Programming Multiagent Systems with JaCaMo

Olivier Boissier[1], Rafael H. Bordini[2],
Jomi F. Hübner[3], Alessandro Ricci[4]

[1]EMSE, France
Olivier.Boissier@emse.fr

[2]PUC-RS, Brazil
R.Bordini@pucrs.br

[3]DAS-UFSC, Brazil
jomi@das.ufsc.br

[4]University of Bologna, Italy
a.ricci@unibo.it

WESAAC 2013 — São Paulo

Outline of the course

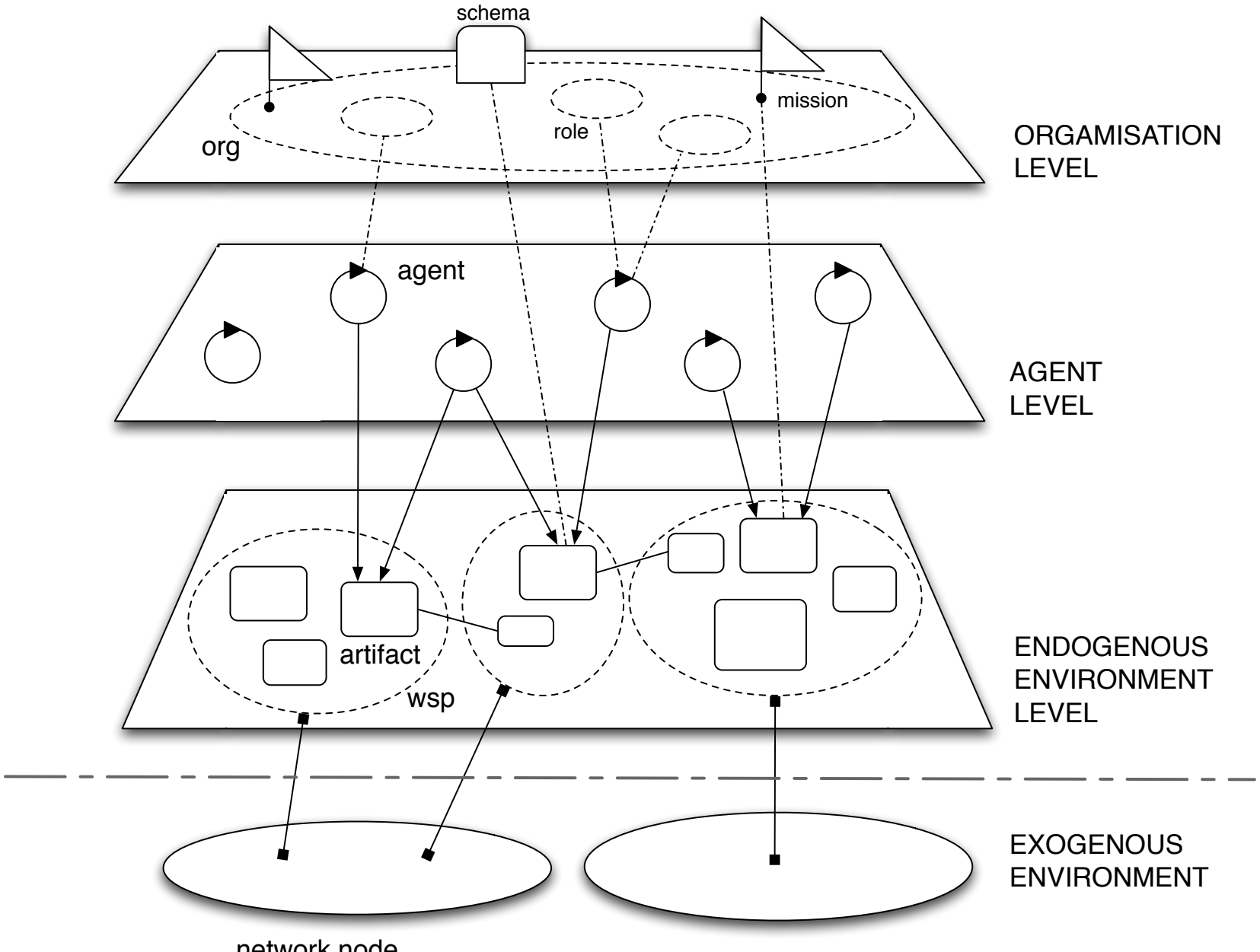
- Introduction
- AOP – Agent Oriented Programming: *Jason*
- EOP – Environment Oriented Programming: CArtAgO
- OOP – Organisation Oriented Programming: $\mathcal{M}$oise
- Conclusions
- Practical Exercise: a hands-on lab session!

Introduction

Abstractions in Multi-Agent Systems

- **Individual Agent Level**: autonomy, situatedness
  - **Cognitive Concepts**: beliefs, desires, goals, intentions, plans
  - **Reasoning Cycle**: sense/reason/act, reactive/pro-active behaviour
- **Environment Level**: resources and services that agents can access and control; sensing and acting in an environment
- **Social and Organisation Level**: cooperation, coordination, regulation patterns
  - **Roles**: rights, responsibilities, ...
  - **Organisational Rules**: constraints on roles and their interactions, norms, deadlines, ...
  - **Organisational Structures**: topology of interaction patterns and relations over activity control

Abstractions in Multi-Agent Systems



Multi Agent Oriented Programming!

- MAS is not only agents
- MAS is not only organisation
- MAS is not only environment
- MAS is not only interaction

MAS has many dimensions
all as **first class entities**

Agent Oriented Programming
— **AOP** —

Agent Oriented Programming

- Use of **mentalistic** notions and a **societal** view of computation [Shoham, 1993]
- Heavily influence by the BDI architecture and reactive planning systems
- Various language constructs for the sophisticated abstractions used in AOSE
  - Agent: Belief, Goal, Intention, Plan
  - Organisation: Group, Role, Norm, Interactions
  - Environment: Artifacts, Percepts, Actions

Agent Oriented Programming

Features

- **Reacting** to events × **long-term** goals
- Course of **actions** depends on **circumstance**
- **Plan failure** (dynamic environments)
- **Rational** behaviour
- **Social** ability
- Combination of **theoretical** and **practical** reasoning

Literature

Books: [Bordini et al., 2005a], [Bordini et al., 2009]

Proceedings: ProMAS, DALT, LADS, ... [Baldoni et al., 2010, Dastani et al., 2010, Hindriks et al., 2009, Baldoni et al., 2009, Dastani et al., 2008b, Baldoni et al., 2008, Dastani et al., 2008a, Bordini et al., 2007a, Baldoni and Endriss, 2006, Bordini et al., 2006b, Baldoni et al., 2006, Bordini et al., 2005b, Leite et al., 2005, Dastani et al., 2004, Leite et al., 2004]

Surveys: [Bordini et al., 2006a], [Fisher et al., 2007] ...

Languages of historical importance: Agent0 [Shoham, 1993], AgentSpeak(L) [Rao, 1996], MetateM [Fisher, 2005], 3APL [Hindriks et al., 1997], Golog [Giacomo et al., 2000]

Other prominent languages: *Jason* [Bordini et al., 2007b], Jadex [Pokahr et al., 2005], 2APL [Dastani, 2008], GOAL [Hindriks, 2009], JACK [Winikoff, 2005]

But many others languages and platforms...

Some Languages and Platforms

Jason (Hübner, Bordini, ...); 3APL and 2APL (Dastani, van Riemsdijk, Meyer, Hindriks, ...); Jadex (Braubach, Pokahr); MetateM (Fisher, Guidini, Hirsch, ...); ConGoLog (Lesperance, Levesque, ... / Boutilier – DTGolog); Teamcore/ MTDP (Milind Tambe, ...); IMPACT (Subrahmanian, Kraus, Dix, Eiter); CLAIM (Amal El Fallah-Seghrouchni, ...); GOAL (Hindriks); BRAHMS (Sierhuis, ...); SemantiCore (Blois, ...); STAPLE (Kumar, Cohen, Huber); Go! (Clark, McCabe); Bach (John Lloyd, ...); MINERVA (Leite, ...); SOCS (Torroni, Stathis, Toni, ...); FLUX (Thielscher); JIAC (Hirsch, ...); JADE (Agostino Poggi, ...); JACK (AOS); Agentis (Agentis Software); Jackdaw (Calico Jack); ...

