Representation and Monitoring of Commitments and Norms using OWL

Nicoletta Fornara [a,*], Marco Colombetti [a,b]
[a] *University of Lugano, via G. Buffi 13, 6900 Lugano, Switzerland*
E-mail: {fornaran,colombem}@usi.ch
[b] *Politecnico di Milano*
piazza Leonardo Da Vinci 32, Milano, Italy
E-mail: marco.colombetti@polimi.it

Monitoring the temporal evolution of obligations and prohibitions 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 with starting points and deadlines, 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 norm, 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 the temporal evolution of social commitments, taking into account the elapsing of time and the actions performed by the agents interacting within the system.

Keywords: Normative Multiagent Systems, Semantic Web Technology, Norms, Obligations, Prohibitions, OWL, Monitoring

1. Introduction

In an *open interaction system* [1,15], agents seek, offer, provide, and consume services by interacting with other agents. Given that interactions develop in time, an agent's ability to deal with temporal specifications is going to be crucial. In many cases of practical interest, moreover, temporal specifications concern deontic relationships, like obligations and prohibitions, that typically have starting points and deadlines. As a consequence, deontic temporal reasoning is going to play an important role in open interaction systems, at specification, verification, and run time. In this paper we are concerned with run-time reasoning on the temporal constraints of obligations and prohibitions. Such deontic relationships may arise from the norms regulating the context of an interaction, or from commitments intentionally undertaken by the agents, for example as a result of a communicative act. The most immediate applications of run-time reasoning on obligations and prohibitions is monitoring [20,10], that is, checking whether the deontic relationships created during an on-going interaction are fulfilled or violated. In turn, monitoring can be exploited for the management of actual interactions, or for simulation purposes (i.e., to check whether an agent will fulfill or violate its obligations or prohibitions under predefined test conditions). In most cases of interest, monitoring deontic relationships involves reasoning. Actually, it is possible to single out three relevant components of reasoning. The first component has to do with the intrinsic logic of deontic relationships; for example, if an action is not prohibited, then it is at least permitted, and may be obligatory. The second component has to do with temporal reasoning; for example, if an obligatory action has not been performed, and the deadline for performing it has elapsed, then the obligation has been violated. Finally, the third component has to do with domain-level reasoning; for example, if agent a is obligated to pay a given amount of money to agent b, and a executes a bank transfer to the order of b for the relevant amount, then the obligation is fulfilled.

To deal with these types of reasoning, different approaches can be taken. A possible solution is to adopt a specialized reasoner, which implements some suitable version of temporal deontic logic (like for example the one proposed in [8]). A different solution is to follow an approach to reasoning based on widely adopted languages and

[*] Corresponding author: Nicoletta Fornara, fornaran@usi.ch.

tools, like for example those developed in the context of Semantic Web technologies (as suggested, for example, in [19]). Either solution has its pros and cons. On the one hand, temporal deontic logics are usually defined as extensions of a propositional language; thus they are not suitable to represent complex domain knowledge, and it is not obvious that they would retain certain desirable properties (like decidable reasoning) if their base language is extended. On the other hand, some logical languages recommended in the Semantic Web area, like the OWL Web Ontology Language[1], are suitable to represent domain knowledge, but it is not clear how to use them to support deontic and temporal reasoning. In any case, there may be several advantages in adopting a language like OWL for implementing run-time monitoring of obligations and prohibitions: even if OWL is a very expressive language, reasoning is still decidable, and efficient reasoners (like HermiT[2] and Pellet[3]) are freely available and widely used; moreover, Semantic Web technologies are increasingly becoming a standard for Internet applications, and thus allow for a high degree of interoperability of data and applications, which is a crucial precondition for the development of open systems. In this paper we explore how to use OWL (in its OWL 2 DL (Description Logic) version) to specify obligations and prohibitions, in such a way that a standard reasoner can then be exploited to implement run-time monitoring. Obligations and prohibitions can respectively be regarded as positive and negative commitments which, as we argued in our previous works on the OCeAN meta-model for the specification of artificial institutions [14,15,13], are created either by the activation of norms associated to an agent's role, or by an agent's performance of a communicative act, like a promise or the acceptance of a proposal. More precisely, we shall use OWL to specify conditional obligations and prohibitions over time intervals, and show how a standard OWL reasoner can be exploited to carry out run-time monitoring. As we shall see, however, "pure" OWL is not sufficient to deal even with simple monitoring problems. We found it necessary to complement it with its standard rule extension, SWRL (Semantic Web Rule Language[4]), and with special-purpose Java code; basically, this is due to two main difficulties:

- The treatment of time. OWL has no native temporal operators; on some occasions it is possible to bypass the problem by using SWRL rules and built-ins for comparing time instants, but this does not provide full temporal reasoning capabilities. Even the use of an ontology of time (like OWL Time Ontology[5]) does not solve the problem, because its axiomatic specification of temporal entities is very weak, and cannot be sufficiently strengthened within the expressive limits of OWL.
- The need for closed world reasoning. In many contexts, not 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 directly. However, it is often possible to simulate closed world reasoning by adding closure axioms, computed by an external routine, at run-time.

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 monitoring and simulation purposes with significant performance improvements with respect to the solution based on the Event Calculus that we presented elsewhere [13]. We propose a hybrid solution based on an application-independent upper ontology of such concepts. This upper ontology is then complemented by two further application-independent components: a set of SWRL rules, and a Java program accessing the ontology through OWL API[6]. Finally, the ontology is adapted to a specific domain by importing an application-dependent OWL ontology, and assertions are used to represent the temporal evolution of an interaction.

