$\mathcal{M}$OISE specifications
— draft —

Jomi F. Hübner

November 12, 2013

Abstract

This document presents $\mathcal{M}$OISE specifications for (*i*) the organisational *model* (OS, OE), and (*ii*) the *semantics* in terms of translation to NOPL. The focus is on the formalisation, no motivations, examples, or detailed explanations are thus provided (these aspects are considered in the papers listed in the end of this document).

The current implementation of $\mathcal{M}$OISE (release 0.7) is considered in this document and not the variants/experiments/extensions published in some papers.

Contents

1 Organisation Model

The $\mathcal{M}$OISE organisation model started on paper [Hannoun, 2002] and then was extended in [Hübner et al., 2002] and [Gâteau et al., 2005]. This section is thus based on these papers. Although, as stated in the abstract, this document considers the implemented version of the model, as available in `http://moise.sourceforge.net`.

Sets, relations, and functions are used to define the elements that compose the organisational model of $\mathcal{M}$OISE. The informal meaning of these elements are presented in the papers cited above. The formal meaning is defined in Sec. 2.

TODO: use Z notation

1.1 Organisational Specification

Definition 1 (Organisational Specification)**.** An Organisational Specification (OS) is defined by its three dimensions: structural, functional, and normative.

$$\langle id, SS, FS, \mathcal{NS} \rangle$$

where

- id is a unique identification of the OS;
- $SS : \mathcal{SS}$ is a structural specification ($\mathcal{SS}$ is the set of all SSs and the *type* of SS);
- $FS : \mathcal{FS}$ is a functional specification ($\mathcal{FS}$ is the set of all FSs); and
- $\mathcal{NS}$ is a set of norms.

1.1.1 Structural Specification

Definition 2 (Structural Specification)**.** A Structural Specification (SS) is defined by the elements of the following tuple:

$$\langle \mathcal{R}, \sqsubset, rg, \mathcal{L} \rangle$$

where

- $\mathcal{R}$ is a set of identifiers of roles of the organisation;
- $\sqsubset : \mathcal{R} \times \mathcal{R}$ is an inheritance relation among roles;[1]

[1] $\rho \sqsubset q$ means q is a sub-role of ρ or ρ is a super-role of q.

- $rg : \mathcal{GS}$ is the specification of the root group of the organisation ($\mathcal{GS}$ is the set of all group specifications and the type of rg);
- $\mathcal{L}$ is a set of links between roles in the scope of the group being defined.

Inheritance properties:

- Anti-symmetric
$$\rho \sqsubset \rho' \wedge \rho' \sqsubset \rho \Rightarrow \rho = \rho' \tag{1}$$
- Transitivity
$$\rho \sqsubset \rho' \wedge \rho' \sqsubset \rho'' \Rightarrow \rho \sqsubset \rho'' \tag{2}$$
- Role hierarchy root (ρ_{soc})
$$\rho_{soc} \in \mathcal{R} \tag{3}$$
$$\forall \rho \in (\mathcal{R} \backslash \{\rho_{soc}\}) \mid \rho_{soc} \sqsubset \rho \tag{4}$$
$$\nexists \rho \in \mathcal{R} \mid \rho \sqsubset \rho_{soc} \tag{5}$$

Definition 3 (Link)**.** A link is defined by a tuple:

$$\langle s, t, k, p \rangle$$

where

- $s : \mathcal{R}$ is the role source of the link;
- $t : \mathcal{R}$ is the role target of the link;
- $k : \{acq, com, auth\}$ is the type the link (acquaintance, communication, or authority).
- $p : \{intra, inter\}$ is the scope of the link (inter-group or intra-group).

Reads: an agent playing the role s has the link of type k to agents playing role t in scope p.

Properties:

- Authority implies communication
$$\langle s, t, auth, p \rangle \Rightarrow \langle s, t, com, p \rangle \tag{6}$$
- Communication implies acquaintance
$$\langle s, t, com, p \rangle \Rightarrow \langle s, t, acq, p \rangle \tag{7}$$

- Inheritance (all links defined for a role is inherited by its sub-roles)

$$\langle s,t,k,p\rangle \in \mathcal{L} \wedge s \sqsubset s' \quad \Rightarrow \quad \langle s',t,k,p\rangle \in \mathcal{L} \tag{8}$$
$$\langle s,t,k,p\rangle \in \mathcal{L} \wedge t \sqsubset t' \quad \Rightarrow \quad \langle s,t',k,p\rangle \in \mathcal{L} \tag{9}$$

The scope *intra* means that the link is valid only inside a group instance: an agent playing the role s in an instance group g has the link to agents playing the role t in the same group g. The scope *inter* means that the link exists only for different group instances: an agent playing the role s in an instance group g has the link to agents playing the role t in the another group g', where $g \neq g'$. In the case where both scopes are defined, the link exists despite the instances of the groups.

Definition 4 (Group Specification)**.** A Group Specification (GS) is defined by the elements of the following tuple:

$$\langle id, compat, maxrp, minrp, maxsg, minsg\rangle$$

where

- id is a unique identification of the GS;
- $compat : \mathcal{R} \rightarrow 2^{\mathcal{R}}$ is a function that maps each role to the set of its compatible roles;
- $maxrp : \mathcal{R} \rightarrow \mathbb{Z}$: is a function that maps each role to the maximum number of players of that role in the group (upper bound of role cardinality);[2]
- $minrp : \mathcal{R} \rightarrow \mathbb{Z}$: is a function that maps each role to the minimum number of players of that role necessary for the group to be considered well-formed (lower bound of role cardinality);
- $maxsg : \mathcal{GS} \rightarrow \mathbb{Z}$: is a function that defines the maximum number of subgroups of the group (upper bound of subgroup cardinality);[3]
- $minsg : \mathcal{GS} \rightarrow \mathbb{Z}$: is a function that defines the minimum number of subgroups of the group (lower bound of subgroup cardinality).

