Distributed Norm Management in Regulated Multi-Agent Systems *

Dorian Gaertner
Dept. of Computing,
Imperial College London,
London SW7 2AZ,
United Kingdom
dg00@doc.ic.ac.uk

Andres Garcia-Camino,
Pablo Noriega,
J.-A. Rodriguez-Aguilar
IIIA-CSIC,
08193 Bellaterra, Spain
{andres,pablo,jar}@iiia.csic.es

Wamberto Vasconcelos
Dept. of Computing Science,
University of Aberdeen,
Aberdeen AB24 3UE,
United Kingdom
wvasconcelos@acm.org

ABSTRACT

Norms are widely recognised as a means of coordinating multi-agent systems. The distributed management of norms is a challenging issue and we observe a lack of *truly distributed* computational realisations of normative models. In order to regulate the behaviour of autonomous agents that take part in multiple, related activities, we propose a normative model, the *Normative Structure* (NS), an artifact that is based on the propagation of normative positions (obligations, prohibitions, permissions), as consequences of agents' actions. Within a NS, conflicts may arise due to the dynamic nature of the MAS and the concurrency of agents' actions. However, ensuring conflict-freedom of a NS at design time is computationally intractable. We show this by formalising the notion of conflict, providing a mapping of NSs into Coloured Petri Nets and borrowing well-known theoretical results from that field. Since online conflict resolution is required, we present a tractable algorithm to be employed distributedly. We then demonstrate that this algorithm is paramount for the distributed enactment of a NS.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Languages and structures

General Terms

Algorithms, Design, Theory

Keywords

Regulated multi-agent systems, normative conflicts, electronic institutions, organisations, coordination

*This work was partially funded by the Spanish Education and Science Ministry and Spanish Council for Scientific Research (CSIC) as part of the projects TIN2006-15662-C02-01 and 2006-5-0I-099. García-Camino enjoys an I3P grant from CSIC.

AAMAS'07 May 14–18 2007, Honolulu, Hawai'i, USA.

1. INTRODUCTION

A fundamental feature of open, regulated multi-agent systems in which autonomous agents interact, is that participating agents are meant to comply with the conventions of the system. Norms can be used to model such conventions and hence as a means to regulate the observable behaviour of agents [6, 29]. There are many contributions on the subject of norms from sociologists, philosophers and logicians (*e.g.*, [15, 28]). However, there are very few proposals for computational realisations of normative models — the way norms can be integrated in the design and execution of MASs. The few that exist (*e.g.* [10, 13, 24]), operate in a centralised manner which creates bottlenecks and single points-of-failure. To our knowledge, no proposal truly supports the distributed enactment of normative environments.

In our paper we approach that problem and propose means to handle conflicting commitments in open, regulated, multi-agent systems in a *distributed* manner. The type of regulated MAS we envisage consists of multiple, concurrent, related activities where agents interact. Each agent may concurrently participate in several activities, and change from one activity to another. An agent's actions within an activity may have consequences in the form of normative positions (*i.e.* obligations, permissions, and prohibitions) [26] that may constrain its future behaviour. For instance, a buyer agent who runs out of credit may be forbidden to make further offers, or a seller agent is obliged to deliver after closing a deal. We assume that agents may choose not to fulfill all their obligations and hence may be sanctioned by the MAS. Notice that, when activities are distributed, normative positions must *flow* from the activities in which they are generated to those in which they take effect. For instance, the seller's obligation above must flow (or be propagated) from a negotiation activity to a delivery activity.

Since in an open, regulated MAS one cannot embed normative aspects into the agents' design, we adopt the view that the MAS should be supplemented with a separate set of norms that further regulates the behaviour of participating agents. In order to model the separation of concerns between the coordination level (agents' interactions) and the normative level (propagation of normative positions), we propose an artifact called the *Normative Structure* (NS).

Within a NS conflicts may arise due to the dynamic nature of the MAS and the concurrency of agents' actions. For instance, an agent may be obliged and prohibited to do the

very same action in an activity. Since the regulation of a MAS entails that participating agents need to be aware of the validity of those actions that take place within it, such conflicts ought to be identified and possibly resolved if a claim of validity is needed for an agent to engage in an action or be sanctioned. However, ensuring conflict-freedom of a NS at design time is computationally intractable. We show this by formalising the notion of conflict, providing a mapping of NSs into Coloured Petri Nets (CPNs) and borrowing well-known theoretical results from the field of CPNs.

We believe that online conflict detection and resolution is required. Hence, we present a tractable algorithm for conflict resolution. This algorithm is paramount for the distributed enactment of a NS.

The paper is organised as follows. In Section 2 we detail a scenario to serve as an example throughout the paper. Next, in Section 3 we formally define the normative structure artifact. Further on, in Section 4 we formalise the notion of conflict to subsequently analyse the complexity of conflict detection in terms of CPNs in Section 5. Section 6 describes the computational management of NSs by describing their enactment and presenting an algorithm for conflict resolution. Finally, we comment on related work, draw conclusions and report on future work in Section 7.

2. SCENARIO

We use a supply-chain scenario in which companies and individuals come together at an online marketplace to conduct business. The overall transaction procedure may be organised as six distributed activities, represented as nodes in the diagram in Figure 1. They involve different participants whose behaviour is coordinated through protocols.

In this scenario agents can play one of four roles: marketplace accountant (*acc*), *client*, supplier (*supp*) and warehouse managers (*wm*). The arrows connecting the activities represent how agents can move from one activity to another.

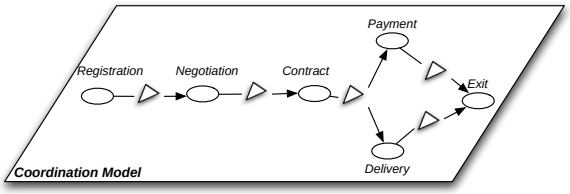

Figure 1: Activity Structure of the Scenario

After registering at the marketplace, clients and suppliers get together in an activity where they negotiate the terms of their transaction, *i.e.* prices, amounts of goods to be delivered, deadlines and other details. In the *contract* activity, the order becomes established and an invoice is prepared. The client will then participate in a *payment* activity, verifying his credit-worthiness and instructing his bank to transfer the correct amount of money. The supplier in the meantime will arrange for the goods to be delivered (*e.g.* via a warehouse manager) in the *delivery* activity. Finally, agents can leave the marketplace conforming to a predetermined *exit* protocol. The marketplace accountant participates in most of the activities as a trusted provider of auditing tools.

