Ontology and Time Evolution of Obligations and Prohibitions Using Semantic Web Technology

Nicoletta Fornara[1] and Marco Colombetti[1,2]

[1] Università della Svizzera italiana, via G. Buffi 13, 6900 Lugano, Switzerland
{nicoletta.fornara,marco.colombetti}@usi.ch
[2] Politecnico di Milano, piazza Leonardo Da Vinci 32, Milano, Italy
marco.colombetti@polimi.it

Abstract. The specification and monitoring of conditional obligations and prohibitions with starting points and deadlines is a crucial aspect in the design of open interaction systems. In this paper we regard such obligations and prohibitions as cases of social commitment, and propose to model them in OWL, the logical language recommended by the W3C for Semantic Web applications. In particular we propose an application-independent ontology of the notions of social commitment, temporal proposition, event, agent, role and norms that can be used in the specification of any open interaction system. We then delineate a hybrid solution that uses the OWL ontology, SWRL rules, and a Java program to dynamically monitor or simulate the temporal evolution of social commitments, due to the elapsing of time and to the actions performed by the agents interacting within the system.

1 Introduction

The specification of *open interaction systems*, where heterogeneous, autonomous, and self-interested agents can interact by entering and leaving dynamically the system, is widely recognized to be a crucial issue in the development of distributed applications on the Internet, like e-commerce applications, or collaborative applications for the automatic creation of virtual organizations. An important aspect of the specification of open systems is the possibility to define the actions that agents should or should not perform in a given interval of time, that is, the possibility to define social commitments with starting points and deadlines, and to monitor and react to their fulfilment or violation.

As we discussed in our previous works [11,12,10] in our OCeAN meta-model for the specification of artificial institutions, commitments for the interacting agents can be created by the activation of norms associated to the agents' roles, or by the performance of agent communicative acts, like promises. In this paper we explore how to use OWL (in its OWL 2 DL version[1]), the logical language recommended by W3C for Semantic Web applications, to specify the deontic part of the *OCeAN* meta-model. More precisely, we show how it is possible to

---

[1] http://www.w3.org/2007/OWL/wiki/OWL_Working_Group

specify social commitment to express conditioned obligations and prohibitions on time intervals, in OWL.

There are many advantages of using a decidable logical language like OWL to specify an open interaction system, and in particular that: (i) Semantic Web technologies are increasingly becoming a standard for Internet applications; (ii) the language is supported by reasoners (like Fact++[2], Pellet[3], and the rule reasoner of the Jena Semantic Web framework[4]) that are more efficient than available alternatives (like the Discrete Event Calculus Reasoner[5]); (iii) it is possible to achieve a high degree of interoperability of data and applications, which is indeed a crucial precondition for the development of open systems.

The idea of using OWL for modelling and monitoring the dynamic evolution of open artificial institutions can be developed following different approaches. A first option would be to implement an institutional model in a object oriented language like Java, and use OWL only to specify the ontology of the content of communicative acts and norms. As a result reasoning may be used to deduce, for example, that the performance of a certain act implies the performance of another act, and thus the fulfillment of a given commitment. An alternative approach, which we investigate in this paper, consists in using OWL to express, as far as possible, the normative component of the OCeAN meta-model. As we shall see, this requires the use of SWRL (Semantic Web Rule Language[6]) and Java code to overcome certain expressiveness limitations of OWL. Indeed, with both OWL 1 (the current standard) and OWL 2 there are at least two major problems:

- The treatment of time. OWL has no temporal operators; on some occasions it is possible to bypass the problem by using SWRL rules and built-ins for comparisons, but in any case this does not provide full temporal reasoning capabilities. Another possible solution would consist in using the OWL Time Ontology[7], but given that its axiomatization is very weak, this alternative presents limitations analogous to those previously discussed.
- The open-world assumption. In many applications, nor being able to infer that an action has been performed is sufficient evidence that the action has not been performed; one would then like to infer, for example, that an obligation to perform the action has been violated. As standard OWL reasoning is carried out under the open world assumption, inferences of this type cannot be drawn. However, it is often possible to simulate a closed world assumption by adding closure axioms to an ontology.

The main contribution of this paper, with respect to our previous works, is to show how obligations and prohibitions can be formalized in OWL and SWRL for

---

[2] http://owl.man.ac.uk/factplusplus/

[3] http://clarkparsia.com/pellet

[4] http://jena.sourceforge.net/inference/

[5] http://decreasoner.sourceforge.net

[6] http://www.w3.org/Submission/SWRL/

[7] http://www.w3.org/TR/owl-time/, http://www.w3.org/2006/time.rdf

monitoring and simulation purposes with significant performance improvements with respect to the solution based on the Event Calculus that we presented elsewhere [10]. Another contribution of this work is a hybrid solution of the problem of monitoring the temporal evolution of obligations and prohibition, based on application independent upper ontology of those concepts, extended including other ontologies specific for the domain of application, a set of SWRL rules, and a Java program implemented using suitable OWL libraries (like the Jena Semantic Web Framework for Java[8] or OWL API[9]).

The paper is organized as follows. In the next section we briefly introduce OWL and SWRL, that is, the Semantic Web languages that we use to formally specify the normative component of an open interaction system. In Section 3 we specify the algorithms that we plan to use to simulate or monitor the temporal evolution of an interaction system. Then in Section 4 we define the classes, properties, axioms, and rules that we take to underlie the normative specification of every interaction system. In Section 5 we present an actual example of a system specified using the proposed ontology. Finally in Section 6 we compare our approach with other proposals and draw some conclusions.

