Multi-Agent Oriented Programming with JaCaMo

an organization-focused approach

Maiquel de Brito

[1]UFSC-Blumenau-Brasil

17th WESAAC

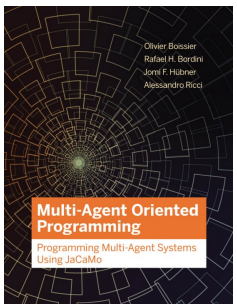
August 2023

Outline

Credits

This presentation is partially based on previous presentations by Andrei Ciortea, Jomi F. Hübner, Luis Gustavo Nardin, Olivier Boissier, and possibly others who may have inspired them before.

Additional information on Multi-Agent Programming with JaCaMo can be found in



[Boissier et al., 2020]

Practical activities are available here

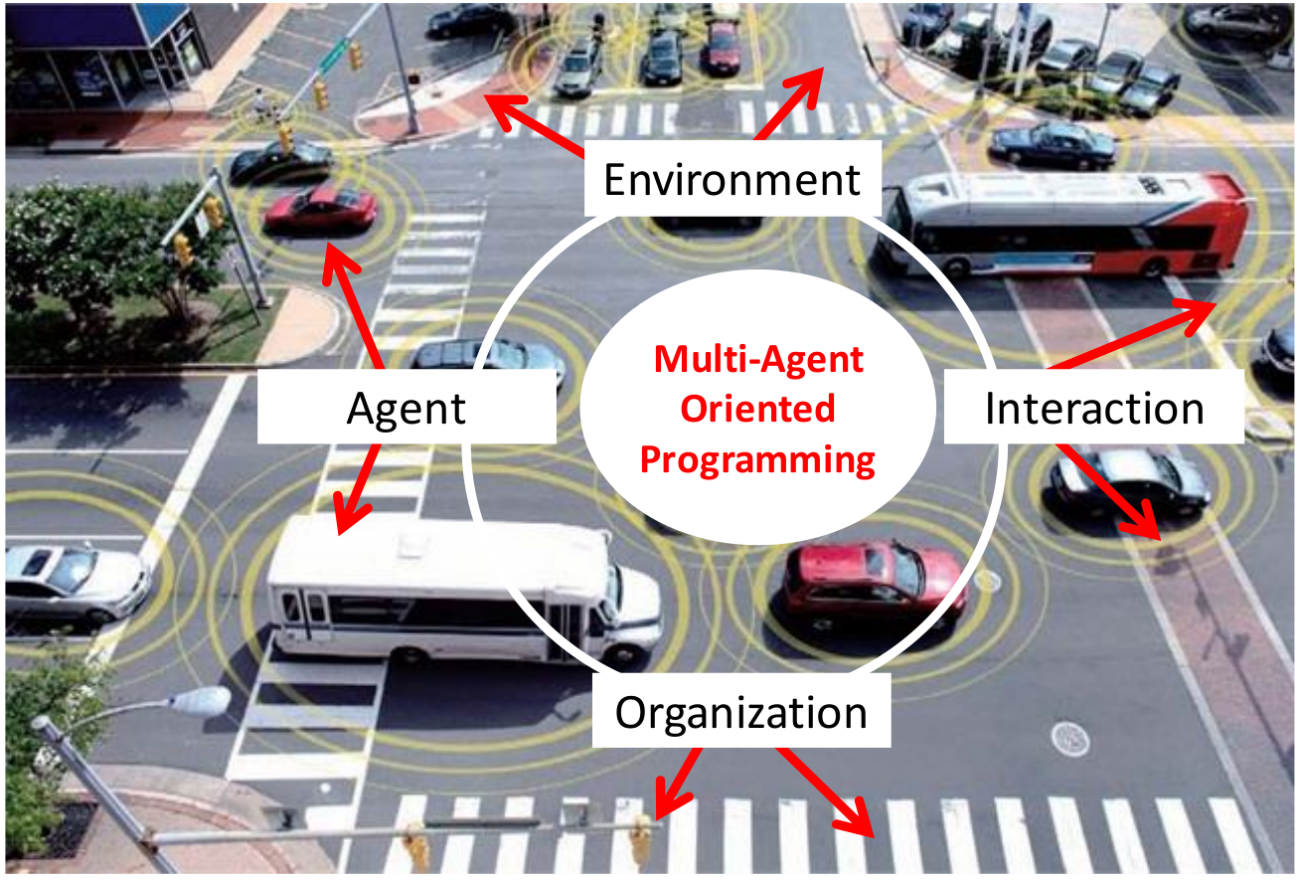
Multi-Agent System

A set of autonomous agents interacting with each other within a shared environment, eventually under one to multiple organizations

- **Agents**: autonomous decision-making entities able to react to events while pursuing (pro-actively defined or delegated) goals and directing actions to achieve them
- **Environment**: shared medium providing the surrounding conditions for agents to exist and act
- **Interaction**: motor of dynamics and interoperability in the MAS
- **Organization**: abstractions to declare and make accessible to agents their collective structure and functioning in a shared environment

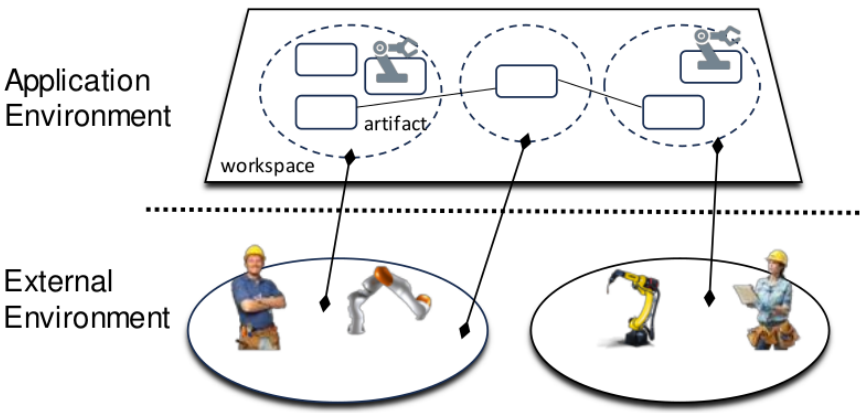
Based on the VOWELS' perspective (Demazeau, 1995)

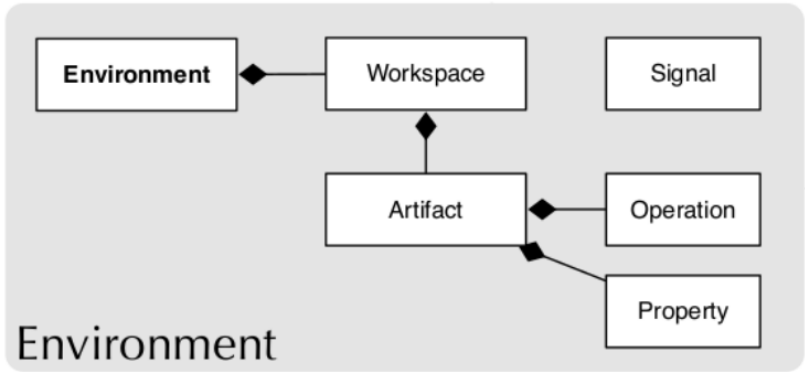
Multi-Agent System



Ciortea, Hübner, Nardin (EASSS'2023)

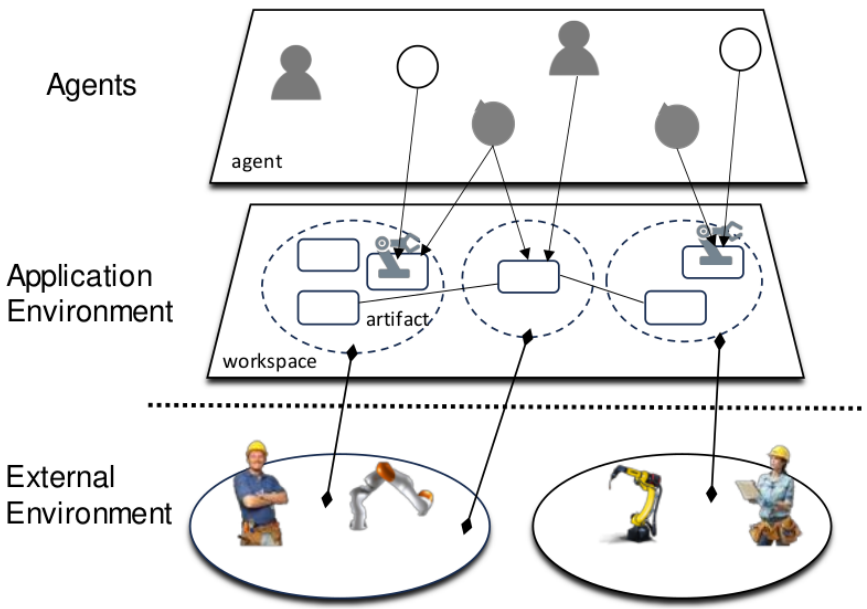
Environment Dimension

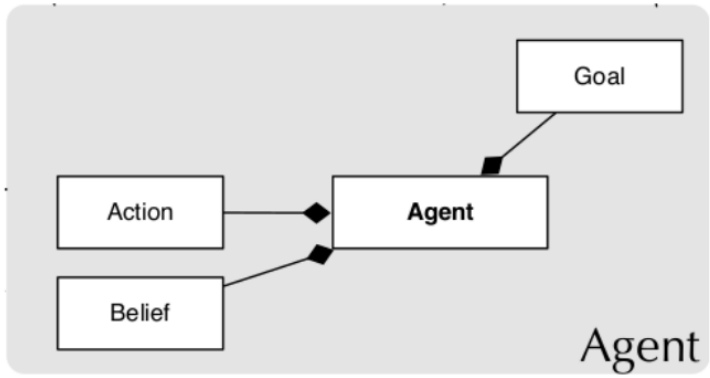




Ciortea, Hübner, Nardin (EASSS'2023)

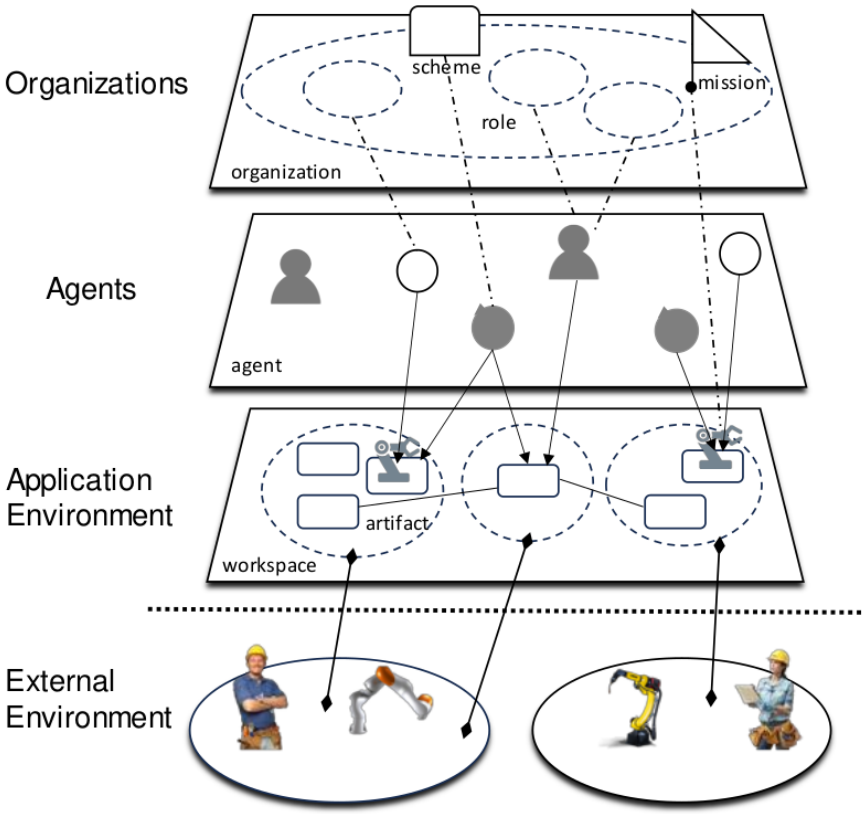
Agent Dimension

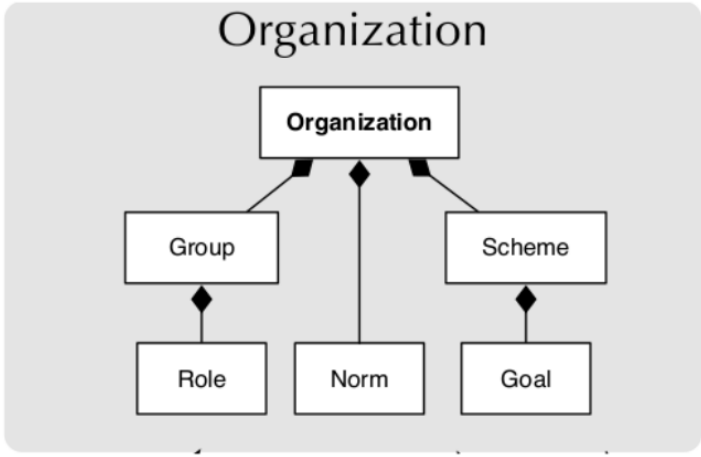




Ciortea, Hübner, Nardin (EASSS'2023)

