Multi-Agent Oriented Programming

The JaCaMo Platform

O. Boissier[1] R.H. Bordini[2] J.F. Hübner[3] A. Ricci[4]

1. Mines Saint-Etienne (ENSMSE), Saint Etienne, France

2 Pontificia Universidade Catolica do Rio Grande do Sul (PUCRS), Porto Alegre, Brazil

3. Federal University of Santa Catarina (UFSC), Florianópolis, Brazil

4. University of Bologna (UNIBO), Bologna, Italy

November 2017

Tutorial Organisation

- Introduction to Multi-Agent Oriented Programming
- Programming Agents
- Programming Agents' Environment
- Programming Agents' Interaction
- Programming Agents' Organisations
- Programming Applications
- Conclusion & Perspectives

Multi-Agent Oriented Programming

Programming Agents' Organisations

Outline

Programming Agents' Organisations
- Fundamentals
- Existing approaches
- $\mathcal{M}$oise Modeling Language (OML)
- $\mathcal{M}$oise Management Infrastructure (OMI)
- $\mathcal{M}$oise and Environment (O-E)
- $\mathcal{M}$oise and Agents (O-A)
- Conclusions and wrap-up

Intuitive notions of organisation

- Organisations are structured, patterned systems of activity, knowledge, culture, memory, history, and capabilities that are distinct from any single agent [Gasser, 2001]
  $\leadsto$ Organisations are supra-individual phenomena
- A decision and communication schema which is applied to a set of actors that together fulfill a set of tasks in order to satisfy goals while guarantying a global coherent state [Malone, 1999]
  $\leadsto$ definition by the designer, or by actors, to achieve a purpose
- An organisation is characterized by : a division of tasks, a distribution of roles, authority systems, communication systems, contribution-retribution systems [Bernoux, 1985]
  $\leadsto$ pattern of predefined cooperation
- An arrangement of relationships between components, which results into an entity, a system, that has unknown skills at the level of the individuals [Morin, 1977]
  $\leadsto$ pattern of emergent cooperation

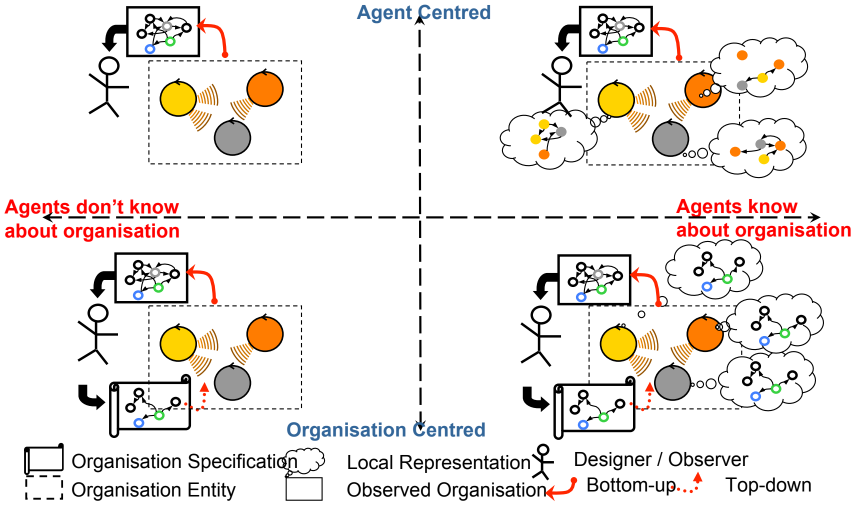
Organisation in MAS

Definition

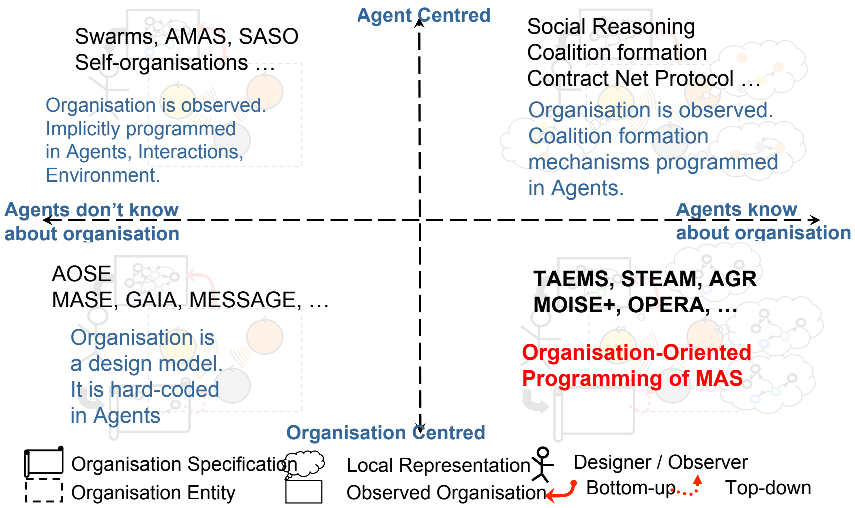
Purposive supra-agent pattern of emergent or (pre)defined agents cooperation, that could be defined by the designer or by the agents themselves.

- Pattern of emergent/potential cooperation
  - called organisation entity, institution, social relations, commitments
- Pattern of (pre)defined cooperation
  - called organisation specification, structure, norms, ...

Perspective on organisations from EASSS'05 Tutorial (Sichman, Boissier)



Perspective on organisations from EASSS'05 Tutorial (Sichman, Boissier)



Perspective on Org.-Oriented Programming of MAS

From organisations as

- an explicit description of the structure of the agents in the MAS in order to help them to interact

To organisations as

- the declarative and explicit definition of the coordination scheme aiming at “controlling/coordinating” the global reasoning of the MAS

$\rightsquigarrow$ Normative Organisations

Norms

Norm

Norms are rules that a society has in order to influence the behaviour of agents.

Norm mechanisms

- Regimentation: norm violation by the agents is prevented
  - e.g. the access to computers requires an user name
  - e.g. messages that do not follow the protocol are discarded
- Enforcement: norm violation by the agents is made possible but it is monitored and subject to incentives
  - e.g. a master thesis should be written in two years

  $\rightsquigarrow$ Detection of violations, decision about ways of enforcing the norms (e.g. sanctions)

Normative Multi-Agent Organisation

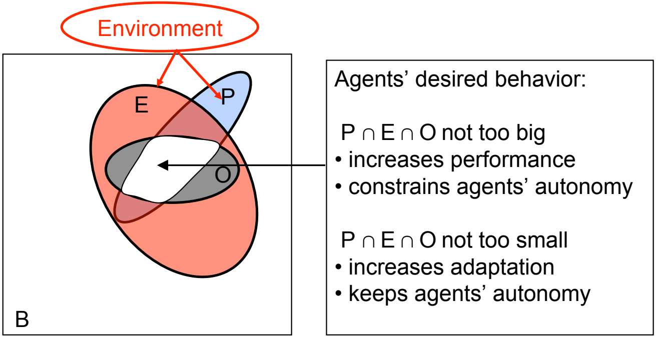
Normative Multi-Agent System [Boella et al., 2008]

A MAS composed of mechanisms to represent, communicate, distribute, detect, create, modify, and enforce norms, and mechanisms to deliberate about norms and detect norm violation and fulfillment.

Normative Multi-Agent Organisation [?]

- Norms are expressed in the organisation specification to clearly define the coordination of the MAS:
  - anchored/situated in the organisation
  - i.e. norms refer to organisational concepts (roles, groups, etc.)
- Norms are interpreted and considered in the context of the organisation entity
- Organisation management mechanisms are complemented with norms management mechanisms (enforcement, regimentation, ...)

Challenges: Normative Organisation vs Autonomy



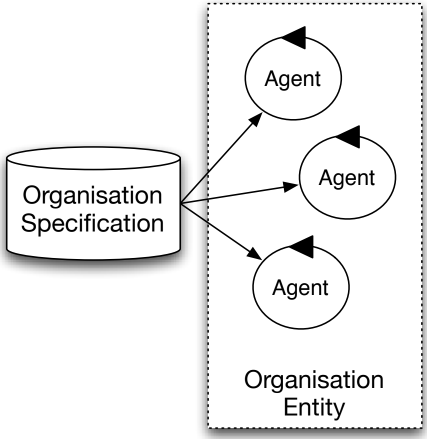
- B: agents' possible behaviors
- P: agents' behaviors that lead to global purpose
- E: agents' possible behaviors constrained by the environment
- O: agents' possible/permitted/obliged behaviors constrained by the normative organisation

Organisation Oriented Programming (OOP)

Organisation as a first class entity in the multi-agent eco-system

- Clear distinction between description of the organisation **wrt** agents, **wrt** environment
- Different representations of the organisation:
  - Organisation specification
    - partially/totally accessible to the agents, to the environment, to the organisation
  - Organisation entity
    - Local representation in the mental state of the agents
      $\rightsquigarrow$ possibly inconsistant with the other agents' representations
    - Global/local representation in the MAS
      $\rightsquigarrow$ difficulty to manage and build such a representation in a distributed and decentralized setting
- Different sources of actions on (resp. of) the organisation by (resp. on) agents / environment / organisation

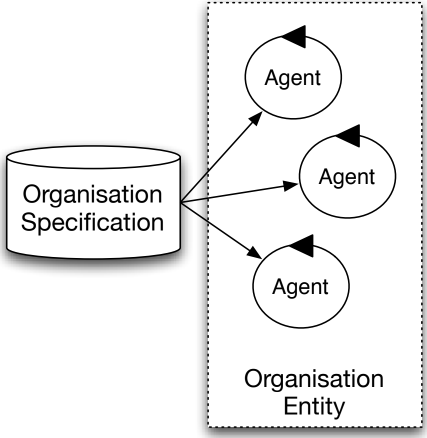
Organisation Oriented Programming (OOP)



- Using organisational concepts
- To define a cooperative pattern
- Programmed outside of the agents and outside of the environment

- Program = Specification
- By changing the organisation, we can change the MAS overall behaviour

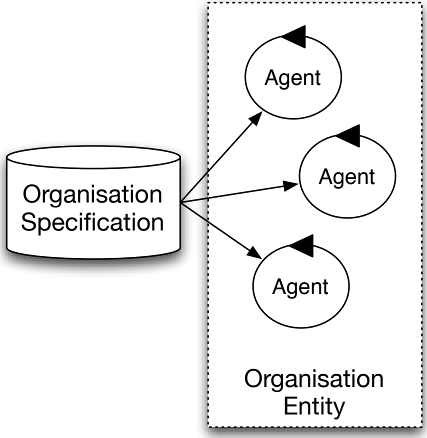
Organisation Oriented Programming (OOP)



First approach

- Agents read the program and follow it

Organisation Oriented Programming (OOP)



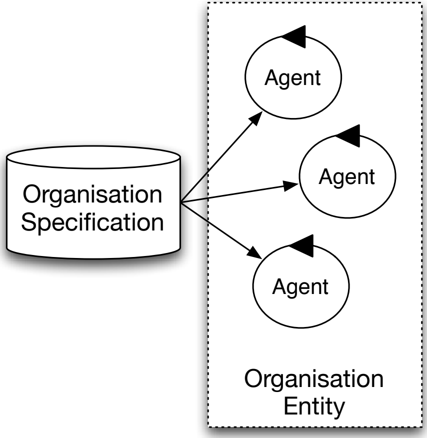
First approach

- Agents read the program and follow it

Second approach

- regimentation
  - Agents are forced to follow the program
- enforcement
  - Agents are rewarded if they follow the program
  - Agents are sanctioned in the other case