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

[1] http://http://www.w3.org/TR/owl2-overview/
[2] http://hermit-reasoner.com/
[3] http://clarkparsia.com/pellet
[4] http://www.w3.org/Submission/SWRL/
[5] http://www.w3.org/TR/owl-time/, http://www.w3.org/2006/time.rdf
[6] http://owlapi.sourceforge.net/

specify the normative component of an open interaction system. In Section 3 we specify the algorithms that we use to monitor the temporal evolution of an open interaction system. 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 a use case where the proposed system is used to specify and monitor the interaction among two agents regulated by a set of norms. Finally in Section 6 we compare our approach with other proposals and draw some conclusions.

2. OWL and SWRL

OWL 2 DL is a practical realization of a Description Logic 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 that specify logical relationships between classes, which constitutes the *Terminological Box* (*TBox*); a set of property axioms to specify logical relationships between properties, which constitutes a *Role Box* (*RBox*); and a collection of assertions that describe individuals, which constitutes an *Assertion Box* (*ABox*).

Classes are formal descriptions of sets of objects (taken from a nonempty universe), and individuals can be regarded as names of objects of the universe. Properties can be either *object properties*, which represent binary relations between objects of the universe, or *data properties*, which represent 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*, which express Boolean operations or different types of restrictions on the members of the class.

Through *class axioms* one may specify that subclass or equivalence relationships hold between certain classes, and that certain classes are disjoint. In particular, class axioms allow one to specify the domain and range of a property, and that a property is functional or inverse functional. *Property axioms* allow one to specify that a given property (or chain of subproperties) is a subproperty of another property, that two properties are equivalent, or that a property is reflexive, irreflexive, symmetric, asymmetric, or transitive. Finally, *assertions* allow one to specify that an individual belongs to a class, that an individual is (or is not) related to another individual through an object property, that an individual is (or is not) 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 precondition for exploiting reasoning in practical applications, is limited expressiveness: even in OWL 2 many useful first-order statements cannot be formalized.

Recently certain OWL reasoners, like Pellet and HermiT, 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 are 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. For further details refer to the recent text book on Semantic Web [18].

In what follows we use the short hand notation p: $\mathsf{C} \rightarrow_O \mathsf{D}$ to specify an *object property* p (not necessarily a function) with class C as domain and class D as range, and the notation q: $\mathsf{C} \rightarrow_D \mathsf{T}$ 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. Moreover we assume that all individuals introduced in the ABox are assumed to be asserted being different individuals.

3. Specification and monitoring of an open interaction system

We regard an open interaction system as the concrete realization of one or more *artificial institutions* [14,15,13]. Coherently with this standpoint, the specification of an open interaction system includes:

– a *general meta-model* of artificial institutions, which includes the definition of the basic enti-

ties that are common to every institution (like the concepts of temporal proposition, commitment, institutional power, role, and norm, and the actions necessary for exchanging messages);
- a set of *domain-level models*, concerning the specific institutions relevant to the interaction; these models cover the concepts pertaining to the concrete domain of the interaction (for example the actions of paying and of delivering a product in e-commerce applications, of bidding in electronic auctions, etc.), and the specific powers and norms that apply to the agents playing the roles defined by the institution. In particular the domain dependent concepts may be taken from domain ontologies that have to share with ontology of institution at least some upper ontology concepts, like the class of actions or the class of events.
- an *extensional model*, which represents all the events that are relevant to describing the specific interactions taking place within the open interaction system (e.g., a specific auction session, etc.); these could be actual events that happen during the run-time of a concrete system, or events that are pre-recorded as a possible history of the system for simulation purposes.

In this paper both the general meta-model and the domain-level models will be specified in OWL and SWRL, with the goal of monitoring the fulfilment or violation of obligations and prohibitions. The extensional model is realized as an OWL ABox, which is updated partly by a standard OWL reasoner, and partly by an external Java program which allows us to account for the flow of time, to update the model with a representation of the actions performed by the agents and with a representation of the relevant events that happen in the system, and to derive certain logical consequences under the closed-world assumption. When the system is used for simulation, the set of events and actions 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.

As it partly relies on the closed-world assumption, the system implements a non-monotonic reasoning procedure. However, as we shall see, the closed-world assumption is applied only to certain OWL classes, for which it is reasonable to assume that, in a typical application:

- an individual may become a member of the class at a given time t, but since then it may never leave the class;
- complete knowledge of the member of the class up the current time is available.

For each such class, C, we define the corresponding class KC of all individuals that at time t are known to be instances of C. If the assumption of complete knowledge of the members of class C is correct, at every time t up to the current time the classes C and KC do coincide; therefore all conclusions reached under the closed-world assumption are valid, and will never have to be retracted when further knowledge is acquired.

3.1. Temporal evolution of the ontology

As a whole, the system operates according to a sequence of update cycles, controlled by the Java program. At every update cycle, the explicit representations of the relevant events (i.e., agents' actions or time events) are asserted in the ABox (we assume that the events or actions that happen between two update cycles are queued in a suitable data structure for being subsequently managed by the Java program). Then a reasoner is invoked, to derive consequences on the basis of the OWL axioms and SWRL rules. Finally, further consequences are drawn under the closed-world assumption. When an update cycle is completed, a *stable state* has been reached, in the sense that no further consequences can be derived on the basis of the currently available knowledge. Now a reasoner can be used to deduce the state of obligations and prohibitions, and the agents may perform queries to know their pending obligations or prohibitions and to react to violations or fulfilments.

The external Java program is used to update the ABox, recording the elapsing of time and the actions performed by the interacting agents at run-time, and to implement closed-world reasoning on certain classes (see Section 4.1 for details). The main operations of the external program can be summarized as follows:

1. initialize the simulation/monitoring time t 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. If the ontology is used for monitoring purposes, and given that internal time (i.e., the time as represented in the ontology) is discrete, it will be necessary to wait the actual number of seconds that elapse between two internal instants.