2   OWL and SWRL

OWL is a practical realization of a Description Logic system known as $\mathcal{SROIQ}(\mathcal{D})$. It allows one to define *classes* (also called *concepts* in the DL literature), *properties* (also called *roles*), and *individuals*. An OWL ontology consists of: a set of class axioms to describe classes, which constitute the *Terminological Box* (*TBox*); a set of property axioms to describe properties, which constitute a *Role Box* (*RBox*); and a collection of assertions to describe individuals, which constitute an *Assertion Box* (*ABox*).

Classes can be viewed as formal descriptions of sets of objects (taken from a nonempty universe), and individuals can be viewed as names of objects of the universe. Properties can be either *object properties* or *data properties*. The former describe binary relations between objects of the universe; the latter, binary relationships between objects and data values (taken from XML Schema datatypes).

A class is either a *basic class* (i.e., an atomic class name) or a *complex class* build through a number of available *constructors* that express Boolean operations and different types of restrictions on the members of the class.

Through *class axioms* one may specify subclass or equivalence relationships between classes, and that certain classes are disjoint. *Property axioms* allow one to specify that a given property is a subproperty of another property, that a property is the inverse of another property, or that a property is functional or transitive. Finally, *assertions* allow one to specify that an individual belongs to a class, that an individual is related to another individual through an object

---

[8] http://jena.sourceforge.net/
[9] http://owlapi.sourceforge.net/

property, that an individual is related to a data value through a data property, or that two individuals are equal or different.

OWL can be regarded as a decidable fragment of First Order Logic (FOL). The price one pays for decidability, which is considered as an essential preconditions for exploiting reasoning in practical applications, is limited expressiveness. Even in OWL 2 (the more expressive version currently under specification) certain useful first-order statements cannot be formalized.

Recently certain OWL reasoners, like Pellet, have been extended to deal with SWRL rules. SWRL is a Datalog-like language, in which certain universally quantified conditional axioms (called *rules*) can be stated. To preserve decidability, however, rules have to be used in the *safe mode*, which means that before being exploited in a reasoning process all their variables must be instantiated by pre-existing individuals. An important aspect of SWRL is the possibility of including *built-ins*, that is, Boolean functions that perform operations on data values and return a truth value.

Conventions

In what follows we use the notation $p : C \rightarrow_O D$ to specify an *object property p* (not necessarily a function) with class C as domain and class D as range, and the notation $q : C \rightarrow_D T$ to specify a *data property q* with class C as domain and the datatype T as range. We use capital initials for classes, and lower case initials for properties and individuals.

3   Specification and Simulation or Monitoring of an Open Interaction System

Our approach is to model an open interaction system using one or more artificial institutions. The definition of a specific artificial institution consists of: (i) a first component, called meta-model, which includes the definition of basic entities common to the specification of every institution, like the concepts of temporal proposition, commitment, institutional power, role, and norm, and the actions necessary for exchanging messages; (ii) a second component, pertaining to the institution in exam, which includes the definition of specific powers and norms that apply to the agents playing roles in the institution, and the definition of the concepts pertaining to the domain of the interaction (for example the actions of paying or delivering a product, bidding in an auction, etc.).

We start from the specification of a system, formalized as an application-independent OWL ontology (including a TBox, an RBox, and an ABox as detailed in Section 4). We then add an application-dependent ontology (as exemplified in Section 5) and use a Java program to let such ABox evolve in time on the basis of the events, with the goal of *monitoring* the fulfilment or violation of obligations and prohibitions. Those events could be actual events that happen during the run time of the system mainly due to the interaction of the agents or events registered in a possible history of the system for simulation purposes.

In particular, when the system is used for run time monitoring, a Java program updates the state of the system, that is, it updates the ABox with new assertions

to model the elapsing of time, to allow for closed-world reasoning on certain classes, and to model the actions performed by the interacting agents. When such updating is completed, a reasoner can be used to deduce the state of obligations and prohibitions. After that, when the ontology has reached a stable state (in the sense that all closed-world reasoning has been completed), the agents may perform queries to know what are their pending obligations or prohibitions or to react to their violation or fulfillment. We assume that the events or actions that happen between two phases of update (that is, between two discrete instant of time) are queued in the data structure `ActionQueue` for being managed by the Java program subsequently.

When the system is used for simulation, the set of events that happen at run-time are known since the beginning, and are represented in the initial version of the ABox. In such a case the Java program simply updates the state of the system to represent the elapsing of time and to allow closed-world reasoning on certain classes; then the reasoner deduces the state of obligations and prohibitions at each time instant.

Temporal evolution of the ontology

An external Java program is used to update the ABox to model the elapsing of time, the actions performed by the interacting agents at run-time (in the monitoring usage), and to allow for closed-world reasoning on certain classes (see Section 4.1 for details). It is important to underline that the external program is used to update the ABox of the ontology and only to add new knowledge acquired due to time elapsing, it is never used to remove knowledge.

The program performs the following operations:

1. initialize the simulation/monitoring time t equal to 0 and close the extensions of the classes C, on which it is necessary to perform closed-world reasoning, by asserting that the class KC is equivalent to the enumeration of all individuals that can be proved to be members of the class C retrieved with the `retrieve(C)` query command;
2. insert in the ABox the assertion $happensAt(elapse, t)$;
3. insert in the ABox the events or actions that happen in the system between $t-1$ and t and that are cached in the `ActionQueue` queue (this involves creating new individuals of the class $Event$);
4. run a reasoner (more specifically, Pellet 2.0) to deduce all assertions that can be inferred (truth values of temporal propositions, states of commitments, etc.);
5. update the closure of the relevant classes C;
6. increment the time of simulation t by 1 and go to the point 2.

After point 5, given that the ontology has reached a stable state it is possible to let agents perform queries about pending, fulfilled, or violated commitments in order to plan their subsequent actions and to apply sanctions or rewards. When the ontology is used for monitoring purposes, and given that internal time (i.e., the time as represented in the ontology) is discrete, it is necessary to wait the actual number of seconds that elapse between two internal instants.

