}). Besides the above, we have the following inertia rules. The first one makes sure that if no action is executed during a timestep then a fluent will keep holding a true value at the next timestep if it already holds true at the current timestep. The second one makes sure that if a fluent holds a true value and the action that is performed at some timestep does not make it false, then in the next timestep it will also hold true. The third one says that if no action is executed during a timestep then a fluent will keep holding a flse value at the next timestep if it already holds false at the current timestep. The fourth one makes sure that if a fluent holds a false value and the action that is performed at some timestep does not make it true, then in the next timestep it will also hold false.

$holds(F,\ T+1) \leftarrow holds(F,\ T),\ not\ terminated(A,\ T).$

$$\begin{aligned} holds(F,\ T+1) \leftarrow\ & holds(F,\ T),\ terminated(A,\ T), \\ & not\ ab(F,\ A,\ T). \end{aligned}$$

$$\begin{aligned} \neg\ holds(F,\ T+1) \leftarrow\ & not\ holds(F,\ T), \\ & not\ terminated(A,\ T). \end{aligned}$$

$$\begin{aligned} \neg\ holds(F,\ T+1) \leftarrow\ & not\ holds(F,\ T),\ terminated(A,\ T), \\ & not\ ab(F,\ A,\ T). \end{aligned}$$

The following proposition demonstrates how the $holds$ predicate of π will hold a true value at some time step if f holds true at some state σ reached by the function Φ_D.

Proposition 2: Let D be a consistent domain description and O be an initial complete set of observations such that $(D,\ O)$ is consistent. Let $(\sigma_0,\ \Phi_D)$ be the unique model of $(D,\ O)$ and M a stable model of π. If f is a fluent in F then

$f \in \Phi_D(\alpha_n, ..., \Phi_D(\alpha_1,\ \sigma_0))$ iff $holds(f,\ T) \in$ M for some $T \in \mathbb{N}$.

(4) Executability rules. It is assumed that every action might be executed whenever the conditions are fulfiled. We use the predicate $executable$ to model the state where the preconditions of action are fulfiled and the action might be executed.

Thus, for every effect proposition of the form (4.1) the following is created:

$$\begin{aligned} executable(a,\ T) \leftarrow\ & holds(p_1, T), ..., holds(p_n, T), \\ & not\ holds(q_1, T), ..., not\ holds(q_n, T), \\ & time(T). \end{aligned}$$

VI. Representation of Deontic Statements

We assume that for every norm N of the form (4.5) a unique id id is assigned to it. In order to represent a norm, a rule of the respective type below is made (π_{norm}).

$$\begin{aligned} \mathit{activating\_condition}(id,\ a,\ T) \leftarrow\ & \mathit{holds}(k_1,\ T), ...,\\ & \mathit{holds}(k_n,\ T),\\ & \mathit{not\ holds}(l_1,\ T), ...,\\ & \mathit{not\ holds}(l_n,\ T),\\ & \mathit{time}(T). \end{aligned}$$

$$\begin{aligned} \mathit{disact\_condition}(id,\ a,\ T) \leftarrow\ & \mathit{holds}(u_1,\ T), ...,\\ & \mathit{holds}(u_n,\ T),\\ & \mathit{not\ holds}(v_1,\ T), ...,\\ & \mathit{not\ holds}(v_n,\ T),\\ & \mathit{time}(T). \end{aligned}$$

$$\begin{aligned} \mathit{maintain\_cond}(id,\ a,\ T) \leftarrow\ & \mathit{holds}(w_1,\ T), ...,\\ & \mathit{holds}(w_n,\ T),\\ & \mathit{not\ holds}(z_1,\ T), ...,\\ & \mathit{not\ holds}(z_n,\ T),\\ & \mathit{action}(\mathit{Action}), \mathit{time}(T). \end{aligned}$$

Keeping in mind the normative semantics described in IV-B we define the rules that implement a norm's lifecycle (π_{lc}).

