From Organisation Specification to Normative Programming in Multi-Agent Organisations

Jomi F. Hübner[1], Olivier Boissier[2], and Rafael H. Bordini[3]

[1] Dept Automation and Systems Engineering
Federal University of Santa Catarina
jomi@das.ufsc.br
[2] Ecole Nationale Supérieure des Mines
Saint Etienne, France
boissier@emse.fr
[3] Institute of Informatics
Federal University of Rio Grande do Sul
R.Bordini@inf.ufrgs.br

Abstract. In this paper, we show how we can automatically translate high-level organisation modelling languages into simpler languages based on the idea of normative programming. With this approach, while designers and agents still use a highly abstract organisational modelling language to specify and reason about the multi-agent organisation, the development of the organisation management infrastructure is facilitated in the following manner. The high-level organisation specification is automatically translated into a simple normative programming language that we have recently introduced and for which we have given formal semantics. The organisation management infrastructure can then be based on an interpreter for the simpler normative language. We illustrate the approach showing how $\mathcal{M}$OISE's organisation modelling language (with primitives such as roles, groups, and goals) can be translated into our normative programming language (with primitives such as norms and obligations). We briefly describe how this all has been implemented on top of ORA4MAS, the distributed artifact-based organisation management infrastructure for $\mathcal{M}$OISE.

1 Introduction

The use of organisational and normative concepts is widely accepted as an appropriate approach for the design and implementation of Multi-Agent Systems (MAS) [3]. They are thus present in several languages and frameworks for intelligent multi-agent systems. They are also used at runtime to make the agents aware of the organisations in which they take part and to support and monitor their activities. While the support aspect is important for any large-scale system, the monitoring one is particularly relevant for open MAS where the behaviour of the entering agents is unknown. A clear trend in the development of such systems is to provide organisation-oriented modelling languages that the MAS designer (human or agent, in the case of self-organisation) uses to write a program that prescribes the *organisational* functioning of the system [5, 4, 3, 15, 17, 7], complementing agent programming languages that define the *individual* functioning

within such system. These languages are interpreted by an Organisation Management Infrastructures (OMI) to realise the monitoring aspect of agent organisations.

In our work, we are particularly interested in flexible and adaptable development of OMIs. The exploratory stage of current OMIs often requires changes in their implementations so that one can experiment with new features. The refactoring of the OMI for such experiments, when the interpreter for the high-level modelling language has ad hoc implementations, is usually an expensive task that we wish to simplify. Our approach aims at expressing the various different constructs of the high-level modelling language into a unified framework by means of *norms*. The OMI is then realised by a mechanism for interpreting and managing the status of such norms instead of specific mechanisms for each of the constructs of the richer modelling language.

The solution proposed allows us to keep the language available to the designer and agents with high-level concepts such as groups, roles, and global plans. That language can be translated into (or compiled to) a simpler normative programming language that is then interpreted by the OMI. The problem of implementing the OMI is thereby reduced to: (1) the development of an interpreter for the normative language and (2) a translation problem (from the organisation modelling language to the normative programming language). More precisely, our starting language is the $\mathcal{M}$OISE Organisation Modelling Language (OML — see Sec. 3) and our target language is the Normative Organisation Programming Language (NOPL — see Sec. 4). NOPL is a particular class of a normative programming language that we introduced and formalised in [9], and we summarise it in Sec. 2. The translation process from OML into NOPL is fully automatic thanks to the contributions in this paper. All of this has been implemented on top of ORA4MAS, a distributed artifact-based approach for OMI (Sec. 5). This paper also gives an interesting contribution in elucidating the power of the *norm* abstraction in normative programming languages, which is enough to cover organisation specifications. The longer organisation specification/program translated into a normative programming language, although less readable for humans, is efficiently interpreted within OMI implementations.

The main components of our approach are, therefore: (i) a normative organisation programming language; (ii) the translation from an organisational modelling language into the normative organisation programming language; and (iii) an implemented artifact-based OMI that interprets the target normative language. The contributions of our approach are better discussed and placed in the context of the relevant literature in Sec. 6.

2 Normative Programming Language

A normative language is usually based on three primitives (obligation, permission, and prohibition) and two enforcement strategies (sanction and regimentation) [17, 20, 7]. While sanction is a reactive strategy applied after the event of a violation, regimentation is a preventive strategy whereby agents are not capable of violation [12]. Regimentation is important for an OMI since it allows the designer to define norms that must be followed because their violation present serious risks for the organisation.

The language we created is based on the following assumptions. (i) Permissions are defined by omission, as in [8]. (ii) Prohibitions are represented either by regimentation or as an obligation for someone else to decide how to handle the situation. For example, consider the norm "it is prohibited to submit a paper with more than 16 pages". In case of regimentation of this norm, attempts to submit a paper with more than 16 pages will fail (i.e. they will be prevented from taking place). In case this norm is not to be regimented, the designer could define a norm such as "when a paper with more than 16 pages is submitted, the chair must decide whether to accept the submission or not". (iii) Sanctions are represented as obligations (i.e. someone else is *obliged* to apply the sanction). (iv) Finally, norms are assumed to be consistent (either the programmer or program generator are supposed to handle this issue). Thus, the language can be relatively simple, reduced to two main constructs: *obligation* and *regimentation*.