In the rest of the paper we shall build on this scenario to exemplify the notion of normative structure and to illustrate our approach to conflict detection and resolution in a distributed setting.

3. NORMATIVE STRUCTURE

In MASs agents interact according to protocols which naturally are distributed. We advocate that actions in one such protocol may have an effect on the enactment of other protocols. Certain actions can become prohibited or obligatory, for example. We take normative positions to be obligations, prohibitions and permissions akin to work described in [26]. The intention of adding or removing a normative position we call *normative command*. Occurrences of normative positions in one protocol may also have consequences for other protocols[1].

In order to define our norm language and specify how normative positions are propagated, we have been inspired by multi-context systems [14]. These systems allow the structuring of knowledge into distinct formal theories and the definition of relationships between them. The relationships are expressed as bridge rules – deducibility of formulae in some contexts leads to the deduction of other formulae in other contexts. Recently, these systems have been successfully used to define agent architectures [11, 23]. The metaphor translates to our current work as follows: the utterance of illocutions and/or the existence of normative positions in some normative scenes leads to the deduction of normative positions in other normative scenes. We are concerned with the propagation and distribution of normative positions within a network of distributed, normative scenes as a consequence of agents' actions. We take normative scenes to be sets of normative positions and utterances that are associated with an underlying interaction protocol corresponding to an activity.

In this section, we first present a simple language capturing these aspects and formally introduce the notions of normative scene, normative transition rule and normative structure. We give the intended semantics of these rules and show how to control a MAS via norms in an example.

3.1 Basic Concepts

The building blocks of our language are *terms* and *atomic formulae*:

DEF. 1. *A term, denoted as t, is (i) any constant expressed using lowercase (with or without subscripts), e.g.* a, b_0, c *or (ii) any variable expressed using uppercase (with or without subscripts), e.g.* X, Y, Z_b *or (iii) any function* $f(t_1, \ldots, t_n)$*, where* f *is an n-ary function symbol and* $t_1, .., t_n$ *are terms.*

Some examples of terms and functions are *Credit*, *price* or *offer*(*bible*, 30) being respectively a variable, a constant and a function. We will be making use of identifiers throughout the paper, which are constant terms and also need the following definition:

DEF. 2. *An atomic formula is any construct* $p(t_1, \ldots, t_n)$*, where* p *is an n-ary predicate symbol and* $t_1, \ldots, t_n$ *are terms. The set of all atomic formulae is denoted as* Δ*.*

We focus on an expressive class of MASs in which interaction is carried out by means of illocutionary speech acts exchanged among participating agents:

DEF. 3. *Illocutions* | *are ground atomic formulae which have the form* $p(ag, r, ag', r', \delta, t)$ *where* p *is an element of*

[1]Here, we abstract from protocols and refer to them generically as activities.

a set of illocutionary particles (e.g. inform, request, offer*); ag, ag' are agent identifiers; r, r' are role identifiers; δ, an arbitrary ground term, is the content of the message, built from a shared content language; $t \in \mathbb{N}$ is a time stamp.*

The intuitive meaning of $p(ag, r, ag', r', m, t)$ is that agent ag playing role r sent message m to agent ag' playing role r' at time t. An example of an illocution is $inform(ag_4, supp, ag_3, client, offer(wire, 12), 10)$. Sometimes it is useful to refer to illocutions that are not fully grounded, that is, those that may contain uninstantiated (free) *variables*. In the description of a protocol, for instance, the precise values of the message exchanged can be left unspecified. During the enactment of the protocol agents will produce the actual values which will give rise to a ground illocution. We can thus define *illocution schemata*:

DEF. 4. *An illocution schema $\bar{\mathsf{I}}$ is any atomic formula $p(ag, r, ag', r', \delta, t)$ in which some of the terms may either be variables or may contain variables.*

3.2 Formal Definition of the Notion of NS

We first define normative scenes as follows:

DEF. 5. *A normative scene is a tuple $s = \langle id_s, \Delta_s \rangle$ where id_s is a scene identifier and Δ_s is the set of atomic formulae δ (i.e. utterances and normative positions) that hold in s. We will also refer to Δ_s as the state of normative scene s.*

For instance, a snapshot of the state of the *delivery* normative scene of our scenario could be represented as:

$$\Delta_s = \left\{ \begin{array}{l} utt(request(sean, client, kev, wm, receive(wire, 200), 20)), \\ utt(accept(kev, wm, sean, client, receive(wire, 200), 30)), \\ obl(inform(kev, wm, sean, client, delivered(wire, 200), 30)) \end{array} \right\}$$

That is, agent Sean taking up the *client* role has requested agent Kev (taking up the warehouse manager role *wm*) to receive 200kg of wire, and agent Kev is obliged to deliver 200kg of wire to Sean since he accepted the request. Note that the state of a normative scene Δ_s evolves over time. These normative scenes are connected to one another via normative transitions that specify how utterances and normative positions in one scene affect other normative scenes.

As mentioned above, activities are not independent since illocutions uttered in some of them may have an effect on other ones. Normative transition rules define the conditions under which a normative command is generated. These conditions are either utterances or normative positions associated with a given protocol (denoted *e.g. activity* : *utterance*) which yield a normative command, *i.e.* the addition or removal of another normative position, possibly related to a different activity. Our transition rules are thus defined:

DEF. 6. *A normative transition rule R is of the form:*

$$\begin{array}{lcl} \mathsf{R} & ::= & \mathsf{V} \Rrightarrow \mathsf{C} \\ \mathsf{V} & ::= & id_s : \mathsf{D} \mid \mathsf{V}, \mathsf{V} \\ \mathsf{D} & ::= & \mathsf{N} \mid utt(\bar{\mathsf{I}}) \\ \mathsf{N} & ::= & per(\bar{\mathsf{I}}) \mid prh(\bar{\mathsf{I}}) \mid obl(\bar{\mathsf{I}}) \\ \mathsf{C} & ::= & \mathtt{add}(id_s : \mathsf{N}) \mid \mathtt{remove}(id_s : \mathsf{N}) \end{array}$$

where $\bar{\mathsf{I}}$ is an illocution schema, N is a normative position (i.e. permission, prohibition or obligation), id_s is an identifier for activity s and C is a normative command.

