An Architecture for Monitoring Norms that combines OWL Reasoning and Forward Chaining over Rules

Nicoletta Fornara[1], Marco Sterpetti[2]

[1]*Università della Svizzera italiana, via G. Buffi 13, 6900 Lugano, Switzerland*
[2]*Politecnico di Milano, piazza Leonardo Da Vinci 32, Milano, Italy*

Abstract
In this paper we presents the architecture of a system able to monitor the fulfillment or violation of a set of norms (formalized with the T-NORM model) that regulates the performance of a class of actions that should or should not be performed in a temporal interval by autonomous agents. The proposed architecture is innovative because it combines an OWL reasoner with a forward chaining interpreter and its implementation requires to solve various problems of interoperability among them.

Keywords
Norm, Policy, Ontology, OWL, Production Rule, Time

1. Introduction

In NormMAS literature [1] it is widely accepted that norms (or policies[1]) models are characterized by: an *activation condition*, which specifies when norms become active, and by a *regulated action* (or target condition), which describes what is prohibited or obliged by the norms [2, 3, 4, 5]. Out of these models, the last three describe the activation condition and the obliged or forbidden action using semantic web technologies. In particular, the OWL-POLAR model [3] and the T-NORM [5] model (which is the model of norm adopted for this demonstration paper) formalize the conceptual model for describing actions, events, and state of affairs, used in the specification of activation conditions and regulated actions, using conjunction of atomic assertions expressed using the classes, properties, and individuals defined in OWL ontologies.

The semantics of the notion of *activation condition* is as follows: when the condition is satisfied by the world state, the norm becomes active or in force. This relevant change in the state of the norm is represented in different ways in the different models: the norm is instantiated in [2] by creating norm instances; the policy is activated for a specific agent in [3] but its activation is not explicitly represented in the data; the norm changes its deontic state from conditional to active or inForce in [4]; and a new specific or general deontic relation is created in the T-NORM model [5].

FOIS 2021 Demonstrations, held at FOIS 2021 - 12th International Conference on Formal Ontology in Information Systems, September 13-17, 2021, Bolzano, Italy
nicoletta.fornara@usi.ch (N. Fornara)
https://people.lu.usi.ch/fornaran/ (N. Fornara)
0000-0003-1692-880X (N. Fornara)

[1]In this paper we use the term norm as synonymous of policy.

The meaning correlated to the description of the *action regulated by a norm* is as follows: when an action that matches with the described one is performed the norm becomes fulfilled (if it is an obligation) or violated (if it is a prohibition). In the models described in [4, 5], it is also possible to specify what happens if an action that matches with the regulated one cannot be performed anymore (for example when the deadline for performing the action is elapsed). In this case the norm becomes violated (if it is an obligation) or fulfilled (if it is a prohibition).

Another well known language for expressing policies by taking advantage of semantic web technologies is the Open Digital Rights Language (ODRL 2.2)[2]. ODRL is a policy expression language, it is a W3C Recommendation since 15 February 2018. In ODRL the Information Model is formalized as an OWL ontology[3]. The ODRL OWL ontology is used as a metamodel for the specification of the policies, it is not used for reasoning on policies components nor it is used for reasoning on the fulfillment or violation of policies. Differently from the approaches discussed before, in the ODRL model there is not a clear distinction between an activation condition and the regulated action. By using the ODRL language it is possible to express constraints on the performance of the actions regulated by a policy (for example it is obligatory to pay before the turn of the New Year in 2019), but it is impossible to specify the effects that the satisfaction of certain constraints have on the policy. Moreover, in ODRL the regulated actions are not described using the classes, properties and individuals defined in an ontology of actions, but they are specified using individuals belonging to the ODRL:Action class and constraints on those individuals. This is due to the fact that in ODRL a policy is an instance of the ODRL:Policy class or of one of its subclasses. In [4] a proposal to extend ODRL 2.2 with an activation condition is proposed. In such a paper the activation conditions and the regulated actions are anonymous individuals belonging to a subclass of the ODRL:Action class and monitoring is realized by matching such an individual in the policy with a real action happening in the world state. In [6] a profile of the ODRL 2.2 language is presented and the semantics of ODRL policies is given by translating them into Answer Set Programming.