Given the above requirements and simplifications, we can now introduce our Normative Programming Language (NPL). A normative program *np* is composed of: (i) a set of facts and inference rules (following the syntax used in *Jason* [1]); and (ii) a set of norms. A NPL norm has the general form

$$\texttt{norm}\ id\ \texttt{:}\ \varphi\ \texttt{->}\ \psi$$

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 available:

- *fail* – $\texttt{fail}(r)$: represents the case where the norm is regimented; argument r represents the reason for the failure;
- *obl* – $\texttt{obligation}(a, r, g, d)$: represents the case where an obligation for some agent a is created. Argument r is the reason for the obligation (which has to include the id of the norm that originated the obligation); g is the formula that represents the obligation itself (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 use 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
```

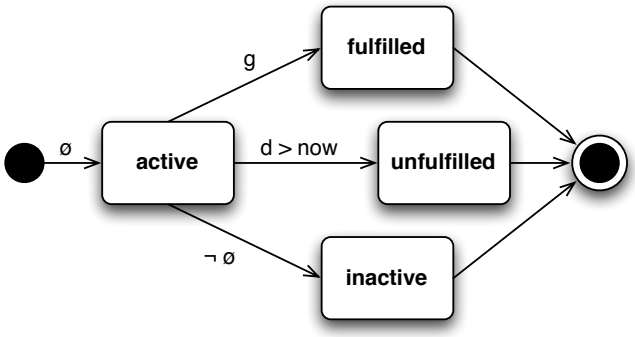


Fig. 1. State Transitions for Obligations