Compatibility properties:

- Reflexivity

$$\rho \in compat(\rho) \tag{10}$$

- Transitivity

$$\rho \in compat(\rho') \wedge \rho' \in compat(\rho'') \Rightarrow \rho \in compat(\rho'') \tag{11}$$

[2] If role ρ is not allowed in the group, we have $maxrp(\rho) = 0$.
[3] If group gs is not allowed as a subgroup, we have $maxsg(gs) = 0$.

- Inheritance (all compatibilities defined for a role is inherited by its sub-roles)

$$\rho_a \in compat(\rho_b) \wedge \rho_a \neq \rho_b \wedge \rho_a \sqsubset \rho' \quad \Rightarrow \quad \rho' \in compat(\rho_b) \qquad (12)$$
$$\rho_a \in compat(\rho_b) \wedge \rho_a \neq \rho_b \wedge \rho_b \sqsubset \rho' \quad \Rightarrow \quad \rho_b \in compat(\rho') \qquad (13)$$

We denote the sets and functions of a group specification by $maxrp_{GS}$, $compat_{GS}$, etc.

The function $compat_{GS}$ contains the compatibilities defined in the scope of instances of groups created based on the specification GS: an agent already playing roles ρ_i in a group instance g is allowed to adopt in g *only* roles in $compat_{GS}(\rho_i)$.

TODO: add scope inter-group for *compat*: put it in SS instead of GS

1.1.2 Functional Specification

Definition 5 (Functional Specification)**.** A Functional Specification (FS) is defined by the elements of the following tuple:

$$\langle \mathcal{M}, \mathcal{G}, \mathcal{S} \rangle$$

where

- $\mathcal{M}$ is a set of identifiers of missions of the organisation;
- $\mathcal{G}$ is a set of identifiers of goals of the organisation;
- $\mathcal{S}$ is the set of scheme specifications of the organisation.

Definition 6 (Scheme Specification)**.** A Scheme Specification (S) is defined by the elements of the following tuple:

$$\langle id, maxmp, minmp, g_r \rangle$$

where

- id is a unique identification of the scheme;
- $maxmp : \mathcal{M} \to \mathbb{Z}$: is a function that maps each mission to the maximum number of commitments of that mission in the scheme (upper bound of mission cardinality). If $maxmp(\cdot) = 0$, the mission is not permitted in the scheme;
- $minmp : \mathcal{M} \to \mathbb{Z}$: is a function that 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);

- $g_r : \mathcal{G}$ is the root-goal of the scheme.

TODO: add preference among missions

Definition 7 (Goal)**.** A goal is defined by the elements of the following tuple:

$$\langle id, gm, type, card, ttf, p \rangle$$

where

- id is a unique identification of the goal;
- $gm : 2^{\mathcal{M}}$ is the set of missions that include the goal;
- $type : \{ach, maint\}$ is the type of the goal (either achievement or maintenance);
- $card : \mathbb{Z}$ is the cardinality of the goal – how many agents have to achieve the goal for the goal to be considered as globally satisfied;
- $ttf : \mathbb{Z}$ is the Time To Fulfil the goal; and
- $p : \mathcal{P}$ is a plan to achieve the goal, it defines the sub-goals of this goal ($\mathcal{P}$ is the set all plans).

Definition 8 (Plan)**.** A plan is defined by the tuple

$$\langle g_1, g_2, ..., g_n, o \rangle$$

where

- $g_i : \mathcal{G}$ $(1 \leq i \leq n)$ are the sub-goals;
- $o : \{sequence, choice, parallel\}$ is the operator among the sub-goals (whether one or all sub-goal have to be achieved and whether in sequence or parallel).

TODO: add *gpc* (goal pre-conditions)

1.1.3 Normative Specification

Definition 9 (Norm)**.** A norm is composed by the following elements:

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

where

- id is the id of the norm;

- c is the activation condition of the norm;
- ρ is the role;
- d is the type (obliged or permitted);
- m is the mission; and
- ttt is the deadline.

We can read 'when c holds, the agents playing ρ are d to commit to the mission m before ttf'.

Properties:

- Inheritance (all obligations and permissions are inherited)

$$\langle id, c, \rho, d, m, ttf \rangle \in \mathcal{NS} \wedge \rho' \sqsubset \rho \quad \Rightarrow \quad \langle id, c, \rho', d, m, ttf \rangle \in \mathcal{NS} \quad (14)$$

1.2 Organisation Entity

Definition 10 (OE)**.** An Organisation Entity (OE) is defined by the elements of the following tuple:

$$\langle OS, \mathcal{A}, \mathcal{GI}, \mathcal{SI} \rangle$$

where

- OS is a organisation specification of the OE;
- $\mathcal{A}$ is a set of agent's identifiers;
- $\mathcal{GI}$ is a set of group instances GI created in the OE; and
- $\mathcal{SI}$ is a set of scheme instances SI created in the OE.

Definition 11 (GI)**.** A Group Instance (GI) is defined by the elements of the following tuple:

$$\langle id, GS, players, subgroups, \mathcal{RS} \rangle$$

where

- id is a unique identifier of the group;
- $GS : \mathcal{GS}$ is the specification of the group;
- $players : \mathcal{R} \rightarrow 2^{\mathcal{A}}$ is function that maps each available role in the corresponding GS to the set of agents that are playing that role;
- $subgroups : \mathcal{GS} \rightarrow 2^{\mathcal{GI}}$ is a function that maps each groups specification to a set of group instances;

- $\mathcal{RS} : 2^{SI}$ is a set of schemes' identification the group is responsible for.