The corresponding Java pseudo code is as follows:

```
t=0
for each class C that has to be closed
   assert KC ≡ {i₁,...iₙ} with {i₁,...iₙ} = retrieve(C)
while t<timeSimulation {
   assert happensAt(elapse,t)
   for each event  eₙ  in ActionQueue
      assert happensAt(eₙ,t)
   run Pellet reasoner
   for each class C that has to be closed
      remove equivalent class axioms of class KC
      assert KC ≡ {i₁,...iₙ} with {i₁,...iₙ} = retrieve(C)
   run agents queries
   t=t+1
}
```

4 The Ontology of Obligations and Prohibitions

In this section we present the TBox, the RBox, and part of the Abox that have to be included in the ontology of any interaction system modelled using the OCeAN concepts of temporal proposition, commitment, role, and norm. In particular we specify the classes, the properties and the axioms for modelling those concepts and introduce some SWRL rules to deduce the truth value of temporal propositions. Social commitments are a crucial concept in our approach because they are used to model obligations and prohibitions due either to the activation of norms or created by the performance of communicative acts, like promises. Thanks to their evolution in time, commitments can be used to monitor the behavior of autonomous agents by detecting their *violation* or *fulfilment*, as a precondition for reacting with suitable *passive or active sanctions* or with a *reward* [10].

Some general classes of our ontology are used as domain or range of the properties used to describe temporal propositions and commitments; they are class *Event*, class *Action* and class *Agent*. In particular, an event may have as a property its *time* of occurrence. Class *Action* is a subclass of *Event*, and has a further property used to represent the *actor* of the action. Such properties are defined as follows:

$Event \sqcap Agent \sqsubseteq \bot$; $Action \sqsubseteq Event$;
$hasActor : Action \rightarrow_O Agent$;
$happensAt : Event \rightarrow_D integer$;

To represent the elapsing of time we introduce in the ABox the individual *elapse*, that is asserted to be a member of class *Event*: *Event(elapse)*.

4.1 Temporal Propositions

Temporal propositions are used to represent the *content* and *condition* of social commitments. They are a construct used to relate in two different ways a *proposition* to an *interval of time*. In the current OWL specification, we distinguish between *positive temporal propositions* used in commitments to represent

obligations (when an action has to be performed within a given interval of time), and *negative temporal propositions* used to model prohibitions (when an action must not be performed during a predefined interval of time).

The classes necessary to model temporal propositions are *TemporalProp*, with the two subclasses *TPPos* and *TPNeg* used to distinguish between positive and negative temporal propositions. The classes *IsTrue* and *IsFalse* are used to model the truth values of temporal propositions. All this is specified by the following axioms:

$TemporalProp \sqcap Agent \sqsubseteq \bot; TemporalProp \sqcap Event \sqsubseteq \bot;$
$TPPos \sqsubseteq TemporalProp; TPNeg \sqsubseteq TemporalProp;$
$TPPos \sqcap TPNeg \sqsubseteq \bot;$
$TemporalProp \equiv TPPos \sqcup TPNeg;$
$IsTrue \sqsubseteq TemporalProp; IsFalse \sqsubseteq TemporalProp;$
$IsTrue \sqcap IsFalse \sqsubseteq \bot;$

The class *TemporalProp* is the domain of the following object and data properties further specified with a cardinality axiom:

$hasAction : TemporalProp \rightarrow_O Action;$
$hasStart : TemporalProp \rightarrow_D integer;$
$hasEnd : TemporalProp \rightarrow_D integer;$
$TemporalProp \sqsubseteq= 1\ hasAction \sqcap\ = 1\ hasStart \sqcap\ = 1\ hasEnd$

The classes $IsTrue$ and $IsFalse$ are used to keep track of the truth value of temporal propositions by means of two SWRL rules, that are different on the basis of the type of temporal proposition. A positive temporal proposition (i.e., a member of class *TPPos*) is used to represent an obligation to do something in a given interval of time, with starting points t_{start} and deadline t_{end}. We therefore introduce a rule that deduces that the truth value of the temporal proposition is true (i.e., the temporal proposition becomes member of the class $IsTrue$) if the action associated to the temporal proposition is performed between the t_{start} (inclusive) and the t_{end} (exclusive) of interval of time associated to the same proposition. In the following SWRL rule we use two built-ins to compare the current time with the interval of time associated to the temporal proposition:

RuleTPPos1
happensAt(elapse,?t) ∧ happensAt(?a,?t) ∧ TPPos(?tp) ∧ hasAction(?tp,?a) ∧ hasStart(?tp,?ts) ∧ hasEnd(?tp,?te) ∧ swrlb:lessThanOrEqual(?ts,?t) ∧ swrlb:lessThan(?t,?te) → IsTrue(?tp)

We then have to define a rule that, when the time t_{end} of a positive temporal proposition elapses, and such a temporal proposition is not true, deduces that the temporal proposition is member of the class $IsFalse$. Here closed-world reasoning comes into play, because we cannot assume the ABox to contain an explicit assertion that an action has not been performed: rather, we want to deduce that an action has not been performed by the lack of an assertion that it has been performed. Clearly, an SWRL rule like

happensAt(elapse,?te) ∧ hasEnd(?tp,?te) ∧ TPPos(?tp) ∧(not IsTrue)(?tp) → IsFalse(?tp)

would not work, given that OWL/SWRL reasoners operate under the open world assumption. This means that the conclusion that a temporal proposition is false can only be reached for those propositions that can be definitely proved not to be members of $IsTrue$. On the contrary, if a temporal proposition is not deduced to be $IsTrue$ by RuleTp1, even if its deadline has been reached it will not be deduced to be $IsFalse$.