We endow our language with the usual semantics of rule-based languages [19]. Rules map an existing normative structure to a new normative structure where only the state of the normative scenes change. In the definitions below we rely on the standard concept of substitution [9].

DEF. 7. *A normative transition is a tuple $b = \langle id_b, r_b \rangle$ where id_b is an identifier and r_b is a normative transition rule.*

We are proposing to extend the notion of MAS, regulated by protocols, with an extra layer consisting of normative scenes and normative transitions. This layer is represented as a bi-partite graph that we term normative structure. A normative structure relates normative scenes and normative transitions specifying which normative positions are to be generated or removed in which normative scenes.

DEF. 8. *A normative structure is a labelled bi-partite graph $NS = \langle Nodes, Edges, \mathcal{L}^{in}, \mathcal{L}^{out} \rangle$. $Nodes$ is a set $S \cup B$ where S is a set of normative scenes and B is a set of normative transitions. $Edges$ is a set $A^{in} \cup A^{out}$ where $A^{in} \subseteq S \times B$ is a set of input arcs labelled with an atomic formula using the labelling function $\mathcal{L}^{in} : A^{in} \mapsto \mathsf{D}$; and $A^{out} \subseteq B \times S$ is a set of output arcs labelled with a normative position using the labelling function $\mathcal{L}^{out} : A^{out} \mapsto \mathsf{N}$. The following must hold:*

1. *Each atomic formula appearing in the LHS of a rule r_b must be of the form $(id_s : \mathsf{D})$ where $s \in S$ and $\mathsf{D} \in \Delta$ and $\exists a^{in} \in A^{in}$ such that $a^{in} = (s, b)$ and $\mathcal{L}^{in}(a^{in}) = \mathsf{D}$.*
2. *The atomic formula appearing in the RHS of a rule r_b must be of the form $\mathtt{add}(id_s : \mathsf{N})$ or $\mathtt{remove}(id_s : \mathsf{N})$ where $s \in S$ and $\exists a^{out} \in A^{out}$ such that $a^{out} = (b, s)$ and $\mathcal{L}^{out}(a^{out}) = \mathsf{N}$.*
3. *$\forall a \in A^{in}$ such that $a = (s, b)$ and $b = \langle id_b, r_b \rangle$ and $\mathcal{L}^{in}(a) = \mathsf{D}$ then $(id_s{:}\mathsf{D})$ must occur in the LHS of r_b.*
4. *$\forall a \in A^{out}$ such that $a = (b, s)$ and $b = \langle id_b, r_b \rangle$ and $\mathcal{L}^{out}(a) = \mathsf{N}$ then $\mathtt{add}(id_s : \mathsf{N})$ or $\mathtt{remove}(id_s : \mathsf{N})$ must occur in the RHS of r_b.*

The first two points ensure that every atomic formulae on the *LHS* of a normative transition rule labels an arc entering the appropriate normative transition in the normative structure, and that the atomic formula on the *RHS* labels the corresponding outgoing arc. Points three and four ensure that labels from all incoming arcs are used in the *LHS* of the normative transition rule that these arcs enter into, and that the labels from all outgoing arcs are used in the *RHS* of the normative transition rule that these arcs leave.

3.3 Intended Semantics

The formal semantics will be defined via a mapping to Coloured Petri Nets in Section 5.1. Here we start defining the intended semantics of normative transition rules by describing how a rule changes a normative scene of an existing normative structure yielding a new normative structure. Each rule is triggered once for each substitution that unifies the left-hand side V of the rule with the state of the corresponding normative scenes. An atomic formula (*i.e.* an utterance or a normative position) holds iff it is unifiable with an utterance or normative position that belongs to the state of the corresponding normative scene. Every time a rule is triggered, the normative command specified on the right-hand side of that rule is carried out, intending to add or remove a normative position from the state of the corresponding normative scene. However, addition is not unconditional as conflicts may arise. This topic will be treated in Sections 4 and 6.1.

3.4 Example

In our running example we have the following exemplary normative transition rule:

$$\left(\begin{array}{l} payment : obl(inform(X, client, Y, acc, pay(Z, P, Q), T)), \\ payment : utt(inform(X, client, Y, acc, pay(Z, P, Q), T')) \end{array}\right)$$
$$\Rightarrow delivery : \texttt{add}(obl(inform(Y, wm, X, client, delivered(Z, Q), T'')))$$

That is, during the *payment* activity, an obligation on *client* X to inform *accountant* Y about the payment P of item Z at time T and the corresponding utterance which fulfills this obligation allows the flow of a norm to the *delivery* activity. The norm is an obligation on agent Y (this time taking up the role of the warehouse manager *wm*) to send a message to *client* X that item Z has been delivered. We show in Figure 2 a diagrammatic representation of how activities and a normative structure relate:

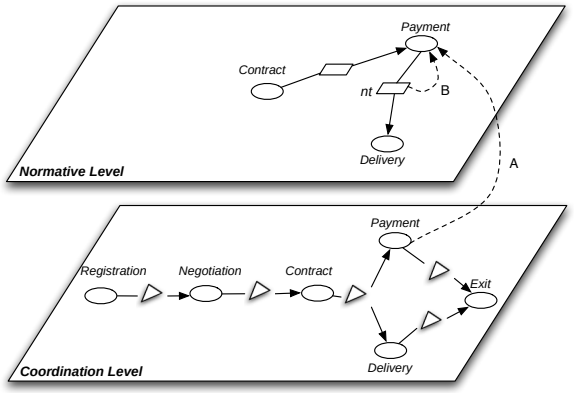

Figure 2: Activities and Normative Structure

As illocutions are uttered during activities, normative positions arise. Utterances and normative positions are combined in transition rules, causing the flow of normative positions between normative scenes. The connection between the two levels is described in Section 6.2.