```
  norm n3: ok(X) & X > 15 -> fail(n3(X)).
}
```

As in other approaches (e.g. [6, 19]), a normative program expresses both static and declarative aspects of norms. The dynamic aspects result from the interpretation of such programs and the consequent creation of obligations for participating agents. An obligation has therefore 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. 1). 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 obligation g before the deadline d. As soon as the activation condition of the norm that created 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 (in 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, when the deadline is eventually reached. Fig. 1 shows the obligation life-cycle.

The syntax and semantics of NPL was introduced in [9]. the semantics was given using the well-known structural operational semantics approach.

3 $\mathcal{M}$OISE

The $\mathcal{M}$OISE framework includes an organisational modelling language (OML) that explicitly decomposes the specification of organisations into structural, functional, and normative dimensions [11]. The structural dimension includes the *roles*, *groups*, and *links* (e.g. communication) within the organisation. The definition of roles is such 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 determines how the *global collective goals* should be achieved, i.e. how these goals are decomposed (through global *plans*) and grouped into coherent sets of subgoals (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 binds the structural dimension with the functional one by means of the specification of *permissions* and *obligations* towards missions given to particular roles. When an agent chooses to play some role in a group, it accepts these permissions and obligations.

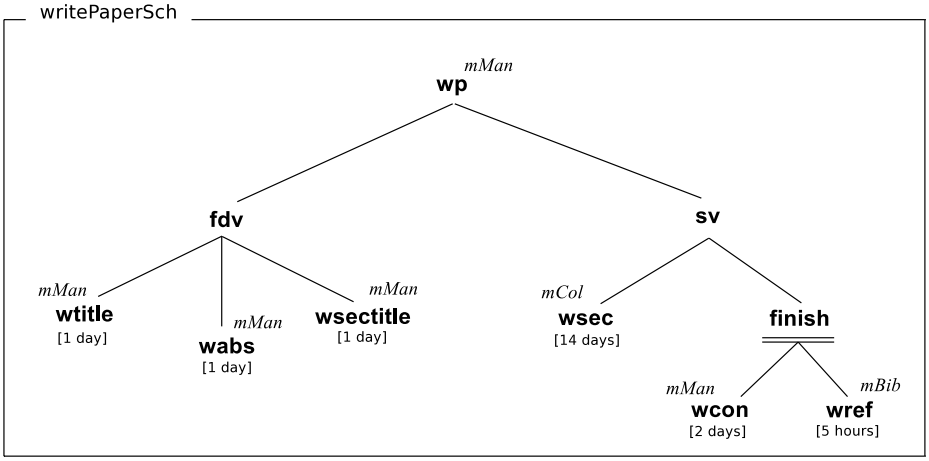
As an illustrative and simple example of an organisation specified using $\mathcal{M}$OISE$^+$, we consider a scenario where agents aiming to write a paper together use an organisational specification to help them collaborate. We will focus on the functional and normative dimensions in the remainder of this paper. As for the structure of the organisation, it suffices 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. 2(a)). In this scheme, a draft version of the paper has to be written first (identified by the goal *fdv* in Fig. 2(a)). This goal is decomposed into three subgoals: writing a title, an abstract, and the section titles; the subgoals have to be achieved in this very sequence. Other goals, such as *finish*, have subgoals that can be achieved in parallel. The specification also includes a "time-to-fulfil" (TTF) attribute for goals indicating a deadline for the agent to achieve the goal. The goals of this scheme are distributed into three missions which have specific cardinalities (see Fig. 2(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 content (from one up 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 the 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 (e.g. *fdv*) are satisfied through the achievement of their subgoals.

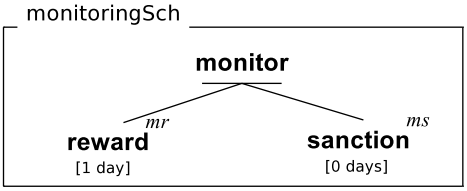
Table 1. Normative Specification for the Paper Writing Example

id	condition	role	type	mission	TTF
`n1`		editor	per	$mMan$	–
`n2`		writer	obl	$mCol$	1 day
`n3`		writer	obl	$mBib$	1 day
`n4`	violation(`n2`)	editor	obl	ms	3 hours
`n5`	conformance(`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"



(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

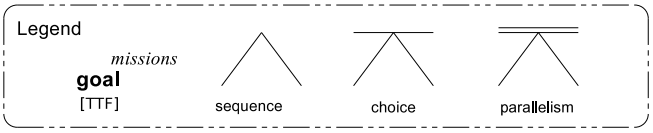


Fig. 2. Functional Specification for the Paper Writing Example

The normative specification relates roles and missions through norms (Table 1). For example, the oml-norm[4] `n2` states that any agent playing the role *writer* has one day to commit to mission $mCol$. Designers can also express application-dependent conditions (as in oml-norms `n4`–`n6`). Oml-norms `n4` and `n5` define sanction and reward strategies for violation and conformance of oml-norms `n2` and `n3` respectively. Oml-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 an oml-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. Prohibitions are assumed 'by default' with respect to the specified missions: if the normative specification does not include a permission or obligation for a role-mission pair, it is assumed that the role does not grant the right to commit to the mission.

The OML is accompanied by a graphical language (see Fig. 2) and XML is used to store the organisational specifications (OS). In Sec. 4, instead of considering all the details of the graphical or the XML representation of an OS, we will consider the data structure produced by the OML parser. The data structure for an OS contains a set $\mathcal{FS}$ of scheme specifications and a set $\mathcal{NS}$ of oml-norms (again, only the functional and normative dimensions are being considered here).

When a scheme specification S is parsed, a tuple of the following type is produced:

$$\langle id, \mathcal{M}, maxmp, minmp, \mathcal{G}, gm, gpc, ttf, g_r \rangle$$

where

- id is a unique identification for S;
- $\mathcal{M}$ is a set of mission identifiers that agents can commit to within the scheme;
- $maxmp : \mathcal{M} \rightarrow \mathbb{Z}$: is a function that maps each mission to the maximum number of commitments of that mission in the scheme (upper bound of mission cardinality);
- $minmp : \mathcal{M} \rightarrow \mathbb{Z}$: maps each mission to the minimum number of commitments of that mission necessary for the scheme to be considered well-formed (lower bound of mission cardinality);