To solve this problem we first assume that our ABox contains complete information on the actions performed before the current time of the system. This allows us to adopt a closed-world perspective as far as the performance of actions is concerned. More specifically, we assume that the program specified in Section 3 will always update the ABox when an event has happened (i.e. the program can only inset in the ABox the information that an event has happened at current time t); we then want to deduce that all temporal propositions, that cannot any longer become true because their deadline has elapsed, are false.

To get this result we need to perform some form of closed world reasoning on class $IsTrue$. As stated in [17] "the DL $\mathcal{ALCK}$ [7] adds a non-monotonic **K** operator (which is a kind of necessity operator) to the DL $\mathcal{ALC}$ to provide the ability to "turn on" the Closed World Assumption (CWA) when needed. The reasoning support for $\mathcal{ALCK}$ language has been implemented in Pellet to answer CWA queries that use the **K** operator". However, our ontology uses a more expressive DL than $\mathcal{ALC}$; moreover, the use of the **K** operator in SWRL rules is not supported.

We therefore take a different approach, based on an explicit *closure* of class $IsTrue$. More precisely, we introduce a new class, $KIsTrue$, which is meant to contain all temporal propositions that, at a given time, are known to be true. Class $KIsTrue$ therefore represents, at any given instant, the explicit closure of class $IsTrue$. Given its intended meaning, class $KIsTrue$ has to be a subclass of $IsTrue$ (and, as a consequence, of $TemporalProp$):

$KIsTrue \sqsubseteq IsTrue$

To maintain class $KIsTrue$ as the closure of class $IsTrue$, we define it periodically as equivalent to the enumeration of all individuals that can be proved to be members of $IsTrue$. This can be done by the Java program used to update the ABox to keep track of the elapsing of time (described in Section 3) by executing the operations described in the following pseudo-code:

```
assert KIsTrue ≡ {tp_1,...tp_n} with {tp_1,...tp_n} = retrieve(IsTrue)
```

We now introduce a new class, $NotKIsTrue$, which is intended to contain all temporal propositions whose deadline is elapsed, and that are not members of $KIsTrue$. Such a class is defined as the difference between the set of all individuals that belong to $TemporalProp$, and the set of all those individuals that are members of $KIsTrue$:

$NotKIsTrue \equiv TemporalProp \sqcap \neg KIsTrue$

We are now ready to write a rule to deduce that the truth value of a positive temporal proposition is false if the deadline of the temporal proposition has elapsed, and it is not known that the associated action has been performed:

RuleTPPos2

happensAt(elapse,?te) $\wedge$ *hasEnd(?tp,?te)* $\wedge$ *TPPos(?tp)* $\wedge$ *NotKIsTrue(?tp)*
$\rightarrow$ *IsFalse(?tp)*

We now turn to negative temporal propositions, that is, temporal propositions that are members of the class *TPNeg* and are used to represent the prohibition to do something in a given interval of time. Such propositions belong to class *IsFalse* when the associated action is performed in the interval between t_{start} (inclusive) and t_{end} (exclusive). This can be deduced by the following rule:

RuleTPNeg1

happensAt(elapse,?t) $\wedge$ *happensAt(?a,?t)* $\wedge$ *TPNeg(?tp)* $\wedge$ *hasAction(?tp,?a)* $\wedge$
hasStart(?tp,?ts) $\wedge$ *hasEnd(?tp,?te)* $\wedge$ *swrlb:lessThanOrEqual(?ts,?t)* $\wedge$
swrlb:lessThan(?t,?te) $\rightarrow$ *IsFalse(?tp)*

Similarly to what we did for RuleTPPos2, we now use the closure of class *IsFalse*, that we call *KIsFalse*, to deduce that a negative temporal proposition *IsTrue* when its t_{end} has been reached and it has not yet been deduced that the proposition *IsFalse*:

$KIsFalse \sqsubseteq IsFalse$
$NotKIsFalse \equiv TemporalProp \sqcap \neg KIsFalse$

RuleTPNeg2

happensAt(elapse, ?te) $\wedge$ *hasEnd(?tp,?te)* $\wedge$ *TPNeg(?tp)* $\wedge$ *NotKIsFalse(?tp)*
$\rightarrow$ *IsTrue(?tp)*

4.2   Commitment

In the *OCeAN* meta-model of artificial institutions, commitments are used to model a social relation between a *debtor* a *creditor*, about a certain *content* and under a *condition*. Our idea is that by means of the performance of communicative acts, or due to the activation of norms, certain agents become committed with respect to another agent to perform a certain action within a given deadline (an *obligation*), or not to perform a given action during a given interval of time (a *prohibition*). Such commitments can be conditional on the truth of some proposition. In our model we assume that if an action is neither obligatory nor prohibited, then it is *permitted*.

In order to detect and react to commitment violation and fulfilment we need to deduce a commitments *state* (in our previous works [10] we also introduced precommitments to define the semantics of requests, but this is not relevant in the current work). We introduce in the ontology the class *Commitment*, disjoint from *Event*, *Agent* and *TemporalProp*.

