Representing and Reasoning about Norm-Governed Organisations with Semantic Web Languages*

Joey Sik-Chun Lam, Frank Guerin, Wamberto Vasconcelos, and Timothy J. Norman
{j.lam, f.guerin, w.w.vasconcelos, t.j.norman}@abdn.ac.uk

Department of Computing Science
University of Aberdeen, Aberdeen, AB24 3UE, U.K.

Abstract. There are many significant contributions to the formal study of norm-governed organisations. A key challenge in this endeavour is to provide designers and engineers with means to specify normative aspects of organisations, allowing phenomena such as norm conflict (i.e., an action being simultaneously obliged and prohibited), to be captured and studied. Ideally, any candidate formalism for the specification of organisations should have a sound underlying semantics to guide the verification of desirable properties (e.g., an organisation has no conflict among its norms). In this paper we propose using Semantic Web languages to represent norm-governed organisations. We represent roles and their relationships, and various kinds of normative notions and power, including norms with conditions and deadlines. With this representation we are able to detect norm violations and norm conflict (also pinpointing the source of the conflict in the organisation). By using description logics to specify our ontologies, and Semantic Web standards to capture our rules, our approach benefits from off-the-shelf technologies such as reasoners and rule engines, which we deploy to verify organisations and infer implicit knowledge from them.

1 Introduction

Organisations are working together in e-business, e-markets, or other domains; they have to coordinate with each other and describe their processes in a machine-understandable way to share and re-use information. They are required to work in open environments which are dynamic and unpredictable. This requirement is compatible with the vision of the Semantic Web [6], which aims to support organisations operating in open environments to share knowledge and process information automatically.

A norm-governed organisation can be expressed in terms of a set of agents (the members of a society), a set of constraints on a society, a set of roles that members can play, a communication language, relationships between roles, as well as the structure of a society [3]. Members of a society are autonomous; they may be self-interested and deviate from expected behaviours. To express the expectation that agents' behaviour follows the constraints of a society, we should

* This work is funded by the European Community (FP7 project ALIVE IST-215890).

not restrict the autonomy of those agents. We should rather specify what is permitted, prohibited and obligatory, and other normative relations (such as duty, right, privilege, authority and etc.) that may exist between agents, and leave it to these agents to decide what to do given these constraints. A key challenge in this endeavour is to provide designers and engineers with means to specify normative aspects of organisations, allowing phenomena such as norm conflict (i.e., an action being simultaneously obliged and prohibited), to be captured and studied. Ideally, any candidate formalism for the specification of organisations should have a sound underlying semantics to guide the verification of desirable properties (e.g., an organisation has no conflict among its norms).

In this paper we propose using ontologies and rules to represent norm-governed organisations. Ontologies, partially machine understandable, are playing an important role in automated processes (aka. "intelligent agents") to access information. In particular, ontologies provide formally structured vocabularies that explicate the relationships between different terms, so that intelligent agents can interpret their meaning flexibly yet unambiguously [13]. OWL-DL, a W3C recommendation from 2004, is a decidable fragment of first-order logic and has desirable computational properties for reasoning systems [5]. However, OWL reasoning and editing tools do not support temporal representation, which is essential in the representation of norm-governed systems. To these ends we use the Semantic Web Rule Language[1] (SWRL). SWRL combines OWL and RuleML, and reasoning on SWRL-enable ontologies is supported by existing DL reasoners[2]. SWRL can be used to represent temporal information[3]. Hence our formalism is able to express conditional and temporal aspects of norms.

We will illustrate that ontologies and rules provide a means of implementing an expressive and decidable organisation specification [3, 8] that captures:

- roles and role classification;
- institutionalised powers;
- permissions, prohibitions and obligations of the agents;
- violations that agents perform certain forbidden actions or do not comply with their obligations;
- temporal relationships, the violation of norms could be dependent on temporal constraints such as norm deadlines and activation times.

This paper is organised as follows: In Section 2 we briefly introduce ontologies and rules. In Section 3 we describe the representation of a norm-governed organisation using ontology languages, and show how to detect the violations of norms in Section 4. Section 5 describes how to reason with axioms to detect contradictions between norms. We compare our approach with existing work in Section 6. We conclude in Section 7, point our directions for future work.

2 Ontology and Rules

The OWL-DL [12] ontology language is a variant of $\mathcal{SHOIN}(\mathbf{D})$ [14] Description Logic, which provides constructs for full negation, disjunction, a restricted form

[1] http://www.w3.org/Submission/SWRL/
[2] Pellet and KAON2 support a subset of SWRL called DL-safe rules.
[3] http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTemporalBuiltIns

of existential quantification, and reasoning with concrete datatypes. We make use of the Open World Assumption, which requires that something is false if and only if it can be proved to contradict other information in the ontology. Since we assume a MAS as an open system, its knowledge of the world is incomplete, and the knowledge is extendable. If a formula cannot be proved true or false, we do not draw any conclusion[4].

Formally, an ontology $\mathcal{O}$ consists of a set of *terminology* axioms $\mathcal{T}$ (TBox) and assertional axioms $\mathcal{A}$ (ABox), that is, $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$. An axiom in $\mathcal{T}$ is either of the form $C \sqsubseteq D$ or $C \doteq D$, where C and D are arbitrary concepts (aka. classes in OWL); an axiom in $\mathcal{A}$ is either of the form $C(a)$ (where C is a concept and a is an individual name; a belongs to C), or of the form $R(a, b)$ (where a, b are individual names (aka. instances in OWL) and R is a role name (aka. a datatype or object property in OWL); b is a filler of the property R for a).

