Normative Programming for Organisation Management Infrastructures

Jomi F. Hübner[†*]
[*]Dept Automation and Systems Eng.
Federal University of Santa Catarina
Florianópolis, Brazil
Email: jomi@das.ufsc.br

Olivier Boissier[†]
[†]Ecole Nationale Supérieure des Mines
Saint Etienne, France
Email: {hubner, boissier}@emse.fr

Rafael H. Bordini[‡]
[‡]Intitute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
Email: R.Bordini@inf.ufrgs.br

***Abstract*—Recent work shows a tendency to use programming languages specific to the social aspects of multi-agent systems, for example in programming norms that agents ought to follow. In this paper, we introduce a simple and elegant normative programming language called NPL and show its operational semantics. We then define a particular class of NPL programs that are suitable for programming Organisation Management Infrastructures (OMI) for $\mathcal{M}$OISE, defining a Normative Organisation Programming Language (NOPL). We show how $\mathcal{M}$OISE's Organisation Modelling Language can be translated into NOPL, and briefly describe how this all has been implemented on top of an artifact-based OMI for $\mathcal{M}$OISE.**

I. Introduction

The use of organisational and normative concepts is widely accepted as a suitable approach for the design and implementation of Multi-Agent Systems (MAS) [1]–[4]. Although these concepts are useful for MAS methodologies and therefore used at design time, in this paper we focus on their use at run-time. We conceive of a multi-agent system as a set of agents participating to an organisation by playing roles in it. An important component of MAS is thus the Organisation Management Infrastructure (OMI), which exists in a system to help and supervise agents in the achievement of the purpose of the organisation.

A recent trend in the development of OMIs is to provide languages that the MAS designer (human or artificial in the case of self-organisation) uses to write a program that will define the *organisational* functioning of the system, complementing agent programming languages that defines the *individual* functioning of the system. The former type of languages can focus on different aspects of the overall system, for example: structural aspects (roles and groups) [5], dialogical aspects [2], coordination aspects [6], and normative aspects [7], [8]. The OMI is then responsible for interpreting such a language and providing corresponding services to the agents. For instance, in the case of $\mathcal{M}$OISE$^+$ [4], the designer can program a norm such as "an agent playing the role 'seller' is *obliged* to deliver some goods after being payed by the agent playing role 'buyer"'. The OMI is responsible for identifying the activation of that obligation and to enforce the compliance to that norm by the agents playing the corresponding roles.

We are particularly interested in a flexible and adaptable implementation of OMIs. Such implementation is normally coded using an object-oriented programming language (e.g. Java). However, the exploratory stage of current OMI languages often requires changes in the implementation so that one can experiment with new features. The refactoring of the OMI for such experiments is usually an expensive task that we would like to simplify. Our work therefore addresses one of the main missing ingredients for the *practical* development of sophisticated multi-agent systems where the macro-level requires complex organisational and normative structures in the context of so many different views and approaches still being actively researched by the MAS research community.

This problem is particularly complex for organisation models that consider elements with different natures like groups, roles, common goals, and norms. These elements have their own life cycle, are bound together, and are *constrained* by a set of properties (e.g. role compatibility and cardinality). Our proposal is thus an uniformed approach where all kinds of constraints are expressed by norms. These norms then can be explicitly and flexibly enforced by different mechanisms. The OMI is then mainly concerned with providing such mechanism instead of considering all kinds of constraints. However, we do not want to force the MAS designer to program the organisation using only norms. The designer should program their organisation using more suitable constructors. For example, using a role cardinality constructor to state "a classroom has one professor" instead of a norm like "it is prohibited that two agents play the role professor in the same classroom").

The solution presented in this paper is to translate a more abstract language into another simpler language. The problem of implementing the OMI is thus reduced to a translation problem, which is usually much simpler and less error prone. We start from an organisational modelling language which is then automatically translated into a normative programming language. The language available to the MAS designer has thus more abstract concepts (such as groups, roles, and global plans) than normative languages. More precisely, our starting language is the $\mathcal{M}$OISE Organisation Modelling Language (OML — see Sec. III) and our target language is the Normative Organisation Programming Language (NOPL — Sec. IV). NOPL is a particular class of programs of a normative programming language presented and formalised in this paper (Sec. II). All of this has been implemented on top of our

previous work on OMI where an artifact-based approach, called ORA4MAS, is used (Sec. V).

The main contributions of this work are: (i) a normative programming language and its formalisation using operational semantics; (ii) the translation from an organisational language into the normative language; and (iii) an implemented artifact-based OMI that interprets the target normative language. These contributions are better discussed and placed in the context of the relevant literature in Sec. VI.