The Java pseudo code corresponding to the algorithm previously described is as follows:

```
t=0;
for each class C
   assert KC $\equiv \{i_1, ...i_n\}$
          with $\{i_1, ...i_n\} = $ retrieve(C);
while t<timeMax {
   assert happensAt(elapse,t);
   for each event $e_n$ in ActionQueue
      assert happensAt($e_n$,t);
   run Pellet reasoner;
   for each class C
      remove equivalent class
               axioms of class KC;
      assert KC $\equiv \{i_1, ...i_n\}$
          with $\{i_1, ...i_n\} = $ retrieve(C);
   run agents queries;
   t=t+1;
}
```

4. The ontology of obligations and prohibitions

In this section we present the OWL ontology that specifies the fragment of the OCeAN metamodel concerning the concepts of temporal proposition, commitment, role, and norm. We specify the class and property axioms that model 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 that are created either by the activation of norms or by the performance of communicative acts, like instances of promising or accepting a proposal. 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 *rewards* [13].

Some general classes of our ontology are introduced to serve as domain or range of the properties used to describe temporal propositions and commitments; these are class Event, class Action and class Agent. In particular, an event may be related through a suitable property to its time of occurrence. Class Action is a subclass of Event, and is the domain of a further property used to represent an action's actor. 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. In the current OWL specification, they can relate a *proposition* to an *interval of time* in two possible ways: we distinguish between *positive temporal propositions*, used to model 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 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:

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

Note that while TemporalProp is defined as the disjoint union of TPPos and TPNeg, the same class is *not* defined as the disjoint union of IsTrue and IsFalse. While a temporal proposition cannot have both truth values, it can have neither, and thus be undefined. To make an intuitive example, the temporal proposition "the payment is done before the end of the month" can become true before the end of the month (if the payment is done), but it will remain undefined if the payment has not be done yet; moreover, if the temporal proposition does not become true, when the end of the month elapses it will become false.

In general, the content of a temporal proposition can be any proposition, describing a state of affairs of any kind, and in particular an event or an action. In the current paper, to keep the treatment simpler we make a more limited use of temporal propositions, and use them only to describe actions. The class TemporalProp is therefore the domain of the following object and data properties, and is further specified with suitable cardinality restrictions:

$\mathsf{hasAction}: \mathsf{TemporalProp} \rightarrow_O \mathsf{Action}$;
$\mathsf{hasStart}: \mathsf{TemporalProp} \rightarrow_D \mathsf{integer}$;
$\mathsf{hasEnd}: \mathsf{TemporalProp} \rightarrow_D \mathsf{integer}$;
$\mathsf{hasDuration}: \mathsf{TemporalProp} \rightarrow_D \mathsf{integer}$;
$\mathsf{TemporalProp} \sqsubseteq {=}1\ \mathsf{hasAction} \sqcap {=}1\ \mathsf{hasStart} \sqcap {=}1 \mathsf{hasEnd} \sqcap {=}1 \mathsf{hasDuration}$.

In general the t_{end} point of all temporal propositions can be deduced as the result of the sum between the t_{start} point and the duration of the interval of the specific temporal proposition. This can be formalized with the following SWRL rule that exploits a built-in to add the duration to the start time of the time interval associated to the temporal proposition:

TemporalProp(?tp) $\wedge$ hasStart(?tp,?tstart) $\wedge$
hasDuration(?tp,?d) $\wedge$
swrlb:add(?tend, ?tstart, ?d) $\rightarrow$
hasEnd(?tp,?tend).

Classes IsTrue and IsFalse are used to keep track of the truth value of temporal propositions. The membership of a temporal proposition relative to these classes cannot be specified by means of OWL axioms, because it involves a temporal comparison (i.e., in the current representation, a comparison between two integers). In order to exploit suitable built-ins, we represent membership relative to IsTrue and IsFalse with two different SWRL rules, depending on the type of temporal proposition.

We use a positive temporal proposition (i.e., a member of class TPPos) to represent the fact that an action does take place in a given interval of time, with starting point t_{start} and end point t_{end} and a duration. We therefore introduce a rule to infer that the truth value of the temporal proposition is true (i.e., the temporal proposition becomes a member of the class IsTrue) if the action described by the temporal proposition is performed between t_{start} (inclusive) and t_{end} (exclusive) of the interval of time associated to the same proposition. The following SWRL rule exploits two built-ins to compare the current time with the extremes of the time interval associated to the temporal proposition:

RuleTPPos1:
happensAt(elapse,?t) $\wedge$ happensAt(?a,?t) $\wedge$
TPPos(?tp) $\wedge$ hasAction(?tp,?a) $\wedge$
hasStart(?tp,?ts) $\wedge$ hasEnd(?tp,?te) $\wedge$
swrlb:lessThanOrEqual(?ts,?t) $\wedge$
swrlb:lessThan(?t,?te) $\rightarrow$ IsTrue(?tp).

We now have to define a rule that, when the time t_{end} of a positive temporal proposition elapses, and the temporal proposition is not true, infers that the temporal proposition is member of the class IsFalse. Here closed-world reasoning comes into play, because we cannot assume that the ontology allows us to infer that an action has not been performed: rather, we want to deduce that an action has not been performed if it cannot be inferred that it has been performed. As a first attempt, we may be tempted to define a class

$\mathsf{NotIsTrue} \equiv \mathsf{TPPos} \sqcap \neg\mathsf{IsTrue}$;

and then use it in a rule like

happensAt(elapse,?te) $\wedge$ hasEnd(?tp,?te) $\wedge$
TPPos(?tp) $\wedge$ NotIsTrue(?tp) $\rightarrow$ IsFalse(?tp).