The State of Multi-Agent Programming

- Already the **right** way to implement MAS is to use an AOSE Methodology (Prometheus, Gaia, Tropos, ...) and an MAS Programming Language!
- Many agent languages have efficient and stable interpreters — used extensively in teaching
- All have some programming tools (IDE, tracing of agents' mental attitudes, tracing of messages exchanged, etc.)
- Finally integrating with **social** aspects of MAS
- Growing user base

Jason

AgentSpeak

The foundational language for *Jason*

- Originally proposed by Rao [Rao, 1996]
- Programming language for BDI agents
- Elegant notation, based on **logic programming**
- Inspired by PRS (Georgeff & Lansky), dMARS (Kinny), and BDI Logics (Rao & Georgeff)
- Abstract programming language aimed at theoretical results

Jason

A practical implementation of a variant of AgentSpeak

- *Jason* implements the **operational semantics** of a variant of AgentSpeak
- Has various extensions aimed at a more **practical** programming language (e.g. definition of the MAS, communication, ...)
- Highly customised to simplify **extension** and **experimentation**
- Developed by **Jomi F. Hübner** and **Rafael H. Bordini**

Main Language Constructs and Runtime Structures

Beliefs: represent the information available to an agent (e.g. about the environment or other agents)

Goals: represent states of affairs the agent wants to bring about

Plans: are recipes for action, representing the agent's know-how

Events: happen as consequence to changes in the agent's beliefs or goals

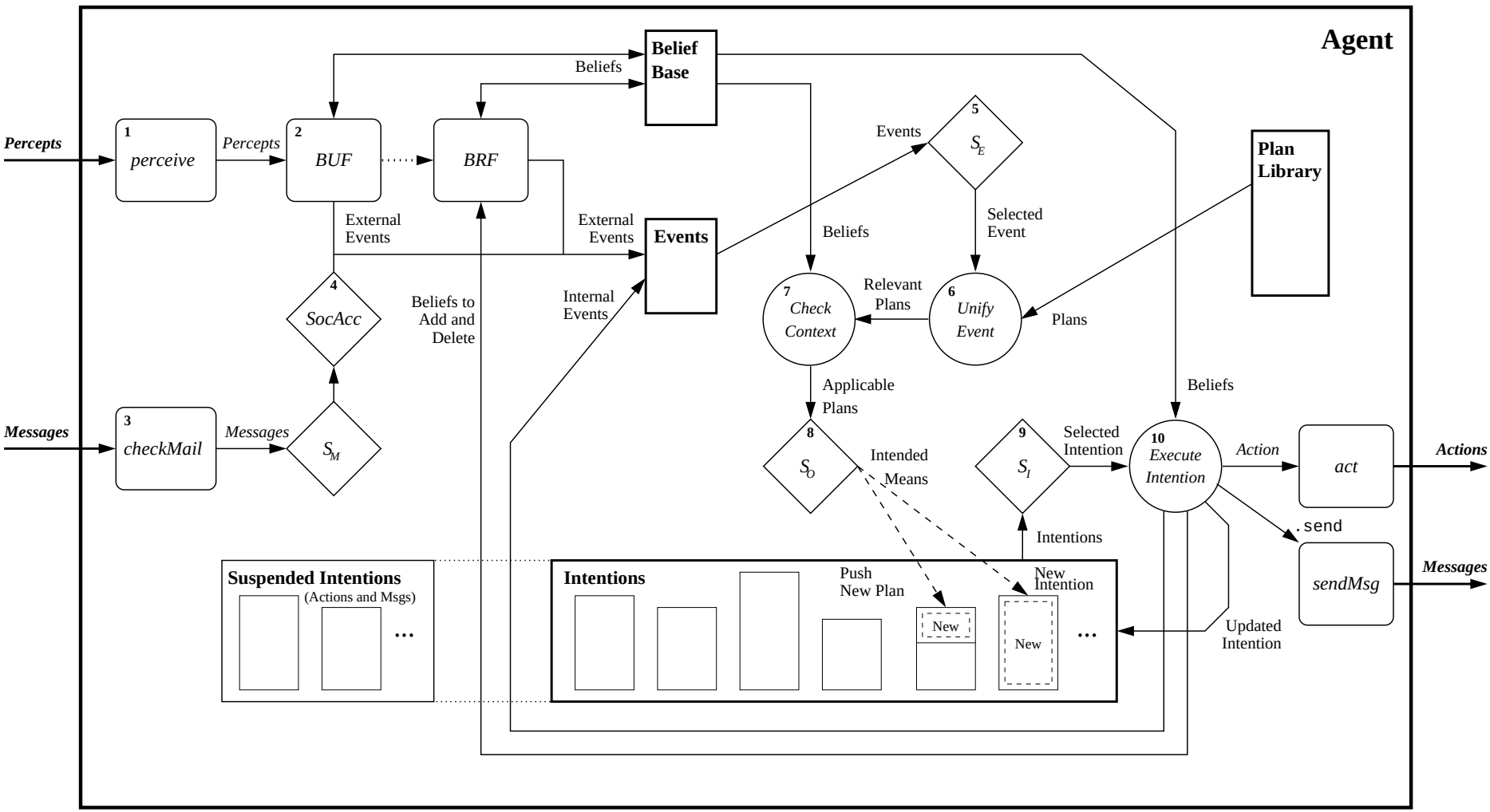
Intentions: plans instantiated to achieve some goal

Jason Interpreter

Basic Reasoning cycle

- perceive the environment and update belief base
- process new messages
- select event
- select **relevant** plans
- select **applicable** plans
- create/update intention
- select intention to execute

Jason Rreasoning Cycle



Beliefs — Representation

Syntax

Beliefs are represented by annotated literals of first order logic

`functor(`$term_1$`, ..., `$term_n$`)[`$annot_1$`, ..., `$annot_m$`]`

Example (belief base of agent Tom)

```
red(box1)[source(percept)].
friend(bob,alice)[source(bob)].
lier(alice)[source(self),source(bob)].
~lier(bob)[source(self)].
```

Beliefs — Dynamics I

by perception

beliefs annotated with source(percept) are automatically updated accordingly to the perception of the agent

by intention

the **plan operators** + and - can be used to add and remove beliefs annotated with source(self) (**mental notes**)

```
+lier(alice); // adds lier(alice)[source(self)]
-lier(john); // removes lier(john)[source(self)]
```

Beliefs — Dynamics II

by communication

when an agent receives a **tell** message, the content is a new belief annotated with the sender of the message

```
.send(tom,tell,lier(alice)); // sent by bob
// adds lier(alice)[source(bob)] in Tom's BB
...
.send(tom,untell,lier(alice)); // sent by bob
// removes lier(alice)[source(bob)] from Tom's BB
```

Goals — Representation

Types of goals

- Achievement goal: goal **to do**
- Test goal: goal **to know**

Syntax

Goals have the same syntax as beliefs, but are prefixed by
! (achievement goal) or
? (test goal)

Example (Initial goal of agent Tom)

```
!write(book).
```

Goals — Dynamics I

by intention

the **plan operators** **!** and **?** can be used to add a new goal annotated with source(self)

```
...
// adds new achievement goal !write(book)[source(self)]
!write(book);

// adds new test goal ?publisher(P)[source(self)]
?publisher(P);
...
```

Goals — Dynamics II

by communication – achievement goal

when an agent receives an **achieve** message, the content is a new 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
...
.send(tom,unachieve,write(book)); // sent by Bob
// removes goal write(book)[source(bob)] for Tom
```

Goals — Dynamics III

by communication – test goal

when an 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 ?publisher(P)[source(bob)] for Tom
// the response of Tom will unify with Answer
```

Triggering Events — Representation

- Events happen as consequence to changes in the agent's beliefs or goals
- An agent reacts to events by executing **plans**
- Types of **plan 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 — Representation

An AgentSpeak plan has the following general structure:

triggering_event : context <- body.

where:

- the triggering event denotes the events that the plan is meant to handle
- the context represent the circumstances in which the plan can be used
- the body is the course of action to be used to handle the event if the context is believed true at the time a plan is being chosen to handle the event

Plans — Operators for Plan **Context**

Boolean operators

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

Arithmetic operators

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

Plans — Operators for Plan **Body**

A plan body may contain:

- Belief operators (+, -, -+)
- Goal operators (!, ?, !!)
- Actions (internal/external) and Constraints

Example (plan body)

```
+rain : time_to_leave(T) & clock.now(H) & H >= T
   <- !g1;          // new sub-goal
      !!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
      jia.get(X);   // internal action
      X > 10;       // constraint to carry on
      close(door).// external action
```

Plans — Example

```
+green_patch(Rock)[source(percept)]
   :  not battery_charge(low)
   <- ?location(Rock,Coordinates);
      !at(Coordinates);
      !examine(Rock).

+!at(Coords)
   :  not at(Coords) & safe_path(Coords)
   <- move_towards(Coords);
      !at(Coords).
+!at(Coords)
   :  not at(Coords) & not safe_path(Coords)
   <- ...
+!at(Coords) :  at(Coords).
```

Plans — Dynamics

The plans that form the plan library of the agent come from

- initial plans defined by the programmer
- plans added dynamically and intentionally by
  - .add_plan
  - .remove_plan
- plans received from
  - **tellHow** messages
  - **untellHow**

Strong Negation

Example

```
+!leave(home)
   :  ~raining
   <- open(curtains); ...

+!leave(home)
   :  not raining & not ~raining
   <- .send(mum,askOne,raining,Answer,3000); ...
```

Prolog-like Rules in the Belief Base

Example

```
likely_color(Obj,C) :-
    colour(Obj,C)[degOfCert(D1)] &
    not (colour(Obj,_)[degOfCert(D2)] & D2 > D1) &
    not ~colour(C,B).
```

Plan Annotations

- Like beliefs, plans can also have **annotations**, which go in the plan **label**
- Annotations contain meta-level information for the plan, which selection functions can take into consideration
- The annotations in an intended plan instance can be changed **dynamically** (e.g. to change intention priorities)
- There are some pre-defined plan annotations, e.g. to force a breakpoint at that plan or to make the whole plan execute atomically

Example (an annotated plan)

```
@myPlan[chance_of_success(0.3), usual_payoff(0.9),
        any_other_property]
+!g(X) : c(t) <- a(X).
```

Failure Handling: Contingency Plans

Example (an agent blindly committed to g)

```
+!g : g.

+!g : ... <- ... ?g.

-!g : true <- !g.
```

Meta Programming

Example (an agent that asks for plans *on demand*)

```
-!G[error(no_relevant)] : teacher(T)
   <- .send(T, askHow, { +!G }, Plans);
      .add_plan(Plans);
      !G.
```

in the event of a failure to achieve **any** *goal G due to no relevant plan, asks a teacher for plans to achieve G and then try G again*

- The failure event is annotated with the error type, line, source, ... error(no_relevant) means no plan in the agent's plan library to achieve G
- { +!G } is the syntax to enclose triggers/plans as terms

Internal Actions

- Unlike actions, internal actions do not change the environment
- Code to be executed as part of the agent reasoning cycle
- AgentSpeak is meant as a high-level language for the agent's practical reasoning and internal actions can be used for invoking legacy code elegantly

- Internal actions can be defined by the user in Java

```
libname.action_name(...)
```

Standard Internal Actions

- Standard (pre-defined) internal actions have an empty library name
  - `.print(`$term_1, term_2, \ldots$`)`
  - `.union(`$list_1$`,` $list_2$`,` $list_3$`)`
  - `.my_name(`var`)`
  - `.send(`ag`,`$perf$`,`$literal$`)`
  - `.intend(`$literal$`)`
  - `.drop_intention(`$literal$`)`
- Many others available for: printing, sorting, list/string operations, manipulating the beliefs/annotations/plan library, creating agents, waiting/generating events, etc.

Jason × Java I

Consider a very simple robot with two goals:

- when a piece of gold is seen, go to it
- when battery is low, go charge it

Jason × Java II

Example (Java code – go to gold)

```
public class Robot extends Thread {
   boolean seeGold, lowBattery;
   public void run() {
      while (true) {
          if (seeGold)
              a = randomDirection();
          else
              a = selectDirection(gold);


          doAction(go(a));

} } }
```

(how to code the charge battery behaviour?)

Jason × Java III

Example (Java code – go to gold)

```
public class Robot extends Thread {
   boolean seeGold, lowBattery;
   public void run() {
      while (true) {
          if (seeGold)
              a = randomDirection();
          else
              a = selectDirection(gold);

          if (lowBattery) charge();
          doAction(go(a));
          if (lowBattery) charge();
} } }
```

(note where the tests for low battery have to be done)

Jason × Java IV

Example (*Jason* code)

```
direction(gold)   :- see(gold).
direction(random) :- not see(gold).

+!find(gold)                          // long term goal
   <- ?direction(A);
      go(A);
      !!find(gold).
+battery(low)                         // reactivity
   <- !charge.

^!charge[state(started)]              // goal meta-events
   <- .suspend(find(gold)).
^!charge[state(finished)]
   <- .resume(find(gold)).
```

Jason × Prolog

- With the *Jason* extensions, nice separation of theoretical and **practical reasoning**
- BDI architecture allows
  - long-term goals (goal-based behaviour)
  - reacting to changes in a dynamic environment
  - handling multiple foci of attention (concurrency)
- Acting on an environment and a higher-level conception of a distributed system

Communication **Infrastructure**

Various communication and execution management infrastructures can be used with *Jason*:

- Centralised: all agents in the same machine, one thread by agent, very fast
- Centralised (pool): all agents in the same machine, fixed number of thread, allows thousands of agents
- Jade: distributed agents, FIPA-ACL
- ... others defined by the user (e.g. AgentScape)

MAS Configuration Language I

- Simple way of defining a multi-agent system

Example (MAS that uses JADE as infrastructure)

```
MAS my_system {
    infrastructure: Jade
    environment: robotEnv
    agents:
         c3po;
         r2d2 at jason.sourceforge.net;
         bob #10; // 10 instances of bob
    classpath: "../lib/graph.jar";
}
```

MAS Configuration Language II

- Configuration of event handling, frequency of perception, user-defined settings, customisations, etc.