Organisation Oriented Programming (OOP)



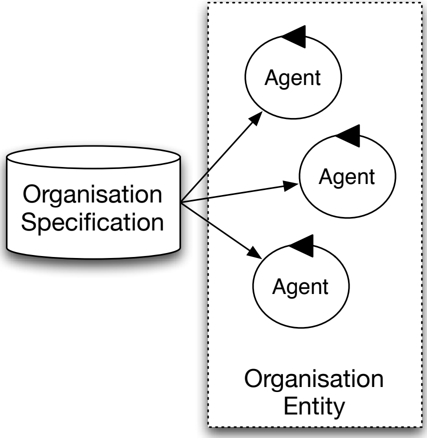
First approach

- Agents read the program and follow it

Second approach

- regimentation
  - Agents are forced to follow the program
- enforcement
  - Agents are rewarded if they follow the program
  - Agents are sanctioned in the other case

Organisation Oriented Programming (OOP)



Components

- Programming Language (Org. Modeling Lang. – OML)
- Management Infrastructure (Org. Mngt Inf. – OMI)
- Integration to Agent architectures and to Environment

Components of OOP: Organisation Modelling Language (OML)

- Declarative specification of the organisation(s)
- Specific constraints, norms and cooperation patterns imposed on the agents

  e.g. AGR [Ferber and Gutknecht, 1998],
  TeamCore [Tambe, 1997],
  Islander [Esteva et al., 2001],
  $\mathcal{M}$oise^{+} [Hübner et al., 2002], ...
- Specific anchors for situating organisations within the environment

  e.g. embodied organisations [Piunti et al., 2009]

Components of OOP: Organisation Management Infrastructure (OMI)

- Coordination mechanisms, i.e. support infrastructure
  - e.g. MadKit [Gutknecht and Ferber, 2000],
    karma [Pynadath and Tambe, 2003],
    ...
- Regulation mechanisms, i.e. governance infrastructure
  - e.g. Ameli [Esteva et al., 2004],
    $\mathcal{S}$-$\mathcal{M}$oise^{+} [Hübner et al., 2006],
    ORA4MAS [Hübner et al., 2009],
    ...
- Adaptation mechanisms, i.e. reorganisation infrastructure

Components of OOP: Integration mechanisms

- Agent integration mechanisms allow agents to be aware of and to deliberate on:
  - entering/exiting the organisation
  - modification of the organisation
  - obedience/violation of norms
  - sanctioning/rewarding other agents

  e.g. $\mathcal{J}$-$\mathcal{M}$oise$^+$ [Hübner et al., 2007], Autonomy based reasoning [Carabelea, 2007], *ProsA*$_2$ Agent-based reasoning on norms [Ossowski, 1999], ...

- Environment integration mechanisms transform organisation into embodied organisation so that:
  - organisation may act on the environment (e.g. enact rules, regimentation)
  - environment may act on the organisation (e.g. count-as rules)

  e.g [de Brito et al., 2012], [**?**], [Okuyama et al., 2008]

Motivations for OOP: **Applications** point of view

- Current applications show an increase in
  - Number of agents
  - Duration and repetitiveness of agent activities
  - Heterogeneity of the agents, Number of designers of agents
  - Agent ability to act, to decide,
  - Action domains of agents, ...
  - Openness, scalability, dynamicity, ...

- More and more applications require the integration of human communities and technological communities (ubiquitous and pervasive computing), building connected communities (ICities) in which agents act on behalf of users
  - Trust, security, ..., flexibility, adaptation

Motivations for OOP: **Constitutive** point of view

- Organisation helps the agents to cooperate with the other agents by defining common cooperation schemes
  - global tasks
  - protocols
  - groups, responsibilities

e.g. ‘to bid’ for a product on eBay is an institutional action only possible because eBay defines the rules for that very action

- the bid protocol is a constraint but it also creates the action

e.g. when a soccer team plays a match, the organisation helps the members of the team to synchronise actions, to share information, etc

Motivations for OOP: **Normative** point of view

- MAS have two properties which seem contradictory:
  - a global purpose
  - autonomous agents

  ⇝ While the autonomy of the agents is essential, it may cause loss in the global coherence of the system and achievement of the global purpose
- Embedding norms within the organisation of a MAS is a way to constrain the agents' behaviour towards the global purposes of the organisation, while explicitly addressing the autonomy of the agents within the organisation

  ⇝ Normative organisation

  e.g. when an agent adopts a role, it adopts a set of behavioural constraints that support the global purpose of the organisation. It may decide to obey or disobey these constraints

Motivations for OOP: **Agents** point of view

An organisational specification is required to enable agents to “reason” about the organisation:

- to decide to enter into/leave from the organisation during execution
  - ⇝ Organisation is no more closed
- to change/adapt the current organisation
  - ⇝ Organisation is no more static
- to obey/disobey the organisation
  - ⇝ Organisation is no more a regimentation

Motivations for OOP: **Organisation** point of view

An organisational specification is required to enable the organisation to "reason" about itself and about the agents in order to ensure the achievement of its global purpose:

- to decide to let agents enter into/leave from the organisation during execution
  - $\rightsquigarrow$ Organisation is no more closed
- to decide to let agents change/adapt the current organisation
  - $\rightsquigarrow$ Organisation is no more static and blind
- to govern agents behaviour in the organisation (i.e. monitor, enforce, regiment)
  - $\rightsquigarrow$ Organisation is no more a regimentation

Outline

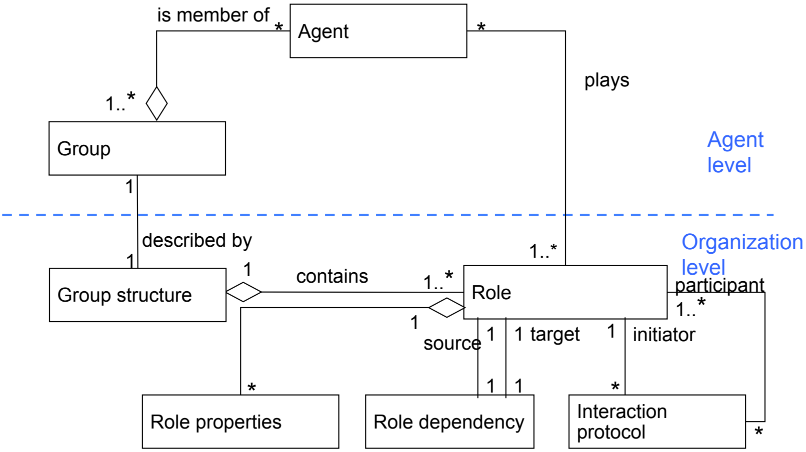
Programming Agents' Organisations

- Fundamentals
- Existing approaches
- $\mathcal{M}$oise Modeling Language (OML)
- $\mathcal{M}$oise Management Infrastructure (OMI)
- $\mathcal{M}$oise and Environment (O-E)
- $\mathcal{M}$oise and Agents (O-A)
- Conclusions and wrap-up

AGR [Ferber and Gutknecht, 1998]

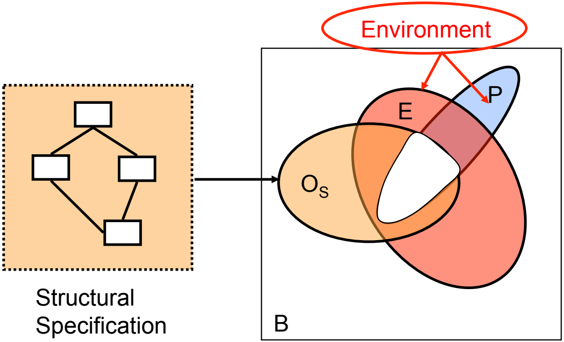
- Agent Group Role, previously known as AALAADIN
  - Agent: Active entity that plays roles within groups. An agent may have several roles and may belong to several groups.
  - Group: set of agents sharing common characteristics, i.e. context for a set of activities. Two agents can't communicate with each other if they don't belong to the same group.
  - Role: Abstract representation of the status, position, function of an agent within a group.
- OMI: the Madkit platform

AGR OML



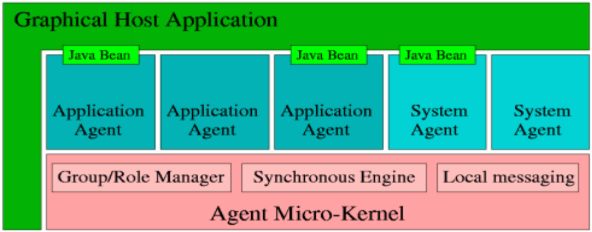
is member of
*
Agent
*
plays
1..*
Group
1
Agent
level
described by
1
1
1..*
Organization
level
Group structure
contains
1..*
Role
participant
1..*
1
source
1
1 target
1
initiator
*
1
1
*
Role properties
Role dependency
Interaction
protocol
*

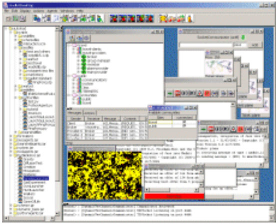
AGR OML Modelling Dimensions



B: agents' possible behaviors
P: agents' behaviors that lead to global purpose
E: agents' possible behaviors constrained by the environment
O_S: agents' possible behaviors structurally constrained by the organization

AGR OMI: Madkit





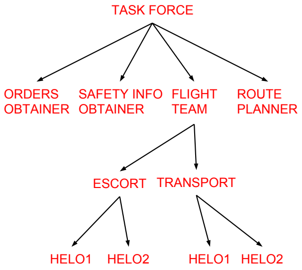
Multi-Agent Development Kit

www.madkit.org

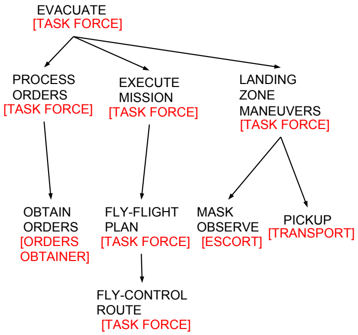
STEAM [Tambe, 1997]

- Shell for TEAMwork is a general framework to enable agents to participate in teamwork.
  - Different applications: Attack, Transport, Robocup soccer
  - Based on an enhanced SOAR architecture and 300 domain independent SOAR rules
- Principles:
  - Team synchronization: Establish joint intentions, Monitor team progress and repair, Individual may fail or succeed in own role
  - Reorganise if there is a critical role failure
  - Reassign critical roles based on joint intentions
  - Decision theoretic communication
- Supported by the TEAMCORE OMI.

STEAM OML [Tambe, 1997]

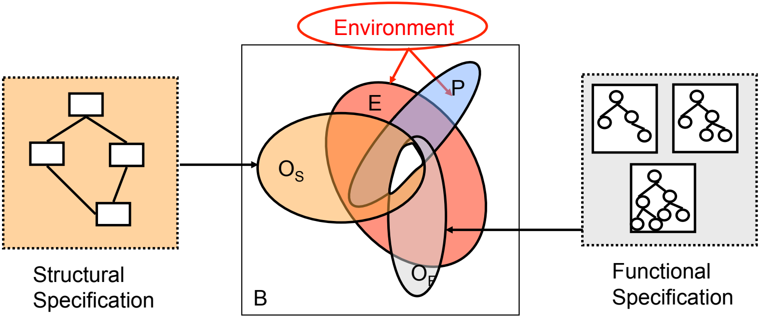


Organization: hierarchy of roles that may be filled by agents or groups of agents.