- $\mathcal{G}$ is the set of goals within the scheme;
- $gm : \mathcal{G} \rightarrow \mathcal{M}$ maps each goal to its mission;
- $gpc : \mathcal{G} \rightarrow 2^{\mathcal{G}}$ maps goals to their precondition goals;[5]
- $ttf : \mathcal{G} \rightarrow \mathbb{Z}$ maps goals to their TTF; and
- $g_r \in \mathcal{G}$ is the root goal of the scheme.

For each oml-norm in the normative specification, the parser produces a tuple

$$\langle id, c, \rho, t, m, ttf \rangle$$

[4] To make clear the distinction between norms at the OML level with the ones at the NPL level, we will use the expression *oml-norm* when necessary.

[5] The precondition goals are deduced from the goal decomposition tree of the scheme (as presented in Fig. 2(a)). For example, the goal of "writing the paper conclusions" (*wcon*) can only be achieved after the goal of "writing sections" (*wsec*) has been achieved.

where id is a unique identification for the oml-norm; c is the activation condition for the oml-norm; ρ is the role; t is the type (obliged or permitted); m is the mission; and ttf is the deadline. We can read that oml-norm as 'when c holds, agents playing ρ are t to commit to mission m by ttf'.

4 From OML to NOPL

After the presentation of NPL (the generic target language of the translation) and OML (the source for the translation), this section defines NOPL, a particular class of NPL programs applied to the $\mathcal{M}$OISE OML. The NOPL syntax and semantics are the same as presented in Sec. 2. However, the set of facts, rules, and norms are specific to the $\mathcal{M}$OISE model and to the $\mathcal{M}$OISE artifact-based OMI presented in Sec. 5. Benefiting from the distributed nature of this OMI, our proposal consists of translating the OS defined in $\mathcal{M}$OISE OML into *different* NOPL programs. For each group type defined in the OML, a separate NOPL program is produced by the translation. The same criteria is used to translate schemes. Since the OMI has one artifact to manage each instance of a group or scheme, the corresponding translated NOPL programs are used in the deployment of the artifacts.

The path from NPL to NOPL consists firstly in the definition of facts and rules to express the different concepts and properties expressed in the OS. Dynamic facts are also introduced to represent the current state of the organisation. Below, we describe these rules and facts, step by step.

We use *translation rules* (briefly "t-rules") to formalise how the OS is translated into NOPL. Such rules have the following format:

$$\frac{condition}{\texttt{<code>}}\ (ID)$$

where *ID* is the name of the t-rule, $condition$ is a Boolean expression, and `<code>` is an excerpt of code in NOPL that is produced in case the condition holds. Details of the application of these rules are provided in the examples given later.

In this paper, we consider only the translation rules for producing scheme normative programs, i.e. NOPL programs used to manage the corresponding scheme artifacts in the OMI. The t-rule that generates the NOPL code for a scheme specification is:

$$\frac{\langle id, \mathcal{M}, maxmp, minmp, \mathcal{G}, gm, gpc, ttf, g_r \rangle \in \mathcal{FS}}{\texttt{np scheme(}id\texttt{) \{}\ \textit{SM}(S)\ \ \textit{SMR}(S)\ \ \textit{SG}(\mathcal{G})\ \ \textit{SR}(S)\ \ \textit{SSP}\ \ \textit{NS}\ \texttt{\}}}\ (S)$$

The condition for this t-rule is simply that the scheme specification belongs to the functional specification. The produced code (typeset in typewriter font) is a normative program with an identification id and facts, rules, and norms that are produced by specific t-rules (*SM*, *SMR*, *SG*, *SR*, *SSP*, and *NS*) defined below. Variables, typeset in italics (as in id), are replaced by their values obtained from the condition of the t-rule.

Facts. For scheme normative programs, the following facts are produced by the translation:

- `mission_cardinality(`m`,`min`,`max`)`: is a fact that defines the cardinality of a mission (e.g. `mission_cardinality(mCol,1,5)`).
- `mission_role(`m`,`ρ`)`: role ρ is permitted or obliged to commit to mission m (e.g. `mission_role(mMan,editor)`).
- `goal(`m`,`g`,`$pre\text{-}cond$`, '`ttf`')`: is a fact that defines the arguments for a goal g: its mission, identification, preconditions, and TTF (e.g. `goal(mMan,wsec, [wcon],'2 days')`).

The t-rules *SM*, *SMR*, and *SG* generate these facts from the specification (all sets and functions, such as $\mathcal{M}_S$ and $minmp_S$, used in the t-rule refer to the scheme S being translated):

$$\frac{m \in \mathcal{M}_S \qquad maxmp_S(m) > 0}{\texttt{mission\_cardinality(}m\texttt{,}minmp_S(m)\texttt{,}maxmp_S(m)\texttt{)}\ .}\ (\textit{SM(S)})$$

$$\frac{\langle id, c, \rho, t, m, ttf\rangle \in \mathcal{NS} \qquad maxmp_S(m) > 0}{\texttt{mission\_role(}m\texttt{,}\rho\texttt{)}\ .}\ (\textit{SMR(S)})$$

$$\frac{g \in \mathcal{G}}{\texttt{goal(}gm(g)\texttt{,}g\texttt{,}gpc(g)\texttt{,}ttf(g)\texttt{)}\ .}\ (\textit{SG(G)})$$

The following dynamic facts will be provided at runtime by the artifact (cf. Sec. 5) that manages 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.

Rules. Besides facts, we define some rules that are useful to infer the state of the scheme (e.g. whether it is well-formed) and goals (e.g. whether it is ready to be adopted or not). The rules produced by *SR* are general for all schemes and those produced by *SRW* are specific to the scheme being translated.

$$\frac{}{\texttt{is\_finished(S) :- satisfied(S,}g_r\texttt{)}\ .}\ (\textit{SR(S)})$$