A group instance g is well formed if the role and subgroup cardinality are respected and all subgroups are also well formed:

$$\begin{array}{lcll} well\_formed(g) & = & \forall_{\rho \in \mathcal{R}} & |players(\rho)| \leq maxrp_{GS}(\rho) \wedge \\ & & & |players(\rho)| \geq minrp_{GS}(\rho) \wedge \\ & & \forall_{g \in \mathcal{GS}} & |subgroups(g)| \leq maxsg_{SG}(g) \wedge \\ & & & |subgroups(g)| \geq minsg_{SG}(g) \wedge \\ & & \forall_{g' \in subgroups(g)} & well\_formed(g') \end{array}$$

Definition 12 (SI)**.** A Scheme Instance (SI) is defined by the elements of the following tuple:

$$\langle id, S, commitments, achievements \rangle$$

where

- id is a unique identifier of the scheme instance;
- $S : \mathcal{S}$ is the specification of the scheme;
- $commitments : \mathcal{M} \rightarrow 2^{\mathcal{A}}$ is a function that maps each mission in the corresponding scheme specification to the set of agents that are committed to that mission; and
- $achievements : \mathcal{A} \rightarrow 2^{\mathcal{G}}$ is a function that maps each agent to the set of goals it has achieved.

TODO: add goal state (satisfied as defined in the goal cardinality).

2 OML Semantics

This section is based on the paper [Hübner et al., 2009] that proposes the use of normative programming language (NOPL) as the basis for both the semantics and implementation of $\mathcal{M}$OISE. Again, the detailed motivations, examples, and justifications are in the papers. However the papers, due to the lack of space and their objectives, do not include all the semantics, which are then include here.[4]

The basic idea of the semantic is to define how the organisational actions change the organisation, their consequences, and constraints. These aspects are written in a normative program that is automatically created from OS/OE. Briefly:

1. The organisation (OS + OE) is translated to a NOPL program.

[4] An alternative semantics for $\mathcal{M}$OISE is presented in [van Riemsdijk et al., 2010].

2. This program is interpreted by the organisation platform where the agents interact with the organisation. That interaction is then managed and regulated by the NOPL program.

3. Since the normative language has formal operational semantics, and the translation from OS/OE to NOPL is automatic and also formal, the semantics of an OS/OE is formally defined by the semantics of the NOPL program.[5]

The result of translation process is exemplified in Appendix A, which contains the result of the translation for a particular OS. More details are also documented in the program that does the translation, it is available in `src/ora4mas/nopl/tools/os2nopl.java`.

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

condition
——————————————— *ID*
`<code>`

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.

2.1 NOPL for OS

The t-rule, identified by OT, that generates the NOPL code for an organisation specification OS is:

——————————————— *OT(OS)*
`scope organisation(`id_{OS}`) {`
  RIH(*OS*)
  FPL
  GT
  ST
}

There is no condition for this t-rule. The produced code is defined by other t-rules. The former defines the role hierarchy and the second includes in the

[5] Not all elements of OS/OE are translated to NOPL, so the semantics presented here is partial.

generated code a rule that verifies whether an agent is playing a role or not based on the role hierarchy.

$\rho_1 \sqsubset \rho_2$ — *RIH(OS)*

`subrole(`ρ_1`,`ρ_2`).`

— *FPL*

```
fplay(A,R,G) :- play(A,R,G).
fplay(A,R,G) :- subrole(R1,R) & fplay(A,R1,G).
```

The rules GT and ST produce code for the groups and schemes and are defined in the sequel.

(you can see an example of the result of this translation in appendix A.)

2.2 NOPL for Groups

The t-rule, identified by GT, that generates the NOPL code for a group instance GI of type GS is:

— *GT(GI,GS)*

`scope group(`id_{GS}`) {`
`  group_id(`id_{GI}`).`
P`(`*GI*`)` *RG*`(`*GI*`)`
RCR`(`*GS*`)` *RCP*`(`*GS*`)` *GSR*`(`*GS*`)`
GSP`(role_in_group)`
GSP`(role_cardinality)`
GSP`(role_compatibility)`
GSP`(well_formed_responsible)`
`}`

There is no condition for this t-rule. The produced code (typeset in typewriter font) is a normative program with an identification id_{GS} and facts, rules, and norms that are produced by specific t-rules (P, RG, ...) defined in the sequel. Variables, typeset in italics (as in id_{GS}), are replaced by their values obtained from the condition of the t-rule. (recall that id_{GS} denotes the element id of the tuple GS.)

2.2.1 Facts

For group normative programs, the following facts are produced by the translation:

- `play(`a`,`ρ`,`gr`)`: agent a plays the role ρ in the group instance identified by gr.

$$\frac{\rho \in \mathcal{R} \qquad a \in players_{GI}(\rho)}{\texttt{play(}a\texttt{,}\rho\texttt{,}id_{GI}\texttt{)}.}\ P(GI)$$

- `responsible(`g`,`s`)`: the group instance g is responsible for the missions of scheme instance s.

$$\frac{s \in \mathcal{S}_{GI}}{\texttt{responsible(}id_{GI}\texttt{,}s\texttt{)}.}\ RG(GI)$$

- `role_cardinality(`ρ`,`max`,`min`)`: the cardinality of some role in the group.

$$\frac{\rho \in \mathcal{R} \qquad maxrp_{GS}(\rho) > 0}{\texttt{role\_cardinality(}\rho\texttt{,}maxrp_{GS}(\rho)\texttt{,}minrp_{GS}(\rho)\texttt{)}.}\ RCR(GS)$$

- `compatible(`ρ_1`,`ρ_2`)`: role ρ_1 is compatible with ρ_2.

$$\frac{\rho_1 \in \mathcal{R} \qquad \rho_2 \in compat_{GS}(\rho_1)}{\texttt{compatible(}\rho_1\texttt{,}\rho_2\texttt{)}.}\ RCP(GS)$$

- `subgroup(`sg`,`gt`,`pg`)`: the group instance sg is a subgroup of group instance pg and the groups specification of sg is $gt \in \mathcal{GS}$.

$$\frac{g \in \mathcal{GS} \qquad sg \in subgroups_{GI}(g)}{\texttt{subgroup(}sg\texttt{,}g\texttt{,}id_{GI}\texttt{)}.}\ SG(GI)$$

- `subgroup_well_formed(`g`)`: the subgroup instance g is well formed.