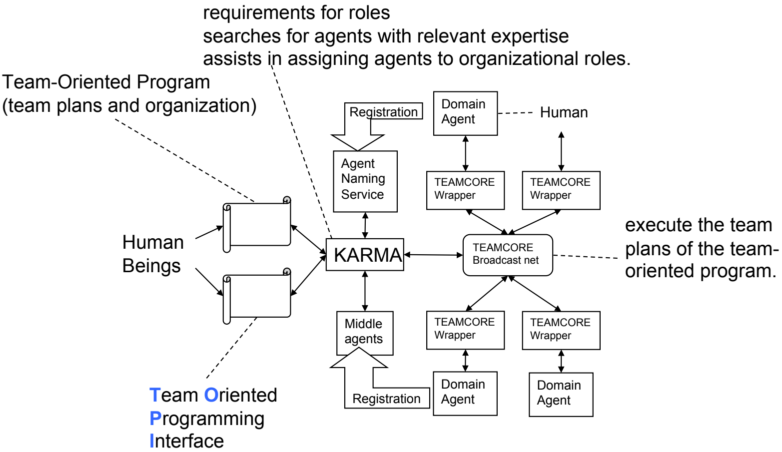
Team Plan:
- initial conditions,
- term. cond. : achievability, irrelevance, unachievability
- team-level actions.

STEAM OML Modelling Dimensions



B: agents' possible behaviors
P: agents' behaviors that lead to global purpose
E: agents' possible behaviors constrained by the environment
O_S: agents' possible behaviors structurally constrained by the organization
O_F: agents' possible behaviors functionally constrained by the organization

STEAM OMI: TEAMCORE [Pynadath and Tambe, 2003]



ISLANDER

- Based on different influences: economics, norms, dialogues, coordination

$\rightsquigarrow$ electronic institutions

- Combining different alternative views: dialogical, normative, coordination
- Institution Description Language:
  - Performative structure (Network of protocols),
  - Scene (multi-agent protocol),
  - Roles,
  - Norms
- Ameli as OMI

ISLANDER OML: IDL [Esteva et al., 2001]

(define-institution

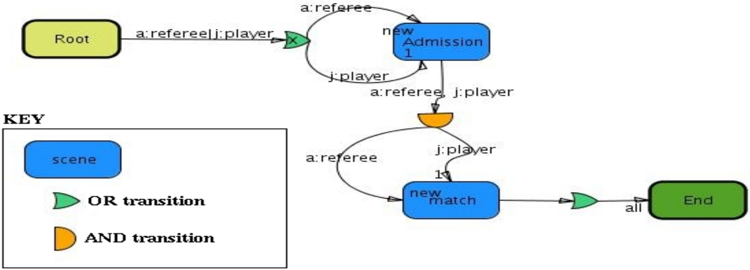
soccer-server as

dialogic-framework = soccer-df

performative-structure = soccer-pf

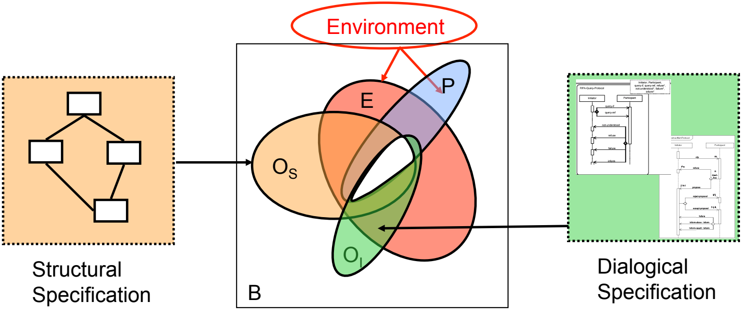
norms = (free-kick coach-messages …)

)



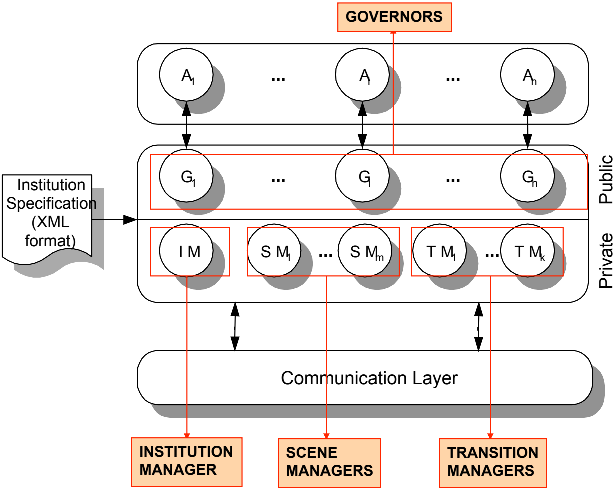
Performative Structure

ISLANDER OML Modelling Dimensions



B: agents' possible behaviors
P: agents' behaviors that lead to global purpose
E: agents' possible behaviors constrained by the environment
O_S: agents' possible/permitted/obliged behaviors structurally constrained by the organisation
O_I: agents' possible/permitted/obliged behaviors interactionally constrained by the organisation

ISLANDER OMI: AMELI [Esteva et al., 2004]



From [Noriega 04]

The aim is to design and develop a programming language to support the implementation of coordination mechanisms in terms of normative concepts.

An organisation

- determines effect of external actions
- normatively assesses effect of agents' actions (monitoring)
- sanctions agents' wrongdoings (enforcement)
- prevents ending up in really bad states (regimentation)

Programming Language for Organisations

Example (Train Station)

```
Facts:
    { -at_platform , -in_train , -ticket }

Effects:
    { -at_platform }    enter            { at_platform },
    { -ticket }         buy_ticket       { ticket },
    { at_platform , -in_train }
                        embark
                            { -at_platform, in_train }
Counts_as rules:
    { at_platform , -ticket } => { viol_ticket },
    { in_train , -ticket }    => { viol_|_ }

Sanction_rules:
    { viol_ticket } => { fined_10 }
```

2OPL Modelling Dimension

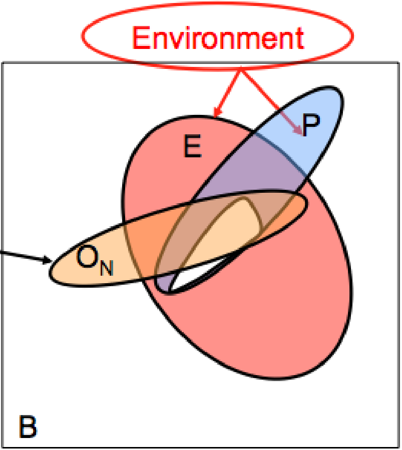
Example (Train Station)

```
Facts:
    { -at_platform , -in_train , -ticket }

Effects:
    { -at_platform }    enter         { at_platform },
    { -ticket }         buy_ticket    { ticket },
    { at_platform , -in_train }
                        embark
                            { -at_platform, in_train }
Counts_as rules:
    { at_platform , -ticket } => { viol_ticket },
    { in_train , -ticket }    => { viol_|_ }

Sanction_rules:
    { viol_ticket } => { fined_10 }
```

Normative Specification



Summary

- Several models
- Several dimensions on modelling organisation
  - Structural (roles, groups, ...)
  - Functional (global plans, ...)
  - Dialogical (scenes, protocols, ...)
  - Normative (norms)
- Several ways of managing organization within the MAS
- Several ways of addressing the autonomy of the agents

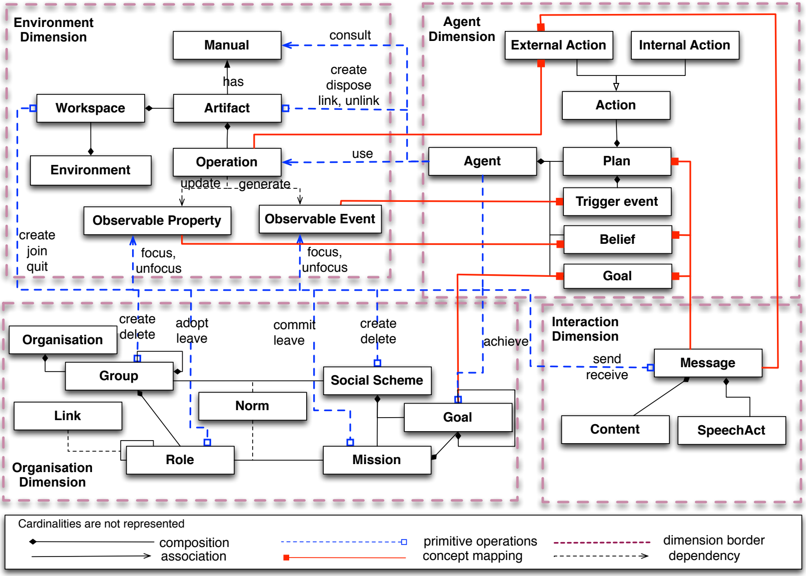
$\mathcal{M}$oise

(let's go programming those nice concepts)

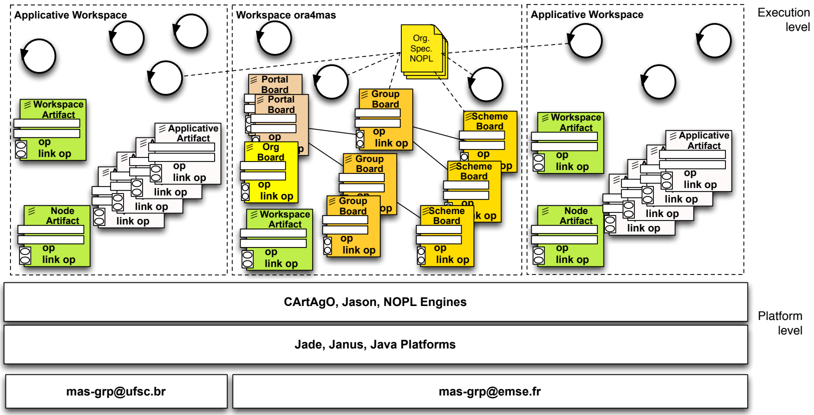
$\mathcal{M}$oise Framework

- OML (language)
  - Tag-based language
    (issued from $\mathcal{M}$oise [Hannoun et al., 2000],
    $\mathcal{M}$oise$^+$ [Hübner et al., 2002], $\mathcal{M}$oiseInst [Gâteau et al., 2005])
- OMI (infrastructure)
  - developed as an artifact-based working environment
    (ORA4MAS [Hübner et al., 2009] based on CArtAgO nodes,
    refactoring of $\mathcal{S}$-$\mathcal{M}$oise$^+$ [Hübner et al., 2006] and
    $\mathcal{S}$ynai [Gâteau et al., 2005])
- Integrations
  - Agents and Environment (c4Jason, c4Jadex [Ricci et al., 2009])
  - Environment and Organisation ([Piunti et al., 2009])
  - Agents and Organisation ($\mathcal{J}$-$\mathcal{M}$oise$^+$ [Hübner et al., 2007])

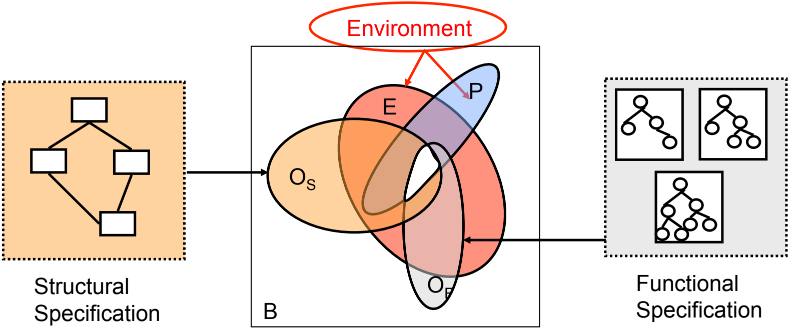
Moise in JaCaMo Metamodel



Moise Framework in JaCaMo



$\mathcal{M}$oise Modelling Dimensions



Groups, links, roles
Compatibilities, multiplicities
inheritance

Normative Specification
Permissions, Obligations
Allows agents autonomy!