II. Normative Programming Language

Although several languages for norms are available, (e.g. [7]–[9]), for this project we need a language that handles *obligations* and *regimentation*. While agents can violate obligations (and sanctions might take place later), regimentation is a preventive strategy of enforcement: agents are not capable to violate a regimented norm [10]. Regimentation is important for an OMI to allow situations where the designer wants to define norms that must be followed because its violation represent a serious risk for the organisation.[1] The current languages either consider obligation or regimentation as enforcement strategies, and do not allow the designers (nor the agents) to dynamically choose the best strategy for their application.

Our language can be relatively simple because we do not need prohibitions nor permission as primitives. By default, everything is permitted and thus the designer does not need to code permissions. Prohibitions can be represented either by regimentation or as an obligation for someone else to decide how to handle the situation (this approach is inspired by the approach by Grossi et al. [10]). For example, consider the norm "it is prohibited to submit a paper with more than 6 pages". In case of regimentation of this norm, tentatives to submit a paper with more than 6 pages will fail. In case this norm is not regimented, the designer has to define a norm such as "when a paper with more than 6 pages is submitted, the chair has to decide wether to accept the submission or not". Another assumption that allowed us to devise a simple language is that we do not consider inconsistent norms. Either the programmer or the program generator are supposed to handle this issue.

A. Syntax

Given the above requirements and simplifications, we introduce below a new Normative Programming Language (NPL) (Fig. 1 contains the definition of its syntax).[2] A normative program *np* is composed of: (i) a set of facts and inference rules (as in Prolog); and (ii) a set of norms. A NPL norm has the general form `norm` id `:` φ `->` ψ, where id is a unique *identifier* of the norm; φ is a formula that determines the *activation condition* for the norm; and ψ is the *consequence* of the activation of the norm. Two types of norm consequences ψ are considered:

- *fail* – `fail`(r): represents the case where the norm is regimented. Argument r represents the reason for the failure;
- *obl* – `obligation`(a, r, g, d): represents the case where a new obligation has to be created for some agent a as the consequence of the norm activation. Argument r is the reason for the obligation (which has to include the norm's id); g is the formula that represents the obligation (a state of the world that the agent must try to bring about, i.e. a goal it has to achieve); and d is the deadline to fulfil the obligation.

A simple example to illustrate the language is given below; we used source code comments to explain the program.

```
np example {
 a(1). a(2).                                  // facts
 ok(X) :- a(A) & b(B) & A>B & X = A*B.  // rule
   // note that b/1 is not defined in the program;
   // it is a dynamic fact provided at run-time

 // alice has 4 hours to achieve a value of X < 5
 norm n1: ok(X) & X > 5
  -> obligation(alice,n1,ok(X) & X<5,`now`+`4 hours`).

 // bob is obliged to sanction alice in case X > 10
 norm n2: ok(X) & X > 10
  -> obligation(bob,n2,sanction(alice),`now`+`1 day`).

 // example of regimented norm; X cannot be > 15
 norm n3: ok(X) & X > 15 -> fail(n3(X)).
}
```

As in other approaches (e.g. [11], [12]), we have a static/declarative aspect of the norm (where norms are expressed in NPL resulting in a normative program) and a dynamic/operational aspect (where obligations are created for existing agents). We call the first aspect simply norm and the second obligation. An obligation has thus a run-time life-cycle. It is created when the activation condition φ of some norm n holds. The activation condition formula is used to instantiate the values of variables a,r,g, and d of the obligation to be created. Once created, the initial state of an obligation is *active* (Fig. 2). The state changes to *fulfilled* when agent a fulfils the norm's obligation g before the deadline d. The obligation state changes to *unfulfilled* when agent a does not fulfil the norm's

np ::= "**np**" *atom* "{" (*rule* | *norm*)* "}"
rule ::= *atom* ["**:-**" *formula*] "**.**"
norm ::= "**norm**" *id* "**:**" *formula* "**->**" (*fail* | *obl*) "**.**"

fail ::= "**fail(**" *atom* "**)**"
obl ::= "**obligation(**"
(*var* | *id*) "**,**" *atom* "**,**" *formula* "**,**" *time* "**)**"

formula ::= *atom* | "**not**" *formula* |
atom ("**&**" | "**|**") *formula*
time ::= "`" ("**now**" |
number ("**second**" | "**minute**" | ...))
"`" [("**+**" | "**-**") *time*]

Fig. 1. EBNF of the NPL

[1] The importance of regimentation is corroborated by relevant implementations of OMI, such as AGR, $\mathcal{S}$-$\mathcal{M}$OISE$^+$, and ISLANDER, which consider regimentation as an important enforcement mechanism.

