AI Communications 32 (2019) 319–334
DOI 10.3233/AIC-190617
IOS Press

Using Semantic Web Technologies and Production Rules for Reasoning on Obligations, Permissions, and Prohibitions [1,2,3]

Nicoletta Fornara [a,*] and Marco Colombetti [b]
[a] *Università della Svizzera italiana, via G. Buffi 13, 6900 Lugano, Switzerland*
[b] *Politecnico di Milano, piazza Leonardo Da Vinci 32, Milano, Italy*

Abstract. Nowadays the studies on the formalization, enforcement, and monitoring of policies and norms is crucial in different fields of research and in numerous applications. ODRL 2.2 (Open Digital Right Language) is a W3C standard policy expression language formalized using semantic web technologies. It is used to represent permitted and prohibited actions over a certain asset, and obligations required to be met by parties involved in the exchange of a digital asset. In this paper, we propose to extend the model of obligation, permission, and prohibition proposed by ODRL 2.2 in two directions. Firstly, by inserting in the model the notion of activation event (or action) and by expressing event and action as complex constructs having types and application-independent properties. Secondly, by considering the temporal aspects of obligations, permissions, and prohibitions (e.g. expiration dates and deadlines) as part of their application independent model. These extensions are necessary in order to be able to propose an application-independent operational semantics of the extended model, which is formalized using State Machines and is computed by a specific production rule system. The proposed approach has been tested by developing a framework in Java able to get as input a set of policies formalized using Semantic Web languages, and to compute their evolution in time based on the events and actions that happen in the interaction among the parties involved in the specified policies.

Keywords: Norms, Policies, Obligations, Prohibitions, Permissions, Multiagent Systems, Semantic Web Technologies, Production Rules

1. Introduction

Nowadays the study of policies and norms is crucial in different fields of research and applications. Policies may be applied for regulating access to data and digital assets in policy-based access control frameworks. They may be employed to specify licenses for usage of software, images, video and data or to formalize contracts, agreements, and offers between different parties in e-commerce applications. Privacy policy may also be used to express regulations on the management of personal and sensitive data.

Policies and norms can be expressed using human-readable formats. However, it is necessary to specify them with formal and machine-readable languages in order to enable machine-to-machine interactions enriched with a number of useful services. Examples of services are: (i) advanced search of resources based on the actions that it is possible to perform on them; (ii) aggregation of different resources released under different policies by computing policies compatibility or conflicts; (iii) *monitoring* the satisfaction or violation of the normative relations that an intensive exchange of digital assets creates in the chain of interactions among data producers, data publishers, and data consumers; (iv) *simulating* the evolution of a set of policies on the

[1]Funded by the SNSF (Swiss National Science Foundation) grant no. 200021_175759/1.

[2]This paper is a substantially revised version of the paper: Fornara N., Chiappa A., Colombetti M. (2019) Using Semantic Web Technologies and Production Rules for Reasoning on Obligations and Permissions. In: Lujak M. (eds) Agreement Technologies. AT 2018. Lecture Notes in Computer Science, vol 11327, pages 49-63. Springer, Cham.

[3]The final publication is available at IOS Press through http://dx.doi.org/10.3233/AIC-190617

[*]Corresponding author. E-mail: nicoletta.fornara@usi.ch.

basis of an hypothetical set of events and actions in order to perform for example a what-if analysis. In order to realize the last two services, it is crucial not only to propose a language for expressing policies, but also to unambiguously specify what it means for an agent to have a set of specific policies.

ODRL 2.2 (Open Digital Right Language)[4] is a W3C standard policy expression language, which is used to represent permitted and prohibited actions over a certain asset, and obligations required to be met by the parties involved in the exchange of a digital asset. Originally, in 2001, ODRL was an XML language for expressing digital rights, that is, digital content usage terms and conditions. In 2012 (version 2.0) and in 2015 (version 2.1) [1], ODRL evolved into a more general policy language: it is no longer focused only on the formalization of rights expressions, but also on the specification of privacy statements, like duties, permissions, and prohibitions. ORDL started to be formalized in RDF with an abstract model specified by an RDF Schema Ontology. In March 2016, a W3C Working Group was created with the goal of bringing the specifications through the W3C Process to "Recommendation" status. ODRL 2.2 became a W3C Recommendation on 15th February 2018. In all the specifications of ODRL, its semantics is described informally in English, and no formal specification is provided. In literature there is a paper [2] where an OWL representation of ODRL 1.1 is presented, but the use of OWL is limited to the static representation of policies, and there is not a specification of their dynamic semantics, that is, of how they evolve in time. In another paper, [3], the semantics of ODRL 2.1 policies, used in the field of access control, is investigated. The focus of this paper is on explicit and implicit dependencies among the regulated actions. In particular, when a request to perform an action on an asset is issued, the system evaluates which rules (prohibition, permission, or duty rules) are triggered, then it checks whether these rules hold based on certain constraints, but there is no hint on how the satisfaction of such constraints can be computed.

In this paper, we propose to extend the model of permission, duty, and obligation proposed by ODRL 2.2 in two directions. Firstly, by inserting in the model the notion of activation event/action and by expressing event and action as complex constructs having types and application-independent properties. Secondly, by considering the temporal aspects of obligations and permissions (their expiration dates and the deadlines) as part of their application-independent model. Then, we propose to formalize the operational semantics of the extended model of obligations, permissions, and prohibitions by using State Machines, which are used to unambiguously specify the temporal evolution of their *deontic state* while time passes and relevant events (e.g. the elapsing of a deadline) or actions (e.g. downloading a music file) happen. Such an operational semantics can be efficiently computed by a *monitor* component by using a production rule system. The proposed approach has been tested by developing a framework in Java able to get as input a set of policies formalized using Semantic Web languages, and to compute their evolution in time based on the events and actions that take place. Such a framework uses the forward chaining rule-based RETE engine of the Apache Jena framework[5] (which is compatible with semantic web languages) for realizing the production system.

The paper is organized as follows. In Section 2, a semantic meta-model for expressing temporal and conditional obligations, permissions, and prohibitions is introduced. In Section 3, the life cycles of those deontic relations is formally specified. In Section 4, a production rule system is used for computing the deontic state of obligations, permissions, and prohibitions. In Section 5, a prototype for simulating the evolution in time of deontic relations is described. Finally, in Section 6 other approaches for expressing policies and norms are presented and discussed.

2. A Semantic Web Meta-model of Conditional Obligations, Permissions, and Prohibitions

Given that ODRL is a W3C standard, the ODRL 2.2 Information Model[6] is specified as an RDF Ontology formalized using RDF Schema[7], a semantic web standard language for expressing data-model vocabulary for RDF data. RDF Schema can be used for defining classes, domain and range of properties and hierarchies of classes and of properties. The provided ODRL specification is compatible with another standard semantic web language for expressing ontologies, the OWL 2 Web Ontology Language, which is a practical serialization of the SROIQ(D) Description Logic. Therefore, given that we want to propose an extension of ODRL, we will formalize our normative meta-model

[4]https://www.w3.org/TR/odrl-model/

[5]https://jena.apache.org/documentation/inference/index.html

[6]https://www.w3.org/ns/odrl/2/

[7]https://www.w3.org/TR/rdf-schema/

using an OWL ontology: the *Normative Language Ontology* (*NL Ontology*). This choice involves the advantage of being able to perform automatic reasoning on the OWL formalization of the proposed deontic concepts and the chance, given by the adoption of Semantic Web Languages, to connect our ontology with other, quite well known, OWL and RDF ontologies.

Coherently with the ODRL 2.2 Information Model[8], in our model a policy must have at least one permission, prohibition or obligation. In ODRL the Permission, Prohibition, and Duty classes are subclass of the Rule class, which is used as domain of the common properties of these three classes. In this paper, we will introduce a different formalization of the notion of obligation, prohibition, and permission. In order to extend the ODRL model in this direction, we introduce in our Normative Language Ontology three new classes: the Permission, Prohibition, and Obligation classes. Given that every obligation, prohibition, or permission is a *deontic relation*, those three classes are subclass of the DeonRelation class, which is a subclass of the ODRL Rule class.