Global goals, plans,
Missions, schemas,
preferences

Outline

Programming Agents' Organisations
- Fundamentals
- Existing approaches
- $\mathcal{M}$oise Modeling Language (OML)
- $\mathcal{M}$oise Management Infrastructure (OMI)
- $\mathcal{M}$oise and Environment (O-E)
- $\mathcal{M}$oise and Agents (O-A)
- Conclusions and wrap-up

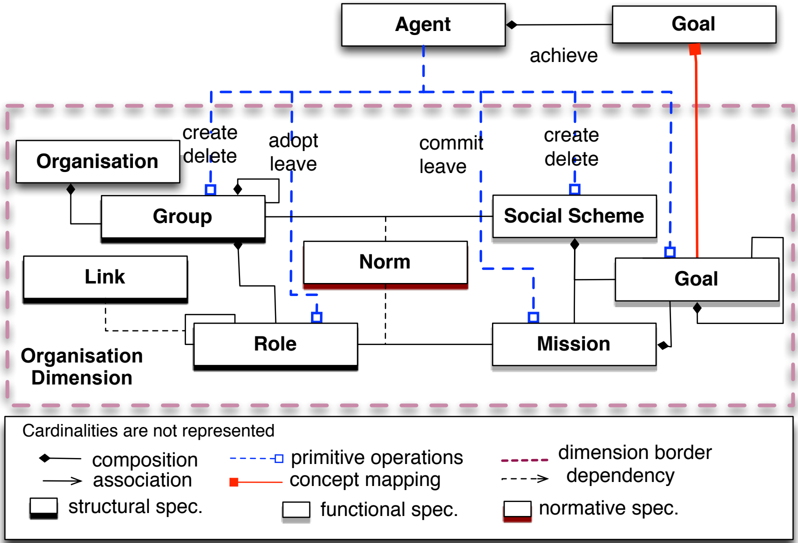
$\mathcal{M}$oise OML

- OML for defining organisation specification and organisation entity
- Three independent dimensions [Hübner et al., 2007]
  ($\rightsquigarrow$ well adapted for the reorganisation concerns):
  - Structural: Roles, Groups
  - Functional: Goals, Missions, Schemes
  - Normative: Norms (obligations, permissions, interdictions)
- Abstract description of the organisation for
  - the designers
  - the agents

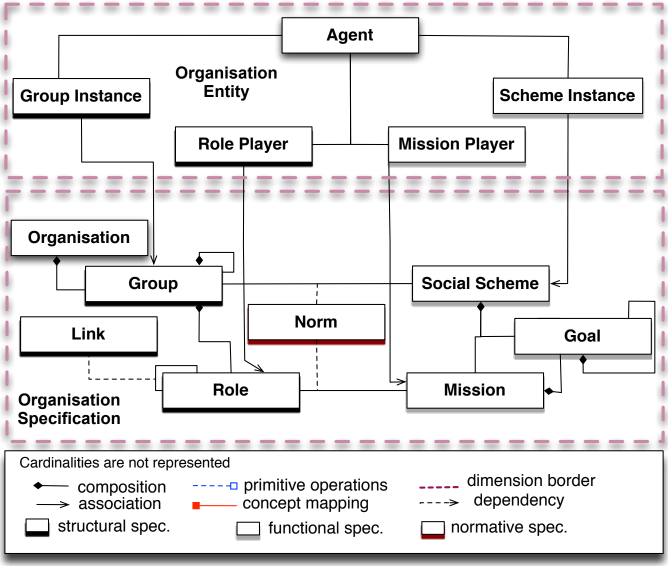
    $\rightsquigarrow$ $\mathcal{J}$-$\mathcal{M}$oise [Hübner et al., 2007]
  - the Organisation Management Infrastructure

    $\rightsquigarrow$ ORA4MAS [Hübner et al., 2009]

Moise OML meta-model (partial & simplified view)



Moise OML global picture



Structural Specification

- Specifies the structure of an MAS along three levels:
  - Individual with Role
  - Social with Link
  - Collective with Group
- Components:
  - Role: label used to assign constraints on the behavior of agents playing it
  - Link: relation between roles that directly constrains the agents in their interaction with the other agents playing the corresponding roles
  - Group: set of links, roles, compatibility relations used to define a shared context for agents playing roles in it

Structural specification

- Defined with the tag structural-specification in the context of an organisational-specification
- One section for definition of all the roles participating to the structure of the organisation (role-definitions tag)
- Specification of the group including all subgroup specifications (group-specification tag)

Example

```
 <organisational-specification
   <structural-specification>
      <role-definitions>  ... </role-definitions>
      <group-specification id="xxx">
        ...
      </group-specification>
   </structural-specification>
   ...
</organisational-specification>
```

Role specification

- Role definition(role tag) in role-definitions section, is composed of:
  - identifier of the role (id attribute of role tag)
  - inherited roles (extends tag) - by default, all roles inherit of the soc role -

Example

```
<role-definitions>
  <role id="player" />
  <role id="coach" />
  <role id="middle"> <extends role="player"/> </role>
  <role id="leader"> <extends role="player"/> </role>
  <role id="r1>
    <extends role="r2" />
    <extends role="r3" />
  </role>
  ...
</role-definitions>
```

Group specification

- Group definition (group-specification tag) is composed of:
  - group identifier (id attribute of group-specification tag)
  - roles participating to this group and their cardinality (roles tag and id, min, max), i.e. min. and max. number of agents that should adopt the role in the group (default is 0 and unlimited)
  - links between roles of the group (link tag)
  - subgroups and their cardinality (subgroups tag)
  - formation constraints on the components of the group (formation-constraints)

Example

```
<group-specification id="team">
   <roles>
        <role id="coach" min="1" max="2"/> ...
   </roles>
   <links> ... </links>
   <subgroups> ... </subgroups>
   <formation-constraints> ... </formation-constraints>
</group-specification>
```

extends-subgroups, scope

extends-subgroups

- Used for links or formation constraints
- if extends-subgroups== true, the link/constraint is also valid in all subgroups
- else it is valid only in the group where it is defined
- Default is false

scope

- Used for links or formation constraints
- if scope==inter-group: link or constraint exists for source or target belonging to different instances of the group
- if scope==intra-group: link or constraint exists for source or target belonging to the same instance of the group

Link specification

- Link definition (link tag) included in the group definition is composed of:
  - role identifiers (from, to)
  - type (type) with one of the following values: authority, communication, acquaintance
  - a scope (scope)
  - and validity to subgroups (extends-subgroups)

Example

```
<link from="coach"
      to="player"
      type="authority"
      scope="inter-group"
      extends-subgroups="true" />
```

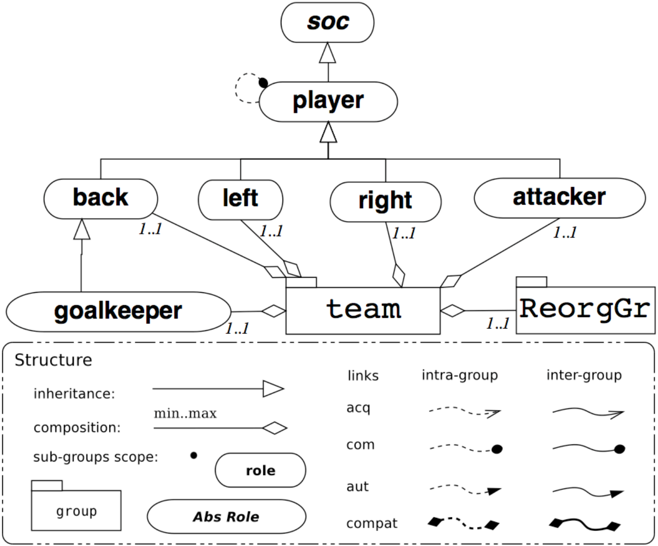
Formation constraint specification

- Formation constraints definition (formation-constraints tag) in a group definition is composed of:
  - compatiblity constraints (compatibility tag) between roles (from, to), with a scope, extends-subgroups and directions (bi-dir)

Example

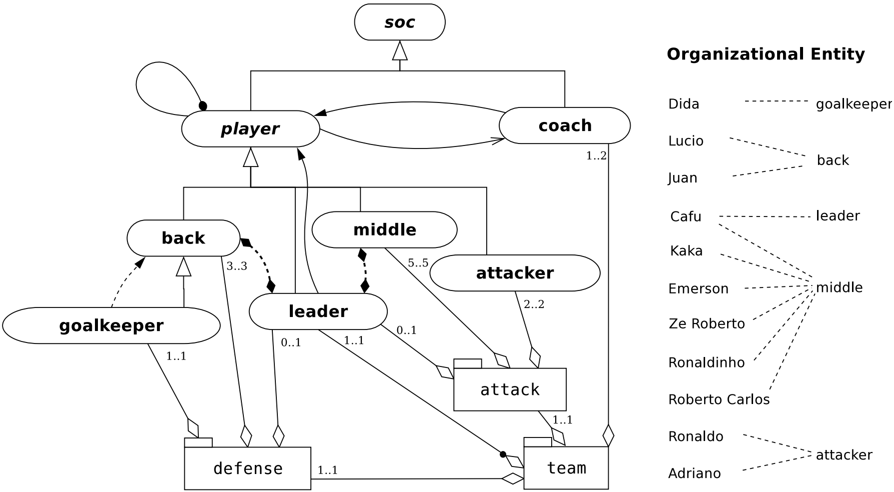
```
<formation-constraints>
  <compatibility from="middle"
                 to="leader"
                 scope="intra-group"
                 extends-subgroups="false"
                 bi-dir="true"/>
  ...
</formation-constraints>
```

Structural specification example (1)



Graphical representation of structural specification of Joj Team

Structural specification example (2)



Graphical representation of structural specification of 3-5-2 Joj Team

Functional Specification

- Specifies the expected behaviour of an MAS in terms of goals along two levels:
  - Collective with Scheme
  - Individual with Mission
- Components:
  - Goals:
    - Performance goal (default type). Goals of this type should be declared as done by the agents committed to them, when realized
    - Achievement goal. Goals of this type should be declared as satisfied by the agents committed to them, when realized
    - Maintenance goal. Goals of this type are not realized at a precise moment but are pursued while the scheme is running.
      The agents committed to them do not need to declare that they are satisfied
  - Scheme: global goal decomposition tree assigned to a group
    - Any scheme has a root goal that is decomposed into subgoals
  - Missions: set of coherent goals assigned to roles within norms