```
mission_accomplished(S,M) :-
   .findall(Goal, goal(M,Goal,_,_), MissionGoals) &
   all_satisfied(S,MissionGoals).

all_satisfied(_,[]).
```

```
all_satisfied(S,[G|T]) :- satisfied(S,G) & all_satisfied(S,T).

// goal G of scheme S is ready to be adopted:
// all its preconditions have been achieved
ready(S,G) :-
   goal(_,G,PCG,_) & all_satisfied(S,PCG).

// number of players of a mission M in scheme S
mplayers(M,S,V) :- .count(committed(_,M,S),V).
   // .count(X) counts how many instances of X are known to the agent

well_formed(S) :- SRW(S).
```

$$\frac{m \in \mathcal{M} \qquad maxmp_S(m) > 0}{\texttt{mission\_accomplished(S,}m\texttt{)}} \; (\textit{SRW(S)})$$

```
|
mplayers(m,S,Vm) & Vm >= minmp_S(m) & Vm <= maxmp_S(m)
```

Note that these rules implement the semantics of *mission accomplishment*, *well-formed* and *ready goal* as intended in the $\mathcal{M}$OISE model.

As an example, the output of the translation produced by *SR* for the paper writing scheme is listed below.

```
is_finished(S) :- satisfied(S,wp). // wp is the root goal

mission_accomplished(S,M) :-
  .findall(Goal, goal(M,Goal,_,_), MissionGoals) &
  all_satisfied(S,MissionGoals).

all_satisfied(_,[]).
all_satisfied(S,[G|T]) :- satisfied(S,G) & all_satisfied(S,T).

ready(S,G) :- goal(_, G, PCG, _) & all_satisfied(S,PCG).

mplayers(M,S,V) :- .count(committed(_,M,S),V).
well_formed(S) :-
  (mission_accomplished(S,mMan) |
   mplayers(mMan,S,VmMan) & VmMan >= 1 & VmMan <= 1)
  &
  (mission_accomplished(S,mCol) |
   mplayers(mCol,S,VmCol) & VmCol >= 1 & VmCol <= 5)
  &
  (mission_accomplished(S,mBib) |
   mplayers(mBib,S,VmBib) & VmBib >= 1 & VmBib <= 1).
```

Norms. We have three classes of norms in NOPL for schemes: norms for goals, norms for properties, and domain norms (which are explicitly stated in the normative speci-

fication as oml-norms). For the former class, we define the following generic norm to express the $\mathcal{M}$OISE semantics for commitment:

```
norm ngoal:  committed(A,M,S) & goal(M,G,_,D) &
             well_formed(S) & ready(S,G)
-> obligation(A,ngoal,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 its deadline `D`". 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 introduced 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 situations where there are more agents committed to a mission than permitted in the scheme specification. As presented in Sec. 2, 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, if there is a norm such as `n6` in Table 1, the consequence will be an obligation. The two t-rules below detail the produced norms for the regimentation and obligation cases of mission cardinality.

$$\frac{\neg\,\exists\,\langle id, c, \rho, t, m, ttf\rangle \in \mathcal{NS} \;.\; c = \#mc}{} \quad (SSP_1)$$

```
norm mc:
      mission_cardinality(M,_,MMax) &
      mplayers(M,S,MP) & MP > MMax
   -> fail(mission_cardinality).
```

$$\frac{\langle id, c, \rho, t, m, ttf\rangle \in \mathcal{NS} \qquad c = \#mc}{} \quad (SSP_2)$$

```
norm mc:
      mission_cardinality(M,_,MMax) &
      mplayers(M,S,MP) & MP > MMax &
      responsible(Gr,S) & plays(A,ρ,Gr)
-> obligation(A,mc,committed(A,m,_),'now'+'ttf').
```

In our running example, the norm produced by SSP_1 to regiment mission cardinality is:

```
norm mc:
   mission_cardinality(M,_,MMax) &
   mplayers(M,S,MP) &  MP > MMax
-> fail(mission_cardinality).
```

and the norm produced if SSP_2 were used instead (for sanction rather than regimentation) would be:

```
norm mc:
   mission_cardinality(M,_,MMax) &
   mplayers(M,S,MP) &  MP > MMax &
   responsible(Gr,S) & plays(A,editor,Gr)
-> obligation(A,mc,committed(A,ms,_), 'now'+'1 hour').
```

where the agent playing editor is obliged to commit to the mission ms within one hour (corresponding to the oml-norm `n6` in Table 1).

For the third class of norms, each oml-norm of type obligation in the normative specification of the OS has a corresponding norm in the NOPL program. Whereas an OML obligation refers to a role and a mission, NPL requires that obligations are for agents and towards a goal. The NOPL norm thus identifies each agent playing the role in groups responsible for the scheme and, if the number of current players still does not reach the maximum cardinality, and the mission was not accomplished yet, the agent is obliged to achieve a state where it is committed to the mission. The following t-rule expresses just that:

$$\frac{\langle id, c, \rho, t, m, ttf\rangle \in \mathcal{NS} \qquad m \in \mathcal{M} \qquad t = obl}{\texttt{norm } id\texttt{:}\ c\ \texttt{\&}\ \texttt{plays(A,}\rho\texttt{,Gr) \& responsible(Gr,S) \& mplayers(}m\texttt{,S,V) \& V <}\ maxmp(m)\ \texttt{\& not mission\_accomplished(S,}m\texttt{)}\ \texttt{->}\ \texttt{obligation(A,}id\texttt{,committed(A,}m\texttt{,S),'now'+'}ttf\texttt{')}.} \ (NS)$$

For instance, the NOPL norm resulting from the translation of oml-norm `n2` in Table 1 with the t-rule above is:

```
norm n2:  plays(A,writer,Gr) & responsible(Gr,S) &
          mplayers(mCol,S,V) & V < 5 &
          not mission_accomplished(S,mCol)
-> obligation(A,n2,committed(A,mCol,S),'now'+'1 day').
```

Note that if some mission is already accomplished (as defined by the t-rule *SR*), there is no obligation to commit to it (this interpretation of "obligation to commit" was originally proposed by [18]). An agent can thus commit to a mission, fulfil its goals, and leave the scheme before it is finished. Without this last condition, the agent has to participate in the scheme until it is removed from the multi-agent system. Note also that if the scheme already has 5 engaged agents, there is no obligation for other players of role *writer* to commit to $mCol$. In fact, if a sixth agent wanted to commit to $mCol$, norm `mc` would produce a failure.

