Using Semantic Web Technologies and Production Rules for Reasoning on Obligations and Permissions ⋆,⋆⋆

Nicoletta Fornara(✉)[1], Alessia Chiappa[2], Marco Colombetti[2]

[1] Università della Svizzera italiana,
via G. Buffi 13, 6900 Lugano, Switzerland
nicoletta.fornara@usi.ch, ORCID ID: 0000-0003-1692-880X
[2] Politecnico di Milano,
piazza Leonardo Da Vinci 32, Milano, Italy
alessia.chiappa@gmail.com,
marco.colombetti@polimi.it, ORCID ID: 0000-0003-2339-9678

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 permission and obligation 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 and permissions (expiration dates and deadlines) as part of their application independent model. The operational semantics of the proposed model of obligations and permissions is specified using Discrete State Machines and is computed 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 happen in the interaction among the parties involved in the policies.

1 Introduction

Nowadays the study of policies and norms is crucial in different fields of research and applications. Policies may be used for regulating access to data and digital assets in policy-based access control frameworks. They may be used to

⋆ Funded by the SNSF (Swiss National Science Foundation) grant no. 200021_175759/1.
⋆⋆ The final publication is available at Springer via https://doi.org/10.1007/978-3-030-17294-7_4.

unambiguously specify licenses for 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.

In principle policies and norms can be specified using human-readable formats; however, it is crucial to specify them with formal and machine-readable languages in order to enable machine-to-machine interactions combined with a number of useful services, like: (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) checking the satisfaction or violation of the normative or legal relations that an intensive exchange of digital assets creates in the chain of interactions among data producers, data publishers, and data consumers.

In order to perform services of this type it is crucial not only to propose a language for expressing policies, but also to unambiguously specify the meaning of such policies. This with the goal of being able to automatically monitor the fulfilment or violation of obligations and the correct use of permissions, and to simulate what would happen if one of the parties, related by a set of policies, performs certain actions.

ODRL 2.2 (Open Digital Right Language)[3] 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) [12], 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 15^{th} February 2018. In all the specifications of ODRL, its semantics is described informally in English, and no formal specification is provided. In [13] an OWL representation of ODRL 1.1 is presented, but the use of OWL is limited to the representation of classes and properties, and no representation is given of the dynamic semantics of policies, that is, of how they evolve in time. In [19] the semantics of ODRL 2.1 policies used for access control is investigated. When a request to perform an action on an asset is issued, the system evaluates which rules (prohibition, permission, or duty rules) are triggered (taking into account explicit and implicit dependencies among regulated actions), then it checks whether these rules hold based on certain constraints (i.e., activation conditions); however there is no hint on how the satisfaction of constraints can be computed.

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

In this paper, we propose to extend the model of permission 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. The operational semantics of the proposed model of obligations and permissions is then specified using Discrete State Machines, which are used to unambiguously specify the temporal evolution of the *deontic state* of obligations and permissions 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[4] (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 permissions and obligations is introduced. In Section 3, the life cycles of those two deontic relations is formally specified. In Section 4, a production rule system is used for computing the deontic state of obligations and permissions. 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 and Permissions

Following ODRL 2.2 Information Model, a policy must have at least one permission, prohibition or obligation. 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 "print less than or equal to 1200 dpi resolution". A permission may be conditioned by a duty and its intuitive meaning is that the duty represents a pre-condition that must be fulfilled to obtain a valid permission. From our perspective, it sounds quite unnatural to say that for acquiring a valid permission, for example to listen to an audio file, I have the duty to do something, for example to pay x euro. This because a duty is an action than an agent is obligated to do, not an action that an agent can freely decide to perform. In ODRL model, an obligation is a duty and it is fulfilled if all constraints are satisfied and if the regulated action, with all constraints satisfied, has been exercised.

The ODRL model does not highlight two crucial application-independent characteristics of the modelled deontic concepts. The first one is the important

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

role played by the event/action that can *activate* an obligation or make a permission *valid*. In ODRL, an activation condition is represented as a generic constraint, even if it is a crucial part of the deontic model. For example, only when an agent enters a limited traffic area he becomes obligated to pay an amount of money within a given interval of time; similarly, only after paying a fee an agent gets a valid permission to play a music file in a party. It is also important to model those actions and events using complex constructs having a type and application-independent properties.

