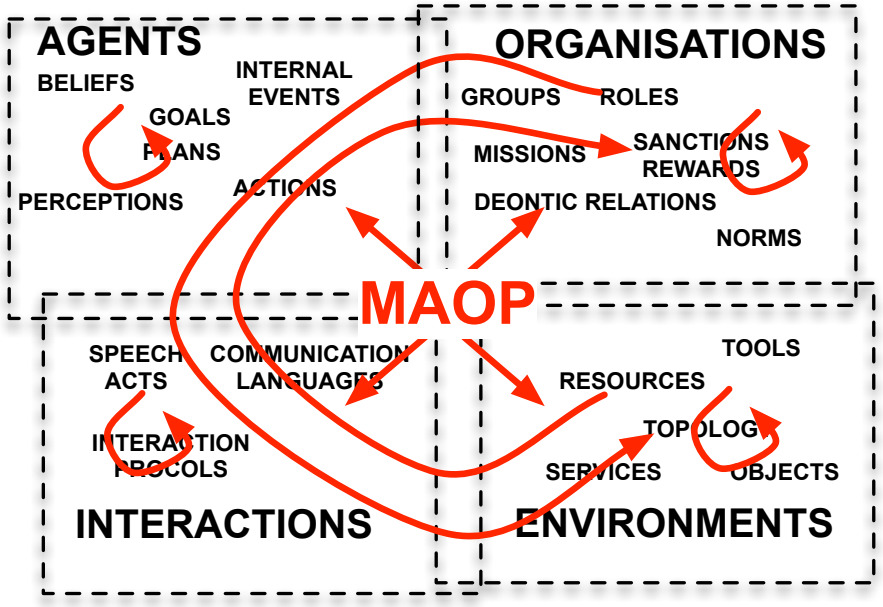
Multi-Agent Oriented Programming

Introduction to Multi-Agent Oriented Programming with JaCaMo

Outline

Multi-Agent Oriented Programming



Multi-Agent Oriented Programming (MAOP) aims at programming systems:

- as **organisation** of autonomous **agents** in **interaction** with each other within a shared **environment**,
- by keeping alive from design to execution, concepts pertaining to each of the A/E/I/O families as well as their control/life cycles.

$\rightsquigarrow$ Going beyond each of the A/E/I/O oriented programming approaches

Key features

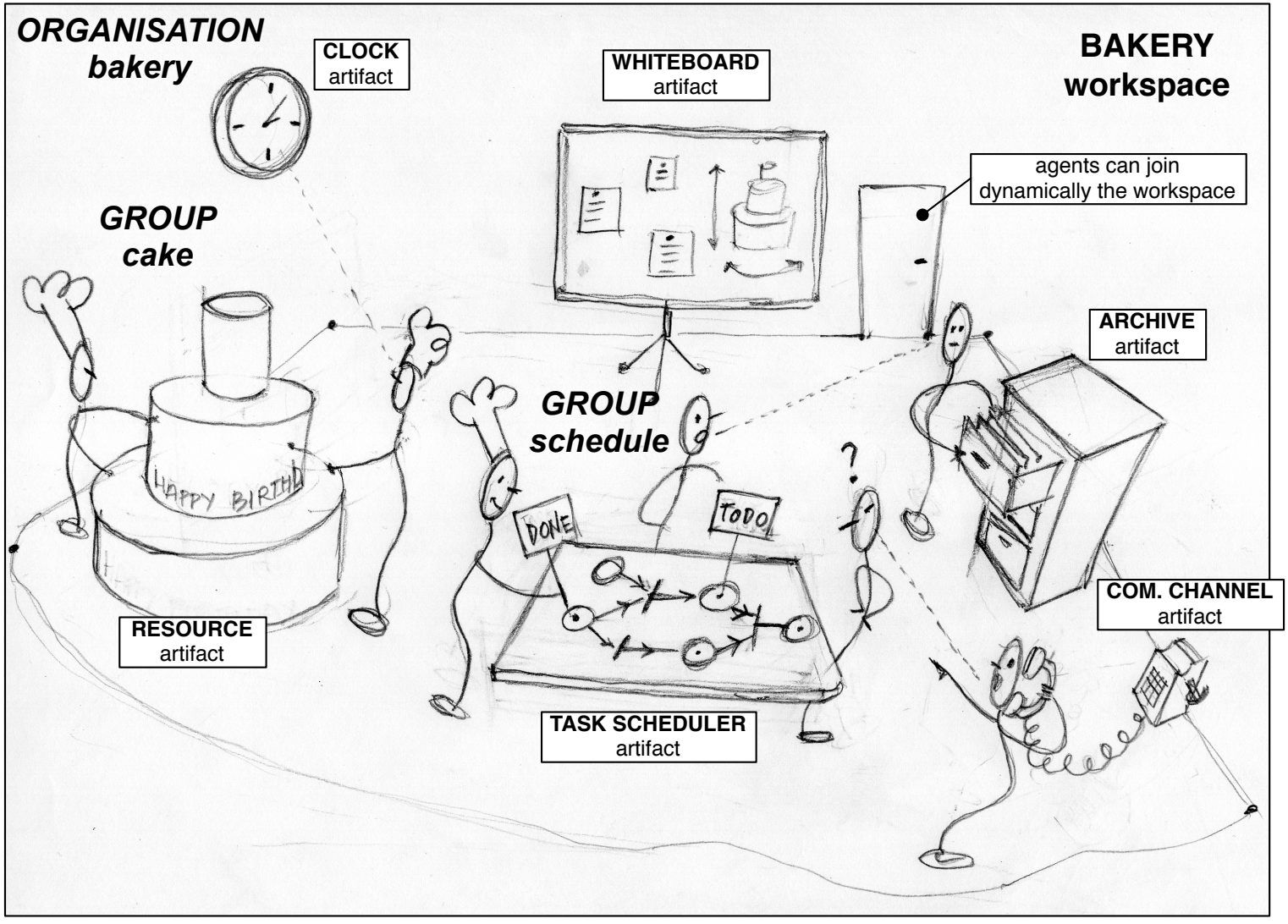
Multi-Agent Oriented Programming

- **Abstraction**
  - keeping the concepts alive from design to execution, e.g. no agents sharing and calling OO objects
  - effective programming models for controllable and observable computational entities
- **Modularity**
  - away from the monolithic and centralised view
- **Orthogonality**
  - wrt models, architectures, platforms
  - support for heterogeneous systems
- **Dynamic extensibility**
  - dynamic construction, replacement, extension of the entities participating to the system
  - support for open systems
- **Reusability**
  - reuse of the entities participating to the system for different kinds of applications

Outline

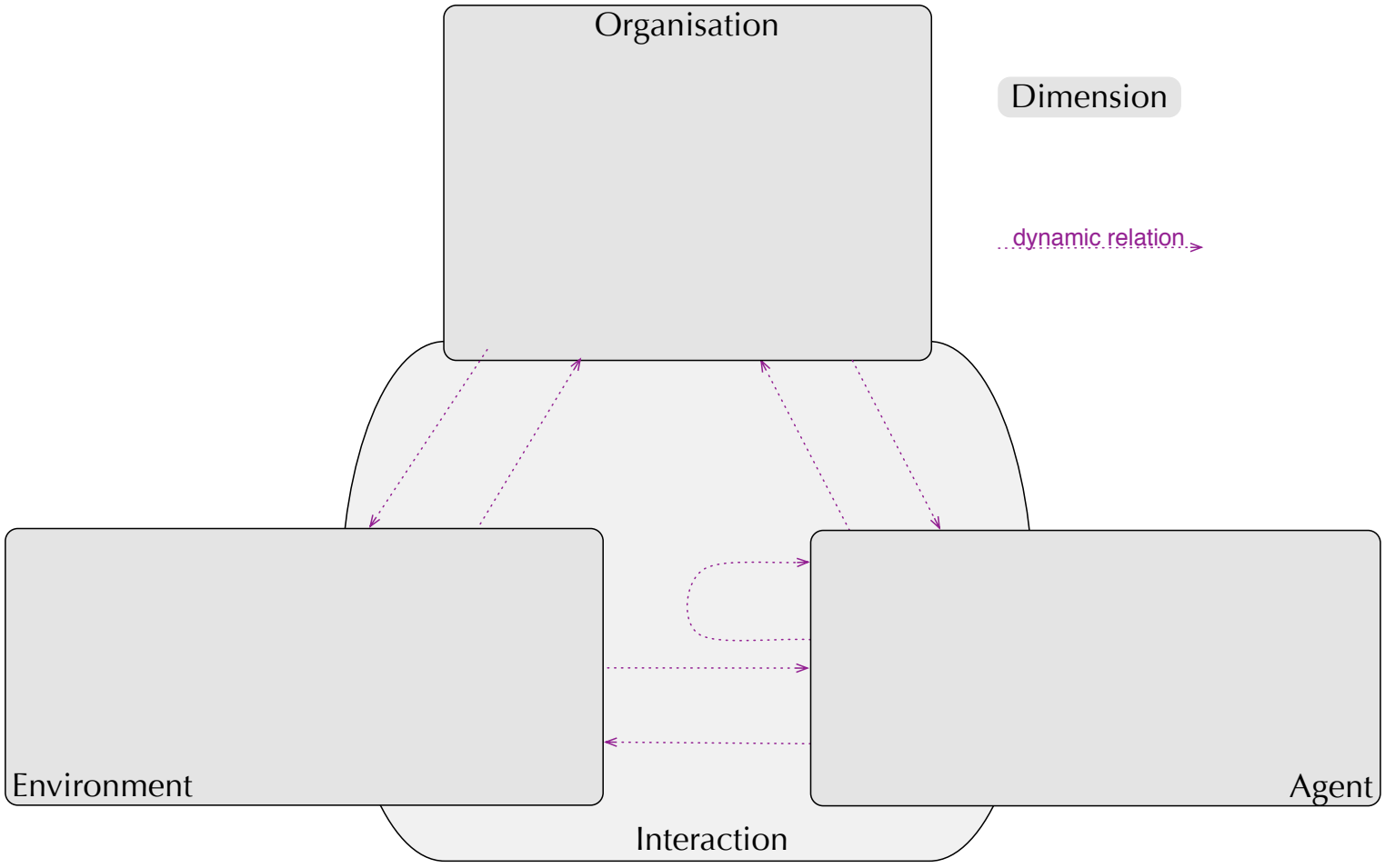
Outline

JaCaMo meta-model overview



Seamless integrated conceptual dimensions

JaCaMo meta-model overview



Simplified view on JaCaMo meta-model [Boissier et al., 2011]