The meaning of concepts, roles and individuals is given by an interpretation. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set of individuals (the *domain* of the interpretation) and an *interpretation function* $(\cdot^{\mathcal{I}})$, which maps each atomic concept $\mathsf{CN} \in \mathbf{C}$ ($\mathbf{C}$ is a set of concept names) to a set $\mathsf{CN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each atomic role $R \in \mathbf{R}$ ($\mathbf{R}$ is a set of role names) to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function can be extended to give semantics to concept descriptions. An interpretation $\mathcal{I}$ is said to be a model of a concept C, or $\mathcal{I}$ models C, if the interpretation of C in $\mathcal{I}$ is not empty. Based on this semantics a TBox can be checked for incoherence, i.e., whether there are *unsatisfiable* concepts:

1. A concept A is *unsatisfiable* w.r.t. a terminology $\mathcal{T}$ if, and only if, $A^{\mathcal{I}} = \emptyset$ for all models of $\mathcal{I}$ of $\mathcal{T}$.
2. A terminology $\mathcal{T}$ is *incoherent* if there exists a concept name in $\mathcal{T}$, which is unsatisfiable.
3. An ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is *inconsistent* if it has no models.

OWL DL benefits from many years of DL research, the benefits include well defined semantics, well-studied reasoning algorithms, highly optimised systems, and well understood formal properties (such as complexity and decidability) [4].

The Semantic Web Rule Language (SWRL) extends the set of OWL axioms to include Horn-like rules that can be expressed in terms of OWL concepts and that can reason about OWL individuals. SWRL provides deductive reasoning capabilities that can infer new knowledge from an existing OWL knowledge base. However, OWL DL extended with SWRL is no longer decidable. To make the extension decidable, Motik et al. [21] proposes DL-safe rules where the applicability of a rule is restricted to individuals explicitly named in a knowledge base (KB). For example:

$$\mathsf{parent(x,y)} \wedge \mathsf{brother(y,z)} \wedge \mathcal{O}\mathsf{(x)} \wedge \mathcal{O}\mathsf{(y)} \wedge \mathcal{O}\mathsf{(z)} \rightarrow \mathsf{uncle(x,z)}$$

[4] We can reason an inconsistent ontology by tolerating the contradictions. A formula is *undefined* (or *undertermined*) if it entails neither true nor false; a formula is *overdefined* (or *over-determined*) if it entails both true and false. In the case of undefined permission axioms, we can model the axioms as *weak permission*. We will further discuss the distinction between weak permissions and strong permission in future research.

where $\mathcal{O}(x)$ for each explicitly named individual x in the ontology. $\mathcal{O}$ is not a concept from the OWL DL ontology. Hence, DL-safe rules are SWRL rules that are restricted to known individuals.

We use SWRL to specify temporal constraints and rules found in our ontologies in terms of the temporal model. Using SWRL's temporal built-in extension facility, we are able to express complex temporal constraints in rules. SWRLTab[5], an editor for SWRL rules in Protégé-OWL[6], contains the temporal built-ins. It defines a set of built-ins that can be used in SWRL rules to perform temporal operations. These built-ins are defined in the SWRL Temporal Ontology[7]. It has the default prefix temporal. Some examples of temporal operators are temporal:before(x, '2008-12-22'), temporal:after(x,y) and temporal:during(2, x, temporal:Years)[8].

3 Norm-Governed Organisations

In this section, we describe how to represent roles, role classification, norms, power, and task descriptions using OWL and SWRL. Norms are further described with conditions and deadlines.

3.1 Roles and Role Classification

The ontology we propose in this paper models the concepts of an agent, its role classification(s), restrictions on roles (such as mutually exclusive roles, cardinality, prerequisite roles), and other aspects of the organisation. Roles are modeled in a classification to reflect the generalisation of role descriptions. Sub-roles inherit the properties from the super-roles; the properties of a sub-role override those of its super-roles. Maximum and minimum cardinality restrictions can be used to restrict the number of roles that an agent can take, and the number of agents that can be assigned to a particular role[9], and the number of roles a permission can be assigned to. The disjointness axioms in ontologies can represent separation of duty restrictions, which aim to prevent conflict of interests that arise when an agent gains permissions associated with conflicting roles (roles that cannot be assigned to the same agent). We now give an example specification to illustrate this. In Figure 1, a role classification is shown. The role Staff is the most general role; the role AccountingManager (sub-role), is more specific than Manager (super-role). Sub-roles inherit the properties from super-roles. For example, Staff are obliged to work from 9am to 5pm during weekdays; its sub-roles inherit this obligation. The properties of a sub-role override those of its super-role (see axioms 4 and 5 below). For the cardinality restrictions, we can model that only one agent can fill the role of the general manager (see axioms 11 and 12 below); a member of a staff works in exactly one department (see axiom 6 below). The range of worksIn is Department. An example of mutually exclusive roles is that a department manager cannot be a general manager simultaneously

[5] http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTab
[6] http://protege.stanford.edu/overview/protege-owl.html
[7] http://swrl.stanford.edu/ontologies/built-ins/3.3/temporal.owl
[8] Currently, no DL reasoner supports this temporal built-ins reasoning.
[9] Possibly the most useful of these is the distinction between roles which can only have one agent assigned to them, and roles which accept many agents simultaneously.