The second relevant application-independent characteristic of the modelled deontic concepts is their relation with time. Usually an obligatory action has to be performed before a specific *deadline*, and a permission can be used within a certain interval of time (e.g., the obligation to pay 5 euro before the end of the month, or the permission to play a music file within one week). A conditional obligation/permission may also become *expired* if it is not activated or made valid before a given expiration date. For example, as long as an agent is a bidder in an auction, such an agent has the obligation to pay his bids if it becomes the winner of the auction. When the auction is closed, the obligation expires and cannot become active anymore.

In this paper, we propose to extend ODRL 2.2 information model with two new types of deontic relations: *conditional obligations* and *conditional permissions*, i.e. obligations and permissions that become activated/valid when a condition is satisfied and having in their meta-model expiration dates and deadlines. What we model is a notion of *strong* permission, i.e. the explicit permission to do an otherwise prohibited action, which is different from the *weak* permission, i.e. the absence of the prohibition to do an action [22].

Given that ODRL is a W3C standard, the ODRL 2.2 ontology[5] is formalized using RDF Schema[6], 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 meta-model of temporal-conditional obligations and permissions using an OWL ontology: the *Normative Language Ontology* (NL Ontology). This choice involves also the advantage of being able to perform automatic reasoning on the OWL formalization of the proposed deontic concepts.

In the definition of the *Normative Language Ontology* we exploit the possibility, given by the adoption of Semantic Web Languages, to connect our ontology with other, quite well known, ontologies. We re-use the core model of temporal entities specified in the *OWL Time Ontology*[7] for being able to specify deadline and expiration dates, and the time when real events or actions happen. We re-

[5] https://www.w3.org/ns/odrl/2/
[6] https://www.w3.org/TR/rdf-schema/
[7] W3C Recommendation 19 October 2017 https://www.w3.org/TR/owl-time/

use the *Schema.org* ontology[8] a well-known ontology that has been developed to support web search engines. We re-use it for the specification of actions as complex objects, contrary to their treatment in ODRL, where they are represented by atomic symbols. We extend the ODRL 2.2 ontology in various ways, as will be detailed in the sequel. Finally, we re-use the *Event Ontology* presented in [5] for expressing events as a super-class of actions and for connecting events to time instants and intervals. The import relationship among the various ontologies is depicted in Figure 1

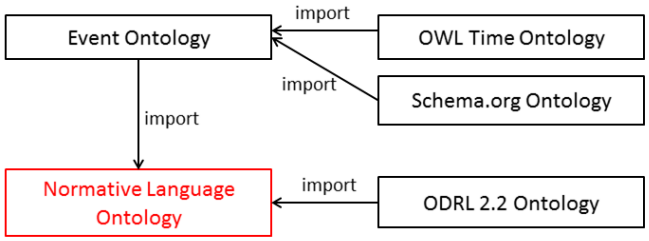

Fig. 1. The import relationship among the various ontologies

The *Normative Language Ontology* is depicted in Figure 2. It defines two new types of deontic relations, as subclass of the ODRL Rule class: the nl:Obligation and the nl:Permission classes.

Obligations and permissions have the following common characteristics. They are *deontic relations* between two *parties*. They are characterized by two fundamental components: the *activation condition* and the *content*. The activation condition describes an event or an action, the content describes an action. When the activation event/action of an obligation actually happens, the obligation to perform the action described in the content component becomes active. Differently, when the activation event/action of a permission actually happens, the permission to perform the action described in the content component becomes valid. A *counter* is used for managing obligations and permissions to perform an action more than once. Obligations and permission have an *expiration* date and a *deadline*. Expiration and deadline usually refer to different instants of time. The expiration is the instant of time when the 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 permitted to exercise a valid permission. Deadlines and expiration dates may be computed at run-time by using the specified interval of time, in those cases their value depends on when the obligation/permission is created or activated/made valid. Finally, obligation and permission have a *deontic state*, it is used to compute their life cycle as discussed in next section.