$$\begin{aligned} & \mathit{obligation\_activated}(Id,\ \mathit{Action},\ T) \leftarrow\\ & \qquad \mathit{activating\_condition}(Id,\ \mathit{Action},\ T),\\ & \qquad \mathit{not\ active}(Id,\ \mathit{Action},\ T-1),\\ & \qquad \mathit{action}(\mathit{Action}), \mathit{time}(T),\\ & \qquad \mathit{obligation\_id}(Id). \end{aligned}$$

$$\begin{aligned} & \mathit{obligation\_disactivated}(Id,\ \mathit{Action},\ T) \leftarrow\\ & \qquad \mathit{disact\_condition}(Id,\ \mathit{Action},\ T),\\ & \qquad \mathit{active}(Id,\ \mathit{Action},\ T-1),\\ & \qquad \mathit{action}(\mathit{Action}), \mathit{time}(T),\\ & \qquad \mathit{obligation\_id}(Id). \end{aligned}$$

The two rules below define when an obligation is active.

$$\begin{aligned} & \mathit{active}(Id, \mathit{Action},\ T) \leftarrow\\ & \qquad \mathit{obligation\_activated}(Id,\ \mathit{Action},\ T),\\ & \qquad \mathit{action}(\mathit{Action}), \mathit{time}(T),\\ & \qquad \mathit{obligation\_id}(Id). \end{aligned}$$

$$\begin{aligned} \mathit{active}(Id, \mathit{Action},\ T) \leftarrow\ & \mathit{active}(Id,\ \mathit{Action},\ T-1),\\ & \mathit{not\ obligation\_disactivated}(\mathit{Action},\ T),\\ & \mathit{action}(\mathit{Action}), \mathit{time}(T), \mathit{obligation\_id}(Id). \end{aligned}$$

VII. Representation of Planning

We assume that for every action we have an estimated duration on its execution time:

$$\mathit{estimated\_time}(a, t) \leftarrow\ .$$

where t is a constant.

The generic rule

$$\begin{aligned} \mathit{started}(\mathit{Action}, T) \leftarrow\ & \mathit{obligation\_activated}(Id, \mathit{Action}, T),\\ & \mathit{executable}(\mathit{Action}, T),\ \mathit{time}(T),\\ & \mathit{action}(\mathit{Action}), \mathit{obligation\_id}(Id). \end{aligned}$$

ensures that an action starts whenever it is executable and an obligation to execute it becomes active.

$$\begin{aligned} & \mathit{terminated}(\mathit{Action},\ T2) \leftarrow \mathit{started}(\mathit{Action},\ T1),\\ & \qquad \mathit{estimated\_time}(\mathit{Action},\ \mathit{EstimatedTime}),\\ & \qquad T2\ =\ T1 + \mathit{EstimatedTime},\\ & \qquad \mathit{time}(T2),\ \mathit{time}(T1). \end{aligned}$$

The generic rule

$$\begin{aligned} & \mathit{violation}(Id,\ \mathit{Action},\ T) \leftarrow \mathit{active}(Id,\ \mathit{Action},\ T),\\ & \qquad \mathit{not\ maintain\_cond}(Id,\ \mathit{Action},\ T),\\ & \qquad \mathit{action}(\mathit{Action}),\ \mathit{time}(T),\ \mathit{obligation\_id}(Id). \end{aligned}$$

detects a violation whenever a clause is active and the maintenance condition is not true.

The following propositions ensure the correctness of the program π implementing the norm's lifecycle properties.

Proposition 3: Let D be a consistent domain description, O be an initial state complete set of observations such that $(D,\ O)$ is consistent, DS a set of norms and M a stable model of π. If there is a state σ reached by Φ_D such that $D \models_{DS,O,\sigma} \mathit{activated}(N)$ then there is a $T \in \mathbb{N}$ such that $\mathit{activating\_condition}(id, a, T) \in M$ where id is the unique identifier of the norm N and a is the $\mathit{DeonticStatement}$ of the norm N.