$Commitment \sqcap Agent \sqsubseteq \bot$; $Commitment \sqcap Event \sqsubseteq \bot$;
$Commitment \sqcap TemporalProp \sqsubseteq \bot$;

The *Commitment* class is the domain of the following object properties:

$hasDebtor : Commitment \rightarrow_O Agent$;
$hasCreditor : Commitment \rightarrow_O Agent$;
$hasContent : Commitment \rightarrow_O TemporalProp$;
$hasCondition : Commitment \rightarrow_O TemporalProp$;
$hasSource : Commitment \rightarrow_O Norm$;
$Commitment \sqsubseteq \exists hasDebtor \sqcap \exists hasCreditor \sqcap =1hasContent \sqcap =1hasCondition$;

The *hasSource* property is used to keep track of the norm that generated a commitment, as explained in Section 4.3. Obviously the debtor of a commitment has to be the actor of the action to which it is committed, as expressed by the following axiom:

$$hasContent \circ hasAction \circ hasActor \sqsubseteq hasDebtor$$

In some situations it is necessary to create unconditional commitments. To avoid writing different rules for conditional and for unconditional commitments, we introduce a temporal proposition individual, $tpTrue$, whose truth value is initially true; that is, we assert: $IsTrue(tpTrue)$. An unconditional commitment is then defined as a conditional commitment whose condition is $tpTrue$.

Our next problem is deducing whether a given commitment is:

- *pending*, when its condition is satisfied but its content is not known to be $IsTrue$ or to be $IsFalse$;
- *fulfilled*, when is content is known to be $IsTrue$;
- *violated*, when its content is known to be $IsFalse$ and its condition is known to be $IsTrue$.

Knowing the state of a commitment may be important for the interacting agents to plan their actions on the basis of the advantages of fulfilling certain commitments. We therefore introduce classes $IsPending$, $IsFulfilled$, and $IsViolated$, defined by the following axioms:

$IsFulfilled \sqcap IsViolated \sqsubseteq \bot$;
$IsPending \sqsubseteq Commitment$; $IsFulfilled \sqsubseteq Commitment$;
$IsViolated \sqsubseteq Commitment$;

We define the following axiom to deduce that a commitment is member of the class $IsPending$:

Axiom1
$IsPending \equiv (\exists\ hasContent.NotKIsTrue) \sqcap (\exists\ hasContent.NotKIsFalse) \sqcap (\exists\ hasCondition.IsTrue))$

Note that as classes $NotKIsTrue$ and $NotKIsFalse$ are updated after running the reasoner, as soon as the *content* of a commitment becomes true the commitment is member of both class $IsPending$ and class $IsFulfilled$.

Lists of fulfilled and of violated commitments can be obtained by retrieving the individuals that are respectively members of class $IsFulfilled$ or $IsViolated$, defined by the following axioms:

Axiom2
$IsFulfilled \equiv \exists\, hasContent.IsTrue$

Axiom3
$IsViolated \equiv (\exists\, hasContent.IsFalse) \sqcap (\exists\, hasCondition.IsTrue)$

4.3   Norms and Roles

In OCeAN, norms are introduced to model obligations and prohibitions that, contrary to those created at run time by the performance of communicative acts, are implied by an institutional setting and can be specified at design time. For example, norms can be used to state the rules of an interaction protocol, like the protocol of a specific type of auction, or the rules of a seller-buyer interaction. Given that norms are usually specified at design time, when it is impossible to know which agents will actually interact in the system, one of their distinctive features is that they have to be expressed in term of the *roles* played by the agents. Therefore at run-time, when a norm becomes *active* (i.e., when its activating event happens), the actual debtor and creditor of the obligation or prohibition generated by the norm have to be computed on the basis of the roles played by the agents in the system at that moment.

Another important aspect of norms is that to enforce their fulfillment in an open system, it must be possible to specify *sanctions* or *rewards*. In [9] we suggested that a satisfactory model of sanctions has to distinguish between two different type of actions: the action that the violator of a norm has to perform to extinguish its violation (which we call *active sanction*), and the action that the agent in charge of norm enforcement may perform to deter agents from violating the norm (which we call *passive sanction*). Active sanctions can be represented in our model through a temporal proposition, whereas passive sanctions can be represented as new specific powers that the agent entitled to enforce the norm acquires when a norm is violated. As far as passive sanctions are concerned, another norm (that in [16]) is called *enforcement norm*) may oblige the enforcer to punish the violation. Due to space limitations, in this paper we do not model the notion of *power*; thus passive sanctions are not treated in this paper. An obligation or prohibition generated by a norm can in turn violated; it will therefore be necessary to monitor the fulfillment or violation of such obligations or prohibition t punish the violation.

Role

Typically, artificial institutions provide for different roles. In a run of an auction, for example, we may have the roles of auctioneer and of participant; in a company, like an auction house, we may have the roles of boss or employee; and so on. More generally, also the debtor and the creditor of a commitment may be regarded as roles. Coherently with these examples, a role is identified by a label (like *auctioneer*, *participant*, etc.) and by the institutional entity that provides for the role. Such an institutional entity may be an organization (like an auction house), an institutional activity (like a run of an auction), or an institutional

relationship (like a commitment). For example an agent may be the *auctioneer* of run 01 of a given auction, or an *employee* of IBM, or the *creditor* of a specific commitment.

We introduce class *Role* to represent the set of possible labels that representing roles and class *InstEntity* to represent the institutional entity within which a given role is played. Elements of class *AgentInRole* are used to reify the fact that an agent plays a given role in a given institutional entity. Those classes are related by the following object properties:

$isPlayedBy : AgentInRole \rightarrow_O Agent$;
$hasRole : AgentInRole \rightarrow_O Role$;
$isIn : AgentInRole \rightarrow_O InstEntity$;

Norm

Summarizing, a norm has: a *content* and a *condition*, modelled using temporal propositions; a *debtor* and a *creditor*, expressed in term of roles; an *activating event*; and a collection of *active* and *passive sanctions*. Norms are represented in our ontology using class *Norm* and the following object properties:

$hasRoleDebtor : Norm \rightarrow_O Role$; $hasRoleCreditor : Norm \rightarrow_O Role$;
$hasNContent : Norm \rightarrow_O TemporalProp$;
$hasNCondition : Norm \rightarrow_O TemporalProp$;
$hasActivation : Norm \rightarrow_O Event$;
$hasASanction : Norm \rightarrow_O TemporalProp$;
$hasPSanction : Norm \rightarrow_O Power$;

When a norm is activated it is necessary to create as many commitments as there are agents playing the role associated to the debtor property of the norm. For example, the activation of a norm that applies to all the agents playing the role of *participant* of an auction, creates a commitment for each participant currently taking part to the auction. The creditors of these commitments are the agents that play the role reported in the creditor property of the norm. All these commitments have to be related by the *hasSource* object property (defined in Section 4.2) to the norm that generated them; this is important to know which norm generated a commitment and what sanctions apply for the violation of such commitment.