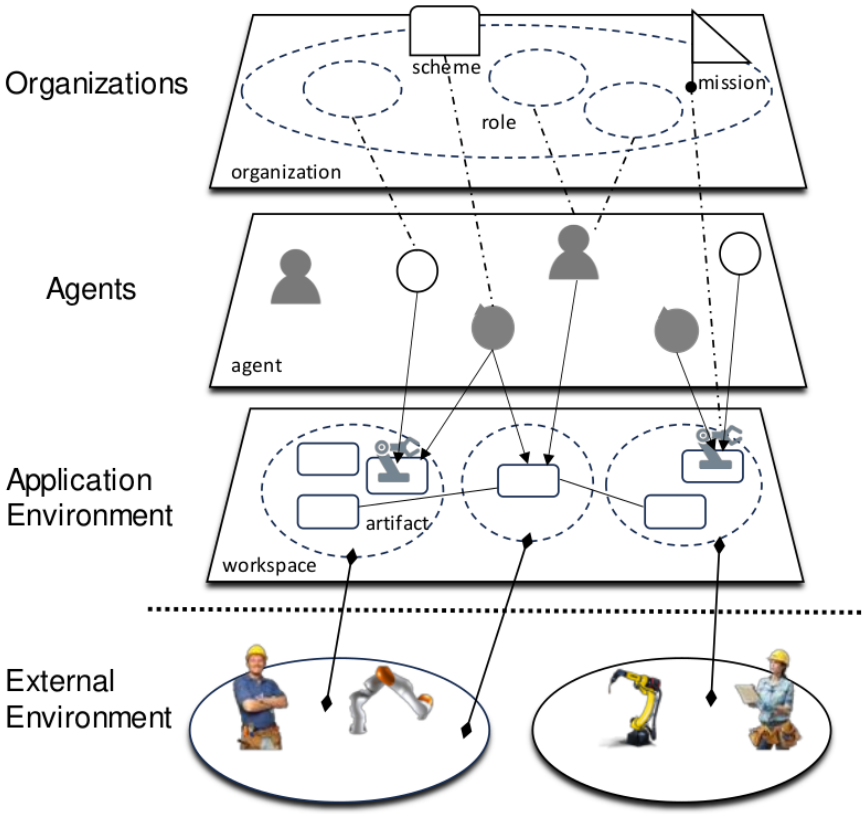
Organization Dimension

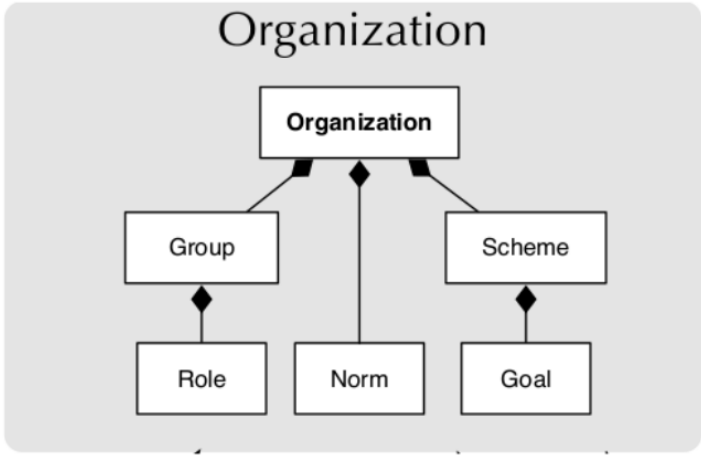




Ciortea, Hübner, Nardin (EASSS'2023)

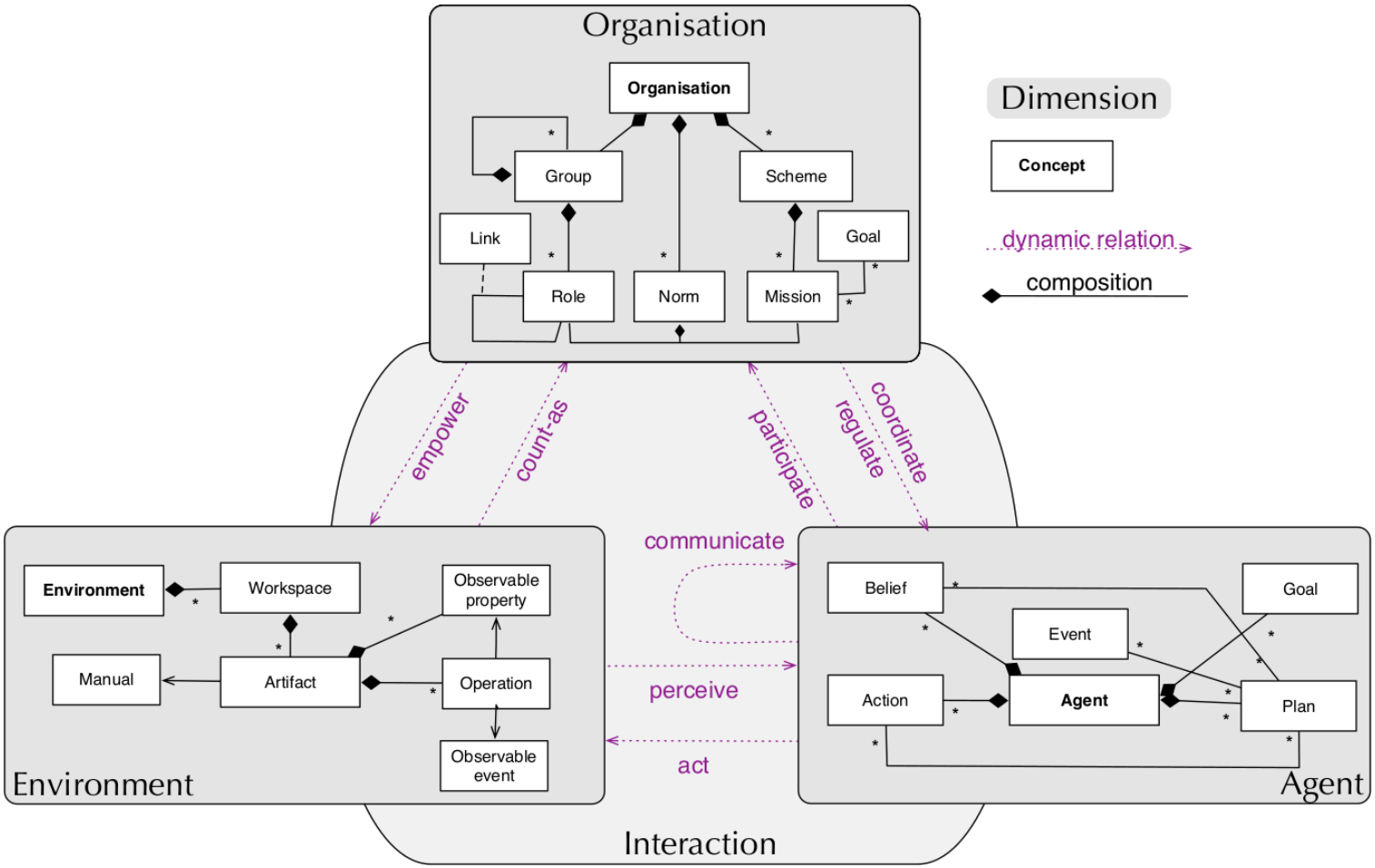
Interaction Dimension





Ciortea, Hübner, Nardin (EASSS'2023)

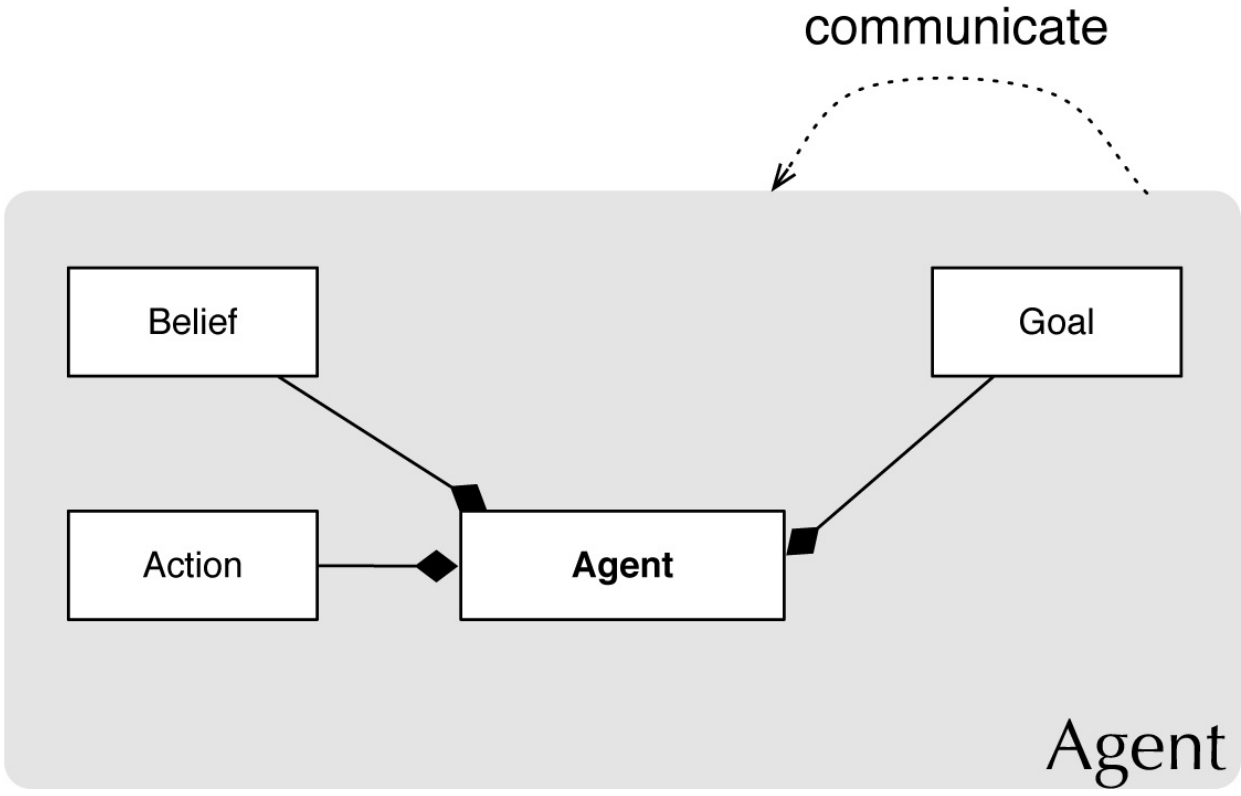
Interaction Dimension



[Boissier et al., 2020]