4. CONFLICT DEFINITION

The terms deontic *conflict* and deontic *inconsistency* have been used interchangeably in the literature. However, in this paper we adopt the view of [7] in which the authors suggest that a deontic inconsistency arises when an action is simultaneously permitted and prohibited – since a permission may not be acted upon, no real conflict occurs. The situations when an action is simultaneously obliged and prohibited are, however, deontic conflicts, as both obligations and prohibitions influence behaviours in a conflicting fashion. The content of normative positions in this paper are illocutions. Therefore, a normative conflict arises when an illocution is simultaneously obliged and prohibited.

We propose to use the standard notion of unification [9] to detect when a prohibition and a permission overlap. For instance, an obligation $obl(inform(A_1, R_1, A_2, R_2, p(c, X), T))$ and a prohibition $prh(inform(a_1, r_1, a_2, r_2, p(Y, d), T'))$ are in conflict as they unify under $\sigma = \{A_1/a_1, R_1/r_1, A_2/a_2, R_2/r_2, Y/c, X/d, T/T'\}$. We formally capture this notion:

Def. 9. *A (deontic) conflict arises between two normative positions* N *and* N' *under a substitution* σ*, denoted as* $\mathbf{conflict}(\mathsf{N}, \mathsf{N}', \sigma)$*, if and only if* $\mathsf{N} = prh(\bar{\mathsf{I}}), \mathsf{N}' = obl(\bar{\mathsf{I}}')$ *and* $unify(\bar{\mathsf{I}}, \bar{\mathsf{I}}', \sigma)$.

That is, a prohibition and an obligation are in conflict if, and only if, their illocutions unify under σ. The substitution σ, called here the *conflict set*, unifies the agents, roles and atomic formulae. We assume that *unify* is a suitable implementation of a unification algorithm which *i)* always terminates (possibly failing, if a unifier cannot be found); *ii)* is correct; and *iii)* has linear computational complexity.

Inconsistencies caused by the same illocution being simultaneously permitted and prohibited can be formalised similarly. In this paper we focus on prohibition/obligation conflicts, but the computational machinery introduced in Section 6.1 can equally be used to detect prohibition/permission inconsistencies, if we substitute modality "*obl*" for "*per*".

5. FORMALISING CONFLICT-FREEDOM

In this section we introduce some background knowledge on CPNs assuming a basic understanding of ordinary Petri Nets. For technical details we refer the reader to [16]. We then map NSs to CPNs and analyse their properties.

CPNs combine the strength of Petri nets with the strength of functional programming languages. On the one hand, Petri nets provide the primitives for the description of the synchronisation of concurrent processes. As noticed in [16], CPNs have a semantics which builds upon true concurrency, instead of interleaving. In our opinion, a true-concurrency semantics is easier to work with because it is the way we envisage the connection between the coordination level and the normative level of a multi-agent system to be. On the other hand, the functional programming languages used by CPNs provide the primitives for the definition of data types and the manipulation of their data values. Thus, we can readily translate expressions of a normative structure. Last but not least, CPNs have a well-defined semantics which unambiguously defines the behaviour of each CPN. Furthermore, CPNs have a large number of formal analysis methods and tools by which properties of CPNs can be proved. Summing up, CPNs provide us with all the necessary features to formally reason about normative structures given that an adequate mapping is provided.

In accordance with Petri nets, the states of a CPN are represented by means of *places*. But unlike Petri Nets, each place has an associated *data type* determining the kind of data which the place may contain. A state of a CPN is called a *marking*. It consists of a number of tokens positioned on the individual places. Each token carries a data value which has the type of the corresponding place. In general, a place may contain two or more tokens with the same data value. Thus, a marking of a CPN is a function which maps each place into a multi-set[2] of tokens of the correct type. One often refers to the token values as token *colours* and one also refers to the data types as colour sets. The types of a CPN can be arbitrarily complex.

Actions in a CPN are represented by means of *transitions*. An incoming arc into a transition from a place indicates that the transition may remove tokens from the corresponding place while an outgoing arc indicates that the transition may add tokens. The exact number of tokens and their data values are determined by the arc expressions, which are encoded using the programming language chosen for the CPN. A transition is *enabled* in a CPN if and only if all the

[2]A *multi-set* (or bag) is an extension to the notion of set, allowing the possibility of *multiple appearances* of the same element.

variables in the expressions of its incoming arcs are bound to some value(s) (each one of these bindings is referred to as a *binding element*). If so, the transition may *occur* by removing tokens from its input places and adding tokens to its output places. In addition to the arc expressions, it is possible to attach a boolean *guard* expression (with variables) to each transition. Putting all the elements above together we obtain a formal definition of CPN that shall be employed further ahead for mapping purposes.

DEF. 10. *A CPN is a tuple* $\langle \Sigma, P, T, A, N, C, G, E, I \rangle$ *where: (i)* Σ *is a finite set of non-empty types, also called colour sets; (ii)* P *is a finite set of places; (iii)* T *is a finite set of transitions; (iv)* A *is a finite set of arcs; (v)* N *is a node function defined from* A *into* $P \times T \cup T \times P$*; (vi)* C *is a colour function from* P *into* Σ*; (vii)* G *is a guard function from* T *into expressions; (viii)* E *is an arc expression function from* A *into expressions; (ix)* I *is an initialisation function from* P *into closed expressions;*

Notice that the informal explanation of the enabling and occurrence rules given above provides the foundations to understand the behaviour of a CPN. In accordance with ordinary Petri nets, the concurrent behaviour of a CPN is based on the notion of *step*. Formally, a step is a non-empty and finite multi-set over the set of all binding elements. Let step $\mathcal{S}$ be enabled in a marking $\mathcal{M}$. Then, $\mathcal{S}$ may *occur*, changing the marking $\mathcal{M}$ to $\mathcal{M}'$. Moreover, we say that marking $\mathcal{M}'$ is *directly reachable* from marking $\mathcal{M}$ by the occurrence of step $\mathcal{S}$, and we denote it by $\mathcal{M}[\mathcal{S} > \mathcal{M}'$.