[2] The non-terminals not included in the specification, *atom*, *id*, *var*, and *number*, correspond, respectively, to predicates, identifiers, variables, and numbers as used in Prolog.

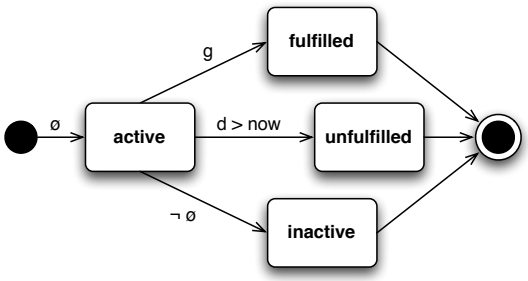


Fig. 2. State Transitions for Obligations

obligation g before the deadline d. As soon as the activation condition of the norm that creates the the obligation (φ) ceases to hold, the state changes to *inactive*. Note that a reference to the norm that led to the creation of the obligation is kept as part of the obligation itself (the r argument), and the activation condition of this norm must remain true for the obligation to stay active; only an active obligation will become either fulfilled or unfulfilled, eventually. Fig. 2 shows the obligation life-cycle.

B. Semantics

We now give semantics to NPL using the well known structural operational semantics approach [13].

A program in NPL is essentially a set of norms where each norm is given according to the grammar in Fig. 1; it can also contain a set of initial facts and inference rules specific to the program's domain (all according to the grammar of the NPL language). The normative system operates in conjunction with an agent execution system; the former is constantly fed by the latter with "facts" which, possibly together with the domain rules, express the current state of the execution system. Any change in such facts leads to a potential change in the state of the normative system, and the execution system checks that the normative system is still in a sound state before committing towards particular execution steps; similarly, it can have access to current obligations generated by the normative system. The overall system's clock also causes potential changes in the state of the transition system, by changing the time component of its configuration.

As we use operational semantics to give semantics to the normative programming language (i.e. the language used to program the normative system specifically), we first need to define a configuration of the transition system that will be defined through the semantic rules presented later. A configuration of our normative system, giving semantics to NPL, is a tuple $\langle F, N, \top, OS, t\rangle$ where:

- F is a set of facts received from the execution system and possibly rules expressing domain knowledge. The former works as a form of input from the agent execution system to the normative system. Each formula $f \in F$ is, as explained earlier, an atomic first order formula or a Horn clause.
- N is a set of norms, where each norm $n \in N$ is a norm in the syntax defined for *norm* in the grammar in Fig. 1.
- The state of the normative system is either a sound state denoted by $\top$ or a failure state denoted by $\bot$; the latter is caused by *regimentation* through the `fail(_)` language construct within norms. This is accessible to the agent execution system which prevents the execution of the action which would lead to the facts causing the failure state, and rolls back the facts about the state of the execution system.
- OS is a set of obligations, each accompanied by its current state; each element $os \in OS$ is of the form $\langle o, ost\rangle$ where o is an obligation, again according to the syntax for obligations given in Fig. 1, and $ost \in \{\mathbf{active}, \mathbf{fulfilled}, \mathbf{unfulfilled}, \mathbf{inactive}\}$ (the possible states of each individual obligation). This is also of interest to the agent execution system and thus accessible to it.
- t is the current time which is automatically changed by the underlying execution system, using, of course, a discrete, linear notion of time. For the purpose of the operational semantics, it is assumed that all rules that apply at a given time are actually applied before the system changes the state to the next time.

Given a normative program P — which is, remember, a set of facts and rules (P_F) and a set of norms (P_N) written in NPL — the initial configuration of the normative system (before the system execution starts) is $\langle P_F, P_N, \top, \emptyset, 0\rangle$.

In the semantic rules, we use the notation T_c to denote the component c of tuple T. The semantic rules are as follows.

1) Norms: The rule below formalises *regimentation*: when any norm n becomes active — i.e. its *condition* component holds in the current state — and its *consequence* is `fail(_)`, we move to a configuration where the normative state is no longer sound but a failure state ($\bot$). Note that we use n_φ to refer to the condition part of norm n (the formula between "`:`" and "`->`" in NPL's syntax) and n_ψ to refer to the consequence part of n (the formula after "`->`").

$$\frac{n \in N \qquad F \models n_\varphi \qquad n_\psi = \texttt{fail(\_)}}{\langle F, N, \top, OS, t\rangle \longrightarrow \langle F, N, \bot, OS, t\rangle} \qquad \textbf{(Regim)}$$

The underlying execution system, after realising a failure state caused by Rule **Regim** above, needs to ensure the facts are rolled back to the previously consistent state, which will make the following rule apply.