(see axiom 8 below). An example of separation of duty is that, a staff submitting a project proposal cannot be the staff who approves the proposal (see axiom 9 below). Prerequisite roles means that a person can be assigned to role $r1$ only if the person already is assigned to role $r2$ (see axiom 10).

(1) Programmer $\sqcup$ Manager $\sqsubseteq$ Staff
(2) DeptManager $\sqcup$ GeneralManager $\sqsubseteq$ Manager
(3) AccountingManager $\sqsubseteq$ DeptManager
(4) Manager $\sqsubseteq \exists$ isPermitted.($\exists$ employs.Staff)
(5) AccountingManager $\sqsubseteq \exists$ isPermitted.($\exists$ employs.AccountingStaff)
(6) Staff $\sqsubseteq =1$ worksIn
(7) range(worksIn) = Department
(8) DeptManager $\sqsubseteq \neg$ GeneralManager
(9) Staff(x) $\wedge$ ProjectProposal(p) $\wedge$ submits(x,p) $\wedge$ approves(x,p) $\wedge \mathcal{O}$(x) $\wedge \mathcal{O}$(p) $\rightarrow$ owl:Nothing(x)
(10) AccountingManager $\sqsubseteq \exists$ prerequisites.Accountant
(11) GeneralManager $\sqsubseteq =1$ takenBy
(12) range(takenBy) = Agent
(13) domain(takenBy) = Role
(14) domain(worksIn) = Staff

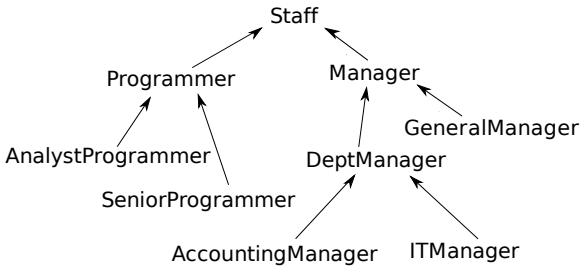

Fig. 1: Roles and a Role Classification

In our formalisation we do not attempt to be prescriptive – we are aware that there could be many other alternative formalisations. We merely aim to illustrate the possibilities of Semantic Web languages for representing organisations.

3.2 Normative Notions and Institutional Power

A number of studies have proposed models for the specification of norms. Some models focus principally on formalising the three basic deontic notions of obligations, prohibitions and permissions, so that it is possible to reason about agents' behaviour and detect the violation of norms; additionally verification is possible using formal theorem provers.

We firstly describe norms concerning agents performing some action Act. The norm on the execution of Act is timeless, that is, the norm is active all the time. A simple and intuitive syntax for unconditional permissions and prohibitions is given in [25]:

PERMITTED(a DO A), FORBIDDEN(a DO A)
We choose this notation here for its simplicity. In OWL, the norms are modeled as:

Agent $\sqsubseteq \exists$ isPermitted.Act

Agent $\sqsubseteq \exists$ isProhibited.Act, where Agent and Act are OWL concepts, isPermitted and isProhibited are OWL object properties; their domain and range is Agent and Act respectively.

Permissions allow the agent to achieve a state of affairs or perform an action. Permission is distinct from power because a member may be empowered to do something even though he is not permitted. The following axiom and rule mean that a finance secretary has permission to query the finance account in the department which the secretary works in.

QueryFinanceAcctAct $\doteq \exists$ queries.FinaceAccount

FinanceStaff(fs) $\wedge$ worksIn(fs,fdept) $\wedge$ hasAccount(fdept,facct) $\wedge$ QueryFinanceAcctAct(act) $\wedge$ queries(act,facct) $\wedge\mathcal{O}$(fs) $\wedge\mathcal{O}$(fdept) $\wedge\mathcal{O}$(facct) $\wedge\mathcal{O}$(act) $\rightarrow$ isPermitted(fs,act)

Prohibitions forbid the agent from achieving a state of affairs or performing an action. The following axiom means that a finance staff is prohibited from approving travel request forms.

ApproveTravelAct $\doteq \exists$ approvesTravelReq.TravelRequestForm

FinanceStaff $\sqsubseteq \exists$ isProhibited.ApproveTravelAct

An obligation indicates some act has to be done. For example, 'staff are obliged to work in weekdays from 9am to 5pm'. This obligation is always true.

Staff $\sqsubseteq \exists$ isObliged.($\exists$ works.($\exists$ hasDays.Weekdays $\sqcap \exists$ hasHour.OfficeHour))

Weekdays $\doteq$ {Monday Tuesday Wednesday Thursday Friday}

OfficeHour $\doteq \exists$ starts.{"09:00:00"^^$\langle$ xsd:time$\rangle$} $\sqcap \exists$ ends.{"17:00:00"^^$\langle$ xsd:time$\rangle$}

It is common to specify a time-limit or a condition for obligations, so that we can test agents' compliance over a finite run of a system. It is no good for an agent to promise something and deliver it "eventually", if there is no upper bound on the time taken. We will describe these constraints in the next section.

We now model the relations between the three basic notions; the relations can be equivalence, compatibility or incompatibility (or conflict). The following rules list some of these relations [26].

