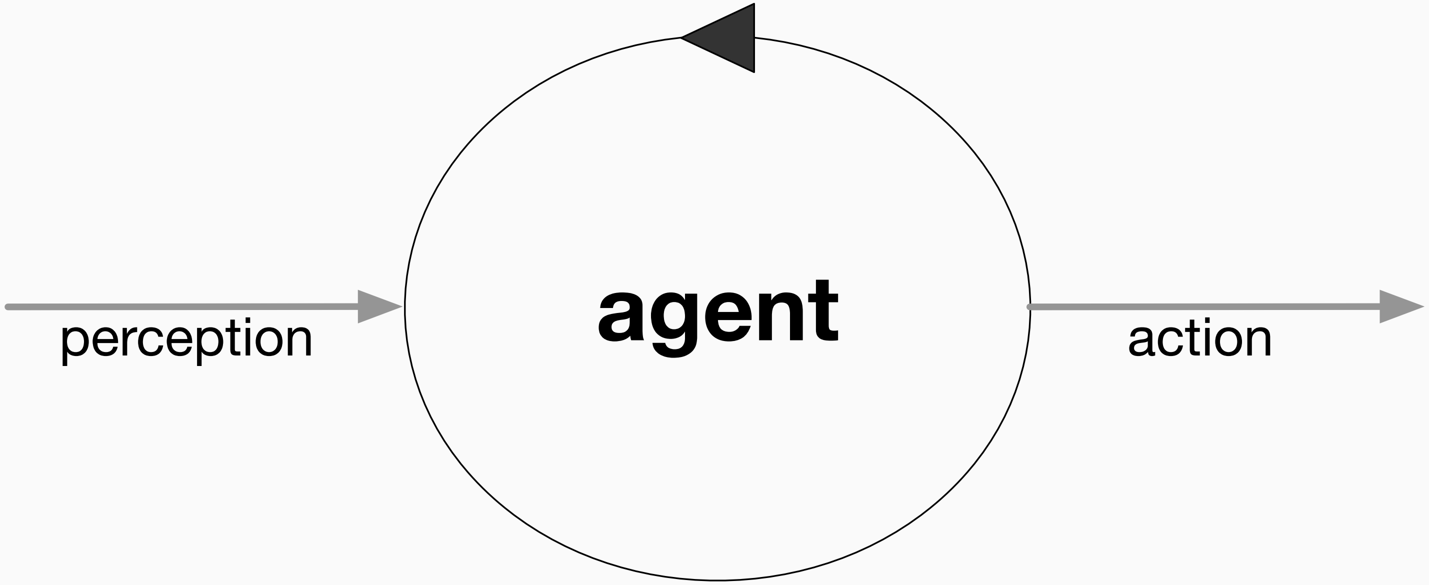
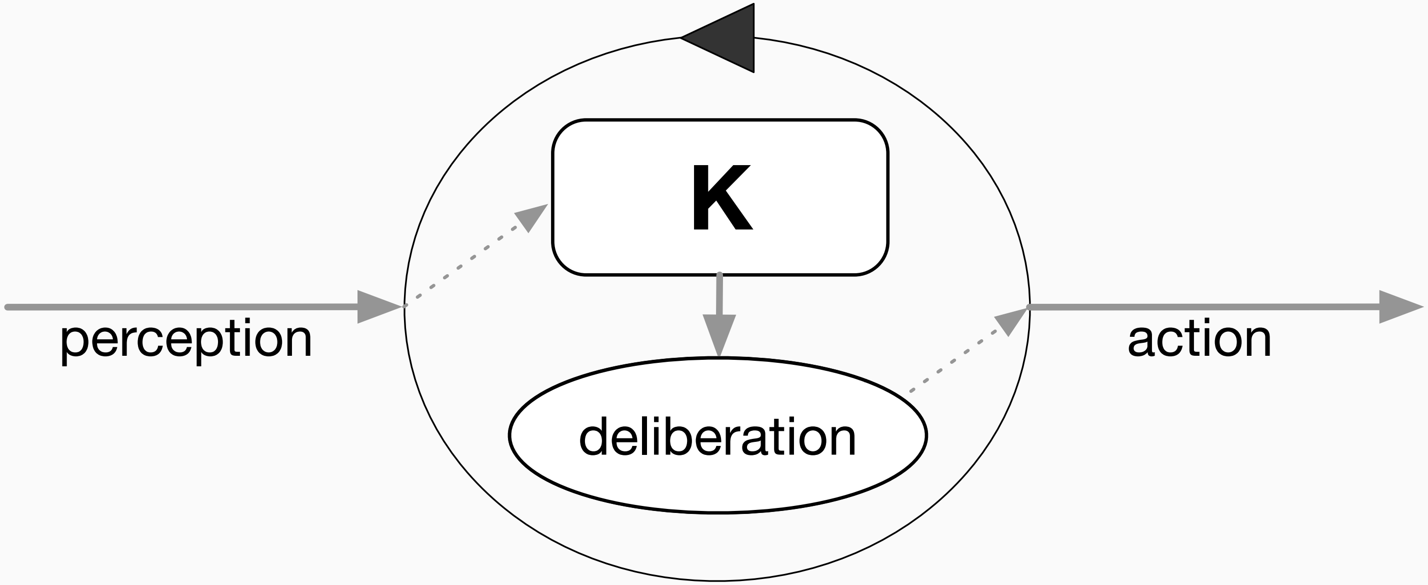
Agent Dimension

Credits: *Slides prepared by Jomi F. Hübner*

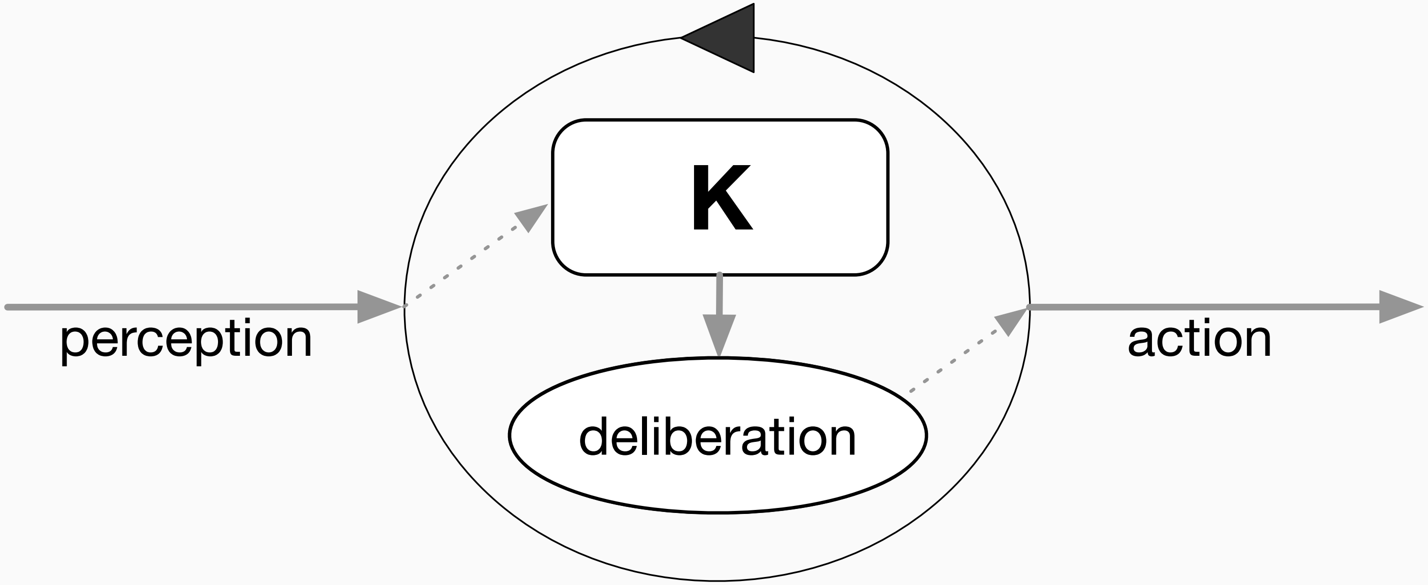
Agent



Agent

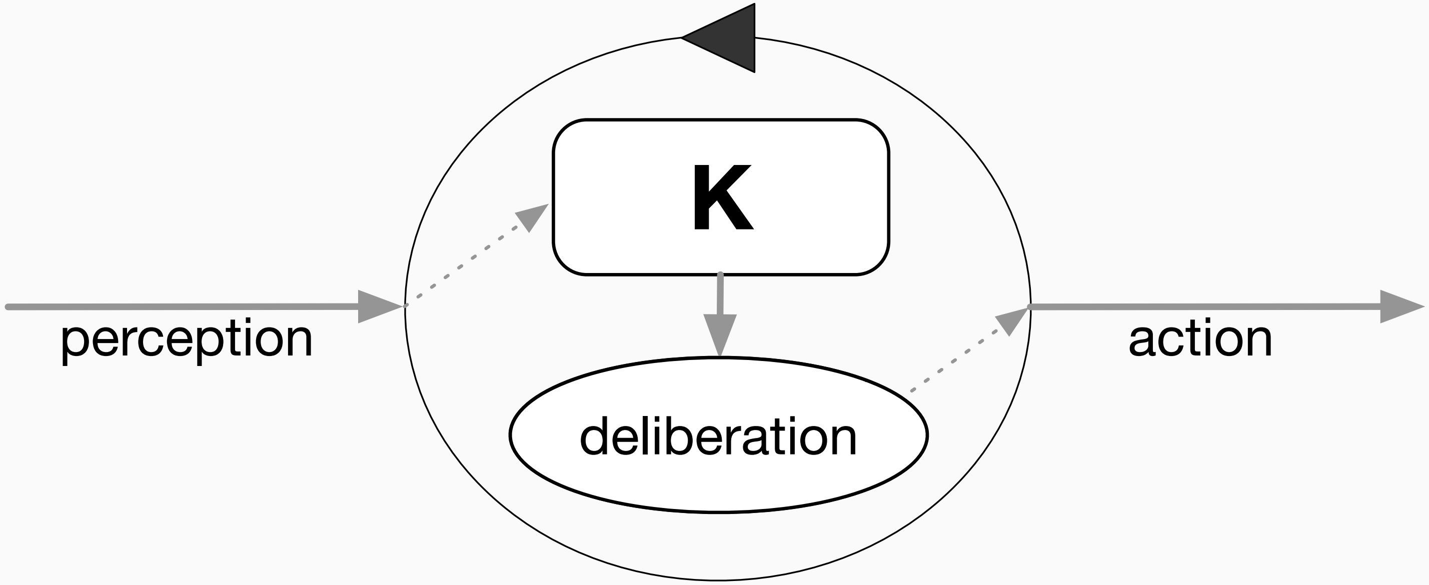


Agent



```
[reasoning cycle]
while true do
    $K \leftarrow K \pm perception()$
    $A \leftarrow deliberation(K)$
    $act(A)$
```

Agent



to **program** an agent is to define K

deliberation ⇝ **autonomy**

Agent Knowledge (in JaCaMo)

Beliefs : information about the environment, other agents, itself, application, ...

```
temperature(20).
happy(bob).
```

Goals : the agent objectives

```
!temperature(20).
!happy(bob).
```

Plans :

Agent Knowledge (in JaCaMo)

Beliefs : information about the environment, other agents, itself, application, ...

```
temperature(20).
happy(bob).
```

Goals : the agent objectives

```
!temperature(20).
!happy(bob).
```

Plans :

Agent Knowledge (in JaCaMo)

Beliefs : information about the environment, other agents, itself, application, ...

```
temperature(20).
happy(bob).
```

Goals : the agent objectives

```
!temperature(20).
!happy(bob).
```

Plans : specifies how goals can be **achieved** by **actions**

```
+!temperature(20) <- startCooling.
+!happy(bob) <- kiss(bob).
```

Agent Knowledge (in JaCaMo)

Beliefs : information about the environment, other agents, itself, application, ...

```
temperature(20).
happy(bob).
```

Goals : the agent objectives

```
!temperature(20).
!happy(bob).
```

Plans : specifies how goals can be **achieved** by **actions**

```
+!temperature(20) <- startCooling.
+!happy(bob) <- kiss(bob).
```

specifies **reactions** to mental state changes

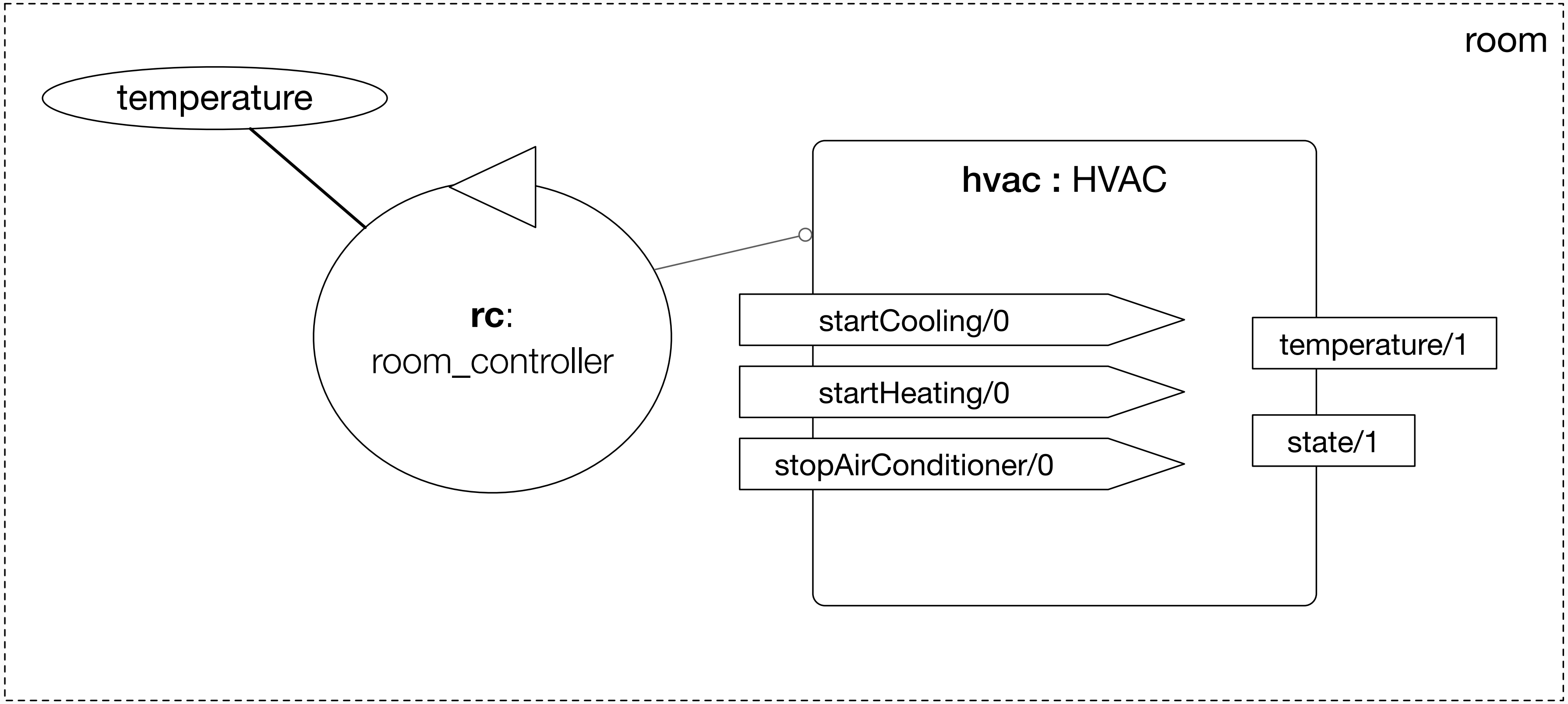
```
+temperature(10) <- !temperature(20).
-happy(bob) <- !happy(bob).
```