$$\frac{\forall n \in N.(F \models n_\varphi \Rightarrow n_\psi \neq \texttt{fail(\_)})}{\langle F, N, \bot, OS, t\rangle \longrightarrow \langle F, N, \top, OS, t\rangle} \qquad \textbf{(Consist)}$$

The next rule is similar to Rule **Regim** but instead of failure, the consequence is the creation of an obligation. In the rule, 'm.g.u.' means "most general unifier" as in Prolog-like unification; the notation $t\theta$ means the application of the variable substitution function θ to formula t. Note that we required that the deadlines of newly created obligations are not

yet past. The notation $\stackrel{\text{obl}}{=}$ is used for equality of obligations, which ignores the deadline in the comparison. That is, we define that an obligation $obligation(a, r, g, d)$ is equals to an obligation $obligation(a', r', g', d')$ if and only if $a = a'$, $r = r'$, and $g = g'$. Because of this, Rule **Oblig** does not allow the creation of the same obligation with two different deadlines. Note also that if there already exists an equal obligation but it has become inactive, this does not prevent the creation of the obligation.

$$\frac{\begin{array}{c} n \in N \qquad F \models n_\varphi \qquad n_\psi = o \qquad o\theta_d > t \\ \neg\exists \langle o', ost\rangle \in OS \;.\; (o' \stackrel{\text{obl}}{=} o\theta \wedge ost \neq \mathbf{inactive}) \end{array}}{\begin{array}{c} \langle F, N, \top, OS, t\rangle \longrightarrow \\ \langle F, N, \top, OS \cup \langle o\theta, \mathbf{active}\rangle, t\rangle \end{array}} \quad \textbf{(Oblig)}$$

where θ is the m.g.u. such that $F \models o\theta$

2) Obligations: Recall that a NPL obligation has the general form `obligation(a,r,g,d)`. With a slight abuse of notation, we shall use o_a to refer to the agent that has the obligation o; o_r to refer to the reason for obligation o; o_g to refer to the state of the world that agent o_a is obliged to achieve (the *goal* the agent should adopt); and o_d to refer to the deadline for the agent to do so. An important aspect of obligation syntax is that the NPL parser always ensures that the programmer used the norm's id as predicate symbol in o_r and so in the semantics, when we say o_r, we are actually referring to the activation condition n_φ of the norm used to create the obligation.

Rule **Fulfil** says that the state of an active obligation o should be changed to **fulfilled** if the state of the world o_g that the agent agent was obliged to achieve has already been achieved (i.e. the domain rules and facts from the underlying system imply g). Note however that such state must have been achieved *within the deadline*.

$$\frac{\begin{array}{c} os \in OS \qquad os = \langle o, \mathbf{active}\rangle \\ F \models o_g \qquad o_d \geq t \end{array}}{\begin{array}{c} \langle F, N, \top, OS, t\rangle \longrightarrow \\ \langle F, N, \top, (OS \setminus \{os\}) \cup \{\langle o, \mathbf{fulfilled}\rangle\}, t\rangle \end{array}} \quad \textbf{(Fulfil)}$$

Rule **Unfulfil** says that the state of an *active obligation* o should be changed to **unfulfilled** if the deadline is already past; note that the rule above would have changed the status to **fulfilled** so the obligation would no longer be active if it had been achieved in time.

$$\frac{os \in OS \qquad os = \langle o, \mathbf{active}\rangle \qquad o_d < t}{\begin{array}{c} \langle F, N, \top, OS, t\rangle \longrightarrow \\ \langle F, N, \top, (OS \setminus \{os\}) \cup \{\langle o, \mathbf{unfulfilled}\rangle\}, t\rangle \end{array}} \quad \textbf{(Unfulfil)}$$

Rule **Inactive** says that the state of an active obligation o should be changed to **inactive** if the reason (i.e. motivation) for the obligation no longer holds in the current system state reflected in F.

$$\frac{os \in OS \qquad os = \langle o, \mathbf{active}\rangle \qquad F \not\models o_r}{\begin{array}{c} \langle F, N, \top, OS, t\rangle \longrightarrow \\ \langle F, N, \top, (OS \setminus \{os\}) \cup \{\langle o, \mathbf{inactive}\rangle\}, t\rangle \end{array}} \quad \textbf{(Inactive)}$$