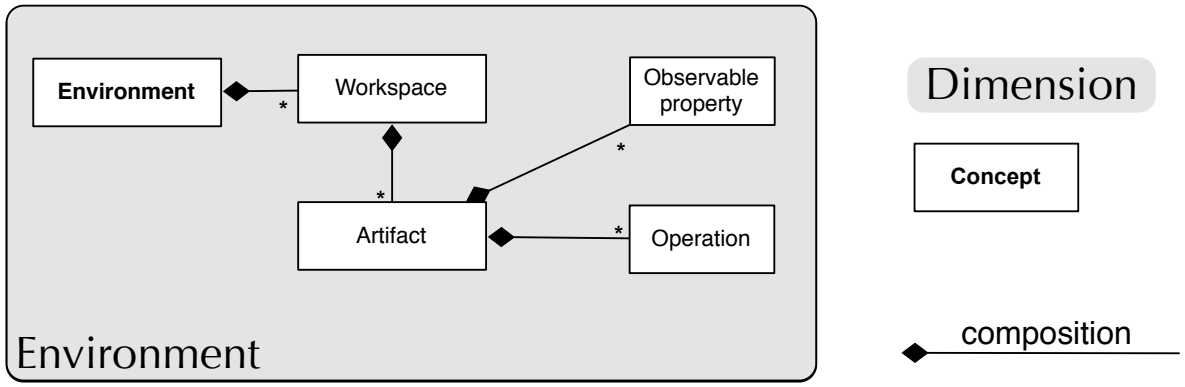
A seamless integration of three dimensions based on Jason [Bordini et al., 2007], Cartago [Ricci et al., 2009], Moise [Hübner et al., 2009] meta-models

Forthcoming book in 2020 @ MIT Press

Outline

Environment dimension – Basic concepts

Environment dimension



Simplified conceptual view (A&A meta-model [Omicini et al., 2008])

Simple artifact program:

```
public class Counter extends Artifact {
    void init(int initialValue) {
        defineObsProperty("count", initialValue);
    }

    @OPERATION void inc() {
        ObsProperty prop = getObsProperty("count");
        prop.updateValue(prop.intValue()+1);
    }
}
```

Environment dimension – Dynamics

Environment life-cycle

- Creation/Deletion of Workspaces

Workspace life-cycle:

- Creation/Deletion of Artifacts
- Creation/Deletion & Entry/Exit of Agents

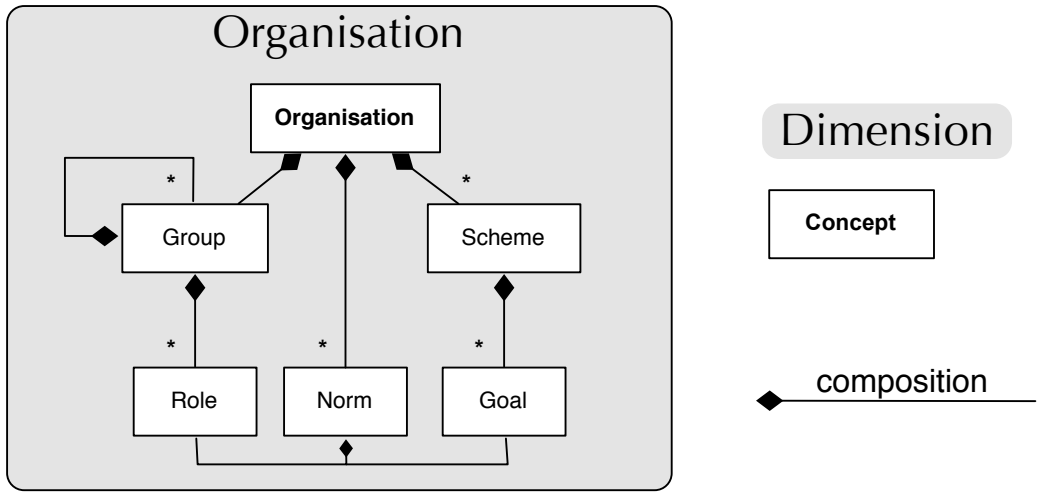
Artifact life-cycle:

- Atomic execution, Success/Failure, Activation/Deactivation of an operation
- Creation/Deletion/Update of Observable Properties
- Linking/Unlinking with other artifacts

Outline

Organisation dimension – Basic concepts

Organisation dimension



Simplified Conceptual View ($\mathcal{M}$oise meta-model [Hübner et al., 2009])

Excerpts from organisation program:

```
<structural-specification>

<role-definitions>
 <role id="auctioneer" />
 <role id="participant" />
</role-definitions>

<group-specification id="auctionGroup">
 <roles>
  <role id="auctioneer"  min="1" max="1"/>
  <role id="participant" min="0" max="300"/>
 </roles>
</group-specification>
</structural-specification>
```

Structural spec.

```
<functional-specification>
 <scheme id="doAuction">
  <goal id="auction">
   <argument id="Id" />
   <argument id="Service" />
   <plan operator="sequence">
    <goal id="start" />
    <goal id="bid"    ttf="10 seconds" />
    <goal id="decide" ttf="1 hour" />
   </plan>
  </goal>
  <mission id="mAuctioneer" min="1" max="1">
   <goal id="start" />
   <goal id="decide" />
  </mission>
```

Functional spec.

```
<normative-specification>
  <norm id="n1"  type="permission"
        role="auctioneer"
        mission="mAuctioneer" />
  <norm id="n2"  type="obligation"
        role="participant"
        mission="mParticipant" />
</normative-specification>
```

Normative spec.

```
norm n1 : plays(A, auctionneer, G) ->
  forbidden(A,n1,plays(A,participant,G),
    'forever').
```

program in NPL

Organisation dimension – Dynamics

Organisation life-cycle

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

Organisation structure life-cycle

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

Coordination activity life-cycle

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

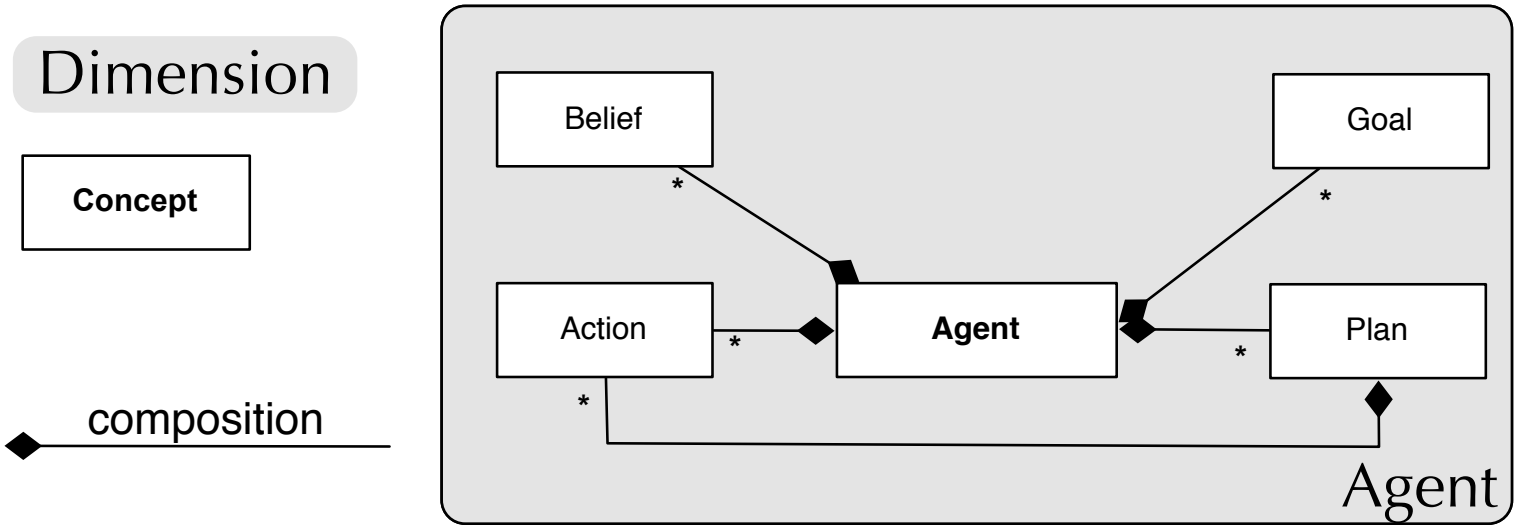
Normative Regulation activity life-cycle