Knowledge Sources

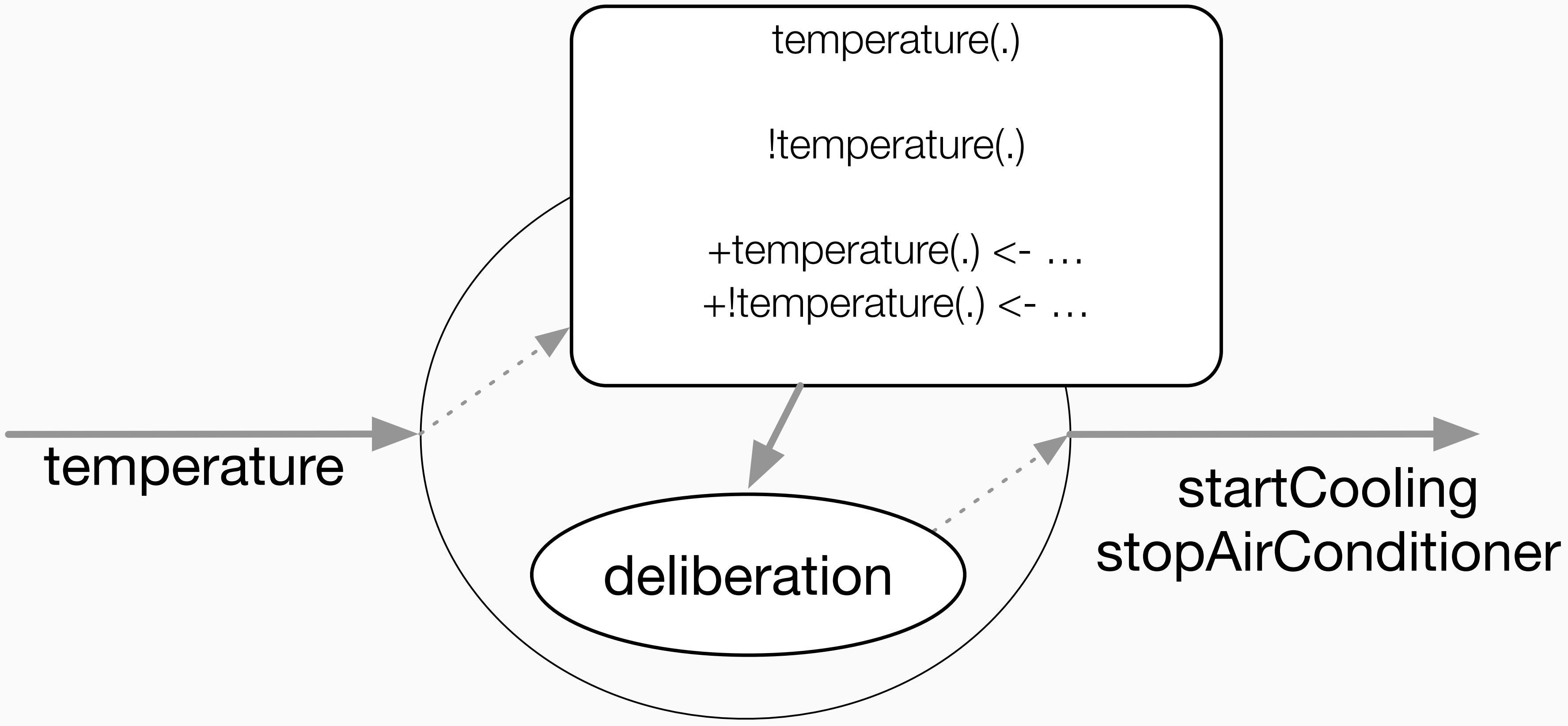
Beliefs, goals, and plans are provided by

- perception: in the case of beliefs
- developers: initial mental state of the agent
- other agents: by communication
- the agent itself: by reasoning or learning

Smart Room Scenario — initial implementation



Agent Programming (in JaCaMo)



Agent Programming (in JaCaMo)

```
+temperature(30)  <- !temperature(20).
+!temperature(20) <- startCooling.
```

Agent Programming (in JaCaMo)

```
+temperature(30)  <- !temperature(20).
+temperature(20)  <- stopAirConditioner.

+!temperature(20) <- startCooling.
```

Agent Programming (in JaCaMo)

```
// initial belief, given by the developer
preference(20).

// reaction to changes in the temperature
+temperature(T)  : preference(P) & math.abs(P-T) > 2
   <- !temperature(P).
+temperature(T)  : preference(T)
   <- stopAirConditioner.

// plans to achieve some temperature
+!temperature(P) : temperature(T) & T >  P
   <- startCooling.
```

Agent Programming (in JaCaMo)

```
// initial belief, given by the developer
preference(20).

// initial goal, given by the developer
!keep_temperature.

// maintenance the goal pattern
+!keep_temperature
    : temperature(T) & preference(P) & T >  P
   <- startCooling;
      !keep_temperature.
+!keep_temperature
    : temperature(T) & preference(P) & T <= P
   <- stopAirConditioner;
      !keep_temperature.
```

Main Features

- **reactivity**: even when achieving some goals
- **pro-activity**: new goals can be created
- **long-term goals**: agents are committed to achieve goals
- **context awareness**: plans are selected based on the circumstances
- **transparency**: we can trace back the reasons for an action
- sound **theoretical background** for agent architectures:
  - practical reasoning [Bratman, 1987]
  - intentions [Cohen and Levesque, 1987]
  - BDI [Rao and Georgeff, 1995]
  - ...

Main Features

- **reactivity**: even when achieving some goals
- **pro-activity**: new goals can be created
- **long-term goals**: agents are committed to achieve goals
- **context awareness**: plans are selected based on the circumstances
- **transparency**: we can trace back the reasons for an action
- sound **theoretical background** for agent architectures:
  - practical reasoning [Bratman, 1987]
  - intentions [Cohen and Levesque, 1987]
  - BDI [Rao and Georgeff, 1995]
  - ...

Main Features

- **reactivity**: even when achieving some goals
- **pro-activity**: new goals can be created
- **long-term goals**: agents are committed to achieve goals
- **context awareness**: plans are selected based on the circumstances
- **transparency**: we can trace back the reasons for an action
- sound **theoretical background** for agent architectures:
    - practical reasoning [Bratman, 1987]
    - intentions [Cohen and Levesque, 1987]
    - BDI [Rao and Georgeff, 1995]
    - ...

Main Features

- **reactivity**: even when achieving some goals
- **pro-activity**: new goals can be created
- **long-term goals**: agents are committed to achieve goals
- **context awareness**: plans are selected based on the circumstances
- **transparency**: we can trace back the reasons for an action
- sound **theoretical background** for agent architectures:
  - practical reasoning [Bratman, 1987]
  - intentions [Cohen and Levesque, 1987]
  - BDI [Rao and Georgeff, 1995]
  - ...

Main Features

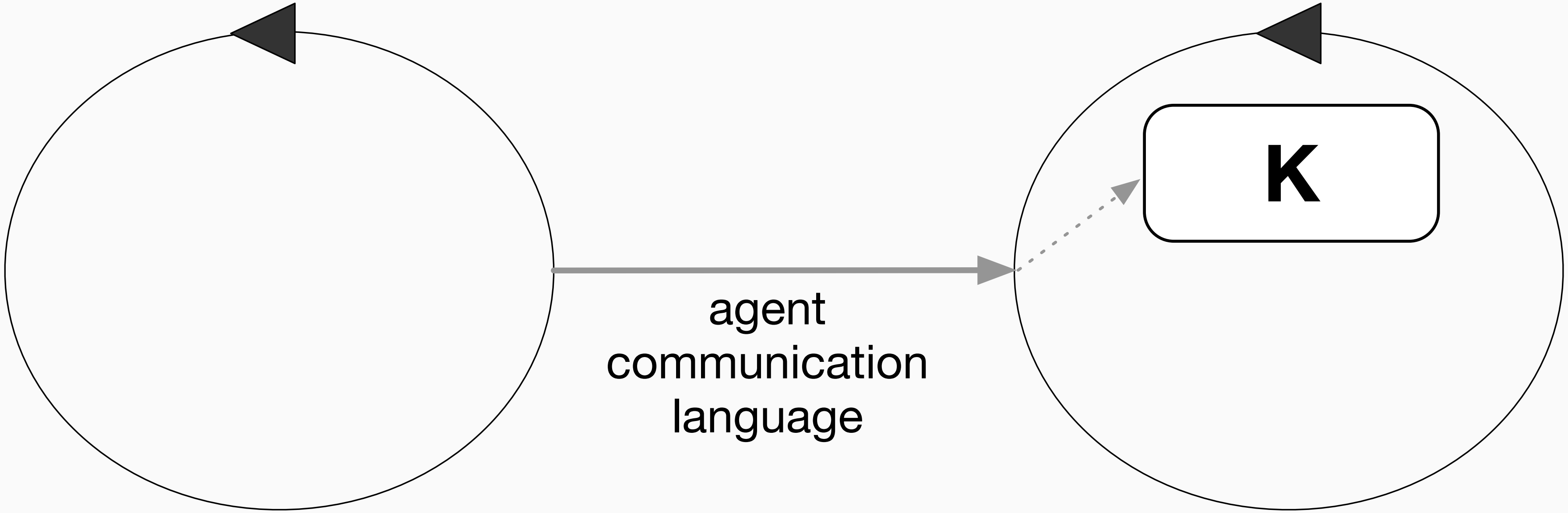
- **reactivity**: even when achieving some goals
- **pro-activity**: new goals can be created
- **long-term goals**: agents are committed to achieve goals
- **context awareness**: plans are selected based on the circumstances
- **transparency**: we can trace back the reasons for an action
- sound **theoretical background** for agent architectures:
  - practical reasoning [Bratman, 1987]
  - intentions [Cohen and Levesque, 1987]
  - BDI [Rao and Georgeff, 1995]
  - ...

Main Features

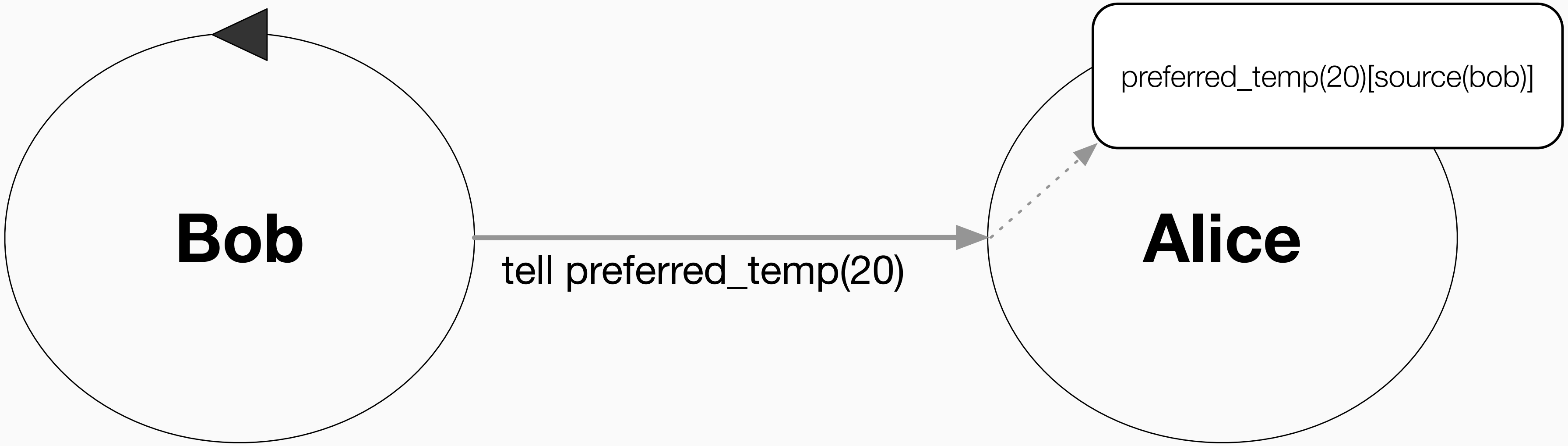
- **reactivity**: even when achieving some goals
- **pro-activity**: new goals can be created
- **long-term goals**: agents are committed to achieve goals
- **context awareness**: plans are selected based on the circumstances
- **transparency**: we can trace back the reasons for an action
- sound **theoretical background** for agent architectures:
  - practical reasoning [Bratman, 1987]
  - intentions [Cohen and Levesque, 1987]
  - BDI [Rao and Georgeff, 1995]
  - ...