2.2.2 Rules

In the group translation we have one rule that states whether the group is well formed.

$$\frac{}{\texttt{rplayers(R,G,V) :- .count(play(\_,R,G),V).}}\ GSR(GS)$$

`well_formed(G) :-` $GSWFR(GS)$ $GSWFSG(GS)$`.`

$$\frac{\rho \in \mathcal{R} \qquad maxrp_{GS}(\rho) > 0}{} \; GSWFR(GS)$$

```
rplayers(ρ,G,Vρ) &
Vρ >= minrp_GS(ρ) &
Vρ <= maxrp_GS(ρ)
```

$$\frac{sg \in \mathcal{R} \qquad maxsg_{GS}(sg) > 0}{} \; GSWFSG(GS)$$

```
.count(subgroup(_,sg,G),Ssg) &
Ssg >= minsg_GS(sg) &
Ssg <= maxsg_GS(sg) &
.findall(GInst, subgroup(GInst,_,G), ListSubgroups) &
all_subgroups_well_formed(ListSubgroups).

all_subgroups_well_formed([]).
all_subgroups_well_formed([H|T]) :-
   subgroup_well_formed(H) &
   all_subgroups_well_formed(T).
```

2.2.3 Norms

Norms in group normative programs are used to manage *properties* (role cardinality, compatibility, etc.). The non compliance with the properties can be either regimented (leading to a fail) of the creation of an obligation to someone to check the case. Regimented properties are those without an entry the NS.

$$\frac{\nexists\, \langle id, c, \rho, d, m, ttf \rangle \in \mathcal{NS} \mid c = \#p}{} \; GSP(p)$$

```
norm p:
       pdc(p)
   -> fail(p).
```

where *pdc* is a function that maps the ids of properties to its condition in NOPL as defined in Table 1.

Non regimented properties have an entry in the NS stating what to do. They are translated by the following t-rule:

$$\frac{\langle id, c, \rho, d, m, ttf \rangle \in \mathcal{NS} \qquad c = \#p}{} \; GSP(p)$$

```
norm id:
       pdc(p) &
       group_id(Gr) & monitor_scheme(MonSch) &
       fplay(A,ρ,Gr)
-> obligation(A,p,committed(A,m,_),‘now‘+‘ttf‘).
```

id	condition
group	
role_in_group	`play(Agt,R,Gr) & not role_cardinality(R,_,_)`
role_cardinality	`group_id(Gr) & role_cardinality(R,_,RMax) & rplayers(R,Gr,RP) & RP > RMax`
role_compatibility	`play(Agt,R1,Gr) & play(Agt,R2,Gr) & R1 < R2 & not compatible(R1,R2)`
well_formed_responsible	`responsible(Gr,S) & not well_formed(Gr)`
scheme	
mission_permission	`committed(Agt,M,S) & not (mission_role(M,R) & responsible(Gr,S) & fplay(Agt,R,Gr))`
mission_leaved	`leaved_mission(Agt,M,S) & not mission_accomplished(S,M)`
mission_cardinality	`scheme_id(S) & mission_cardinality(M,_,Max) mplayers(M,S,MP) & MP > Max`
ach_not_enabled_goal	`achieved(S,G,Agt) & goal(M,G,_,_,_,_) & not mission_accomplished(S,M) & not enabled(S,G)`
ach_not_committed_goal	`achieved(S,G,Agt) & goal(M,G,_,_,_,_) & not mission_accomplished(S,M) & not committed(Agt,M,S)`
goal_non_compliance	`obligation(Agt,ngoa(S,M,G),Obj,TTF) & not Obj & 'now' > TTF`

Table 1: Pre-defined conditions for norms (*pdc* function)

2.3 NOPL for Schemes

The t-rule that generates the NOPL code for a scheme instance SI specified by S is:

---------- *ST(SI,S)*

`scope scheme(`id_S`) {`
`scheme_id(`id_{SI}`).`
SM(*S*) *SMR*(*S*) *SG*(*S*)
SR(*S*)
SSP(`mission_permission`)
SSP(`mission_leaved`)
SSP(`mission_cardinality`)
SSP(`ach_not_enabled_goal`)
SSP(`ach_not_committed_goal`)
SSP(`goal_non_compliance`)
NS
}

2.3.1 Facts

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

- `committed(`a`,`m`,`s`)`: agent a is committed to mission m in scheme s.

  TODO: add t-rule

- `achieved(`s`,`g`,`a`)`: goal g in scheme s has been achieved by agent a.

  TODO: add t-rule

  TODO: add leaved_mission

  TODO: add satisfied

- `mission_cardinality(`m`,`min`,`max`)`: is a fact that defines the cardinality of a mission (e.g. `mission_cardinality(mCol,1,5)`).

  The t-rule that produces these facts are:

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

- `mission_role(`m`,`ρ`)`: the role ρ is permitted or obliged to commit to mission m (e.g. `mission_role(mMan,editor)`).

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

- `mission_goal(`m`,`g`)`: the mission m comprises goal g (e.g. `mission_goal(mMan,wsec)`).

  TODO: add rw rule for mission_goal

- `goal(`m`,`g`,`$pre\text{-}cond$`,`t`,`$card$`,'`ttf`')`: is a fact that defines the arguments for a goal g: its missions, identification, pre-conditions, type, cardinality, and TTF (e.g. `goal([mMan],wsec,[wcon],achievement,all,'2 days')`).

  $$\frac{\langle id, gm, type, card, ttf, p\rangle \in \mathcal{G} \qquad id \in schemegoals(S)}{\texttt{goal(}gm\texttt{,}id\texttt{,}gpc(g)\texttt{,}type\texttt{,}card\texttt{,}ttf\texttt{)}.}\ SG(\mathcal{G})$$

  TODO: define function *schemegoals* as the goals included in the scheme. It is computed from the goals tree defined by the plans.