Proposition 4: Let D be a consistent domain description, O be an initial state complete set of observations such that $(D,\ O)$ is consistent, DS a set of norms and M a stable model of π. If there is a state σ reached by Φ_D such that $D \models_{DS,O,\sigma} \mathit{disactivated}(N)$ then there is a $T \in \mathbb{N}$ such that $\mathit{disact\_condition}(id, a, T) \in M$ where id is the unique identifier of the norm N and a is the $\mathit{DeonticStatement}$ of the norm N.

Proposition 5: Let D be a consistent domain description, O be an initial state complete set of observations such that $(D,\ O)$ is consistent, DS a set of norms and M a stable model of π. If there is a state σ reached by Φ_D such that $D \models_{DS,O,\sigma} \mathit{violated}(N)$ then there is a $T \in \mathbb{N}$ such that $\mathit{maintain\_cond}(id, a, T)$ and $\mathit{active}(id, a, T) \in M$ where

id is the unique identifier of the norm N and a is the $DeonticStatement$ of the norm N.

Example 4: Let D, O, DS be the domain description, observations and deontic statements of the examples 1 and 2, π the program modeling D, O and DS and M one of its stable models. As seen in the example 3, in the state $\sigma = [repair(rc)]\sigma_0$ $activated(N_{RC2})$ will hold. Thus, there for some $T \in \mathbb{N}$ $activating\_condition(id_{RC2}, repair(rc), T) \in M$ where id_{RC2} is the unique identifier of the norm RC2.

VIII. Example

In our example, the initial observation can be:

$holds(at\_shop,\ 0)$.

$time(0..30)$.

The model will include the following:

$obligation\_activated(rc1, repair(rc), 0)$
$obligation\_disactivated(rc1, repair(rc), 10)$

$obligation\_activated(rc2, pay(cl), 10)$
$obligation\_disactivated(rc2, pay(cl), 11)$

$action\_started(repair(rc), 0)$
$action\_finished(repair(rc), 10)$

$action\_started(pay(cl), 10)$
$action\_finished(pay(cl), 11)$

For further information and the ASP code, the reader is referred to the web page where the code is hosted: *http://www.lsi.upc.edu/ panagiotidi/aspcode*

IX. Conclusions, Discussion and Future Work

Assuming the environment that an agent operates can be modelled by an action languange, norms impose restrictions on the behaviour of the agent. This paper presents ongoing work on the formal specification and development of a framework that enables to model an action language with respect to a normative environment. Semantics of the action language are based on [6] and the language is partially implemented and further enriched with temporal projection. The deontic statements such as obligations and permissions as well as properties related to the norm's lifecycle (activation, disactivation, violation) are formalised and implemented according to [20]. An attempt to define and apply planning techniques within the formal model and the implementation is made.

This piece of work can be extended in various directions. Formal aspects of planning need to be carefully considered and taken into consideration while advancing with the use of the framework in order to reach conclusions on the future states. A formal notion of goal needs to be introduced in order to reach a desired state. Another interesting aspect to be examined is the use of weighted rules when reasoning with respect to norms against the possible effects of one or more violations. This will ensure that each agent can make decisions on execution time based not only on a set of norms indicating a desired behaviour but also on criteria such as desires, preferences and time constraints.

X. Acknowledgements

This research has been funded by the FP7-215890 ALIVE project, funded by the European Commission. Javier Vázquez-Salceda's work has been also partially funded by the "Ramón y Cajal" program of the Spanish Ministry of Education and Science.

References

[1] J. Vázquez-Salceda and F. Dignum, "Modelling electronic organizations," *Multi-Agent Systems and Applications III: 3rd. International/Central and Eastern European Conference on Multi-Agent Systems -CEEMAS'03, Lecture Notes in Artificial Intelligence 2691*, pp. 584–593, 2003. [Online]. Available: http://people.cs.uu.nl/dignum/papers/harmonia-CEEMAS03.pdf

[2] J.-J. C. Meyer and R. J. Wieringa, "Deontic logic: a concise overview," pp. 3–16, 1993.