Agent Interaction (communication)

Agent–Agent Communication



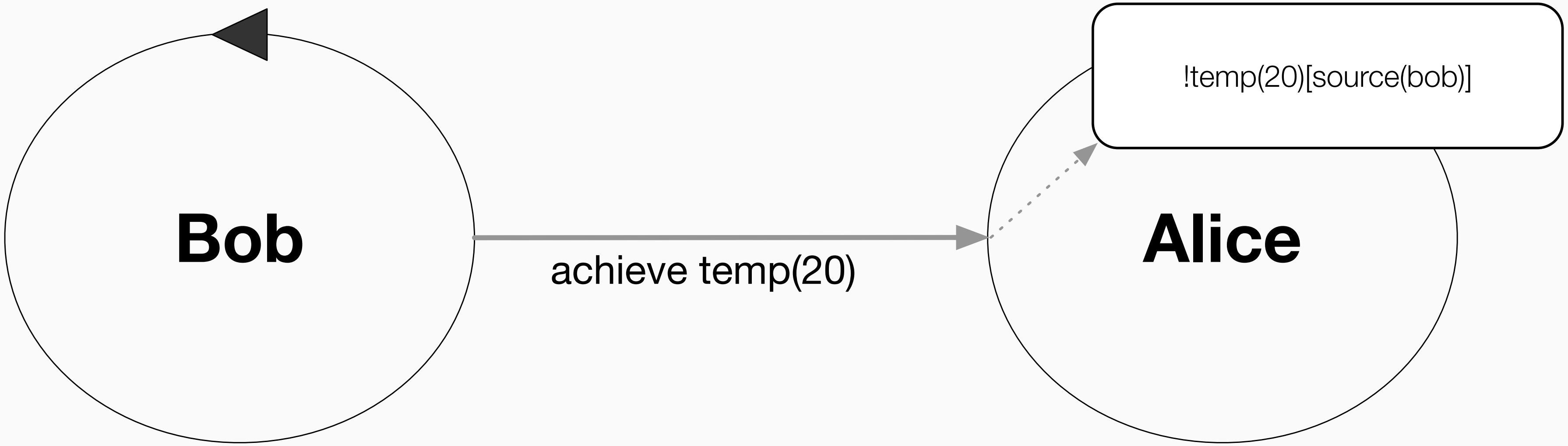
Semantic of messages



A message has:

- an intention (tell, ask, achieve, …)
- a content (belief, goal, plan)

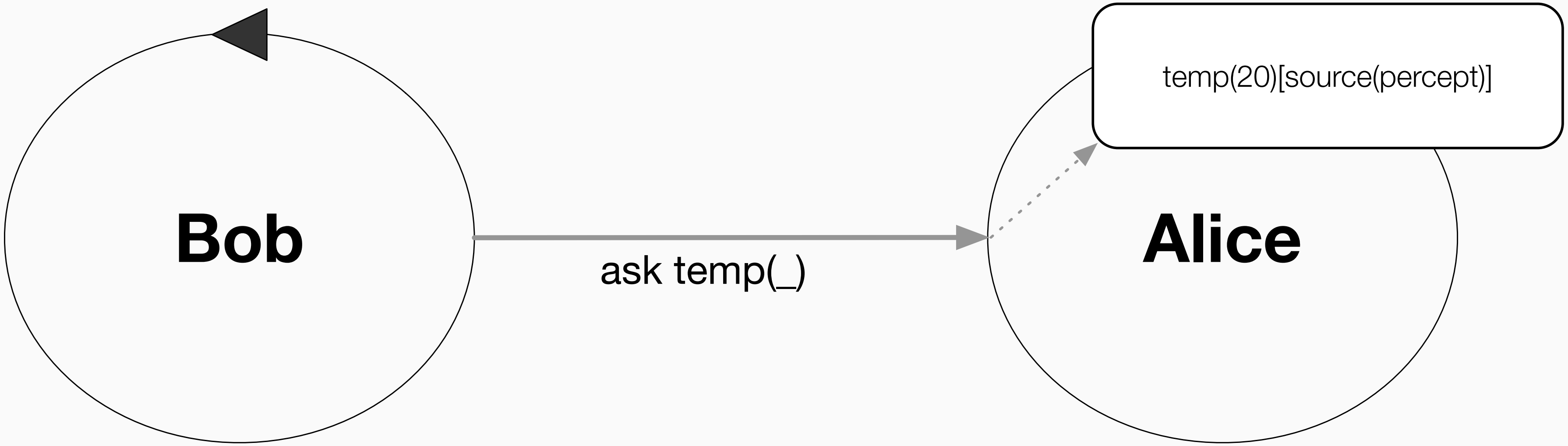
Semantic of messages



A message has:

- an intention (tell, ask, achieve, ...)
- a content (belief, goal, plan)

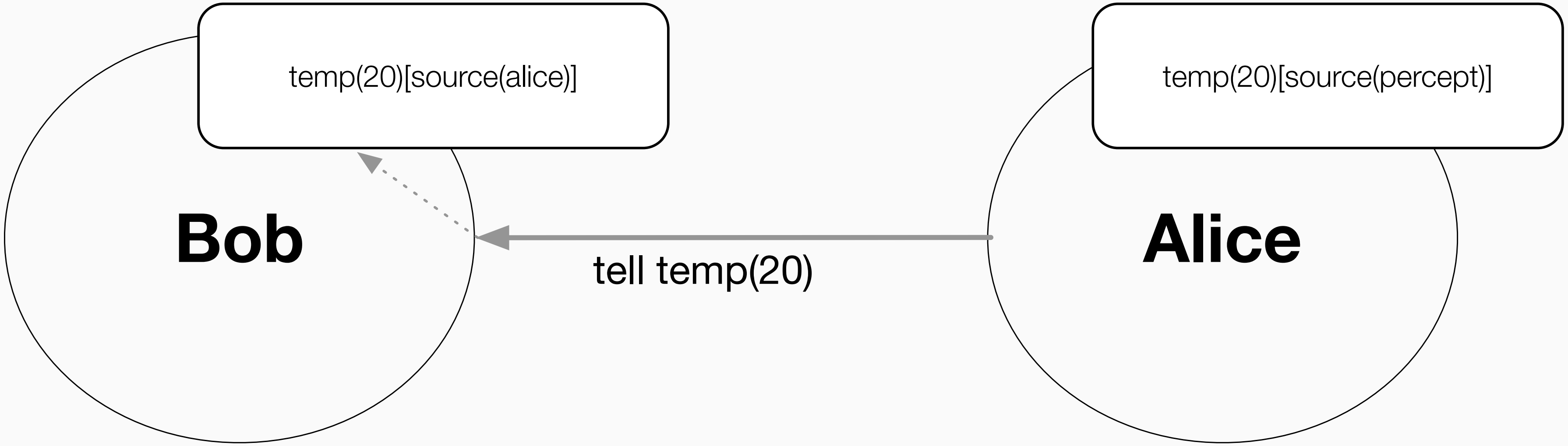
Semantic of messages



A message has:

- an intention (tell, ask, achieve, ...)
- a content (belief, goal, plan)

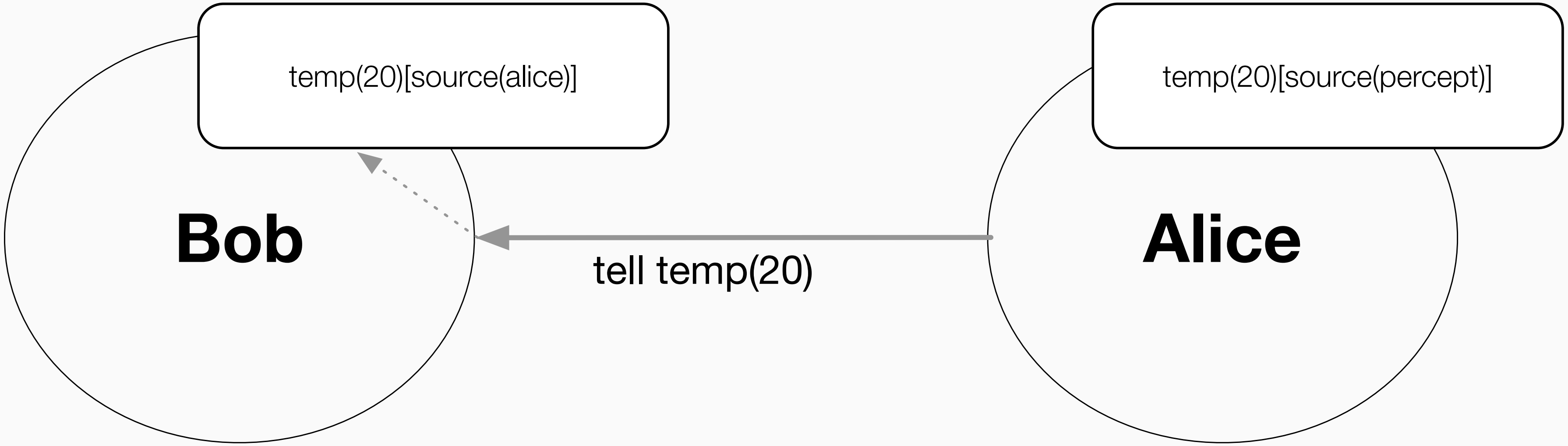
Semantic of messages



A message has:

- an intention (tell, ask, achieve, ...)
- a content (belief, goal, plan)

Semantic of messages



- we are not programming computers,
  we are programming agents, which are based on knowledge
- communication is not about data exchange, but
  knowledge sharing

JaCaMo implementation

Sender: `.send(bob,tell,happy(alice))`

- receiver: agent unique name
- performative: tell, achieve, askOne, askHow, ...
- content: a literal

Receiver

- nothing is needed

Properties

- distributed & support for decentralized
- (usually) asynchronous
- KQML vs FIPA-ACL
- not reduced to method invocation

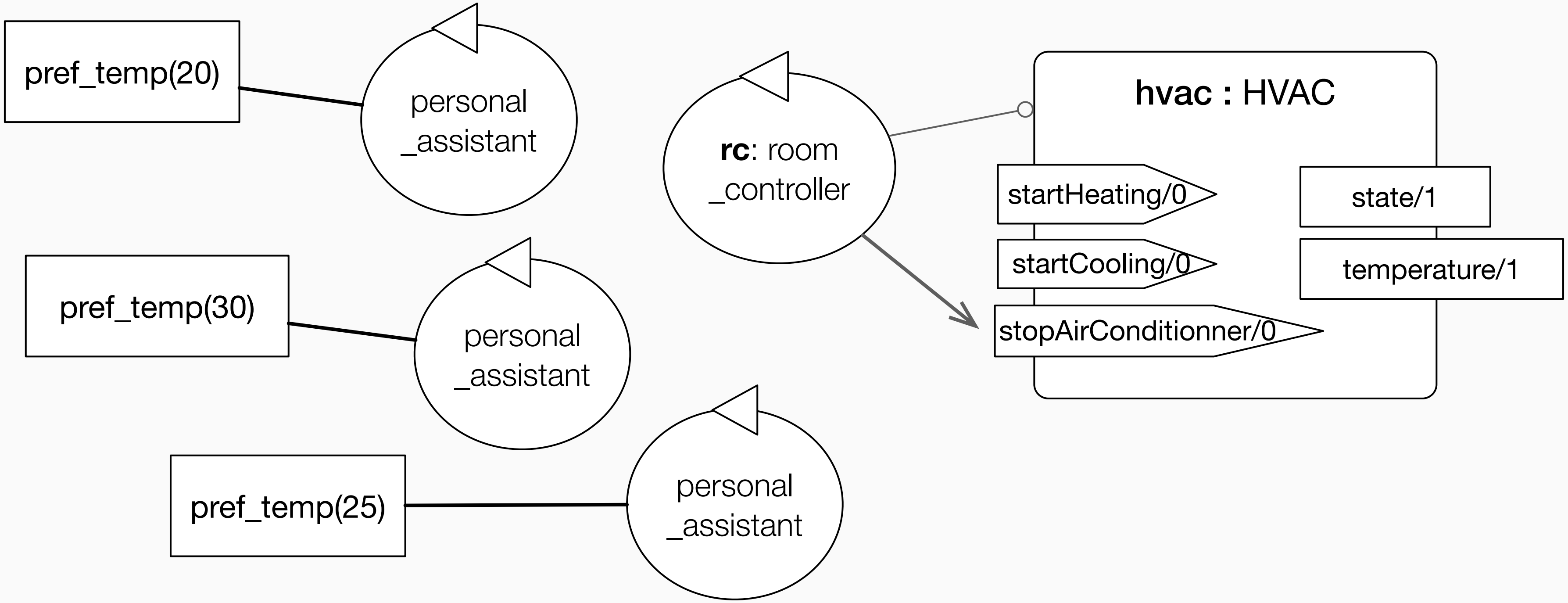
JaCaMo Performatives

- **tell** and untell: change beliefs of receiver
- **achieve** and unachieve: change goals of receiver
- **askOne** and askAll: ask for beliefs of the receiver
- **askHow**, tellHow, and untellHow: exchange plans with other agent
- **signal**: add an event in the receiver