An example of a conditional permission may be: when `person:bob:01` pays `5` euros to `organization:SOYN`, he obtains a valid permission to listen to, at most

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

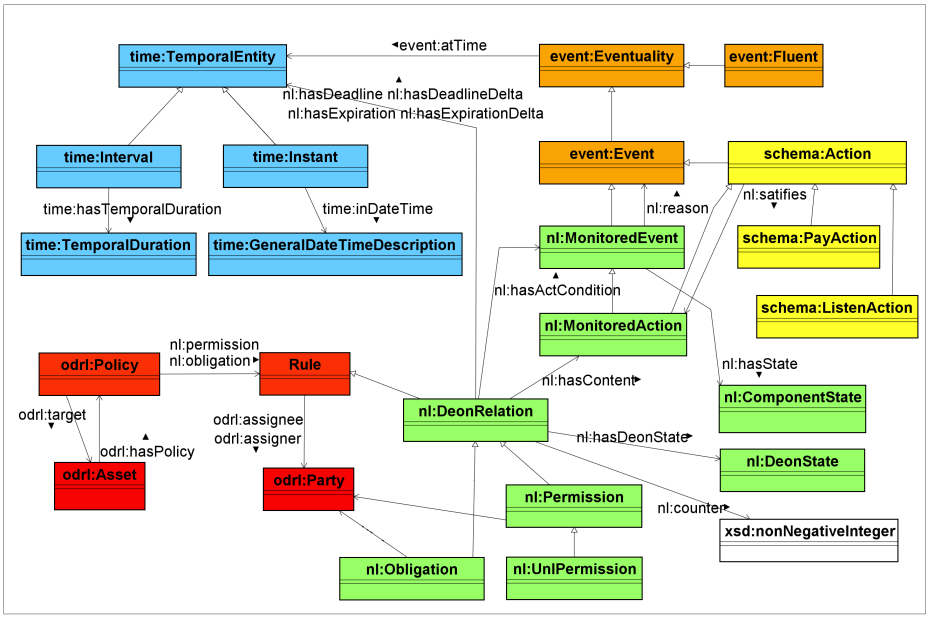

Fig. 2. The OWL Normative Language Ontology and its connections with other ontologies

ten times, a music record by the Beatles within 48 hours from the payment; this permission expires at the end of 2018. Its formalization using the proposed ontology serialized using the Turtle language[9] is as follows:

```
ex:policy:01 a odrl:Policy;
   nl:permission    ex:perm:policy:01:1.
ex:perm:policy:01:1  a   nl:Permission;
   odrl:assigner    ex:org:SOYN;
   odrl:assignee    ex:person:bob:01;
   nl:hasActCond    ex:actCond:1;
   nl:hasContent    ex:content:1;
   nl:hasDeadlineDelta [time:hasTemporalDuration "PT48H0M0S"^^xsd:duration];
   nl:hasExpiration    [time:inDateTime "2018-12-31T09:00:00Z"^^xsd:dateTime];
   nl:counter       10^^xsd:integer.
ex:actCond:01:1   a  schema:PayAction;
   schema:agent     ex:person:bob:01;
   schema:recipient ex:org:SOYN;
   schema:price     5.00;
   schema:priceCurrency "euro".
```

[9] https://www.w3.org/TR/2014/REC-turtle-20140225/. In Turtle every row is an RDF triple statement (subject, predicate, object) terminated by '.'; a ';' symbol is used to repeat the subject of triples that vary only in the predicate and object parts.

```
ex:content:01:1   a  schema:ListenAction;
   schema:agent     ex:person:bob:01;
   schema:object [  a schema:MusicRecording;
                    schema:byArtist ex:Beatles].
```

This is an example of a *policy instance*, with all its properties filled with a specific value. 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 contain variables and are transformed into policy instances through a procedure of substitution of variables with actual values, specified during an interaction with a specific user.

3 Life cycles of Obligation and Permission

Obligations and *permissions* are two fundamental deontic relations widely used for regulating the *actions* that various individuals should or may perform. *Actions*, and more generally *events* (i.e. something that happens in a system, but is not necessarily done by an actor) change the state of the interaction among various parties, and their effects are strictly related to the instant of time when they happen. In order to specify what it means for an agent to have an obligation or a permission, it is fundamental to model their *deontic state* and to formally specify its evolution in time based on actual events (e.g. the elapsing of a deadline) and actions (e.g. downloading a music file).