Besides the obligations defined in the OML, we also have permissions and (by default) prohibitions. Since everything is permitted by default in NPL, OML permissions

do not need to be translated. The OML prohibitions are handled in NOPL by a generic norm that fails when an agent is committed to a mission not permitted by its roles (according to the `mission_role` relation):

```
norm mission_permission:
   committed(Agt,M,S) &
   not (plays(Agt,R,_) & mission_role(M,R))
-> fail(mission_permission(Agt,M,S)).
```

The norm above uses regimentation to prohibit an agent to commit to a mission if it is not allowed to do so. Obligations could be used instead of regimentation, as illustrated by the following norm:

```
norm mission_permission:
   committed(Agt,M,S) &
   not (plays(Agt,R,_) & mission_role(M,R)) &
   plays(E,editor,_) // agent playing editor is obliged to
                     // to commit to a sanction mission
-> obligation(E,mp,committed(E,ms,_), 'now'+'1 hour').
```

The type of the consequence for the mission permission norm, whether a `fail` or an `obligation`, is defined by parameters passed on to the translator program.

Also regarding prohibitions, one could ask: Is an agent prohibited to leave its missions without fulfilling the mission's goals? There are two answers depending whether the following norm is included or not.

```
norm mission_leaved:
   leaved_mission(Agt,M,S) &
   not mission_accomplished(S,M)
-> fail(mission_leaved(Agt,M,S)).
```

If the above norm is included, the leave-mission action (which adds the fact `leaved_mission`) will fail. The action is regimented in this case. Otherwise, no error will be produced by the action. However, the norms generated from t-rule *NS* will be activated again and the agent becomes again obliged to commit to the mission.

5 Artifact-Based Architecture

The ideas presented in this paper have been implemented as part of an OMI that follows the Agent & Artifact model [13, 10][6]. In this approach, a set of organisational artifacts is available in the MAS environment providing operations and observable properties for the agents so that they can interact with the OMI. For example, each scheme instance is managed by a "scheme artifact". A scheme artifact, shown in Fig. 3, provides operations such as "commit to mission" and "goal x has been achieved" (whereby agents can act upon the scheme) and observable properties (whereby agents can perceive the current

[6] An implementation of the translator and the OMI is available at `http://moise.sourceforge.net`.

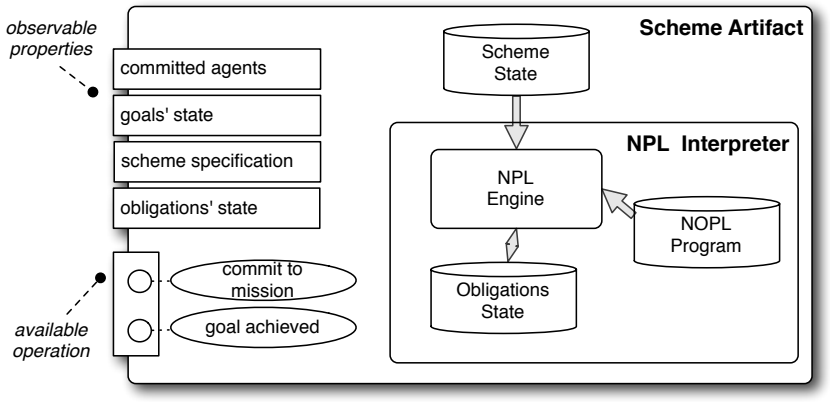


Fig. 3. General View of the Scheme Artifact

state of the scheme). We can effortlessly distribute the OMI by deploying as many artifacts as necessary for the application.

Following the ideas introduced in this paper, each organisational artifact has within it an NPL interpreter that is given as input: (i) the NOPL program automatically generated from the OS for the type of the artifact (e.g. the artifact that will manage the writing paper scheme will receive as input the NOPL program translated from that scheme specification), and (ii) dynamic facts representing the current state of (part of) the organisation (e.g. the scheme artifact itself will produce dynamic facts related to the current state of the scheme instance). The interpreter is then used to compute: (i) whether some operation will bring the organisation into an inconsistent state (where inconsistency is defined by means of the specified regimentations), and (ii) the current state of the obligations.