A *finite occurrence sequence* is a finite sequence of steps and markings: $\mathcal{M}_1[\mathcal{S}_1 > \mathcal{M}_2 \ldots \mathcal{M}_n[\mathcal{S}_n > \mathcal{M}_{n+1}$ such that $n \in \mathbb{N}$ and $\mathcal{M}_i[\mathcal{S}_i > \mathcal{M}_{i+1}\ \forall i \in \{1, \ldots, n\}$. The set of all possible markings reachable for a net Net from a marking $\mathcal{M}$ is called its *reachability set*, and is denoted as $R(Net, \mathcal{M})$.

5.1 Mapping to Coloured Petri Nets

Our normative structure is a labelled bi-partite graph. The same is true for a Coloured Petri Net. We are presenting a mapping f from one to the other, in order to provide semantics for the normative structure and prove properties about it by using well-known theoretical results from work on CPNs. The mapping f makes use of correspondences between normative scenes and CPN places, normative transitions and CPN transitions and finally, between arc labels and CPN arc expressions.

$$\begin{array}{rcl} S & \mapsto & P \\ B & \mapsto & T \\ \mathcal{L}^{in} \cup \mathcal{L}^{out} & \mapsto & E \end{array}$$

The set of types is the singleton set containing the colour NP (*i.e.* $\Sigma = \{\mathsf{NP}\}$). This complex type is structured as follows (we use CPN-ML [4] syntax):

```
color NPT = with Obl | Per | Prh | NoMod
color IP  = with inform | declare | offer
color UTT = record
              illp   : IP
              ag1, role1, ag2, role2 : string
              content: string
              time   : int
color NP  = record
              mode   : NPT
              illoc  : UTT
```

Modelling illocutions as norms without modality (NoMod) is a formal trick we use to ensure that sub-nets can be combined as explained below. Arcs are mapped almost directly. A is a finite set of arcs and N is a node function, such that $\forall a \in A\ \exists a' \in A^{in} \cup A^{out}.\ N(a) = a'$. The initialisation function I is defined as $I(p) = \Delta_s$ ($\forall s \in S$ where p is obtained from s using the mapping; remember that $s = \langle id_s, \Delta_s \rangle$). Finally, the colour function C assigns the colour NP to every place: $C(p) = \mathsf{NP}$ ($\forall p \in P$). We are not making use of the guard function G. In future work, this function can be used to model constraints when we extend the expressiveness of our norm language.

5.2 Properties of Normative Structures

Having defined the mapping from normative structures to Coloured Petri Nets, we now look at properties of CPNs that help us understand the complexity of conflict detection. One question we would like to answer is, whether at a given point in time, a given normative structure is conflict-free. Such a snapshot of a normative structure corresponds to a marking in the mapped CPN.

DEF. 11. *Given a marking* $\mathcal{M}_i$*, this marking is conflict-free if* $\neg\exists p \in P.\ \exists np_1, np_2 \in \mathcal{M}_i(p)$ *such that* $np_1.mode = Obl$ *and* $np_2.mode = Prh$ *and* $np_1.illoc$ *and* $np_2.illoc$ *unify under a valid substitution.*

Another interesting question would be, whether a conflict will occur from such a snapshot of the system by propagating the normative positions. In order to answer this question, we first translate the snapshot of the normative structure to the corresponding CPN and then *execute* the finite *occurence sequence* of markings and steps, verifying the conflict-freedom of each marking as we go along.

DEF. 12. *Given a marking* $\mathcal{M}_i$*, a finite occurrence sequence* $\mathcal{S}_i, \mathcal{S}_{i+1}, ..., \mathcal{S}_n$ *is called conflict-free, if and only if* $\mathcal{M}_i[\mathcal{S}_i > \mathcal{M}_{i+1} \ldots \mathcal{M}_n[\mathcal{S}_n > \mathcal{M}_{n+1}$ *and* $\mathcal{M}_k$ *is conflict-free for all* k *such that* $i \leq k \leq n+1$*.*

However, the main question we would like to investigate, is whether or not a given normative structure is *conflict-resistant*, that is, whether or not the agents enacting the MAS are able to bring about conflicts through their actions. As soon as one includes the possibility of actions (or utterances) from autonomous agents, one looses determinism.

Having mapped the normative structure to a CPN, we now add CPN models of the agents' interactions. Each form of agent interaction (*i.e.* each activity) can be modelled using CPNs along the lines of Cost et al. [5]. These non-deterministic CPNs *"feed"* tokens into the CPN that models the normative structure. This leads to the introduction of non-determinism into the combined CPN.

The lower half of figure 3 shows part of a CPN model of an agent protocol where the arc denoted with '1' represents some utterance of an illocution by an agent. The target transition of this arc, not only moves a token on to the next state of this CPN, but also places a token in the place corresponding to the appropriate normative scene in the CPN model of the normative structure (via arc '2'). Transition '3' finally could propagate that token in form of an obligation, for example. Thus, from a given marking, many different occurrence sequences are possible depending on the agents' actions. We make use of the reachability set R to define a situation in which agents cannot cause conflicts.

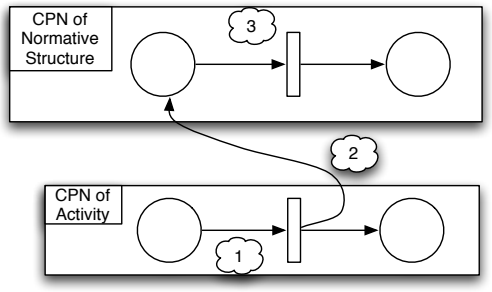

Figure 3: Constructing the combined CPN

DEF. 13. *Given a net N, a marking $\mathcal{M}$ is conflict-resistant if and only if all markings in R(N,$\mathcal{M}$) are conflict-free.*

Checking conflict-freedom of a marking can be done in polynomial time by checking all places of the CPN for conflicting tokens. Conflict-freedom of an occurrence sequence in the CPN that represents the normative structure can also be done in polynomial time since this sequence is deterministic given a snapshot.

Whether or not a normative structure is designed safely corresponds to checking the conflict-resistance of the initial marking $\mathcal{M}_0$. Now, verifying conflict-resistance of a marking becomes a very difficult task. It corresponds to the reachability problem in a CPN: "can a state be reached or a marking achieved, that contains a conflict?". This reachability problem is known to be NP-complete for ordinary Petri Nets [22] and since CPNs are functionally identical, we cannot hope to verify conflict-resistance of a normative structure off-line in a reasonable amount of time. Therefore, distributed, run-time mechanisms are needed to ensure that a normative structure maintains consistency. We present one such mechanism in the following section.