There are three main differences between ODRL model of deontic concepts and the one proposed in this paper. The first one is that in ODRL model the regulated *actions* are expressed by means of their textual name (e.g. "print"), and their semantics can be narrowed by using *constraints* (i.e. expressions which compare two operands). For example it is possible to express the permission to print less than or equal to 1200 dpi resolution[9]. We think that the characteristics of the regulated actions have not simply to be expressed using some generic constraints, but they must be expressed using an application-independent model of those actions. Therefore, we propose to specify the regulated actions using complex constructs having a type (i.e. a class) and a set properties shared as much as possible with de facto standard used for describing actions in the Web.

The other differences are due to the fact that the ODRL model does not put in evidence two fundamental application-independent aspects of deontic relations. The first aspect is the important role played by the *activation condition* [4–6], i.e. the event or action that can *activate* an obligation, put *in force* a prohibition, or make *valid* a conditional permit. Think for example to the obligation to pay a ticket that becomes active when someone enters a limited traffic area, or to the permit to play a music file during a party that becomes valid only after a fee is paid. In ODRL, an activation condition may be represented using a *generic* constraint or a duty (only for permissions), but this choice does not highlight a crucial component in the model of a deontic relation, which is fundamental for the operational semantics discussed in the next section. Moreover, the ODRL proposal to constrain a permission using a duty, where the duty is a pre-condition that must be fulfilled to obtain a valid permission, is rather unnatural because a duty is an action than an agent is obligated to do, not an action that an agent can freely decide to perform.

The other application-independent characteristic of the modelled deontic concepts, which has important implications on the operational semantics proposed in this paper, is the relation between those deontic concepts and *time*. Usually an obligatory action has to be performed before a specific *deadline*, a prohibition may have a termination instant of time, and a permission can be used within a certain interval of time. For example, an agent may have the obligation to pay 5 euro before the end of the month, or the permission to play a music file for one week. Moreover, a conditional deontic relation may become *expired* (it ceases to exist) if it is not activated before a given *expiration date*. For example, when an auction is open, every participant is obliged to pay for her/his bids in case s/he becomes the winner of the auction, when an auction is closed, such an obligation expires and cannot become active anymore.

Taking into account all these considerations, in our meta-model a *deontic relation* is defined, using a tuple notation, as follows:

Definition 1 (Deontic relation) A deontic relation is a tuple: $\langle$*type, debtor, creditor, activation condition, content, deadline, expiration date, counter*$\rangle$ where:

- *type* is used for distinguishing between an obligation, a prohibition, or a permission;
- *debtor* and the *creditor* are the two *parties* involved in the *deontic relation*. The debtor of an obligation is the agent who has to execute the action described in the content, and the creditor is the agent to whom the execution is owed. The debtor of a prohibition is the agent who has not to execute the action described in the content and the creditor is the agent who is wronged by the execution of the action. The debtor and the creditor of a permission used for derogating to a prohibition are respectively the creditor and the debtor of the prohibition.

[8]https://www.w3.org/TR/odrl-model/#infoModel

[9]The ODRL 2.2 formalization of this permission is available at https://www.w3.org/TR/odrl-model/#constraint-action

- *activation condition* is the description of the class of activating events and *content* is the description of the class of regulated actions. Those classes of events or actions are described by specifying their type (e.g. listen action or pay action) and the value of a set of relevant properties (e.g. the author of the track or the amount of money);
- *deadline* and *expiration date* are relevant instant of time. The expiration date is the instant of time when a conditional deontic relation ceases to exist. The deadline is the instant of time before which it is obligatory to satisfy the content of an active obligation, or it is permitted to exercise a valid permission, or the instant of time when an existing and in force prohibition is terminated. Deadlines and expiration dates may be computed at run-time by using a specified interval of time. In those cases their value depends on when the deontic relation has changed its deontic state. When the deadline and the expiration date of a deontic relation are fixed (i.e. they are not computed at run-time) the expiration date must be before or equal the deadline, this because the deontic relation should not become active (or valid or in force) when the deadline is already expired;
- *counter* is used for managing the obligations and permissions to perform an action more than once.

A deontic relation is formalized in our *Normative Language Ontology* as an individual of the DeonRelation class. The *type* attribute is expressed introducing the Permission, Prohibition, and Obligation subclasses of the DeonRelation class. The other properties are expressed by introducing in the OWL ontology specific object properties, like for example the hasActCondition or hasContent properties. The complete *Normative Language Ontology* is depicted in Figure 1[10]. The DeonState and ComponentState classes have been introduced for representing the state of a deontic relation and the state of its components (i.e. the activation condition and the regulated content) as discussed in the next section.

In the definition of the *Normative Language Ontology* we exploit the possibility, given by the adoption of Semantic Web Languages, to import in our ontology other quite well known and already existing ontologies. We import the ODRL 2.2 ontology because we extend it by defining a new subclass of the ODRL Rule class[11]. We import the core model of temporal entities specified in the *OWL Time Ontology*[12] for being able to specify deadline and expiration dates, and the time when specific events or actions happen. We import the *Schema.org* vocabularies[13], a shared vocabulary developed to support web search engines. We use it for the specification of actions as complex objects, contrary to their treatment in ODRL, where they are represented by atomic symbols. Finally, the *Normative Language Ontology* uses the *Event Ontology* presented in [4] for expressing events as a super-class of actions and for connecting events to time instants and intervals. The various import relationships defined among those ontologies are depicted in Figure 2.

2.1. Policies examples

In a real system, it is desirable that digital assets are associated with a *policy schema* that can be re-used in different circumstances, for example the schema of a contract or of an agreement. Policy schemas may contain variables, for example the price of the negotiated product may change from one contract to another, or they may be related to classes of objects, for example one schema of a contract may be applied to every agent playing a certain role. Policy schema may be transformed into *policy instances* through a procedure of substitution of variables with actual values specified during an interaction with a user or through a procedure of substitution of the name of a class with specific individuals belonging to that class.

Given that the ODRL policy language can only be used for expressing policy instances, in the extension of the ODRL model presented in this paper, our focus is, similarly, on the formalization of specific policies. The primary aim of the ODRL model is to "*cover as many permission, prohibition, and obligation use cases as possible, while keeping the policy modelling easy even when dealing with complex cases*". ODRL was specified considering 27 use cases[14], for example the specification of common licenses (e.g. the RDFLicense

[10]The name of the classes and properties in Figure 1 are prefixed with the name of the specific ontology where they are defined as discussed below. The `nl` prefix is used for the Normative Language Ontology. The arrows with the black tip are used to represent the subclass relationship, and the arrows with the thin tip are used to represent OWL object and data properties.

[11]This procedure is consistent with the one specified in the "ODRL profile mechanism" section of the ODRL Information Model 2.2.

[12]https://www.w3.org/TR/owl-time/ W3C Recommendation 19 October 2017

[13]http://schema.org/docs/developers.html

[14]http://w3c.github.io/poe/ucr/

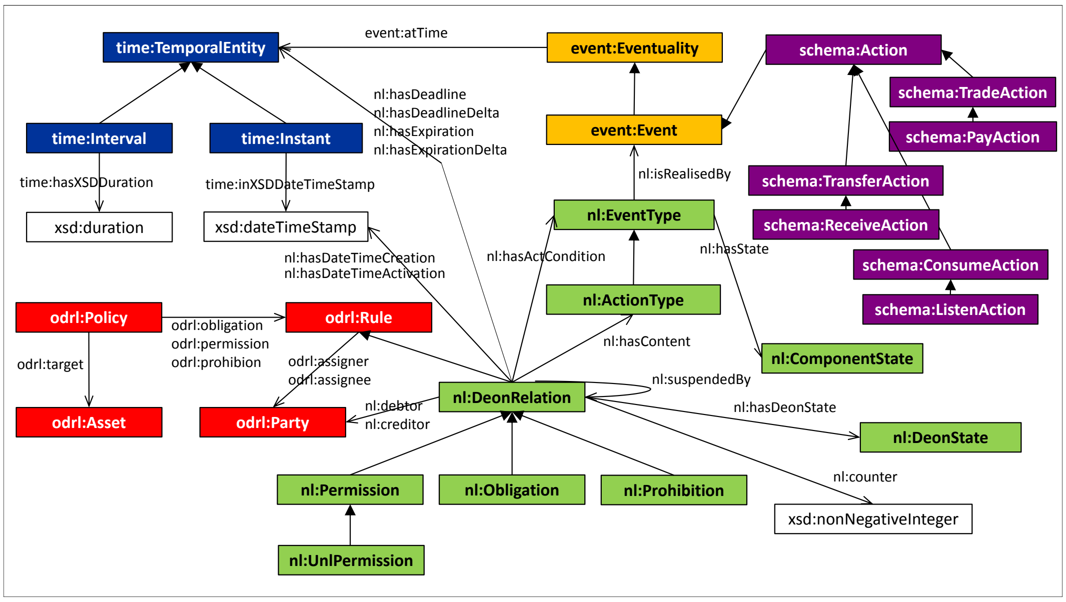

Fig. 1. The OWL Normative Language Ontology and its connections with other imported ontologies