1. If an act is permitted, then it is not prohibited.
   isPermitted(x,act) $\wedge$ isProhibited(x,act) $\wedge\mathcal{O}$(x) $\wedge\mathcal{O}$(act) $\rightarrow$ owl:Nothing(x)
2. If an act is obligatory, then it is permitted.
   isObliged(x,act) $\wedge\mathcal{O}$(x) $\wedge\mathcal{O}$(act) $\rightarrow$ isPermitted(x,act)
3. If an act is obligatory, then it is not prohibited.
   isObliged(x,act) $\wedge$ isProhibted(x,act) $\wedge\mathcal{O}$(x) $\wedge\mathcal{O}$(act) $\rightarrow$ owl:Nothing(x)
4. If a prohibited act is performed then there is a violation.
   performed(x,act) $\wedge$ isProhibited(x,act) $\wedge\mathcal{O}$(x) $\wedge\mathcal{O}$(act) $\rightarrow$ violated(x,act)

Note that compared to deontic logic, here these deontic notions are being given a different interpretation by description logic. For example in deontic logic we could say that "obliged" is equivalent to "not permitted not to", however in description logic we cannot express this. SWRL does not allow us to negate the atoms within the scope of isPermitted(...).

In this paper we define power as the ability of an agent to bring about creation of, or changes in, facts in the knowledge base. An action is valid at a point in time if and only if the agent that performed that action had the institutional power to perform it at that point in time [3] (cf. the importantly different notional of institutional power characterised by Jones and Sergot [15]). For example we say that that action 'manager x fires staff y' is valid if x is empowered to fire a staff at that time, therefore y is no longer a staff. Otherwise, it is an invalid action due to its lacking of institutional power. In OWL, we model 'power' as an object property hasPower; its domain and range is Agent and Act respectively. The following axiom means a manager has power to fire a staff.

$$\mathsf{Manager} \sqsubseteq \exists\, \mathsf{hasPower}.(\exists\, \mathsf{fires}.\mathsf{Staff})$$

When a person is fired, we have to update the ABox to state the person is no longer a staff in the company. The ABox is only updated if an agent has a power to perform some action and has performed that action; otherwise nothing is changed in the ABox.

3.3 Norm with Conditions and Deadline

Norms can be conditional or can have temporal constraints, that is, they establish relationships between time-points or events or they hold periodically [10]. We now show how our approach captures these normative notions.

Conditional Norms The norm concerning an action A is conditional under some circumstance C. A simple example is again given in [25]; these norms are defined as:

OBLIGED($(a$ DO $A)$ IF C), PERMITTED($(a$ DO $A)$ IF C),
FORBIDDEN($(a$ DO $A)$ IF C)

In OWL, we can place additional restrictions to axioms to introduce the condition of axioms to be applied.

$$\mathsf{Agent} \sqsubseteq \exists\, \mathsf{isPermitted}.(\mathsf{Act} \sqcap \exists\, \mathsf{hasCond}.\mathsf{Cond})$$
$$\mathsf{Agent} \sqsubseteq \exists\, \mathsf{isProhibited}.(\mathsf{Act} \sqcap \exists\, \mathsf{hasCond}.\mathsf{Cond})$$
$$\mathsf{Agent} \sqsubseteq \exists\, \mathsf{isObliged}.(\mathsf{Act} \sqcap \exists\, \mathsf{hasCond}.\mathsf{Cond})$$

For example, staff is permitted to take sick leave with the doctor's approval, where DoctorApproval is the condition. This is formalised as:

$$\mathsf{Staff} \sqsubseteq \exists\, \mathsf{isPermitted}.(\exists \mathsf{takes}.\mathsf{SickLeave} \sqcap \exists\, \mathsf{hasCond}.\mathsf{DoctorApproval})$$

Another example, staff is prohibited to take a elevator or escalator if there is a fire. This is formalised as:

$$\mathsf{Staff} \sqsubseteq \exists\ \mathsf{isProhibited}.(\exists \mathsf{takes}.(\mathsf{Elevator} \sqcup \mathsf{Escalator}) \sqcap \exists\ \mathsf{hasCond}.\mathsf{OnFire})$$

Conditional Norms with deadlines. This conditional norm is defined with a deadline, which can be either an absolute or a relative deadline. A simple example is again given in [25]:

PERMITTED(a DO A BEFORE D), FORBIDDEN(a DO A AFTER D)

For conditional norms with absolute deadlines, we can represent them as follows:

$$\mathsf{Agent} \sqsubseteq \exists\ \mathsf{isObliged}.(\mathsf{Act} \sqcap \exists\ \mathsf{before}.\mathsf{Deadline})$$
$$\mathsf{Agent} \sqsubseteq \exists\ \mathsf{isProhibited}.(\mathsf{Act} \sqcap \exists\ \mathsf{after}.\mathsf{Deadline})$$

For example, an ITStaff is obliged to update the web server before a deadline.

$$\mathsf{ITStaff} \sqsubseteq \exists\ \mathsf{isObliged}.(\exists \mathsf{upgrades}.\mathsf{WebServer} \sqcap \exists \mathsf{before}.\mathsf{Deadline})$$

Note that before and after are OWL object properties; their domain and range is Act and xsd:dateTime respectively. To detect whether the current date/time is before or after the deadline, a DL reasoner has to be extended to reason with this temporal knowledge.