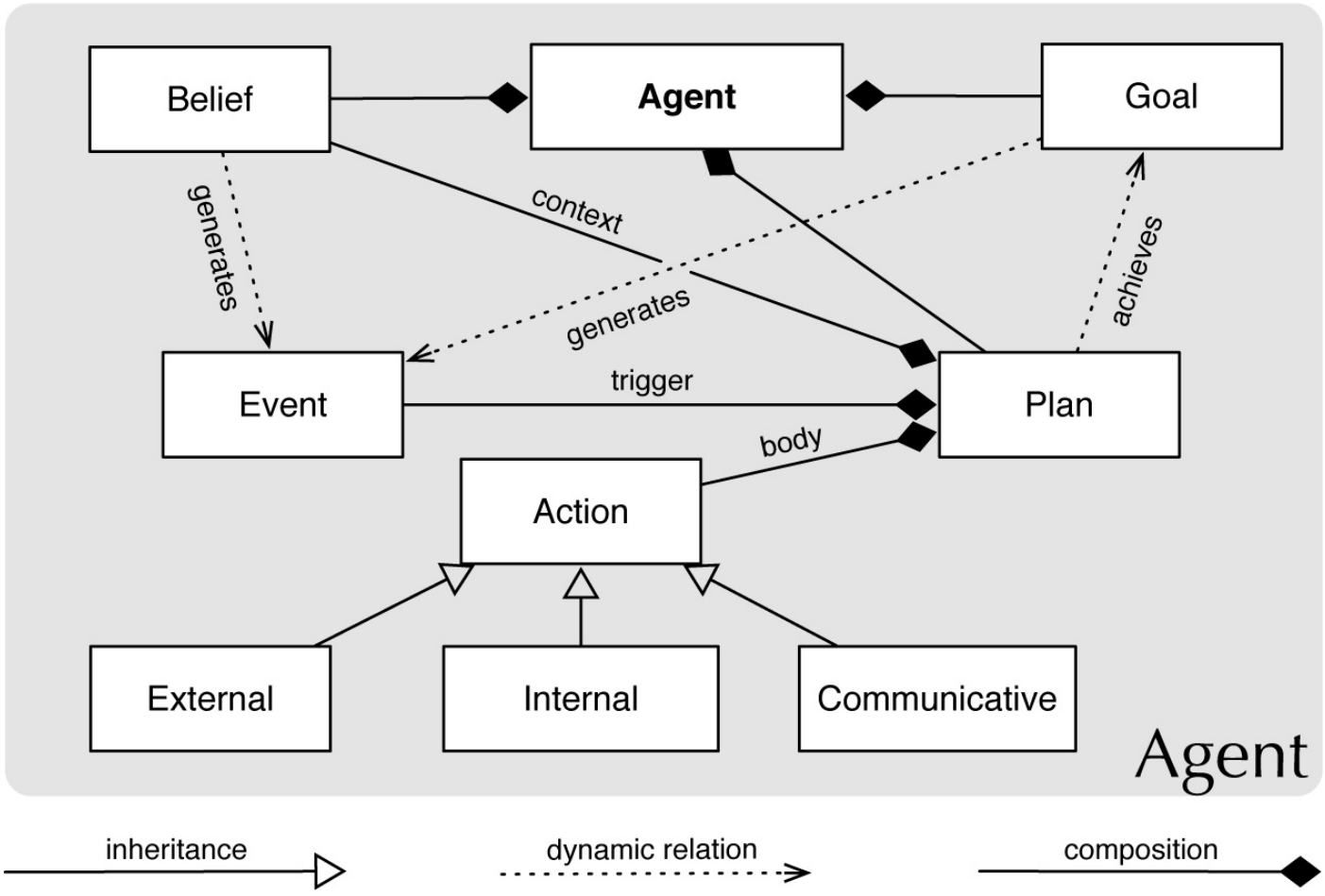
Part I
— Agents —

The *agent* dimension



[Boissier et al., 2020]

The *agent* dimension



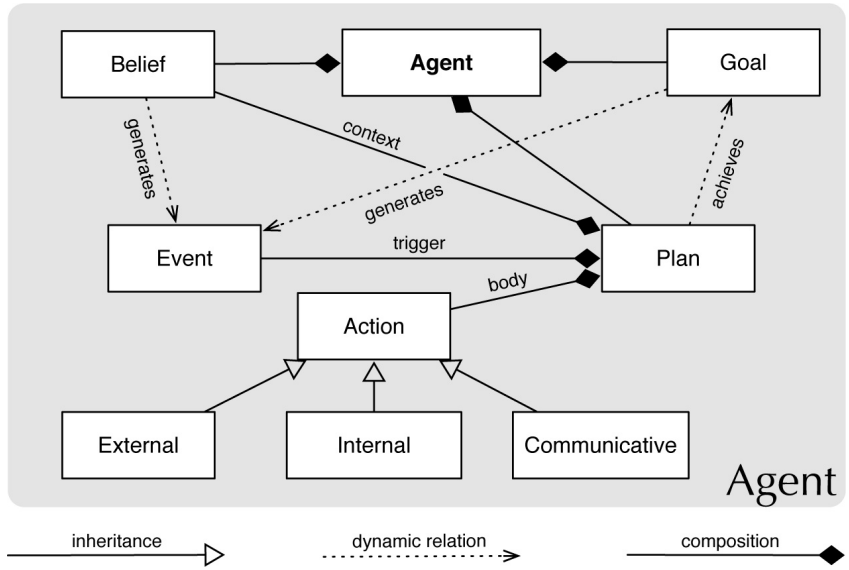
[Boissier et al., 2020]

The *agent* dimension

Beliefs: information available to the agent about the environment, other agents, itself...

Goals: states of affairs the agent wants to bring about

Plans: recipes for action (agent's know how)



[Boissier et al., 2020]

go to Lab1 - Task 1 and Task 2

Hello World - The first JaCaMo program

```
!greet. //initial desire (or goal)


//plans to satisfy the goal
+!greet : language(english)
    <- .print("hello world.").

+!greet : language(french)
    <- .print("bonjour.").
```

Hello World - The first JaCaMo application - version 1

JaCaMo app specification

```
mas agents_intro {
  agent bob: personal_assistant.asl
}
```

agent specification

```
language(english). initial belief
!greet. //initial desire (or goal)

//plans to satisfy the goal
+!greet : language(english)
    <- .print("hello world.").

+!greet : language(french)
    <- .print("bonjour.").
```

Hello World - The first JaCaMo application - version 2

JaCaMo app specification

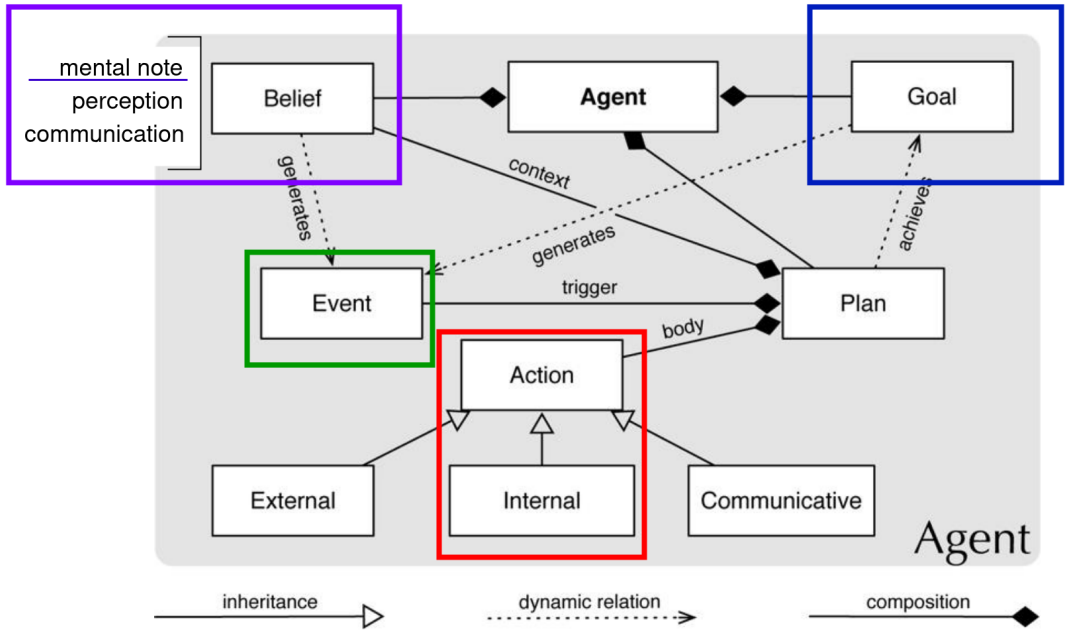
```
mas agents_intro {
  agent bob: personal_assistant.asl{
    goals: greet
    beliefs: language(english)
  }
}
```

agent specification

```
//plans to satisfy the goal
+!greet : language(english)
    <- .print("hello world.").

+!greet : language(french)
    <- .print("bonjour.").
```

Hello World - The first JaCaMo program



adapted from [Boissier et al., 2020]

JaCaMo app specification

```
mas agents_intro {
  agent bob: personal_assistant.asl{
    goals: greet
    beliefs: language(english)
  }
}
```

agent specification

```
!greet. //initial desire (or goal)

//plans to satisfy the goal
+!greet : language(english)
    <- .print("hello world.").

+!greet : language(french)
    <- .print("bonjour.").
```

Main agent constructs

Beliefs: information available to the agent

about the environment, other agents, itself...

Goals: states of affairs the agent wants to bring about

Plans: recipes for action (agent's know how)

Main agent constructs – Beliefs

Syntax

Beliefs are represented by annotated literals of first order logic
$functor(term_1, \cdots, term_n)[annot_1, \cdots, annot_n]$

Example

```
language(english)[source(self)] //added by the agent itself
temperature(20)[source(percept)] //added by perception
friend(bob,alice)[source(bob)] //added by communication from bob
```

Where do the beliefs come from? I

Coded as agent's initial beliefs

either in the .asl file
or in the .jcm file

```
language(english). //initial belief
!greet. //initial goal
//plan to satisfy the goal
+!greet : language(english)
    <- print(``Hello world'').
```

Added as part of plans

```
language(english). //initial belief
!greet. //initial desire
//plan to satisfy the desire
+!greet : language(english)
    <- print(``Hello world'').
      -last_greeting_date(friday); //remove belief
      +last_greeting_date(saturday); //add belief
      -+last_greeting_date(sunday). //update belief
```

Beliefs added by the agent itself include the `source(self)` annotation
e.g.: `language(english)[source(self)]`

Where do the beliefs come from? II

From the agent's perception of the environment

include the annotation `source(percept)`
are automatically updated as perceptions change

```
language(english).

!greet.

+!greet : language(english) &
          temperature(X)[source(percept)] & X <= 10
   <- .print("hello world.");
      .print("It is so cool").
```

Where do the beliefs come from? III

Beliefs added by communication

- Have the `source(ag)` annotation

  where ag is the agent that provides the information

- Dynamics:
  - when the agent receives a *tell* message, its content becomes a new belief

    ```
    .send(tom,tell,lier(alice));//sent by bob
    //adds lier(alice)[source(bob)] in tom's bb
    ```

  - when the agent receives an *untell* message, the corresponding belief is removed from the receiver's belief base

    ```
    .send(tom,untell,lier(alice));//sent by bob
    //removes lier(alice)[source(bob)] in tom's bb
    ```

Goals

Goals correspond to the notion of *desire* from BDI.

Represent the states of affairs the agent wants to bring about.

Types of goals:

- *achievement goal*: goal to do

  syntax: `!g.` : the agent wants to achieve *g*

- *test goal*: objetivo de saber

  sintaxe: `?g.` : the agent wants to know *g*

Exemplos

!greet. //achievement goal

?is_holiday. //test goal

Where do the goals come from? I

Agent's initial goal

```
language(english). //belief

!greet. //initial goal

//plan to satisfy the goal
+!greet : language(english)
    <- print("Hello world").
```

Part of plans

```
language(english). //belief

!greet. //initial goal

//plan to satisfy the desire
+!greet : language(english)
    <- print("Hello world.");
       //add test [sub]goal
       ?raining(now);
       //add achievement [sub]goal
       !check_temperature.
```

Where do the goals come from? II

Goals added by communication

- when the agent receives an *achieve* message, its content becomes an achievement goal annotated with the sender of the message

```
.send(tom,achieve,write(book));//sent by bob
//adds new goal write(book)[source(bob)] for Tom
```

- when the agent receives an *unachieve* message, the goal corresponding to the message's content is removed

```
.send(tom,unachieve,write(book));//sent by bob
//removes the goal write(book)[source(bob)] for Tom
```

- when the agent receives an *askOne* or *askAll* message, the content is a new test goal annotated with the sender of the message

```
.send(tom,askOne,published(P),Answer);//sent by bob
//adds new goal ?published(P)[source(bob)] for Tom
//the response of Tom unifies with Answer
```

Plans

```
triggering_event :  context <-- body.
```

`triggering_event`: event the plan is meant to handle

`context`: circumstances in which the plan is selected

`body`: course of action to be followed when the plan is selected

Triggering events

`+b` (belief addition)
`-b` (belief deletion)
`+!g` (achievement goal addition)
`-!g` (achievement goal deletion)
`+?g` (test goal addition)
`-?g` (test goal deletion)

Plans – context

Context is typically formed of conjunctions of beliefs that the agent is required to have at the time of choosing the plan

```
+!greet : language(english) & temperature(X) & X>=30
   <- . . .
```

Boolean operators

&	(and)
|	(or)
not	(not)
=	(not)
>,>=,<,<=	(relational)
==	(equals)
\ ==	(different)

Arithmetic operators

+	(sum)
–	(subtraction)
*	(multiply)
/	(divide)
div	divide – integer
mod	remainder
**	power

Plans - body

What may the plan body include?

```
+rain : time_to_leave(T) & clock.now(H) & H>=T
    <- !g1; //new subgoal
       !!g2; //new goal
       ?b(X); //new test goal
       +b1(T-H); //add mental note
       -b2(T+H); //remove mental note
       -+b3(T*H); //update mental note
       close(door); //external action
       .print(``Go!''). internal action
```

go to Lab1 - Task 3

Plans *vs.* actions of agents

Plans may include **actions** the agents are expected to execute.