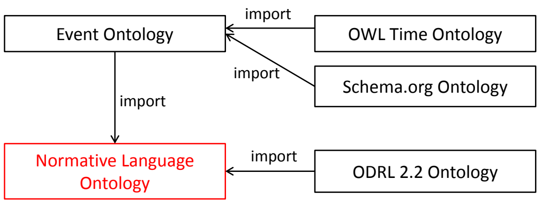

Fig. 2. The import relationship among the various ontologies

dataset contains over 100 licenses written in RDF by using ODRL 2.0 [7]) or of access policies to digital resources.

The expressive power of the model of deontic relations presented in this paper is not substantially different from the expressive power of ODRL 2.2. This because in both models it is possible to express the parties involved in the deontic relation and some conditions/refinements on the regulated action. Using the ODRL model it is possible to express some conditions (i.e. constraints) that have to be satisfied for obligations and prohibitions (or duties for permissions). Differently, in the model presented in this paper the notion of *activation condition* is introduced. In ODRL deadlines and expiration dates are simply expressed using temporal constrains and are not part of the application independent model. In addition in ODRL it is also possible to express constrains with the less than or greater than operator and the consequences of not fulfilling an obligation or the remedy for exercising a prohibited action. In our future works, we plan to extend the proposed model in order to be able to express and manage sanctions for the violation of deontic relations. The substantial difference between the two models is not in their expressive power, but on the fact that by inserting in the application independent model the notion of activation condition, deadline and expiration date we are able to propose an application independent operational semantics for our deontic relations as will be discussed in next sections.

Due to space limitations, in this section we present only one example of a conditional obligation and one example of a prohibition that can be suspended by a permission. One example of conditional obligation that may appear in an agreement is *"the obligation for Bob to pay (within 5 minutes) the organization Zally 15 euro when a specific book is delivered at his home"*. It can be formalized as a tuple as:

$\langle$*obligation, Bob, Zally, ReceiveAction(Bob, Zally, Book(Semantic Web)), PayAction(Bob,Zally,15euro), 5m, 2029-01-01, 1*$\rangle$

It can be formalized as a set of RDF statements, i.e. a collection of triples, each consisting of a subject, a predicate and an object, and expressed in the textual syntax for RDF called Turtle[15] as follows:

[15]https://www.w3.org/TR/turtle/, in Turtle every row is an RDF triple terminated by '.'. The ';' symbol is used to repeat the same subject for different triples. Square brackets, '['...']', are used for unlabelled blank nodes.

```
policy01 rdf:type odrl:Agreement;
         odrl:obligation obl01.
obl01 rdf:type nl:Obligation;
   nl:debtor Bob;
   nl:creditor Zally;
   nl:hasDeadlineDelta [time:hasXSDDuration
        "P0DT0H5M"^^xsd:duration];
   nl:hasExpiration [time:inXSDDateTimeStamp
      "2029-01-01T00:00:00Z"^^xsd:dateTime];
   nl:counter 1;
   nl:hasActCondition [
      rdf:type schema:ReceiveAction;
      schema:agent Bob;
      schema:sender Zally;
      schema:object [
        rdf:type schema:Book;
        schema:headline Semantic Web]];
   nl:hasContent [
      rdf:type schema:PayAction;
      schema:agent Bob;
      schema:recipient Zally;
      schema:price 15.00;
      schema:priceCurrency "euro"].
```

One example of prohibition is the *"prohibition for Bob, with respect to Soyn, to listen a song by Beatles until the end of 2020"*. Such a prohibition may be suspended by a *"conditional permit allowing Bob, until the end of 2019, to listen to, at most ten times, a song by Beatles, in the 48 hours following the payment of 5 euro to Soyn"*. Those deontic relations can be formalized as tuples as:

⟨*prohibition, Bob, Soyn, -, ListenAction(Bob, MusicRecording(Beatles)), 2020/12/31, -, -* ⟩

⟨*permission, Soyn, Bob, PayAction(Bob,Soyn,5euro), ListenAction(Bob,MusicRecording(Beatles)), 48h, 2019/12/31, 10* ⟩

They can be formalized as one prohibition and one permission belonging to an agreement by using the Normative Language Ontology. The resulting formalization as RDF statements is as follows:

```
policy02 rdf:type odrl:Agreement;
   odrl:prohibition proh02;
   odrl:permission perm02.

proh02 rdf:type nl:Prohibition;
   nl:debtor Bob;
   nl:creditor Soyn;
   nl:suspendedBy perm02;
   nl:hasDeadline [ time:inXSDDateTimeStamp
      "2020-12-31T23:59:59Z"^^xsd:dateTime];
   nl:hasContent [
     rdf:type schema:ListenAction;
     schema:agent Bob;
     schema:object [
       rdf:type schema:MusicRecording;
          schema:byArtist Beatles]].

perm02 rdf:type nl:Permission;
  nl:debtor Soyn;
  nl:creditor Bob;
  nl:hasDeadlineDelta [time:hasXSDDuration
              "PT48H0M0S"^^xsd:duration];
  nl:hasExpiration [time:inXSDDateTimeStamp
     "2019-12-31T09:00:00Z"^^xsd:dateTime];
  nl:counter     10^^xsd:integer.
  nl:hasActCondition  [
    rdf:type schema:PayAction;
    schema:agent Bob;
    schema:recipient Soyn;
    schema:price 5.00;
    schema:priceCurrency "euro"].
  nl:hasContent  [
    rdf:type schema:ListenAction;
    schema:agent Bob;
    schema:object [
    rdf:type schema:MusicRecording;
    schema:byArtist Beatles]].
```

3. Dynamic evolution of Obligation, Prohibition, and Permission

ODRL 2.2 has not a formal semantics. The intuitive meaning of its notion of duty, prohibition, and permission is provided only in English. The lack of a formal semantics makes it hard to develop a framework for automatically monitoring or simulating the time evolution of a set of deontic relation instances. For example, it would be hard to automatically evaluate if a given action, for example inserting an image in a presentation for a conference, will bring to the violation of a specific prohibition or if this action will activate a specific obligation. Being able to automatically detect, or simulate for a what if analysis, violations of deontic relation instances, by taking into account the specific actions performed by the agents, is a useful service that is getting more and more strategic in nowadays digital scenario.

Our goal, in this section, is to present, in an unambiguous way, what it means for an agent to have a specific obligation, or prohibition, or permission expressed using the deontic model introduced in the previous section. As we already discussed, deontic relations are widely used for regulating the *actions* that various individuals should, should not, or may perform. The performance of *actions*, and more generally the happening of *events* (i.e. something that happens in a system, but is not necessarily done by an actor like for example a time event) may change the state of the interaction among various parties and the *state* of their deontic relations. In particular, the sate of a deontic relation will change in different ways based on the instant of time when events happen. In order to unambiguously specify what it means for an agent to have an obligation, a prohibition, or a permission, it is therefore fundamental to model the *deontic state* of deontic relations, and to unambiguously specify how it evolves in time, when temporal events happen (e.g. the expiration of a deadline) and actions are performed (e.g. an agent listen a song).