In the T-NORM model of norms [5] and in OWL-POLAR [3], one specific norm is not an instance of an OWL class, but it is a complex object made by different components. In particular, in OWL-POLAR the different components of a norm may be represented in a table and the activation of policies and their violation is computed with an ad-hoc software and by executing SPARQL-DL queries. In the T-NORM model the norm designer may specify norms by using basic building blocks that consist of rules of the form `IF...THEN...ELSE` as presented in Section 3. From the operational prospective, an interesting proposal for computing the activation and fulfillment or violation of norms consists in translating every norm into a set of production rules and then, by using a forward engine, computing their execution. This approach has the advantage that the norm designer will never need to change the software that manages the effect of events on norms' state, but they can simply change the declarative form of norms. The architecture of a system, able to monitor (or to simulate) the evolution over time of the state of a set of norms expressed using the T-NORM model, needs to combine an OWL reasoner and a forward chaining reasoner or interpreter [7]. The combination of those two different types of reasoners is innovative and presents some interesting problems that we solved for testing

[2]https://www.w3.org/TR/odrl-model/
[3]https://www.w3.org/ns/odrl/2/

our proposal and that we will discuss in this paper by using as running example a norm in the Covid-19 domain.

This paper is organized as follows. In Section 2 the OWL ontologies required for formalizing and monitoring norms are presented. In Section 3 the T-NORM model and one example of norm is introduced. In Section 4 the architecture of the system for norms monitoring is described and its implementation is discussed.

2. OWL Ontologies for Modelling and Monitoring Norms

The goal of the proposed architecture is to realize norms monitoring. For doing that it is crucial to represent: (i) the world state; (ii) the activation condition and the actions regulated by the norms; (iii) a mechanism for matching the real actions performed by agents with the one regulated by norms. It is feasible to realize this matching when the model for representing the wold state and the one used for specifying the relevant actions and events described in the norms are the same or are at least compatible.

The world state consists of state of affairs, events, and actions, that are events performed by an actor. We propose to model the world state by using the *Event Ontology* in OWL. The advantages of using an OWL ontology are: first of all the possibility to use OWL reasoning for deducing new knowledge; secondly OWL reasoning has an effect on the normative position of the regulated actions. In fact, if the performance of an action a implies another one a' (for example when I sell a product I am also transferring the property of the product) we have that: (i) if action a' is obliged (transferring the property), the performance of action a (selling) will fulfill the obligation; (ii) if action a' is prohibited (transferring the property), the performance of action a (selling) will violate the prohibition. In order to make it easy to realize a matching between real actions and the regulated ones we propose to use the *Event Ontology* also for modelling events and actions descried in the norms. The *Event Ontology* imports the W3C Candidate Recommendation *Time Ontology* in OWL for connecting events to instants or intervals of time. To be used in the formalization of concrete examples, the *Event Ontology* needs to be extended with a domain-specific *Action Ontology* that defines a hierarchy of classes and a set of properties for modelling various types of actions like for instance paying, selling, testing, and so on.

The second ontology introduced in our system is the *T-NORM Ontology*, it is required for managing and representing the effects of the activation of norms and of their fulfillment or violation. The *T-NORM Ontology* imports the *Event Ontology* for representing events that are relevant for norms, for example the deadlines for obligations that are represented as individuals of the TimeEvent class. It is used for representing the activation of norms by creating *specific* or *general* deontic relations and for representing the fulfillment and violation of those deontic relations. The *T-NORM Ontology* is depicted in Figure 1 together with the *Event Ontology* and the *Time Ontology*[4].