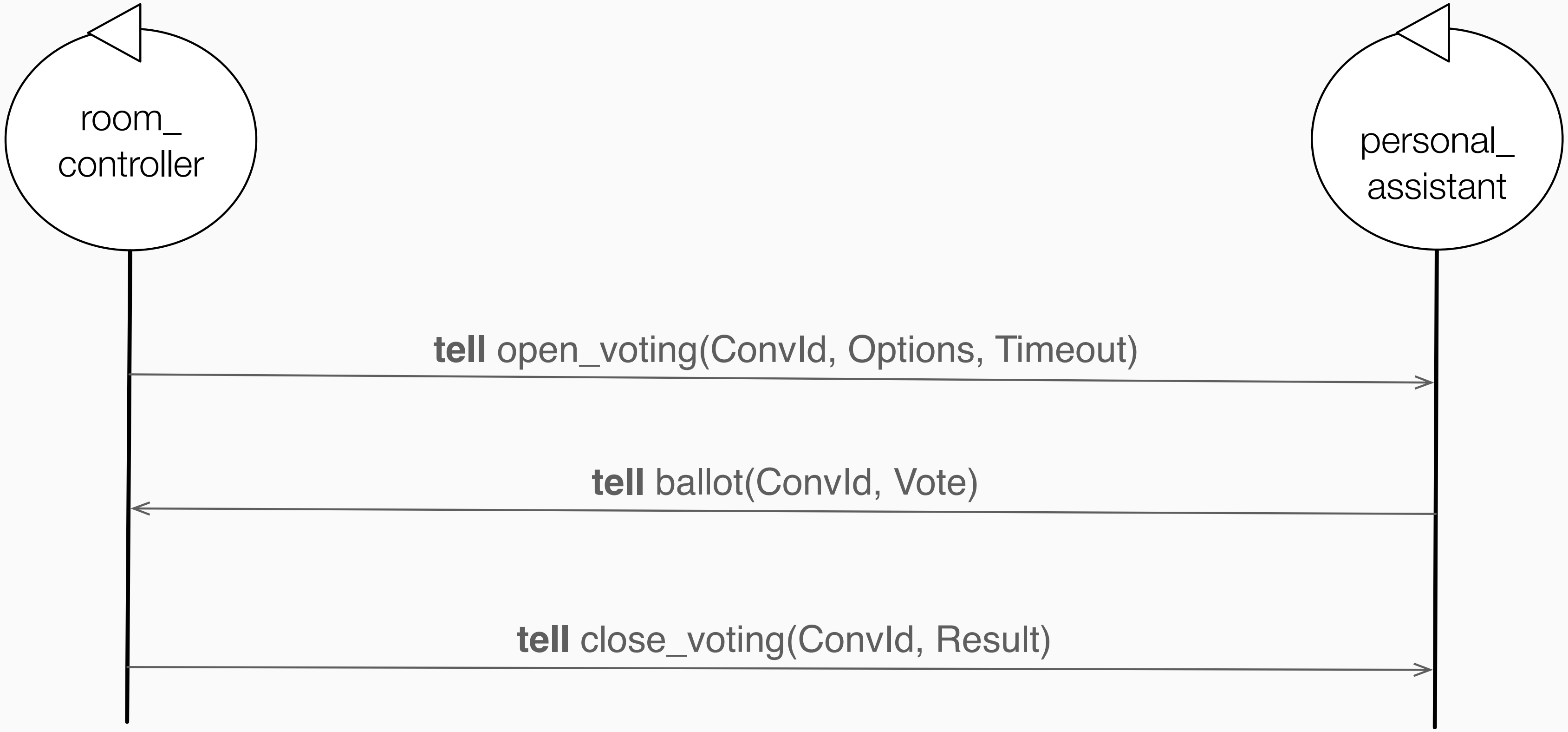
Smart Room Scenario

many users

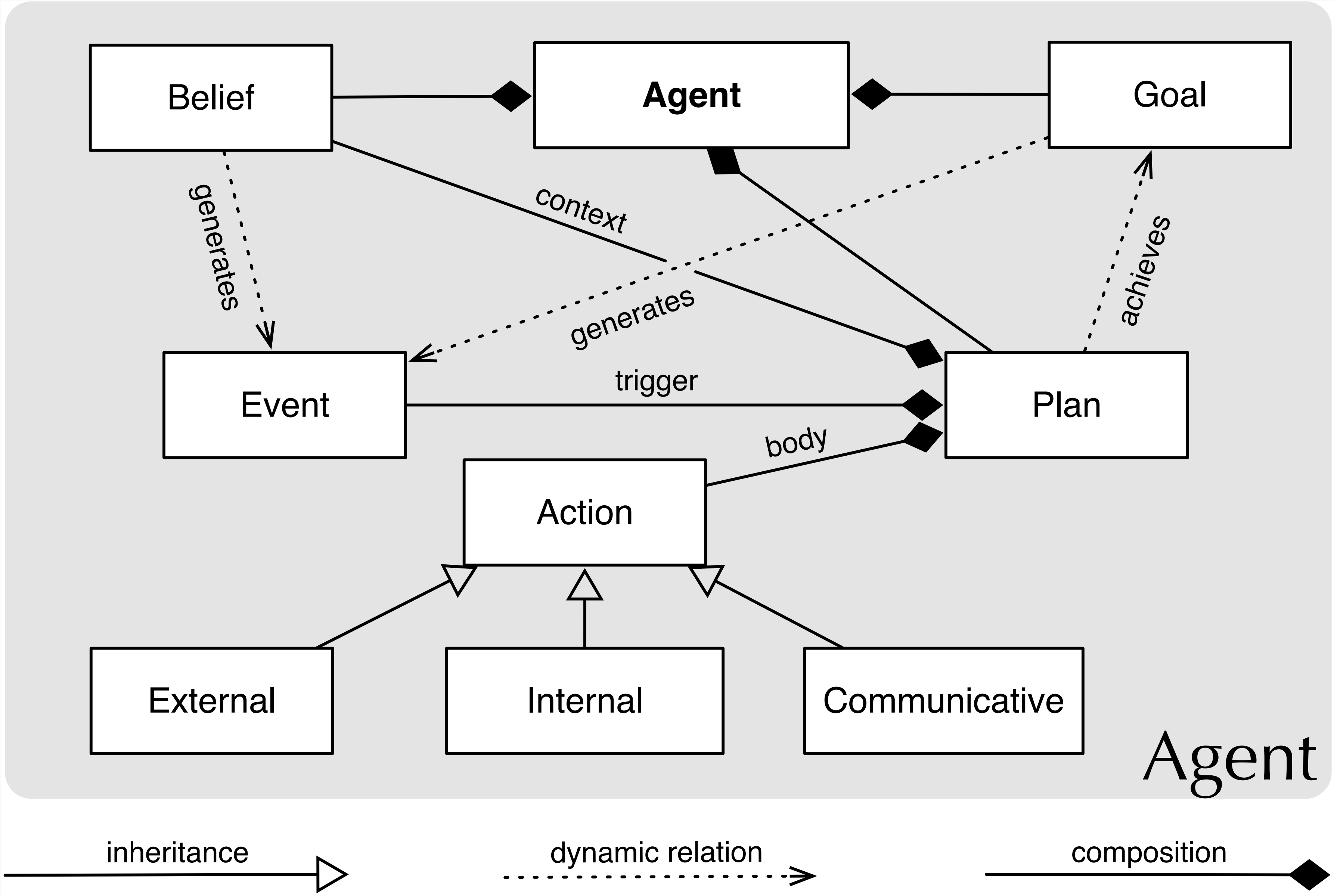
The system have to consider the preference of temperature of many users and use a voting strategy to define the target temperature



Interaction Protocols ⇝ *coordination*


room_
controller
personal_
assistant
tell open_voting(ConvId, Options, Timeout)
tell ballot(ConvId, Vote)
tell close_voting(ConvId, Result)

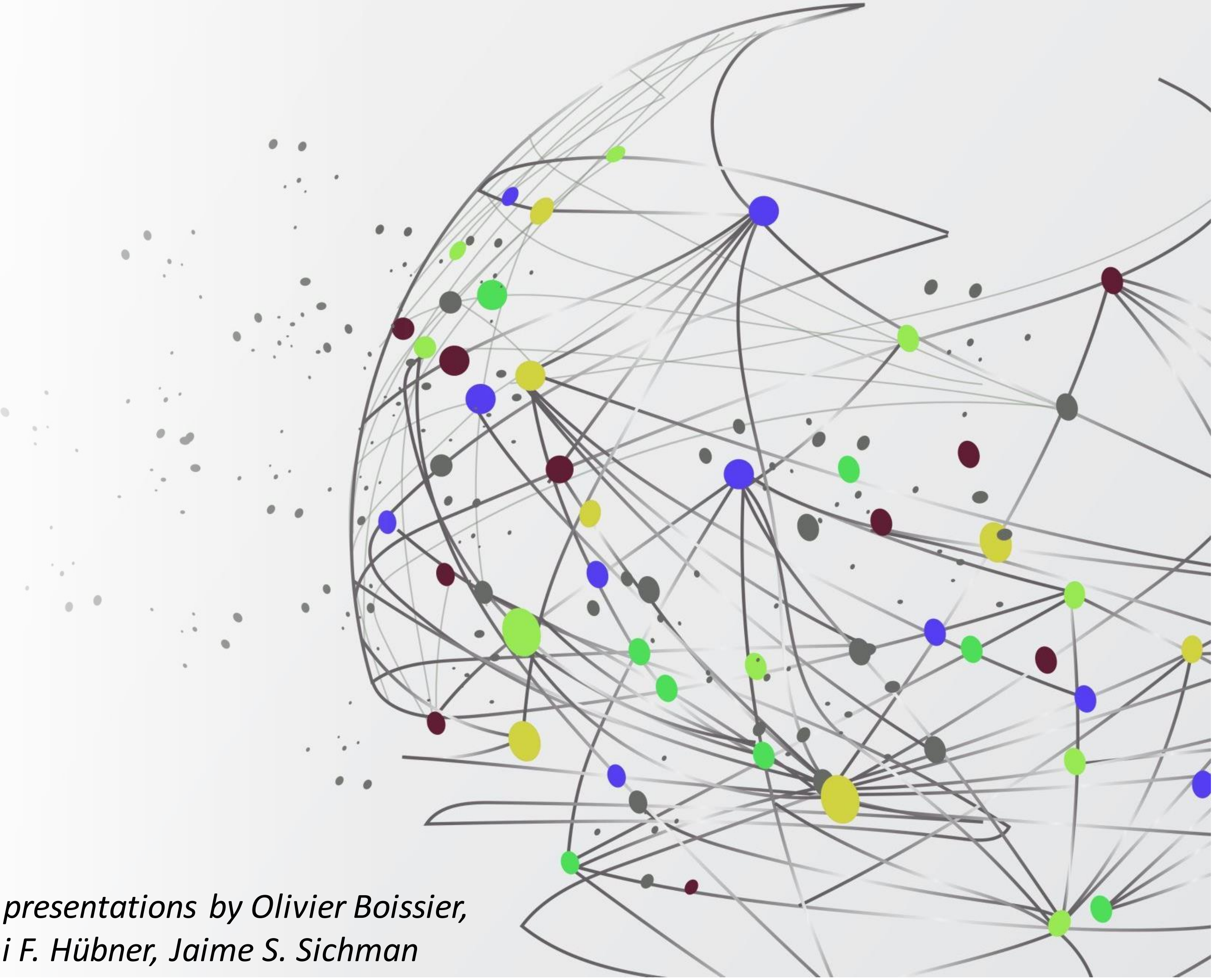
Wrap-up: Agent Model



Wrap-up: Agent Programming

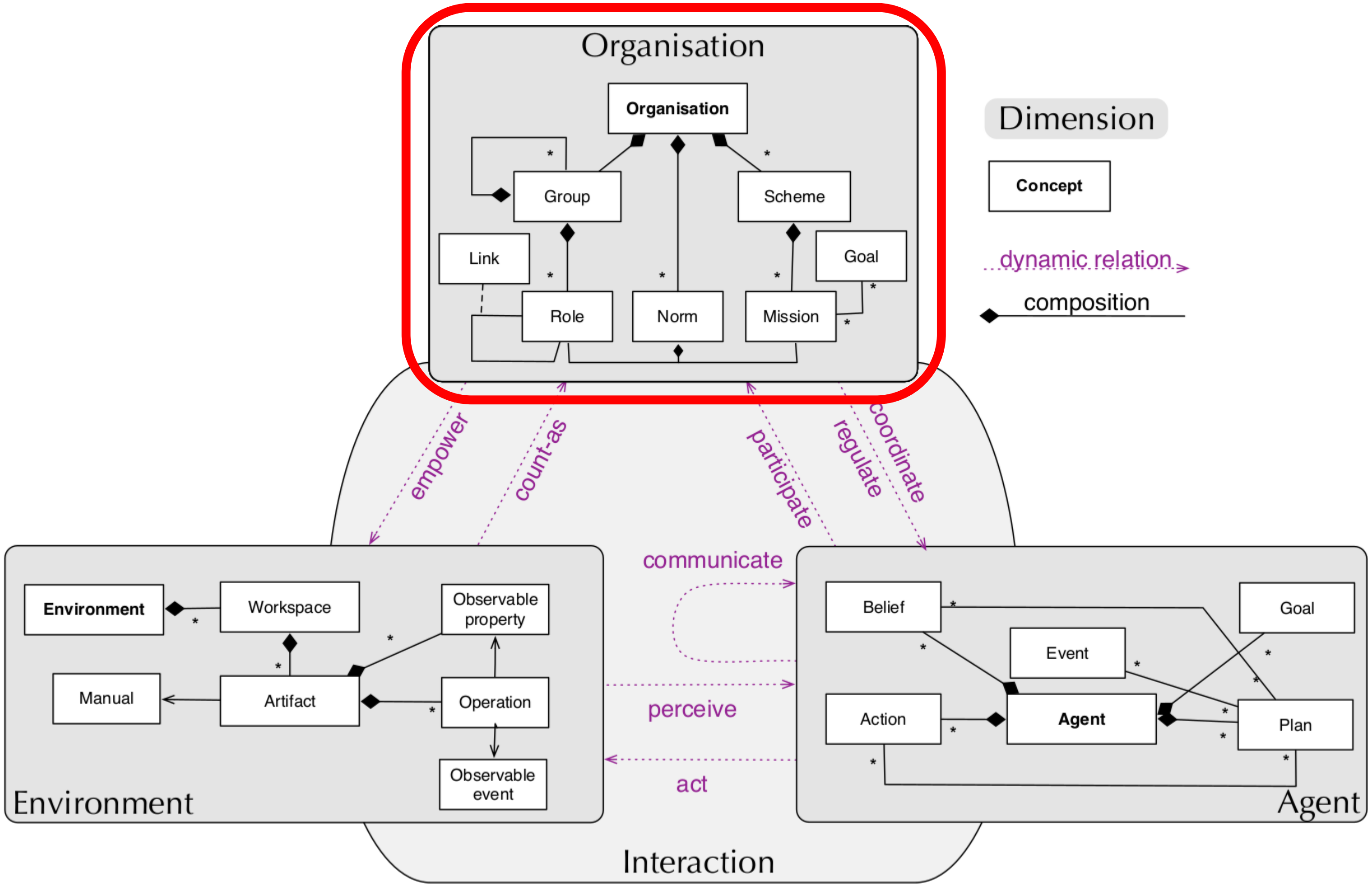
- **AgentSpeak**
  - Logic + BDI
  - Agent programming language
- ***Jason***
  - AgentSpeak interpreter
  - Implements the operational semantics of AgentSpeak
  - Speech-act based communicaiton
  - Highly customisable
  - Useful tools
  - Open source