But...
Where are these actions implemented?

There are two kinds of actions:

- Internal actions
- External actions

Internal actions

- are not necessarily performed upon an external device
- are part of the agent implementation
- are executed as part of the agent reasoning cycle
- are available to the agent in any environment

Internal actions are implemented in Java

- New *internal actions* can be implemented
  and added to the agent
- Jason provides a default set of internal actions
  available to every Jason agent
- More info here

Example (download here)

```
+!start <-
   .print("hello world.");
   .random(R); .print(R);
   .date(Y,M,D);
   .print(Y,-",M,-",D);
   .time(H,MN,S);
   .print(H,-",MN,-",S).
```

External actions

Executed on elements external to the agent's implementation i.e. *environment*

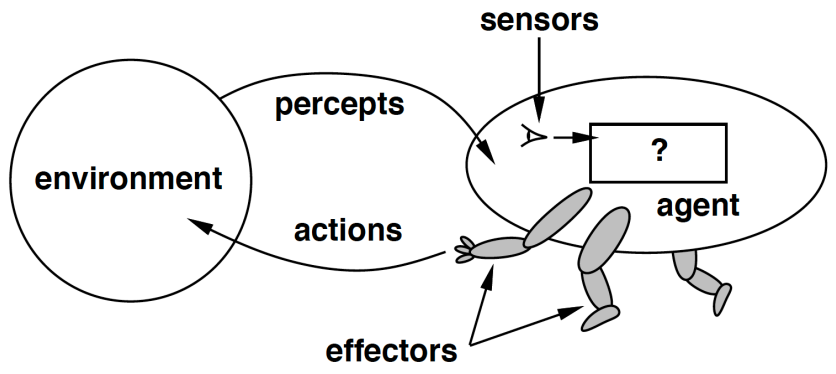
In JaCaMo, the environment is based on CArtAgO (details here and here)

go to Lab1 - Task 3 and Lab 2

Part II
— Environment —

Some perspectives about environment

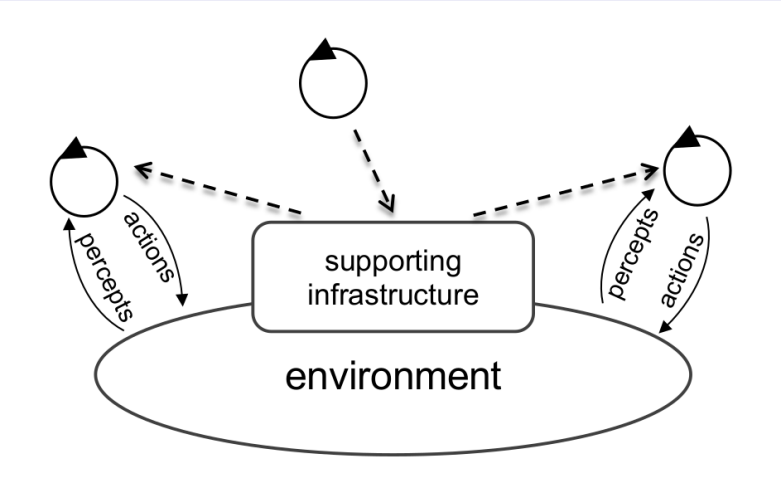
Single-agent perspective



[Russel e Norvig, 1995]

The environment is **external** to the system

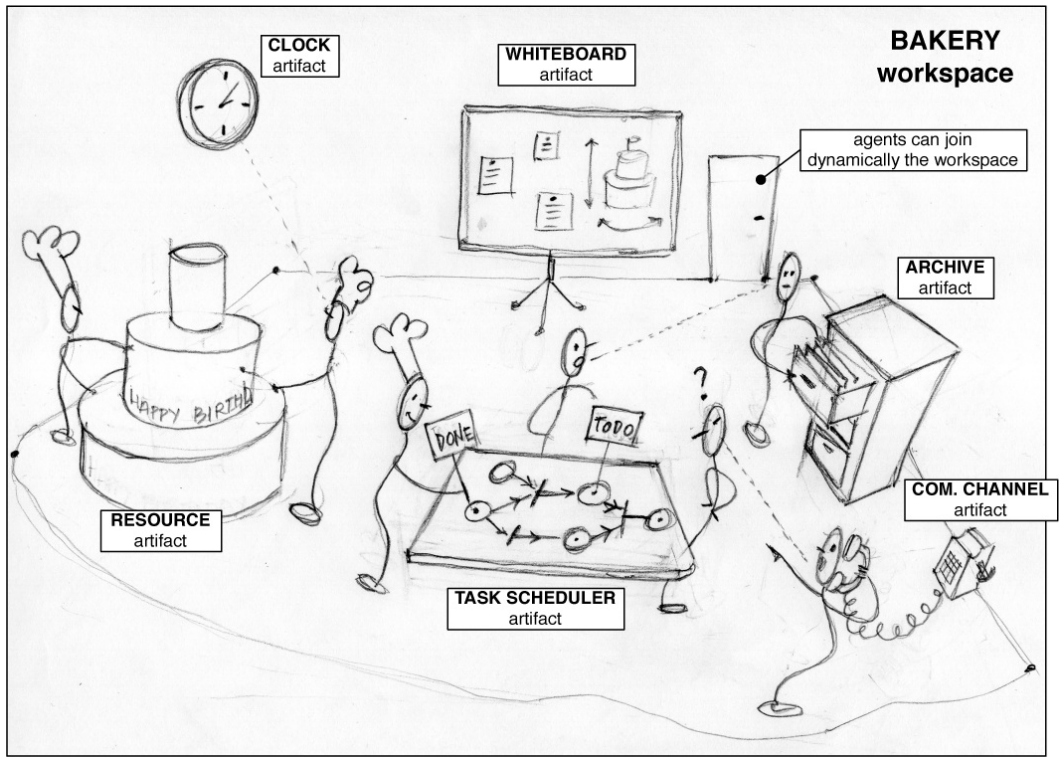
MAOP perspective



The environment is **part** of the system

Environment as a first-class abstraction

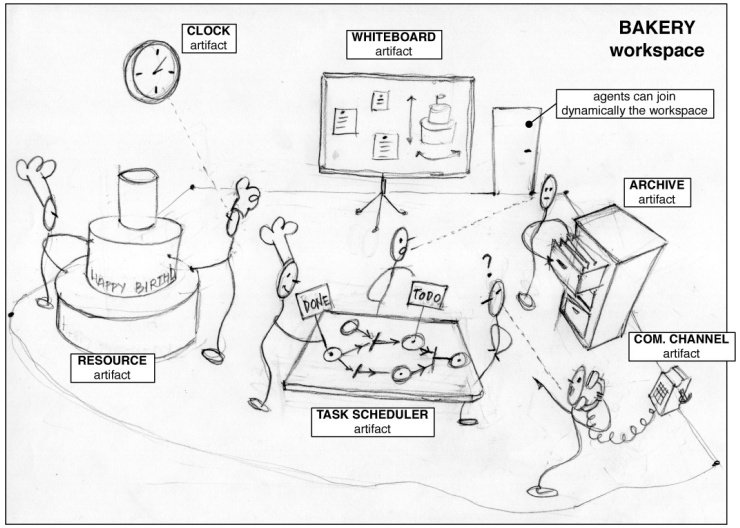
Programming MAS = programming Agents + programming Environment

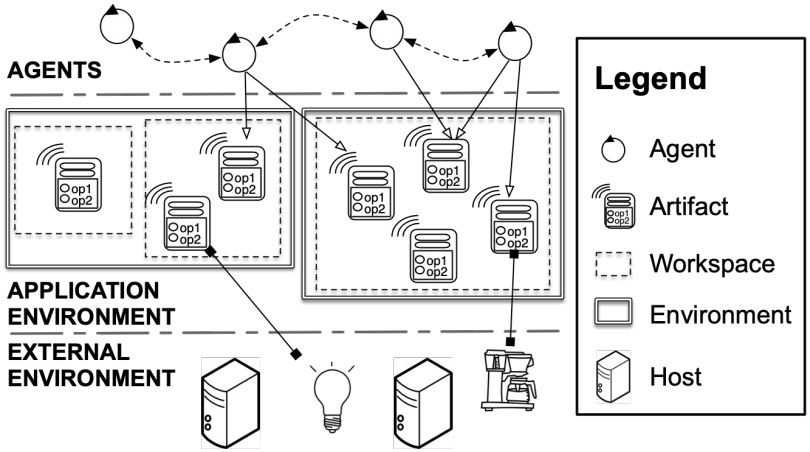


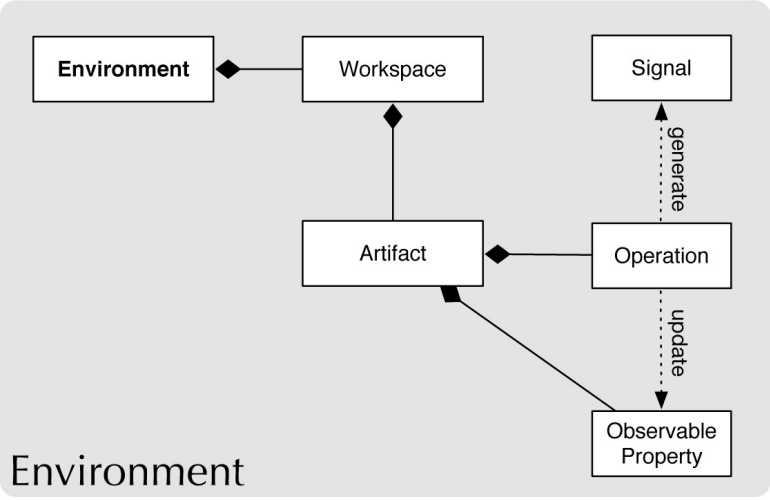
[Boissier et al., 2020]

Agents are **autonomous** entities that exploit resources
provided by **non-autonomous artifacts** available in the environment

Environment as a first-class abstraction







[Boissier et al., 2020]

Environment = a set of workspaces that contain sets of artifacts

Artifacts

- basic building blocks of the environment
- the agents
  - perecive its observable properties and signals
  - exploit its provided operations

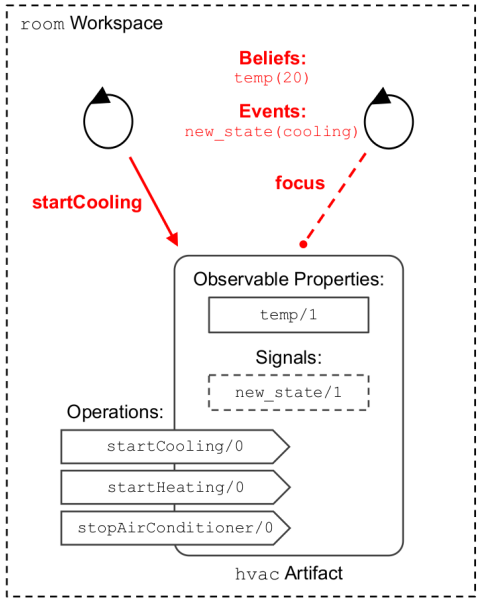
Environment as a first-class abstraction

Environment = a set of workspaces that contain sets of artifacts

Artifacts

- basic building blocks of the environment
- agents
  - perecive its observable properties and signals
  - exploit its provided operations