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

Outline

Agent dimension – Basic concepts

Agent dimension



Simplified Conceptual View (Jason meta-model [Bordini et al., 2007]):

Simple Agent Program:

```
happy(bob). // initial belief
!say(hello). // initial goal
/* Plans */
+!say(X) : happy(bob) <- .print(X).
// ...
```

example bob.asl

```
+happy(A) <- !say(hello(A)).
+!say(A) : not today(friday) <- .print(X); !say(X).
+!say(X) : today(friday) <- .print("stop").
-happy(A) : .my_name(A) <- .drop_intention(say(_)).
```

example carl.asl

Agent dimension – Dynamics

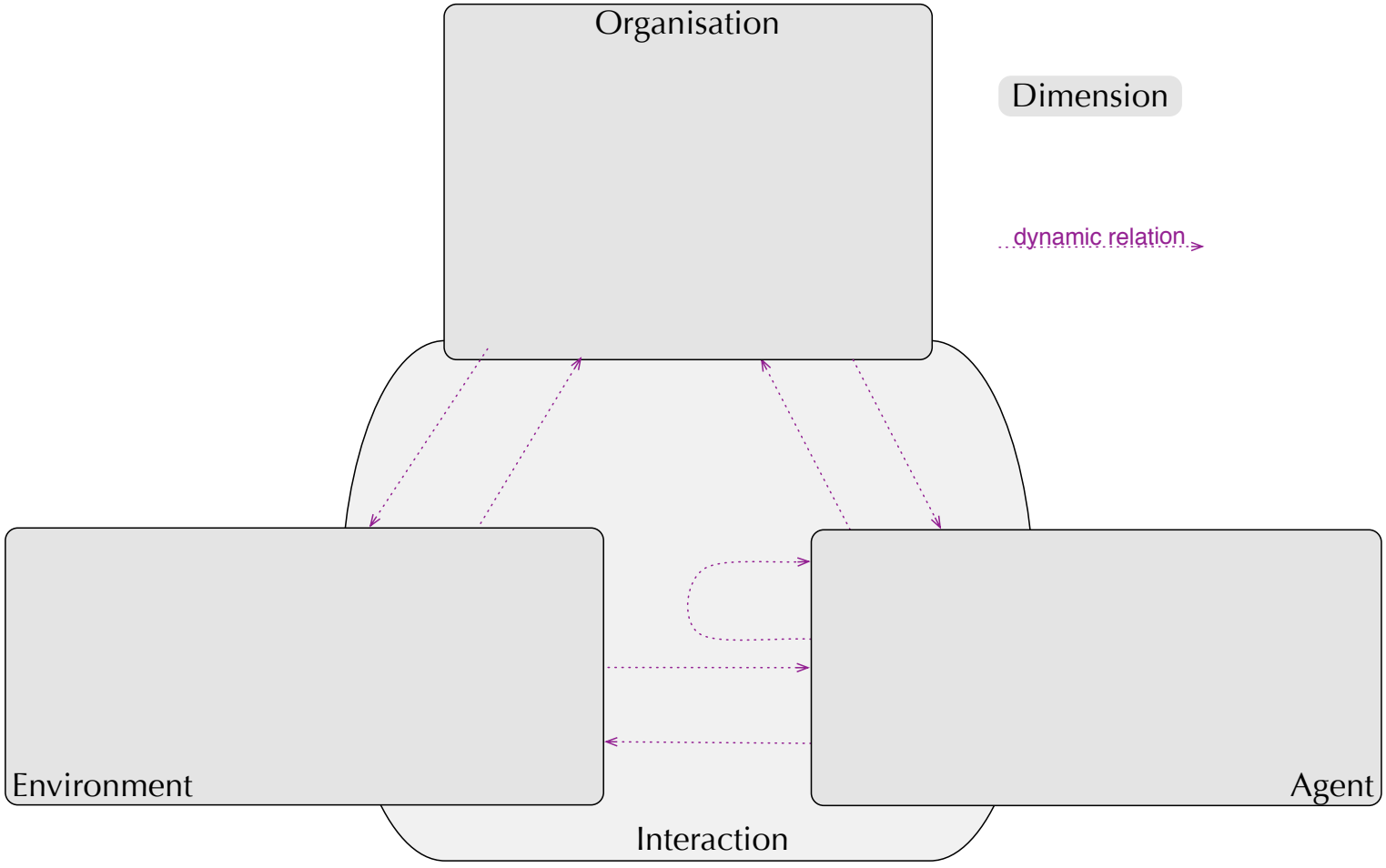
Agent dimension

1. Perceive the environment and update belief base
2. Process new messages
3. Select event
4. Select **relevant** plans
5. Select **applicable** plans
6. Create/update intention
7. Select intention to execute
8. Execute one step of the selected intention

Outline

Seamless integrated dimensions

Integrated dimensions

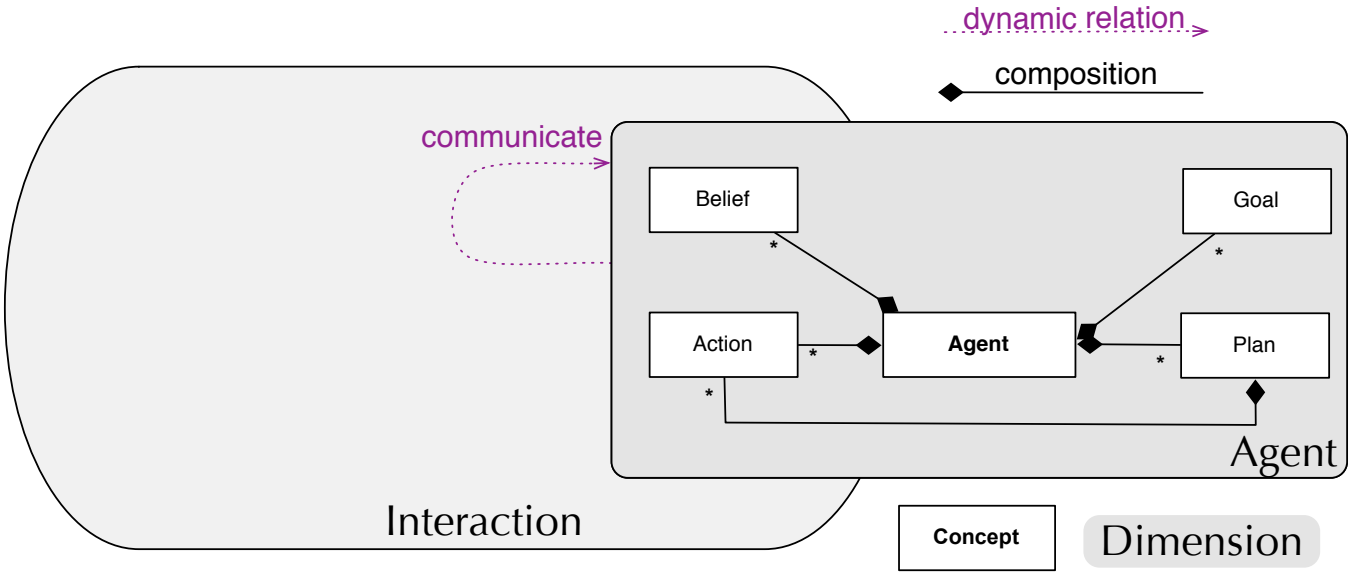


Simplified view on JaCaMo meta-model [Boissier et al., 2011]

A seamless integration of three dimensions based on Jason [Bordini et al., 2007], Cartago [Ricci et al., 2009], Moise [Hübner et al., 2009] meta-models

Integrating **A** & **A** dimensions – Interacting agents

Integrated dimensions – Integrated dimensions



based on KQML or Jade/FIPA ACL

```
!start.
+!start <- .send(bob,tell,happy(bob));
           .send(bob,tell,happy(alice));
           .send(bob,achieve,count).
```