In this section, we propose to formally describe such a temporal evolution using two *life cycles*, one for the notion of obligation and one for the notion of permission. They are the result of a deep analysis of the literature and of the textual description of permissions and duties given in the ODRL Information Model 2.2. Those life cycles are unambiguously specified using Discrete State Machines. More precisely, we use two simple types of *Discrete State Machines* (DSMs): pure *Finite State Machines* for modelling the conditional permission to perform an action for an unlimited number of times, and *Finite State Machines* augmented with a *decreasing counter* (with 0 lower bound), for modelling conditional obligations and permissions to perform an action for a limited number of times.

It is important to notice that the evolution of the *deontic state* of obligations and permissions in turn depends on the satisfaction of their *activation condition* and *content*. It is necessary to define a *procedure* for checking if actual events or actions satisfy the description of events and actions that appear in obligations and permissions. One possible approach for the realization of such a procedure will be described and discussed in next section.

In order to represent in our model the satisfaction of the content or of the activation condition of a deontic object, we introduce in the *NL Ontology* the following two properties. The hasState property is used to connect the content and the activation condition of a deontic object with their *state*. Such a state is initially *unsatisfied* and becomes *satisfied* when the described event/action occurs. The reason property is used to connect the content and the activation condition

of a deontic object with the real event or action that produced its satisfaction. This property is necessary for comparing expiration dates and deadlines of the deontic objects with the instant of time when events or actions actually happen.

The life cycle of conditional obligations is depicted in Figure 3. When a conditional obligation is created, it is in the *conditional* state and its condition and content are *unsatisfied*. When the activation condition becomes *satisfied* and the current time is less than or equal to the expiration date of the deontic relation, the transition from the conditional state to the *activated* state is fired. Differently if the expiration date is elapsed the transition leads from the conditional to the *expired* state. Then, whenever the content becomes satisfied before the deadline and its counter is greater than zero the conditional obligation remains *activated*, the counter is decremented, and the content is set back to *unsatisfied*. If the content becomes again satisfied before the deadline and the counter is equal to zero the state becomes *fulfilled*. Differently if the state is still activated and the deadline becomes elapsed, the state becomes *violated*.

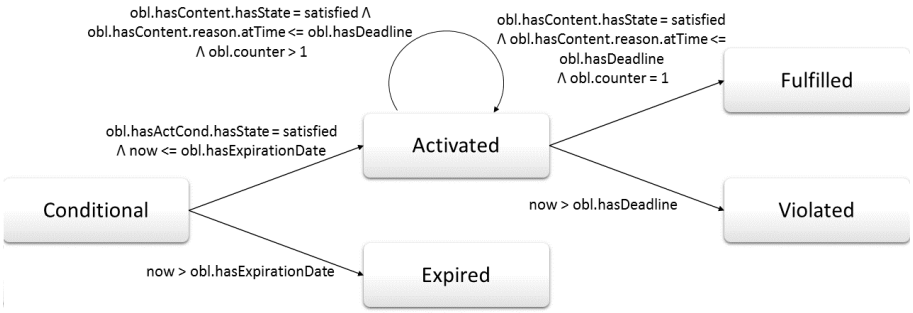

Fig. 3. Life cycle of Conditional Obligation

The life cycle of a conditional permission is depicted in Figure 4. A *conditional* permission becomes *valid* when its condition is satisfied. Then every time it is exercised, a counter is decremented and when the counter is equal to zero the state becomes *exercised*. If the expiration date of a conditional permission is expired its state becomes *expiredConditional*, differently if the deadline of a valid permission is expired the permission becomes *expiredValid*. The conditional permission to perform an action for an unlimited number of times is represented with a Finite State Machine and the main difference is the absence of the counter and of the exercised state, in fact only when the deadline is elapsed the permission becomes expiredValid.

It is interesting to observe that the conditions that trigger the transitions of the two life cycles are identical. The fundamental difference between the two deontic relations, 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* final state has no positive or negative connotation.