Artifacts



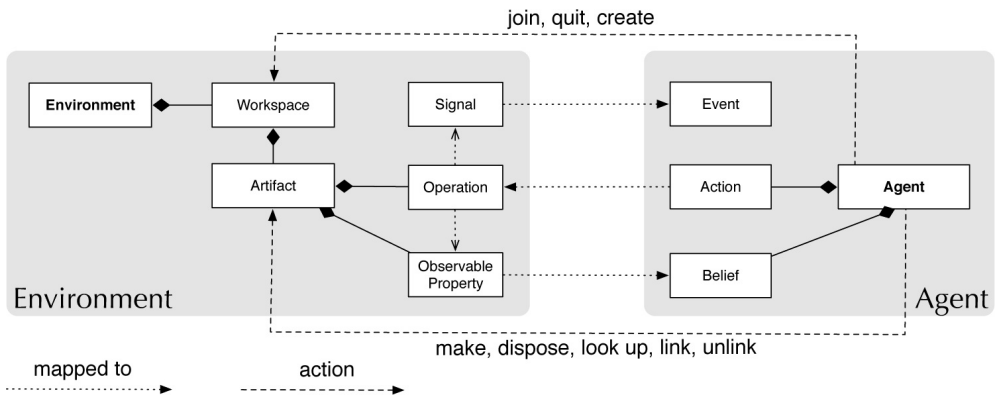
Ciortea, Hübner, Nardin
(EASSS'2023)

Usage interface:

- *observable properties*: state variables that can be perceived by the agents (are mapped to beliefs)
- *observable events*: non-persistent signals that can be perceived by the agents (are mapped to beliefs)
- *operations*: artifact functionalities available to the agents (are mapped to actions)

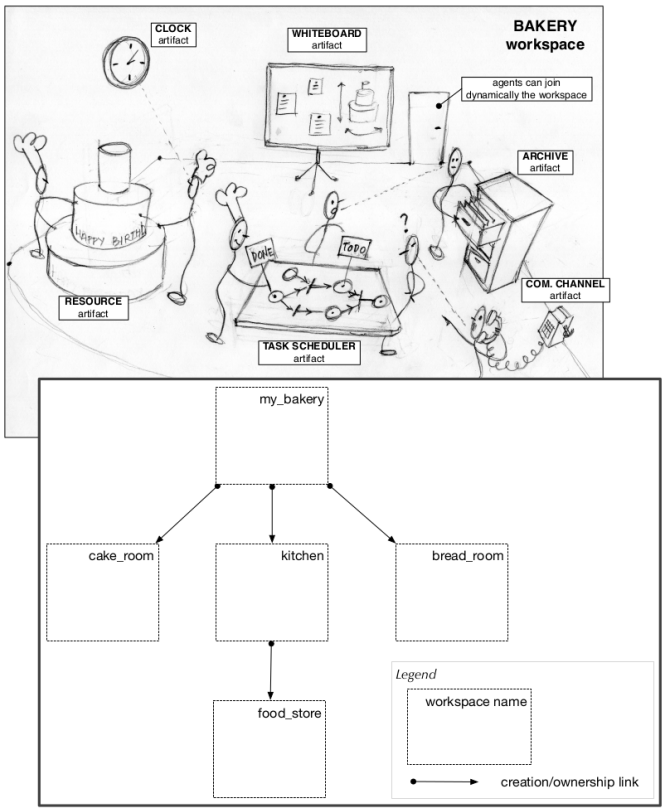
Intentional focus:

- Agents focus on artifacts to perceive observable properties and signals



[Boissier et al., 2020]

Workspaces



Ciortea, Hübner, Nardin
(EASSS'2023)

A **logical place** containing artifacts

Agents can **join**, **leave**, and **work** in multiple workspaces at the same time

Multiple agents can work in the same workspace simultaneously

Workspaces can be distributed over a network

Can structure complex environment through hierarchical organization

Part III
— Organization —

Organizations

supra-individual phenomena: *Organisations are structured, patterned systems of activity, knowledge, culture, memory, history, and capabilities that are distinct from any single agent* [Gasser, 2001]

directed to a purpose: *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]

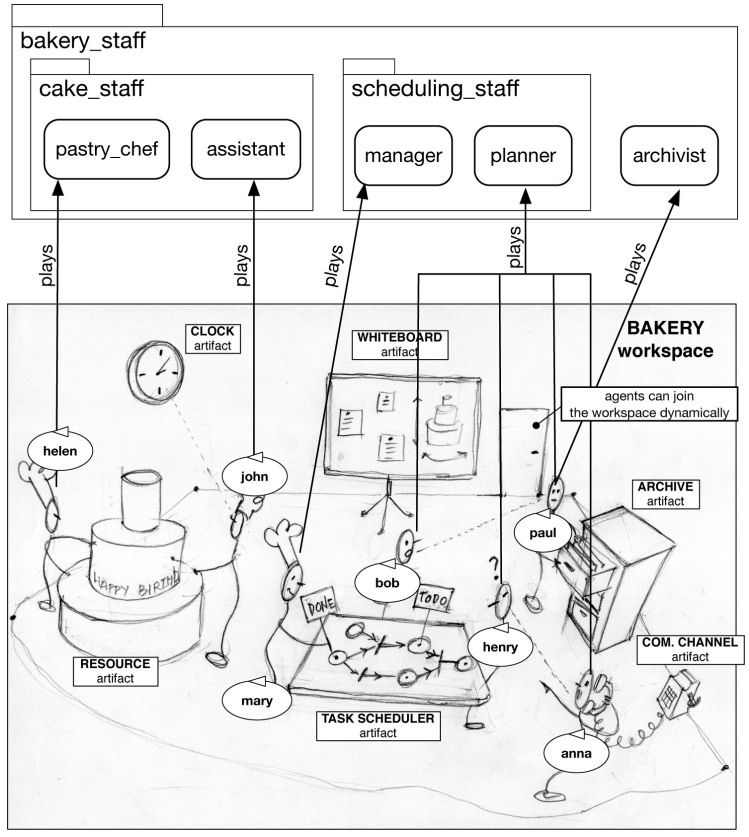
predefined guidelines: *An organisation is characterised by: a division of tasks, a distribution of roles, authority systems, communication systems, contribution-retribution systems* [Bernoux, 1985]

emergent 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]

Organizations in MAS

Organizations provide concepts and abstractions to specify and govern MAS from a **macro** perspective

i.e. considering the combinated behaviour of multiple agents



[Boissier et al., 2020]

Organizations in JaCaMo

Organization Specification

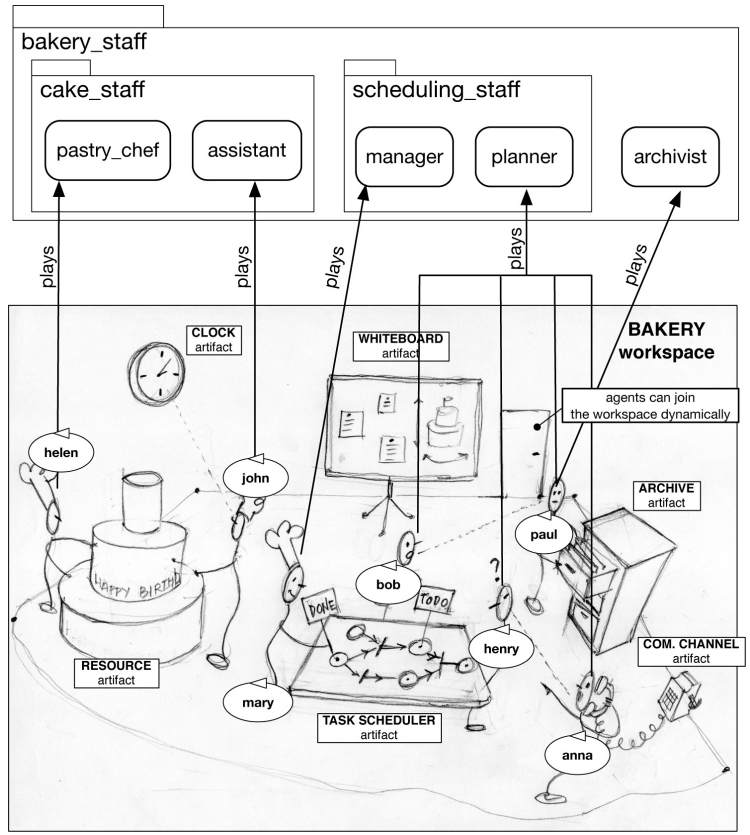
defines **what** is the expected outcome to be produced by the agents without explaining *how* to produce it

the agents themselves decide *how* to act to produce these outcomes

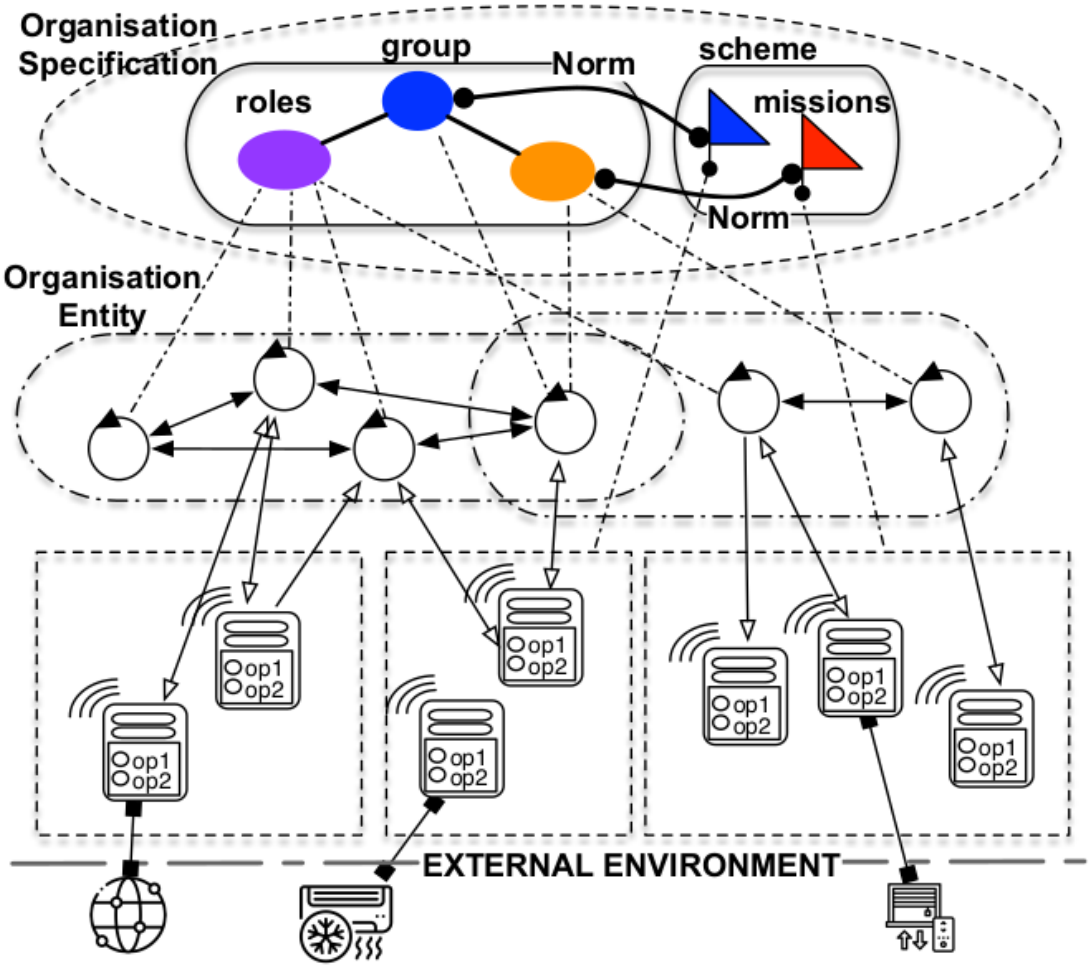
Organization Entity

enacts the organization specification

agents play roles, join groups, achieve goals, etc.