However, this solution would not work, given that OWL/SWRL reasoners operate under the open world assumption. This means that the conclusion that a temporal proposition is not true can only be reached for those propositions that can be positively proved not to be members of IsTrue. On the contrary, if a temporal proposition is not inferred to be IsTrue by RuleTPPos1, even if its deadline has elapsed it will not be inferred to be IsFalse.

To solve this problem we first assume that our ABox contains complete information on the actions performed up to the current time of the system. This allows us to safely adopt a closed-world assumption 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 action has been performed; therefore, all actions that have been performed up to the current time are explicitly represented in the ontology. We then want to infer that all the temporal propositions, that cannot any longer become true because their deadline has elapsed, are false.

To get this result we perform a form of closed world reasoning on class IsTrue. In principle, it would be possible to solve the problem using an epistemic operator K ("it is known that"), and defining NotIsTrue as:

NotIsTrue $\equiv$ TPPos $\sqcap \neg$ K IsTrue.

At the moment, however, the only available reasoner that handles the K operator, described in [22], deals with the $\mathcal{ALCK}$ Description Logic [9], obtained by adding the K operator to the $\mathcal{ALC}$ Description Logic. However, our ontology uses a much more expressive Description Logic, for which no comparable reasoner is available. 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 guarantee that class KIsTrue is the closure of class IsTrue, we re-define it at every update cycle as equivalent to the enumeration of all individuals that can be proved to be members of IsTrue. This is 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 then 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 the TemporalProp class, and the set of all the 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, the temporal propositions that are members of the class TPNeg and are used to represent the fact that a given action is not performed 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 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)

Temporal propositions have a three-valued logic, because they can be undefined, true, or false. Note however that this logic, as defined by the previous axioms and rules, is *temporally monotonic*, in the sense that if at a certain instant a temporal proposition becomes true (false), then it will be true (false) forever since.

4.2. Commitment

In the *OCeAN* meta-model of artificial institutions, a commitment is used to model a social relationship between a *debtor* a *creditor*, about a *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 to other agents to perform, or not to perform, a certain action during a given interval of time; such commitments can be conditional on the truth of some proposition. Both the condition and the content of a commitment are taken to be temporal propositions, as defined in the previous subsection. In particular, this means that a commitment as such is timeless, as its temporal aspects are specified by its content. For example, a commitment to execute a payment before the end of the month is regarded as a timeless commitment to perform the action described by the temporal proposition "pay from now to the end of the month."

We can regard a commitment to perform an action as a conditional *obligation*, and a commitment not to perform an action as a conditional *prohibition*. More precisely, an obligation will be a commitment whose content is a positive temporal proposition, and a prohibition will be a commitment whose content is a negative temporal proposition. We assume that if an action is neither obligatory nor prohibited, then it is *permitted*.

In terms of truth values, both the conditions and the content can be undefined, true, or false. Intuitively, a commitment becomes *pending* when its condition becomes true and its content is still undefined. Then, if its content also becomes true, the commitment is fulfilled, and if its content becomes false the commitment is violated. As the logic of temporal propositions is temporally monotonic, if at a certain instant a commitment becomes fulfilled (violated), it will be fulfilled (violated) forever since. On the contrary, we prefer to consider the state of being pending as a transient state, which is assumed when the condition becomes true (and the content is still undefined) and holds until the commitment becomes either fulfilled or violated. In our previous works [13] we introduced a further state, called *precommitment*, to deal with the semantics of directive communicative acts, but this is not relevant to the current paper.

To deal with commitments, we introduce in the ontology the class Commitment, disjoint from Event, Agent and TemporalProp.

$\mathsf{Commitment} \sqcap \mathsf{Agent} \sqsubseteq \bot;$
$\mathsf{Commitment} \sqcap \mathsf{Event} \sqsubseteq \bot;$
$\mathsf{Commitment} \sqcap \mathsf{TemporalProp} \sqsubseteq \bot;$

Class Commitment is the domain of the following object properties:

$\mathsf{hasDebtor}\colon \mathsf{Commitment} \rightarrow_O \mathsf{Agent};$
$\mathsf{hasCreditor}\colon \mathsf{Commitment} \rightarrow_O \mathsf{Agent};$
$\mathsf{hasContent}\colon \mathsf{Commitment} \rightarrow_O \mathsf{TemporalProp};$
$\mathsf{hasCondition}\colon \mathsf{Commitment} \rightarrow_O \mathsf{TemporalProp};$
$\mathsf{hasSource}\colon \mathsf{Commitment} \rightarrow_O \mathsf{Norm};$
$\mathsf{Commitment} \sqsubseteq \exists\ \mathsf{hasDebtor} \sqcap \exists\ \mathsf{hasCreditor} \sqcap =1\mathsf{hasContent} \sqcap =1\mathsf{hasCondition}.$

The hasSource property will be used to keep track of the norm that generated a commitment, as explained in Section 4.3. The debtor of a commitment is constrained to be the actor of the action that is the content of the commitment, as expressed by the following subproperty axiom:

$\mathsf{hasContent} \circ \mathsf{hasAction} \circ \mathsf{hasActor} \sqsubseteq \mathsf{hasDebtor}.$

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 always 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 to specify that a given commitment is:

- *fulfilled*, when its content is true (we assume that in such a case a commitment is fulfilled even if its condition is not true);
- *violated*, when its condition is true and its content is false;
- *pending*, when its condition is true but its content is not yet true or false.

First we introduce classes IsFulfilled, IsViolated, and IsPending as subclasses of Commitment (class IsPending is not specified to be disjoint from IsViolated and IsPending for reasons that will soon be explained):

$\mathsf{IsFulfilled} \sqcup \mathsf{IsViolated} \sqcup \mathsf{IsPending} \sqsubseteq \mathsf{Commitment}$;
$\mathsf{IsFulfilled} \sqcap \mathsf{IsViolated} \sqsubseteq \bot$.