For conditional norms with a relative deadline, we use SWRL to specify temporal constraints and rules found in our ontologies in terms of the temporal model. SWRL Temporal Ontology[10] is a SWRL built-in extension to address temporal relationships and reasoning. For example, the department manager is obliged to approve travel request forms. The deadline to approve a form should be five days after the form is submitted. We capture this via the rule:

$$\mathsf{ApproveTravelRequestTask} \doteq \exists\ \mathsf{approvesTravelReq}.\mathsf{TravelRequestForm}$$
$$\mathsf{DeptManager(mngr)} \wedge \mathsf{ApproveTravelRequestTask(task)} \wedge \mathsf{TravelRequestForm(form)}$$
$$\wedge\ \mathsf{submittedDate(form,date)} \wedge \mathsf{submittedTo(form,\ mngr)} \wedge \mathcal{O}\mathsf{(mngr)}$$
$$\wedge\ \mathsf{CurrentDateTime(now)} \wedge \mathcal{O}\mathsf{(task)} \wedge \mathcal{O}\mathsf{(form)} \wedge \mathcal{O}\mathsf{(date)} \wedge \mathcal{O}\mathsf{(now)} \wedge$$
$$\mathsf{swrl{:}addDayTimeDurationToDate(deadline,date,xsd{:}dayTimeDuration(\text{“}P5D\text{”}))}$$
$$\rightarrow \mathsf{isObliged(mngr,task)} \wedge \mathsf{before(task,deadline)}$$

The ontology and rules describe static knowledge; i.e., it can give a snapshot of the state of the agent system at some time. When agents are acting at run time, we assume there is a program which updates the ABox appropriately to record performed actions and performed date time; for example $\mathsf{Agent} \sqsubseteq \exists\ \mathsf{performed}.(\mathsf{Action} \sqcap \exists\ \mathsf{perfomedTime}.\mathsf{DateTime})$.
The domain and range of perfomedTime is Act and xsd:dateTime respectively. The compliance with obligations can thus be checked by querying the ABox of the ontology. For example, when agents perform their obligatory actions, their obligation axioms are removed from the ABox (i.e., the information of the obligatory action does not exist in the ABox anymore); if agents perform their prohibited actions, the violations of their behaviours are updated in the ABox (see rule (4) in Section 3.2).

[10] http://swrl.stanford.edu/ontologies/built-ins/3.3/temporal.owl

4 Detection of Violations

We have described how to define restrictions on the behaviours of agents. We now describe how we can detect the violation of these restrictions. If an agent violates an act which is defined in the norms, then we update the ABox, i.e., violated(agent,act) is added to the ABox. The domain and range of violated is Agent and Act respectively.

It is easy to detect the violation of prohibitions in the ontology. The prohibition axioms are of the form $\mathsf{Agent} \sqsubseteq \exists\ \mathsf{isProhibited.Act1}$; the axioms which record agents performed some action are of the form $\mathsf{Agent} \sqsubseteq \exists\ \mathsf{performed.Act2}$. If $\mathsf{Act2} \sqsubseteq \mathsf{Act1}$, then there is a violation; if $\mathsf{Act1} \sqsubseteq \mathsf{Act2}$, then we need to check whether the instance of Act2 is equivalent to that of Act1. For example, finance staff is prohibited from purchasing computer equipment. If a staff working in the finance department purchased a printer which is a subclass of computer equipment, then a DL reasoner can infer the implicit information that the staff is prohibited from purchasing the printer; hence there is a violation.

To detect the violation of obligations which are always valid, we query the ABox corresponding to the obligation axiom. For example,

$\mathsf{Manager} \sqsubseteq \exists\ \mathsf{isObliged.AttendMeetingAct}$
$\mathsf{AttendMeetingAct} \doteq \exists\ \mathsf{attends.AGMeeting}$

We check if attends(attendAct, meeting342) and performed(agentX,attendAct) are in the ABox, where $\mathsf{attendAct} \in \mathsf{AttendMeetingAct}$, $\mathsf{meeting342} \in \mathsf{AGMeeting}$, $\mathsf{agentX} \in \mathsf{Manager}$. If there are no such axioms, it means agentX did not attend meetingJuly; and hence there is a violation.

The conditional obligation axioms are of the form $\mathsf{Agent} \sqsubseteq \exists\ \mathsf{obliged.}(\mathsf{Act} \sqcap \mathsf{Cond})$. If there exists an instance of the concept Cond, and no axiom in the ABox such that performed(agentX, actionX) where $\mathsf{agentX} \in \mathsf{Agent}$ and $\mathsf{actionX} \in \mathsf{Act}$, then the agent agentX violates the obligation.

The conditional obligation with absolute deadline axioms are of the form $\mathsf{Agent} \sqsubseteq \exists\ \mathsf{obliged.}(\mathsf{Act} \sqcap \exists\ \mathsf{before.Deadline})$. The following SWRL rule checks if the agent performs the obligation after the required deadline:

$\mathsf{Agent(ag)} \wedge \mathsf{Act(act)} \wedge \mathsf{isObliged(ag,act)} \wedge \mathsf{performed(ag,act)} \wedge \mathsf{performedTime(act,time)}$
$\wedge\ \mathsf{before(act,deadline)} \wedge \mathsf{temporal{:}after(time,deadline)} \wedge \mathcal{O}(\mathsf{ag}) \wedge \mathcal{O}(\mathsf{act}) \wedge \mathcal{O}(\mathsf{time}) \wedge \mathcal{O}(\mathsf{deadline})$
$\rightarrow\ \mathsf{violated(ag,act)}$

If the agent does not perform the obliged act, then performed(?ag,?act) does not exist in the ABox. While SWRL does not support negated atoms, we cannot state if there does not exist performed(?ag,?act) in the ABox when the deadline is passed. To address this limitation, we use a SPARQL query to check this after the deadline. If the query returns empty set, then agent violates the obligation.

```
PREFIX org:⟨http://www.csd.abdn.ac.uk/∼jlam/organisation.owl⟩
SELECT ?agent ?act
WHERE { ?agent org:performed ?act }
```

5 Reasoning

One advantage of OWL ontologies is the availability of tools that can reason with and about them. OWL DL is a version of OWL retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time) [5]. Most ontology designers find it difficult to understand the logical meaning and potential statements inferred in description logics, including OWL DL [22]. Contradictions between axioms can occur easily, and this is also a concern for norms in agent systems. Kollingbaum and Norman [19], for example, use the term *conflict* for situations when an action is simultaneously permitted and prohibited, while the term *inconsistency* is applied to situations when an action is both obliged and prohibited.