Similarly to what has been done for the notion of commitment in [8] and of norm in [6], the dynamic evolution in time of the *deontic state* of deontic relations, i.e. their life cycle, can be clearly illustrated using *State Machines*. The proposed dynamic evolution of the *deontic state* of the three different types of deontic relations is the result of a deep analysis of the literature (as discussed in 6) on deontic logic and normative systems combined with the textual description of the notion of permission, prohibition, and duty presented in the ODRL Information Model 2.2. It is important to underline that we are able to propose an application independent life cycle for every type of deontic relation thanks to the changes introduced in their model.

It is important to underline that the proposed dynamic evolution of the *deontic state* of obligations, prohibitions, and permissions in turn depends on the satisfaction or not of their *activation condition* and of the action described in the *content*. Such a satisfaction can be computed by a specific procedure, described in the next section, consisting in checking if the representation of a specific event happening in the system realises an event type specified in the condition or content of deontic relations. We introduce in the Normative Language Ontology the hasState property for connecting the activation condition or content component with its state. Such a state is initially *unsatisfied* and becomes *satisfied* when a successful realisation is detected. The isRealisedBy property is used to connect the description of an event/action type, in a deontic relation, with a specific happened event.

We will now introduce the life cycles of our model of obligation, permission, and prohibition, which are depicted using State Machines. The life cycle of a *conditional obligation* to perform an action a certain number of times is depicted in Figure 3, where the black dot is the starting state and the states with the double outline are a final states. When a conditional obligation is created by a *creation event* and its activation condition and content components are *unsatisfied*, it goes in the *conditional* state. The action of creating a deontic relation and the action of cancelling it are institutional actions [9] and their actor needs to have the *institutional power* to successfully perform them [10][16]. For example, the creditor of an active obligation may have the power to cancel it. If an *activating event* of a conditional deontic relation happens (i.e. the activation condition becomes *satisfied*) the deontic relation gets *active*. Differently, if the *cancellation event* of a conditional deontic relation happens the deontic relation becomes *cancelled*. In the model presented in this paper, the cancellation event happens when the *expiration date* of the deontic relation is elapsed. In a future improvement of the model, other types of cancellation events may be introduced. When an obligation is in the *active* state, if the *regulated event* happens (i.e. the content component becomes *satisfied*) and the counter is greater than one, the conditional obligation remains *active*, the counter is decremented, and the *content* is set back to *unsatisfied*. If the *regulated event* of an active obligation happens and the counter is equal to one the deontic state becomes the final *fulfilled* state. Differently, if the state is still *active* and the *termination event* happens, i.e. the deadline becomes elapsed, the deontic state becomes the final *violated* state and the obligation cannot be fulfilled anymore. If the final state is *violated* a sanction may be applied [11], similarly if the final state is *fulfilled* a reward may be given.

The notion of *permission* has been widely discussed in the literature and different types of permission have been detected. One first traditional distinction is between *weak* (or *negative*) and *strong* permission [12]. A *weak* permission of doing an action a in a given code is equivalent to the absence of the prohibition to do a in that code, i.e. in a logic formalism, it is not provable that $\neg a$ is mandatory [13]. Differently, a *strong* permission is given in a code when there is the explicit permission do to an action a and usually they are used to explicitly derogate to existing *prohibitions*. Strong permission makes sense even when there is not an incompatible prohibition, but the permission may be used for preventing the creation of future prohibitions [13]. Sometime a strong permission may be used to create an *exemption* to an obligation. Finally, in some scenarios, for example in a P2P data exchange scenario, the notion of strong *permission* may be related to the notion of *obligation*. In fact, having a permission, for ex-

[16]A complete formalization of the notion of institutional power is beyond the scope of this article and we plan to investigate it in our future works.

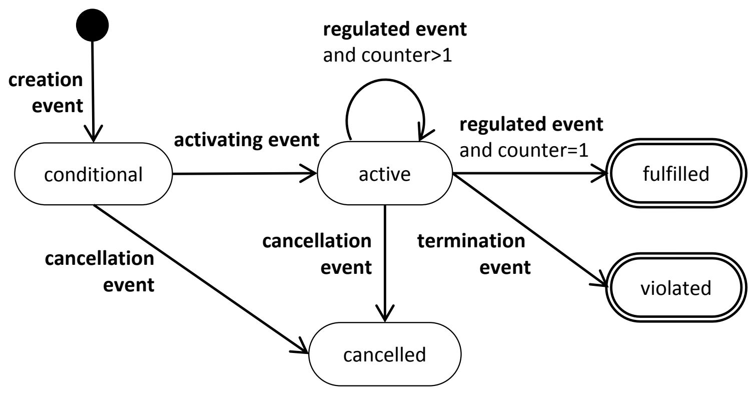

Fig. 3. Life cycle of Conditional Obligation

ample to get access to certain data, may coincide with the *obligation* for the debtor of the permission to enable the holder of the permission to exercise it [14]. Given that in this paper our focus is on the deontic relations formalized in ODRL 2.2 and on the examples discussed in that specification, we will propose a model of the notion of *strong permission* used for explicitly derogating to existing *prohibitions*.

Our assumption in this paper is that a strong permission may be created only by an agent having the *institutional power* to successfully perform it. The life cycle of a *conditional permission* to perform an action a certain number of times is depicted in Figure 4. It is interesting to notice that the conditions that trigger the transitions of the obligation and permission life cycles are identical. The fundamental difference between the two life cycles, which is enlightened by the different name of the deontic states, is that for obligations the *fulfilled* state is a final and desired state, contrary to the *violated* final state which may bring about a sanction. Differently, for permissions the *exercised* or *expired* final states have no positive or negative connotation. The positive and negative connotation can be found again in the life cycle of prohibitions as discussed below.

By using a Finite State Machine it is also possible to formalize the life cycle of a conditional permission to perform an action for an *unlimited* number of times. It is a machine similar to the one depicted in Figure 4 with the difference that there is not a counter to be decreased whenever the permission is used and there is not the exercised state, in fact only when the deadline is elapsed the permission becomes *expired*.

The last fundamental deontic relation, which is part of the ODRL 2.2 model, is the notion of *prohibition*. As previously discussed, in those cases when a permission is used for suspending a prohibition those two deontic relations should be connected, and we connect them in the Normative Language Ontology using the suspendedBy property. We assume that when a permission that regulates a prohibited action is created, the connection between the permission and the prohibition is automatically initialized. We have not modelled a prohibition as an obligation not to perform the regulated action, as it is usual in deontic logic, because in RDF there is not a standard way to represent the negation of an action and its time constraints. The life cycle of a *conditional prohibition* to perform a certain action is depicted in Figure 5. The condition that triggers the transition from *conditional* to *inForce* and then from *inForce* to *cancelled*, *infringed*, and *notInfringed* are the same conditions described for the obligation and for the permission state machines (except for the management of the counter). When a permission connected with an *inForce* prohibition becomes *valid* the prohibition goes to the *suspended* state. When such an active permission goes to a final, *exercised* or *expired*, state the prohibition goes back to the *inForce* state.

It is interesting to observe that the prohibition life cycle differs from the others for the name of the states. Another crucial difference is that the final state that brings to a violation for an obligation (i.e. the *violated* state) is not the corresponding state that brings to a violation for a prohibition (i.e. the *infringed* state). Another interesting novelty is the introduction of the *suspended* state in the prohibition life cycle. We think that that such a difference may be explained in the following way. In this paper, we focused on three deontic concepts: obligation, permission, and prohibition. If we