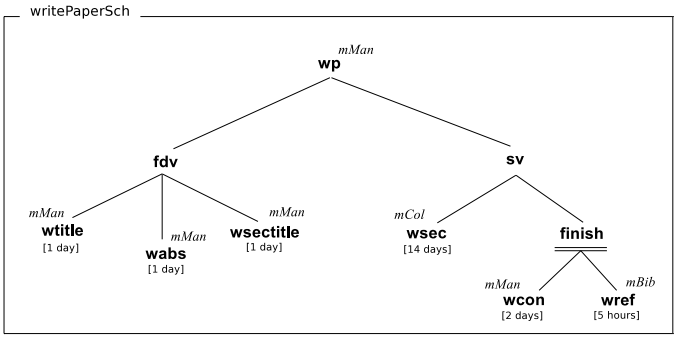
III. $\mathcal{M}$OISE ORGANISATIONAL MODELLING LANGUAGE

$\mathcal{M}$OISE proposes an organisational modelling language (OML) that explicitly decomposes the specification of organisation into structural, functional, and normative dimensions [4]. The structural dimension specifies the *roles*, *groups*, and *links* of the organisation. The definition of roles states that when an agent chooses to play some role in a group, it is accepting some behavioural constraints and rights related to this role. The functional dimension specifies how the *global collective goals* should be achieved, i.e. how these goals are decomposed (within global *plans*), grouped in coherent sets (through *missions*) to be distributed among the agents. The decomposition of global goals results in a goal-tree, called *scheme*, where the leaf-goals can be achieved individually by the agents. The normative dimension is added in order to bind the structural dimension with the functional one by means of the specification of the roles' *permissions* and *obligations* within missions.

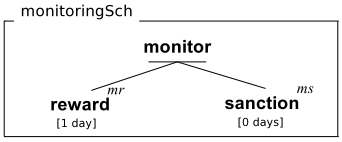
As an illustrative and simple example of an organisation specified using $\mathcal{M}$OISE$^+$, we consider agents that aim at writing a paper and therefore have an organisational specification to help them collaborate. Due to lack of space, we will focus on the functional and normative dimensions in the remainder of this paper. For the structure of the organisation, it is enough to know that there is only one group (`wpgroup`) where two roles (*editor* and *writer*) can be played.

To coordinate the achievement of the goal of writing a paper, a scheme is defined in the functional specification of the organisation (Fig. 3(a)). In this scheme, a draft version of the paper has to be written first (identified by the goal *fdv* in Fig. 3(a)). This goal is decomposed into three sub-goals: write a title, an abstract, and the section titles; the sub-goals have to be achieved in this very sequence. Other goals, such as *finish*, have sub-goals that can be achieved in parallel. The specification also includes a "time-to-fulfil" (TTF) attribute for goals indicating how much time an agent has to achieve the goal. The goals of this scheme are distributed in three missions which have specific cardinalities (see Fig. 3(c)): the mission $mMan$ is for the general management of the process (one and only one agent must commit to it), mission $mCol$ is for the collaboration in writing the paper's content (from one to five agents can commit to it), and mission $mBib$ is for gathering the references for the paper (one and only one agent must commit to it). A mission defines all goals an agent commits to when participating in the execution of a scheme; for example, a commitment to mission $mMan$ is effectively a commitment to achieve four goals of the scheme. Goals without an assigned mission are satisfied by the achievement of their sub-goals.

The normative specification relates roles to missions (Table I). For example, norm n2 states that any agent playing the role *writer* has one day to commit to mission $mCol$. Designers can also define their own application-dependent conditions (as



(a) Paper Writing Scheme



(b) Monitoring Scheme

mission	cardinality
$mMan$	1..1
$mCol$	1..5
$mBib$	1..1
mr	1..1
ms	1..1

(c) Mission Cardinalities

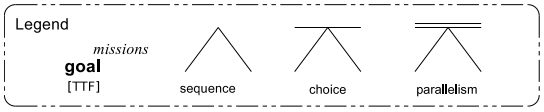


Fig. 3. Functional Specification for the Paper Writing Example

id	condition	role	type	mission	TTF
n1		editor	per	$mMan$	–
n2		writer	obl	$mCol$	1 day
n3		writer	obl	$mBib$	1 day
n4	unfulfilled(n2)	editor	obl	ms	3 hours
n5	fulfilled(n3)	editor	obl	mr	3 hours
n6	#mc	editor	obl	ms	1 hour

#mc stands for the condition "more agents committed to a mission than permitted by the mission cardinality"

TABLE I
NORMATIVE SPECIFICATION FOR THE PAPER WRITING EXAMPLE

in norms n4–n6). Norms n4 and n5 define sanction and reward strategies for fulfilment and unfulfilment of norms n2 and n3 respectively. Norm n5 can be read as "the agent playing role 'editor' has 3 hours to commit to mission mr when norm n3 is fulfilled". Once committed to mission mr, the editor has to achieve the goal *reward*. Note that a norm in $\mathcal{M}$OISE is always an obligation or permission to commit to a mission. Goals are therefore indirectly linked to roles since a mission is a set of goals.