The classes IsFulfilled and IsViolated are then specified by the following axioms:

AxiomIsFulfilled:
$\mathsf{IsFulfilled} \equiv \exists\ \mathsf{hasContent.IsTrue}$;

AxiomIsViolated:
$\mathsf{IsViolated} \equiv (\exists\ \mathsf{hasContent.IsFalse}) \sqcap (\exists\ \mathsf{hasCondition.IsTrue})$.

We now want to specify when a commitment is pending. As we have said before, this is the case when the condition is true, and the content is neither true not false. To establish that a temporal proposition is neither true not false, we exploit again classes NotKIsTrue and NotKIsFalse which allow us to reason under the closed-world assumption. We define the following axiom:

AxiomIsPending:
$\mathsf{IsPending} \equiv (\exists\ \mathsf{hasContent.NotKIsTrue}) \sqcap (\exists\ \mathsf{hasContent.NotKIsFalse}) \sqcap (\exists\ \mathsf{hasCondition.IsTrue})$.

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; however, the commitment will cease to be a member of IsPending as soon as the update cycle is completed.

The previous specifications do not distinguish between obligations and prohibitions, because the axioms can be applied to both positive and negative temporal propositions. Of course, the circumstances in which an obligation and a prohibition are fulfilled or violated are different, but this difference is dealt with by the axioms and rules regulating the membership of temporal propositions to IsTrue and IsFalse. This means, for example, that if action X is executed, an obligation to do X will be fulfilled, and a prohibition to do X will be violated.

4.3. Norms and Roles

In OCeAN, norms are introduced to model commitments (i.e., either obligations or prohibitions) that, contrary to those created at run time by the performance of communicative acts, are brought about by an institutional setting that is 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 typically specified at design time, when it is impossible to know which individual agents will interact in the system, one of their distinctive features is that the debtor and/or creditor of a commitment generated by a norm have to be specified with reference to the *roles* played by the agents within the institutional entity to which the norm belongs. At run time, when a norm becomes *active* (i.e., when a pre-defined activating event happens), the actual debtor and creditor of the commitment generated by a norm are computed on the basis of the roles played by the agents in the relevant institutional entity at that moment.

Another important aspect of norms is that to enforce their fulfilment in an open system, it must be possible to specify *sanctions* or *rewards.* In [12] 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 by a temporal proposition, whereas passive sanctions can be represented as new institutional 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 [21] is called *enforcement norm*) may oblige the enforcer to punish the violation. In this paper we do not submit a model of the notion of *power*, and thus leave the treatment of passive sanctions to a future occasion.

4.3.1. Role

Typically, artificial institutions define 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 and 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 *run01* of a given auction, or an *employee* of IBM, or the *debtor* of a commitment to pay a given amount of money to some beneficiary (who will typically, but not necessarily, be the *creditor* of the same commitment).

We introduce class RoleName to represent the set of possible role labels (like auctioneer, employee, etc.), and class InstEntity to represent the institutional entity within which a given role is played. Elements of class Role are used to reify the role played by an agent in the context of a given institutional entity. Those classes are related by the following object properties:

isPlayedBy: Role $\rightarrow_O$ Agent;
hasRoleName: Role $\rightarrow_O$ RoleName;
isRoleOf: Role $\rightarrow_O$ InstEntity;

4.3.2. Norms

Summarizing, a norm has: a *reference* to an institutional entity, within which the norm is to be applied; 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:

isNormOf: Norm $\rightarrow_O$ InstEntity;
hasNormDebtor: Norm $\rightarrow_O$ RoleName;
hasNormCreditor: Norm $\rightarrow_O$ RoleName;
hasNormContent: Norm $\rightarrow_O$ TemporalProp;
hasNormCondition: Norm $\rightarrow_O$ TemporalProp;
hasActivation: Norm $\rightarrow_O$ Event;
hasASanction: Norm$\rightarrow_O$ TemporalProp;
hasPSanction: Norm $\rightarrow_O$ Power.

When a norm is activated, that is when the event described through the hasActivation property is happened, it is necessary to create as many commitments as there are agents playing the role associated to the debtor 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 standard OWL axiom or SWRL rules. 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 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. Use case: vehicle repair contract

In this section we show how it is possible to describe the state of an interaction system and to monitor its evolution in time using the proposed model. The first required step is to integrate the ontology defined in the previous sections with an application-dependent ontology, and to insert in the ABox a set of individuals for representing specific institutional entities, roles, norms, commitments, and temporal propositions. In a real application the commitments and their temporal propositions will be created by a proper component as consequence of the performance of communicative acts (defined in the *OCeAN* agent communication library [13]) or by the activation of norms. If the system is used for monitoring purposes, we assume

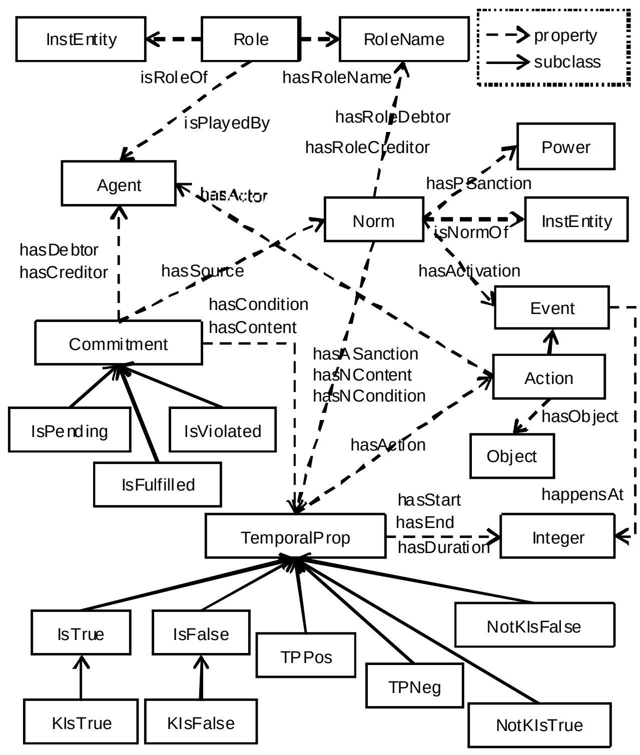

Fig. 1. Graphical representation of the ontology.

that there is a way of mapping the actions that are actually executed onto their counterparts in the ontology.