In the following we show two examples of conflict and inconsistency. The issue we would like to emphasise here is that the clash of norms might not be immediately obvious to a knowledge engineer – there are no two norms that directly clash – the clash is detected only after the reasoner has inferred implicit information in the ontology. To detect these contradictions in the ontology, we make use of existing DL reasoners such as Pellet[11] and KAON2[12] which support DL-safe subset [21] of SWRL rules. For a large ontology with hundreds of axioms, it is useful to pinpoint which axioms actually caused the contradictions. We make use of existing ontology debugging techniques [18, 20] which not only detect unsatisfiable concepts, but also pinpoint the set of problematic axioms for the contradictions.

Example 1. This is a conflict example. We assume an ontology which contains all axioms introduced above and the following additional axioms. By using DL reasoners and debugging techniques, we conclude that the concept Staff is unsatisfiable, and the following six axioms cause the problem.

(1) Staff $\sqsubseteq \exists$ isProhibited.WasteResources
(2) Staff $\sqsubseteq \exists$ isPermitted.PrintDocuments
(3) Papers $\sqsubseteq$ Resources
(4) isPermitted(x,act) $\wedge$ isProhibited(x,act) $\wedge\mathcal{O}$(x) $\wedge\mathcal{O}$(act) $\rightarrow$ owl:Nothing(x)
(5) WasteResources $\doteq \exists$ consumes.Resources
(6) PrintDocuments $\sqsubseteq \exists$ consumes.Papers

When a smaller subset of axioms is produced as the cause of contradiction, it is easier for the knowledge engineer to understand the reason for contradictions. The above axioms state that staff are permitted to print documents, which implies consuming papers; at the same time, staff are prohibited from wasting resources, which includes consuming papers. This results in a conflict.

[11] http://pellet.owldl.com
[12] http://kaon2.semanticweb.org/

Example 2. This is an inconsistency example. Similarly, by using reasoners and debugging techniques, we obtain the set of axioms which cause the inconsistency and infer the reason: a manger is obliged to approve travel request forms, a department manager is a type of manager. However, a department manager is prohibited from approving the travel request form which is requested by himself. Using reasoners, the following axioms causing the inconsistency are listed:

(7) Manager $\sqsubseteq \exists$ isObliged.ApprovalTravelAct
(8) DeptManager $\sqsubseteq$ Manager
(9) ApprovalTravelAct $\doteq \exists$ approvesTravelReq.TravelRequestForm
(10) DeptManager(mng) $\wedge$ TravelRequestForm(form) $\wedge$ ApproveTravelAct(act) $\wedge$ approves-TravelReq(act,form) $\wedge$ requestedFrom(form,mng) $\wedge$ performs(mng,act) $\wedge\mathcal{O}$(mng) $\wedge\mathcal{O}$(form) $\wedge\mathcal{O}$(act) $\rightarrow$ SelfApprovalTravelAct(act)
(11) DeptManager $\sqsubseteq \exists$ isProhibited.SelfApprovalTravelAct
(12) obliged(x, act) $\wedge$ isProhibited(x,act) $\wedge\mathcal{O}$(x) $\wedge\mathcal{O}$(act) $\rightarrow$ owl:Nothing(x)

5.1 Dealing with Inconsistency

There are many ways to deal with inconsistency of norms. We can assign priority (or importance) to the axioms, and choose to ignore the lowest priority or the lowest importance. To do so in OWL, we can tag the axioms with a priority index, and ignore the one with the lowest index, when the problematic axioms are pinpointed by DL reasoners.

Vasconcelos et al. [23] propose another method to resolve the conflicts in a set of norms, the authors *curtail* the influence of the prohibition. The curtailment of the prohibition eliminates the intersection: it moves the scope of the norm influence to outside the influence of the permission. In their approach, prohibitions are always curtailed. We can achieve the *curtailment* in OWL by weakening the restriction of the axioms or raising an exception in the axioms. Given two inconsistent axioms:

(1) Agent $\sqsubseteq \exists$ obliged.Act1
(2) Agent $\sqsubseteq \exists$ isProhibited.Act2

If Act1 $\sqcap$ Act2 $\neq \emptyset$, we can weaken the restriction of the prohibition or obligation axiom, such that Agent $\sqsubseteq \exists$ obliged.(Act1 $\sqcap \neg$ (Act1 $\sqcap$ Act2)) or Agent $\sqsubseteq \exists$ isProhibited.(Act2 $\sqcap \neg$ (Act2 $\sqcap$ Act1)). The option of curtailing obligations or prohibitions is left to ontology designers.

The third approach [24] extends the work of [23]. Curtailment policies are introduced to explicitly state which norm should be removed. Given two norms in conflict, the norm is to be curtailed if its time of declaration precedes that of another norm. It will be worthwhile for us to consider the time of declaration of norms in our approach in future work.