Functional specification

- Defined with the tag functional-specification in the context of an organisational-specification
- Specification in sequence of the different schemes participating to the expected behaviour of the organisation

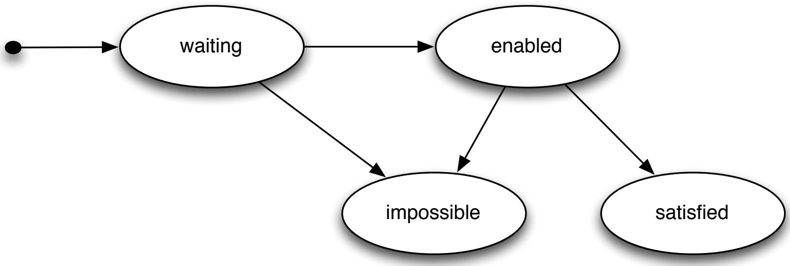
Example

```
<functional-specification>
    <scheme id="sideAttack" >
        <goal id="dogoal" > ... </goal>
        <mission id="m1" min="1" max="5">
           ...
        </mission>
        ...
    </scheme>
    ...
</functional-specification>
```

Scheme specification

- Scheme definition (scheme tag) is composed of:
  - identifier of the scheme (id attribute of scheme tag)
  - the root goal of the scheme with the plan aiming at achieving it (goal tag)
  - the set of missions structuring the scheme (mission tag)
- Goal definition within a scheme (goal tag) is composed of:
  - an idenfier (id attribute of goal tag)
  - a type (performance default, achievement or maintenance)
  - min. number of agents that must satisfy it (min) (default is "all")
  - optionally, an argument (argument tag) that must be assigned to a value when the scheme is created
  - optionally a plan
- Plan definition attached to a goal (plan tag) is composed of
  - one and only one operator (operator attribute of plan tag) with sequence, choice, parallel as possible values
  - set of goal definitions (goal tag) concerned by the operator

Goal States from the Organization Point of View



waiting initial state

enabled goal pre-conditions are satisfied & scheme is well-formed

satisfied agents committed to the goal have achieved it

impossible the goal is impossible to be satisfied

Note: goal state from the Organization point of view may be different of the goal state from the Agent point of view

Scheme specification example

```
<scheme id="sideAttack">
 <goal id="scoreGoal" min="1" >
  <plan operator="sequence">
    <goal id="g1" min="1" ds="get the ball" />
    <goal id="g2" min="3" ds="to be well placed">
      <plan operator="parallel">
        <goal id="g7" min="1" ds="go toward the opponent's field" />
        <goal id="g8" min="1" ds="be placed in the middle field" />
        <goal id="g9" min="1" ds="be placed in the opponent's goal area" />
      </plan>
    </goal>
    <goal id="g3" min="1" ds="kick the ball to the m2Ag" >
       <argument id="M2Ag" />
    </goal>
    <goal id="g4"       min="1" ds="go to the opponent's back line" />
    <goal id="g5"       min="1" ds="kick the ball to the goal area" />
    <goal id="g6"       min="1" ds="shot at the opponent's goal" />
  </plan>
 </goal>
 ...
```

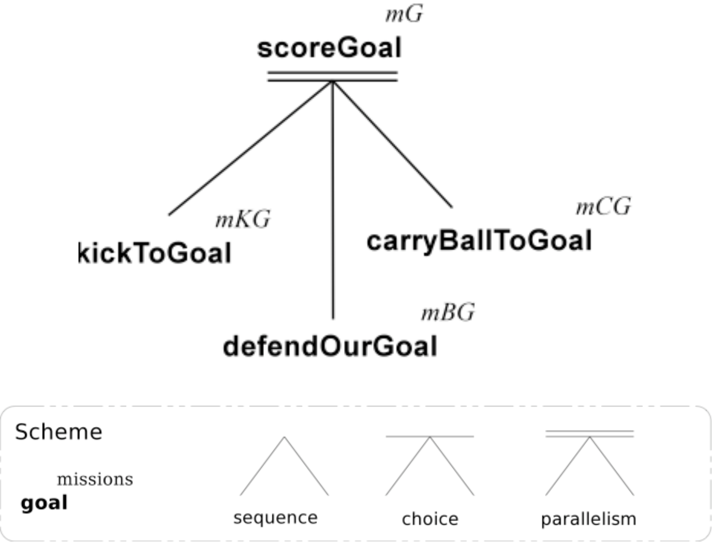
Mission specification

- Mission definition (mission tag) in the context of a scheme definition, is composed of:
  - identifier of the mission (id attribute of mission tag)
  - cardinality of the mission min (0 is default), max (unlimited is default) specifying the number of agents that can be committed to the mission
  - the set of goal identifiers (goal tag) that belong to the mission

Example

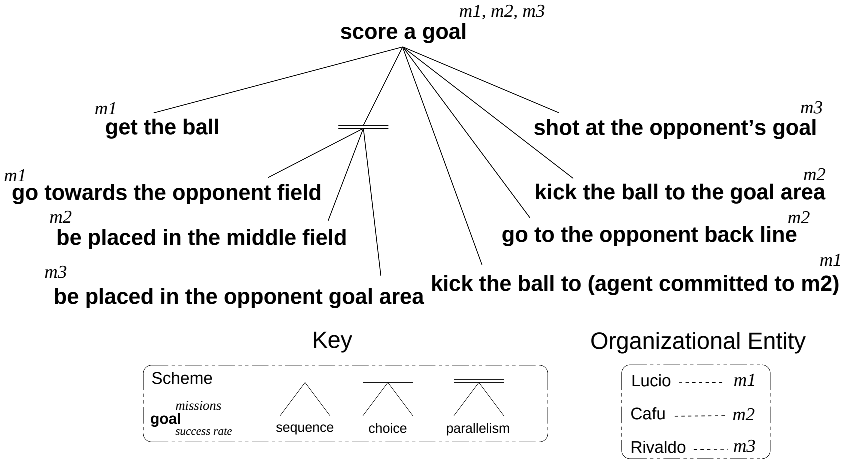
```
<scheme id="sideAttack">
  ... the goals ...
 <mission id="m1" min="1" max="1">
    <goal id="scoreGoal" /> <goal id="g1" />
    <goal id="g3" /> ...
  </mission>
  ...
</scheme>
```

Functional specification example (1)



Graphical representation of social scheme for joj team

Functional specification example (2)



Graphical representation of social scheme "side_attack" for joj team

Normative Specification

- Explicit relation between the functional and structural specifications
- Permissions and obligations to commit to missions in the context of a role
- The normative specification makes explicit the normative dimension of a role

Normative specification

- Defined in-between the tag normative-specification in the context of an organisational-specification
- Definition in sequence of the different norms participating to the governance of the organisation
- Definition of programs written in Normative Programming Language (NPL)

Example

```
<normative-specification>
    <norm id="n1" ... />
     ...
    <norm id="..." ... />
    <npl-norms>
    ...
    </npl-norms>
</normative-specification>
```

Norm Definition

- Norm definition with norm tag, in the context of a normative-specification definition, with attributes:
  - the identifier of the norm (id)
  - the type of the norm (type) with obligation, permission as possible values
  - a condition of activation (condition) – optional – checking:
    - properties of the organisation (e.g. #role_compatibility, #mission_cardinality, #role_cardinality, #goal_non_compliance)

    ⇝ unregimentation of organisation properties !!!
    - (un)fulfillment of an obligation stated in a particular norm (unfulfilled, fulfilled)
  - the role identifier (role) on which the norm is applied
  - the mission identifier (mission) object of the norm
  - a time constraint (time-constraint) – optional –

Norm Definition – example

- Any agent playing *back* is *obliged* to commit to mission *m*1 and achieve its goals within 1 minute

```
<norm id = "n1" type="obligation"
      role="back" mission="m1" time-constraint="1 minute"/>
```

- Any agent playing *left* is *obliged* to commit to mission *m*2 and achieve its goals within 1 day

```
<norm id = "n2" type="obligation"
      role="left" mission="m2" time-constraint="1 day"/>
```

- Any agent playing *coach* is *obliged* to commit to mission *ms* and achieve its goals within 3 hour in case obligation of norm n2 has not been fulfilled

```
<norm id = "n4" type="obligation"
      condition="unfulfilled(obligation(_,n2,_,_))"
      role="coach" mission="ms" time-constraint="3 hour"/>
```

Normative Programming Language (NPL)

Norms written in NPL have:

- an activation condition
- a consequence
  Two kinds of consequences are considered
  - regimentations (`fail`)
  - obligations (`obligation`)
- terms starting with an upper case letter are variables

Example (Norm)

```
norm n1: plays(A,writer,G) -> fail.
```

or

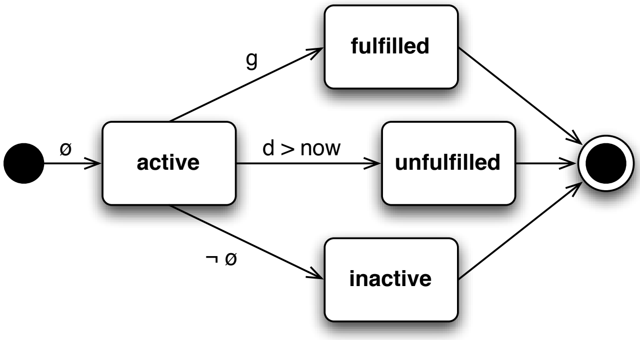
```
norm n1: plays(A,writer,G)
      -> obligation(A,n1,plays(A,editor,G),
         `now + 3 min`).
```

Normative Programming Language (NPL)

Example (NPL Program)

```
<npl-norms>
     a :- t &amp; k.
     norm npl1: a &amp; v(X) ->
          obligation(bob,true,g(X),‘now‘+‘1 day‘).
     norm npl2: a &amp; b    -> fail(test).
</npl-norms>
```

Obligations life cycle



$$norm\ n:\ \phi -> \ obligation(a, r, g, d)$$

- ϕ: activation condition of the norm (e.g. play a role)
- g: the goal of the obligation (e.g. commit to a mission)
- d: the deadline of the obligation

Organisation Entity Dynamics

1. Organisation is created (by the agents)
    - instances of groups
    - instances of schemes
2. Agents enter into groups adopting roles
3. When a group is well formed, it may become responsible for schemes
    - Agents from the group are then obliged to commit to missions in the scheme
4. Agents commit to missions
5. Agents fulfil mission's goals
6. Agents leave schemes and groups
7. Schemes and groups instances are destroyed

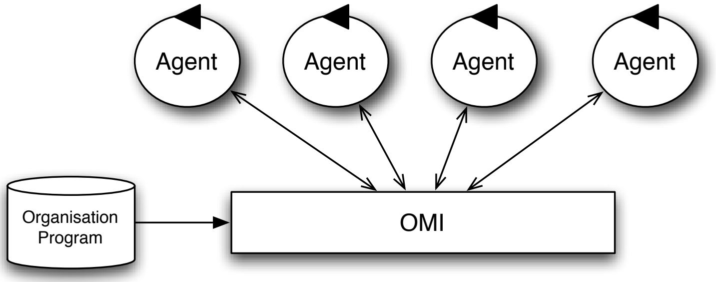
Outline

Programming Agents' Organisations
- Fundamentals
- Existing approaches
- $\mathcal{M}$oise Modeling Language (OML)
- $\mathcal{M}$oise Management Infrastructure (OMI)
- $\mathcal{M}$oise and Environment (O-E)
- $\mathcal{M}$oise and Agents (O-A)
- Conclusions and wrap-up

Organisation management infrastructure (OMI)

Responsibility

- Managing – coordination, regulation – the agents' execution within organisation defined by an organisational specification

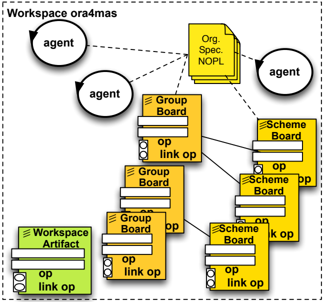


(e.g. MadKit, AMELI, $\mathcal{S}$-$\mathcal{M}$oise$^+$, ...)