2.3.2 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 enabled to be pursued by the agents or not). The rules produced by *SR* are general for any kind of scheme and those produced by *SRW* are specific for the scheme being translated.

---------- *SR(S)*

```
is_finished(S) :- satisfied(S,g_r).

mission_accomplished(S,M) :-
   .findall(Goal, goal(M,Goal,_,achievement,_,_), 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 enabled to be pursued:
// all its pre-conditions have been achieved
enabled(S,G) :-
   goal(_,G,PCG,_, NP,_) & NP \== 0 & 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

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

$m \in \mathcal{M} \qquad maxmp_S(m) > 0$ ---------- *SRW(S)*

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

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

———————————————————— *SR(S)*

```
norm ngoal:
      committed(A,M,S) & mission_goal(M,G) & goal(_,G,_,_,_,D) &
      well_formed(S) & enabled(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 enabled, then agent `A` is obliged to achieve the goal `G` before its deadline `D`".

The second class of norms is related to properties (see Table 1). For instance, 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. 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 for the property in the normative specification. Otherwise the consequence will be an obligation. The two t-rules below detail the produced norms for the regimentation and obligation cases of mission cardinality.

$$\frac{\nexists\, \langle id, c, \rho, d, m, ttf\rangle \in \mathcal{NS} \mid c = \#p}{}\ \textit{SSP(p)}$$

```
norm p:
      pdc(p)
   -> fail(p).
```

$$\frac{\langle id, c, \rho, d, m, ttf\rangle \in \mathcal{NS} \qquad c = \#p}{}\ \textit{SSP(p)}$$

```
norm id:
      pdc(p) &
      scheme_id(Gr) & responsible(Gr,S) &
      monitor_scheme(MonSch) &
      fplay(A,ρ,Gr)
-> obligation(A,p,committed(A,m,_),‘now‘+‘ttf‘).
```

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 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 current players still does not reach the maximum cardinality, the agent is obliged to achieve a state where it is committed to the mission. The following t-rule expresses just that:

$$\langle id, c, \rho, t, m, ttf\rangle \in \mathcal{NS} \qquad m \in \mathcal{M} \qquad t = obl$$

———————————————————— *NS*

```
norm id:
      c &
      scheme_id(S) & responsible(Gr,S) &
      mplayers(m,S,V) & V < maxmp(m) &
      fplay(A,ρ,Gr) &
      not mission_accomplished(S,m)
-> obligation(A,id,committed(A,m,S),'now'+'ttf').
```

2.4 Organisational Actions

Organisational actions change the state of the organisational entity (OE). Change the OE, new facts are included/removed in the normative program triggering norms.

2.4.1 Role adoption

When agent a adopts the role ρ in the group instance gi, the state of gi is changed as follows:

$$\langle id, GS, players, subgroups, \mathcal{RS}\rangle \longrightarrow \langle id, GS, players \oplus \rho \mapsto \{players(\rho) \cup \{a\}\}, subgroups, \mathcal{RS}\rangle$$

This change in the state produces a new fact `play` (see t-rule *P(GI)*). The fact `play` is used in the following norms.

- property role_in_group: the role being adopted must belong to the group.
- property role_cardinality: the maximal cardinality of the role is not achieved yet.
- property role_compatibility: the new role is compatible with previous.
- norms produced by the t-rule *NS*: agents playing some roles are obliged to commit to some mission.

The role adoption may thus trigger these norms.

2.4.2 Leave role

When agent a leaves the role ρ in the group instance gi, the state of gi is changed as follows:

$$\langle id, GS, players, subgroups, \mathcal{RS}\rangle \longrightarrow \langle id, GS, players \ominus \rho \mapsto \{players(\rho) \cup \{a\}\}, subgroups, \mathcal{RS}\rangle$$

Related norms:

- property well_formed_responsible: this norms is triggered if the role leaving brings the group to a not well formed state and the group is responsible for a scheme. A group can be responsible for a scheme only if well formed.
- norms produced by the t-rule *NS*: if the agent does not play the role anymore, it is not obliged to commit the corresponding missions.

2.4.3 Add responsible group

When the group gi starts being responsible for the scheme instance si, its state changes as follows:

$$\langle id, GS, players, subgroups, \mathcal{RS}\rangle \longrightarrow \langle id, GS, player, subgroups, \mathcal{RS} \cup \{si\}\rangle$$

Related norms:

- property well_formed_responsible: a group can be responsible for a scheme only if well formed.
- norms produced by the t-rule *NS*: agent playing roles in gi are responsible to fulfil the missions of the scheme si.

2.4.4 Commit to mission

When agent a commits to the mission m in the scheme instance si, the state of si is changed as follows:

$$\begin{gathered}\langle id, S, commitments, achievements\rangle \\ \downarrow \\ \langle id, S, commitments \oplus m \mapsto \{commitments(m) \cup \{a\}\}, achievements\rangle\end{gathered}$$

Related norms:

- property mission_permission: the agent have to have the permission for the mission (based on its roles and groups).

- property mission_cardinality.
- norm `ngoal`: the agent has to achieve the enabled goals of its missions.
- norms produced by the t-rule *NS*: the obligations state by these norms are fulfilled.

2.4.5 Leave mission

When agent a commits to the mission m in the scheme instance si, the state of si is changed as follows:

$$\langle id, S, commitments, achievements\rangle$$
$$\downarrow$$
$$\langle id, S, commitments \ominus m \mapsto \{commitments(m) \cup \{a\}\}, achievements\rangle$$

Related norms:

- property mission_leaved: the agent should not leave a mission not accomplished yet.
- norm `ngoal`: the agent is not obliged to achieve the enabled goals of the leaved mission.
- norms produced by the t-rule *NS*: the obligation to commit to the mission may be reintroduced (if the leave operation succeeds).