The case study that we decide to adopt is the vehicle repair contract described in [20]. The scenario is as follows: a *repair contract* regulate the interactions between a client and a vehicle repair company and specifies details concerning a particular repair. In an initial phase the interaction between the two participants is devoted to the definition of the properties of a specific repair contract that is characterized by the type of the repair, the price, and the deadlines for the performance of the actions regulated by the contract. This phase consists in the exchange of proper communicative acts between the client and the repair company, like request, propose, accept, and reject. The second phase of the interaction is devoted to the execution of the contract and it is described as follows: the client agent has to send the vehicle to the repair company within *k1* days from the acceptance of the contract. The repair company then waits for the vehicle to arrive, failing which it sends two reminders to client. If the vehicle fails to arrive, it takes an offline action. As per the contract, if the vehicle arrives the repair company is obligated to assess the damage, repair the vehicle, and send a report to client within *k2* days from the reception of the vehicle. On receiving the report, the client is obligated to send payment to the repair company within *k3* days from the reception of the report. If the payment is not sent, the repair company sends two reminders to the client and then takes an offline action. If the payment is sent the client has to pick-up the vehicle within *k4* days from the reception of the report.

The execution of the contract is regulated by a set of norms that become active when the contract is accepted. When the norms become active they generate a set of commitments and temporal propositions for the agent playing the role of client or the role of repair company in a specific contract. In this example the repair contract is the institutional entity where the norms and the roles are defined and they are represented as follows:

RoleName(repairCompany); RoleName(client);
InstEntity(vRepairContract1);
Role(client1); Role(repairCompany1);
hasRoleName(client1,client);
hasRoleName(repairCompany1, repairCompany);
isRoleOf(client1,vRepairContract1);
isRoleOf(repairCompany1,vRepairContract1);

The specific agents involved in the execution of the specific contract vRepairContract1 are:

Agent(clAgent); Agent(rcAgent);
isPlayedBy(client1,clAgent);
isPlayedBy(repairCompany1,rcAgent);

The norm for formalizing the obligation for the agent playing the role of *client* to send the vehicle to the agent playing the role of *repair company* within *k1* days from the acceptance of the contract is described with the following assertions, we assume that *k1* in the specific accepted contract vRepairContract1 is equal to 3 days:

Norm(norm1);
isNormOf(norm1,vRepairContract1);
hasNormDebtor(norm1, client);
hasNormCreditor(norm1, repairCompany);
hasNormContent(norm1, tpSendVehicle1);
hasNormCondition(norm1,tpTrue);
hasActivation(norm1,acceptContract1);

This norm specification refers to two application dependent actions: the “send vehicle” action, in the temporal proposition used to describe the content of the norm, and the “accept contract” action, in the condition for the activation of the norm. As consequence the ontology described in the previous sections has to be integrated with new classes and property, in order to be able to represent those actions. The actions introduced up to here have an actor, usually they have also a *recipient* and an *object*; but for simplicity in this example we represent the various actions with individuals inserted in the ABox having only the hasActor property as described in the following assertions:

Action(acceptContract1);
hasActor(acceptContract1, clAgent);
Action(sendVehicle1);
hasActor(sendVehicle1, clAgent);
TPPos(tpSendVehicle1);
hasAction(tpSendVehicle1, sendVehicle1);
hasDuration(tpSendVehicle1, 3);

The temporal proposition tpSendVehicle1, like all the temporal propositions that are in the content of a unconditional norm, has an interval that starts when the norm is activated. The following SWRL rule can be used to deduce the t_{start} point of the temporal propositions used in the content of this first type of norms:

hasActivation(?n, ?e) $\wedge$ happensAt(?e,?t) $\wedge$
hasNormCondition(?n, tpTrue) $\wedge$
hasNormContent(?n,?tp) $\rightarrow$ hasStart(?tp,?t).

The commitment generated by the activation of norm1 is represented as follows:

Commitment(c1); hasSource(c1,norm1);
hasDebtor(c1, clAgent);
hasCreditor(c1,rcAgent);
hasContent(c1,tpSendVehicle1);
hasCondition(c1, tpTrue);

If the sendVehicle1 action will be performed by agent clAgent within the interval of time of the temporal proposition tpSendVehicle1 the temporal proposition becomes member of the IsTrue class, and the commitment c1 becomes fulfilled. Otherwise when the t_{end} instant of time of the temporal proposition interval is elapsed the temporal proposition becomes member of the IsFalse class and the commitment becomes violated.

The second norm described in the vehicle repair contract represents the obligation, if the vehicle arrives, for the repair company to assess the damage, repair the vehicle, and send a report to client within *k2* days from the reception of the vehicle. We assume that *k2* in the specific accepted contract vRepairContract1 is equal to 30 days. The actions of assessing the damage and repairing the vehicle have not directly observable effects on the interaction of the agents. As consequence, given that the model presented in this paper has the goal to represent and monitor the external observable interactions among the involved agents, we do not explicitly model those actions. We represent and monitor only the action of sending the report to the client, which has to be performed by the repair company.