6. MANAGING NORMATIVE STRUCTURES

Once a conflict (as defined in Section 4) has been detected, we propose to employ the unifier to resolve the conflict. In our example, if the variables in $prh(inform(a_1, r_1, a_2, r_2, p(Y, d), T'))$ do not get the values specified in substitution σ then there will not be a conflict. However, rather than computing the complement set of a substitution (which can be an infinite set) we propose to annotate the prohibition with the unifier itself and use it to determine what the variables of that prohibition *cannot* be in future unifications in order to avoid a conflict. We therefore denote annotated prohibitions as $prh(\bar{\mathsf{I}}) \odot \Sigma$, where $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, is a set of unifiers. Annotated norms[3] are interpreted as deontic constructs with *curtailed* influences, that is, their effect (on agents, roles and illocutions) has been limited by the set Σ of unifiers. A prohibition may be in conflict with various obligations in a given normative scene $s = \langle id, \Delta \rangle$ and we need to record (and possibly avoid) all these conflicts. We define below an algorithm which ensures that a normative position will be added to a normative scene in such a way that it will not cause any conflicts.

[3]Although we propose to curtail prohibitions, the same machinery can be used to define the curtailment of obligations instead. These different policies are dependent on the intended deontic semantics and requirements of the systems addressed. For instance, some MASs may require that their agents should not act in the presence of conflicts, that is, the obligation should be curtailed.

6.1 Conflict Resolution

We propose a fine-grained way of resolving normative conflicts via unification. We detect the overlapping of the influences of norms , *i.e.* how they affect the behaviour of the concerned agents, and we curtail the influence of the normative position, by appropriately using the annotations when checking if the norm applies to illocutions. The algorithm shown in Figure 4 depicts how we maintain a conflict-free set of norms. It adds a given norm N to an existing, conflict-free normative state Δ, obtaining a resulting new normative state Δ' which is conflict-free, that is, its prohibitions are annotated with a set of *conflict sets* indicating which bindings for variables have to be avoided for conflicts not to take place.

```
    algorithm addNorm(N, Δ)
    begin
 1    timestamp(N)
 2    case N of
 3      per(Ī): Δ' := Δ ∪ {N}
 4      prh(I): if N' ∈ Δ s.t. conflict(N, N', σ) then Δ' := Δ
 5                else Δ' := Δ ∪ {N}
 6      prh(Ī):
 7        begin
 8          Σ := ∅
 9          for each N' ∈ Δ do
10            if conflict(N, N', σ) then Σ := Σ ∪ {σ}
11          Δ' := Δ ∪ {N ⊙ Σ}
12        end
13      obl(Ī):
14        begin
15          Δ'1 := ∅; Δ'2 := ∅
16          for each (N' ⊙ Σ) ∈ Δ do
17            if N' = prh(I) then
18              if conflict(N', N, σ) then Δ'1 := Δ'1 ∪ {N' ⊙ Σ}
19              else nil
20            else
21              if conflict(N', N, σ) then
22              begin
23                Δ'1 := Δ'1 ∪ {N' ⊙ Σ}
24                Δ'2 := Δ'2 ∪ {N' ⊙ (Σ ∪ {σ})}
25              end
26          Δ' := (Δ − Δ'1) ∪ Δ'2 ∪ {N}
27        end
28    end case
29    return Δ'
    end
```

Figure 4: Algorithm to Preserve Conflict-Freedom

The algorithm uses a **case of** structure to differentiate the different possibilities for a given norm N. Line 3 addresses the case when the given norm is a permission: N is simply added to Δ. Lines 4-5 address the case when we attempt to add a ground prohibition to a normative state: if it conflicts with any obligation, then it is discarded; otherwise it is added to the normative state. Lines 6-12 describe the situation when the normative position to be added is a non-ground prohibition. In this case, the algorithm initialises Σ to an empty set and loops (line 9-10) through the norms N' in the old normative state Δ. Upon finding one that conflicts with N, the algorithm updates Σ by adding the newly found conflict set σ to it (line 10). By looping through Δ, we are able to check any conflicts between the new prohibition and the existing obligations, adequately building the annotation Σ to be used when adding N to Δ in line 11.

Lines 13-27 describe how a new obligation is accommodated to an existing normative state. We make use of two initially empty, temporary sets, Δ'_1, Δ'_2. The algorithm loops through Δ (lines 16-25) picking up those annotated prohibitions $\mathsf{N}' \odot \Sigma$ which conflict with the new obligation. There are, however, two cases to deal with: the one when a ground

prohibition is found (line 17), and its exception, covering non-ground prohibitions (line 20). In both cases, the old prohibition is stored in Δ'_1 (lines 18 and 23) to be later removed from Δ (line 26). However, in the case of a non-ground prohibition, the algorithm updates its annotation of conflict sets (line 24). The loop guarantees that an exhaustive (linear) search through a normative state takes place, checking if the new obligation is in conflict with any existing prohibitions, possibly updating the annotations of these conflicting prohibitions. In line 26 the algorithm builds the new updated Δ' by removing the old prohibitions stored in Δ'_1 and adding the updated prohibitions stored in Δ'_2 (if any), as well as the new obligation N.

Our proposed algorithm is correct in that, for a given normative position N and a normative state Δ, it provides a new normative state Δ' in which all prohibitions have annotations recording how they unify with existing obligations. The annotations can be empty, though: this is the case when we have a ground prohibition or a prohibition which does not unify/conflict with any obligation. Permissions do not affect our algorithm and they are appropriately dealt with (line 3). Any attempt to insert a ground prohibition which conflicts, yields the same normative state (line 4). When a new obligation is being added then the algorithm guarantees that all prohibitions are considered (lines 14-27), leading to the removal of conflicting ground prohibitions or the update of annotations of non-ground prohibitions. The algorithm always terminates: the loops are over a finite set Δ and the **conflict** checks and set operations always terminate. The complexity of the algorithm is linear: the set Δ is only examined once for each possible case of norm to be added.