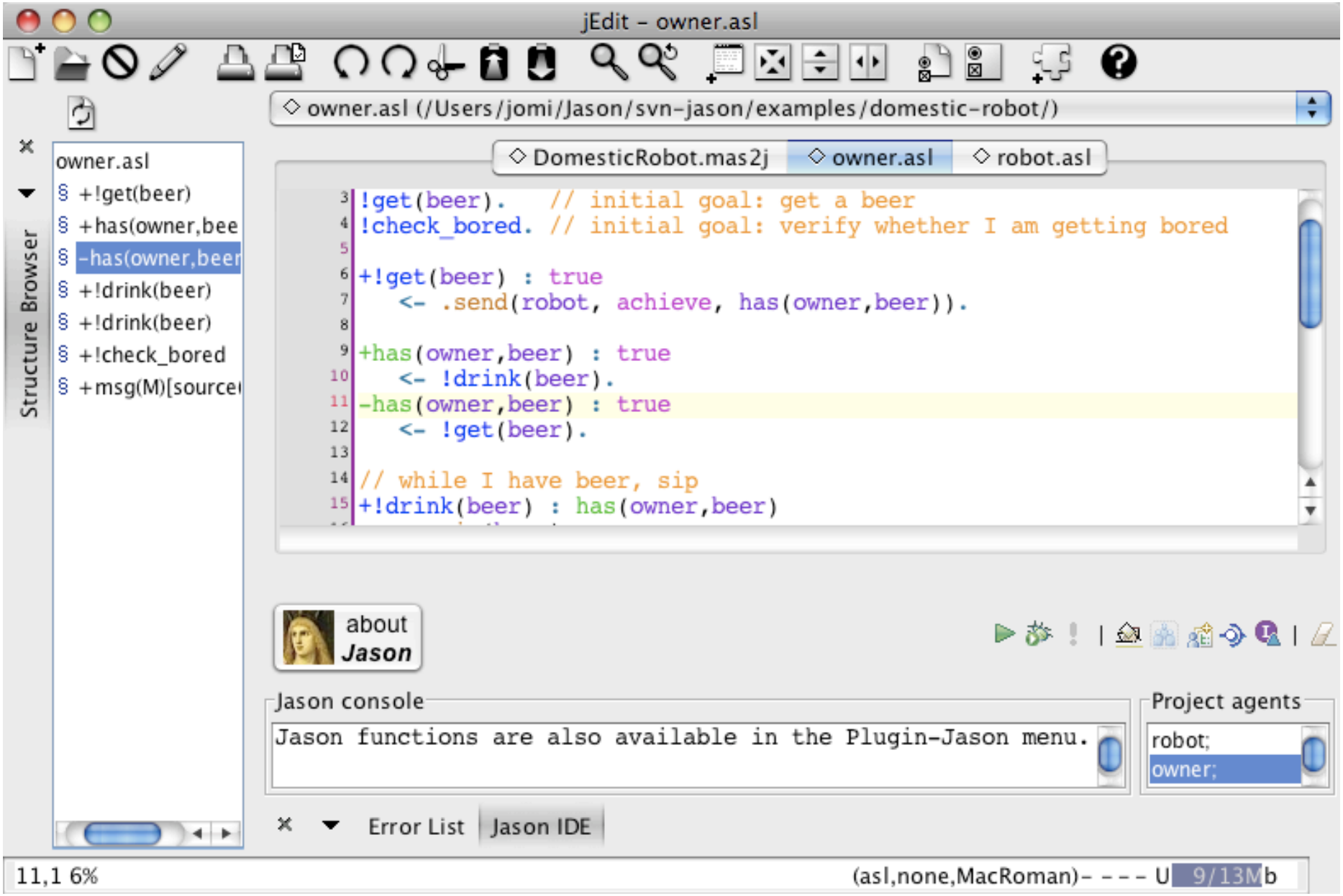
Example (MAS with customised agent)

```
MAS custom {
 agents: bob [verbose=2,paramters="sys.properties"]
             agentClass MyAg
             agentArchClass MyAgArch
             beliefBaseClass jason.bb.JDBCPersistentBB(
                 "org.hsqldb.jdbcDriver",
                 "jdbc:hsqldb:bookstore",
                 ...
}
```

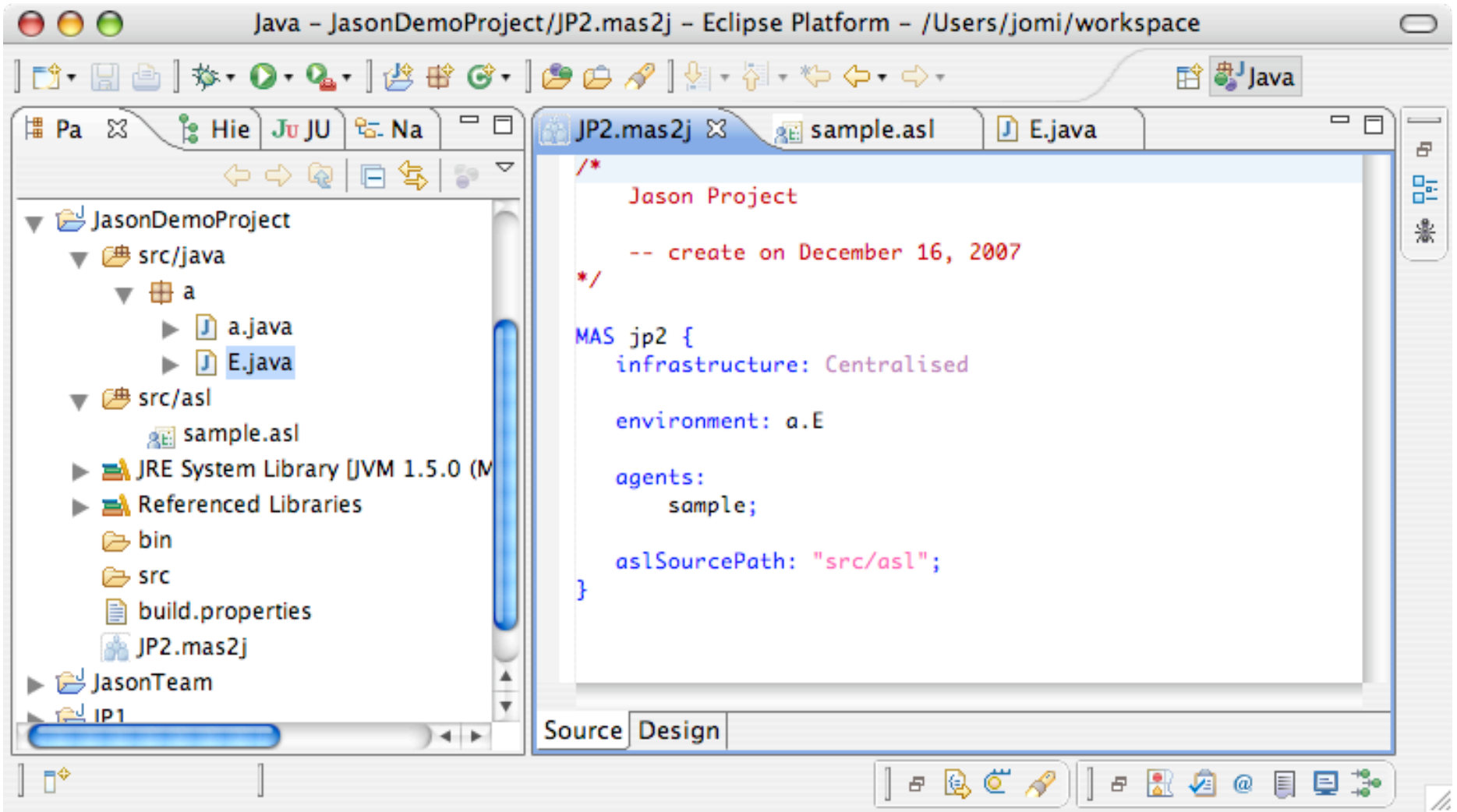
Jason Customisations

- **Agent** class customisation:
  selectMessage, selectEvent, selectOption, selectIntetion, buf, brf, ...
- Agent **architecture** customisation:
  perceive, act, sendMsg, checkMail, ...
- **Belief base** customisation:
  add, remove, contains, ...
  - Example available with *Jason*: persistent belief base (in text files, in data bases, ...)