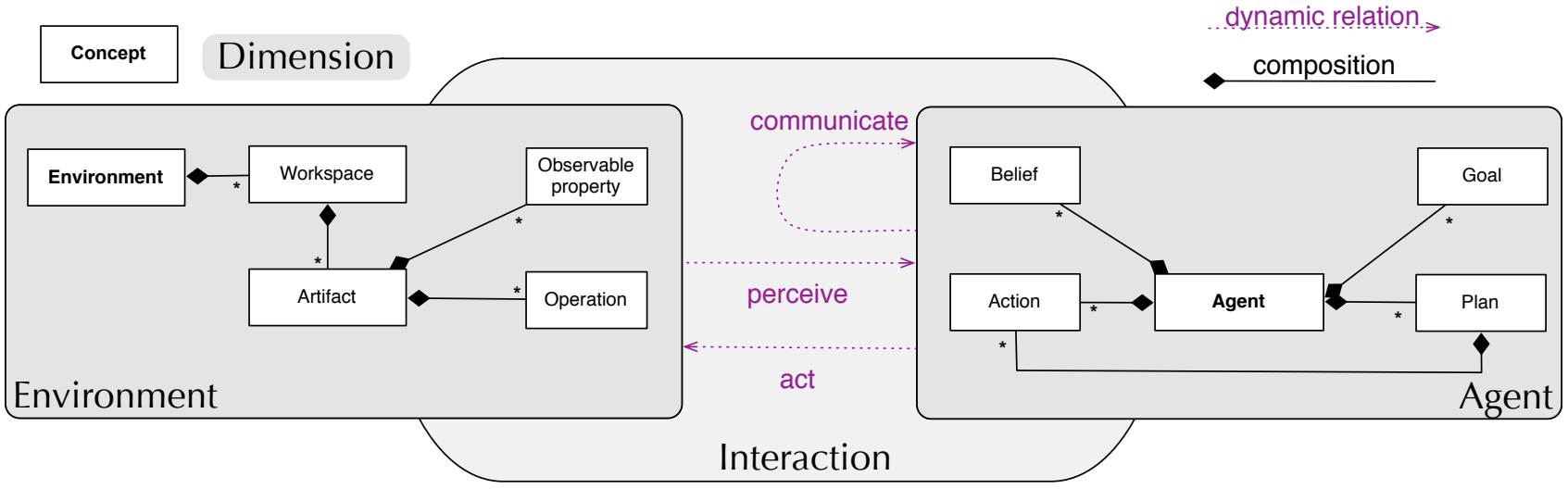
example alice.asl

```
happy(bob).
!say(hello).
+!say(X) : happy(bob) <- .print(X).
+!count <- ...
```

example bob.asl

Integrating A & E dimensions – Interacting agents

Integrated dimensions – Integrated dimensions



based on JaCa bridge

```
!start.
+count(X) <- .print("counter incremented").
+!start <- .send(bob,tell,happy(bob));
           .send(bob,tell,happy(alice));
           .send(bob,achieve,count).
```

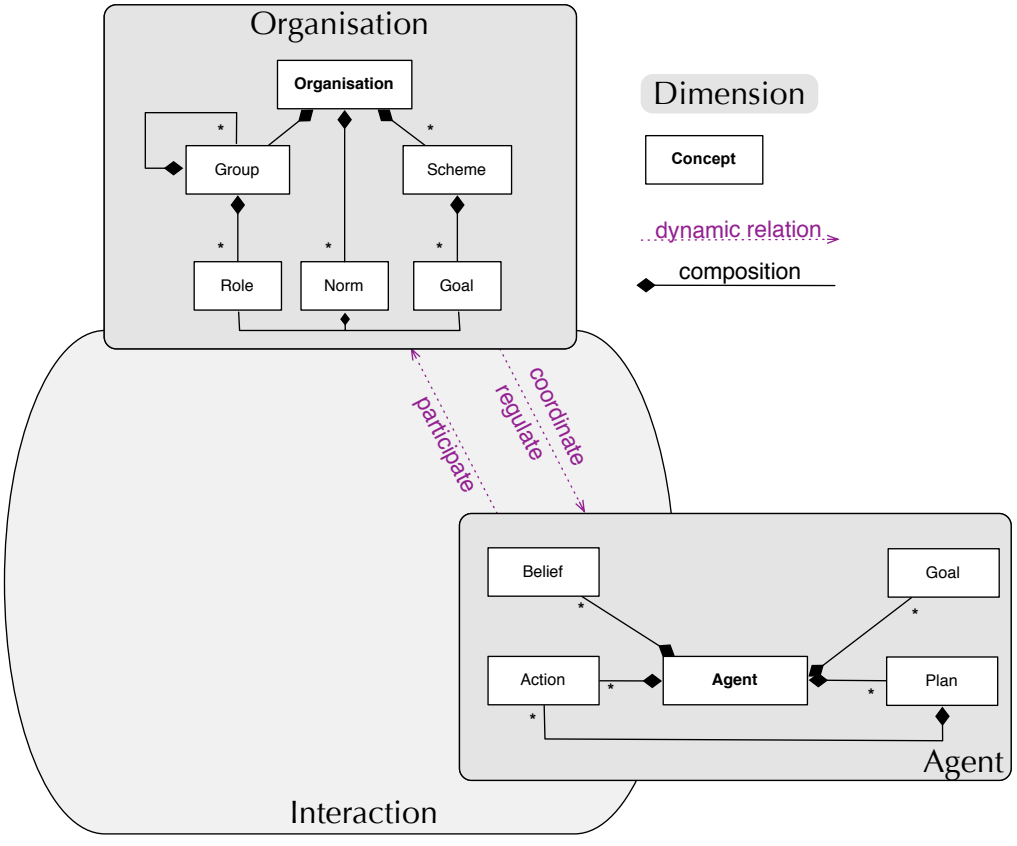
example alice.asl

```
happy(bob).
!say(hello).
+!say(X) : happy(bob) <- .print(X).
+!count : count(0) <- inc.
```

example bob.asl

Integrating A & O dimensions

Integrated dimensions – Integrated dimensions

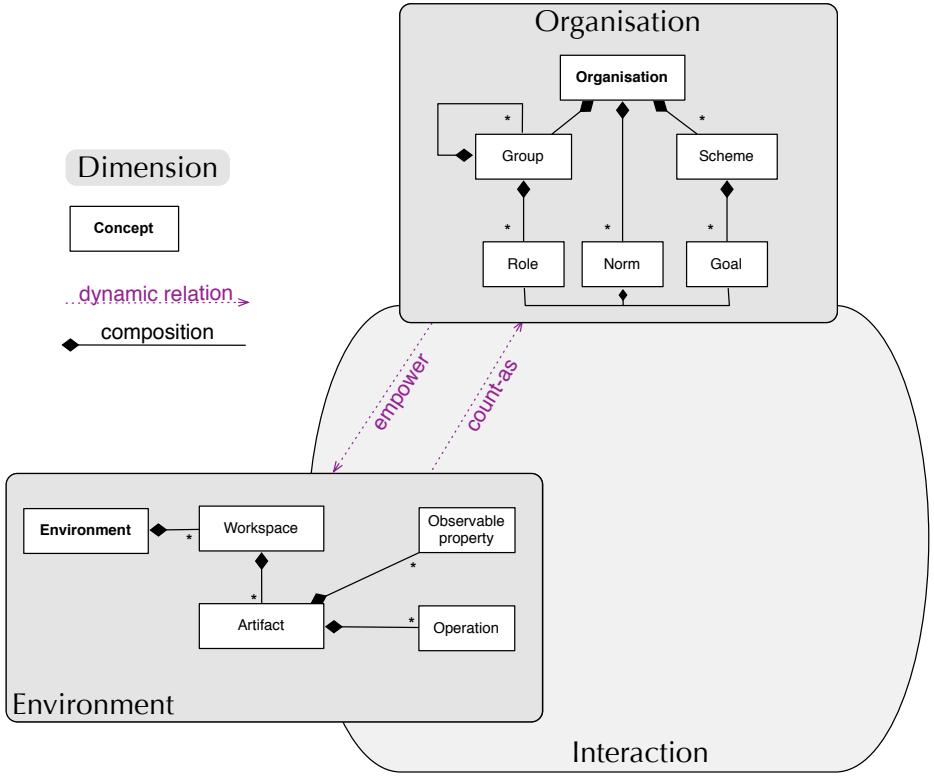



```
+!contract("SitePreparation",
      GroupBoardId)
   <- adoptRole(site_prep_contractor);
      focus(GroupBoardId).

+obligation(Ag,Norm,What, Deadline)
    [artifact_id(ArtId)]
    : .my_name(Ag) &
        (satisfied(Scheme,Goal)=What |
            done(Scheme,Goal,Ag)=What)
    <- !Goal[scheme(Scheme)];
       goalAchieved(Goal)
            [artifact_id(ArtId)].
```

based on ORA4MAS [Hübner et al., 2009]