Organization Dimension



Credits: *Slides are based on previous presentations by Olivier Boissier, Rafael Bordini, Maiquel de Brito, Jomi F. Hübner, Jaime S. Sichman*

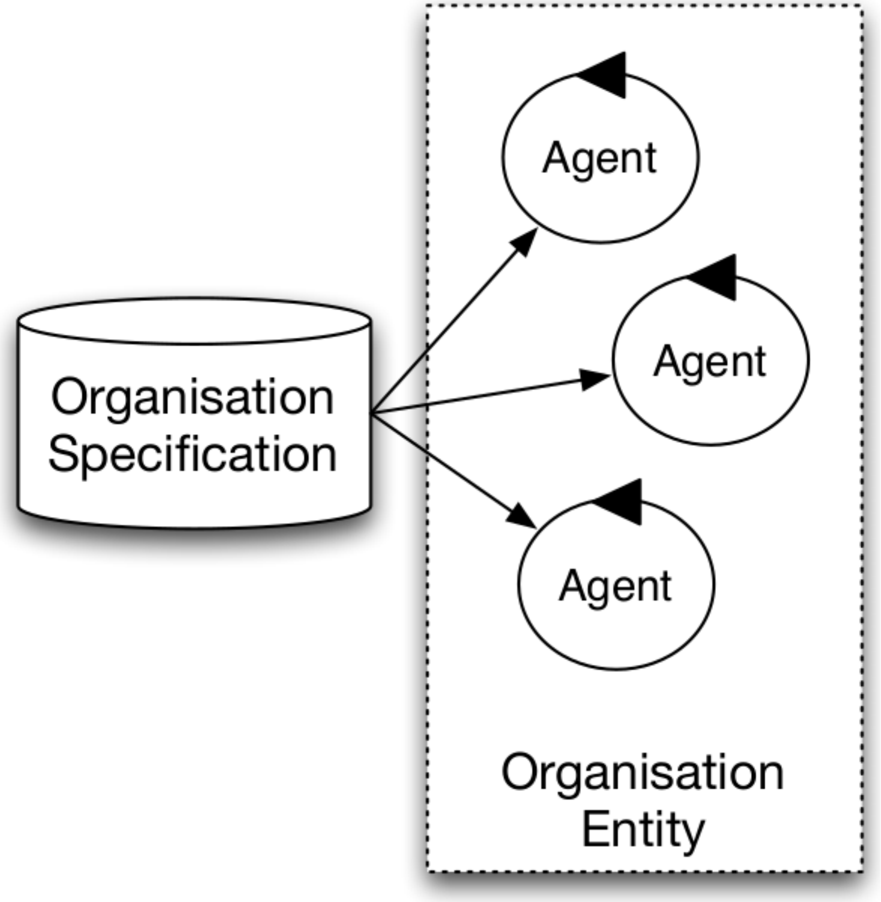
JaCaMo Metamodel – Multi-Agent Concepts



Organization in MAS

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

Organization Oriented Programming



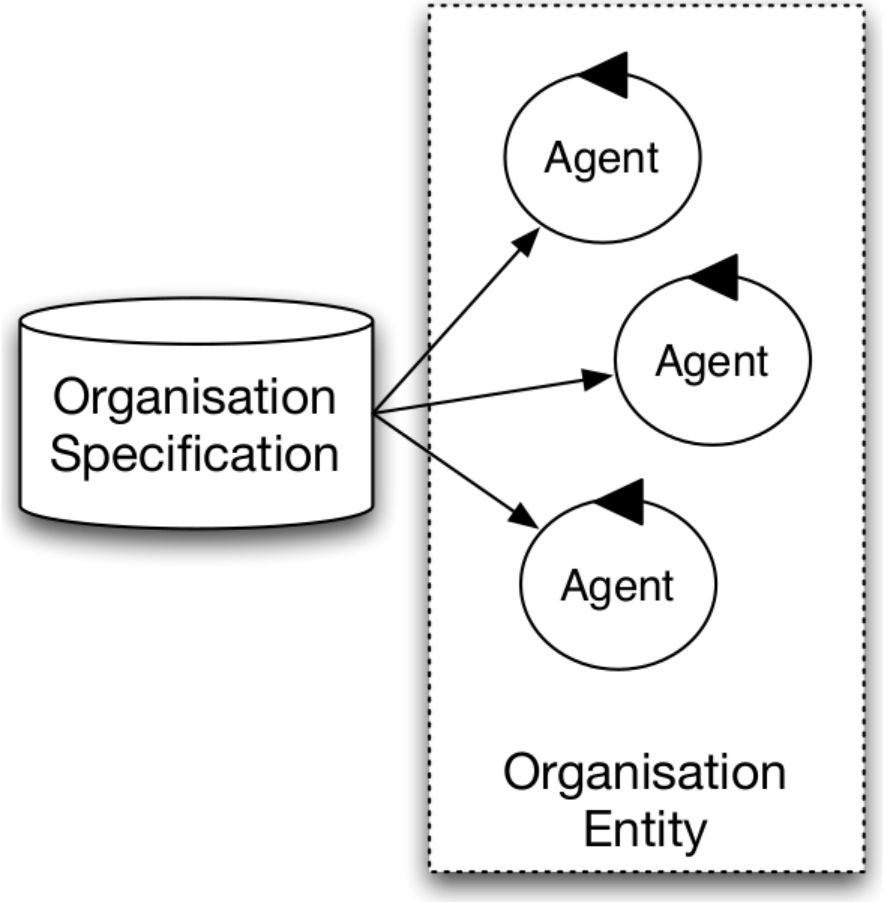
Programming MAS = Programming **Agents** + Programming the **Environment** + Programming the **Organization**

Programming **outside the agents** using of **organizational concepts** to **coordinating and regulating** autonomous agents

```
Program = Specification
```

By changing the specification, we can change the MAS behavior

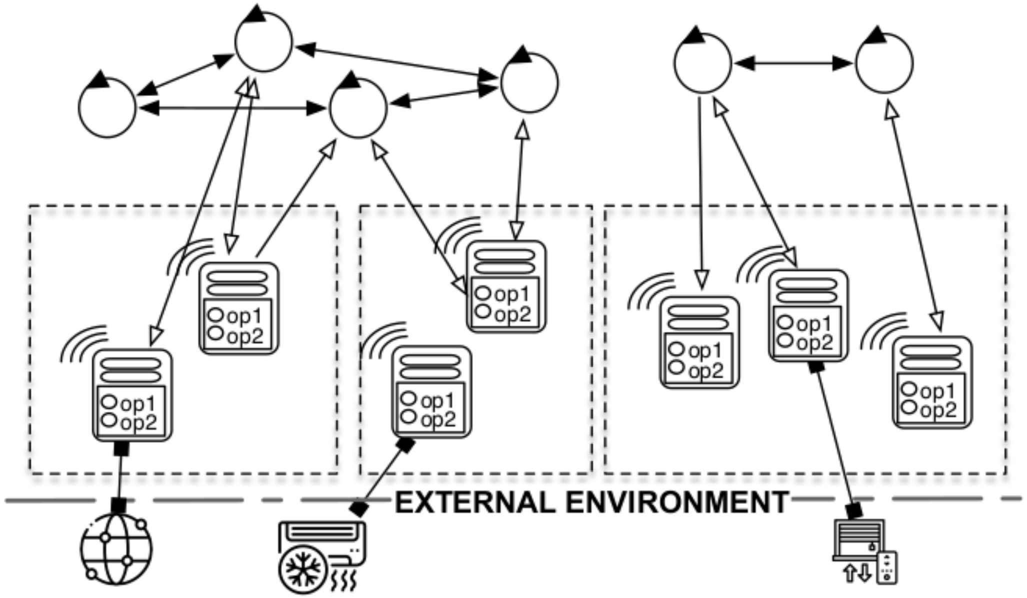
Organization Oriented Programming



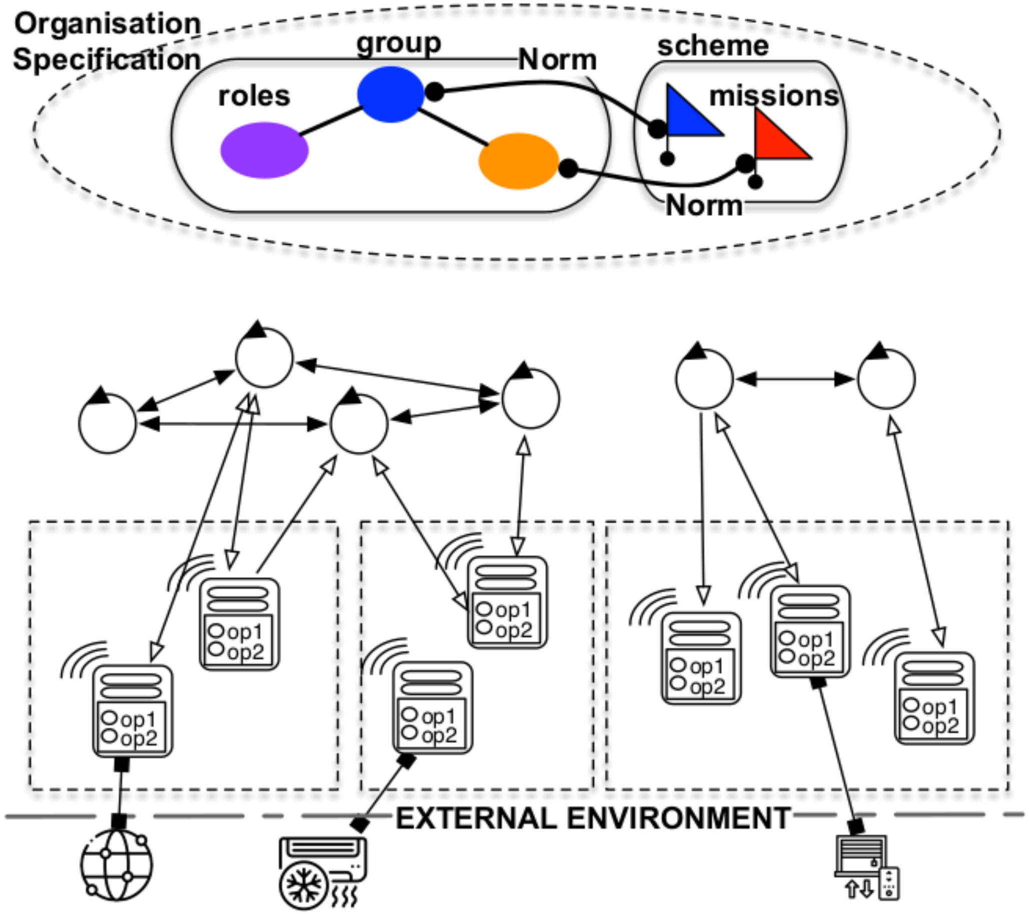
Components

1. Programming language
2. Organization Management Infrastructure
3. Integration to agent architectures and to the environment