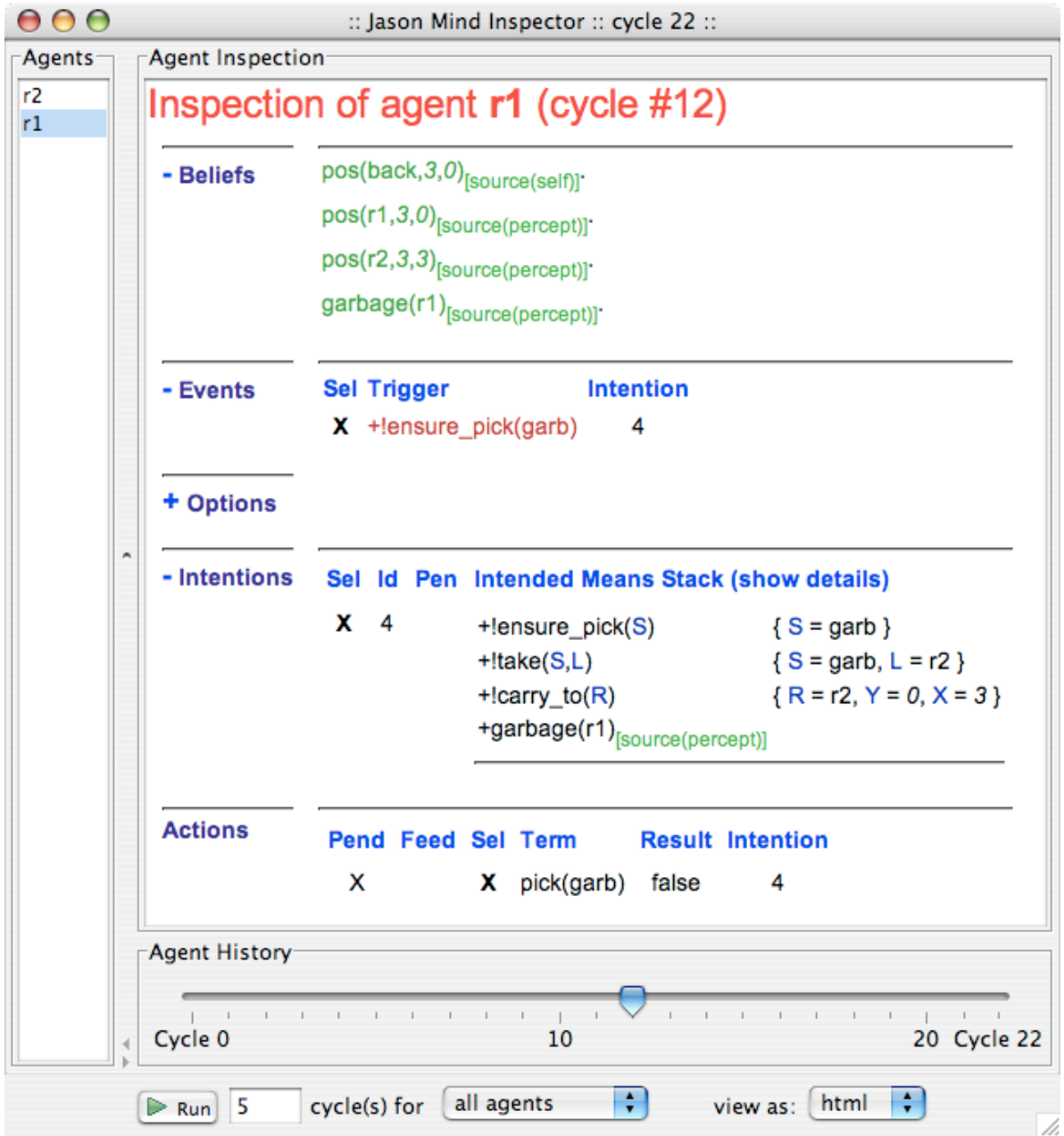
jEdit Plugin



Eclipse Plugin



Mind Inspector



Summary

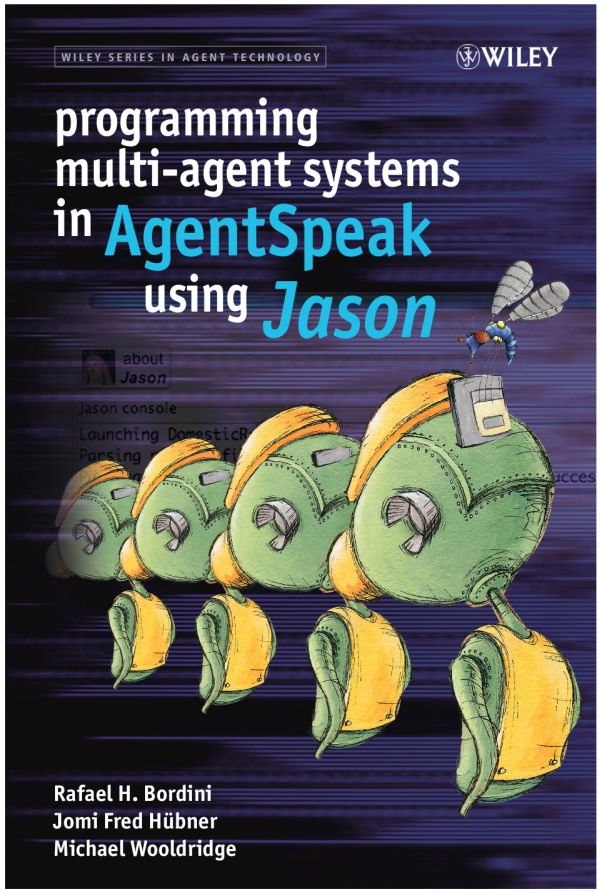
- AgentSpeak
  - Logic + BDI
  - Agent programming language
- *Jason*
  - AgentSpeak interpreter
  - Implements the operational semantics of AgentSpeak
  - Speech-act based communicaiton
  - Highly customisable
  - Useful tools
  - Open source
  - Open issues

Acknowledgements

- Many thanks to the
  - Various colleagues acknowledged/referenced throughout these slides
  - *Jason* users for helpful feedback
  - CNPq for supporting some of our current researh

Further Resources

- `http://jason.sourceforge.net`
- R.H. Bordini, J.F. Hübner, and M. Wooldrige
  Programming Multi-Agent Systems in AgentSpeak using Jason
  John Wiley & Sons, 2007.