2.4.6 Goal achievement

When agent a achieves goal g in the scheme instance si, the state of si is changed as follows:

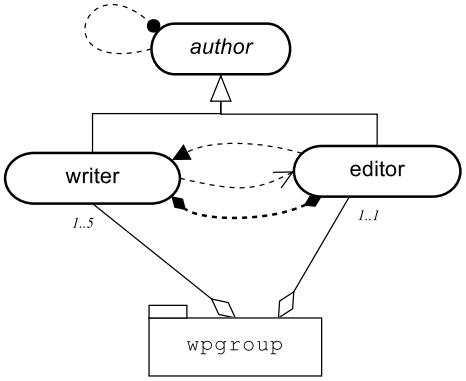
$$\langle id, S, commitments, achievements\rangle$$
$$\downarrow$$
$$\langle id, S, commitments, achievements \oplus a \mapsto \{achievements(a) \cup \{g\}\}\rangle$$

Related norms:

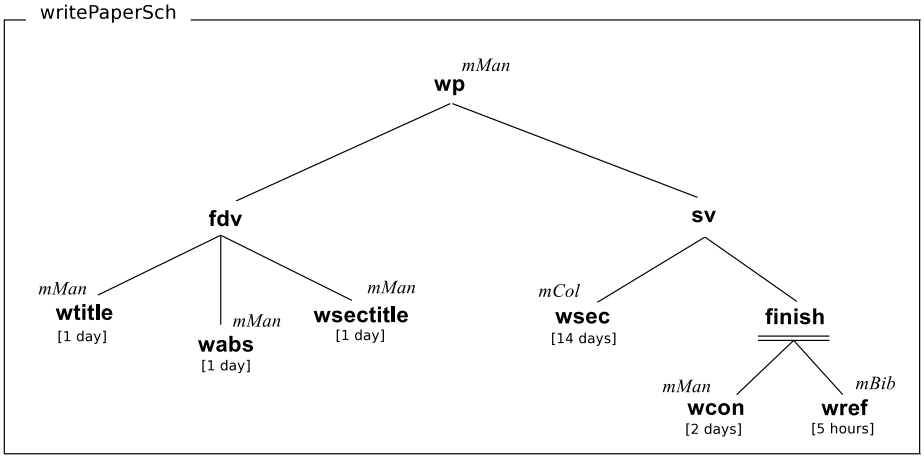
- property ach_not_enabled_goal: the goal has to be enabled.
- property ach_not_committed_goal: the agent have to be committed to a mission that includes the goal.
- norm `ngoal`: the obligation of this norm is fulfilled.

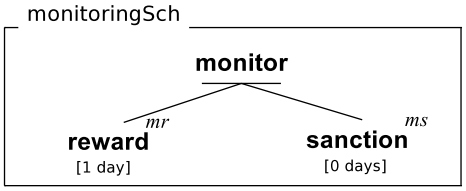
A Writing Paper example

A.1 Structural Specification



A.2 Functional Specification





Mission Cardinalities

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

Goals

goal	type	cardinality	TTF
wtitle	achievement	1	1 day
wabs	achievement	1	1 day
wsectitle	achievement	all	1 day
wsec	achievement	all	7 days
wcon	achievement	1	2 days
wrefs	achievement	all	1 hour
reward	achievement	all	0 day
sanction	achievement	all	0 day

A.3 Normative Specification

id	condition	role	type	mission	TTF
n1		editor	permission	$mMan$	–
n2		writer	obligation	$mCol$	1 day
n3		writer	obligation	$mBib$	1 day
n4	unfulfilled(n2)	editor	obligation	ms	3 hours
n5	fulfilled(n3)	editor	obligation	mr	3 hours
n6	#goal_non_compliance	editor	obligation	ms	3 hours
n7	#role_compatibility	editor	obligation	ms	30 minutes
n8	#mission_cardinality	editor	obligation	ms	1 hour