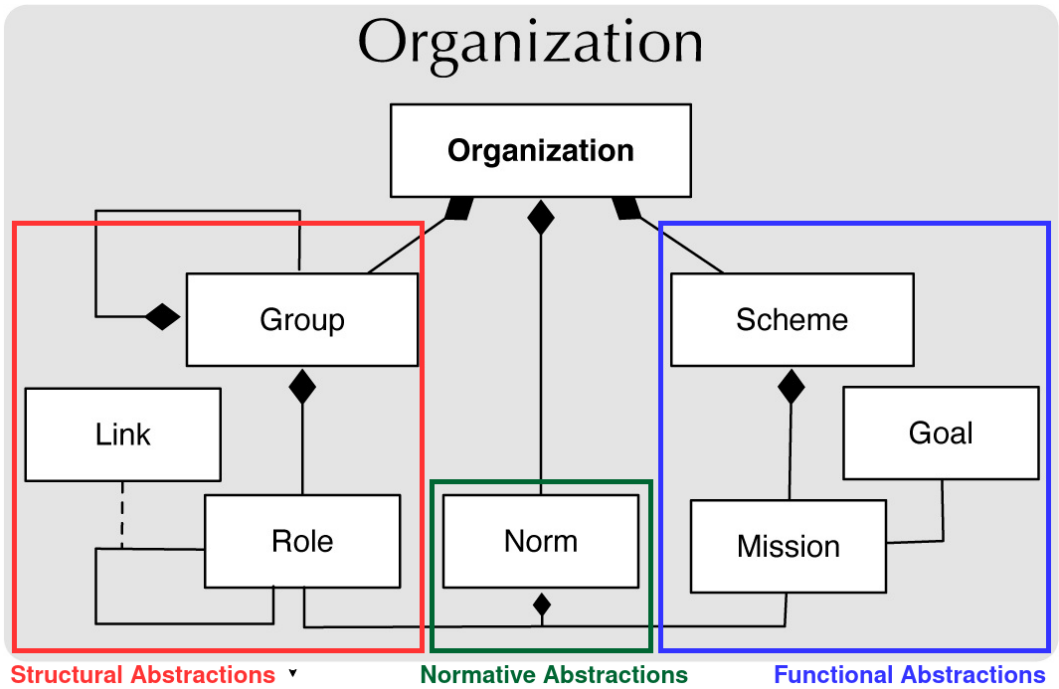
Organizations in JaCaMo



Organization specification in JaCaMo

Three dimensions:

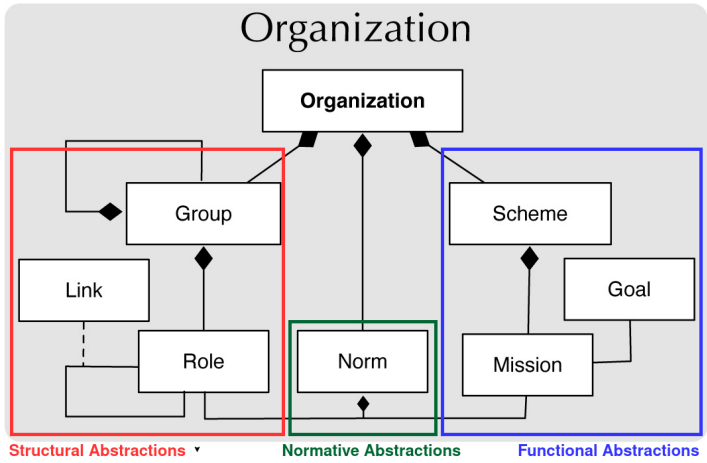
1. Structural: specifies how the organization is structured
2. Functional: specifies how the agents' activities are coordinated
3. Normative: specifies the expected agents' behaviour



Organization specification in JaCaMo

Structural specification

Structures of the organization in three levels:



- **Individual**, where **roles** label the positions the agents can occupy in the organization
- **Social**, where **links** specify the possible interactions among the agents
- **Collective**, where **groups** define the community of agents within the orgnanization

Organization specification in JaCaMo

Structural specification

```
1  <structural-specification>
2
3  <role-definitions>
4      <role id="role0"/>
5      <role id="role1"> <extends role="role0"/> </role>
6      <role id="role2"> <extends role="role0"/> </role>
7  </role-definitions>
8
9  <group-specification id="group1">
10     <roles>
11         <role id="role1" min="1" max="2"/>
12         <role id="role2" min="1" max="1"/>
13     </roles>
14
15     <links>
16        <link from="role1" to="role2" type="authority"
17              scope="intra-group" bi-dir="false" />
18        <link from="role0" to="role0" type="communication"
19              scope="intra-group" bi-dir="true" />
20     </links>
21     <formation-constraints>
22        <compatibility from="role1" to="role2" bi-dir="true"/>
23     </formation-constraints>
24 </group-specification>
25 </structural-specification>
```

Organization specification in JaCaMo

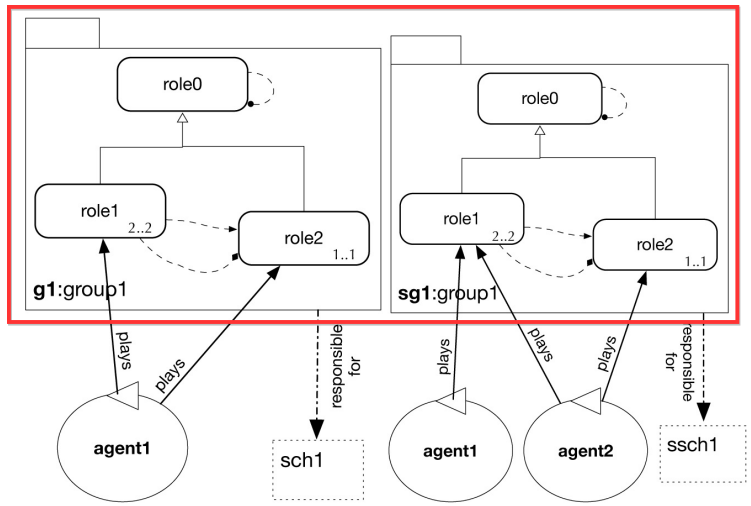
Structural specification

```
<structural-specification>

<role-definitions>
    <role id="role0"/>
    <role id="role1"> <extends role="role0"/> </role>
    <role id="role2"> <extends role="role0"/> </role>
</role-definitions>

<group-specification id="group1">
    <roles>
        <role id="role1" min="1" max="2"/>
        <role id="role2" min="1" max="1"/>
    </roles>

    <links>
       <link from="role1" to="role2" type="authority"
             scope="intra-group" bi-dir="false" />
       <link from="role0" to="role0" type="communication"
             scope="intra-group" bi-dir="true" />
    </links>
    <formation-constraints>
       <compatibility from="role1" to="role2" bi-dir="true"/>
    </formation-constraints>
</group-specification>
</structural-specification>
```



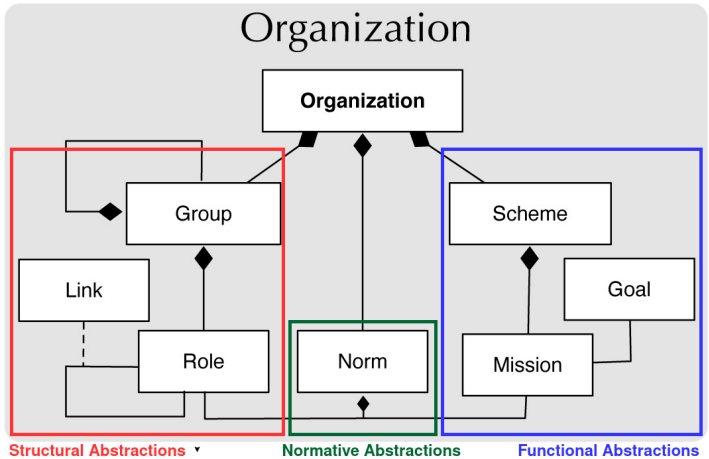
Structural specification elements

- List of unique roles
- List of unique groups
- Social links
- Inheritance relations among roles
- Formation constraints

Organization specification in JaCaMo

Functional specification

Specifies the expected agents' behaviour in two levels:



- **Individual**, where **goals** are gathered in **missions** to be accomplished by the agents
- **Collective**, where a **social scheme** states guidelines for goal achievements and mission accomplishments

Organization specification in JaCaMo

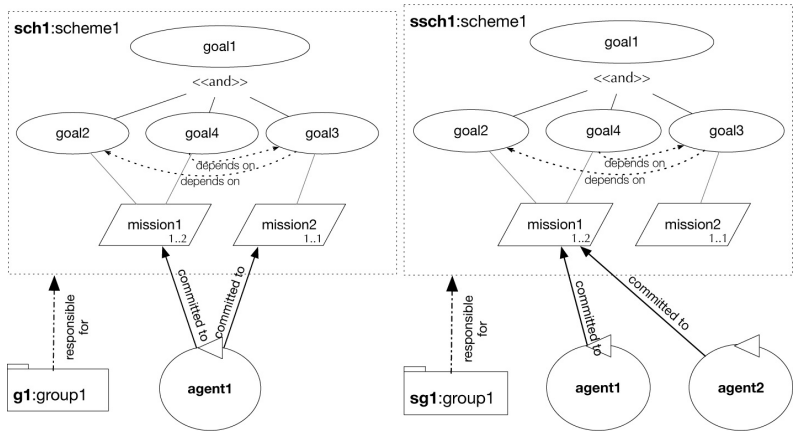
Functional specification

```
1  <functional-specification>
2      <scheme id="scheme1">
3          <goal id="goal1" ds="description of goal1">
4              <plan operator="sequence">
5                  <goal id="goal2" ttf="20 minutes" ds="description of goal2"/>
6                  <goal id="goal3" ds="description of goal3"/>
7                  <goal id="goal4" ds="description of goal4"/>
8              </plan>
9          </goal>
10
11         <mission id="mission1" min="1" max="2">
12             <goal id="goal2" />
13             <goal id="goal4" />
14         </mission>
15         <mission id="mission2" min="1" max="1">
16             <goal id="goal3"/>
17         </mission>
18     </scheme>
19 </functional-specification>
```

Organization specification in JaCaMo

Functional specification

```
1  <functional-specification>
2      <scheme id="scheme1">
3          <goal id="goal1" ds="description of goal1">
4              <plan operator="sequence">
5                  <goal id="goal2" ttf="20 minutes" ds="description
6                  <goal id="goal3" ds="description of goal3"/>
7                  <goal id="goal4" ds="description of goal4"/>
8              </plan>
9          </goal>
10
11         <mission id="mission1" min="1" max="2">
12             <goal id="goal2" />
13             <goal id="goal4" />
14         </mission>
15         <mission id="mission2" min="1" max="1">
16             <goal id="goal3"/>
17         </mission>
18     </scheme>
19 </functional-specification>
```



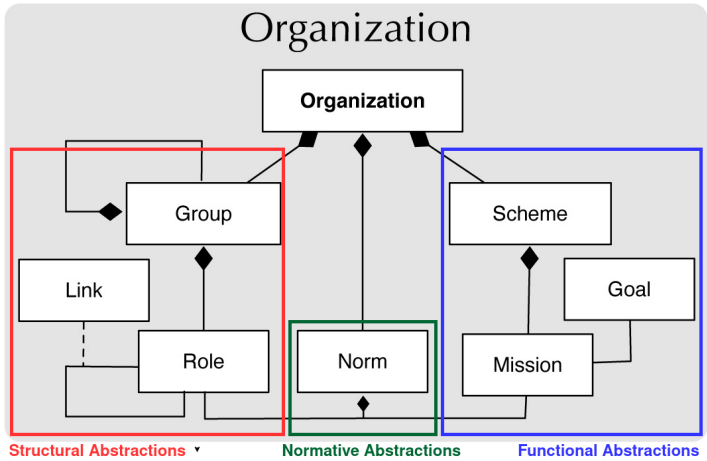
Functional specification elements

- Goal decomposition
  - Subgoals (plans)
- Plan operators
  - sequence
  - choice
  - parallel
- Missions
  - Mission cardinalities

Organization specification in JaCaMo

Normative specification

Norms specifiy what is obliged, permitted, or prohibited and for the agents



Organization specification in JaCaMo

Normative specification

```
<normative-specification>
    <!-- the norms of the application -->
    <norm id="norm1" type="obligation"
          role="role1" mission="mission1"/>
    <norm id="norm2" type="permission"
          role="role2" mission="mission2"/>
</normative-specification>
```

- *Deontic modality* or *type*, which can be an *obligation* or a *permission*
- *Role*, representing the *bearer* of the deontic modality
- *Mission* that the deontic modality refers to