Integrating O & E dimensions

Integrated dimensions – Integrated dimensions



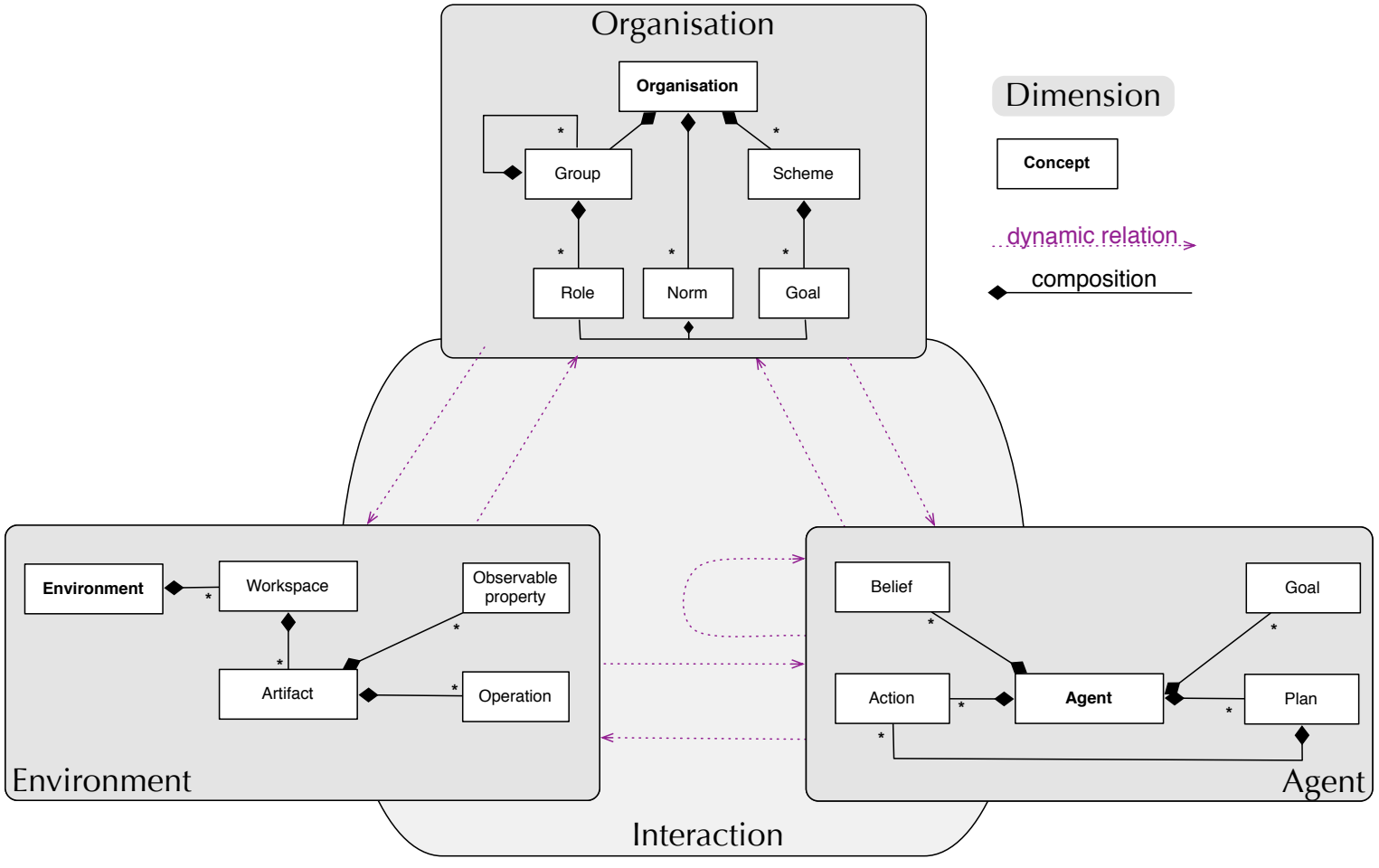

```
institution_id : bhInst.
status_functions:
states: play(A,R,G),
  responsible(G,S),
  committed(A,Mission,S),
  achieved(S,G,A),
  done(S,G,A).

constitutive_rules:
...
2:
currentWinner(auction_for_SitePreparation,Agent)
count-as play(Agent,site_prep_contractor,
              "hsh_group")
while nticks(clock,Time)&(Time>=8000).
...
12:
play(A,house_owner,"hsh_group")
count-as committed(A,management_of_house_building,
                   "bhsch")
while responsible("hsh_group","bhsch").
...
22:
count-as achieved("bhsch",site_prepared,Agent)
when prepareSite[sai__agent(Agent)].
...
```

based on Situated Artificial Institution [de Brito et al., 2015]

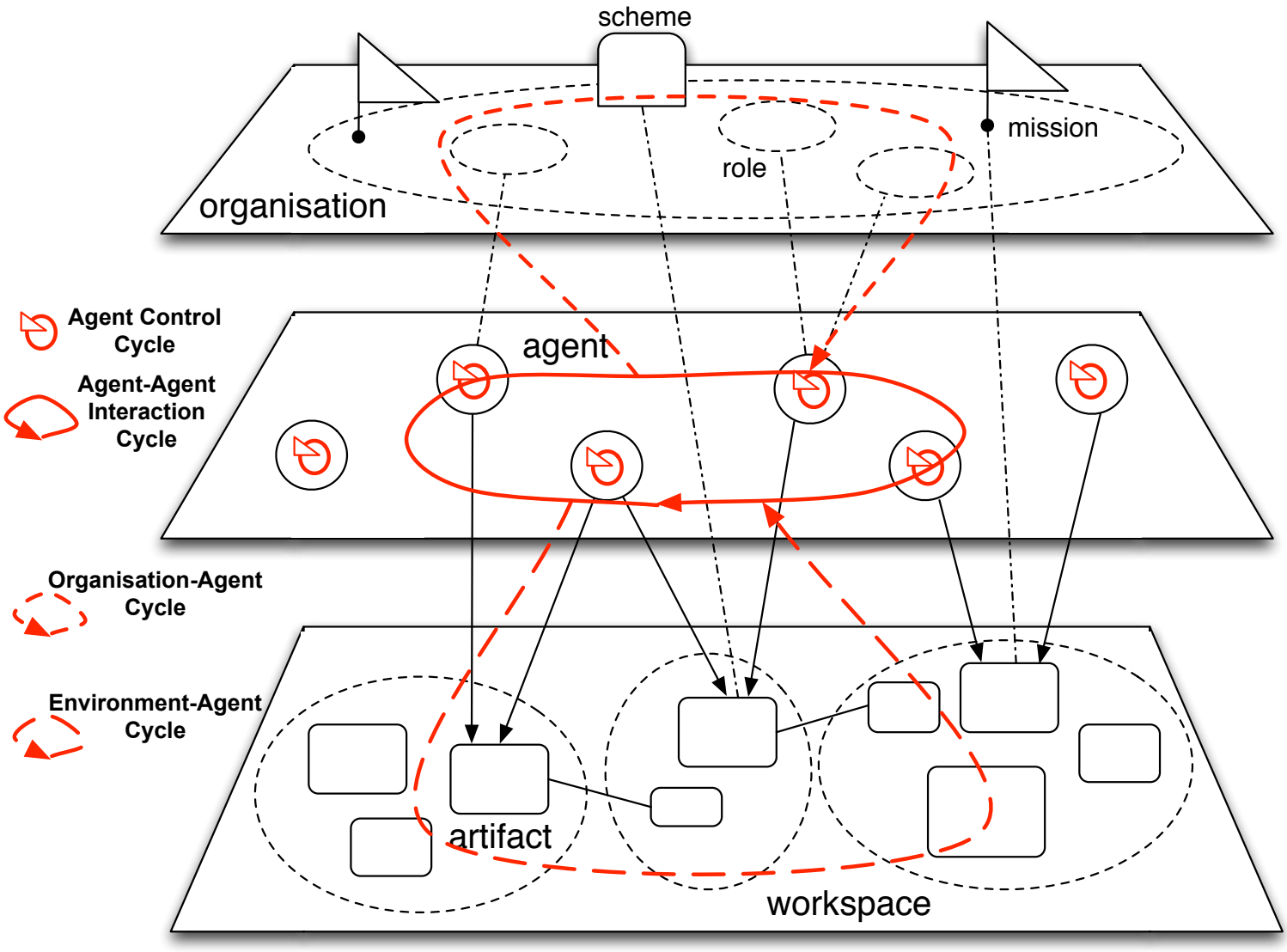
Outline

Synthesis

Synthesis



Synthesis: MAO Dynamics



Multi-Agent Oriented Programming

The JaCaMo platform

Outline

Outline

JaCaMo multi-agent platform

- Multi-agent technologies currently integrated:
  - **A**gent dimension: *Jason* agents [Bordini et al., 2007]
  - **E**nvironment dimension: CArtAgO platform [Ricci et al., 2009]
  - **O**rganisation dimension: $\mathcal{M}$oise framework [Hübner et al., 2009]