ORA4MAS: OMI within JaCaMo

Based on A&A and $\mathcal{M}$oise.
Agents' working environment is instrumented with Organizational Artifacts (OA) offering "organizational" actions
$\leadsto$ Distributed management of the organization with a clear separation of concerns:

- Agents:
  - create, handle OAs and act on them
    $\leadsto$ deploy and manage their OMI
  - perceive the organization state and violations of norms from the OAs
  - decide about:
    - actions on the organization, on norms
    - sanctions to apply
- OAs are in charge of interpreting Normative Programs
  - to detect and evaluate norms compliance
  - or to regiment norms



ORA4MAS– OrgBoard artifact

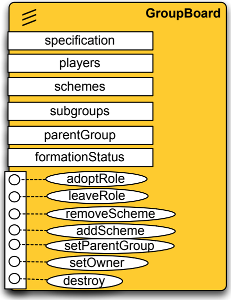
Manages all artifacts of an organisation.

- Observable Properties:
  - group(group_id,group_type,artid): list of the group_id of group_type that exist in the organizational entity
  - scheme(scheme_id,scheme_type,artid): list of the scheme_id of scheme_type that exist in the organizational entity
- Operations:
  - createGroup(group) (resp. removeGroup(grid)): attempts to create (resp. remove) group in the organization
  - createScheme(scheme) (resp. removeScheme(schid)): attempts to create (resp. remove) scheme in the organization

ORA4MAS– GroupBoard artifact

Manages the functioning of an instance of group in the organization.

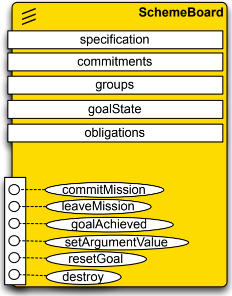
- Observable Properties:
  - specification: group spec. in the OS
  - player: list of play(agent, role, group)
  - schemes: list of scheme identifiers that the group is responsible for
  - subgroups, parentGroup, formationStatus (if the group is well formed or not)
- Operations:
  - adoptRole(role) (resp. leaveRole(role)): attempts to adopt (resp. leave) role in the group
  - addScheme(schid) (resp. removeScheme(schid)): attempts to set (resp. unset) the group responsible for the scheme managed by the SchemeBoard schId
  - setParentGroup(groupid), setOwner(agtid), destroy



ORA4MAS– SchemeBoard artifact

Manages the functioning of an instance of social scheme in the organization.

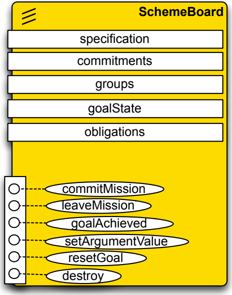
- Observable Properties:
  - specification: scheme spec. in the OS
  - commitments: list of commitment(agent, mission, scheme)
  - groups: list of groups resp. for the scheme
  - goalState: list of goals' current state
  - goalArgument(schemeId,goalId,argId,value): added only if the argument has a value, usually defined by the operation setArgumentValue
  - obligations: list of active obligations in the scheme (obligation(agt,norm,goal,deadline))
  - permissions: list of active permissions in the scheme (permission(agt,norm,goal,deadline))
  - goalArgument: value of goals' arguments, defined by the operation setArgumentValue



ORA4MAS– SchemeBoard artifact (Contd)

Manages the functioning of an instance of social scheme in the organization.

- Operations:
  - commitMission(mission) (resp. leaveMission): attempts to "commit" (resp "leave") a mission in the scheme
  - goalAchieved(goal): declares that goal is achieved
  - setArgumentValue(goal, argument, value): defines the value of goal's argument
  - resetGoal(goal) (reset the status of a goal), destroy



admCommand in Scheme/Group Boards

```
// in some plan of some agent
admCommand(setCardinality(role,editor,0,10));
admCommand(setCardinality(role,writer,0,20));

lookupArtifact("s1", SId); // get artifact id of scheme "s1"
admCommand(setCardinality(mission,mColaborator,0,3))[aid(SId)];
admCommand(setCardinality(mission,mManager,0,2))[aid(SId)];
```

Only the owner of the group/scheme can perform admCommands

ORA4MAS– NormativeBoard artifact

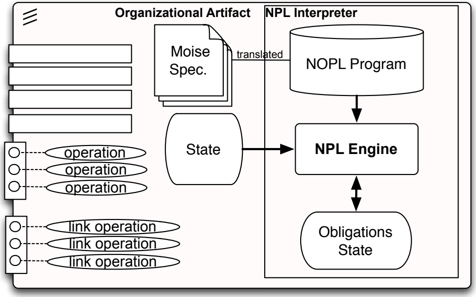
- It can be loaded with any NPL program
- is used to manage obligations/permissions defined in the normative specification
- When a group becomes responsible for a scheme, an instance of this artifact is created automatically.

- Observable Properties:
  - obligation: current active obligations
- Operations:
  - load(nplprogram)
  - addFact (resp. removeFact)

Organisational Artifact Architecture

Org. Artifacts managing groups and social schemes execution:

- interpret programs written in Normative Programming Language (NPL) [?] coming from the automatic translation of $\mathcal{M}$oise programs
- generate signals
  - oblCreated(o), oblFulfilled(o), oblUnfulfilled(o)
  - oblInactive(o), normFailure(f)
    (o = obligation(to whom, reason, what, deadline))



Generic control cycle of an Organisational Artifact

```
// oe: current state of the org. managed by the artifact
// p: current NOPL program
// npi: NPL interpreter
When operation o is triggered by agent a do
 oe' <- oe \\ creates a ``backup'' of current oe
 oe <- executes(o,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 observable properties from obligations state
   success operation o
```

Structural Operational Semantics

A normative system configuration is a tuple: $\langle F, N, ns, OS, t\rangle$
with

- F is a set of facts
- N is a set of norms
- ns is the state of the normative system (sound state $\top$ or a failure state $\bot$)
- OS is a set of obligations
  each element $os \in OS$ is $\langle o, ost\rangle$
  where o obligation and ost its state
- t is the current time

The initial configuration of a NP P is $\langle P_F, P_N, \top, \emptyset, 0\rangle$

- P_F and P_N are the initial facts and norms defined in the normative program P

Rules for Norm Management

- Failure detection:

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

when any norm n becomes active (i.e., its condition component holds in the current state) and its consequence is $\texttt{fail}(\_)$, the normative state is no longer sound but in failure ($\bot$).

- Roll back from failure:

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

Rules for Norm Management (continued)

- Creation of obligation:

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

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

Rules for Obligation Management

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

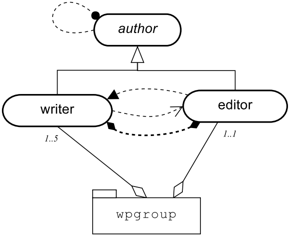
$$\frac{os \in OS \qquad os = \langle o, \textbf{active} \rangle \qquad o_d < t}{\langle F, N, \top, OS, t \rangle \longrightarrow \langle F, N, \top, (OS \setminus \{os\}) \cup \{\langle o, \textbf{unfulfilled} \rangle\}, t \rangle} \quad (\textbf{Unfulfil})$$

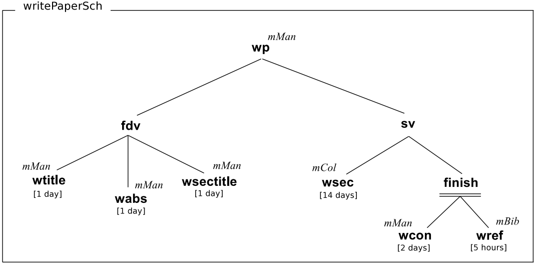
$$\frac{os \in OS \qquad os = \langle o, \textbf{active} \rangle \qquad F \not\models o_r}{\langle F, N, \top, OS, t \rangle \longrightarrow \langle F, N, \top, (OS \setminus \{os\}) \cup \{\langle o, \textbf{inactive} \rangle\}, t \rangle} \quad (\textbf{Inactive})$$

NOPL

Normative Organisation Programming Language

- NOPL is a particular class of NPL: facts, rules and norms are specific to a OML (eg. $\mathcal{M}$oise NOML):





id	condition	role	type	mission	TTF
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	#gnc	editor	obl	*ms*	3 hours
n7	#rc	editor	obl	*ms*	30 minutes
n6	#mc	editor	obl	*ms*	1 hour
…	…	…	…	…	…

#gnc = goal_non_compliance
#rc = role_compatibility
#mc = mission_cardinality

OS in $\mathcal{M}$oise OML to NOPL translation

Example (role cardinality norm – regimentation)

```
group_role(writer,1,5).

norm ncar: group_role(R,_,M) &
           rplayers(R,G,V) & V > M
  -> fail(role_cardinality(R,G,V,M)).
```

Example (role cardinality norm – agent decision)

```
norm ncar: group_role(R,_,M) &
           rplayers(R,G,V) & V > M &
           plays(E,editor,G)
  -> obligation(E,ncar,committed(E,ms,_),
                `now + 1 hour`).
```

$\mathcal{M}$oise Social scheme — NOPL — Facts

- Static facts:
  - scheme_mission(*m*,*max*,*min*): cardinality of mission *m*;
  - goal(*m*,*g*,*pre-cond*,'*ttf*'): mission, preconditions and TTF for goal *g*.
- Dynamic facts (provided at run-time by the organisational artifact in charge of the management of the social 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 the scheme instance *s*.
  - committed(*a*,*m*,*s*): the agent *a* is committed to mission *m* in scheme *s*.
  - achieved(*s*,*g*,*a*): the goal *g* has been achieved in the scheme *s* by the agent *a*.

$\mathcal{M}$oise Social scheme — NOPL — Rules

- Example of rules used to infer the state of the scheme:
  - Number of players of mission M in scheme S:

```
mplayers(M,S,V) :-
   .count(committed(_,M,S),V).
```

  - Wellformedness property of 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.
```

  - Readyness of goal G in scheme S (i.e. goal is ready to be 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).
```

*M*oise Social scheme — NOPL — Norms

Norms for goals

- Agents are obliged to achieve 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).
```

Norms for properties

- Mission cardinality as regimentation

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

- Mission cardinality as obligation

```
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‘).
```

$\mathcal{M}$oise — NOPL — Norms

⇝ Definition of similar kinds of facts, rules and norms for the groups, roles in the structural specification

- Domain norms:
  - Each norm in the normative specification of the OS has a corresponding norm in the NOP
  - Since in the OS, obligations refer to roles and missions, norms in corresponding NOP identify the agents playing the role in groups responsible for the scheme and take into account the property conditions.