[4]The T-NORM ontology is available at https://raw.githubusercontent.com/fornaran/T-Norm-Model/main/tnorm.owl

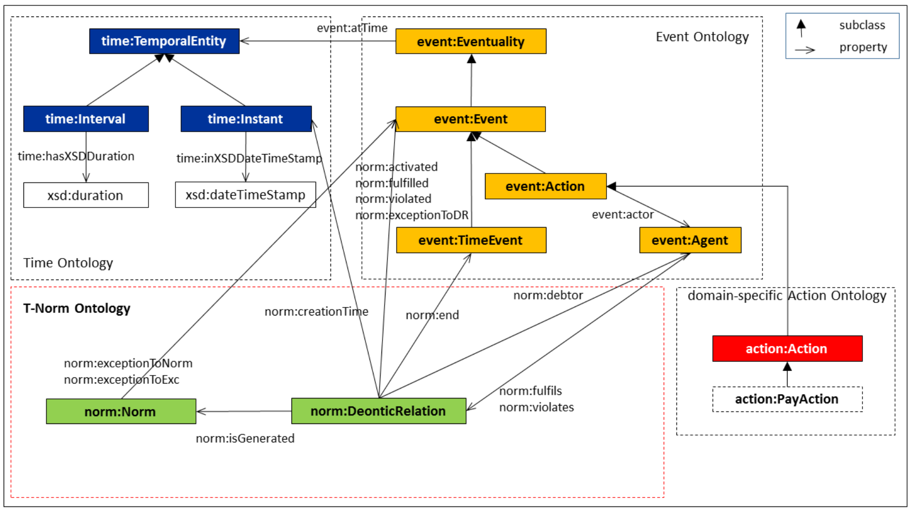

Figure 1: The T-NORM Ontology, the Event Ontology, and the Time Ontology.

3. The T-NORM Model and a Running Example

The T-NORM model is presented in [5]. By using this model the norm designer may specify norms by using basic building blocks that consist of rules of the form `IF...THEN...ELSE`. There is not a pre-defined type of norm, like obligation or prohibition, but it is the norm designer that has the instrument to specify that when an activation event happens and some contextual conditions are satisfied, some parameters must be computed (for example the deadline for obligations) and then the monitoring of the performance of an action belonging to a class of regulated actions has to start. The result of the monitoring process is the fulfillment or violation of a set of norms. In particular, if a specific action that belongs to the class of the regulated one is performed there is a fulfillment, when the action is obligated, or a violation, when the action is prohibited; otherwise, if a specific action cannot be performed anymore (for example because it must be performed before the deadline and the deadline has elapsed) there is a violation (this is the case of obligations) or a fulfillment (when the norm represents a prohibition). The abstract form of a norm is:

```
NORM Norm_n
[ON ?event1 WHERE conditions on ?event1
THEN
   COMPUTE]
   CREATE DeonticRelation(?dr);
   ASSERT isGenerated(?dr,Norm_n); [activated(?dr,?event1);]
   ON ?event2 [BEFORE ?event3 WHERE conditions on ?event3]
      WHERE actor(?event2,?agent) AND conditions on ?event2
   THEN ASSERT fulfills(?agent,?dr); fullfilled(?dr,?event2) |
               violates(?agent,?dr); violated(?dr,?event2)
  [ELSE ASSERT violates(?agent,?dr); violated(?dr,?event3) |
               fulfills(?agent,?dr); fulfilled(?dr,?event3)]
```

Where conditions are a conjunctions of atomic assertions over the *Event Ontology* with the use of variables[5]. In the ASSERT parts only the variables introduced in the previous parts can be used. One useful example of prohibition that can be represented with the T-NORM model is: *"a person who has a positive swab to Covid-19 cannot leave the house for the next 15 days"*. It is formalized in the following way:

```
NORM Norm01
ON ?e1 WHERE PositiveTest(?e1) AND affectedPerson(?e1,?agent) AND
             atTime(?e1,?inst1) AND inXSDDateTimeStamp(?inst1,?t1)
THEN
   COMPUTE ?tend=?t1.days+15
   CREATE DeonticRelation(?dr); TimeEvent(?tev_end); Instant(?inst_end);
   ASSERT isGenerated(?dr,Norm01); activated(?dr,?e1); atTime(?tev_end,?inst_end);
      inXSDDateTimeStamp(?inst_end,?tend); end(?dr,?tev_end); debtor(?dr,?agent);
   ON ?e2 BEFORE ?tev_end
      WHERE LeaveHouse(?e2) AND actor(?e2,?agent) AND from(?e2,?house) AND home(?agent,?house)
   THEN ASSERT violates(?agent,?dr); violated(?dr,?e2)
   ELSE ASSERT fulfills(?agent,?dr); fulfilled(?dr,?tev_end)
```