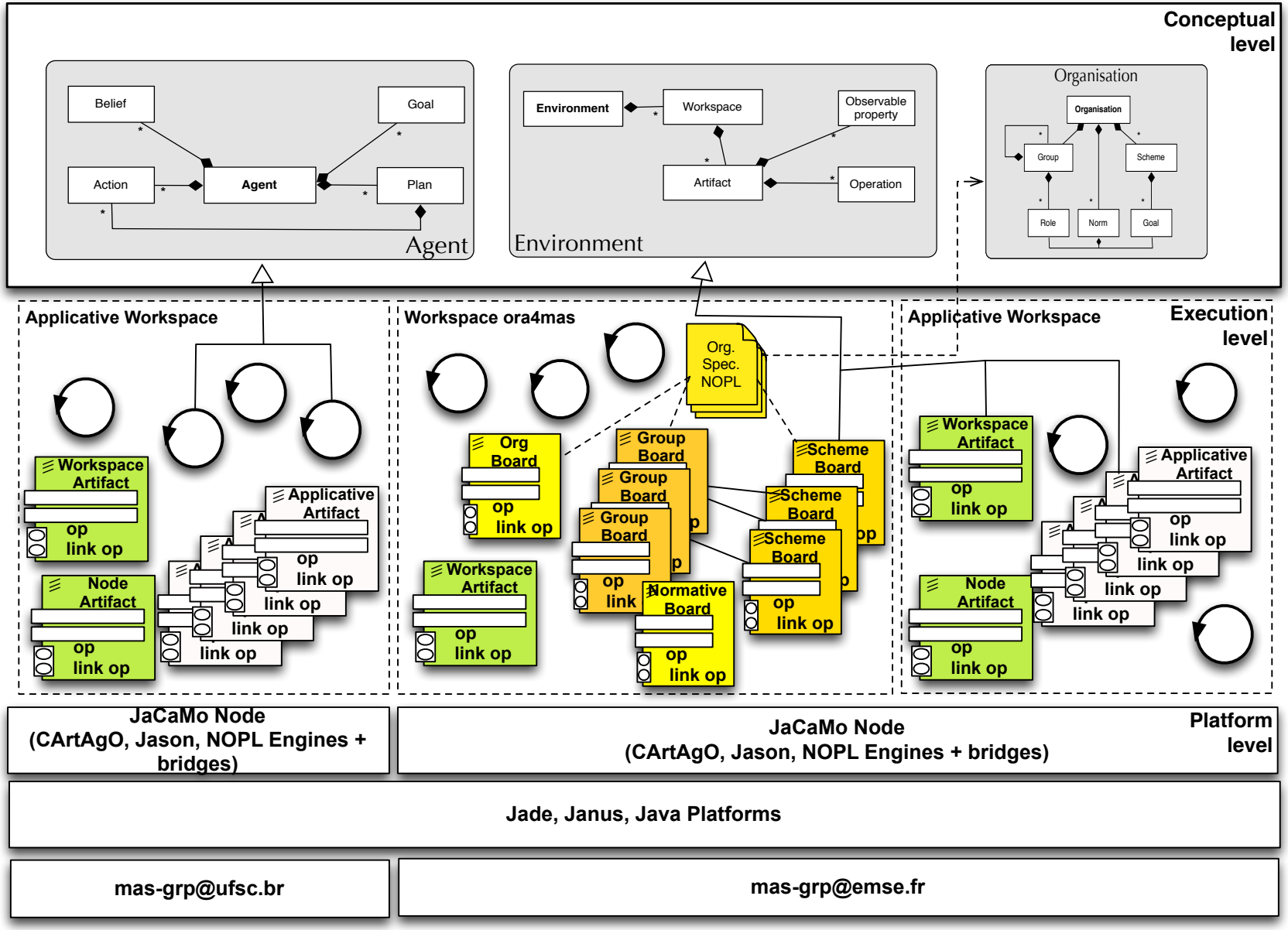
- Dedicated bridges integrate each of the dimensions altogether:
  - **A**gent – **E**nvironment integration: c4Jason, c4Jadex [Ricci et al., 2009]
  - **E**nvironment – **O**rganisation integration: count-as/enact rules [Piunti et al., 2009] [de Brito et al., 2015]
  - **A**gent – **O**rganisation integration: artifacts dedicated to organisation management [Hübner et al., 2009]

↝ `http://jacamo.sourceforge.net`, `https://github.com/jacamo-lang/jacamo/`

Open to integrate other multi-agent technologies

Execution Architecture

JaCaMo multi-agent platform



Integration with other technologies

JaCaMo multi-agent platform

- Web 2.0 – `http://jaca-web.sourceforge.net`
  - implementing Web 2.0 applications
- Android Platforms – `http://jaca-android.sourceforge.net`
  - implementing mobile computing applications on top of the Android platform
- Web Services – `http://cartagows.sourceforge.net`
  - building SOA/Web Services applications
- Arduino Platforms – `http://jacamo.sourceforge.net`
- JaCaMo with hypermedia environment – (see next slides)
- Jason-ROS: modular interface between Jason, CArtAgO, and ROS – `https://github.com/lsa-pucrs/jason-ros-releases/releases`
- Semantic Technologies
  - JaSA: Semantically Aware Agents
  - JASDL: Combining agent-oriented programming and semantic web technologies
- JaCaDDM: Distributed Data Mining system founded on the Agents and Artifacts paradigm – `https://sourceforge.net/projects/jacaddm/`

Outline

JaCaMo multi-agent system development

- Available at:
  - `http://jacamo.sourceforge.net/`
  - `https://github.com/jacamo-lang/jacamo`
- Documentation:
  - Getting started guides, tutorials, FAQ,
  - Reference documentation on "JaCaMo project files", on debugging in JaCaMo, on Agent Programming Language, on Organisation Programming Language (most of the JaCaMo documentation is available in the doc folder of the distribution)

All the documentation is available at: `http://jacamo.sourceforge.net/`

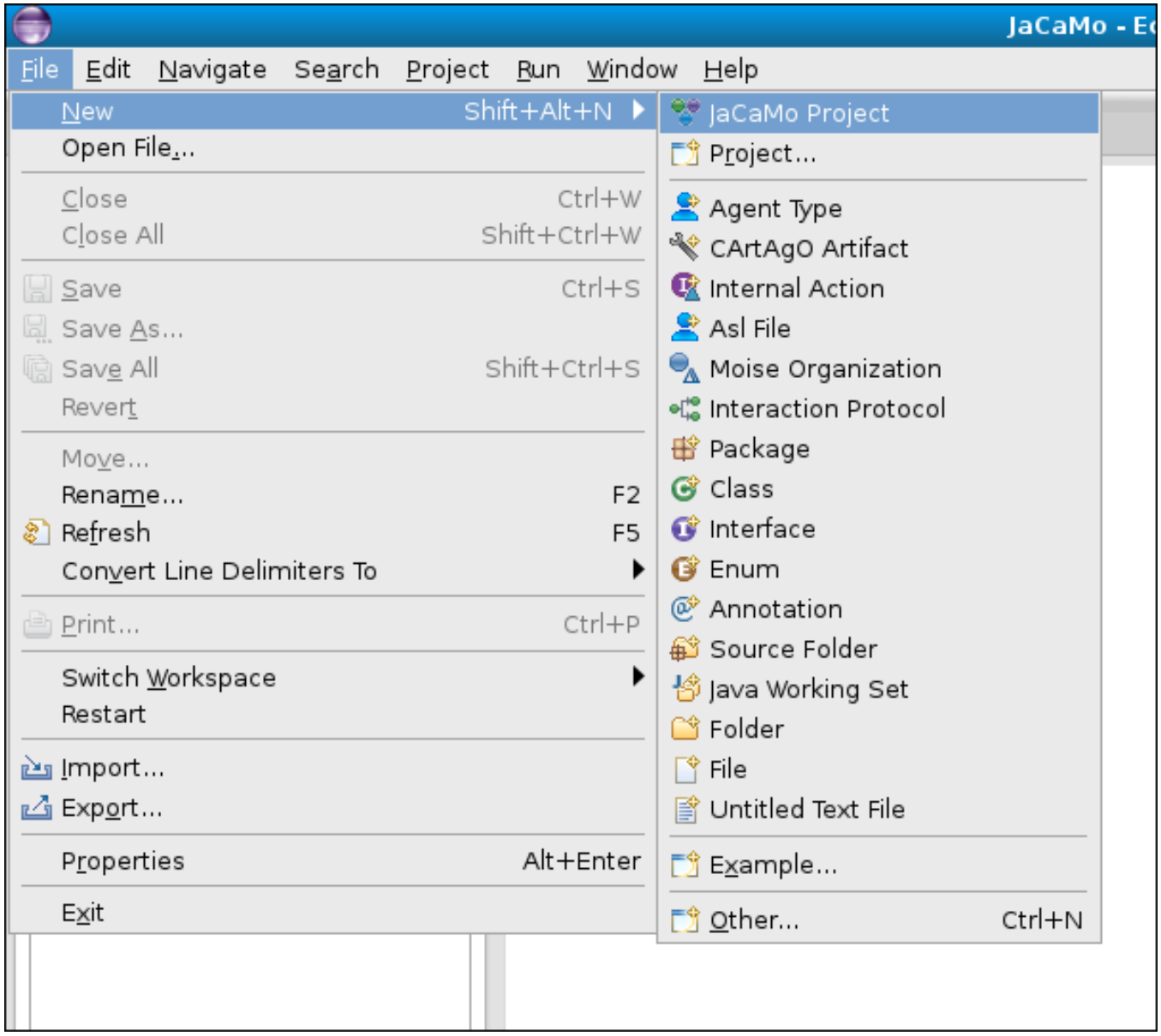
- Examples of codes, of demos
- Configuration of the platform (.jacamo file in the home directory):
  - to be done after each installation of the platform

JaCaMo multi-agent system development

- Structure of a JaCaMo project:
  - **src** groups all the source code of the project
    - **agt** groups all the agents' code (.asl)
    - **env** groups all the artifacts' code (.java)
    - **org** groups all the organisations' code (.xml, .npl)
  - one or several JaCaMo project file (.jcm)
  - **logging.properties** is the log configuration file
- Development environment:
  - Use of shell commands:
    - `jacamo-new-project projectName`: new project creation,
    - `jacamo projecName`: project execution,
    - `jacamo-jar fileName`: create a jar with all resources to run the application calling java)
  - Use of eclipse IDE (JaCaMo plugin for eclipse)
  - Use of Gradle
  - Use of Docker