Norm(norm2);
isNormOf(norm2,vRepairContract1);
hasNormDebtor(norm2, repairCompany);
hasNormCreditor(norm2, client);
hasNormContent(norm2, tpSendReport1);
hasNormCondition(norm2, tpSendVehicle1);
hasActivation(norm2,acceptContract1);

The temporal proposition in the condition of norm2 has been previously defined because it is also in the content of norm1. The temporal proposition in the content of this norm is defined as follows:

TPPos(tpSendReport1); Action(sendReport1);
hasDuration(tpSendReport, 30);
hasAction(tpSendReport1,sendReport1);
hasObject(sendReport1, report1);
hasActor(sendReport1, rcAgent);

Where the property
hasObject: Action $\rightarrow_O$ Object;
is introduced to explicitly represent the object of a general action, and it will be crucial for the formalization of the next prohibition. The temporal proposition tpSendReport1, like all temporal propositions that are in the content of conditional norms, has an interval that starts when the action referred in the condition of the norm is performed. The following SWRL rules can be used to deduce the t_{start} point of the interval of the temporal propositions used in the content of this second type of norms:

hasNormCondition(?n, ?tpCond) $\wedge$
hasAction(?tpCond, ?a) $\wedge$ happensAt(?a,?t) $\wedge$
hasNormContent(?n,?tpCont) $\rightarrow$
hasStart(?tpCont,?t).

The commitment generated by the activation of norm2 is:

Commitment(c2); hasSource(c2,norm2);
hasDebtor(c2, rcAgent);
hasCreditor(c2,clAgent);

hasContent(c2,sendReport1);
hasCondition(c2, tpSendVehicle1);

A reasonable prohibition that can be introduced in the vehicle repair contract is the prohibition for the repair company to use non original spare parts to repair the vehicle. We assume that a spare part is not original if its brand is different from the brand of the repaired vehicle. To model this prohibition we have to introduce in our application dependent ontology the following new properties. One property is necessary to model the brand of the specific vehicle that has to be repaired on basis of the contract vRepairContract1, whose range is a new class Brand:

hasBrand: InstEntity $\rightarrow_O$ Brand;

Another property is necessary to model the brand of the spare parts used by the repair company and that has to be written in the report that the repair company has to send to the client:

hasSpareBrand: Object $\rightarrow_O$ Brand;

We now introduce an SWRL rule to deduce that if the brand of the vehicle is different from the brand of the spare parts written in the report sent to the client, the send report action performed by the repair company counts as performing a sendBadReport1 action:

Action(sendBadReport1);
hasObject(sendBadReport1, report1);

hasObject(?a, report1) $\wedge$ happensAt(?a, ?t) $\wedge$
hasSpareBrand(report1, ?sBrand) $\wedge$
hasBrand(vRepairContract1, ?vBrand) $\wedge$
differentFrom(?vBrand,?sBrand) $\rightarrow$
happensAt(sendBadReport1, ?t);

The norm that has to be introduced to represent the prohibition to perform the sendBadReport1 action, whose content belongs to the TPNeg class, is formalized as follows:

Norm(norm3);
isNormOf(norm3,vRepairContract1);
hasNormDebtor(norm3, repairCompany);
hasNormCreditor(norm3, client);
hasNormContent(norm3, tpSendBadReport1);
hasNormCondition(norm3, tpSendVehicle1);
hasActivation(norm3,acceptContract1);
TPNeg(tpSendBadReport1);
hasAction(tpSendBadReport1, sendBadReport1);
hasDuration(tpSendBadReport1, 30);

The commitment generated by the activation of norm3 is formalized as:

Commitment(c3); hasSource(c3,norm3);
hasDebtor(c3, rcAgent);
hasCreditor(c3,clAgent);
hasContent(c3,tpSendBadReport1);
hasCondition(c3, tpSendVehicle1);

If report1 is sent by the repair company within the specified interval of time specified in the content of commitment c2, c2 becomes fulfilled, but if the brand of the spare components used to repair the vehicle as it appear in report1 is different from the brand of the repaired vehicle, commitment c3 becomes violated.

Due to space limitation we do not report the other two norms that regulate the execution of the vehicle repair contract but their structure is similar to the structure of the norm formalized in this section.

We created the ontology of the described interaction system with the free, open source ontology editor *Protégé*[7]. We implemented the Java program described in Section 3 using OWL-API library to operate on the ontology, and the source code of the open source OWL 2 reasoner Pellet [8] to reason and query it[9].

6. Conclusions and Related Works

The problem of modelling, using formal languages, norms is widely recognized as a crucial problem by the multiagent community [3,24], and the problem of runtime monitoring those norms is becoming more and more an interesting open question for the multiagent community and for the web service community as demonstrated by recent papers on this topic [10,20]. One of the most relevant contribution of this paper, with respect to our previous works and with respect to other approaches that use other formal languages, is to propose a model to represent and monitor conditional obligations and prohibitions with stating points and deadlines. An significant aspect of the proposed model is that the mentioned limits of using OWL 2 DL for representing and monitoring obligations and prohibitions have been overcame by proposing

[7]http://protege.stanford.edu/
[8]http://clarkparsia.com/pellet
[9]The .owl file with the classes and properties of the model, the .owl file with the formalization of the proposed use case, and the Java program can be found at http://www.people.lu.unisi.ch/fornaran/ontology/ObligationsProhibitionsTP.ht

an hybrid solution formalized using OWL ontologies, SWRL rules, and a Java program.