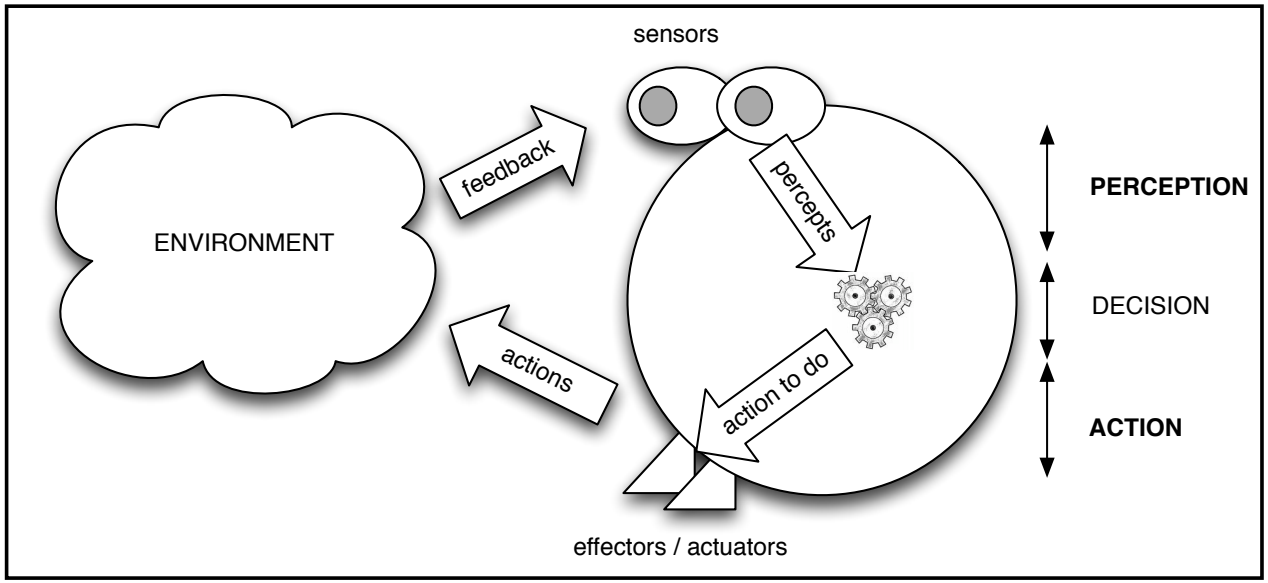
Environment Oriented Programming

— **EOP** —

Back to the Notion of Environment in MAS

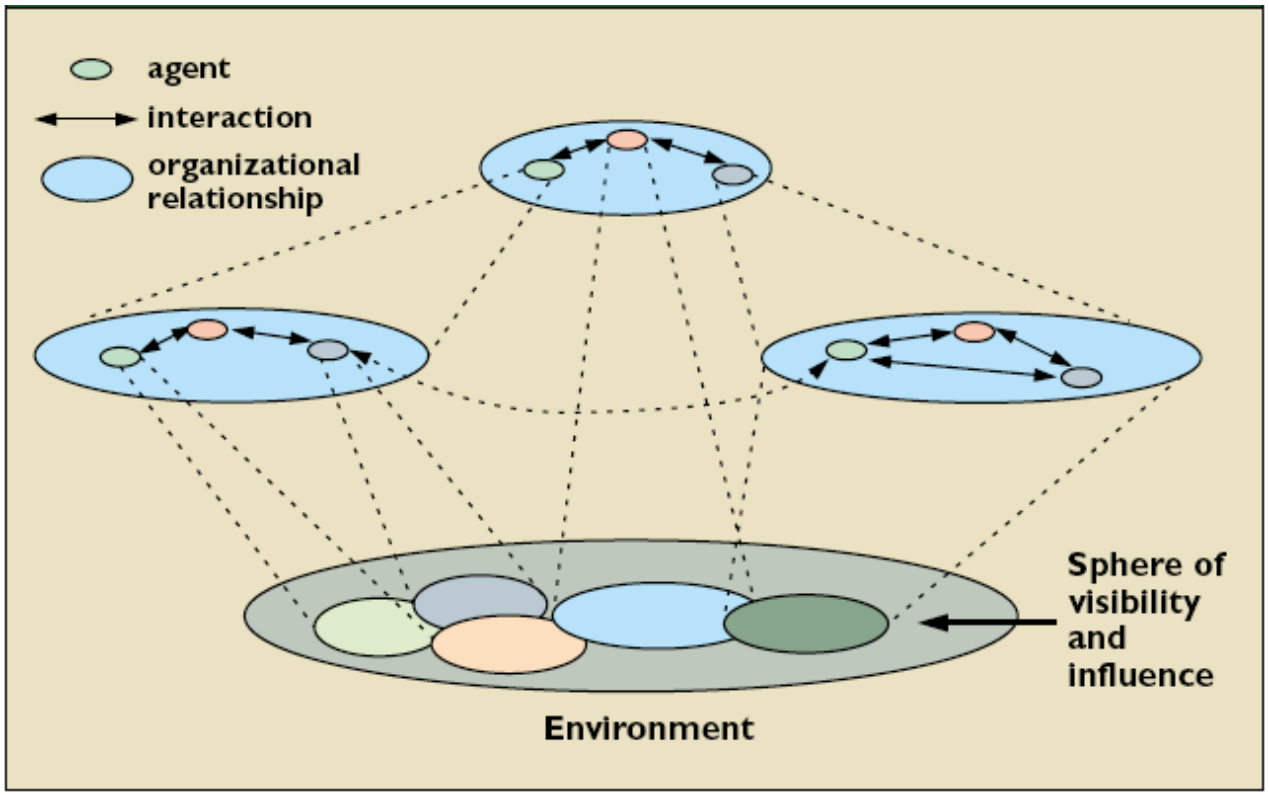
- The notion of environment is intrinsically related to the notion of agent and multi-agent system
  - **"An agent is a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design objective"** [Wooldridge, 2002]
  - **"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors."** [Russell and Norvig, 2003]
- Including both physical and software environments

Single Agent Perspective



- **Perception**
  - process inside agent inside of attaining awareness or understanding sensory information, creating percepts perceived form of external stimuli or their absence
- **Actions**
  - the means to affect, change or inspect the environment

Multi-Agent Perspective

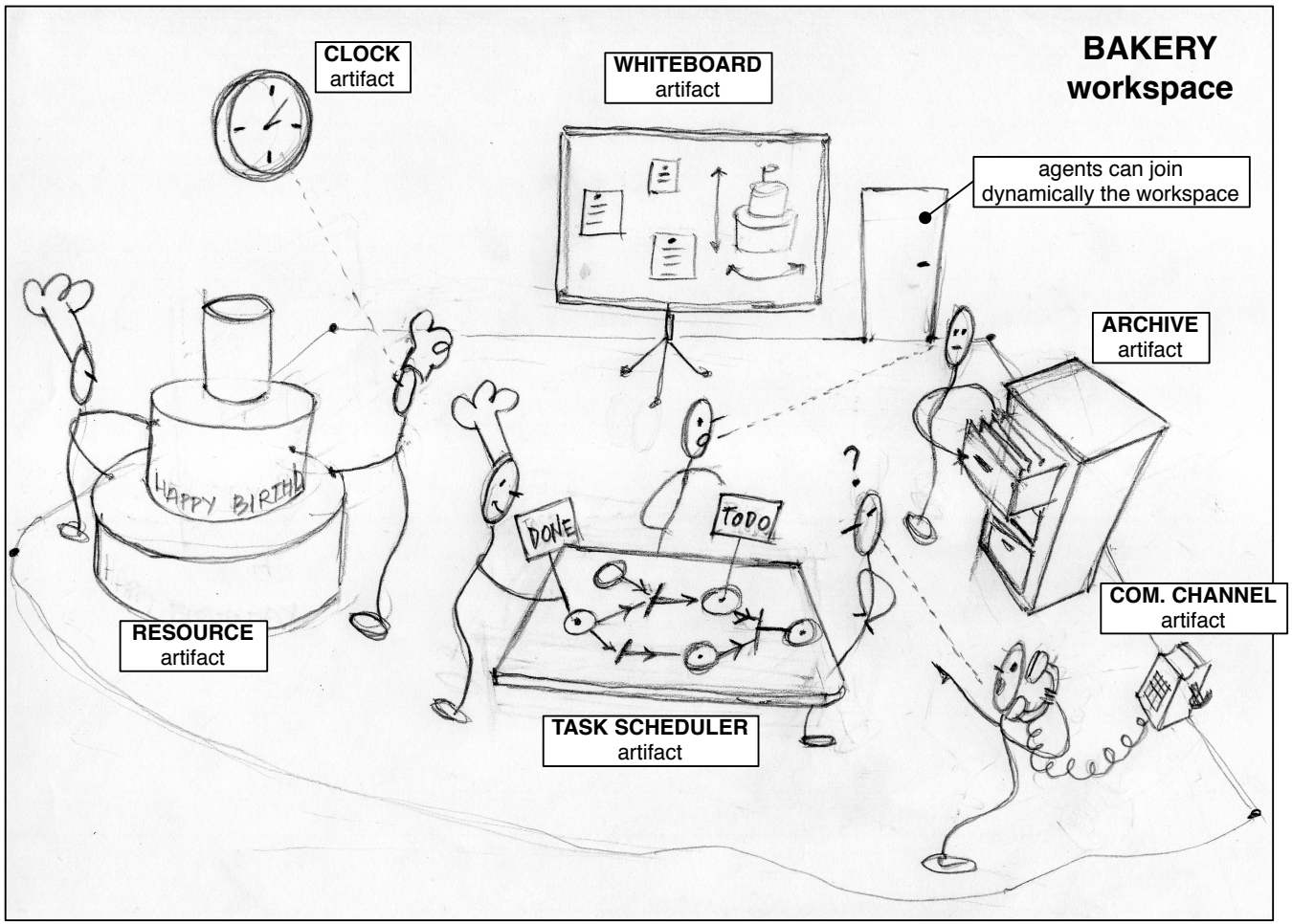


- In evidence
  - overlapping spheres of visibility and influence
  - ..which means: **interaction**

Why Environment Programming

- Basic level
  - to create testbeds for real/external environments
  - to ease the interface/interaction with existing software environments
- Advanced level
  - to uniformly **encapsulate** and **modularise** functionalities of the MAS out of the agents
    - typically related to interaction, coordination, organisation, security
    - **externalisation**
  - this implies changing the perspective on the environment
    - environment as a **first-class abstraction** of the MAS
    - **endogenous** environments (vs. exogenous ones)
    - **programmable** environments