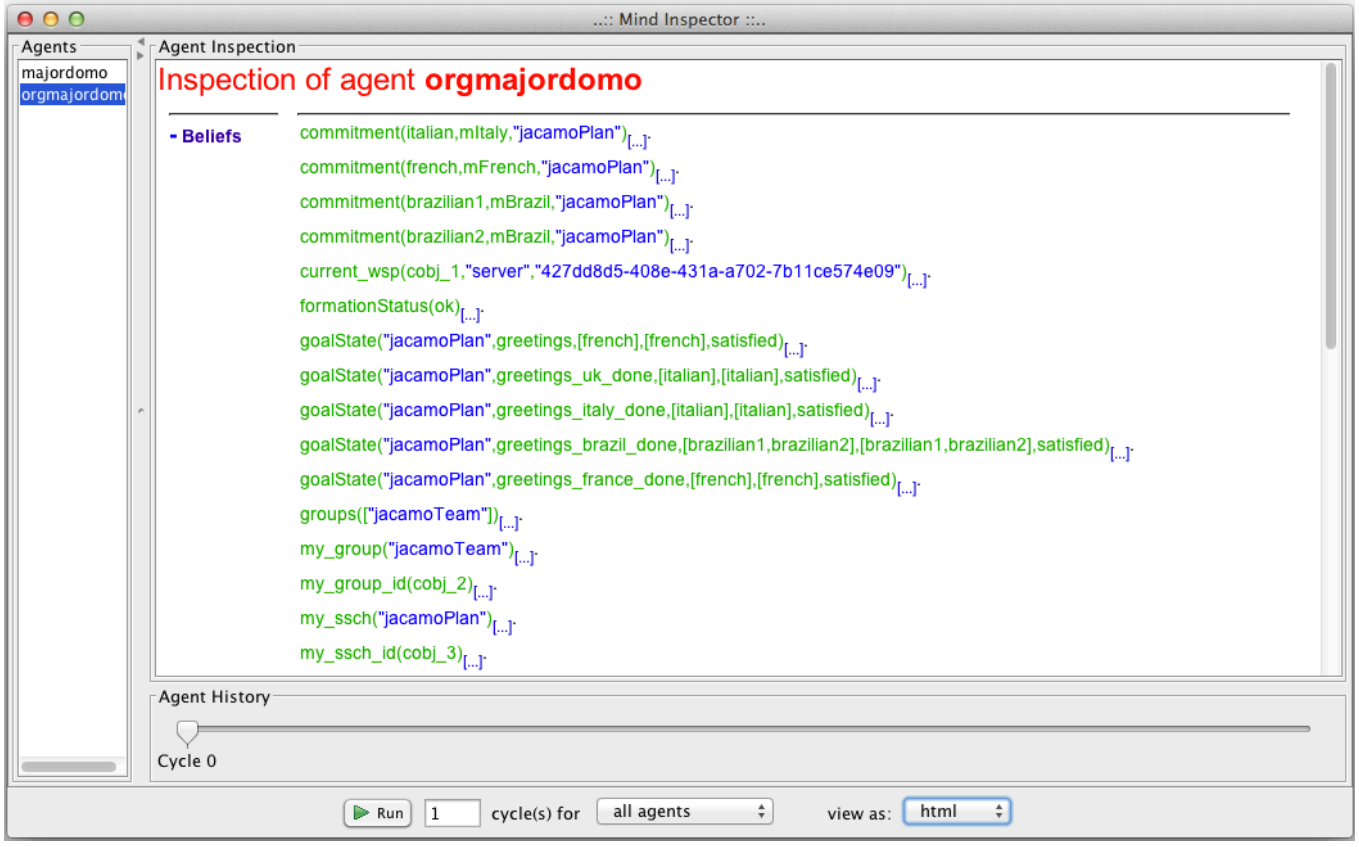
Eclipse JaCaMo plugin

JaCaMo multi-agent system development



Agent's inspector

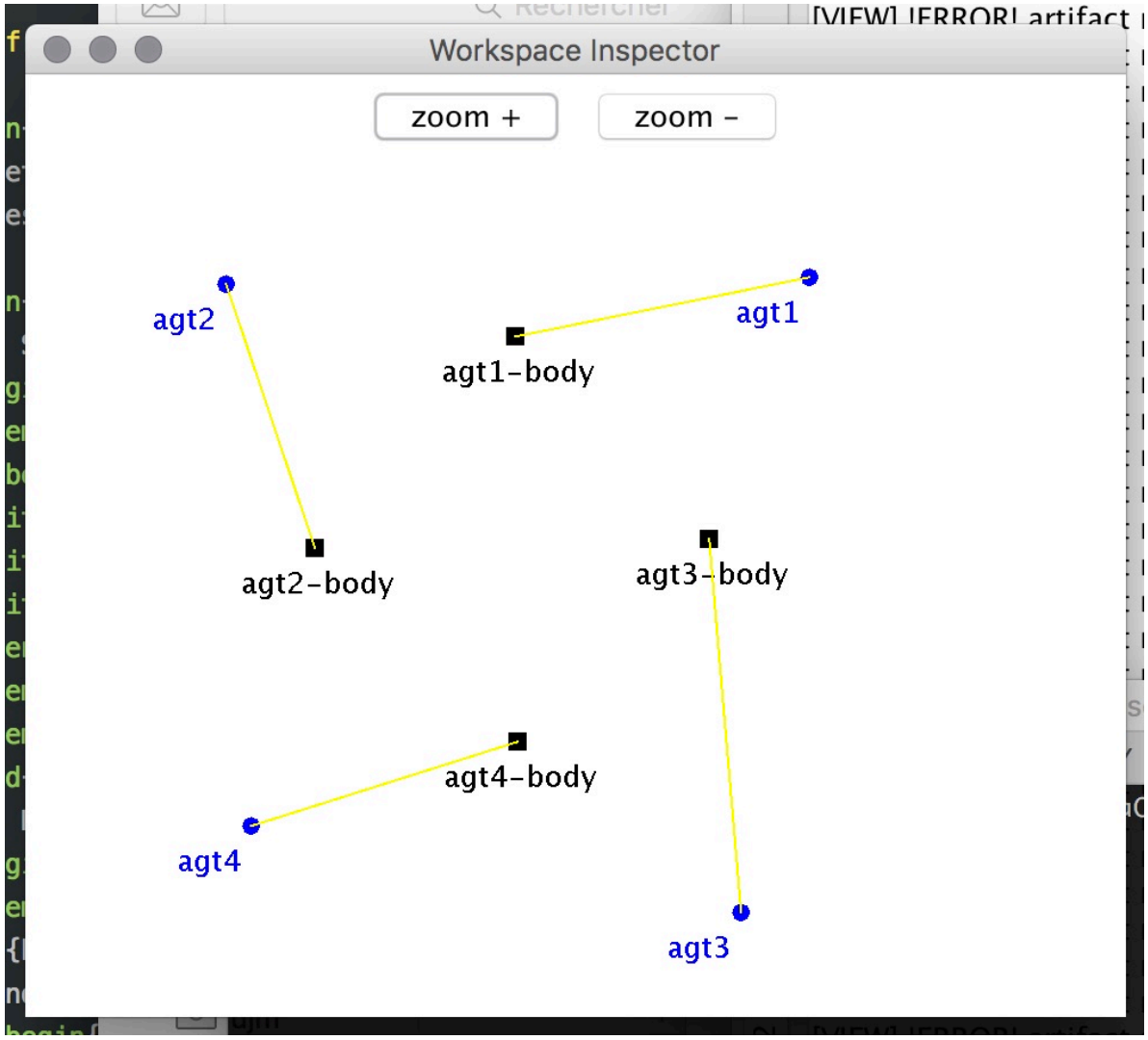
JaCaMo multi-agent system development



Runs also as an http server

Environment's inspector

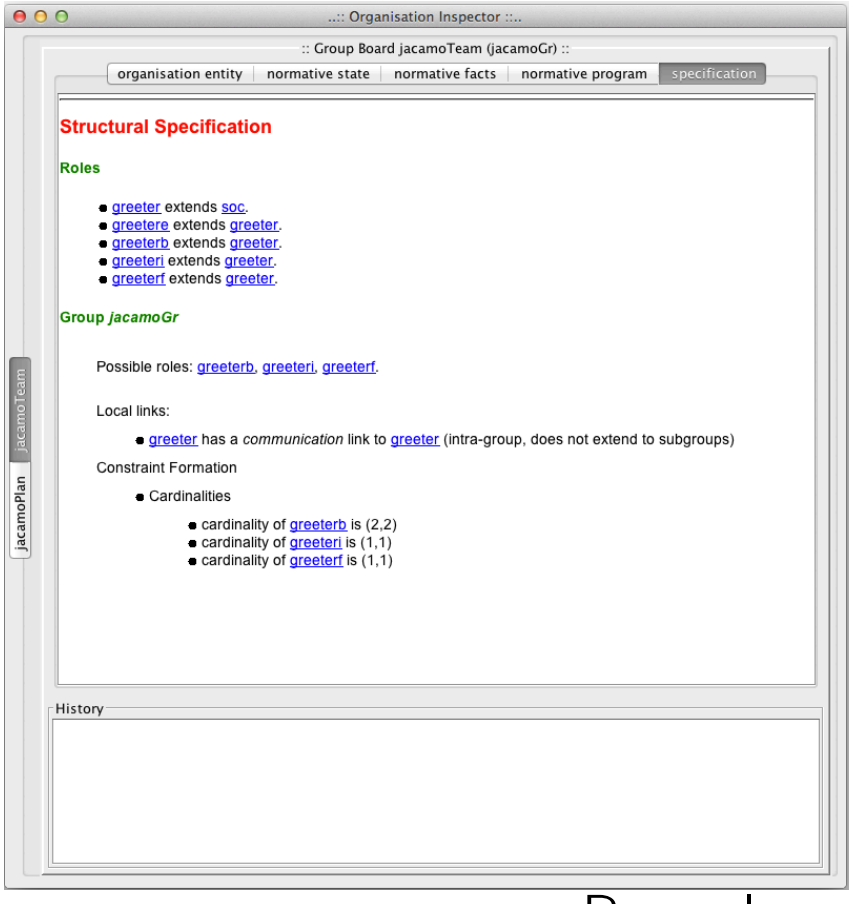
JaCaMo multi-agent system development

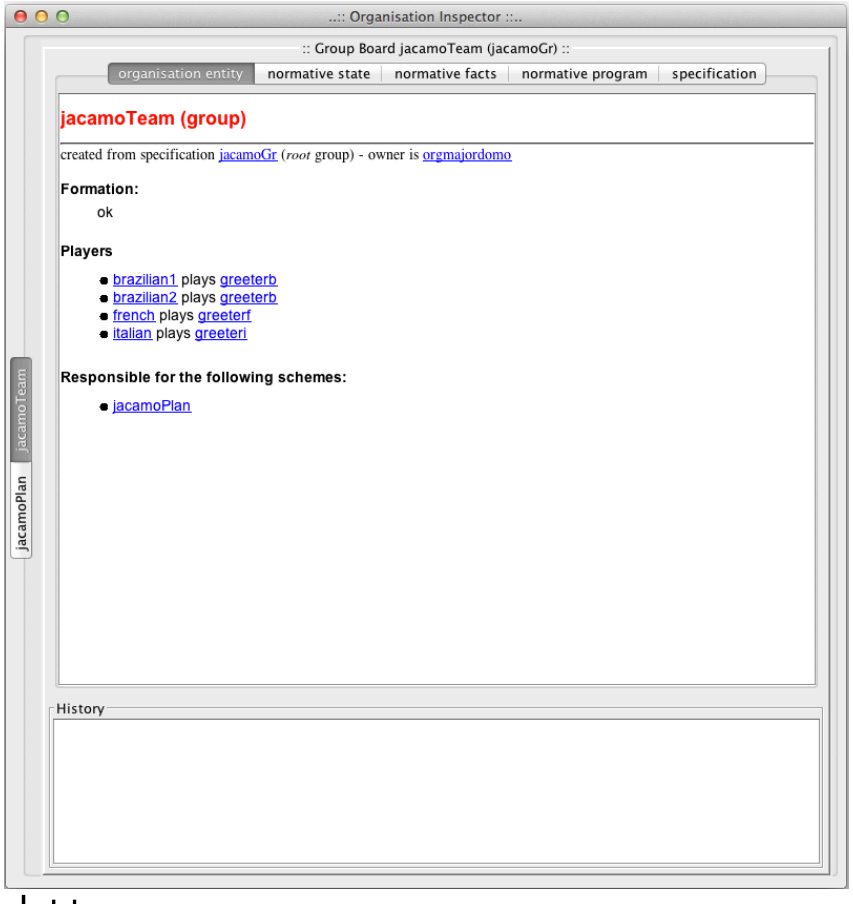


Runs also as an http server

Organization structure's inspector

JaCaMo multi-agent system development

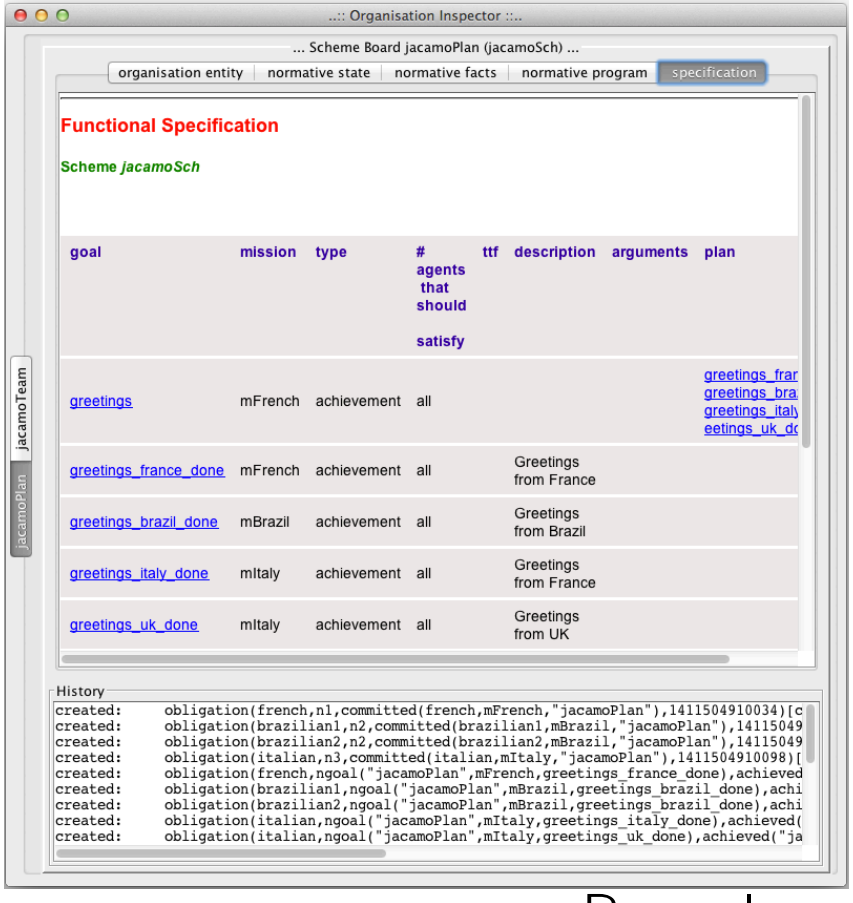


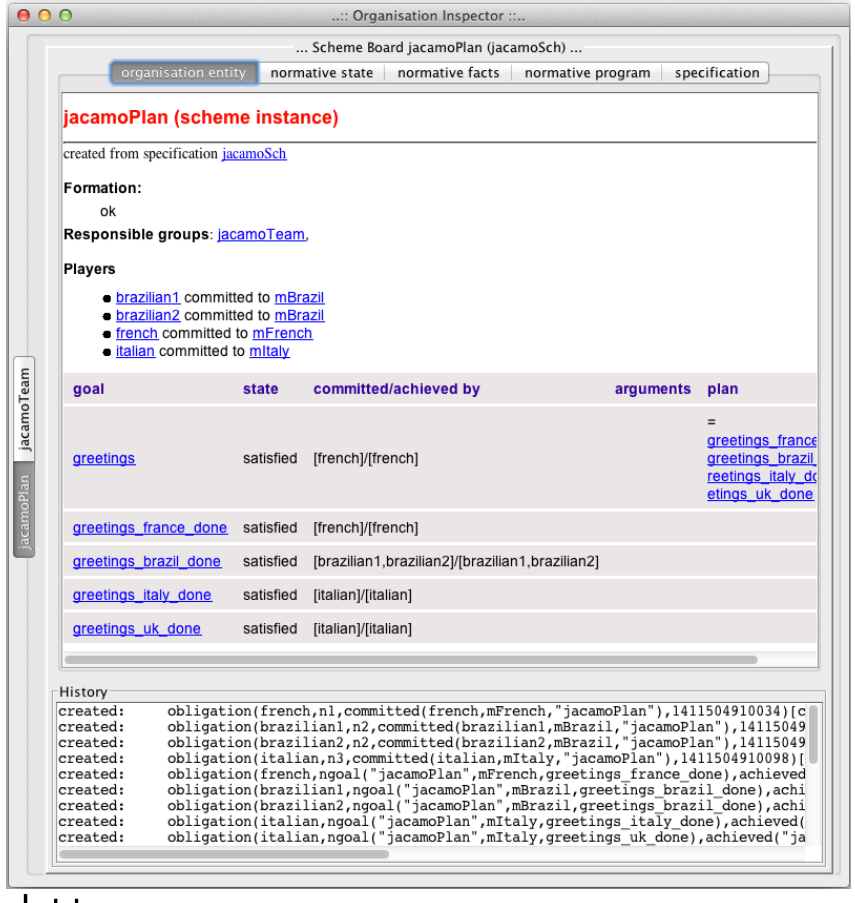


Runs also as an http server

Organization functioning's inspector

JaCaMo multi-agent system development





Runs also as an http server

Bibliography I

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2011).

Multi-agent oriented programming with jacamo.

Science of Computer Programming, pages –.

Bordini, R. H., Hübner, J. F., and Wooldrige, M. (2007).

Programming Multi-Agent Systems in AgentSpeak using Jason.

Wiley Series in Agent Technology. John Wiley & Sons.

de Brito, M., Hübner, J. F., and Boissier, O. (2015).

Bringing constitutive dynamics to situated artificial institutions.

In *Proc. of 17th Portuguese Conference on Artificial Intelligence (EPIA 2015)*, volume 9273 of *LNCS*, pages 624–637. Springer.

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.

Omicini, A., Ricci, A., and Viroli, M. (2008).

Artifacts in the A&A meta-model for multi-agent systems.

Autonomous Agents and Multi-Agent Systems, 17(3):432–456.

Bibliography II

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.

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.