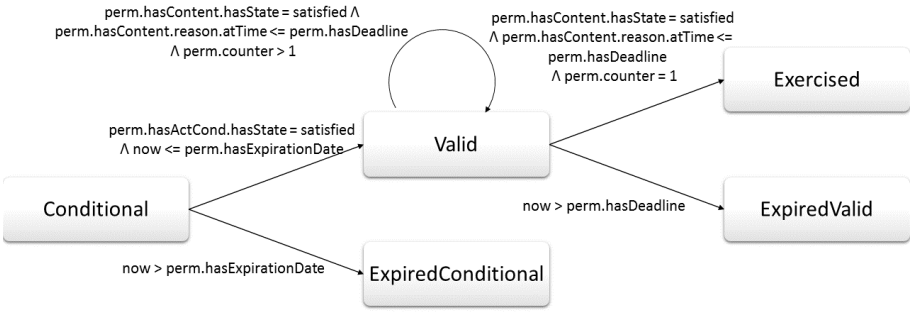

Fig. 4. Life cycle of Conditional Permission

4 Operational Semantics of Obligation and Permission

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*. Such a deontic state, in turn, depends on the satisfaction of the *state* of the *activation condition* and *content* of the relevant deontic objects.

This procedure requires an application-dependent component, able to "sense" the actual events or actions and represent them as individuals of the *State Ontology* by using the vocabulary defined by the *Event* and *Schema.org* ontologies. This procedure also requires two application-independent components: one for computing the satisfaction of the *state* and one for computing the *deontic state*.

In this section we propose to realize those components using a *production system* [1], i.e. a forward-chaining reasoning system that uses *production rules*. The *ongoing memory* of assertions (stored in the working memory of the proposed production system) is composed of: the *State Ontology*, a representation, in the form of RDF triples, of the state of the interaction in terms of actual actions, events, and current time; the *Normative Ontology*, a representation of the set of policies containing obligations and permissions. 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 production rules.

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 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 and permissions introduced in the previous section. A crucial advantage of 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.

In this section we will formalize our production rules using the Abstract Syntax of the W3C Recommendation RIF Production Rule Dialect[10] (RIF PRD). Initially, we present the application independent productions rules for computing the satisfaction of *activation conditions* and *content*. Subsequently, we present the productions rules for computing the *deontic state* of obligations and permissions.

The first type of production rules are used for matching the description of events and actions with actual events or actions as soon as they are represented in the system. In the meta-model of obligations and permissions proposed in this paper, a description of an event or an action is characterized by the specification of its class and of a list of values for significant properties. The matching can be realized by a production rule used for checking the exact match between the described values and the real values. Given that the list of properties used for describing an event or an action can vary, we need to specify a production rule for every monitored type of event or action (characterized by the list of properties used for its description). In the future we plan to study a more flexible mechanism for matching an actual event/action with an event/action description.

Due to space limitation we report only one production rule of this type. It can be used for matching a real payment with the activation condition of the permission presented in Section 2 and with all the activation conditions where the same parameters are used[11].

```
(* PayAction *)
Forall  ?realAction ?component ?agent ?recipient ?price ?currency(
  If And( rdf:type(?perm nl:Permission)
    nl:hasDeonState(?perm nl:conditional)
    nl:hasActCond(?perm ?activation)
    nl:hasState(?activation nl:unSatisfied)
    rdf:type(?activation schema:PayAction)
    schema:agent(?activation ?agent)
    schema:recipient(?activation ?recipient)
    schema:price(?activation ?price)
    schema:priceCurrency(?activation ?currency)
    rdf:type(?realAction schema:PayAction)
    schema:agent(?realAction ?agent)
```

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

[11] We assume that it is impossible to insert in the State Ontology an event that will happen in the future.

```
    schema:recipient(?realAction ?recipient)
    schema:price(?realAction ?price)
    schema:priceCurrency(?realAction ?currency) )
  Then (Assert(nl:reason(?activation ?realAction))
        Retract(nl:hasState(?activation nl:unSatisfied))
        Assert( nl:hasState(?activation nl:satisfied))  )  )
```