As every commitment is an individual of the ontology, the activation of a norm involves the generation of new individuals. However, the creation of new individuals in an *ABox* cannot be performed using OWL or SWRL. There are at least two possible solutions to this problem, which we plan to investigate in our future work. The first consists in defining a set of axioms in the ontology that allows the reasoner to deduce the existence of those commitments as anonymous objects with certain properties. With this solution, an agent that needs to know its pending commitments instead of simply retrieving the corresponding individuals will have to retrieve their contents, conditions and debtors. Another possible solution consists in defining a new built-in that makes it possible for SWRL rules to create new individuals as members of certain classes and with given properties. A similar problem will have to be solved to manage the creation of a sanctioning
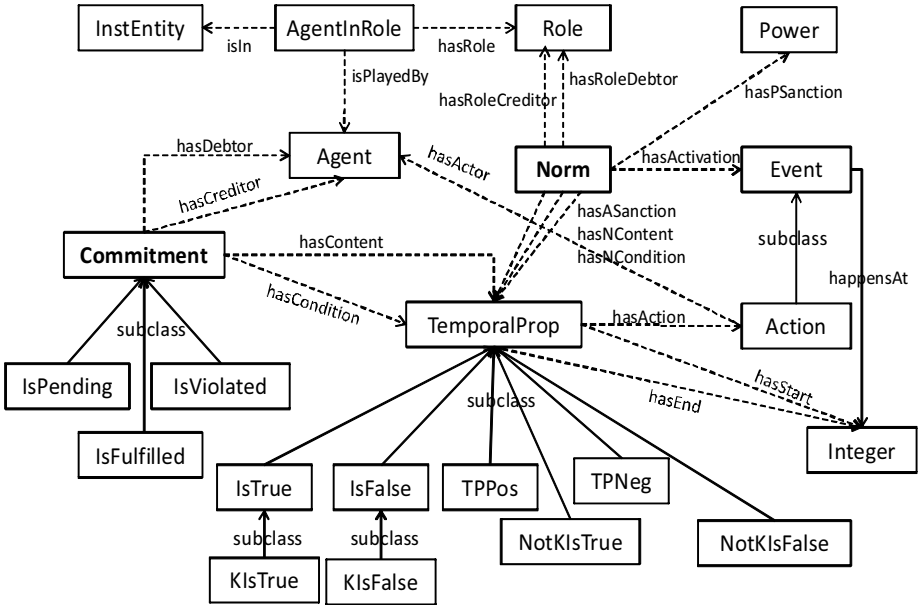
Fig. 1. Graphical representation of the ontology

commitment generated by the violation of a commitment related to a norm, which has as content the temporal proposition associated to the active sanction of the norm.

In Figure 1 classes, subclasses, and properties (dotted lines) of the ontology described in this section are graphically represented.

5   Example

In this section we show how it is possible to specify the state of an interaction system and to simulate or monitor its evolution in time. To do so it is necessary to integrate the ontology defined in the previous sections with an application-dependent ontology, and to insert a set of individuals for representing commitments and temporal propositions in the ABox. In a real application these commitments and their temporal propositions will be created by the performance of communicative acts (defined in the *OCeAN* agent communication library [10]) or by the activation of norms. If the system is used for monitoring purposes, we assume that there is a way of mapping the actions that are actually executed onto their counterparts in the ontology.

Here we describe an example of interaction where a buyer agent, Ann, promises to pay a certain amount of money for a product (a book) to a seller agent, Bob, on condition that the seller agent delivers the product to Ann. We also represent the

prohibition for the seller to deliver a different product (a CD). Different possible evolution of the state of the interaction are possible on the basis of the agents' actions.

The ontology described in the previous sections has to be integrated with the action for paying an amount of money and for delivering a product. In a realistic application they would be described in a detailed domain-dependent ontology introducing the class *Pay* and the class *Delivery* as subclass of the class *Action* described in our application independent ontology. Both of those type of action have a *receiver* and an *object*, the pay action type also has an *amount of money*. For simplicity in this example we represent the action of payment of the book and its delivery with individuals inserted in the ABox.

The agents are represented with the following assertions:

$Agent(ann); Agent(bob); \neq (ann, bob);$

The actions that we are interested to model in the ontology are represented by the following assertions:

$Action(payBook1); Action(deliverBook1); Action(deliverCD1);$
$hasActor(payBook1, ann); hasActor(deliverBook1, bob);$
$hasActor(deliverCD1, bob);$
$\neq (payBook1, deliverBook1, deliverCD1, elapse);$

Temporal propositions are represented by the following assertions:

$TPPos(tpPayBook1); TPPos(tpDeliverBook1); TPNeg(tpNotDeliverCD1);$
$hasAction(tpPayBook1, payBook1); hasStart(tpPayBook1, 1);$
$hasEnd(tpPayBook1, 3);$
$hasAction(tpDeliverBook1, deliverBook1); hasStart(tpDeliverBook1, 1);$
$hasEnd(tpDeliverBook1, 2);$
$hasAction(tpNotDeliverCD1, deliverCD1); hasStart(tpNotDeliverCD1, 0);$
$hasEnd(tpNotDeliverCD1, 3);$
$\neq (tpPayBook1, tpDeliverBook1, tpTrue); \neq (tpNotDeliverCD1, tpTrue);$