Organization entity in JaCaMo

Setup in design time (by the systems' designer)

```
smart-room.jcm
   mas smart_room {
       . . .
      organisation smart_house_org : smart_house.xml {
         group r1 : room {
            players: pa1 assistant
               pa2 assistant
               pa3 assistant
               rc controller
            responsible-for: temp_r1
         }
         scheme temp_r1: decide_temp
      }
   }
```

Organization entity in JaCaMo

Setup at runtime (by the agents)

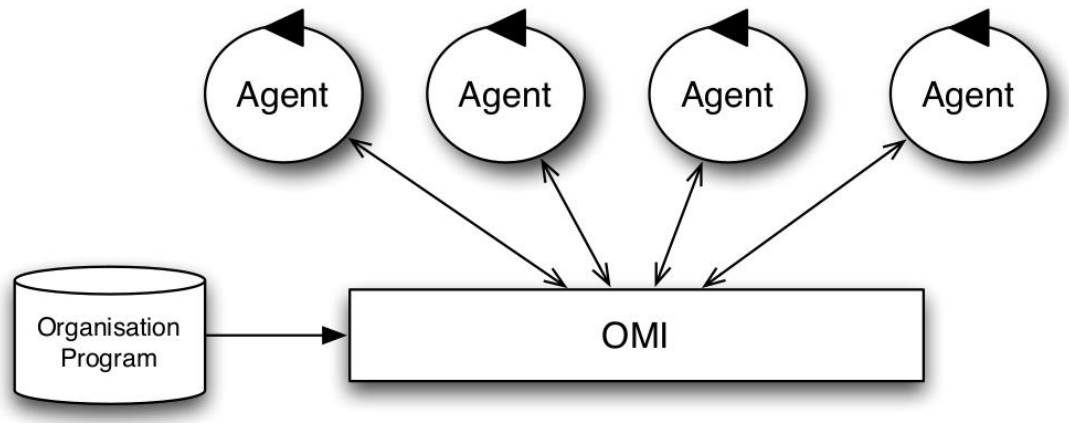
```
7   +!start : true
8      <- // Creation of organization entity "smorg"
9         createWorkspace(smorg);
10        joinWorkspace(smorg,WspId);
11        makeArtifact(smorg, "ora4mas.nopl.OrgBoard",
12                     ["src/org/org.xml"], OrgArtId)[wid(WspId)];
13        focus(OrgArtId)[wid(WspId)];
14        // Group-entity lifecycle: Creation of group entity "g1" in "smorg"
15        createGroup(g1, group1, GrArtId)[artifact_id(OrgArtId)];
16        focus(GrArtId)[wid(WspId)];
17        // Group-entity lifecycle: Adoption of role "role1" in "g1"
18        adoptRole(role1)[artifact_id(GrArtId)];
19        // Group-entity lifecycle: Adoption of role "role2" in "g2"
20        adoptRole(role2)[artifact_id(GrArtId)];
21        // Group-entity lifecycle: Waiting that "g1" is well-formed
22        .wait(formationStatus(ok)[artifact_id(GrArtId)]);
23        // Scheme-entity lifecycle: Creation of social scheme entity "sch1"
24        createScheme(sch1, scheme1, SchArtId)[artifact_id(OrgArtId)];
25        // Group-entity lifecycle: Adding of "sch1" under the responsibility of "g1"
26        addScheme(sch1)[artifact_id(GrArtId)];
27        focus(SchArtId)[wid(WspId)];
28        // organization entity "smorg" created
29        .
30
31  // Scheme-entity lifecycle: commitment to missions
32  +permission(Ag, MCond, committed(Ag, Mission, Scheme), Deadline) : .my_name(Ag)
33      <- commitMission(Mission)[artifact_name(Scheme)].

46  // Common definitions pertaining to Scheme-entity and Normative-entity lifecycle
47  // Obedience to norm prescriptions
48  { include("$moiseJar/asl/org-obedient.asl") }
```

Organization Management Infrastructure (OMI)

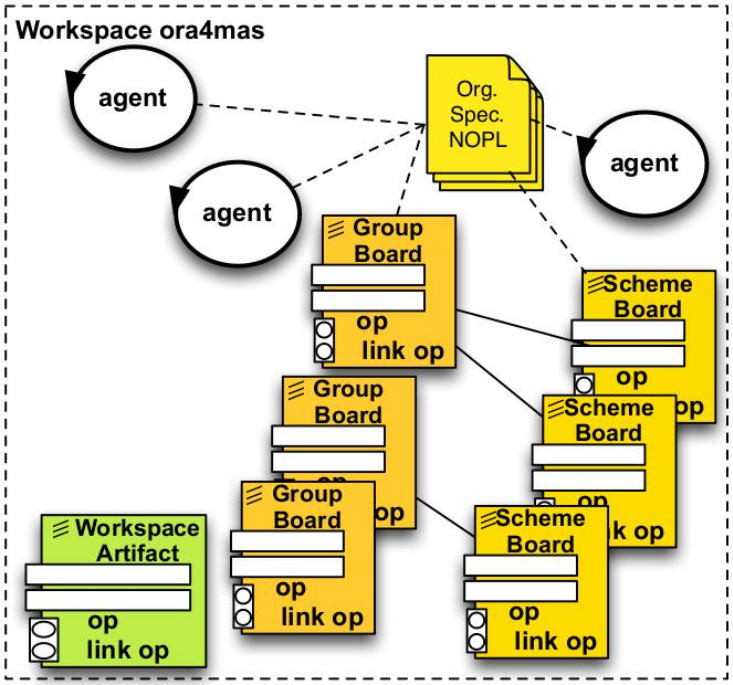
Software entity that manages the organization execution according to an organization specification.

Provides interfaces for the agents to interact with the organization

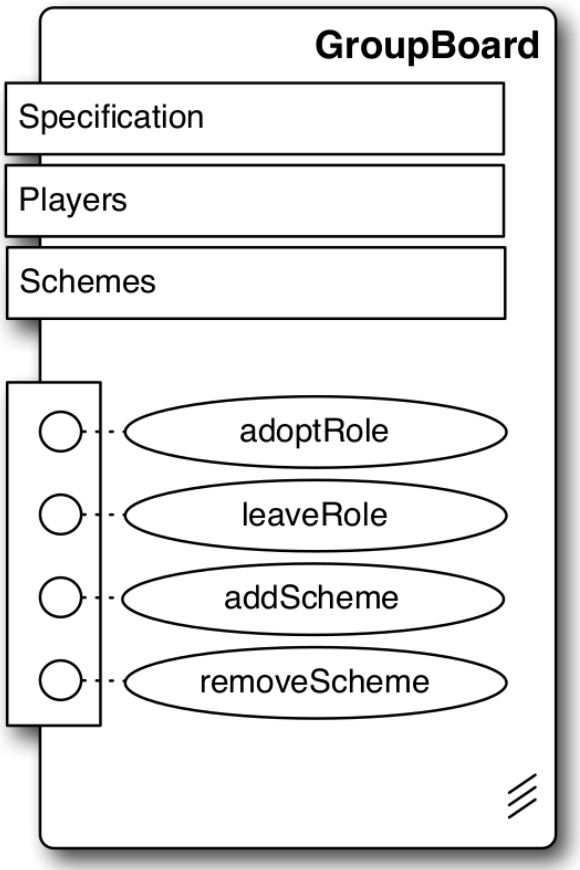


OMI in JaCaMo

- Based on A& A
- Agents can create, observe and act upon organizational artifacts
- Artifacts monitor norm compliance



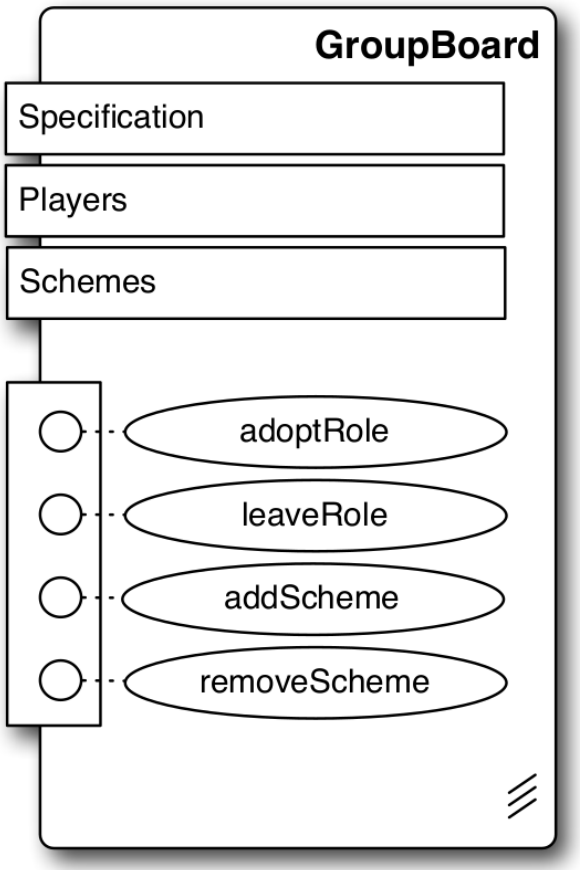
GroupBoard Artifact



Observable properties

- *specification*: the specification of the group in the OS
- *players*: a list of agents playing roles in the group. Each element of the list is a pair (agent role)
- *schemes*: a list of scheme identifiers that the group is responsible for

GroupBoard Artifact



Operations

- *adoptRole(role)*: the agent executing this operation tries to adopt a role in the group
- *leaveRole(role)*
- *addScheme(schid)*: the group becomes responsible for the scheme managed by the SchemeBoard schId
- *removeScheme(schid)*

SchemeBoard Artifact

Observable properties

- *specification*: the specification of the scheme in the OS
- *groups*: a list of groups responsible for the scheme
- *players*: a list of agents committed to the scheme. Each element of the list is a pair (agent, mission)
- *goals*: a list with the current state of the goals
- *obligations*: list of obligations currently active in the scheme

SchemeBoard Artifact

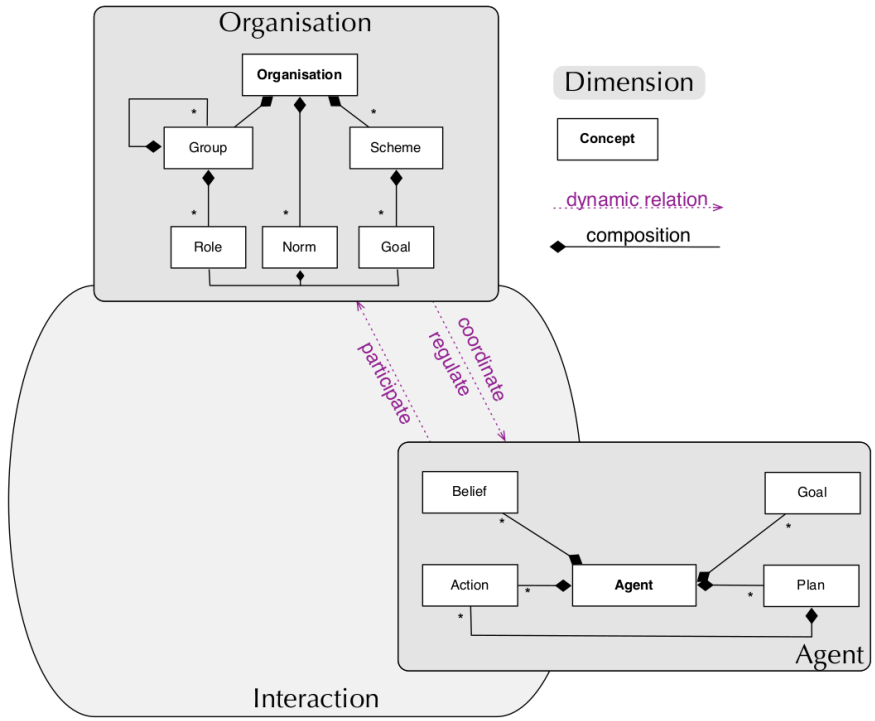
Operations

- *commitMission(mission)* and *leaveMission*: operations to enter and leave the scheme
- *goalAchieved(goal)*: defines that some goal is achieved by the agent performing the operation
- *setGoalArgument(goal,argument,value)*: defines the value of some goal's argument

Agent-Organization Integration

Artifact-based OMI:

- organizational state is represented through observable properties that are mapped to beliefs of the agents
- agents can use operations of OMI artifacts to interact with the organization
- agents perceive organizational events