To resolve the conflict of the department manager's obligation and prohibition in Example 2, we can add an axiom such that:

(13) DeptManager $\sqsubseteq \exists$ isObliged.(ApprovalTravelAct $\sqcap \neg$ SelfApprovalTravelAct)

Now department managers are obliged to approve travel request forms, except for the forms requested from themselves. This axiom overrides the obligation of manager (axiom 7 above).

If a department manager issues a travel request form, the ontology does not explicitly state who is responsible for the approval. Axiom (7) in Example 2 states managers are obliged to approve travel request form; this obligation applies to its sub-roles, i.e., DeptManager and GeneralManager. DeptManager overrides this obligation with an exception (axiom (13) above). GeneralManager still inherits this obligation, therefore, GeneralManager is obliged to approve the request forms from DeptManager.

6 Related Work

The work on norms in multi-agent systems and norm formalisation has emerged as an essential research area in the last few years. In this section, we summarise the literature briefly and compare it with our work,

Vázquez-Salceda et al. [25] provides a norm formalisation that includes conditional and temporal aspects. The paper focuses on norm enforcement which include the detection of norm violations and the repair plan. The repair plan contains sanctions to punish the violator and a set of actions to recover the system from the violation. The authors list a number of situation for detecting violations, i.e., detecting the violations all the time, or when the system is not busy, periodic schedules, and random checking. For repair plans, an example is given to illustrate how a set of actions can solve the violated situation. In our approach, enforcement of norms, sanctions and repair plans are not mentioned. To check the violation of conditional norms, we need to detect when certain conditions are fulfilled; to check the norms with deadlines, the program needs to keep track of the current date time and the deadline. However, it will be costly to make the program check the condition and deadline all the time. It is our future work to investigate the implementation of norm enforcement mechanism. Furthermore, their formalism is machine-readable; our semantic approach is partially machine-understandable, that means, agents are able to (1) implement reasoning to detect inconsistencies and infer implicit knowledge; (2) share and re-use information with other organisations.

Artikis et al. [1, 2] propose two alternative languages for the formalisation of norm-governed computational systems - the event calculus or the $\mathcal{C}+$ language. Obligations, permissions, prohibitions, capabilities, empowerment and sanctions are expressed as changing predicates called *fluents*. The formalism focuses on norms triggered by actions, it does not explicitly state the temporal interval associated with a permission, obligation, or a power. This interval is implicitly defined by constraints that initiate and terminate the permission, obligation or power. The authors state that they are still experimenting with the use of standard model checking techniques to prove general properties of a society specification, such as the consistency of a protocol specification (no agent is forbidden and obliged to perform an action at the same time).

Garcia-Camino et al. [9] introduces a normative language which is expressive enough to represent a MAS normative framework. The language captures the deontic notions of permission, prohibition and obligation in several cases such as absolute norms, conditional norms, norms with deadlines and norms in temporal

relation with other events. The defined norms are executable by translating into Jess. Their work supports action-dependent and time-dependent norms which include BEFORE, AFTER and BETWEEN, such as OBLIGED($utter(s, w, i)$ BETWEEN $t1, t2$). The expressions for action-dependent norms and the BETWEEN operator are our future work. Also, they do not cover checking contradictions in norms; we make use of DL reasoners to detect the contradictions and infer implicit knowledge.

In OperA [8], roles are described in terms of objectives, sub-objectives, norms, rights and the type of enactment. Roles can be organized into groups. The aim is to provide means to collectively refer to a set of roles, the norms in a group hold for all roles in the group. Compared to our approach, we can use ontologies to describe role definitions in OperA, where the objectives, sub-objectives, rights and norms are described by OWL axioms or SWRL rules. The relationship between objectives and sub-objectives are represented as superconcepts and subconcepts in a classification hierarchy. In OperA the sub-roles of Manager inherit all of its norms and rights. The definition of groups is done in a similar fashion to our super-roles in the role classification. The terms *compatibility* and *consistency* are introduced in OperA. An agent is compatible with a role when the goals of the agent are a subset of the objectives of the role; an agent is consistent with a role if the goals of the agent do not conflict with the objectives and norms of the role. However, the consistency between the norms of a role is not covered; this consistency checking will be important for system designers in order to discover if a role has implicit contradictions within itself, with respect to the norms it is placing on agents occupying the role.

Grossi et al. [11] propose that institutions can be based on ontologies, and these ontologies are contextual. A notion of *contextual ontology* is introduced, ontologies are used by institutions to determine the meaning of concepts used in the norms under different contexts. To define an eInstitution's ontology, several contextual ontologies should be considered, because a context may contain other (sub)contexts or belong to one or several (super)contexts. That means, one needs to create some links from the ontologies of different supercontexts to the institutional ontology; and so terms in the institutional ontology are inherited from its supercontexts. Different institutions can implement the set of norms in different ways; the formalism in [11] enables the possibility of representing and reasoning about divergent ontologies from different institutions. Their future work is to extend the framework with more normative reasoning issues, such as reasoning with violations and reasoning with instances of concepts. The difference of their work and our work is that they aim to support ontological discrepancies via contextual subsumption relations, i.e., inheritance from supercontexts, and then reason about these divergent ontologies. We focus on one single organisation represented in an ontology, in which reasoning about norms and inferring implicit knowledge are enabled.

Kagal and Finin [16] describe how ontologies can help in developing high level policies that are independent of the specific agent communication language being used. Their framework is implemented in Rei [17], in which a policy language is described in OWL. Rei extends OWL in order to support rule-based policies; it has a reasoning engine for OWL and RDF, enabling it to understand and reason over policies and specifications defined in both OWL and RDF. To resolve

conflicts between policies, their framework includes meta-policies which include setting modality precedence and stating the priority between rules. Our approach is similar to some extent that using ontologies to represent policies and deontic concepts, it is possible to reason over the current obligations or permissions to perform an action or a speech act, even through there exist conflicts between policies.