Commitments are represented by the following assertions:

$Commitment(c1); Commitment(c2); Commitment(c3);$
$hasDebtor(c1, ann); hasCreditor(c1, bob);$
$hasContent(c1, tpPayBook1); hasCondition(c1, tpDeliverBook1);$
$hasDebtor(c2, bob); hasCreditor(c2, ann);$
$hasContent(c2, tpDeliverBook1); hasCondition(c2, tpTrue);$
$hasDebtor(c3, bob); hasCreditor(c3, ann);$
$hasContent(c3, tpNotDeliverCD1); hasCondition(c3, tpTrue);$
$\neq (c1, c2, c3);$

The history of the system is represented by the following assertions (the action happens at time 1):

$happensAt(deliverBook1, 1)$

We created the ontology of the interaction system with the free, open source ontology editor Protege 4.0 beta[10]. As this version of Protege does not support the

---

[10] http://protege.stanford.edu/

editing of SWRL rules, we created them with Protege 3.4 and inserted their RDF/XML code in the ontology file. We implemented the Java program described in Section 3 using OWL-API library to operate on the ontology and the source code of Pellet 2.0 to reason and query it[11].

In Table 1 we report the evolution of the ontology ABox in time, with particular regard to the truth value of the temporal propositions and the state of commitments. As the extension of classes $KIsTrue$ and $KIsFalse$ are computed by an external program, when the reasoner runs their extensions are specified in the axiom relative to the previous state. In the table we abbreviate the assertion $happensAt(elapse, n)$ with the expression $t = n$.

Table 1. Dynamic evolution of the state of the system

time	$t = 0$	$t = 1$	$t = 2$	$t = 3$
$tpPayBook1[1, 3]$				$IsFalse$
$tpDeliverBook1[1, 2]$		$IsTrue$	$IsTrue$	$IsTrue$
$tpNotDeliverCD1[0, 3]$				$IsTrue$
$c1(ann, bob,$ $tpPayBook1,$ $tpDeliverBook1)$		$IsPending$	$IsPending$	$IsViolated$
$c2(bob, ann,$ $tpDeliverBook1,$ $tpTrue)$	$IsPending$	$IsFulfilled$	$IsFulfilled$	$IsFulfilled$
$c3(bob, ann,$ $tpNotDeliverCD1,$ $tpTrue),$	$IsPending$	$IsPending$	$IsPending$	$IsFulfilled$

Classes updated by the external program

$KIsTrue$	$\{tpTrue\}$	$\{tpTrue,$ $tpDeliverBook1\}$	$\{tpTrue,$ $tpDeliverBook1\}$	$\{tpTrue,$ $tpDeliverBook1,$ $tpNotDeliverCD1\}$
$KIsFalse$	$nothing$	$nothing$	$nothing$	$\{tpPayBook1\}$

6   Conclusions and Related Works

The main contributions of this paper, with respect to our previous works and with respect to other approaches, are as follows. We show how conditional obligations and prohibitions with stating points and deadlines may be specified and monitored using OWL and SWRL with significant advantages with respect to other approach that use other formal languages. Moreover we propose a hybrid solution, based on an OWL ontology, SWRL rules, and a Java program, to the problem of monitoring the time evolution of obligations and prohibitions.

In particular if we compare this specification with another one that we presented elsewhere based on Event Calculus [10] we observe significant

---

[11] The ontology file, its representation in DL, the Java program and its output that represent the time evolution of the state of the system as depicted in Table 1 can be found at http://www.people.lu.unisi.ch/fornaran/ontology/DALT09Ontology.html

improvement in performance (even if a complete comparison will be possible only when the complete *OCeAN* meta-model will be formalized with Semantic Web Technology). Moreover semantic web technologies are becoming an international standard for web applications and numerous tools, reasoners, and libraries are available to support the development and usage of ontologies. This, in spite of the drawbacks on time reasoning and due to the limits of OWL language expressivity, is a crucial advantage with respect to other languages used in the multiagent community for the specification of norms and organizations, like as we already mentioned the Event Calculus [19,1], or other specific formal languages like the one required by the rule engine Jess [13,5], or a variant of Propositional Dynamic Logic (PDL) used to specify and verify liveness and safety properties of multi-agent system programs with norms [6], or Process Compliance Language (PCL) [14].

In literature there are few approaches that use semantic web languages for the specification of multiagent systems and in particular of obligations and prohibitions. One of the most interesting one is the approach for policy specification and management presented in the KAoS framework [18]. Even if in English the word *norm* and *policy* have different meaning and also in informatics literature they could be referred to two different concepts [3], in the MAS community they may have very close meanings. In KAoS a policy could be a positive or negative authorization to perform an action (that is a permission or a prohibition) or it can be an obligation. Like in our approach in KAoS policies are specified using a set of concepts defined in an OWL *core ontology* that could be extended with application dependent ontologies. A crucial difference between KAoS approach and the approach presented in this paper is in the methods used for monitoring and enforcement of policies or norms. In KAoS policies are usually regimented (as far as is possible given that it is almost impossible to regiment obligations [9]) by means of "guards" and are monitored by means of platform specific mechanisms. Differently in our proposal norms are enforced by means of sanctions or rewards and are monitored by deducing their fulfillment or violation with an OWL reasoner (we use Pellet but other OWL 2 reasoners could be used) and an external Java program.