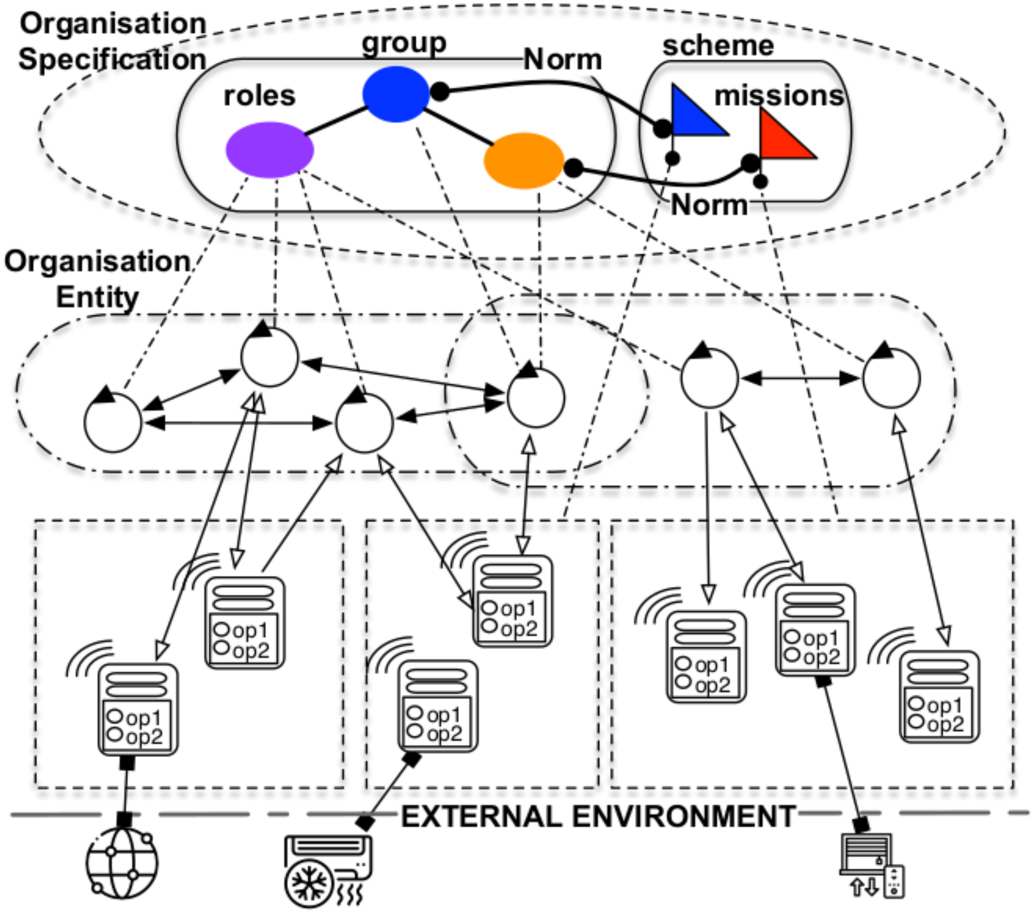
JaCaMo Organization Dimension



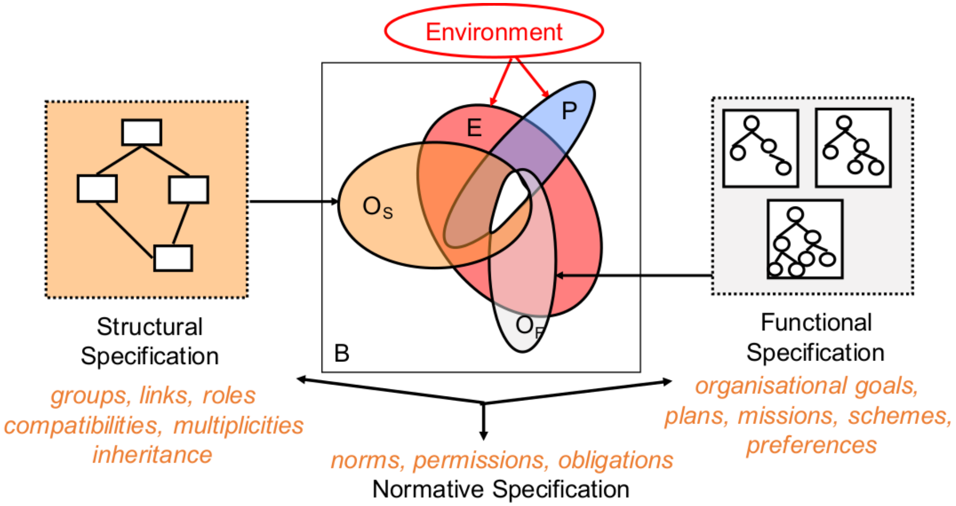
JaCaMo Organization Dimension



JaCaMo Organization Dimension



JaCaMo Organization Dimension

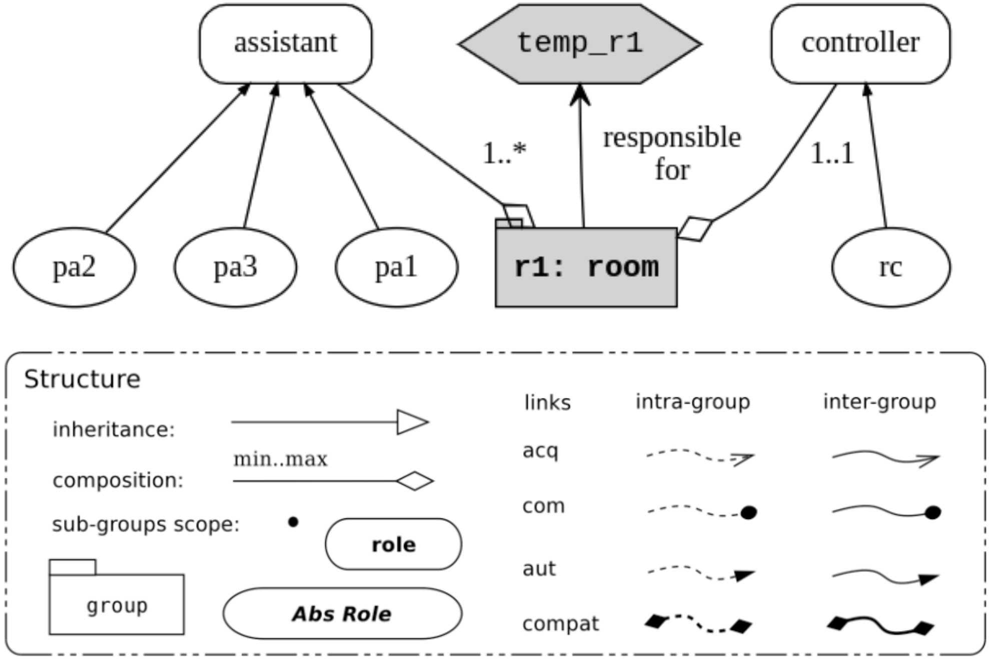

Environment
E
P
Os
OF
B
Structural Specification
groups, links, roles compatibilities, multiplicities inheritance
Functional Specification
organisational goals, plans, missions, schemes, preferences
norms, permissions, obligations
Normative Specification

- Dimensions (Hübner et al. 2007)
  - **Structural** (i.e., Roles, Groups),
  - **Functional** (i.e., Organizational Goals, Missions, Schemes)
  - **Normative** (i.e., Norms with obligations, permissions, interdictions)
- Enable agent's autonomy w.r.t. organization (enforcement vs regimentation)
- Programming and representing the organization
  - make it accessible to the designers, the agents, the coordination and regulation management infrastructure (Hübner et al., 2010)

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 rights and 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 Example

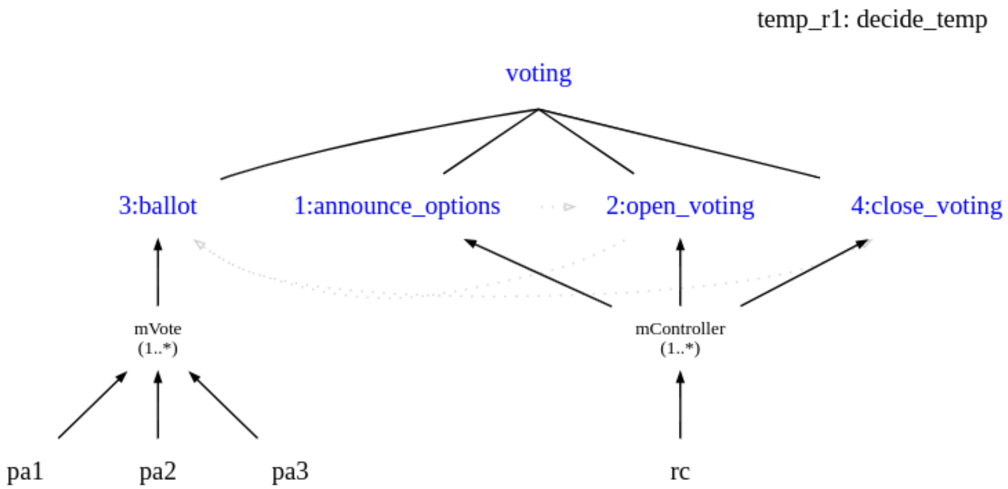


```
<structural-specification>
    <group-specification id="room">
        <roles>
            <role id="assistant" min="1" />
            <role id="controller" min="1" max="1" />
        </roles>
    </group-specification>
</structural-specification>
```

Functional Specification

- Specifies the expected behavior of an MAS in terms of **goals** along two levels:
  - **Collective** with **Scheme**
  - **Individual** with **Mission**
- Components:
  - **Goals**:
    - **Achievement goal** (default type). Goals of this type should be declared as satisfied by the agents committed to them, when achieved
    - **Maintenance goal**. Goals of this type are not satisfied 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 Example

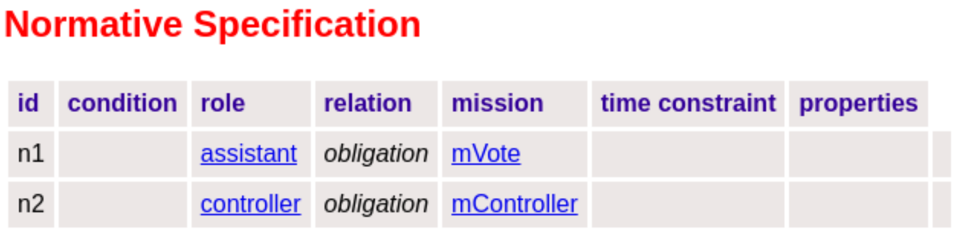


```
<functional-specification>
    <scheme id="decide_temp">
        <goal id="voting">
            <plan operator="sequence">
                <goal id="announce_options" />
                <goal id="open_voting" />
                <goal id="ballot" ttf="10 seconds">
                    <argument id="voting_machine_id"
                    />
                </goal>
                <goal id="close_voting" />
            </plan>
        </goal>
        <mission id="mVote" min="1">
            <goal id="ballot" />
        </mission>
        <mission id="mController" min="1">
            <goal id="announce_options" />
            <goal id="open_voting" />
            <goal id="close_voting" />
        </mission>
    </scheme>
</functional-specification>
```

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 Example



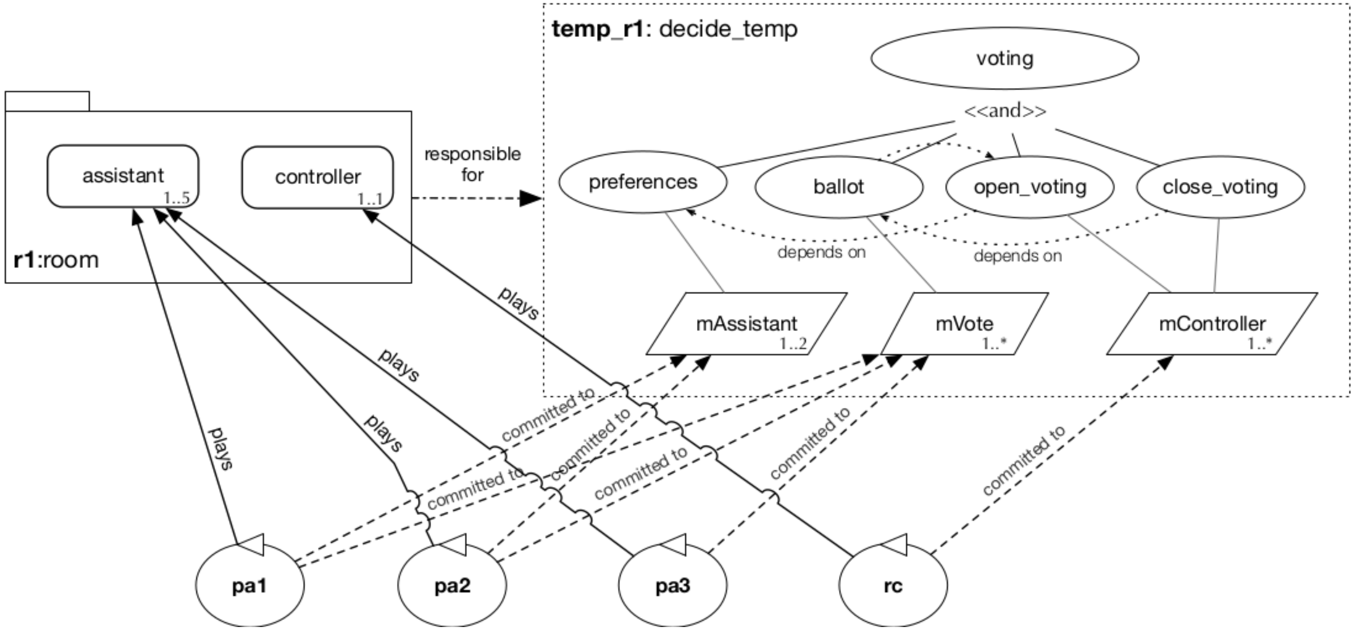
Normative Specification

id	condition	role	relation	mission	time constraint	properties
n1		assistant	*obligation*	mVote		
n2		controller	*obligation*	mController		

```
<normative-specification>
  <norm id="n1" type="obligation"
  role="assistant" mission="mVote" />

  <norm id="n2" type="obligation"
  role="controller" mission="mController" />
</normative-specification>
```

Declarative Organization Programming



- Structural patterns (groups (r1:room), roles (assistant, controller), links)
- Coordination patterns (
  - goal decomposition trees (voting, preferences, ballot, open_voting, close_voting)
  - missions (mAssistant, mVote,mController)
- Rights and duties (norms)

Organization Dynamics

In the context of Organization life-cycle

- Creation/Deletion of an Organization from an Organization specification
- Entrance/Exit of an agent
- Change of Organization specification

In the context of Organization structure life-cycle

- Creation/Deletion of a group
- Adoption/Leave of a role

In the context of Coordination activity life-cycle

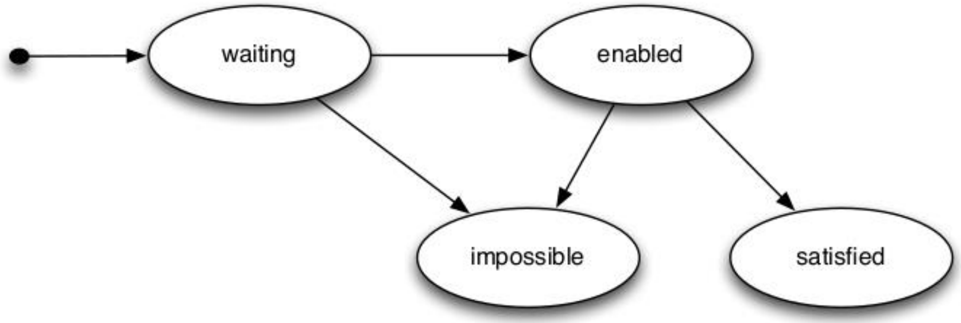
- Creation/End of a schema
- Commitment/Release of a mission
- Change of goal state

In the context of Normative Regulation activity life-cycle

- Activation/De-activation of norms
- Fulfillment/Violation of norms
- Enforcement of norms