7 Conclusion and Future Work

In this paper we have demonstrated that Semantic Web languages are sufficiently expressive to capture many of the important concepts underpinning organisational models of multi-agent systems. Our proposal includes a rich representation for roles and their relationships, and allows us to capture various kinds of normative notions and power, including norms with conditions and deadlines, and the definition of restrictions on unwanted behaviours in agents. By using description logics to specify our ontologies, and Semantic Web standards to capture our rules, our approach benefits from off-the-shelf technologies such as reasoners and rule engines, which we deploy to verify organisations and infer implicit knowledge from them. For example, we have shown how to reason about such things as consistency of norms, and to uncover implicit information revealing contradictions, also pinpointing the source of the conflict in the organisation; many have ignored these issues in MAS research thus far. Other advantages of using Semantic Web languages are making use of existing techniques in the Semantic Web community. For example, if two organisations are going to be merged, we can use existing ontology merging techniques.

Finally, we note that deontic logic gives quite a different semantics to normative notions; we merely aim to formalise a basic approximation of these notions, which could be of practical use in agent systems. For example, there are very basic deontic logic statements such as those describing contrary-to-duty situations (see Chisholm's paradox [7]) which are not considered in our work. In future work, we would like to extend our approach to support a wider range of norms such as right, duty, delegation, representation and entitlement.

References

1. A. Artikis. *Executable Specification of Open Norm-Governed Computational Systems*. PhD thesis, Imperial College London, 2003.
2. A. Artikis, L. Kamara, J. Pitt, and M. Sergot. A protocol for resource sharing in norm-governed ad hoc networks. In J. A. Leite, A. Omicini, P. Torroni, and P. Yolum, editors, *Proceedings of the Declarative Agent Languages and Technologies (DALT)*, pages 221–238. Springer, 2004.
3. A. Artikis, M. Sergot, and J. Pitt. Specifying norm-governed computational societies. *ACM Transactions on Computational Logic*, 2009. To appear.
4. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2002.
5. S. Bechoffer, F. van Harmlen, J. Hendler, I. Horrocks, D. McGuinnes, P. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference, February 2004. http://www.w3.org/TR/owl-ref/.
6. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2004.

7. R. M. Chisholm. Contrary-to-duty imperatives and deontic logic. *Analysis*, 24:33–36, 1963.
8. V. Dignum. *A Model for Organizational Interaction: Based on Agents, Founded in Logic*. PhD thesis, Utrecht University, 2004.
9. A. Garcia-Camino, P. Noriega, and J. A. Rodriguez-Aguilar. Implementing norms in electronic institutions. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 667–673, New York, NY, USA, 2005. ACM.
10. A. García-Camino, J. A. Rodríguez-Aguilar, C. Sierra, and W. Vasconcelos. Constraint rule-based programming of norms for electronic institutions. *Journal of Autonomous Agents and Multi-Agent Systems*, 2008. to appear.
11. D. Grossi, H. Aldewereld, J. Vázquez-Salceda, and F. Dignum. Ontological aspects of the implementation of norms in agent-based electronic institutions. *Journal of Computational and Mathematical Organization Theory*, 12(2-3):251–275, 2006.
12. I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proc. of the 2nd International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 17–29. Springer, 2003.
13. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
14. I. Horrocks and U. Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence*, pages 448–453, 2005.
15. A. I. J. Jones and M. J. Sergot. A formal characterisation of institutionalised power. *Journal of the IGPL*, 4(3):429–445, 1996.
16. L. Kagal and T. Finin. Modeling conversation policies using permissions and obligations. *Journal of Autonomous Agents and Multi-Agent Systems*, 14(2):187–206, 2007.
17. L. Kagal, T. Finin, and A. Joshi. A policy based approach to security for the semantic web. In *Proc. of the 2nd International Semantic Web Conference (ISWC 2003)*, pages 402–418. Springer, October 2003.
18. A. Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, University of Maryland College Park, 2006.
19. M. J. Kollingbaum and T. J. Norman. Informed deliberation during norm-governed practical reasoning. In *Proceedings of the International Workshop on Agents, Norms and Institutions for Regulated Multiagent Systems*, 2005.
20. J. Lam, D. Sleeman, J. Pan, and W. Vasconcelos. A fine-grained approach to resolving unsatisable ontologies. *Journal on Data Semantics X*, 4900:62–95, 2008.
21. B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. In *Proc. of the 3rd Int. Semantic Web Conference*, pages 549–563. Springer, 2004.
22. A. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. OWL pizzas: Common errors and common patterns. In *Proceedings of the European Conference on Knowledge Acquistion*, pages 63–81. Springer-Verlag, 2004.
23. W. Vasconcelos, M. J. Kollingbaum, and T. J. Norman. Resolving conflict and inconsistency in norm-regulated virtual organizations. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.
24. W. Vasconcelos, M. J. Kollingbaum, and T. J. Norman. Management of norms and their conflicts in multi-agent systems. 2008. Submitted for publication.
25. J. Vázquez-Salceda, H. Aldewereld, and F. Dignum. Implementing norms in multiagent systems. In *MATES*, volume LNAI 3187, pages 313–327. Springer, 2004.
26. G. H. von. Wright. Deontic logic. *Mind, New Series*, 60(237):1–15, January 1951.