The operational semantics of the T-NORM model is specified by providing an unambiguous procedure for translating every norm into a set of production rules, i.e. a formalism that already has an operational semantics[6]. The translation of the IF...THEN part is straightforward, because the form of a production rule is: *IF conditions THEN actions.* Differently, the translation of the ELSE part requires to capture its meaning. That is, the ELSE branch is followed whenever an event that matches the description of event2 cannot happen anymore. For example in Norm01 the ELSE branch is followed when the deadline is elapsed[7].

4. Architecture for Norms Monitoring and its Implementation

The architecture of the system designed to monitor norms formalized with the T-NORM model is depicted in Figure 2. It combines an OWL reasoner with a forward chaining engine. The depicted process consists in the following steps: (1) the norms, in force at a given instant of time, are translated into productions rules[8]; (2) the state of the world is loaded into the working memory of the production system and the value of the now variable is set to the current time; (3) an OWL reasoner is executed on the working memory; (4) a forward chaining engine is executed on the result of the OWL reasoner by using as input the production rules resulting from step 1; (5) the effects of the production rules are stored in the working memory. Whenever the working memory is updated with an event (including a Time Event) the process restarts from point 2.

[5]The syntax of conditionsis equal to the Human Readable Syntax of the antecedent of SWRL rules https://www.w3.org/Submission/SWRL/#2.2

[6]The operational semantics of a production rule system is given in the W3C Recommendation of the RIF Production Rule Dialect10 w3.org/TR/rif-prd/#Operational_semantics_of_rules_and_rule_sets

[7]Assuming that there is complete knowledge of all the actions or events happened in a given instant of time

[8]The translation of Norm01 into the corresponding production rules is available at https://raw.githubusercontent.com/fornaran/T-Norm-Model/main/Rules-Covid19

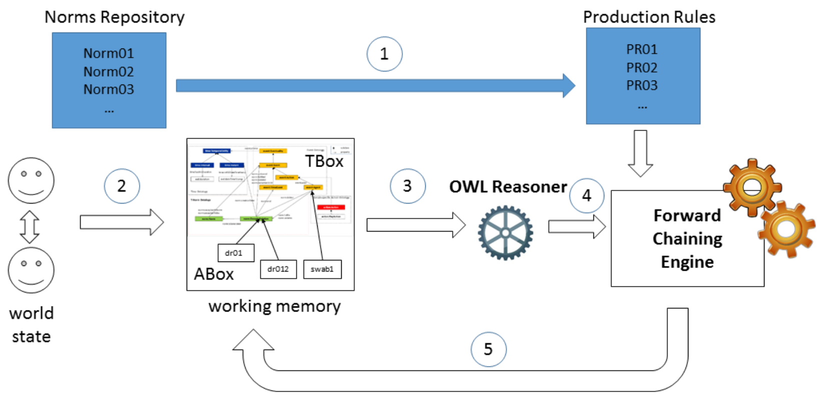

Figure 2: The architecture of the system for norms monitoring

An interesting research question would be whether it is possible to avoid combining two different types of reasoning. In [8] we studied the approach of using only an OWL reasoner for computing the activation and the fulfillment or violation of norms. Due to the open-world assumption of OWL reasoning, we faced the problem of deducing that when an action is not performed before a given deadline the related norm becomes violated. This problem was solved by computing the closure of certain OWL classes using a Java program.