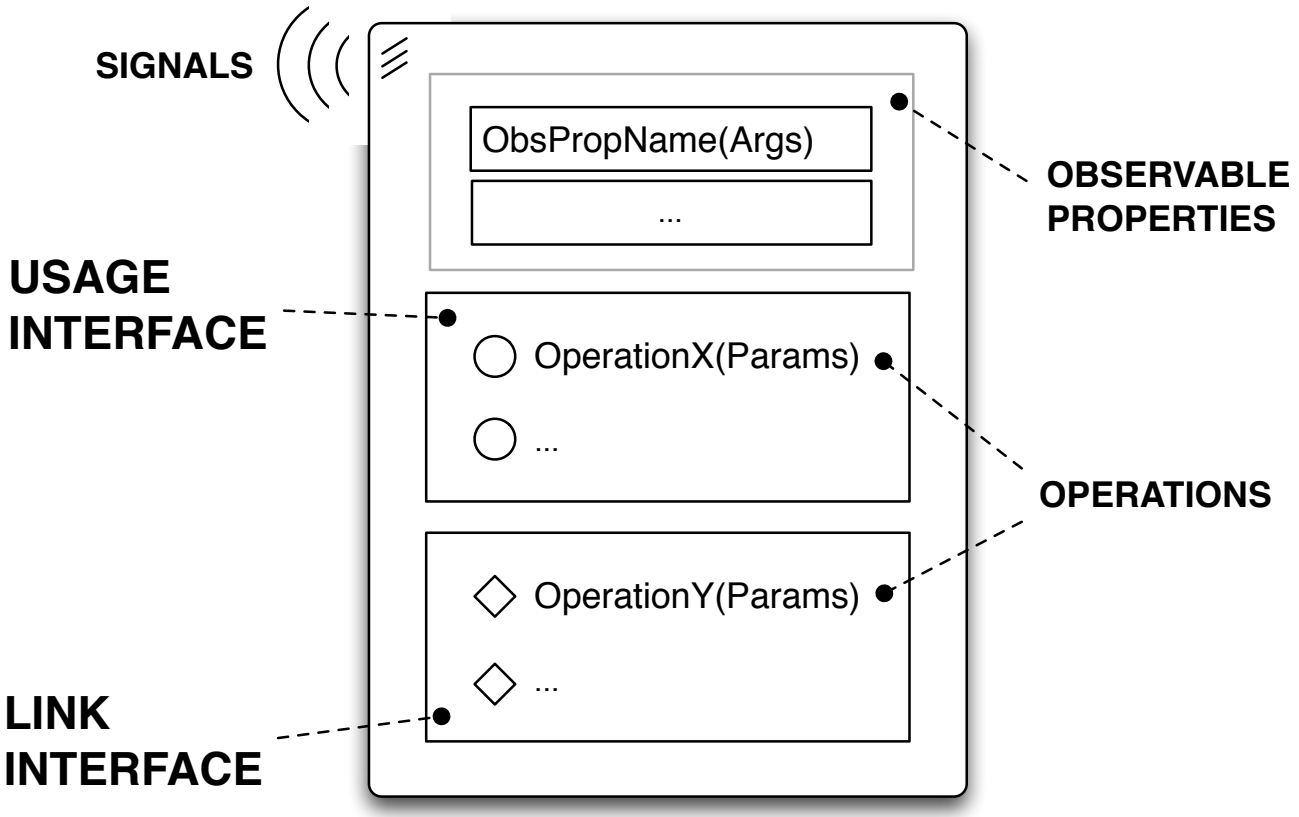
Agents and Artifacts (A&A) Conceptual Model: Background Human Metaphor



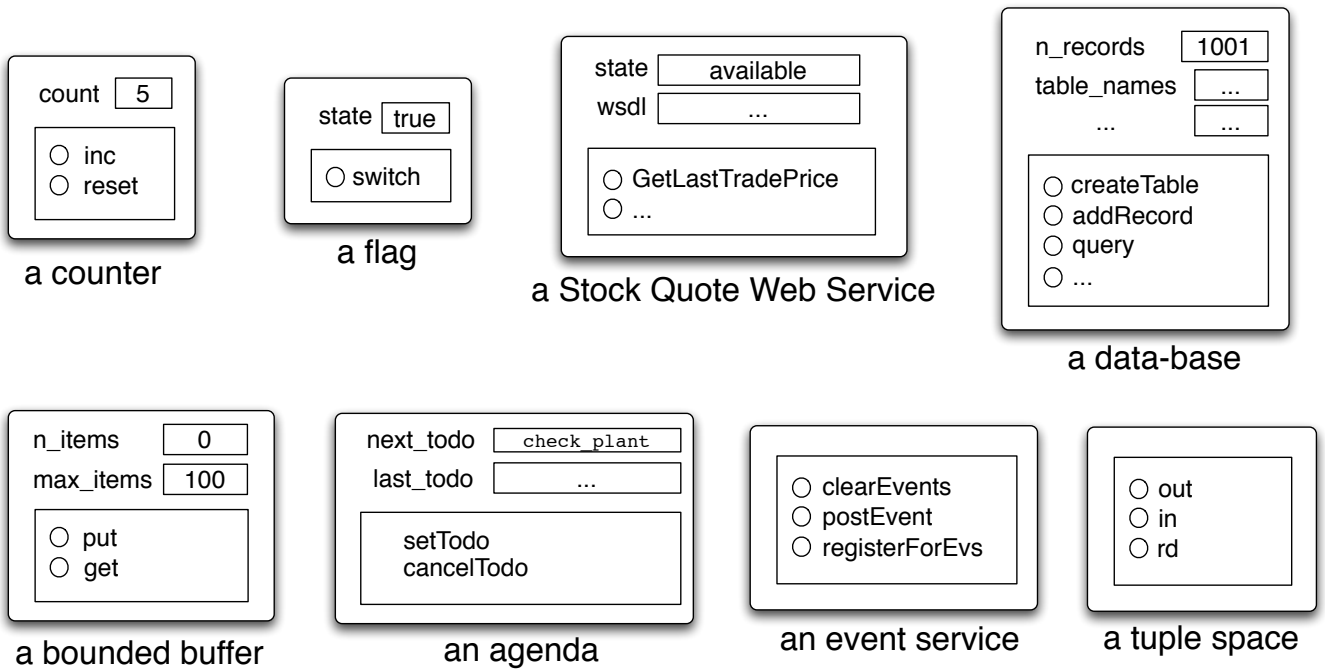
A&A Basic Concepts [Omicini et al., 2008]

- Agents
  - autonomous, goal-oriented pro-active entities
  - create and co-use artifacts for supporting their activities
    - besides direct communication
- Artifacts
  - non-autonomous, function-oriented, stateful entities
    - controllable and observable
  - modelling the tools and resources used by agents
    - designed by MAS programmers
- Workspaces
  - grouping agents & artifacts
  - defining the topology of the computational environment