Organization Beliefs

Belief	Description
`play(A,R,G)`	Agent `A` is playing role `R` in group `G`
`commitment(A,M,S)`	Agent `A` is committed to mission `M` in scheme `S`
`formationStatus(S)[artifact_name(_,A)]`	The formation status for scheme or group `A` is `S` (possible values for `S` are `ok` or `nok`)
`goalState(S,G,LC,LA,T)`	Goal `G`, of scheme `S`, is in state `T` (possible values for `T` are `waiting`, `enabled`, `satisfied`); `LC` is a list of agents committed to the goal, and `LA` is the list of agents that have already achieved the goal
`goalArgument(S,G,A,V)`	Argument `A` of goal `G` has value `V` in scheme `S`
`obligation(A,R,G,D)`	Agent `A` is obliged to achieve `G` before `D` while `R` holds
`permission(A,R,G,D)`	Agent `A` is permitted to achieve `G` before `D` while `R` holds

Organization Events

Event	Description
`oblCreated(O)`	Obligation o was created
`oblFulfilled(O)`	Obligation o was fulfilled
`oblUnfulfilled(O)`	Obligation o was unfulfilled
`oblInactive(O)`	Obligation o is inactive

Organization Actions

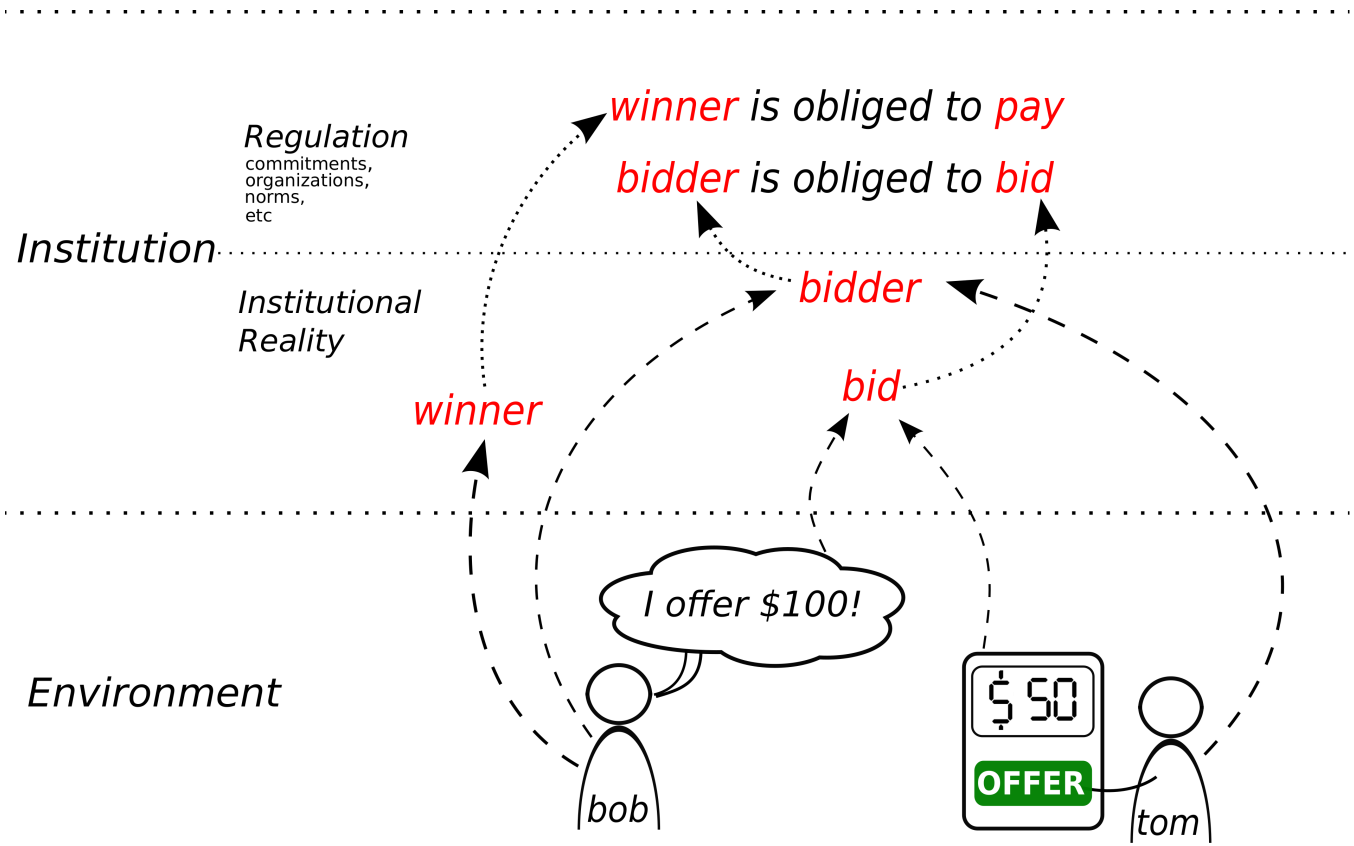
Belief	Description
`createGroup(Name,Type,ArtId)[artifact_name(O)]`	Creates a new group of name `Name`, following the group specification `Type` defined in the organization specification used to create organization `O`. The organization artifact `GroupBoard` that manages it is identified by `ArtId`
`createScheme(Name,Type,ArtId)[artifact_name(O)]`	Creates a new scheme of name `Name`, following the scheme specification `Type` defined in the organization specification used to create organization `O`. The organization artifact `SchemeBoard` that manages it is identified by `ArtId`
`adoptRole(R)[artifact_name(G)]`	Adopt role `R` in group `G`
`leaveRole(R)[artifact_name(G)]`	Leave role `R` in group `G`
`addScheme(S)[artifact_name(G)]`	Add scheme `S` to the responsibility of group `G`
`removeScheme(S)[artifact_name(G)]`	Remove scheme `S` of the responsibility of group `G`

Organization Actions

Belief	Description
`commitMission(M)[artifact_name(S)]`	Commit to mission `M` in scheme `S`
`leaveMission(M)[artifact_name(S)]`	Leave mission `M` in scheme `S`
`resetGoal(G)[artifact_name(S)]`	Set goal `G` to not satisfied in scheme `S`
`setArgumentValue(G,A,V)[artifact_name(S)]`	Set `V` as the value of argument `A` of goal `G` in scheme `S`

Institutions in [agent] societies

Institutions provide a social interpretation of brute facts [Searle 1995, Searle 2009]

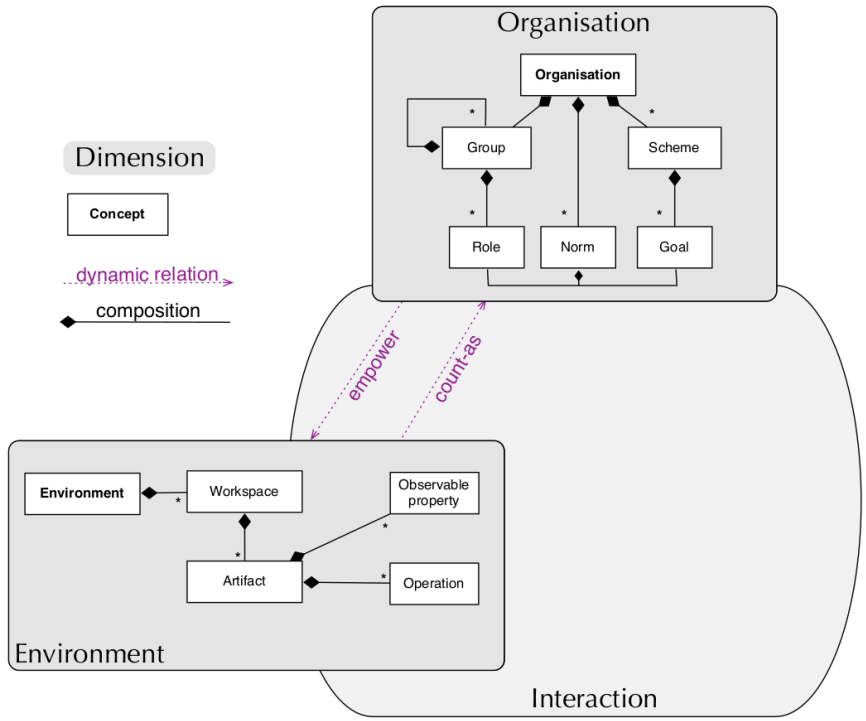


Institutions in JaCaMo

- Organizations are part of institutions
- Institutions *situate* the organizations in the environment

  i.e. connect the organization to the environment
- Environmental circumstances *count as* organizational states

  e.g. agents playing roles, goals achieved, etc.



Institutions in JaCaMo

The Situated Artificial Institutions (SAI) model [Brito et al. 2018]

- provides a model of institution
- provides a language specify institutions
- provides an interpreter to animate the institutional state
- integrated with JaCaMo

Situated Artificial Institutions (SAI)

Status functions

Status functions: elements that exist within the institution

constituted from elements external the institution (e.g. money, professors, marriages)

Default status functions in JaCaMo:

- `play(A,R,G)`. There are environmental and constitutive states that count as the agent A playing the role R in the group G.
- `responsible(G,S)`. There are environmental and constitutive states that count as the group G being responsible to execute the scheme S.
- `committed(A,M,S)`. There are environmental and constitutive states that count as the agent A being committed to the mission M in the scheme S.
- `done(S,G,A)`: There are environmental and constitutive states that count as the goal G of the scheme S has been achieved by the agent A.

Situated Artificial Institutions (SAI)

Constitutive rules

Constitutive rules: specify the constitution of status functions

Express that an element X starts to count as Y when the *Event* occurrs. The constitution stands while some *State* holds

General form

```
X count-as Y      /* X is either an environmental or an institutional element */
   when Event     /* event occurring in the environment or in the institution  */
   while State.   /* logical formula expressing an environmental/institutional state  */
```

- `X`: agents, events triggered by operations and signals of environmental artifacts, observable properties of artifacts (including organizational ones)
- `Y`: `play(A,R,G)`, `responsible(G,S)`, `committed(A,M,S)`, `done(S,G,A)`
- `Event`: events triggered by operations and signals of environmental artifacts
- `State`: logical formula composed of observable properties of artifacts (including organizational ones)

Situated Artificial Institutions (SAI)

Constitutive rules

Constitutive rules: specify the constitution of status functions

Express that an element X starts to count as Y when the *Event* occurrs. The constitution stands while some *State* holds

General form

```
X count-as Y     /* X is either an environmental or an institutional element */
   when Event    /* event occurring in the environment or in the institution  */
   while State.  /* logical formula expressing an environmental/institutional state  */
```

Example

```
play(A,assistant,r1)
   count-as committed(A,mVote,temp_r1)
   while responsible(r1,temp_r1).
```

Institution setup in JaCaMo

```
mas smart_room{
    . . .
    workspace room {
    artifact hvac: devices.HVAC(30)
    artifact vote: voting.VotingMachine
    }

    /* the institution smart_house_int is specified in the file smart_house.sai */
    /* facts occurring in the workspace ‘‘room’’ are considered in the constitution proccess */
    institution smart_house_inst:smart_house.sai{
        workspaces: room
    }

    /* this organization is part of the smart_house_inst */
    organisation smart_house_org : smart_house.xml {
        group r1 : room
        scheme temp_r1: decide_temp
        situated: smart_house_inst
    }
}
```

References I

La sociologie des organisations. Seuil, 3ème edition

Multi-Agent Oriented Programming: Programming Multi-Agent Systems Using JaCaMo. 1 ed. The MIT Press, 2020.

Rafael H. Bordini, Jomi F. Hübner, Michael Wooldridge
Programming Multi-Agent Systems in AgentSpeak using Jason.. 1 ed. Willey, 2007.

BRITO, M. de; HÜBNER, J. F; BOISSIER, O. Situated Artificial Institutions: Stability, consistency, and flexibility in the regulation of agent societies. JAAMAS. 2018

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.

La méthode (1) : la nature de la nature Points Seuil

References II

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

Stuart Russel and Peter Norvig.
Artificial Intelligence: A Modern Approach. 1 ed. Prentice Hall, 1995.

SEARLE, J. The Construction of Social Reality. [S.l.]: Free Press, 1995.

SEARLE, J. Making the Social World:The Structure of Human Civilization. [S.l.]: Oxford University Press, 2009.

Michael Wooldridge
An Introduction to MultiAgent Systems.. 2 ed. Willey, 2009.