When managing normative states we may also need to remove normative positions. This is straightforward: permissions can be removed without any problems; annotated prohibitions can also be removed without further considerations; obligations, however, require some housekeeping. When an obligation is to be removed, we must check it against all annotated prohibitions in order to update their annotations. We apply the **conflict** check and obtain a unifier, then remove this unifier from the prohibition's annotation. We invoke the removal algorithm as $\mathit{removeNorm}(\mathsf{N}, \Delta)$: it returns a new normative state Δ' in which N has been removed, with possible alterations to other normative positions as explained.

6.2 Enactment of a Normative Structure

The enactment of a normative structure amounts to the parallel, distributed execution of normative scenes and normative transitions. For illustrative purposes, hereafter we shall describe the interplay between the *payment* and *delivery* normative scenes and the normative transition *nt* linking them in the upper half of figure 2. With this aim, consider for instance that $\mathit{obl}(\mathit{inform}(\mathit{jules}, \mathit{client}, \mathit{rod}, \mathit{acc}, \mathit{pay}(\mathit{copper}, 400, 350), T) \in \Delta_{\mathit{payment}}$ and that $\Delta_{\mathit{delivery}}$ holds $\mathit{prh}(\mathit{inform}(\mathit{rod}, \mathit{wm}, \mathit{jules}, \mathit{client}, \mathit{delivered}(Z, Q), T))$. Such states indicate that client Jules is obliged to pay £400 for 350kg of copper to accountant Rod according to the *payment* normative scene, whereas Rod, taking up the role of warehouse manager this time, is prohibited to deliver anything to client Jules according to the *delivery* normative scene.

For each normative scene, the enactment process goes as follows. Firstly, it processes its incoming message queue that contains three types of messages: utterances from the activity it is linked to; and normative commands either to add or remove normative positions. For instance, in our example, the *payment* normative scene collects the illocution $I = \mathit{utt}((\mathit{inform}(\mathit{jules}, \mathit{client}, \mathit{rod}, \mathit{acc}, \mathit{pay}(\mathit{copper}, 400, 350), 35))$ standing for client Jules' pending payment for copper (via arrow A in figure 2). Utterances are time-stamped and subsequently added to the normative state. We would have $\Delta_{\mathit{payment}} = \Delta_{\mathit{payment}} \cup \{I\}$, in our example. Upon receiving normative commands to either add or remove a normative position, the normative scene invokes the corresponding addition or removal algorithm described in Section 6.1. Secondly, the normative scene acknowledges its state change by sending a trigger message to every outgoing normative transition it is connected to. In our example, the *payment* normative scene would be signalling its state change to normative transition *nt*.

For normative transitions, the process works differently. Because each normative transition controls the operation of a single rule, upon receiving a trigger message, it polls every incoming normative scene for substitutions for the relevant illocution schemata on the LHS of its rule. In our example, *nt* (being responsible for the rule described in Section 3.4), would poll the *payment* normative scene (via arrow B) for substitutions. Upon receiving replies from them (in the form of sets of substitutions together with time-stamps), it has to unify substitutions from each of these normative scenes. For each unification it finds, the rule is fired, and hence the corresponding normative command is sent along to the output normative scene. The normative transition then keeps track of the firing message it sent on and of the time-stamps of the normative positions that triggered the firing. This is done to ensure that the very same normative positions in the LHS of a rule only trigger its firing once.

In our example, *nt* would be receiving $\sigma = \{X/\mathit{jules}, Y/\mathit{rod}, Z/\mathit{copper}, Q/350\}$ from the *payment* normative scene. Since the substitions in σ unify with *nt*'s rule, the rule is fired, and the normative command $\mathtt{add}(\mathit{delivery} : \mathit{obl}(\mathit{rod}, \mathit{wm}, \mathit{jules}, \mathit{client}, \mathit{delivered}(\mathit{copper}, 350), T))$ is sent along to the *delivery* normative scene to oblige Rod to deliver to client Jules 350kg of copper. After that, the *delivery* normative scene would invoke the *addNorm* algorithm from figure 4 with $\Delta_{\mathit{delivery}}$ and $N = \mathit{obl}(\mathit{rod}, \mathit{wm}, \mathit{jules}, \mathit{client}, \mathit{delivered}(\mathit{copper}, 350))$ as arguments.

7. RELATED WORK AND CONCLUSIONS

Our contributions in this paper are three-fold. Firstly, we introduce an approach for the management of and reasoning about norms in a distributed manner.

To our knowledge, there is little work published in this direction. In [8, 21], two languages are presented for the distributed enforcement of norms in MAS. However, in both works, each agent has a local message interface that forwards legal messages according to a set of norms. Since these interfaces are local to each agent, norms can only be expressed in terms of actions of that agent. This is a serious disadvantage, *e.g.* when one needs to activate an obligation to one agent due to a certain message of another one.

The second contribution is the proposal of a normative structure. The notion is fruitful because it allows the separation of normative and procedural concerns. The normative structure we propose makes evident the similarity between the propagation of normative positions and the propagation

of tokens in Coloured Petri Nets. That similarity readily suggests a mapping between the two, and gives grounds to a convenient analytical treatment of the normative structure, in general, and the complexity of conflict detection, in particular. The idea of modelling interactions (in the form of conversations) via Petri Nets has been investigated in [18], where the interaction medium and individual agents are modelled as CPN sub-nets that are subsequently combined for analysis. In [5], conversations are first designed and analysed at the level of CPNs and thereafter translated into protocols. Lin et al. [20] map conversation schemata to CPNs. To our knowledge, the use of this representation in the support of conflict detection in regulated MAS has not been reported elsewhere.

Finally, we present a distributed mechanism to resolve normative conflicts. Sartor [25] treats normative conflicts from the point of view of legal theory and suggests a way to order the norms involved. His idea is implemented in [12] but requires a central resource for norm maintenance. The approach to conflict detection and resolution is an adaptation and extension of the work on instantiation graphs reported in [17] and a related algorithm in [27]. The algorithm presented in the current paper can be used to manage normative states distributedly: normative scenes that happen in parallel have an associated normative state Δ to which the algorithm is independently applied each time a new norm is to be introduced.