Algorithm 1, implemented on top of CArtAgO [16], shows the general pattern we used to implement every operation (e.g. commitment to mission) in the organisational artifacts. Whenever an operation is triggered by an agent, the algorithm first stores a "backup" copy of the current state of the artifact (line 5). This backup is restored (line 10) if the operation leads to failure (e.g. committing to a mission not permitted). The overall functioning is that invalid operations do not change the artifact state.[7] A valid operation is thus an operation that changes the state of the artifact to one where no `fail` (i.e. regimentation) is produced by the NPL interpreter. In case the operation is valid, the algorithm simply updates the current state of the obligations (line 13). Although the NPL handles *states* in the norm conditions, this pattern of integration has allowed us to use NPL to manage agent *actions*, i.e. the regimentation of operations on artifacts.

Notice that the NOPL program is not seen by the agents. They continue to perceive and reason on the scheme specification as written in the OML. The NOPL is used only within the artifact to simplify its development.

[7] This functioning requires that operations are not executed concurrently, which can be easily configured in CArtAgO.

Algorithm 1 Artifact Integration with NOPL

1: oe is the state of the organisation managed by the artifact
2: p is the current NOPL program
3: npi is the NPL interpreter
4: **when** an operation o is triggered by agent a **do**
5: $oe' \leftarrow oe$ // creates a "backup" of the current oe
6: executes operation o to change oe
7: $f \leftarrow$ a list of predicates representing oe
8: $r \leftarrow npi(p, f)$ // runs the interpreter for the new state
9: **if** r = `fail` **then**
10: $oe \leftarrow oe'$ // restore the backup state
11: **return** fail operation o
12: **else**
13: update obligations in the observable properties
14: **return** succeed operation o

6 Related Work

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 proposal, the NPL, should be considered specially for two properties of the language: its simplicity and its formal basis (that led to an available implementation). Similar work has been done by Tinnemeier et al. [17], where the operational semantics for a normative language was also proposed. Their approach and ours are similar on certain points. For instance, both consider norms as "declarative" norms (i.e. "ought-to-be" norms) in the sense that obligations and regimentation bear on goals. However, our work differs in several aspects. The NOPL class of NPL programs is for the OMI and not for programmers to use. The designers/programmers continue to use OML to define both an organisation and the norms that have to be managed within such a structure. Organisation primitives of the OML are higher-level and tailored for organisation modelling, therefore an OML specification is significantly more *concise* than its translation into a normative language.

Another clear distinction is that we rely on a dedicated programming model (the Agent & Artifact model) providing a clear connection of the organisation to the environment and allowing us to implement regimentation on physical actions [14]. The artifacts model also simplified the distribution of the management of the state of the organisation with several instances and kinds of artifacts, avoiding over-centralisation in the management of organisational and normative aspects of multi-agent systems.

Going back to the issue of conciseness and expressiveness, we do not claim that OML is more expressive than NPL. In fact, since the OML can be translated into NPL and some NPL programs cannot be translated into OML (for instance the program in the end of Sec. 2), NPL is strictly more expressive in theoretical terms. NOPL, on the other hand, has the same expressiveness of OML, since we can translate NOPL back into OML (this translation is not the focus of this paper but is feasible). However, we are not looking for general purpose or more expressive programming languages, but

languages that help automating part of the OMI development. In this regard, the OML was designed to be more concise and more abstract than NOPL. The OML allows the designer to specify complex properties of the organisation with natural abstractions and fewer lines of code, which are translated into several lines of NOPL code that is interpreted by the NPL engine.

Regarding the second contribution, namely the automatic translation, we were inspired by work on ISLANDER [2, 7]. The main difference here is the initial and target languages. While they translate a normative specification into a rule-based language, we start from a high-level organisation modelling language and the target is a simple normative programming language. NOPL is more specific than rule-based languages, being specifically tailored from our NPL for the $\mathcal{M}$OISE OML.

Regarding the third contribution, the OMI, we started from ORA4MAS [10]. 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, which is a well-known organisational model that has not yet been fully formalised.

$\mathcal{M}$OISE shares some concepts with a variety of organisational models available in the multi-agent systems literature, so we expect to be able to use our approach to give concrete semantics and efficient implementations for a variety of other modelling languages too.

7 Conclusions

In this paper, we introduced a translation from 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, we can bring flexibility to the development of OMIs. Our work also emphasises the point that a normative programming 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 to sanction in case of violation of the prohibitions. The resulting NPL is thus simpler to formalise and implement.

Another result of this work is to show that an organisational language (OML) can be translated (and reduced to) a normative programming language. Roughly, all management within an OMI can be based on the management of norms and obligations. This result emphasises the importance of norms as a fundamental concept for the development of OMIs. Future work will explore that capacity of NPL for other organisational and institutional languages.

We also plan to investigate further possible relationships among norms, for instance when the activation of a norm triggers another. This form of chain triggering of norms is already possible in the language. The current state of an obligation is one of the dynamic facts updated by the artifact and accessible to the NPL interpreter, and it can be used in the condition of norms. For example, we can write: `norm x: active(obligation(....)) -> fail(....).`. However, this feature of the language requires further experimentation.