A.4 NOPL program

```
/*
    This program was automatically generated from
    the organisation specification 'wp'
    on Novembro 12, 2013 - 16:08:32

    This is a MOISE tool, see more at http://moise.sourceforge.net
```

```
*/

scope organisation(wp) {


   // Role hierarchy
   subrole(author,soc).
   subrole(editor,author).
   subrole(writer,author).

   // f* rules implement the role hierarchy transitivity
   // t* rules implement the transitivity of some relations

   // fplay(A,R,G) is true if A play R in G or if A play a subrole of R in G
   fplay(A,R,G) :- play(A,R,G).
   fplay(A,R,G) :- subrole(R1,R) & fplay(A,R1,G).

   // fcompatible(R1,R2,S) is true if R1 or its sub-roles are compatible with R2 in scope S
   fcompatible(R1,R2,S) :- tsubrole(R1,R2).
   fcompatible(R1,R2,S) :- tsubrole(R1,R1a) & tsubrole(R2,R2a) & compatible(R1a,R2a,S).
   fcompatible(R1,R2,S) :- tcompatible(R1,R2,S,[R1,R2]).
   tcompatible(R1,R2,S,Path) :- compatible(R1,R3,S) & not .member(R3,Path) & tcompatible(R3,R2,S,[R3|P
   tsubrole(R,R).
   tsubrole(R1,R2)    :- subrole(R1,R2).
   tsubrole(R1,R2)    :- subrole(R1,R3) & tsubrole(R3,R2).


scope group(wpgroup) {

   // ** Facts from OS
   role_cardinality(editor,1,1).
   role_cardinality(writer,1,5).

   compatible(editor,writer,gr_inst).
   compatible(writer,editor,gr_inst).

   // ** Rules
   rplayers(R,G,V)    :- .count(play(_,R,G),V).
   well_formed(G) :-
      rplayers(editor,G,Veditor) & Veditor >= 1 & Veditor <= 1 &
      rplayers(writer,G,Vwriter) & Vwriter >= 1 & Vwriter <= 5 &
      .findall(GInst, subgroup(GInst,_,G), ListSubgroups) & all_subgroups_well_formed(ListSubgroups).
   all_subgroups_well_formed([]).
   all_subgroups_well_formed([H|T]) :- subgroup_well_formed(H) & all_subgroups_well_formed(T).

   // ** Properties check
   norm role_in_group:
           play(Agt,R,Gr) &
           group_id(Gr) &
           not role_cardinality(R,_,_)
```

```
        -> fail(role_in_group(Agt,R,Gr)).
   norm role_cardinality:
           group_id(Gr) &
           role_cardinality(R,_,RMax) &
           rplayers(R,Gr,RP) &
           RP > RMax
        -> fail(role_cardinality(R,Gr,RP,RMax)).
   norm n7:  // role_compatibility
           play(Agt,R1,Gr) & play(Agt,R2,Gr) & R1 < R2 & not fcompatible(R1,R2,gr_inst) &
           group_id(Gr) & monitor_scheme(MonSch) &
           fplay(A,editor,Gr)
        -> obligation(A,n7(R1,R2,Gr),committed(A,ms,MonSch), `now`+`30 minutes`).
   norm well_formed_responsible:
           responsible(Gr,S) &
           not monitor_scheme(S) &
           not well_formed(Gr)
        -> fail(well_formed_responsible(Gr)).
   norm subgroup_in_group:
           group_id(Gr) &
           subgroup(G,GT,Gr) &
           not subgroup_cardinality(GT,_,_)
        -> fail(subgroup_in_group(G,GT,Gr)).
   norm subgroup_cardinality:
           group_id(Gr) &
           subgroup_cardinality(SG,_,SGMax) &
           .count(subgroup(_,SG,Gr),SGP) &
           SGP > SGMax
        -> fail(subgroup_cardinality(SG,Gr,SGP,SGMax)).
} // end of group wpgroup

scope scheme(writePaperSch) {

   // ** Facts from OS
   mission_cardinality(mManager,1,1).
   mission_cardinality(mColaborator,1,5).
   mission_cardinality(mBib,1,1).

   mission_role(mManager,editor).
   mission_role(mColaborator,writer).
   mission_role(mBib,writer).

   mission_goal(mManager,wtitle).
   mission_goal(mManager,wsectitles).
   mission_goal(mManager,wabs).
   mission_goal(mManager,wconc).
   mission_goal(mManager,wp).
   mission_goal(mColaborator,wsecs).
   mission_goal(mBib,wrefs).

   goal([],fdv,[wsectitles],achievement,0,`1 year`).
```

```
goal([],finish,[wconc, wrefs],achievement,0,‘1 year‘).
goal([mManager],wtitle,[],achievement,all,‘1 day‘).
goal([mManager],wsectitles,[wabs],achievement,all,‘1 day‘).
goal([mColaborator],wsecs,[fdv],achievement,all,‘7 days‘).
goal([mManager],wabs,[wtitle],achievement,all,‘1 day‘).
goal([mManager],wp,[sv],achievement,all,‘5 seconds‘).
goal([mManager],wconc,[wsecs],achievement,all,‘1 day‘).
goal([],sv,[finish],achievement,0,‘1 year‘).
goal([mBib],wrefs,[wsecs],achievement,all,‘1 hour‘).

// ** Rules
mplayers(M,S,V) :- .count(committed(_,M,S),V).
well_formed(S) :-
   (mission_accomplished(S,mManager) | mplayers(mManager,S,VmManager) & VmManager >= 1 & VmManager
   (mission_accomplished(S,mColaborator) | mplayers(mColaborator,S,VmColaborator) & VmColaborator >
   (mission_accomplished(S,mBib) | mplayers(mBib,S,VmBib) & VmBib >= 1 & VmBib <= 1).
is_finished(S) :- satisfied(S,wp).
mission_accomplished(S,M) :- .findall(Goal, mission_goal(M,Goal), MissionGoals) & all_satisfied(S,M
all_satisfied(_,[]).
all_satisfied(S,[G|T]) :- satisfied(S,G) & all_satisfied(S,T).

// enabled goals (i.e. dependence between goals)
enabled(S,G) :- goal(_, G, PCG, _, NP, _) & NP \== 0 & all_satisfied(S,PCG).

// ** Norms

norm n3:
        scheme_id(S) & responsible(Gr,S) &
        mplayers(mColaborator,S,V) & V < 5 &
        fplay(A,writer,Gr) &
        not mission_accomplished(S,mColaborator) // if all mission’s goals are satisfied, the agent
     -> obligation(A,n3,committed(A,mColaborator,S), ‘now‘+‘1 day‘).
norm n2:
        scheme_id(S) & responsible(Gr,S) &
        mplayers(mBib,S,V) & V < 1 &
        fplay(A,writer,Gr) &
        not mission_accomplished(S,mBib) // if all mission’s goals are satisfied, the agent is not
     -> obligation(A,n2,committed(A,mBib,S), ‘now‘+‘1 day‘).

// --- Goals ---
// agents are obliged to fulfill their enabled goals
norm ngoal:
        committed(A,M,S) & mission_goal(M,G) & goal(_,G,_,achievement,_,D) &
        well_formed(S) & not satisfied(S,G) & enabled(S,G)
     -> obligation(A,ngoal(S,M,G),achieved(S,G,A),‘now‘ + D).

// --- Properties check ---
norm n6:  // goal_non_compliance
        obligation(Agt,ngoal(S,M,G),Obj,TTF) & not Obj & ‘now‘ > TTF &
        scheme_id(S) & responsible(Gr,S) & monitor_scheme(MonSch) &
```

```
            fplay(A,editor,Gr)
         -> obligation(A,n6(obligation(Agt,ngoal(S,M,G),Obj,TTF)),committed(A,ms,MonSch), `now`+`3 hour
      norm mission_permission:
            committed(Agt,M,S) &
            not (mission_role(M,R) &
            responsible(Gr,S) &
            fplay(Agt,R,Gr))
         -> fail(mission_permission(Agt,M,S)).
      norm mission_left:
            leaved_mission(Agt,M,S) &
            not mission_accomplished(S,M)
         -> fail(mission_left(Agt,M,S)).
      norm n8:  // mission_cardinality
            scheme_id(S) & mission_cardinality(M,_,MMax) & mplayers(M,S,MP) & MP > MMax &
            scheme_id(S) & responsible(Gr,S) & monitor_scheme(MonSch) &
            fplay(A,editor,Gr)
         -> obligation(A,n8(M,S,MP,MMax),committed(A,ms,MonSch), `now`+`1 hour`).
      norm ach_not_enabled_goal:
            achieved(S,G,Agt) &
            mission_goal(M,G) &
            not mission_accomplished(S,M) &
            not enabled(S,G)
         -> fail(ach_not_enabled_goal(S,G,Agt)).
      norm ach_not_committed_goal:
            achieved(S,G,Agt) &
            mission_goal(M,G) &
            not mission_accomplished(S,M) &
            not committed(Agt,M,S)
         -> fail(ach_not_committed_goal(S,G,Agt)).
   } // end of scheme writePaperSch

   scope scheme(monitoringSch) {

      // ** Facts from OS
      mission_cardinality(ms,1,1).
      mission_cardinality(mr,1,1).

      mission_role(mr,editor).
      mission_role(ms,editor).

      mission_goal(ms,sanction).
      mission_goal(mr,reward).

      goal([],monitor,[],achievement,0,`1 year`).
      goal([mr],reward,[],achievement,all,`1 year`).
      goal([ms],sanction,[],achievement,all,`1 year`).

      // ** Rules
      mplayers(M,S,V) :- .count(committed(_,M,S),V).
      well_formed(S) :-
```

```
   (mission_accomplished(S,ms) | mplayers(ms,S,Vms) & Vms >= 1 & Vms <= 1) &
   (mission_accomplished(S,mr) | mplayers(mr,S,Vmr) & Vmr >= 1 & Vmr <= 1).
is_finished(S) :- satisfied(S,monitor).
mission_accomplished(S,M) :- .findall(Goal, mission_goal(M,Goal), MissionGoals) & all_satisfied(S,M
all_satisfied(_,[]).
all_satisfied(S,[G|T]) :- satisfied(S,G) & all_satisfied(S,T).

// enabled goals (i.e. dependence between goals)
enabled(S,G) :- goal(_, G, PCG, _, NP, _) & NP \== 0 & all_satisfied(S,PCG).

// --- Goals ---
// agents are obliged to fulfill their enabled goals
norm ngoal:
        committed(A,M,S) & mission_goal(M,G) & goal(_,G,_,achievement,_,D) &
        well_formed(S) & not satisfied(S,G) & enabled(S,G)
     -> obligation(A,ngoal(S,M,G),achieved(S,G,A),`now` + D).

// --- Properties check ---
norm goal_non_compliance:
        obligation(Agt,ngoal(S,M,G),Obj,TTF) &
        not Obj &
        `now` > TTF
     -> fail(goal_non_compliance(obligation(Agt,ngoal(S,M,G),Obj,TTF))).
norm mission_permission:
        committed(Agt,M,S) &
        not (mission_role(M,R) &
        responsible(Gr,S) &
        fplay(Agt,R,Gr))
     -> fail(mission_permission(Agt,M,S)).
norm mission_left:
        leaved_mission(Agt,M,S) &
        not mission_accomplished(S,M)
     -> fail(mission_left(Agt,M,S)).
norm mission_cardinality:
        scheme_id(S) &
        mission_cardinality(M,_,MMax) &
        mplayers(M,S,MP) &
        MP > MMax
     -> fail(mission_cardinality(M,S,MP,MMax)).
norm ach_not_enabled_goal:
        achieved(S,G,Agt) &
        mission_goal(M,G) &
        not mission_accomplished(S,M) &
        not enabled(S,G)
     -> fail(ach_not_enabled_goal(S,G,Agt)).
norm ach_not_committed_goal:
        achieved(S,G,Agt) &
        mission_goal(M,G) &
        not mission_accomplished(S,M) &
        not committed(Agt,M,S)
```

```
        -> fail(ach_not_committed_goal(S,G,Agt)).
} // end of scheme monitoringSch

} // end of organisation wp
```