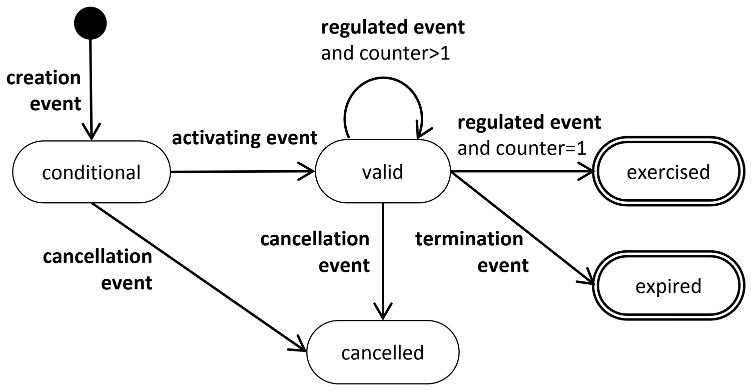

Fig. 4. Life cycle of Conditional Permission

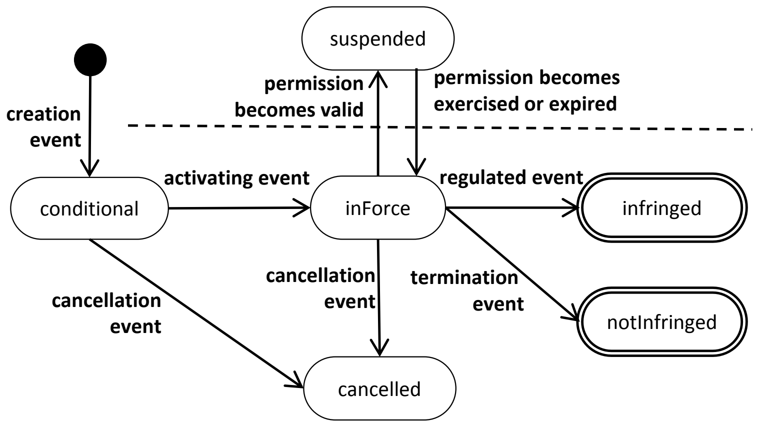

Fig. 5. Life cycle of Conditional Prohibition

will extend our treatment to other concepts, like for example the notion of *exemption*, used for suspending an obligation, the obligation and the prohibition machine will become quite similar. We can observe that the states that all these three machines have in common (except for their name) are the states whose conditions are triggered by internal properties of the deontic relation. Differently, the *suspended* state (that in Figure 5 is separated from the other states by a dashed line) is a state that is reached when some conditions on the deontic state of another deontic relation are satisfied. Given that a deontic relation creates deontic connection between two parties, we can see it as a *first level* connection, therefore the connection between two deontic relations can be seen as a *second level* connection. Therefore, we can conclude that the *suspended* state appears only in the prohibition life cycle because in the model presented in this paper we decided to represent only one *second level* relation among deontic concepts. These observations lead us to think about the possibility to define only one state machine for all these three types of deontic relations, we will investigate this improvement in our future works.

4. Operational Semantics

In order to realize services able to automatically monitor or simulate the dynamic evolution of the proposed deontic relations, it is necessary to define a procedure for automatically computing their *deontic state* on the basis of the life cycles introduced in the previous section. This by taking into account that such a deontic state, in turn, depends on the satisfaction of the *state* of the *activation condition* and *content* of the relevant deontic relations.

Such a procedure can be realized, with a modular architecture, by combining three different components. The first one that we can call "sense and reason", is an application-dependent component, able to: (i) "sense" the actual events or actions that happens in the interaction among agents; (ii) represent them as individuals of the OWL *State Ontology* where events, actions, and the instants of time when they happen are represented using the vocabulary defined in the *Event*, *Schema.org*, and *OWL Time* ontologies; (iii) given that OWL is a practical realization of the SROIQ(D) Description Logic (which is sound and complete), "reason" on the *State Ontology* by using one of the available OWL reasoners. We are then able to exploit our choice of using sematic web technologies for deriving all the entailed assertions. For example given that the PayCash class is a subclass of the PayAction class, it is possible to deduce that if a certain payment belong to the PayCash class it belongs also to the PayAction class.

The other two application-independent components are: the component for computing the satisfaction of the *state* of the condition and content of deontic relation; and the component for computing the dynamic evolution of the *deontic state*. In this paper we propose to realize both of them using a *production system* [15], i.e. a forward-chaining reasoning system that uses *production rules* for reasoning and draw conclusions from facts store in an *ongoing memory*. A *production rule* has two parts: a set of *conditions*, to be tested on the ongoing memory, and a set of *actions*, whose execution has an effect on the content of the ongoing memory. The generic form of a production rule is:

`IF` *conditions* `THEN` *actions*

Production rules may be used to generate an event-based computation, which can carry out an inference process or implement a discrete dynamic system. We will exploit the latter use of production rules for proposing an operation model of the life cycles of obligations, permissions, and prohibitions introduced in the previous section. A crucial advantage of using production rules is the possibility to represent the logic of the dynamic evolution of the proposed deontic relations using a declarative paradigm, where rules can be easily modified, instead of embedding such a logic in the code written in an imperative programming language.

The *ongoing memory* of the proposed production system contains the RDF statements inserted in the *State Ontology*, which is a representation of the state of the interaction among agents in terms of actual actions, events, and current time. The ongoing memory contains also the RDF statements representing the set of policies of a specific normative system (for example the policies reported in Section 2.1) which are expressed using the OWL *Normative Language* ontology. While time flows, the *ongoing memory* is continuously updated with new assertions representing actual events or actions and the elapsing of time, and it is updated due to the execution of those production rules having all conditions satisfied.

Production rules must be written in a formal language in order for computer programs to reason with them using a *rule engine*. There are many rule languages having different syntax for the various existing rules engines (for example the rule language for Drools, Oracle Business Rules, Jess, or Jena). The abundance of different rule languages can create difficulties in integrating rule sets. The W3C Rule Interchange Format (RIF) [17] is a standard developed to facilitate rule sets integration. In order to avoid to bind the production rules presented in this section to a specific rule engine we will formalize the them using the Abstract Syntax of the W3C Recommendation RIF Production Rule Dialect[18] (RIF PRD). In this language the conditions of the production rules (that have to be evaluated on facts written as RDF statements, having a subject a predicate (or RDF property) and an object) are written as binary relations (i.e. the name of the relation is the name of the RDF property) over the subject and object of RDF triples. In RIF-PRD variables are prefixed by '?'.

In the proposed production system, there are two different types of production rules. One type of rules are used for computing the satisfaction of *activation conditions* and *content* component of deontic relations. The second type of rules are used for computing the *deontic state* of obligations, permissions, and prohibitions based on their life cycle.

4.1. First type of production rules

The first type of production rules is used for evaluating if the description of an event or action type

[17]https://www.w3.org/TR/rif-overview/

[18]https://www.w3.org/TR/rif-prd/,http://www.w3.org/TR/rif-primer/

(which appears in the activation condition or content of a deontic relation) is realised by a specific event instance as soon as it is represented in the system. In the meta-model of obligations, permissions, and prohibitions proposed in this paper, a description of an event type (or an action type) is characterized by the specification of its class and by a list of relevant property-values pairs. The satisfaction of one event type can be computed by one production rule by checking the exact match between the described event type and the available event instances, this by taking into account also some time constraints on the instant of time when such event instances happened. Given that the list of properties used for describing an event or an action type can change from one deontic relation to another, we need to specify one production rule for every description of an event or action type. This solution is due to the choice, in line with ODRL, of formalizing policies, and therefore their event types, using the RDF language. In our future works we plan to study the possibility to express our policies using the OWL language and then exploit OWL reasoners for computing if an event type (i.e. an OWL class) is realized by a specific event instance (i.e. an OWL individual).

Due to space limitation, we report here only one example of this type of production rules. It is the rule used for computing if the event type of listening a song by Beatles, which is used in the *content* component of the prohibition and permission reported in Section 2.1, is realized by one of the event instances contained in the *State Ontology*[19].

```
(* ListenActionContent *)
Forall ?deon ?component ?agent ?object
    ?objectClass ?artist ?realEvent ?instant
    ?dateTime ?object1 ?datetimeAct(
If And(
 rdf:type(?deon nl:DeonRelation)
 nl:hasContent(?deon ?component)
 nl:hasDeonState(?deon nl:active)
 nl:hasState(?component nl:unSatisfied)

 rdf:type(?component schema:ListenAction)
 schema:agent(?component ?agent)
 schema:object(?component ?object)
 rdf:type(?object ?objectClass)
 schema:byArtist(?object ?artist)
 rdf:type(?realEvent schema:ListenAction)
 event:atTime(?realEvent ?instant)
 time:inDateTime(?instant ?dateTime)
 schema:agent(?realEvent ?agent)
 schema:object(?realEvent ?object1)
 schema:type(?object1 ?objectClass)
 schema:byArtist(?object1 ?artist)

 nl:hasDateTimeActivation(
                     ?deon ?datetimeAct),
 External(pred:numeric-greater-than(
                ?dateTime,?datetimeAct)))
Then(
 Assert
   (nl:hasState(?component nl:satisfied))
 Assert
   (nl:isRealizedBy(?component ?realEvent))
 Retract
   (nl:hasState(?component nl:unSatisfied)))
```

[19]We assume that it is impossible to insert in the *State Ontology* an event that will happen in the future, therefore, as soon as an event instance is represented in the ontology, it can be considered the realization of one event type.