IV. NORMATIVE ORGANISATION PROGRAMMING LANGUAGE

The NOPL is a particular class of NPL programs applied to $\mathcal{M}$OISE. The syntax and semantics are the same as presented in Sec. II, but the set of facts, rules, and norms are specific for $\mathcal{M}$OISE model and the organisational artifacts presented in Sec. V. The main idea is that an OS is translated to different programs in NOPL, such programs define then the management of norms for groups and schemes. In this section we consider only the programs for schemes.

A. Facts

For scheme programs, the following facts, defined in the OS, are considered:

- `scheme_mission(`m`,`min`,`max`)`: is a fact that defines the cardinality of a mission (e.g. `scheme_mission(mCol,1,5)`).
- `goal(`m`,`g`,`$pre\text{-}cond$`,'`ttf`')`: is a fact that defines the arguments for a goal g: its mission, pre-conditions, and TTF (e.g. `goal(mMan,wsec,[wcon],'2 days')`).

The NOPL also defines some dynamic facts that represent the current state of the organisation and will be provided by the artifact that manage the scheme instance:

- `plays(`a`,`ρ`,`gr`)`: agent a plays the role ρ in the group instance identified by gr.
- `responsible(`gr`,`s`)`: the group instance gr is responsible for the missions of scheme instance s.
- `committed(`a`,`m`,`s`)`: agent a is committed to mission m in scheme s.
- `achieved(`s`,`g`,`a`)`: goal g in scheme s has been achieved by agent a.

B. Rules

Besides facts, we define some rules that will be useful for the norms. The rules are used to infer the state of the scheme (e.g. whether it is well-formed) and goals (e.g. whether it is ready to be achieved or not). Note that the semantics of *well-formed* and *ready goal* are formally given by these rules. As an example, some of such rules for the paper writing scheme are listed below.

```
// number of players of a mission M in scheme S
mplayers(M,S,V) :- .count(committed(_,M,S),V).

// status of a scheme S
well_formed(S) :-
 mplayers(mBib,S,V1) & V1 >= 1 & V1 <= 1 &
 mplayers(mCol,S,V2) & V2 >= 1 & V2 <= 5 &
 mplayers(mMan,S,V3) & V3 >= 1 & V3 <= 1.

// ready goals: all pre-conditions have been achieved
ready(S,G) :-
 goal(_, G, PCG, _) & all_achieved(S,PCG).
all_achieved(_,[]).
all_achieved(S,[G|T]) :-
 achieved(S,G,_) & all_achieved(S,T).
```

C. Norms

We have three classes of norms in NOPL: norms for goals, norms for properties, and domain norms (which are explicitly stated in the normative specification). For the former class, we have the following norm:

```
// agents are obliged to fulfil their ready goals
norm ngoa:
  committed(A,M,S) & goal(M,G,_,D) &
  well_formed(S) & ready(S,G)
-> obligation(A,ngoa,achieved(S,G,A),'now' + D).
```

This norm can be read as "when an agent A: (1) is committed to a mission M that (2) includes a goal G, and (3) the mission's

scheme is well-formed, and (4) the goal is ready, then agent A is obliged to achieve the goal G before the deadline for the goal". This norm thus gives a precise semantics for commitment. It also illustrates the advantage of using a translation to implement the OMI instead of an object oriented programming language. For example, if some application or experiment requires a semantics of commitment where the agent is obliged to achieve the goal even if the scheme is not well-formed, it is simply a matter of changing the translation to a norm that does not include the `well_formed(S)` predicate in the activation condition of the norm. One could even conceive an application using schemes being managed by different NOPL programs (i.e. schemes translated differently).

For the second class of norms, only the mission cardinality property is considered in this paper since other properties are handled in a similar way. In the case of mission cardinality, the norm has to define the consequences of a circumstance where there are more agents committed to a mission than permitted in the scheme specification. As presented in Sec. II, two kinds of consequences are possible, obligation and regimentation, and the designer chooses one or the other when writing the OS. Regimentation is the default consequence and it is used when there is no norm with condition #mc in the normative specification. Otherwise, as in norm n6 of Table I, the consequence will be an obligation. The norm for mission cardinality regimentation is:

```
// norm for the cardinality regimentation
norm mission_cardinality:
   scheme_mission(M,_,MMax) &
   mplayers(M,S,MP) &  MP > MMax
-> fail(mission_cardinality).
```

and the norm without regimentation is:

```
// norm for the cardinality regimentation
norm mission_cardinality:
   scheme_mission(M,_,MMax) &
   mplayers(M,S,MP) &  MP > MMax
   responsible(Gr,S) & plays(A,editor,Gr)
-> obligation(A,mission_cardinality,
              committed(A,ms,_), `now`+`1 hour`).
```

where the agent playing editor is obliged to commit to the mission ms in one hour.