The production rules of the second type are used for computing the *deontic state* of obligations and permissions. 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 (permission or obligation), the 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 present only the production rule used for computing the transition from *conditional* to *valid* of a permission.

```
(* validatePermission *)
Forall ?perm ?activation ?expiration ?expirationDateTime ?now (
  If And( rdf:type(?perm nl:Permission)
    nl:hasDeonState(?perm nl:conditional)
    nl:hasActCond(?perm ?activation)
      nl:hasState(?activation nl:satisfied)
    nl:hasExpiration(?perm ?expiration)
      time:inDateTime(?expiration ?expirationDateTime)
      time:inDateTime(ex:currentTime ?now)
    External(pred:numeric-less-than(?now ?expirationDateTime) ) )
  Then (Retract(nl:hasDeonState(?perm nl:conditional))
        Assert(nl:hasDeonState(?perm nl:valid)) )  )
```

5 Implementation of a Prototype

For testing our proposal, we have developed a Java prototype, specifically a system able to *simulate* the evolution of policies containing obligations and permissions. We use Apache Jena[12], 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-based reasoner and a translation of the RIF PRD rules into Jena Rules[13]. This reasoner supports rule-based inference over RDF graphs, and provides forward chaining realized by means of an internal RETE-based forward chaining interpreter[4]. 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 [16].

[12] `https://jena.apache.org/`

[13] `https://jena.apache.org/documentation/inference/\#rules`

An interesting feature of the Jena forward chaining interpreter is that it works incrementally, meaning that if the inference model is modified by adding or removing statements, the 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 Finite 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.

A useful service is the *monitoring* of deontic relations for those applications where it is crucial to check the fulfilment or violation of norms and the use of valid 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 a permission or obligation. 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[14] 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 and permissions eventually evolve.

6 Related Work

Studies on Normative Multiagent Systems (NorMAS) concern mainly the proposals of formalisms for expressing norms or policies containing obligations, permissions, and prohibitions. Those studies also investigate the realization of fundamental functionalities for norm promulgation, monitoring, and enforcement, as well as norm adoption and reasoning. In the NorMAS literature there are various proposals for the formalization of norms and policies using different languages [3, 5] and different frameworks [18, 2] for their management.

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

As we already discussed, the W3C standard for expressing policies is ODRL 2.2. Another interesting proposal, which is in the process of becoming an OASIS standard in the legal domain, is the LegalRuleML language[15].

Many approaches to the formalization of norms are based on different logics, which are declarative in nature. The most well-known are the studies on Deontic Logic [21], a family of logical systems where the essential features of obligations, prohibitions and related concepts are captured. An interesting approach to the specification of the semantics of obligations, permissions, and prohibitions is given in [11], where the L4LOD vocabulary for expressing licenses for Linked Open Data is presented. The semantics of the deontic component of licenses is formalized using an extension of Defeasible Logic [10]. This extension is a non-monotonic logic able to deal with permissions as defeaters of prohibitions (understood as negative obligations). The actions that are regulated are those typical of linked open data licences (e.g., ShareAlike, Attribution, etc.). Such actions are represented as atomic symbols, and no treatment of time or relevant action attributes is proposed. Another interesting approach, where time is taken into account, is based on Linear Temporal Logic (LTL) [17]. This paper proposes a life cycle for obligations where deadlines and expiration dates are not modelled, and the content of the obligation is a maintenance condition, like for example "do not cross on a red light". Similarly to our proposal the transition rules are computed using a production system. In [9], 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[16]. Differently in this paper, we propose a production rule system for computing the application-independent life cycle of policies containing deontic relations.

In this paper, we propose to formalize policies using Semantic Web Technologies; therefore here we will mainly discuss other approaches where those technologies were adopted. In particular, an interesting literature review of various approaches to policies specification using Semantic Web Technologies is given in [14]. [7, 5] presents a proposal to specify and reason on obligations using OWL 2, SWRL rules, 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 are fulfilled of violated on the basis of the actions performed by the agents) is realized by means of a specific framework used for managing the elapsing of time and for performing closed-world reasoning on certain classes. Unfortunately, the scalability of this approach is not good enough to make it usable in real applications.

An interesting approach that uses Semantic Web Technologies for policy formalization and management is the OWL-POLAR framework [18]. This framework investigates the possibility of using OWL ontologies for representing the

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

[16] `http://www.jessrules.com/`

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.