These three contributions we present in this paper open many possibilities for future work. We should mention first, that as a broad strategy we are working on a generalisation of the notion of normative structure to make it operate with different coordination models, with richer deontic content and on top of different computational realisations of regulated MAS. As a first step in this direction we are taking advantage of the de-coupling between interaction protocols and declarative normative guidance that the normative structure makes available, to provide a normative layer for electronic institutions (as defined in [1]). We expect such coupling will endow electronic institutions with a more flexible –and more expressive– normative environment.

Furthermore, we want to extend our model along several directions: (1) to handle negation and constraints as part of the norm language, and in particular the notion of time; (2) to accommodate multiple, hierarchical norm authorities based on roles, along the lines of Cholvy and Cuppens [3] and power relationships as suggested by Carabelea et al. [2]; (3) to capture in the conflict resolution algorithm different semantics relating the deontic notions by supporting different axiomations (*e.g.*, relative strength of prohibition versus obligation, default deontic notions, deontic inconsistencies).

On the theoretical side, we intend to use analysis techniques of CPNs in order to characterise classes of CPNs (*e.g.*, acyclic, symmetric, etc.) corresponding to families of Normative Structures that are susceptible to tractable offline conflict detection. The combination of these techniques along with our online conflict resolution mechanisms is intended to endow MAS designers with the ability to incorporate norms into their systems in a principled way.

8. REFERENCES

[1] J. L. Arcos, M. Esteva, P. Noriega, J. A. Rodríguez, and C. Sierra. Engineering open environments with electronic institutions. *Journal on Engineering Applications of Artificial Intelligence*, 18(2):191–204, 2005.

[2] C. Carabelea, O. Boissier, and C. Castelfranchi. Using social power to enable agents to reason about being part of a group. In *5th Internat. Workshop, ESAW 2004*, pages 166–177, 2004.

[3] L. Cholvy and F. Cuppens. Solving normative conflicts by merging roles. In *Fifth International Conference on Artificial Intelligence and Law*, Washington, USA, 1995.

[4] S. Christensen and T. B. Haagh. Design CPN - overview of CPN ML syntax. Technical report, University of Aarhus, 1996.

[5] R. S. Cost, Y. Chen, T. W. Finin, Y. Labrou, and Y. Peng. Using colored petri nets for conversation modeling. In *Issues in Agent Communication*, pages 178–192, London, UK, 2000.

[6] F. Dignum. Autonomous Agents with Norms. *Artificial Intelligence and Law*, 7(1):69–79, 1999.

[7] A. Elhag, J. Breuker, and P. Brouwer. On the Formal Analysis of Normative Conflicts. *Information & Comms. Techn. Law*, 9(3):207–217, Oct. 2000.

[8] M. Esteva, W. Vasconcelos, C. Sierra, and J. A. Rodríguez-Aguilar. Norm consistency in electronic institutions. volume 3171 (LNAI), pages 494–505. Springer-Verlag, 2004.

[9] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, New York, U.S.A., 1990.

[10] N. Fornara, F. Viganò, and M. Colombetti. An Event Driven Approach to Norms in Artificial Institutions. In *AAMAS05 Workshop: Agents, Norms and Institutions for Regulated Multiagent Systems (ANI@REM)*, Utrecht, 2005.

[11] D. Gaertner, P. Noriega, and C. Sierra. Extending the BDI architecture with commitments. In *Proceedings of the 9th International Conference of the Catalan Association of Artificial Intelligence*, 2006.

[12] A. García-Camino, P. Noriega, and J.-A. Rodríguez-Aguilar. An Algorithm for Conflict Resolution in Regulated Compound Activities. In *7th Int.Workshop - ESAW '06*, 2006.

[13] A. García-Camino, J.-A. Rodríguez-Aguilar, C. Sierra, and W. Vasconcelos. A Distributed Architecture for Norm-Aware Agent Societies. In *DALT III*, volume 3904 (LNAI), pages 89–105. Springer, 2006.

[14] F. Giunchiglia and L. Serafini. Multi-language hierarchical logics or: How we can do without modal logics. *Artificial Intelligence*, 65(1):29–70, 1994.

[15] J. Habermas. *The Theory of Communication Action, Volume One, Reason and the Rationalization of Society*. Beacon Press, 1984.

[16] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Uses (Volume 1)*. Springer, 1997.

[17] M. Kollingbaum and T. Norman. Strategies for resolving norm conflict in practical reasoning. In *ECAI Workshop Coordination in Emergent Agent Societies 2004*, 2004.

[18] J.-L. Koning, G. Francois, and Y. Demazeau. Formalization and pre-validation for interaction protocols in a multi agent systems. In *ECAI*, pages 298–307, 1998.

[19] B. Kramer and J. Mylopoulos. Knowledge Representation. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 1, pages 743–759. John Wiley & Sons, 1992.

[20] F. Lin, D. H. Norrie, W. Shen, and R. Kremer. A schema-based approach to specifying conversation policies. In *Issues in Agent Communication*, pages 193–204, 2000.

[21] N. Minsky. Law Governed Interaction (LGI): A Distributed Coordination and Control Mechanism (An Introduction, and a Reference Manual). Technical report, Rutgers University, 2005.

[22] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[23] S. Parsons, C. Sierra, and N. Jennings. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998.

[24] A. Ricci and M. Viroli. Coordination Artifacts: A Unifying Abstraction for Engineering Environment-Mediated Coordination in MAS. *Informatica*, 29:433–443, 2005.

[25] G. Sartor. Normative conflicts in legal reasoning. *Artificial Intelligence and Law*, 1(2-3):209–235, June 1992.

[26] M. Sergot. A Computational Theory of Normative Positions. *ACM Trans. Comput. Logic*, 2(4):581–622, 2001.

[27] W. W. Vasconcelos, M. Kollingbaum, and T. Norman. Resolving Conflict and Inconsistency in Norm-Regulated Virtual Organisations. In *Proceedings of AAMAS '07*, Hawai'i, USA, 2007. IFAAMAS.

[28] G. H. von Wright. *Norm and Action: A Logical Inquiry*. Routledge and Kegan Paul, London, 1963.

[29] M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, Chichester, UK, Feb. 2002.