For the third class of norms, each norm in the normative specification of the OML has a corresponding norm in NOPL. Whereas OML obligations refer to roles and missions, NPL requires that obligations are for agents and towards a goal. The NOPL norm thus identifies the agents playing the role in groups responsible for the scheme and, if the number of players still does not reach the maximum, the agent is obliged to achieve a state where it is committed to the mission. For example, the NOPL norm for norm n2 of Table I is:

```
norm n2:
  plays(A,writer,Gr) & responsible(Gr,S) &
  mplayers(mCol,S,V) & V < 5
-> obligation(A,n2,committed(A,mCol,S),`now`+`1 day`).
```

V. Artifact-Based Architecture

The proposals of this paper have been implemented on an OMI that follows the Agent & Artifact [14], [15]. In this approach, a set of organisational artifacts is available in the MAS environment providing operations for the agents so that they can interact with the OMI. For example, each scheme instance is managed by an artifact. We can effortlessly distribute the OMI by deploying as many artifacts as necessary for the application.

Each organisational artifact has an NPL interpreter loaded with (i) the NOPL program automatically generated from the OS for the type of the artifact and (ii) dynamic facts representing the current state of (part of) the organisation. The interpreter is then used to compute: (i) whether some operation will bring the organisation into a inconsistent state (where inconsistency is defined by the designer by means of regimentations), and (ii) the current state of the obligations. The following algorithm, implemented on top of CArtAgO [16], shows the general pattern we used to implement every operation (e.g. role adoption, commitment to mission) in the organisational artifacts. In this new approach, the artifacts still provide an interface for the agents, and are now mostly programmed in NOPL instead of Java.

```
// let oe be the current state of the organisation managed by the
   artifact
// let p be the current NOPL program
// let npi be the NPL interpreter
when an operation o is triggered by agent a do
    oe' ← oe // creates a "backup" of current oe
    executes operation o to change oe
    f ← a list of predicates representing oe
    r ← npi(p, f) // runs the interpreter for the new state
    if r = fail then
        oe ← oe' // restore the state backup
        fail operation o
    else
        update obligations in the observable properties
        succeed operation o
```

We also developed a program that automatically generate the NOPL given an OS, however, due the lack of space, it is not presented here. The reader will find more details about this architecture, the translation, and a complete implementation of this OMI at https://sourceforge.net/scm/?type=svn&group_id=163721.

VI. Related Works

This work is based on several approaches to organisation, institutions, and norms (cited throughout the paper). In this section, we briefly relate and compare our main contributions to such work.

The first contribution of the paper, the NPL, should be considered specially for two properties of the language: its simplicity and its formalisation (that led to an available implementation). Similar work has been done by Tinnemeier et al. [7], where the operational semantics for a normative language was also proposed. They assume the availability of a snapshot of the global state of the organisation to evaluate activation of norms, which may hinder the implementation in a distributed scenario. Our NPL also requires a snapshot of

the organisational artifact state to evaluate norms, however the distribution problem is solved by generating different normative programs for several distributed artifacts where only the local state of the organisation is required. Another important difference is that in our approach the designer specifies the organisation in a higher-level language (OML) that is translated into a normative programmin language (NOPL).

Regarding the second contribution, namely the automatic translation, we were inspired by work on ISLANDER [8], [17]. The main difference here is the initial and target languages. While they translate a normative specification into a rule-based language, we start from an organisational language and target at a normative language. It is simpler to translate organisational norms into NPL norms, since we have norms in both sides of the translation it is a 1-to-1 translation, than translate organisational norms into rules.

Regarding the third contribution, the OMI, we started from ORA4MAS [15]. The advantages of the approach presented here are twofold: (i) it is easier to change the translation than the Java implementation of the OMI; and (ii) from the operational semantics of NPL and the formal translation we are taking significant steps towards a formal semantics for $\mathcal{M}$OISE.

VII. Conclusion

In this paper we showed an approach for translating an organisation specification written in $\mathcal{M}$OISE OML into a normative program that can be interpreted by an artifact based OMI. Focusing on the translation instead of Java coding, we have brought flexibility to the development of the OMI. We also stressed the point that such a normative language can be based on only two basic concepts: regimentation and obligation. Prohibitions are considered either as regimentation or as an obligation for someone else. The resulting NPL is thus simpler to formalise (only 6 rules in operational semantics) and implement. Future work will explore NPL translations for other organisational and institutional languages. We also plan to prove correctness of the translation from OML into NOPL in future work.