Organization Dynamics

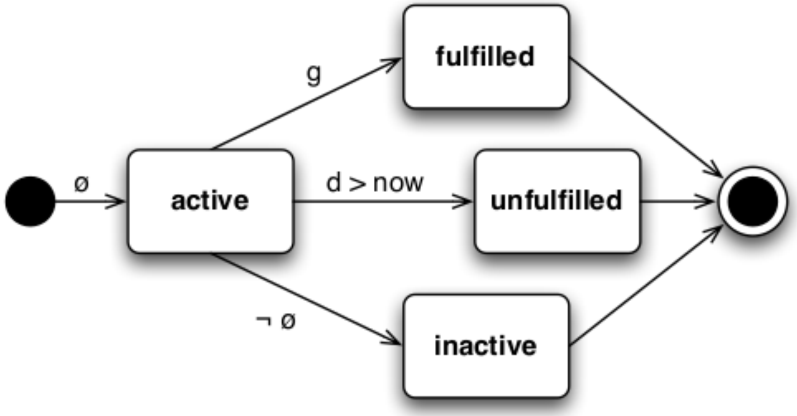
Organization Goal Dynamics



waiting initial state
enabled goal pre-conditions are satisfied and
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

Norm Dynamics

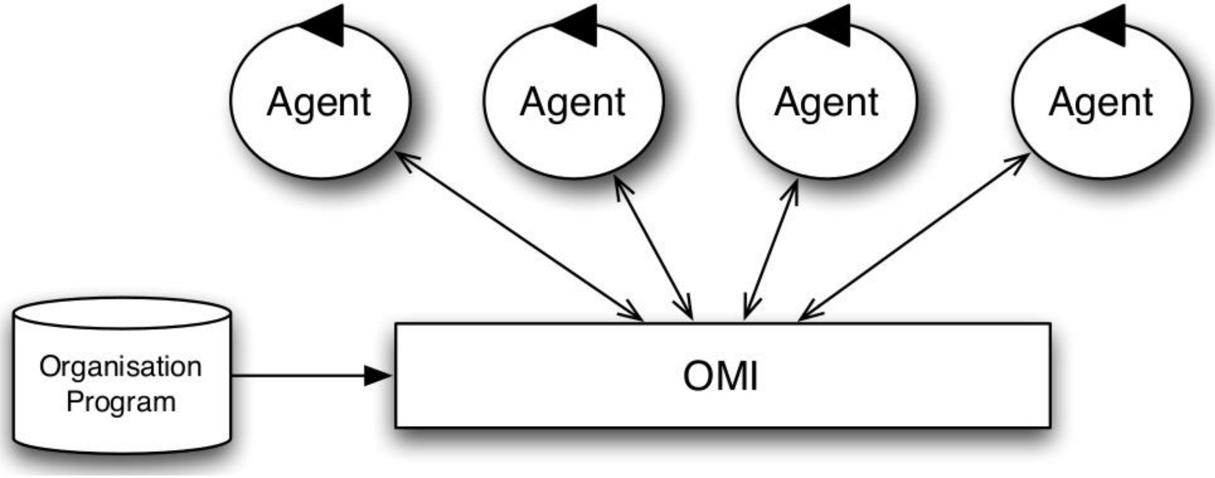


norm n : ϕ -> obligation(a, r, g, d)

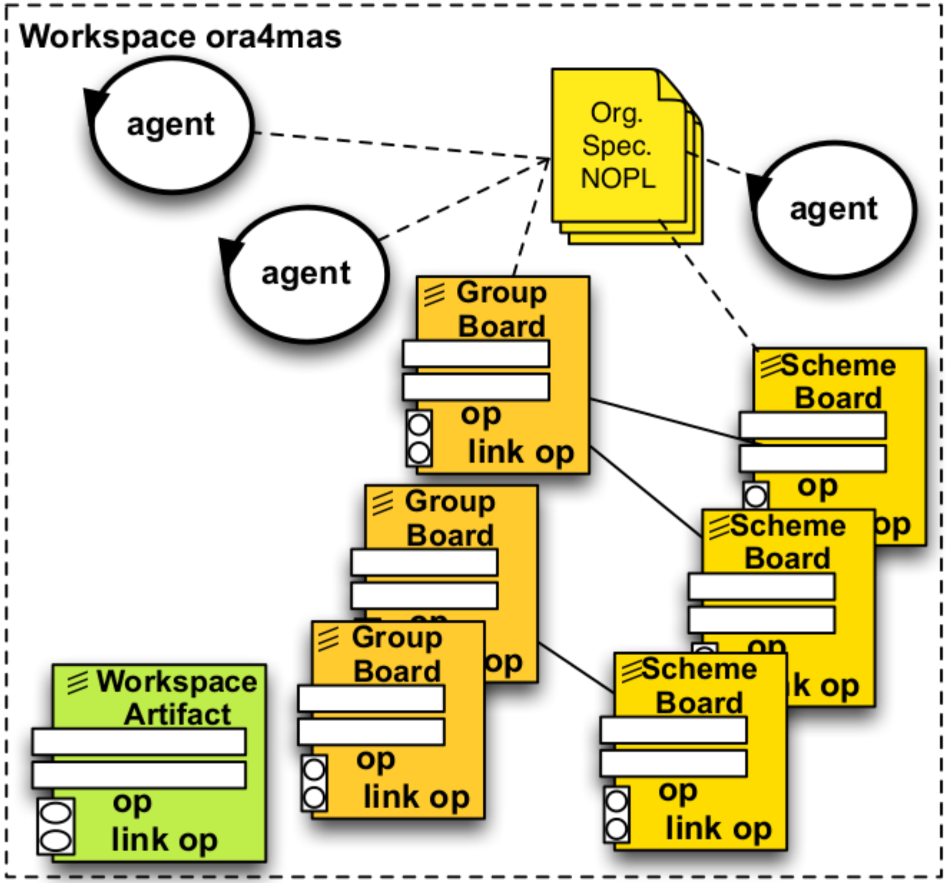
ϕ: activation condition of the norm (e.g., play a role)
g: the goal of the obligation (e.g., commit to a mission)
d: the deadline of the obligation

Organization Management Infrastructure (OMI)

Managing – coordination, regulation – the agents' execution within organization defined in an organization specification

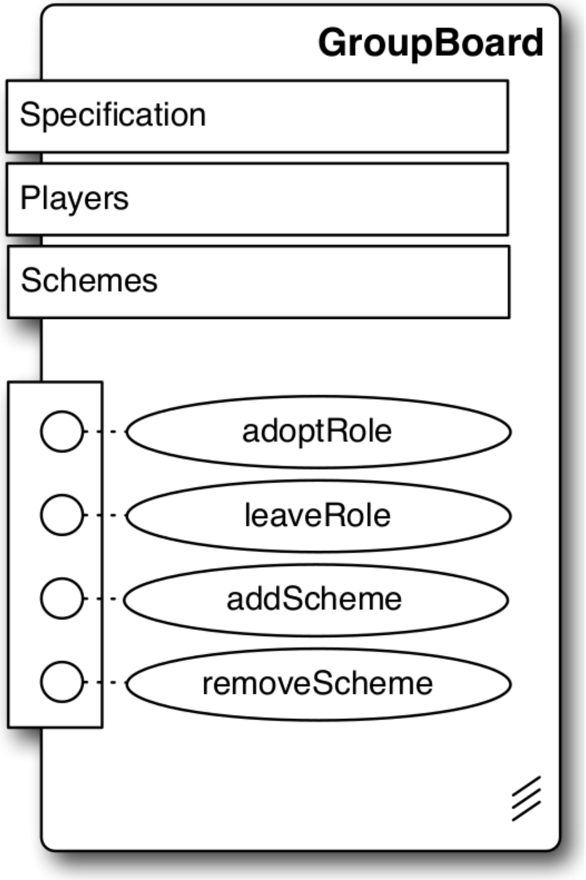


Organizational Artifacts in JaCaMo



- based on A&A and Moise
- agents create and handle organizational artifacts
- artifacts in charge of regimentations, detection and evaluation of norms compliance
- agents are in charge of decisions about sanctions
- distributed solution

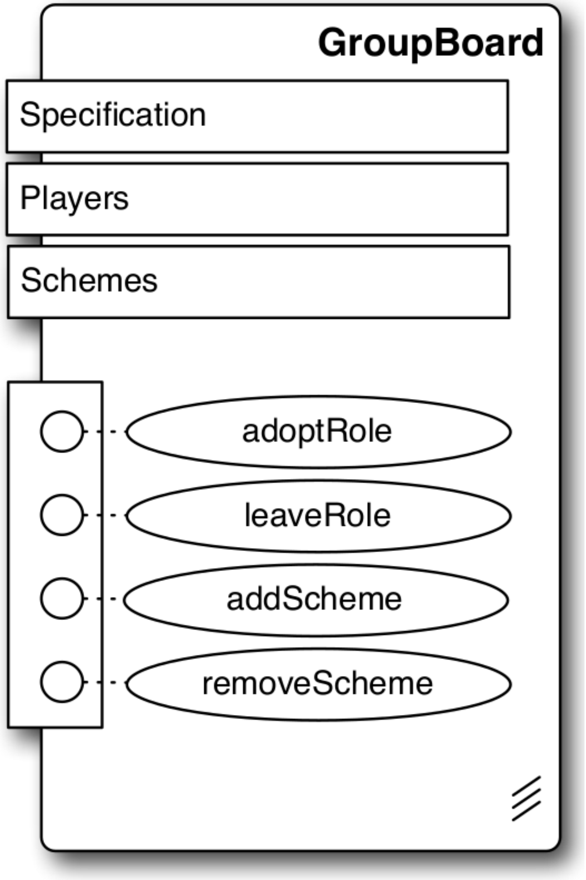
GroupBoard Artifact



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 x role)
- **schemes**: a list of scheme identifiers that the group is responsible for

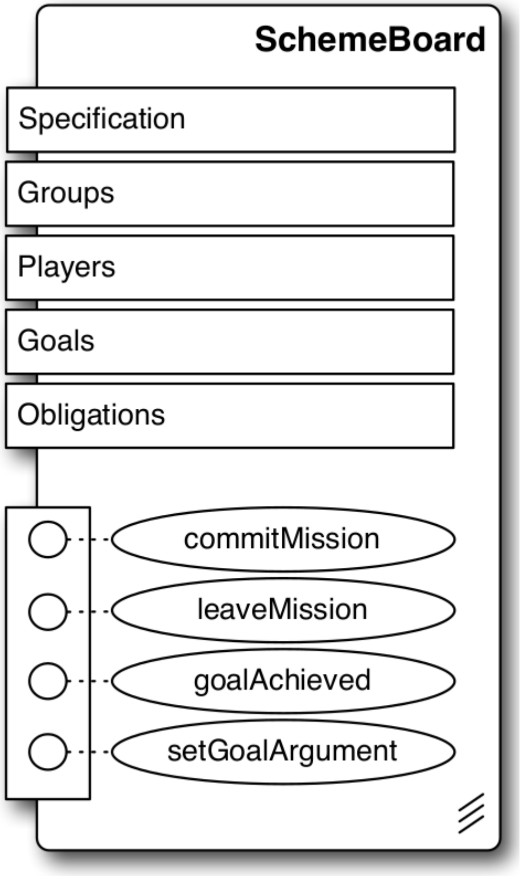
GroupBoard Artifact



Operations

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

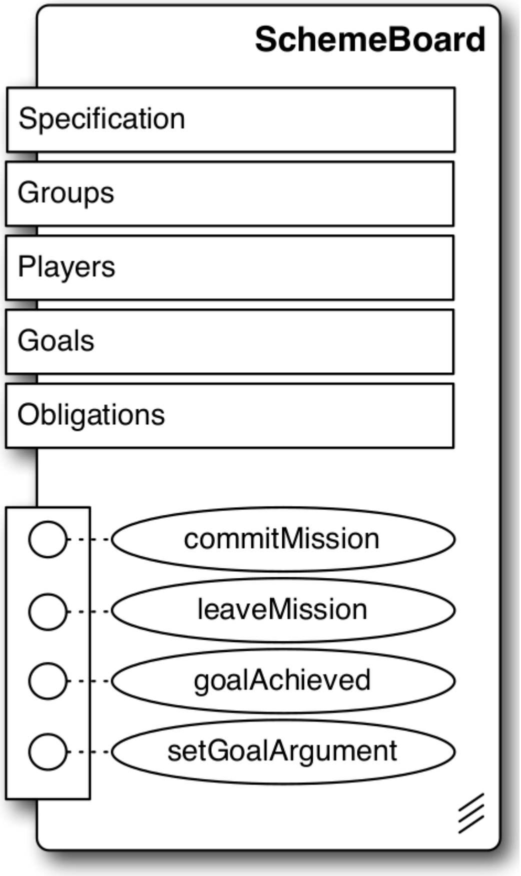
SchemeBoard Artifact



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

Organization Entity

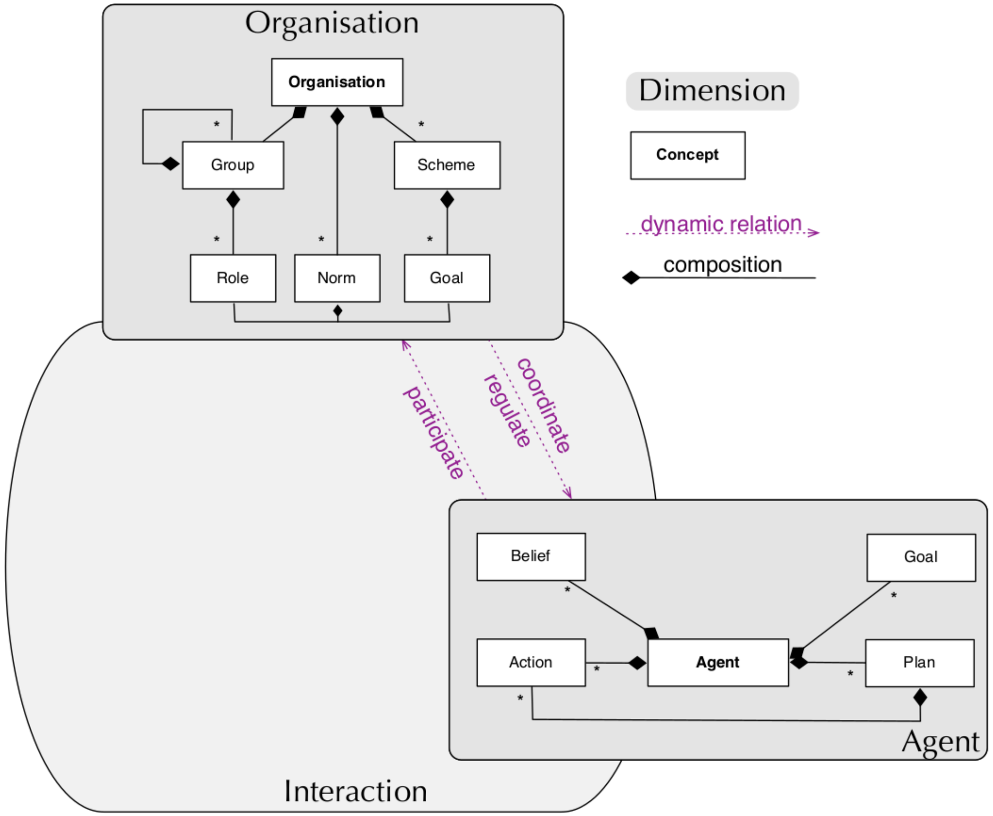
smart-room.jcm

```
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
    }
}
```