The last two conditions of the rule are used to check if the action instance that realizes the action type, described in the content of a deontic relation, is happened after the activation of the deontic relation. This check is required in order to avoid that action instances happening before the activation of the deontic relation may erroneously satisfy the conditions of the rule.

4.2. *Second type of production rules*

The second type of production rules is used for computing the *deontic state* of obligations, permissions, and prohibitions coherently with their life cycle. It is necessary to define one production rule for every transition of the life cycles presented in Section 3. The *conditions* of these production rules are used for testing the type of the deontic relation (obligation, permission, or prohibition), its current deontic state, and the conditions that appear on the transition of the life cycle. The *action* component of the rules is used to retract the current deontic state, assert the new one and, if necessary, decrement the counter.

Due to space limitation, we report here only two production rules. The first one is used for computing the transition from the *conditional* to the *valid* state in the life cycle of permissions.

```
(* validatePermission *)
Forall ?perm ?activation ?now (
If And( rdf:type(?perm nl:Permission)
 nl:hasDeonState(?perm nl:conditional)
 nl:hasActCond(?perm ?activation)
 nl:hasState(?activation nl:satisfied)
 time:inDateTime(currentTime  ?now) ))
Then(
 Assert
  (nl:hasDeonState(?perm nl:valid))
 Assert
```

```
   (nl:hasDateTimeActivation(?perm ?now))
  Retract
   (nl:hasDeonState(?perm nl:conditional))
  )
```

When the state of a permission becomes *valid*, another rule is in charge of computing at runtime the deadline value for those cases when it depends on the time of the activation of the permission (e.g. the permission to listen a song for two days starting from the time of the payment that activated it). The second rule is as follows.

```
(* suspendProhibition *)
Forall ?proh ?permission ?dateTime(
If And(
 rdf:type(?proh  nl:Prohibition)
 nl:hasDeonState(?proh  nl:inForce)
 nl:suspendedBy(?proh ?permission)
 nl:hasDeonState(?permission nl:active)
 nl:hasDateTimeActivation(?proh ?dateTime)
))
Then(
 Assert
  (nl:hasDeonState(?proh nl:suspended))
 Retract
  (nl:hasDeonState(?proh  nl:inForce))
 Retract
  (nl:hasDateTimeActivation(?proh ?dateTime)
 ))
```

This rules computes the transition from the *inForce* to the *suspended* state in the life cycle of prohibitions. The instant of time when the prohibition became *inForce* (i.e. when its activation condition was satisfied) is retracted by the last action of the rule. Such an instant of time will be initialized again by the production rule that computes the transition from *suspended* to *inForce*.

5. Implementation of a Java Prototype

For testing our proposal, we have developed a Java prototype, specifically a system able to *simulate* the evolution of policies containing a set of obligations, permissions, and prohibitions. We use Apache Jena[20], a free and open source Java framework for building semantic web applications. For the implementation of the *production system*, described in Section 4, we use the Jena general-purpose rule engine and a translation of the RIF PRD rules into Jena Rules[21]. Differently from other forward rule-based systems, this reasoner natively supports rule-based inference over RDF graphs, and provides forward chaining realized by means of an internal RETE-based forward chaining interpreter[16]. The main advantage of using the JENA interpreter with respect to other Java compatible production rules interpreters, like for instance the Jess engine inspired by the open-source CLIPS project, is its direct compatibility with RDF data [17].

In general, a forward-chaining reasoner of a production system allows data processing through cycles of three stages: matching rules-facts, selection of satisfied rules, and execution of rules. In the first stage reasoner finds all the rules whose *condition* part is satisfied by the facts contained in the ongoing memory. A positive matching leads to the activation of a rule that is inserted in an agenda. In each iteration, an activation in the agenda is chosen and then executed (fired). After the execution of a rule, the facts contained in the ongoing memory are updated as specified in its *action* part, then a new reasoning iteration starts until no rules can be activated with updated facts.

An interesting feature of the Jena forward chaining interpreter is that it works *incrementally*, meaning that if the ongoing memory is modified by adding or removing statements, the production/reasoning process automatically resumes, potentially producing the activation of new rules. The efficiency of the reasoning with respect to these incremental changes is guaranteed by the use of the RETE algorithm, through which matching tests are performed only for those rules whose conditions include an updated fact in the previous iteration.

In our prototype, the RIF PRD external built-in operations of *Retract()* and *Assert()* are realized by means of the default Jena built-in *remove(n)* and the ad-hoc realized *add(triple)* built-in. The *remove(n)* Jena built-in has the side effect of recursively retracting, from the inference model, the consequences of the already fired rules, if their conditions matched with the removed statement. In fact, coherently with its main goal of implementing logical reasoning in RDF and OWL, the Jena interpreter is designed to have a monotonic behaviour. Given that our productions rules are meant to implement State Machines (and not monotonic logical reasoning), we implemented the ad-hoc *add(triple)* built-in, having the effect of inserting a new triple that will not be retracted as a side effect of removing another statement.

As we already discussed in the introduction, one useful service that can be provided in relation to a set

[20]https://jena.apache.org/
[21]https://jena.apache.org/documentation/inference/#rules

of policies is the *monitoring* of their deontic state for those applications where it is crucial to check the fulfilment or violation of obligations, the observance of a prohibitions, or the correct use of permissions. Another relevant service is the *simulation* of the evolution of deontic relations based on a set of hypothetical actions. In order to realize these services it is important to take into account that a few *relevant instants* are truly significant in the life cycle of an obligation, a permission, or a prohibition. Significant instants are the instants when real actions and events happen and the elapsing of deadline and expiration dates. Therefore, in order to realize an efficient simulator[22] it is important that the comparison between the simulated current time and the *significant instants* (stored in an ordered list) occurs only if strictly necessary, i.e. when one of these relevant instants are reached. This is obtained by forcing the current time to evolve to the nearest relevant instant of time. Each update of the current time in the inference model leads to a new cycle of the interpreter, during which the states of obligations, permissions, and prohibitions eventually evolve.

6. Related Work

Studies on Normative Multiagent Systems (NorMAS) [18] concern mainly the proposals of formalisms for expressing norms or policies containing mainly obligations, permissions, and prohibitions. Those studies investigate the realization of fundamental functionalities on those norms/policies, for example promulgation, monitoring and enforcement, simulation, consistency check, as well as norm adoption and reasoning.

In the literature, there are various proposals where a model of norms and policies is proposed using different formal languages and where different frameworks are investigated with the goal of providing different services on policies. The most well-known are the studies on Deontic Logic [19], a family of logical systems where the essential features of obligations, prohibitions and related concepts are captured. Examples of other formal languages, sometime used together with various reasoning techniques, are: the UML notation adopted in [20]; the Jess rule language adopted in [21]; the RuleML language used in the LegalRuleML language[23], which is in the process of becoming an OASIS standard in the legal domain; the OWL semantic web language exploited in [4, 5, 22, 23]; the RDF+RDF Schema language used for the specification of ODRL 2.2 and for the proposal in [3], the deontic logic and the linear temporal logic together with production systems as discussed in [6]; defeasible logic on propositional atoms analysed in [24]; the $Ans-Prolog$ language (a logic programming language under the answer set semantics) used in InstAL [25]; or tuples defined in a set theory like in [14].