Another relevant proposal is the KAoS policy management framework [20, 2]. 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 [15] social commitments [6] 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 [8] a proposal 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. In this paper we improved that work by proposing an extension of the new version of ODRL (ODRL 2.2), by formalizing the life cycle of both condition permissions and obligations which regulate the performance of an action for a limited number of times, and by expressing the operational semantics of those deontic concepts using a production system.

In our future work, we plan to investigate the dynamic connections between obligations, permissions and prohibitions. We plan also to study how to efficiently integrate OWL reasoning into the proposed production system, and to further investigate the possibility to use the event of violation or fulfilment of an obligation for applying rewards or sanctions.

References

1. R. Brachman and H. Levesque. *Knowledge Representation and Reasoning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.
2. 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. Eighth Cyber Security and Information Intelligence Research Workshop (CSIIRW 2013). Oak Ridge, TN: Oak Ridge National Labs, 2013.
3. K. da Silva Figueiredo, V. Torres da Silva, and C. de Oliveira Braga. *Modeling Norms in Multi-agent Systems with NormML*, pages 39–57. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
4. C. L. Forgy. *On the Efficient Implementation of Production Systems.* PhD thesis, Pittsburgh, PA, USA, 1979. AAI7919143.
5. N. Fornara. Specifying and Monitoring Obligations in Open Multiagent Systems using Semantic Web Technology. In *Semantic Agent Systems: Foundations and*

Applications, volume 344 of *Studies in Computational Intelligence*, chapter 2, pages 25–46. Springer-Verlag, 2011.
6. 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, pages 536–542, New York, NY, USA, 2002. ACM.
7. N. Fornara and M. Colombetti. Representation and monitoring of commitments and norms using OWL. *AI Commun.*, 23(4):341–356, 2010.
8. N. Fornara and M. Colombetti. Operational semantics of an extension of ODRL able to express obligations. In F. Belardinelli and E. Argente, editors, *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*, volume 10767 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 2017.
9. 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, pages 667–673, New York, NY, USA, 2005. ACM.
10. G. Governatori and A. Rotolo. BIO logical agents: Norms, beliefs, intentions in defeasible logic. *Autonomous Agents and Multi-Agent Systems*, 17(1):36–69, 2008.
11. 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 H. Alani and L. K. et al., editors, *ISWC 2013, Sydney, Australia, October 21-25, 2013, Proceedings, Part I*, volume 8218 of *Lecture Notes in Computer Science*, pages 151–166. Springer, 2013.
12. R. Iannella, S. Guth, D. Paehler, and A. Kasten. ODRL Version 2.1 Core Model. https://www.w3.org/community/odrl/model/2.1/ (accessed 15/09/2017), 2015.
13. A. Kasten and R. Grimm. Making the semantics of odrl and urm explicit using web ontologies. In *Virtual Goods*, pages 77–91, 2010.
14. 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):153–161, 2018.
15. 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):2724–2737, Oct. 2016.
16. J. Moskal and C. J. Matheus. Detection of Suspicious Activity Using Different Rule Engines - Comparison of BaseVISor, Jena and Jess Rule Engines. In N.Bassiliades, G. Governatori, and A. Paschke, editors, *RuleML 2008, Orlando, FL, USA, October 30-31, 2008. Proceedings*, volume 5321 of *LNCS*, pages 73–80. Springer, 2008.
17. 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*, pages 346–363, New York, NY, USA, 2014. Springer-Verlag New York, Inc.
18. 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.*, 12:148–160, 2012.
19. S. Steyskal and A. Polleres. Towards Formal Semantics for ODRL Policies. In N. Bassiliades, G. Gottlob, F. Sadri, A. Paschke, and D. Roman, editors, *RuleML 2015, Berlin, Germany, August 2-5, 2015, Proceedings*, volume 9202 of *Lecture Notes in Computer Science*, pages 360–375. Springer, 2015.

20. 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*, pages 145–152. IEEE Computer Society, 2008.
21. G. H. von Wright. Deontic logic. *Mind, New Series*, 60(237):1–15, 1951.
22. G. H. von Wright. *Norm and Action: A Logical Enquiry*. Routledge and Kegan Paul, 1963.