```
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`).
```

Partial Synthesis

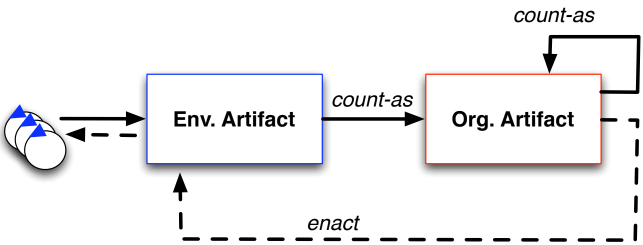
- NPL, based on obligation and regimentation, formalised using operational semantics, specialised into NOPL
- Automatic translation of OS written in $\mathcal{M}$oise OML into several NOPs
- Implementation in ORA4MAS, artifact-based OMI: Organisational Artifacts act as interpreters of NOPs.
  - NOPL (80%): dynamic of obligations (several aspects of the $\mathcal{M}$oise OS have been translated to norms)
  - CArtAgO (10%): interface for agents
  - Java (10%): dynamic of organisational state

Outline

Programming Agents' Organisations
- Fundamentals
- Existing approaches
- $\mathcal{M}$oise Modeling Language (OML)
- $\mathcal{M}$oise Management Infrastructure (OMI)
- $\mathcal{M}$oise and Environment (O-E)
- $\mathcal{M}$oise and Agents (O-A)
- Conclusions and wrap-up

Environment integration

- Organisational Artifacts enable organisation and environment integration
- Embodied organisation [Piunti et al., 2009]



status: ongoing work

Constitutive rules

Count-As rule

An event occurring on an artifact, in a particular context, may "count-as" an institutional event

- transforms the events created in the working environment into activation of an organisational operation

$\rightsquigarrow$ indirect automatic updating of the organisation

Enact rule

An event produced on an organisational artifact, in a specific institutional context, may "enact" change and updating of the working environment (i.e., to promote equilibrium, avoid undesiderable states)

- Installing automated control on the working environment
- Even without the intervention of organisational/staff agents (regimenting actions on physical artifacts, enforcing sanctions, ...)

Outline

Programming Agents' Organisations

- Fundamentals
- Existing approaches
- $\mathcal{M}$oise Modeling Language (OML)
- $\mathcal{M}$oise Management Infrastructure (OMI)
- $\mathcal{M}$oise and Environment (O-E)
- $\mathcal{M}$oise and Agents (O-A)
- Conclusions and wrap-up

Agent integration

- Agents can interact with organisational artifacts as with ordinary artifacts by perception and action

⇝ Any Agent Programming Language integrated with CArtAgO can use organisational artifacts

Agent integration provides some “internal” tools for the agents to simplify their interaction with the organisation:

- maintenance of a local copy of the organisational state
- production of organisational events
- provision of organisational actions

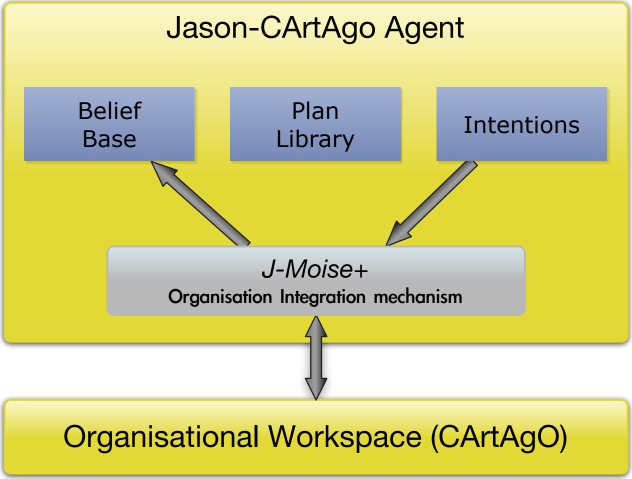
$\mathcal{J}$-$\mathcal{M}$oise: *Jason* + $\mathcal{M}$oise

- Agents are programmed with *Jason*

$\leadsto$ BDI agents (reactive planning) – suitable abstraction level

- The programmer has the possibility to express sophisticated recipes for adopting roles, committing to missions, fulfilling/violating norms, ...
- Organisational information is made accessible in the mental state of the agent as beliefs
- Integration is totally independent of the distribution/communication layer

$\mathcal{J}$-$\mathcal{M}$oise: *Jason* + $\mathcal{M}$oise– General view



Organisational **actions** in *Jason* I

Example (GroupBoard)

```
...
joinWorkspace("ora4mas",O4MWsp);
makeArtifact(
    "auction",
    "ora4mas.nopl.GroupBoard",
    ["auction-os.xml", auctionGroup, false, true ],
    GrArtId);
adoptRole(auctioneer);
focus(GrArtId);
...
```

Organisational **actions** in *Jason* II

Example (SchemeBoard)

```
...
makeArtifact(
   "sch1",
   "ora4mas.nopl.SchemeBoard",
   ["auction-os.xml", doAuction, false, true ],
   SchArtId);
focus(SchArtId);
addScheme(Sch);
commitMission(mAuctioneer)[artifact_id(SchArtId)];
...
```

Organisational **actions** in *Jason* III

- For roles:
  - `adoptRole`
  - `leaveRole`
- For missions:
  - `commitMission`
  - `leaveMission`

- Those actions usually are executed under regimentation (to avoid an inconsistent organisational state)
  e.g. the adoption of role is constrained by
  - the cardinality of the role in the group
  - the compatibilities of the roles played by the agent

Organisational perception

When an agent focus on an Organisational Artifact, the observable properties (Java objects) are translated to beliefs with the following predicates:

- specification
- schemeSpecification
- play(agent, role, group)
- commitment(agent, mission, scheme)
- goalState(scheme, goal, list of committed agents, list of agent that achieved the goal, state of the goal)
- obligation(agent,norm,goal,dead line)
- normFailure(norm)

Organisational perception – example

Inspection of agent **bob** (cycle #0)

- Beliefs	commitment(bob,mManager,"sch2")$_{\text{[artifact\_id(cobj\_4),c}}$
	cept),artifact_name(cobj_4,"sch2"),artifact_type(cobj_4,"ora4m
	commitment(bob,mManager,"sch1")$_{\text{[artifact\_id(cobj\_3),c}}$
	cept),artifact_name(cobj_3,"sch1"),artifact_type(cobj_3,"ora4m
	current_wsp(cobj_1,"ora4mas","308b05b0-2994-4fe8
	formationStatus(ok)$_{\text{[artifact\_id(cobj\_2),obs\_prop\_id("obs\_i}}$
	obj_2,"mypaper"),artifact_type(cobj_2,"ora4mas.nopl.GroupBo
	goalState("sch2",wp,[bob],[bob],satisfied)$_{\text{[artifact\_id(cob}}$

Handling organisational **events** in *Jason*

Whenever something changes in the organisation, the agent architecture updates the agent belief base accordingly producing events (belief update from perception)

Example (new agent entered the group)

```
+play(Ag,boss,GId) <- .send(Ag,tell,hello).
```

Example (change in goal state)

```
+goalState(Scheme,wsecs,_,_,satisfied)
    : .my_name(Me) & commitment(Me,mCol,Scheme)
   <- leave_mission(mColaborator,Scheme).
```

Example (signals)

```
+normFailure(N) <- .print("norm failure event: ", N).
```

Typical plans for obligations

Example

```
+obligation(Ag,Norm,committed(Ag,Mission,Scheme),DeadLine)
     : .my_name(Ag)
   <- .print("I am obliged to commit to ",Mission);
      commit_mission(Mission,Scheme).

+obligation(Ag,Norm,achieved(Sch,Goal,Ag),DeadLine)
     : .my_name(Ag)
   <- .print("I am obliged to achieve goal ",Goal);
      !Goal[scheme(Sch)];
      goal_achieved(Goal,Sch).

+obligation(Ag,Norm,What,DeadLine)
    : .my_name(Ag)
   <- .print("I am obliged to ",What,
             ", but I don't know what to do!").
```

Writing paper example

Organisation Specification

```
<organisational-specification
  <structural-specification>
     <role-definitions>
        <role id="author" />
        <role id="writer"> <extends role="author"/> </role>
        <role id="editor"> <extends role="author"/> </role>
     </role-definitions>