Many of those approaches have in common the idea, supported also in this paper but disregarded by ODRL, that it is crucial to represent in the model of norms their *activation* and *deactivation* or *expiration condition* [5, 6, 14, 20]. An idea that is shared also by papers where the notion of commitment, which is really close to the notion of obligation, is modelled [23, 26]. Even if, in few of those approaches there is the possibility to express those activation conditions and the regulated actions using a complex model of actions (easily sharable with other applications) like it is done in this paper. Those approaches share also the idea that deontic relations can be used for regulating the performance of an action or for regulating the maintenance of certain conditions or, of a certain state of affairs. Differently from what we proposed in this paper, only few of the mentioned approaches investigate the relation between deontic relations and relevant instant of time like the deadline and the expiration date, moreover few of them study the dynamic evolution in time of deontic relation instances.

An interesting proposal where an extension of Defeasible Logic (a non-monotonic logic) is proposed for reasoning about obligations, prohibitions, and permissions in force in a given license, and on the composition of different licenses is presented in [24]. In this paper different licenses (having as content obligations, permissions, and prohibitions) for Linked Open Data are formalized in a machine-readable format using the L4LOD (Licenses for Linked Open Data) vocabulary. L4LOD is a vocabulary devised for expressing with one language various licenses languages, like for example the Creative Common licenses or the Open Government License (OGL). The proposed logic allows dealing with permissions as defeaters of prohibitions and on prohibitions understood as negative obligations. The actions that are regulated are those typical of linked open data licences, like *ShareAlike*, *Attribution*, *Commercial* and so on, which are represented as atomic symbols. There is not a representation of time but the focus of the paper is on reasoning on licenses

[22]Taking into account that the *monitoring* service can be realized using the simulator where time and events are real.

[23]https://www.oasis-open.org/committees/legalruleml/

composition, i.e. understanding what actions are permitted, prohibited, or obliged when two or more licences are combined together. Differently in this paper, the focus is not on reasoning on the deontic aspects of a set of policies, but it is on monitoring or simulating the dynamic evolution in time of policies instances in order to detect any misbehaviour of the involved agents. This is an interesting service when the regulated actions are characterized by a complex action type, by some value of their properties and by some time constraints. When the regulated actions are simply propositions that may be true or false is less interesting.

The model presented in [6] shares with the model presented in this paper the idea of representing the life cycle of norm instances with a state machine. Even if, given that the model of norms are different, in [6] there is only one state machine for all deontic relations and such a state machine is substantially different from the three machines presented in this paper. In particular, in [6] a norm instance is used for expressing the obligation to keep a maintenance condition, and if at some point the maintenance condition is not fulfilled the norm gets into a violation state. When the obligation has as maintenance condition a negated action, like for example "do not cross in red light when driving" it becomes a prohibition. Alternatively, the model presented in this paper avoids the problem of treating negated action and can be used for regulating the performance of actions instead of maintenance conditions. Differently from our model, where deadlines and expiration dates are modelled, in [6] only a timeout property, i.e. a deadline for the reparation of the violation of a norm, is taken into account. Another difference is that in [6] the deontic interpretation of a norm is reduced to a temporal logic formulae written in Linear Temporal Logic (LTL) which in turn can be translated into both rule-based and planning operational semantics, whereas in our approach an application independent set of rules is used for computing the deontic state of all policy instances.

Similarly in [21], a normative language for the specification of norms is presented. In such a normative language norms have the form of *preconditions* $\rightarrow$ *postconditions*, and the execution of every norm is implemented by means of an ad-hoc forward rule written for the Jess interpreter[24]. Differently in this paper, we propose a production rule system for computing the application-independent life cycle of policies containing three types of deontic relations.

[24]http://www.jessrules.com/

Given that, in this paper, policies are formalized using Semantic Web Technologies and their life cycle is computed by using an OWL reasoner and production rules, we will compare our approach with other papers where Semantic Web Technologies are used. An interesting literature review of various approaches to policies specification using Semantic Web Technologies is given in [27].

In [4, 26] we presented our first approach to formalize obligations using OWL 2 with SWRL rules and to reason on them using available OWL reasoners and OWL-API. These papers present an OWL ontology of obligations whose content is a class of possible actions that have to be performed within a given deadline. The monitoring of such obligations, checking if they active and then fulfilled or violated on the basis of the actions performed by the agents, is realized by means of a specific Java framework used for managing the elapsing of time and for performing closed-world reasoning on certain classes. This is necessary due to the open-world assumption of OWL logic that creates a problem for successfully monitoring obligations, i.e. when trying to deduce that when the deadline is elapsed an active obligation has to be permanently violated. The Java program was used for computing the explicit closure of certain classes used for representing the cancel and fulfilled states. Unfortunately, the scalability of this approach was not good enough to make it usable in real applications. Therefore, we started to look for another approach for dynamically computing the state of obligations and other deontic relations, and the solution proposed in this paper consists in using production rules for computing the life cycles expressed with state machines.

In [28] a first attempt of expressing conditional obligations to perform one action, as an extension of ODRL 2.1 having a life cycle computed using Jena Rules is presented. The work proposed in this paper is a substantially revised version of the paper [29]. In this new version, we added and discussed the model of prohibitions with its interesting connection with the dynamic evolution of permissions. Moreover, we introduced the idea of running an OWL reasoner on the *State Ontology* before to use it an input for the rule-engine of the production system and we extended the comparison with related works.

An interesting approach that uses Semantic Web Technologies for policy formalization and management is the OWL-POLAR framework [5]. This framework investigates the possibility of using OWL ontologies for representing the state of the interaction among agents and SPARQL queries for reasoning on policies activation, for anticipating possible conflicts among policies, and for conflicts avoidance and resolution. In the OWL-POLAR model, the activation condition and the content of the policies are represented using conjunctive semantic formulas. Reasoning on a set of policies for deducing their state is realized by translating the activation condition and the content of a policy into the SPARQL query language and then evaluating the resulting queries on the OWL ontology used for representing the state of the world. In OWL-POLAR, there is no treatment of time. We think that an advantage of our approach, with respect to OWL-POLAR, is the possibility to compute the operational semantics of policies directly on their RDF formalization without the need of translating them in other languages. Another advantage is in the use of production rules, i.e. a declarative paradigm, for computing the dynamic evolution of the proposed deontic relations instead of coding such a computation with an imperative programming language.

Another relevant proposal is the KAoS policy management framework [22, 30]. In KAoS Semantic Web technologies are used for policy specification and management, in particular policy monitoring and enforcing is realized by a component that compiles OWL policies into an efficient format. In [23] social commitments [8] are used for modelling privacy requirements for social networks formalized using OWL. Similarly to our approach, the antecedent of commitments is a description of conditions that have to be matched with the content of the ontology. However, the consequent of commitments is limited to permissions or prohibitions to see a set of posts, and time is not modelled at all.

In our future work, we plan to further investigate the dynamic connections between obligations, permissions, and prohibitions and other deontic relations like the for example the exemption to an obligation and the right to perform a permitted action. We intend to study a mechanism for improving the procedure for checking if an event instance realizes an event type described in the deontic relations. We need also to further investigate the possibility to use the event of violation or fulfilment of an obligation or of a prohibition for applying rewards or sanctions, and to extend our model with the notion of institutional power.

Acknowledgements

We acknowledge the contribution to this research of Ms Alessia Chiappa during her master thesis at Politecnico di Milano.

References

[1] R. Iannella, S. Guth, D. Paehler and A. Kasten, ODRL Version 2.1 Core Model, 2015, https://www.w3.org/community/odrl/model/2.1/ (accessed 15/09/2017).

[2] A. Kasten and R. Grimm, Making the Semantics of ODRL and URM Explicit Using Web Ontologies, in: *Virtual Goods*, 2010, pp. 77–91.

[3] S. Steyskal and A. Polleres, Towards Formal Semantics for ODRL Policies, in: *Rule Technologies: Foundations, Tools, and Applications - 9th International Symposium, RuleML 2015, Berlin, Germany, August 2-5, 2015, Proceedings*, N. Bassiliades, G. Gottlob, F. Sadri, A. Paschke and D. Roman, eds, Lecture Notes in Computer Science, Vol. 9202, 2015, pp. 360–375. doi:10.1007/978-3-319-21542-6_23.

[4] N. Fornara, Specifying and Monitoring Obligations in Open Multiagent Systems using Semantic Web Technology, in: *Semantic Agent Systems: Foundations and Applications*, Studies in Computational Intelligence, Vol. 344, Springer-Verlag, 2011, pp. 25–46, Chapter 2.