4.1. Implementation and Testing

We tested the proposed architecture by using Pellet[9], as OWL 2 reasoner, and the JENA general purpose rule engine[10] as forward chaining engine. We decided to use JENA, instead of another production system, like Drools (which is used in [2]), because its rule engine natively supports rule-based computations over an OWL ontology serialized as an RDF graph. The main problem with Drools is due to the need of translating the OWL ontologies into Java classes for being able to execute the production rules on the state of the world. Another advantage of choosing the JENA forward engine is that it allows to combine RDFS/OWL reasoning with custom rules[11] by cascading reasoners. The main problem we had in the implementation and testing of this part is the lack of a clear documentation on how to realize this combination in Java. A very useful source of information is represented by the slides about the integration of Pellet and JENA presented in the ISWC2009 tutorial “Building Ontology-based Applications using Pellet”. The most critical part of the Java code required for combining Pellet with Jena is the following:

```
//Create an ontology model (OntModel) by reading the world state ontology
OntModel rawModel = ModelFactory.createOntologyModel(OntModelSpec.OWL_DL_MEM);
rawModel.read("file:" + modelSourceFile);
```

[9]http://pellet.owldl.com/
[10]https://jena.apache.org/documentation/inference/#rules
[11]https://jena.apache.org/documentation/inference/#RDFSPlusRules

```
//Create another ontology model using Pellet reasoner and the rawModel above
OntModel model=ModelFactory.createOntologyModel(PelletReasonerFactory.THE_SPEC,rawModel);

//Create a generic inferencing model (InfModel) using the ontology model above
//and the rules that represent the norms
GenericRuleReasoner reasoner = new GenericRuleReasoner(rules);
InfModel infModel = ModelFactory.createInfModel(reasoner, model);
```

Whenever the `rawModel` is updated or queried the Pellet reasoner is executed, this may introduce some overhead due to the queries to the OWL reasoner. One possible weakness of this approach is that it is “layered”, i.e. the outer `InfModel` can work on the results of the inner model but not viceversa. For our architecture this is not a problem because we do not need to perform OWL reasoning on the effects of production rules.

Given that the JENA forward chaining engine does not support the specification of a priority among rules, and we need it for the management of exceptions (for example for expressing permissions), we introduced into the rules the evaluation of the value of a variable called `salience`.

5. Acknowledgments

Funded by the SNSF grant no. 200021_175759/1. We acknowledge the contribution to the definition of the model by Marco Colombetti e Soheil Roshankish.

References

[1] S. Villata (Ed.), Special Issue: Normative Multi-Agent Systems, volume 5 of *Journal of Applied Logics - IfCoLog Journal*, College Publications, 2018.

[2] S. Álvarez-Napagao, H. Aldewereld, J. Vázquez-Salceda, F. Dignum, Normative monitoring: Semantics and implementation, in: M. D. Vos, et al. (Eds.), COIN@AAMAS 2010, COIN@MALLOW 2010, Revised Selected Papers, volume 6541 of *LNCS*, Springer, 2010, pp. 321–336.

[3] M. Sensoy, T. J. Norman, W. W. Vasconcelos, K. P. Sycara, OWL-POLAR: A framework for semantic policy representation and reasoning, J. Web Sem. 12 (2012) 148–160.

[4] N. Fornara, M. Colombetti, Using semantic web technologies and production rules for reasoning on obligations, permissions, and prohibitions, AI Commun. 32 (2019) 319–334. URL: https://doi.org/10.3233/AIC-190617. doi:10.3233/AIC-190617.

[5] N. Fornara, S. Roshankish, M. Colombetti, A framework for automatic monitoring of norms that regulate time constrained actions, in: Proc. of the COINE 2021 co-located with AAMAS 2021, 3rd May 2021, London, UK., 2021.

[6] M. De Vos, S. Kirrane, J. Padget, K. Satoh, Odrl policy modelling and compliance checking, in: P. Fodor, M. Montali, D. Calvanese, D. Roman (Eds.), Rules and Reasoning, Springer International Publishing, Cham, 2019, pp. 36–51.

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

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