References

[Gâteau et al., 2005] Gâteau, B., Boissier, O., Khadraoui, D., and Dubois, E. (2005). MOISEinst: An organizational model for specifying rights and duties of autonomous agents. In *Third European Workshop on Multi-Agent Systems (EUMAS 2005)*, pages 484–485, Brussels Belgium.

[Hannoun, 2002] Hannoun, M. (2002). *MOISE: un modèle organisationnel pour les systèmes multi-agents*. Thèse (doctorat), École Nationale Supérieure des Mines de Saint-Etienne.

[Hübner et al., 2009] Hübner, J. F., Boissier, O., and Bordini, R. H. (2009). Normative programming for organisation management infrastructures. In Polleres, A. and Padget, J., editors, *Workshop on Coordination, Organization, Institutions and Norms in agent systems (COIN09@MALLOW) Torino, Italy, 7th–11th September*, volume 494. CEUR.

[Hübner et al., 2002] Hübner, J. F., Sichman, J. S., and Boissier, O. (2002). A model for the structural, functional, and deontic specification of organizations in multiagent systems. In Bittencourt, G. and Ramalho, G. L., editors, *Proceedings of the 16th Brazilian Symposium on Artificial Intelligence (SBIA'02)*, volume 2507 of *LNAI*, pages 118–128, Berlin. Springer.

[van Riemsdijk et al., 2010] van Riemsdijk, B., Hindriks, K., Jonker, C. M., and Sierhuis, M. (2010). Formal organizational constraints: A semantic approach. In van der Hoek, W., Kaminka, G. A., Lespérance, Y., Luck, M., and Sen, S., editors, *Proc. of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 823–830.