It also remains future work to evaluate how the reorganisation process available in $\mathcal{M}$OISE will impact on the normative-based artifacts. Changes in the organisation specification imply changes in the corresponding NOPL programs. We can simply change these programs in the artifacts, but there are problems that require investigation. For example, the problem of what to do with the active obligations, created from an organisation that changed, and which might need to be dropped in some cases and prevented from being dropped just because of a change in the NOPL program in other cases. The revision of such obligations is one of the main issue we will consider in this area of future work.

8 Acknowledgements

The authors are grateful for the supported given by CNPq, grants 307924/2009-2, 307350/2009-6, and 478780/2009-5.

References

1. Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldrige. *Programming Multi-Agent Systems in AgentSpeak using Jason*. Wiley Series in Agent Technology. John Wiley & Sons, 2007.
2. Viviane Torres 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*, 17(1):113–155, Aug 2008.
3. Virginia Dignum, editor. *Handbook of Research on Multi-agent Systems: Semantics and Dynamics of Organizational Models*. Information Science Reference, 2009.
4. Marc Esteva, David de la Cruz, and Carles Sierra. ISLANDER: an electronic institutions editor. In Cristiano Castelfranchi and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2002)*, LNAI 1191, pages 1045–1052. Springer, 2002.
5. Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agents systems. In Yves Demazeau, editor, *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS'98)*, pages 128–135. IEEE Press, 1998.
6. Nicoletta Fornara and Marco Colombetti. Specifying and enforcing norms in artificial institutions. In A. Omicini, B. Dunin-Keplicz, and J. Padget, editors, *Proceedings of the 4th European Workshop on Multi-Agent Systems (EUMAS 06)*, 2006.
7. Andrés García-Camino, Juan A. Rodríguez-Aguilar, Carles Sierra, and Wamberto Vasconcelos. Constraining rule-based programming norms for electronic institutions. *Journal of Autonomous Agents and Multi-Agent Systems*, 18(1):186–217, Feb 2009.
8. Davide Grossi, Huib Aldewered, and Frank Dignum. *Ubi Lex, Ibi Poena*: Designing norm enforcement in e-institutions. In P. Noriega, J. Vázquez-Salceda, G. Boella, O. Boissier, V. Dignum, N. Fornara, and E. Matson, editors, *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, volume 4386 of *LNAI*, pages 101–114. Springer, 2007. Revised Selected Papers.
9. Jomi Fred Hübner, Olivier Boissier, and Rafael H. Bordini. A normative organisation programming language for organisation management infrastructures. In Julian Padget et al., editor, *Coordination, Organizations, Institutions and Norms in Agent Systems V*, volume 6069 of *LNAI*, pages 114–129. Springer, 2010.

10. Jomi Fred Hübner, Olivier Boissier, Rosine Kitio, and Alessandro 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*, 20(3):369–400, May 2010.
11. Jomi Fred Hübner, Jaime Simão Sichman, and Olivier 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*, 1(3/4):370–395, 2007.
12. Andrew J. I. Jones and Marek Sergot. On the characterization of law and computer systems: the normative systems perspective. In *Deontic logic in computer science: normative system specification*, pages 275–307. John Wiley and Sons Ltd., Chichester, UK, 1993.
13. Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Journal of Autonomous Agents and Multi-Agent Systems*, 17(3):432–456, 2008.
14. Michele Piunti, Alessandro Ricci, Olivier Boissier, and Jomi F. Hübner. Embodying organisations in multi-agent work environments. In *Proceedings of International Joint Conferences on Web Intelligence and Intelligent Agent Technologies (WI-IAT 2009)*, pages 511–518. IEEE/WIC/ACM, 2009.
15. David V. Pynadath and Milind Tambe. An automated teamwork infrastructure for heterogeneous software agents and humans. *Autonomous Agents and Multi-Agent Systems*, 7(1-2):71–100, 2003.
16. Alessandro Ricci, Michele Piunti, Mirko Viroli, and Andrea Omicini. Environment programming in CArtAgO. In Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Tools and Applications*, chapter 8, pages 259–288. Springer, 2009.
17. Nick Tinnemeier, Mehdi Dastani, and John-Jules Meyer. Roles and norms for programming agent organizations. In Jaime Sichman, Keith Decker, Carles Sierra, and Cristiano Castelfranchi, editors, *Proc. of AAMAS 09*, pages 121–128, 2009.
18. Birna van Riemsdijk, Koen Hindriks, Catholijn M. Jonker, and Maarten Sierhuis. Formal organizational constraints: A semantic approach. In Hoek, Kaminka, Lesperance, Luck, and Sen, editors, *Proc. of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 823–830, 2010.
19. J. Vázquez-Salceda, H. Aldewereld, and F. Dignum. Norms in multiagent systems: some implementation guidelines. In *Proceedings of the Second European Workshop on Multi-Agent Systems (EUMAS 2004)*, 2004. `http://people.cs.uu.nl/dignum/papers/eumas04.PDF`.
20. Fabiola López y López, Michael Luck, and Mark d'Inverno. Constraining autonomy through norms. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 674 – 681. ACM Press, 2002.