References

[1] O. Boissier, J. F. Hübner, and J. S. Sichman, "Organization oriented programming from closed to open organizations," in *Engineering Societies in the Agents World VII (ESAW 06)*, ser. LNCS, G. O'Hare, M. O'Grady, O. Dikenelli, and A. Ricci, Eds., vol. 4457. Springer, 2007, pp. 86–105.

[2] M. Esteva, D. de la Cruz, and C. Sierra, "ISLANDER: an electronic institutions editor," in *Proc. of the First International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2002)*, ser. LNAI 1191, C. Castelfranchi and W. L. Johnson, Eds. Springer, 2002, pp. 1045–1052.

[3] V. Dignum, J. Vazquez-Salceda, and F. Dignum, "OMNI: Introducing social structure, norms and ontologies into agent organizations," in *Proc. of the Programming Multi-Agent Systems (ProMAS 2004)*, ser. LNAI 3346, R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, Eds. Berlin: Springer, 2004.

[4] J. F. Hübner, J. S. Sichman, and O. Boissier, "Developing organised multi-agent systems using the MOISE+ model: Programming issues at the system and agent levels," *International Journal of Agent-Oriented Software Engineering*, vol. 1, no. 3/4, pp. 370–395, 2007.

[5] J. Ferber and O. Gutknecht, "A meta-model for the analysis and design of organizations in multi-agents systems," in *Proc. of the 3rd International Conference on Multi-Agent Systems (ICMAS'98)*, Y. Demazeau, Ed. IEEE Press, 1998, pp. 128–135.

[6] D. V. Pynadath and M. Tambe, "An automated teamwork infrastructure for heterogeneous software agents and humans," *Autonomous Agents and Multi-Agent Systems*, vol. 7, no. 1-2, pp. 71–100, 2003.

[7] N. Tinnemeier, M. Dastani, and J.-J. Meyer, "Roles and norms for programming agent organizations," in *Proc. of AAMAS 09*, J. Sichman, K. Decker, C. Sierra, and C. Castelfranchi, Eds., 2009.

[8] A. García-Camino, J. A. Rodríguez-Aguilar, C. Sierra, and W. Vasconcelos, "Constraining rule-based programming norms for electronic institutions," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 18, no. 1, pp. 186–217, Feb 2009.

[9] F. L. y López, M. Luck, and M. d'Inverno, "Constraining autonomy through norms," in *Proc. of first ICMAS*. ACM, 2002, pp. 674 – 681.

[10] D. Grossi, H. Aldewered, and F. Dignum, "*Ubi Lex, Ibi Poena*: Designing norm enforcement in e-institutions," in *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, ser. LNAI, P. Noriega, J. Vázquez-Salceda, G. Boella, O. Boissier, V. Dignum, N. Fornara, and E. Matson, Eds., vol. 4386. Springer, 2007, pp. 101–114, revised Selected Papers.

[11] N. Fornara and M. Colombetti, "Specifying and enforcing norms in artificial institutions," in *Proc. of the 4th European Workshop on Multi-Agent Systems (EUMAS 06)*, A. Omicini, B. Dunin-Keplicz, and J. Padget, Eds., 2006.

[12] J. Vázquez-Salceda, H. Aldewereld, and F. Dignum, "Norms in multiagent systems: some implementation guidelines," in *Proc. of the Second European Workshop on Multi-Agent Systems (EUMAS 2004)*, 2004,

[13] G. D. Plotkin, "A structural approach to operational semantics," Computer Science Department, Aarhus University, Aarhus, Denmark, Tech. Rep., 1981.

[14] A. Omicini, A. Ricci, and M. Viroli, "Artifacts in the A&A meta-model for multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 17 (3), pp. 432–456, Dec. 2008.

[15] J. F. Hübner, O. Boissier, R. Kitio, and A. Ricci, "Instrumenting multi-agent organisations with organisational artifacts and agents: "giving the organisational power back to the agents"," *Journal of Autonomous Agents and Multi-Agent Systems*, 2009.

[16] A. Ricci, M. Piunti, M. Viroli, and A. Omicini, "Environment programming in CArtAgO," in *Multi-Agent Programming: Languages, Tools and Applications*, R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, Eds. Springer, 2009, ch. 8, pp. 259–288.

[17] V. T. da Silva, "From the specification to the implementation of norms: an automatic approach to generate rules from norm to govern the behaviour of agents," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 1, pp. 113–155, Aug 2008.