An important aspect of the approach proposed in this paper with respect to another one that we presented elsewhere based on Event Calculus [13] is that 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 [25,1], or other specific formal languages like the one required by the rule engine Jess [16,6], or a variant of Propositional Dynamic Logic (PDL) used to specify and verify liveness and safety properties of multi-agent system programs with norms [7], or Process Compliance Language (PCL) [17]. A similar advantage can be highlighted if we compare our proposal to use Semantic Web technology for obligations and prohibitions monitoring with other approaches based on augmented transition networks [10] or timed automata with discrete data [20].

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 [23]. 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 [4], 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 [12]) 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 [19] 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 [11].

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 [5].

Indeed this model is still incomplete and 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.

Acknowledgements

The first author is supported by the Swiss State Secretariat for Education and Research SER within the COST action number IC0801 ”Agreement Technologies”. Project number C08.0114, title “Open Interaction Frameworks: A Model Based On Artificial Institutions”

References

[1] A. Artikis, M. Sergot, and J. Pitt. Animated Specifications of Computational Societies. In C. Castelfranchi and W. L. Johnson, editor, *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, pages 535–542. ACM Press, 2002.

[2] I. Berges, J. Bermdez, A. Goi, and A. Illarramendi. Semantic web technology for agent communication protocols. In *The Semantic Web: Research and Applications 5th European Semantic Web Conference, ESWC 2008, Tenerife, Canary Islands, Spain, June 1-5, 2008 Proceedings*, pages 5–18, 2008.

[3] G. Boella, P. Noriega, G. Pigozzi, and H. Verhagen, editors. *Normative Multi-Agent Systems*, number 09121 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[4] J. Bradshaw, P. Beautment, M. Breedy, L. Bunch, S. Drakunov, P. Feltovich, R. Hoffman, R. Jeffers, M. Johnson, S. Kulkarni, J. Lott, A. Raj, N. Suri, and A. Uszok. Making agents acceptable to people. pages 355–400. Springer Berlin / Heidelberg, 2004.

[5] M. Bräuer and H. Lochmann. An Ontology for Software Models and Its Practical Implications for Semantic Web Reasoning. In S. Bechhofer, M. Hauswirth, J. Hoffmann, and M. Koubarakis, editors, *ESWC*, volume 5021 of *LNCS*, pages 34–48. Springer, 2008.

[6] V. T. da Silva. 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, August 2008.

[7] M. Dastani, D. Grossi, J.-J. Meyer, and N. Tinnemeier. Normative multi-agent programs and their logics. In G. Boella, P. Noriega, G. Pigozzi, and H. Verhagen, editors, *Normative Multi-Agent Systems*, number 09121 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[8] F. Dignum and R. Kuiper. Combining dynamic deontic logic and temporal logic for the specification of deadlines. In *HICSS ’97: Proceedings of the 30th Hawaii International Conference on System Sciences*, page 336, Washington, DC, USA, 1997. IEEE Computer Society.

[9] F. M. Donini, M. Lenzerini, D. Nardi, A. Schaerf, and W. Nutt. An epistemic operator for description logics. *Artificial Intelligence*, 200(1-2):225274, 1998.

[10] N. Faci, S. Modgil, N. Oren, F. Meneguzzi, S. Miles, and M. Luck. Towards a monitoring framework for agent-based contract systems. In *CIA ’08: Proceedings of the 12th international workshop on Cooperative Information Agents XII*, pages 292–305, Berlin, Heidelberg, 2008. Springer-Verlag.

[11] C. Felicissimo, J.-P. Briot, C. Chopinaud, and C. Lucena. 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’08)*, 23rd AAAI Conference on Artificial Intelligence, Chicago, IL, Etats-Unis, 2008. AAAI, AAAI Press.

[12] N. Fornara and M. Colombetti. Specifying and enforcing norms in artificial institutions. In M. Baldoni, T. Son, B. van Riemsdijk, and M. Winikoff, editors, *Declarative Agent Languages and Technologies VI 6th International Workshop, DALT 2008, Estoril, Portugal, May 12, 2008, Revised Selected and Invited Papers*, volume 5397 of *LNCS*, pages 1–17. Springer Berlin / Heidelberg, 2009.

[13] N. Fornara and M. Colombetti. *Specifying Artificial Institutions in the Event Calculus*, volume Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models of *Information science reference*, chapter XIV, pages 335–366. IGI Global, 2009.

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

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

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

[17] G. Governatori and A. Rotolo. How do agents comply with norms? In G. Boella, P. Noriega, G. Pigozzi, and H. Verhagen, editors, *Normative Multi-Agent Systems*, number 09121 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[18] P. Hitzler, R. Sebastian, and M. Krötzsch. *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, London, 2009.

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

[20] A. Lomuscio, W. Penczek, M. Solanki, and M. Szreter. Runtime monitoring of contract regulated web services (extended abstract). In *Proceedings of the 9th International Conference on Autonomous Agents and Multi-Agent systems (AAMAS10). Toronto, Canada.*, pages 1449–1450, New York, NY, USA, 2010. ACM.

[21] F. López y López, M. Luck, and M. d'Inverno. A Normative Framework for Agent-Based Systems. In *Proceedings of the First International Symposium on Normative Multi-Agent Systems, Hatfield*, 2005.

[22] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007.

[23] A. Uszok, J. M. Bradshaw, J. Lott, M. Breedy, L. Bunch, P. Feltovich, M. Johnson, and H. Jung. New Developments in Ontology-Based Policy Management: Increasing the Practicality and Comprehensiveness of KAoS. *Policies for Distributed Systems and Networks, IEEE International Workshop on*, 0:145–152, 2008.

[24] G. E. L. van der Torre, G. Boella, and H. Verhagen, editors. *Special Issue on Normative Multiagent Systems*, volume 17 of *Autonomous Agents and Multi-Agent Systems*. Springer Netherlands, August 2008.

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