[3] H. Aldewereld, *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols*. PhD thesis, Utrecht University, 2007.

[4] C. Baral, M. Gelfond, and A. Provetti, "Representing actions: Laws, observations and hypotheses," *The Journal of Logic Programming*, vol. 31, no. 1-3, pp. 201–243, 1997.

[5] C. Baral and M. Gelfond, "Reasoning agents in dynamic domains," 2000, pp. 257–279.

[6] M. Gelfond and V. Lifschitz, "Representing action and change by logic programs," *Journal of logic programming*, 1993.

[7] M. Gelfond and J. Lobo, "Authorization and obligation policies in dynamic systems," *Proceedings of the 24th International Conference on Logic Programming*, pp. 22–36, 2008.

[8] N. Tinnemeier, M. Dastani, and J.-J. Meyer, "Roles and norms for programming agent organizations," in *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 121–128.

[9] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge: Cambridge University Press, 2003.

[10] S. DLV, "Vienna University of Technology," http://www.dbai.tuwien.ac.at/proj/dlv/, 1996.

[11] S. SMODELS, "Helsinki University of Technology," http://www.tcs.hut.fi/Software/smodels/, 1995.

[12] Potassco, "University of Potsdam," http://potassco.sourceforge.net/, 2007.

[13] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres, "A logic programming approach to knowledge-state planning, ii: The DLV^{k} system." *Artif. Intell.*, vol. 144, no. 1-2, pp. 157–211, 2003.

[14] S. Dworschak, S. Grell, V. Nikiforova, T. Schaub, and J. Selbig, "Modeling Biological Networks by Action Languages via Answer Set Programming," *Constraints*, vol. 13, no. 1, pp. 21–65, 2008.

[15] C. Baral, K. Chancellor, N. Tran, N. Tran, A. M. Joy, and M. E. Berens, "A knowledge based approach for representing and reasoning about signaling networks," in *ISMB/ECCB (Supplement of Bioinformatics)*, 2004, pp. 15–22.

[16] J. C. Nieves, M. Osorio, and U. Cortés, "Preferred Extensions as Stable Models," *Theory and Practice of Logic Programming*, vol. 8, no. 4, pp. 527–543, July 2008.

[17] N. Oren, S. Panagiotidi, J. Vázquez-Salceda, S. Modgil, M. Luck, and S. Miles, “Towards a formalisation of electronic contracting environments,” *Coordination, Organization, Institutions and Norms in Agent Systems, the International Workshop at AAAI 2008, pages 61-68, Chicago, Illinois, USA*, 2008.

[18] R. Confalonieri, S. Alvarez-Napagao, S. Panagiotidi, J. Vázquez-Salceda, and S. Willmott, “A middleware architecture for building contract-aware agent-based services,” *LECTURE NOTES IN COMPUTER SCIENCE*, Jan 2008. [Online]. Available: http://www.springerlink.com/index/m84n55h118j86178.pdf

[19] M. Gelfond and V. Lifschitz, “The stable model semantics for logic programming.” *R. Kowalski and K. Bowen, editors, 5th Conference on Logic Programming, pages 1070âĂŞ1080. MIT Press*, 1988.

[20] S. Panagiotidi, J. Vázquez-Salceda, S. ÃĄlvarez Napagao, S. Ortega-Martorell, S. Willmott, R. Confalonieri, and P. Storms, “Intelligent contracting agents language,” *Proceedings of the Symposium on Behaviour Regulation in Multi-Agent Systems -BRMAS’08-, Aberdeen, UK, April 2008*, 1988.

[21] G. Boella and L. van der Torre, “Permissions and obligations in hierarchical normative systems,” *Proceedings of the 9th international conference on Artificial intelligence and law*, pp. 109–118, 2003.