     <group-specification id="wpgroup">
        <roles>
           <role id="writer" min="1" max="5" />
           <role id="editor" min="1" max="1" />
        </roles>
        ...
```

Writing paper sample I

Execution

jaime	action: jmoise.create_group(wpgroup)
all	perception: group(wpgroup,g1)[owner(jaime)]
jaime	action: jmoise.adopt_role(editor,g1)
olivier	action: jmoise.adopt_role(writer,g1)
jomi	action: jmoise.adopt_role(writer,g1)
all	perception: play(jaime,editor,g1) play(olivier,writer,g1) play(jomi,writer,g1)

Writing paper sample II

Execution

jaime	action: jmoise.create_scheme(writePaperSch, [g1])
all	perception: scheme(writePaperSch,s1)[owner(jaime)]
all	perception: scheme_group(s1,g1)
jaime	perception: permission(s1,mManager)[role(editor),group(wpgroup)]
jaime	action: jmoise.commit_mission(mManager,s1)
olivier	perception: obligation(s1,mColaborator)[role(writer),group(wpgroup), obligation(s1,mBib)[role(writer),group(wpgroup)
olivier	action: jmoise.commit_mission(mColaborator,s1)
olivier	action: jmoise.commit_mission(mBib,s1)
jomi	perception: obligation(s1,mColaborator)[role(writer),group(wpgroup), obligation(s1,mBib)[role(writer),group(wpgroup)]
jomi	action: jmoise.commit_mission(mColaborator,s1)

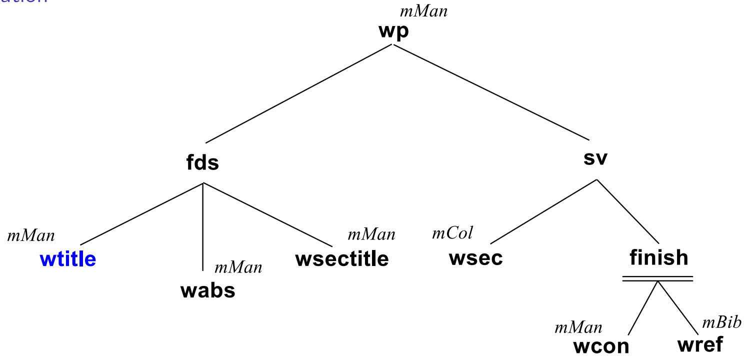
Writing paper sample III

Execution

all perception:
commitment(jaime,mManager,s1)
commitment(olivier,mColaborator,s1)
commitment(olivier,mBib,s1)
commitment(jomi,mColaborator,s1)

Writing paper sample IV

Execution

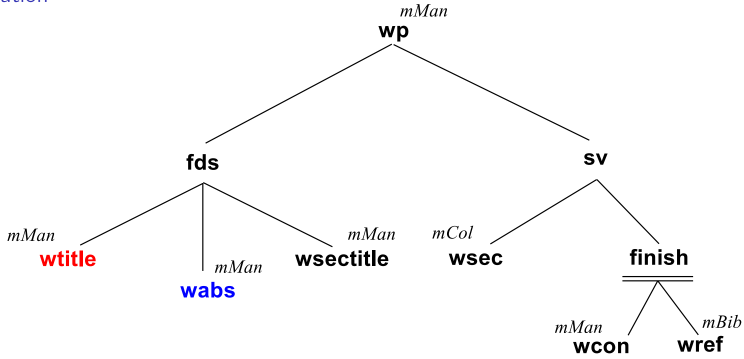


all perception: goal_state(s1,*,unsatisfied)

jaime (only wtitle is possible, Jaime should work)
event: +!wtitle
action: jmoise.set_goal_state(s1,wtitle,satisfied)

Writing paper sample V

Execution

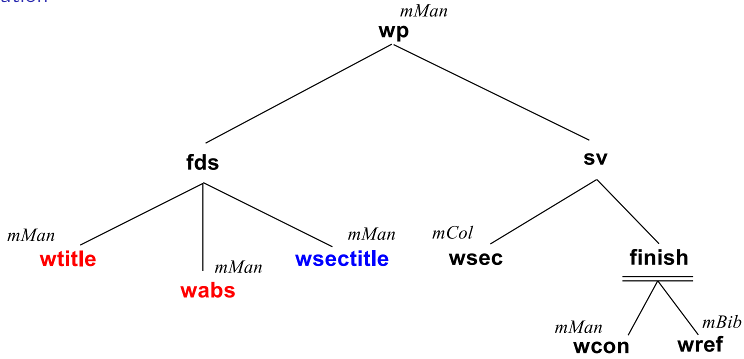


jaime event: +!wabs

action: jmoise.set_goal_state(s1,wabs,satisfied)

Writing paper sample VI

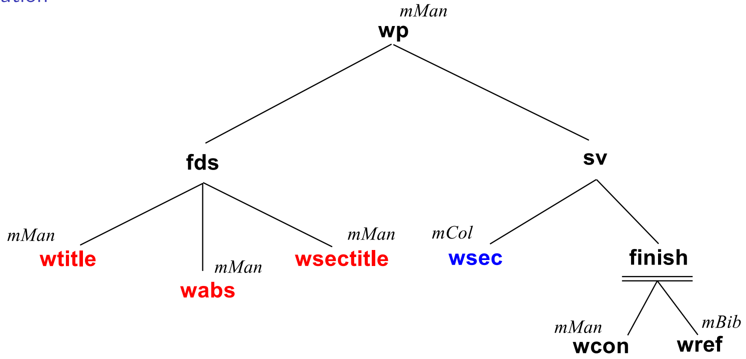
Execution



jaime event: +!wsectitles
action: jmoise.set_goal_state(s1,wsectitles,satisfied)

Writing paper sample VII

Execution

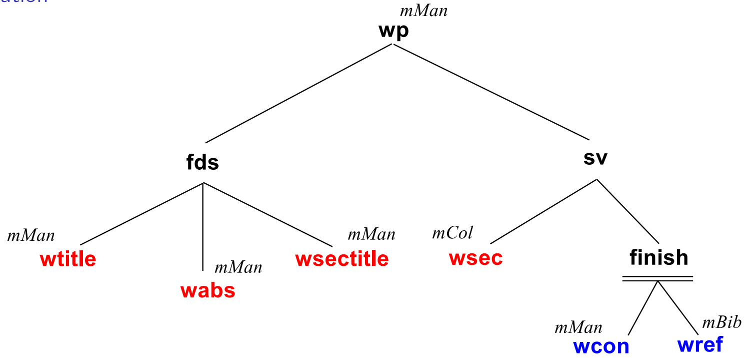


olivier, jomi event: +!wsecs

action: jmoise.set_goal_state(s1,wsecs,satisfied)

Writing paper sample VIII

Execution



jaime event: +!wcon; …

olivier event: +!wref; …

Writing paper sample IX

Execution

all	action: jmoise.remove_mission(s1)
jaime	action: jmoise.jmoise.remove_scheme(s1)

Useful tools — Mind inspector

play(gaucho1,herder,gr_herding_grp_13)[source(orgManager)].
play(gaucho4,herdboy,gr_herding_grp_13)[source(orgManager)].
play(gaucho5,herdboy,gr_herding_grp_13)[source(orgManager)].
pos(*45*,*44*,*128*)[source(percept)].
scheme(herd_sch,sch_herd_sch_18)[owner(gaucho3),source(orgManager)].
scheme(herd_sch,sch_herd_sch_12)[owner(gaucho1),source(orgManager)].
scheme_group(sch_herd_sch_12,gr_herding_grp_13)[source(orgManager)].
steps(*700*)[source(self)].
target(*6*,*44*)[source(gaucho1)].

- Rules

random_pos(X,Y) :-
    (pos(AgX,AgY,_418) & (jia.random(RX,*40*) & ((RX > *5*) & ((X = ((RX-*20*)+AgX)) & ((X >

- Intentions

Sel	Id	Pen	Intended Means Stack (hide details)
	16927	**suspended-self**	+!be_in_formation[scheme(sch_herd_sch_12),mission(hel **+!be_in_formation[scheme(Sch),mission(Mission)]**

Outline

Programming Agents' Organisations

- Fundamentals
- Existing approaches
- $\mathcal{M}$oise Modeling Language (OML)
- $\mathcal{M}$oise Management Infrastructure (OMI)
- $\mathcal{M}$oise and Environment (O-E)
- $\mathcal{M}$oise and Agents (O-A)
- Conclusions and wrap-up

Wrap-up

- Model to specify global orchestration

⇝ team strategy defined at a high level
- Ensures that the agents follow some of the constraints specified for the organisation
- Helps the agents to work together
- The organisation is interpreted at runtime, it is not hardwired in the agents code
- The agents 'handle' the organisation (i.e. their artifacts)
- It is suitable for open systems as no specific agent architecture is required
- Organization can easily be changed by the developers or by the agents themselves

- All available as open source at

`http://moise.souceforge.net`

Multi-Agent Oriented Programming

The JaCaMo Platform

O. Boissier[1] R.H. Bordini[2] J.F. Hübner[3] A. Ricci[4]

1. Mines Saint-Etienne (ENSMSE), Saint Etienne, France

2 Pontificia Universidade Catolica do Rio Grande do Sul (PUCRS), Porto Alegre, Brazil

3. Federal University of Santa Catarina (UFSC), Florianópolis, Brazil

4. University of Bologna (UNIBO), Bologna, Italy

November 2017

Bibliography I

Bernoux, P. (1985).
La sociologie des organisations.
Seuil, 3ème edition.

Boella, G., Torre, L., and Verhagen, H. (2008).
Introduction to the special issue on normative multiagent systems.
Autonomous Agents and Multi-Agent Systems, 17(1):1–10.

Carabelea, C. (2007).
Reasoning about autonomy in open multi-agent systems - an approach based on the social power theory.
in french, ENS Mines Saint-Etienne.

de Brito, M., Hübner, J. F., and Bordini, R. H. (2012).
Programming institutional facts in multi-agent systems.
In *COIN-12, Proceedings*.

Bibliography II

Esteva, M., Rodriguez-Aguiar, J. A., Sierra, C., Garcia, P., and Arcos, J. L. (2001).

On the formal specification of electronic institutions.

In Dignum, F. and Sierra, C., editors, *Proceedings of the Agent-mediated Electronic Commerce*, LNAI 1191, pages 126–147, Berlin. Springer.

Esteva, M., Rodríguez-Aguilar, J. A., Rosell, B., and Arcos, J. L. (2004).

AMELI: An agent-based middleware for electronic institutions.

In Jennings, N. R., Sierra, C., Sonenberg, L., and Tambe, M., editors, *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'2004)*, pages 236–243, New York. ACM.

Ferber, J. and Gutknecht, O. (1998).

A meta-model for the analysis and design of organizations in multi-agents systems.

In Demazeau, Y., editor, *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS'98)*, pages 128–135. IEEE Press.

Bibliography III

Gasser, L. (2001).
Organizations in multi-agent systems.
In *Pre-Proceeding of the 10th European Worshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW'2001)*, Annecy.

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.

Gutknecht, O. and Ferber, J. (2000).
The MadKit agent platform architecture.
In *Agents Workshop on Infrastructure for Multi-Agent Systems*, pages 48–55.

Hannoun, M., Boissier, O., Sichman, J. S., and Sayettat, C. (2000).
Moise: An organizational model for multi-agent systems.
In Monard, M. C. and Sichman, J. S., editors, *Proceedings of the International Joint Conference, 7th Ibero-American Conference on AI, 15th Brazilian Symposium on AI (IBERAMIA/SBIA'2000), Atibaia, SP, Brazil, November 2000*, LNAI 1952, pages 152–161, Berlin. Springer.

Bibliography IV

Hübner, J. F., Boissier, O., Kitio, R., and Ricci, A. (2009).
Instrumenting Multi-Agent Organisations with Organisational Artifacts and Agents.
Journal of Autonomous Agents and Multi-Agent Systems.

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.

Hübner, J. F., Sichman, J. S., and Boissier, O. (2006).
S-MOISE+: A middleware for developing organised multi-agent systems.
In Boissier, O., Dignum, V., Matson, E., and Sichman, J. S., editors, *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, volume 3913 of *LNCS*, pages 64–78. Springer.

Bibliography V

Hübner, J. F., Sichman, J. S., and Boissier, O. (2007).
Developing Organised Multi-Agent Systems Using the MOISE+ Model: Programming Issues at the System and Agent Levels.
Agent-Oriented Software Engineering, 1(3/4):370–395.

Malone, T. W. (1999).
Tools for inventing organizations: Toward a handbook of organizational process.
Management Science, 45(3):425–443.

Morin, E. (1977).
La méthode (1) : la nature de la nature.
Points Seuil.

Okuyama, F. Y., Bordini, R. H., and da Rocha Costa, A. C. (2008).
A distributed normative infrastructure for situated multi-agent organisations.
In Baldoni, M., Son, T. C., van Riemsdijk, M. B., and Winikoff, M., editors, *DALT*, volume 5397 of *Lecture Notes in Computer Science*, pages 29–46. Springer.

Bibliography VI

Ossowski, S. (1999).
Co-ordination in Artificial Agent Societies: Social Structures and Its Implications for Autonomous Problem-Solving Agents, volume 1535 of *LNAI*.
Springer.

Piunti, M., Ricci, A., Boissier, O., and Hubner, J. (2009).
Embodying organisations in multi-agent work environments.
In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2009)*, Milan, Italy.

Pynadath, D. V. and Tambe, M. (2003).
An automated teamwork infrastructure for heterogeneous software agents and humans.
Autonomous Agents and Multi-Agent Systems, 7(1-2):71–100.

Ricci, A., Piunti, M., Viroli, M., and Omicini, A. (2009).
Environment programming in CArtAgO.
In *Multi-Agent Programming: Languages,Platforms and Applications,Vol.2*.
Springer.

Bibliography VII

Tambe, M. (1997).
Towards flexible teamwork.
Journal of Artificial Intelligence Research, 7:83–124.