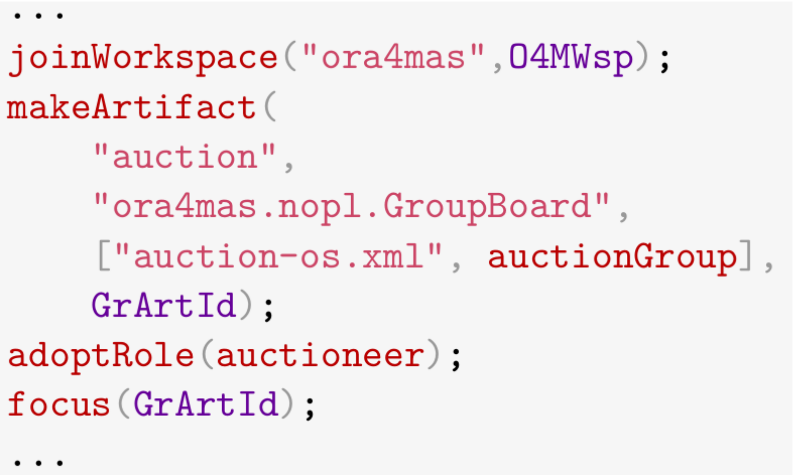
Integrating Agent and Organization Dimensions



- Agents can interact with organizational artifacts as with ordinary artifacts by perception and action
- Agent integration provides “internal” tools for the agents to simplify their interaction with the organization:
  - maintenance of a local copy of the organizational state
  - production of organizational events
  - provision of organizational actions

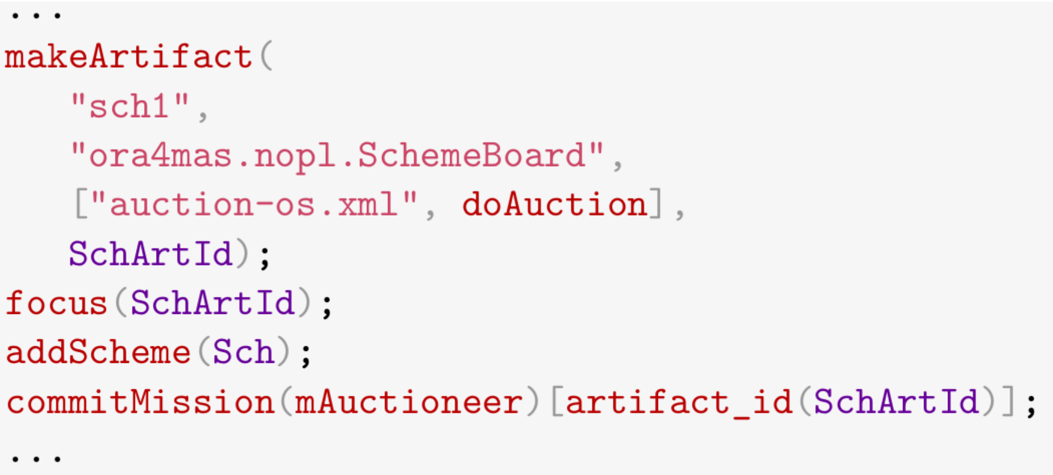
Integrating Agent and Organization Dimensions

GroupBoard



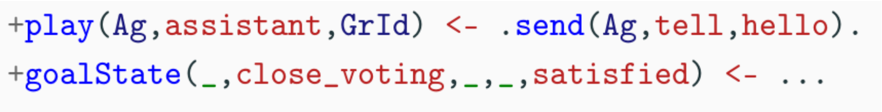
```
...
joinWorkspace("ora4mas",O4MWsp);
makeArtifact(
    "auction",
    "ora4mas.nopl.GroupBoard",
    ["auction-os.xml", auctionGroup],
    GrArtId);
adoptRole(auctioneer);
focus(GrArtId);
...
```

SchemeBoard



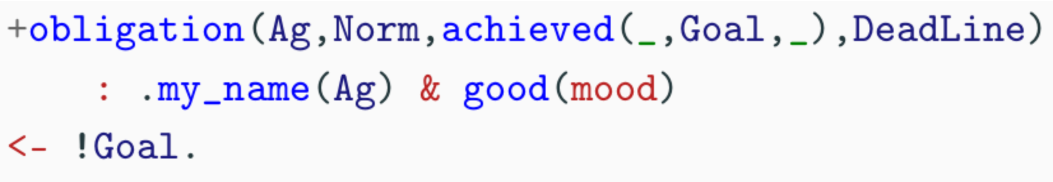
```
...
makeArtifact(
   "sch1",
   "ora4mas.nopl.SchemeBoard",
   ["auction-os.xml", doAuction],
   SchArtId);
focus(SchArtId);
addScheme(Sch);
commitMission(mAuctioneer)[artifact_id(SchArtId)];
...
```

Including organization-reasoning abilities into agents



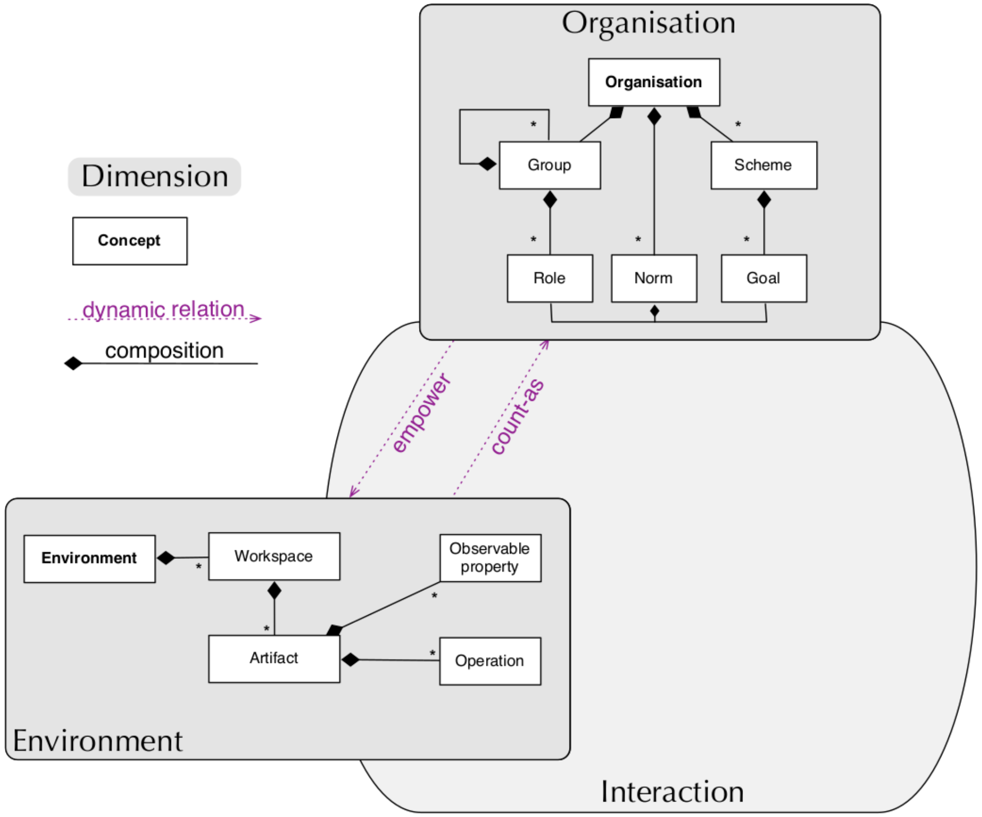
```
+play(Ag,assistant,GrId) <- .send(Ag,tell,hello).
+goalState(_,close_voting,_,_,satisfied) <- ...
```

Including norm-reasoning abilities into agents



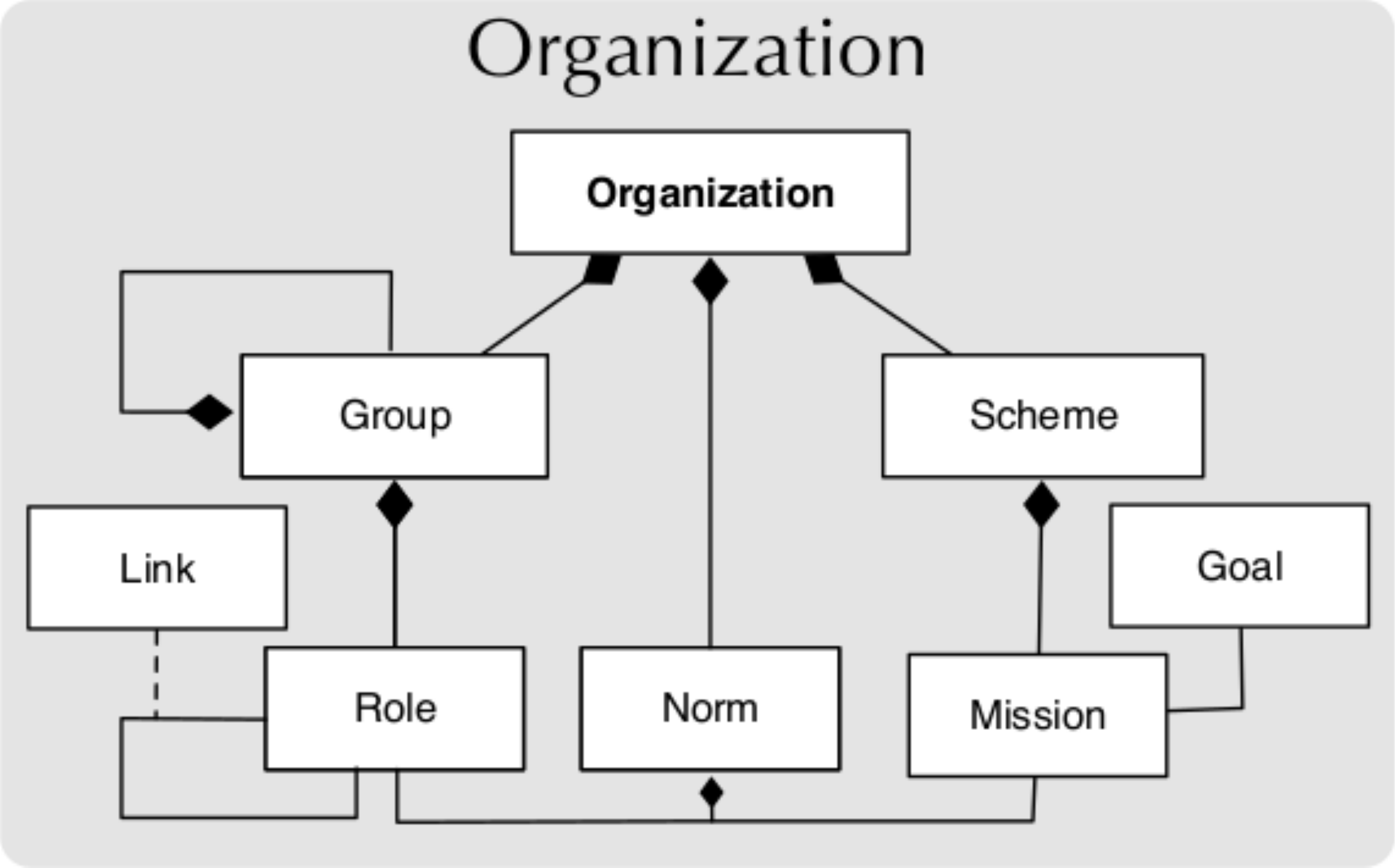
```
+obligation(Ag,Norm,achieved(_,Goal,_),DeadLine)
    : .my_name(Ag) & good(mood)
<- !Goal.
```

Integrating Environment and Organization Dimensions



- Changes in the state of the environment may **count-as** changes in the state of the organization (de Brito et al., 2015)
- This dynamic relation is a **practical way of situating organizations in an environment**, as happens for the agents, regulating some part of the environment (e.g., a traffic light at a crossroads) in a particular way and ruling it differently in other parts
- Organizations may **empower** the elements of the environment by allowing them to control and regulate actions or perception of the agents

Wrap-up: Organization Dimension



Wrap-up: Organization Dimension

- Model to specify global orchestration
  team strategy is defined at a high level
- Ensure agents follow some of the constraints specified by the organization
- Help agents to work together
- The organization is interpreted at runtime, it is not hardwired in the agents' code
- The agents can ‘handle’ the organization (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

References

- de Brito, M., Hübner, J. F., & Boissier, O. (2015). Bringing constitutive dynamics to situated artificial institutions. In *Proc. of 17th Portuguese Conference on Artificial Intelligence (EPIA 2015)*, LNCS, vol. 9273, pp. 624–637. Springer.
- Hübner, J. F., Boissier, O., Kitio, R., & Ricci, A. (2010). Instrumenting multi-agent organisations with organisational artifacts and agents: “Giving the organisational power back to the agents”. *Journal of Autonomous Agents and Multi-Agent Systems*, 20(3):369–400.