[5] M. Sensoy, T.J. Norman, W.W. Vasconcelos and K.P. Sycara, OWL-POLAR: A framework for semantic policy representation and reasoning, *J. Web Sem.* **12** (2012), 148–160. doi:10.1016/j.websem.2011.11.005.

[6] S. Panagiotidi, S. Alvarez-Napagao and J. Vázquez-Salceda, Towards the Norm-Aware Agent: Bridging the Gap Between Deontic Specifications and Practical Mechanisms for Norm Monitoring and Norm-Aware Planning, in: *Revised Selected Papers of the COIN 2013 - Volume 8386*, Springer-Verlag New York, Inc., New York, NY, USA, 2014, pp. 346–363. ISBN 978-3-319-07313-2. doi:10.1007/978-3-319-07314-9_19.

[7] V. Rodríguez-Doncel, S. Villata and A. Gómez-Pérez, A dataset of RDF licenses, in: *Legal Knowledge and Information Systems - JURIX 2014: The Twenty-Seventh Annual Conference, Jagiellonian University, Krakow, Poland, 10-12 December 2014*, R. Hoekstra, ed., Frontiers in Artificial Intelligence and Applications, Vol. 271, IOS Press, 2014, pp. 187–188. doi:10.3233/978-1-61499-468-8-187.

[8] N. Fornara and M. Colombetti, Operational Specification of a Commitment-based Agent Communication Language, in: *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2*, AAMAS '02, ACM, New York, NY, USA, 2002, pp. 536–542. ISBN 1-58113-480-0. doi:10.1145/544862.544868.

[9] N. Fornara, F. Viganò, M. Verdicchio and M. Colombetti, Artificial institutions: a model of institutional reality for open multiagent systems, *Artificial Intelligence and Law* **16**(1) (2008), 89–105. doi:10.1007/s10506-007-9055-z.

[10] A.J.I. Jones and M. Sergot, A Formal Characterisation of Institutionalised Power, *Logic Journal of the IGPL* **4**(3) (1996), 427–443. doi:10.1093/jigpal/4.3.427.

[11] N. Fornara and M. Colombetti, Specifying and Enforcing Norms in Artificial Institutions, in: *Declarative Agent Languages and Technologies VI*, M. Baldoni, T.C. Son, M.B. van Riemsdijk and M. Winikoff, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 1–17. ISBN 978-3-540-93920-7.

[12] G.H. von Wright, *Norm and Action: A Logical Enquiry*, Routledge and Kegan Paul, 1963.

[13] G. Governatori, F. Olivieri, A. Rotolo and S. Scannapieco, Computing Strong and Weak Permissions in Defeasible Logic, *Journal of Philosophical Logic* **42**(6) (2013), 799–829. doi:10.1007/s10992-013-9295-1.

[14] S. Cauvin, N. Oren and W. Vasconcelos, Policies to Regulate Distributed Data Exchange, in: *AT 2018: 6th International Conference on Agreement Technologies*, Lecture Notes in Computer Science, Springer, 2018, This research is partially sponsored by the EPSRC grant EP/P011829/1, funded under the UK Engineering and Physical Sciences Council Human Dimensions of Cyber Security call (2016)..

[15] R. Brachman and H. Levesque, *Knowledge Representation and Reasoning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. ISBN 1558609326.

[16] C.L. Forgy, On the Efficient Implementation of Production Systems., PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1979, AAI7919143.

[17] J. Moskal and C.J. Matheus, Detection of Suspicious Activity Using Different Rule Engines - Comparison of BaseVISor, Jena and Jess Rule Engines, in: *Rule Representation, Interchange and Reasoning on the Web, International Symposium, RuleML 2008, Orlando, FL, USA, October 30-31, 2008. Proceedings*, N.Bassiliades, G. Governatori and A. Paschke, eds, LNCS, Vol. 5321, Springer, 2008, pp. 73–80. doi:10.1007/978-3-540-88808-6_10.

[18] M. Dastani, J. Dix, H. Verhagen and S. Villata, Normative Multi-Agent Systems (Dagstuhl Seminar 18171), *Dagstuhl Reports* **8**(4) (2018), 72–103. doi:10.4230/DagRep.8.4.72. http://drops.dagstuhl.de/opus/volltexte/2018/9761.

[19] G.H. von Wright, Deontic logic, *Mind, New Series* **60**(237) (1951), 1–15.

[20] K. da Silva Figueiredo, V. Torres da Silva and C. de Oliveira Braga, *Modeling Norms in Multi-agent Systems with NormML*, in: *Coordination, Organizations, Institutions, and Norms in Agent Systems VI: COIN 2010 International Workshops, COIN@AAMAS 2010, Toronto, Canada, May 2010, COIN@MALLOW 2010, Lyon, France, August 2010, Revised Selected Papers*, M. De Vos, N. Fornara, J.V. Pitt and G. Vouros, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 39–57. ISBN 978-3-642-21268-0. doi:10.1007/978-3-642-21268-0_3.

[21] A. Garcia-Camino, P. Noriega and J.A. Rodriguez-Aguilar, Implementing Norms in Electronic Institutions, in: *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, ACM, New York, NY, USA, 2005, pp. 667–673. ISBN 1-59593-093-0. doi:10.1145/1082473.1082575.

[22] A. Uszok, J.M. Bradshaw, J. Lott, M.R. Breedy, L. Bunch, P.J. Feltovich, M. Johnson and H. Jung, New Developments in Ontology-Based Policy Management: Increasing the Practicality and Comprehensiveness of KAoS, in: *POLICY 2008, 2-4 June 2008, Palisades, New York, USA*, IEEE Computer Society, 2008, pp. 145–152. doi:10.1109/POLICY.2008.47.

[23] N. Kokciyan and P. Yolum, PriGuard: A Semantic Approach to Detect Privacy Violations in Online Social Networks, *IEEE Trans. on Knowl. and Data Eng.* **28**(10) (2016), 2724–2737. doi:10.1109/TKDE.2016.2583425.

[24] G. Governatori, A. Rotolo, S. Villata and F. Gandon, One License to Compose Them All - A Deontic Logic Approach to Data Licensing on the Web of Data, in: *ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, H. Alani and L.K. et al., eds, Lecture Notes in Computer Science, Vol. 8218, Springer, 2013, pp. 151–166. doi:10.1007/978-3-642-41335-3_10.

[25] J. Padget, E. Elakehal, T. Li and M. De Vos, *InstAL: An Institutional Action Language*, in: *Social Coordination Frameworks for Social Technical Systems*, Law, Governance and Technology Series, Vol. 30, Springer Verlag, 2016, p. 101.

[26] N. Fornara and M. Colombetti, Representation and monitoring of commitments and norms using OWL, *AI Commun.* **23**(4) (2010), 341–356.

[27] S. Kirrane, S. Villata and M. d'Aquin, Privacy, security and policies: A review of problems and solutions with semantic web technologies, *Semantic Web* **9**(2) (2018), 153–161. doi:10.3233/SW-180289.

[28] N. Fornara and M. Colombetti, Operational Semantics of an Extension of ODRL Able to Express Obligations, in: *Multi-Agent Systems and Agreement Technologies - 15th European Conference, EUMAS 2017, and 5th International Conference, AT 2017, Évry, France, December 14-15, 2017, Revised Selected Papers*, F. Belardinelli and E. Argente, eds, Lecture Notes in Computer Science, Vol. 10767, Springer, 2018, pp. 172–186. ISBN 978-3-030-01713-2. doi:10.1007/978-3-030-01713-2_13.

[29] N. Fornara, A. Chiappa and M. Colombetti, Using Semantic Web Technologies and Production Rules for Reasoning on Obligations and Permissions, in: *Agreement Technologies*, M. Lujak, ed., Springer International Publishing, Cham, 2019, pp. 49–63. ISBN 978-3-030-17294-7.

[30] J.M. Bradshaw, A. Uszok, M. Breedy, L. Bunch, T.C. Eskridge, P.J. Feltovich, M. Johnson, J. Lott and M. Vignati, The KAoS Policy Services Framework, 2013.