Another example is the one presented in [15] where prohibited, obliged and permitted actions are represented as object properties from agents to actions. But without the reification of the notion of obligation and prohibition that we propose here, it is very difficult to find a feasible solution to express conditional commitments with deadlines and it is impossible to detect what norms and how many time were fulfilled or violated. Moreover the approach proposed for detecting violations is based on the external performance of SPARQL queries and on the update of the ABox to register that an obligation/prohibition resulted violated; however SPARQL queries do not exploit the semantics specified by the ontology, moreover it is necessary to write different queries for every possible different action that has to be monitored and for the execution of SPARQL queries it is necessary to use a proper additional tool.

In [2] a hybrid approach is presented: they define a communication acts ontology using OWL and express the semantics of those acts through social commitments that are formalized in the Event Calculus. This work is complementary with respect to our approach, in fact we specify also the semantics of social commitments using semantic web technologies. Semantic web technologies in multiagent systems can be used also to specify domain specific ontologies used in the content of norms like in [8].

Another interesting contribution of this work is due also to the exemplification of a solution to the problem to performing closed world reasoning on certain classes in OWL. Another work that tackles a similar problem in a different domain, the ontology of software models, is [4].

Indeed this model is still incomplete e we plan to investigate how it is possible to manage the creation of commitments to model norm activations, and to model active sanctions, moreover we plan to study how to formalize the notion of power to express the semantics of declarative communicative acts and of passive sanctions.

References

1. Artikis, A., Sergot, M., Pitt, J.: Animated Specifications of Computational Societies. In: Castelfranchi, C., Johnson, W.L. (eds.) Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002), pp. 535–542. ACM Press, New York (2002)
2. Berges, I., Bermúdez, J., Goñi, A., Illarramendi, A.: Semantic web technology for agent communication protocols. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 5–18. Springer, Heidelberg (2008)
3. Bradshaw, J., Beautment, P., Breedy, M., Bunch, L., Drakunov, S., Feltovich, P., Hoffman, R., Jeffers, R., Johnson, M., Kulkarni, S., Lott, J., Raj, A., Suri, N., Uszok, A.: Making agents acceptable to people, pp. 355–400. Springer, Heidelberg (2004)
4. Bräuer, M., Lochmann, H.: An ontology for software models and its practical implications for semantic web reasoning. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 34–48. Springer, Heidelberg (2008)
5. da Silva1, V.T.: From the specification to the implementation of norms: an automatic approach to generate rules from norms to govern the behavior of agents. Autonomous Agents and Multi-Agent Systems 17(1), 113–155 (2008)
6. Dastani, M., Grossi, D., Meyer, J.-J., Tinnemeier, N.: Normative multi-agent programs and their logics. In: Boella, G., Noriega, P., Pigozzi, G., Verhagen, H. (eds.) Normative Multi-Agent Systems, Germany. Dagstuhl Seminar Proceedings, vol. 09121. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Dagstuhl (2009)
7. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A., Nutt, W.: An epistemic operator for description logics. Artificial Intelligence 200(1-2), 225–274 (1998)

8. Felicissimo, C., Briot, J.-P., Chopinaud, C., Lucena, C.: How to concretize norms in NMAS? An operational normative approach presented with a case study from the television domain. In: International Workshop on Coordination, Organization, Institutions and Norms in Agent Systems (COIN@AAAI 2008), 23rd AAAI Conference on Artificial Intelligence, Chicago, IL, Etats-Unis. AAAI Press, Menlo Park (2008)

9. Fornara, N., Colombetti, M.: Specifying and enforcing norms in artificial institutions. In: Baldoni, M., Son, T.C., van Riemsdijk, M.B., Winikoff, M. (eds.) DALT 2008. LNCS (LNAI), vol. 5397, pp. 1–17. Springer, Heidelberg (2009)

10. Fornara, N., Colombetti, M.: Specifying Artificial Institutions in the Event Calculus. In: Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models of Information science reference, ch. XIV, pp. 335–366. IGI Global (2009)

11. Fornara, N., Viganò, F., Colombetti, M.: Agent communication and artificial institutions. Autonomous Agents and Multi-Agent Systems 14(2), 121–142 (2007)

12. Fornara, N., Viganò, F., Verdicchio, M., Colombetti, M.: Artificial institutions: A model of institutional reality for open multiagent systems. Artificial Intelligence and Law 16(1), 89–105 (2008)

13. García-Camino, A., Rodríguez-Aguilar, J.A., Sierra, C., Vasconcelos, W.: Constraint rule-based programming of norms for electronic institutions. Autonomous Agents and Multi-Agent Systems 18(1), 186–217 (2009)

14. Governatori, G., Rotolo, A.: How do agents comply with norms? In: Boella, G., Noriega, P., Pigozzi, G., Verhagen, H. (eds.) Normative Multi-Agent Systems, Dagstuhl, Germany. Dagstuhl Seminar Proceedings. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2009)

15. Lam, J.S.-C., Guerin, F., Vasconcelos, W., Norman, T.J.: Representing and reasoning about norm-governed organisations with semantic web languages. In: Sixth European Workshop on Multi-Agent Systems Bath, UK, December 18-19 (2008)

16. López, F., López, Luck, M., d'Inverno, M.: A Normative Framework for Agent-Based Systems. In: Proceedings of the First International Symposium on Normative Multi-Agent Systems, Hatfield (2005)

17. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Web Semantics: Science, Services and Agents on the World Wide Web 5(2), 51–53 (2007)

18. Uszok, A., Bradshaw, J.M., Lott, J., Breedy, M., Bunch, L., Feltovich, P., Johnson, M., Jung, H.: New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of kaos. In: IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 145–152 (2008)

19. Yolum, P., Singh, M.: Reasoning about commitment in the event calculus: An approach for specifying and executing protocols. Annals of Mathematics and Artificial Intelligence 42, 227–253 (2004)