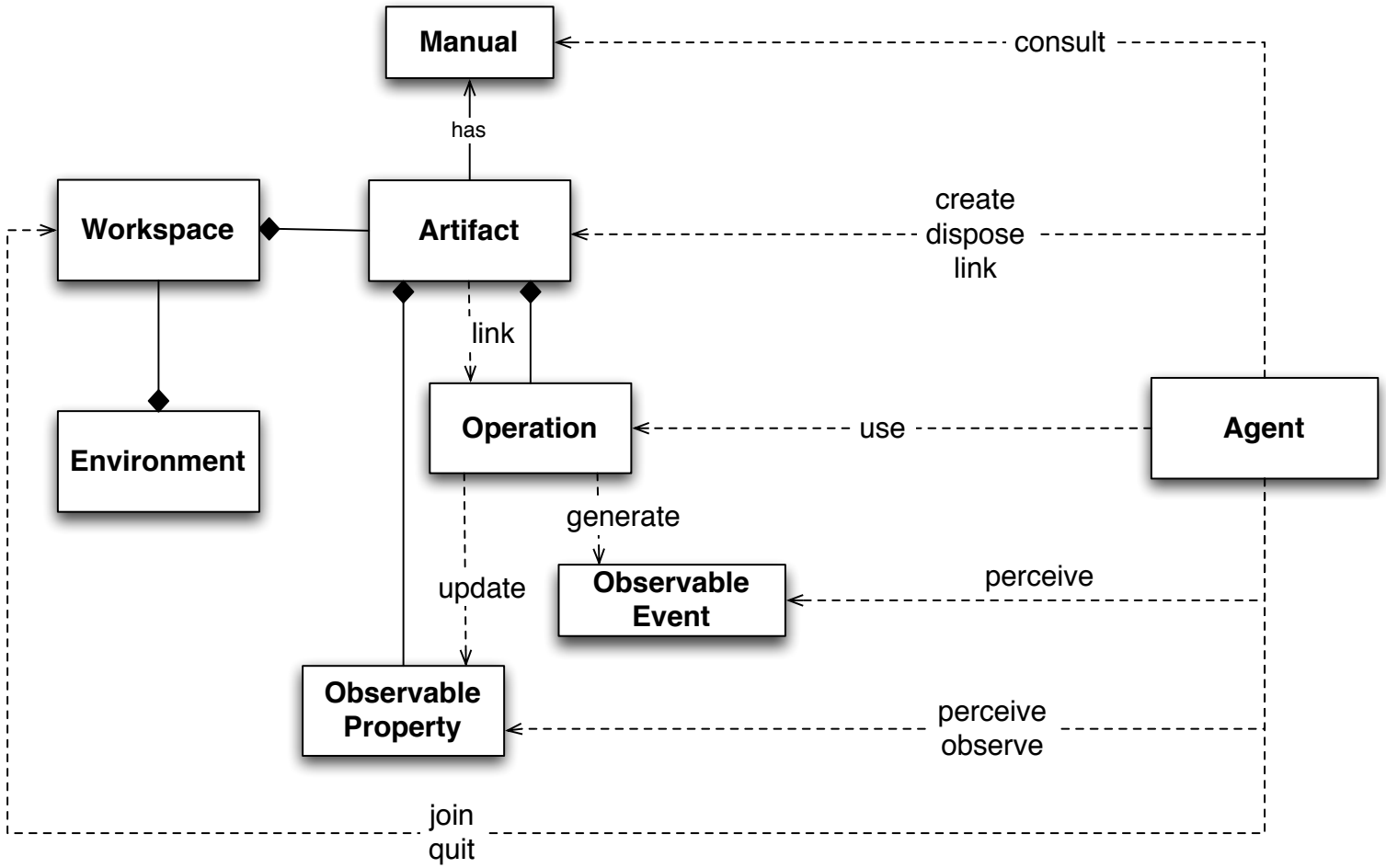
Artifact Abstract Representation



A World of Artifacts



A&A Meta-Model in More Detail [Ricci et al., 2010b]



A Simple Taxonomy

- Individual or personal artifacts
  - designed to provide functionalities for a single agent use
    - e.g. an agenda for managing deadlines, a library...
- Social artifacts
  - designed to provide functionalities for structuring and managing the interaction in a MAS
  - coordination artifacts [Omicini et al., 2004], organisation artifacts, ...
    - e.g. a blackboard, a game-board,...
- Boundary artifacts
  - to represent external resources/services
    - e.g. a printer, a Web Service
  - to represent devices enabling I/O with users
    - e.g GUI, console, etc.

Actions and Percepts in Artifact-Based Environments

- Explicit semantics defined by the (endogenous) environment [Ricci et al., 2010c]
  - success/failure semantics, execution semantics
  - defining the **contract** (in the SE acceptation) provided by the environment

actions $\longleftrightarrow$ artifacts' operation

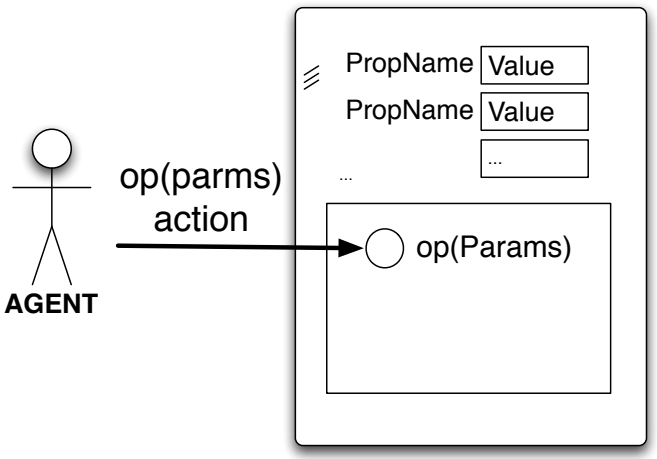
the action repertoire is given by the dynamic set of operations provided by the overall set of artifacts available in the workspace can be changed by creating/disposing artifacts

- action success/failure semantics is defined by operation semantics

percepts $\longleftrightarrow$ artifacts' observable properties + signals

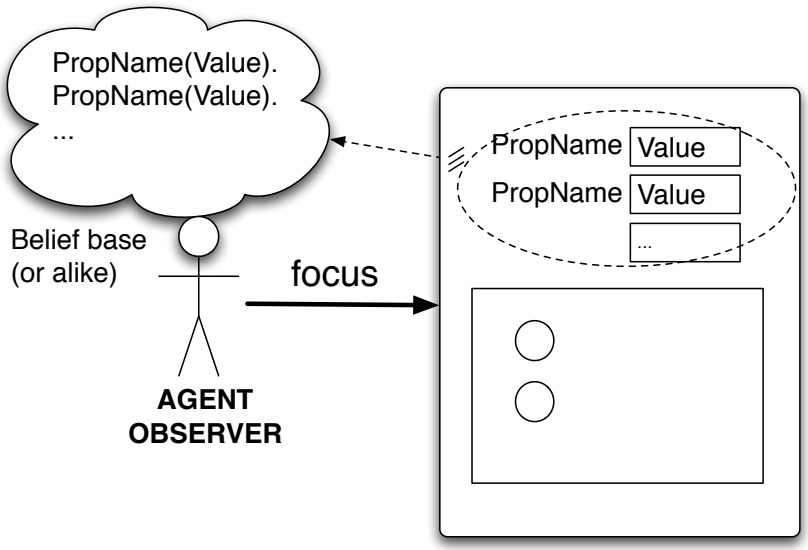
properties represent percepts about the state of the environment
signals represent percepts concerning events signalled by the environment

Interaction Model: Use



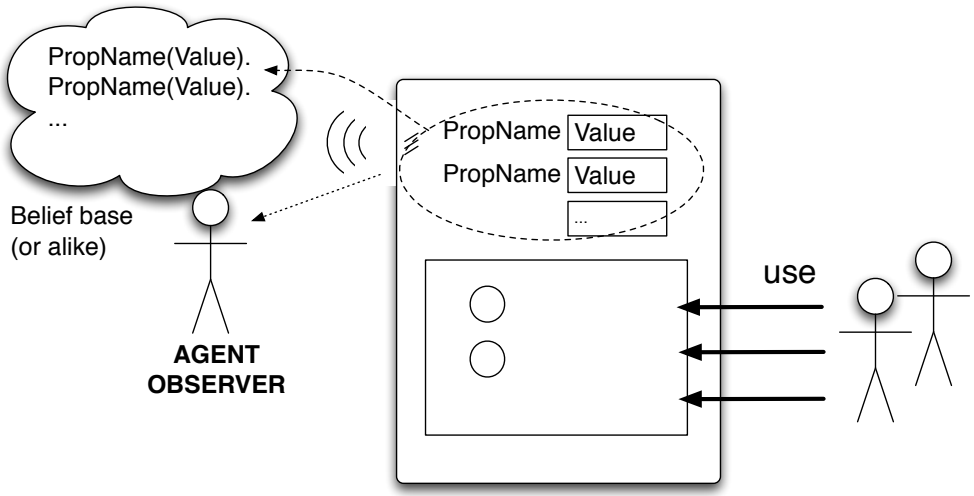
- Performing an action corresponds to triggering the execution of an operation
  - acting on artifact's usage interface

Interaction Model: Observation



- Agents can dynamically select which artifacts to observe
  - predefined `focus`/`stopFocus` actions

Interaction Model: Observation



- By focussing an artifact
  - observable properties are mapped into agent dynamic knowledge about the state of the world, as percepts
    - e.g. belief base
  - signals are mapped as percepts related to observable events

CArtAgO

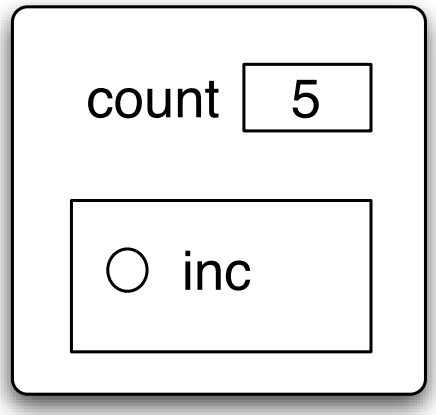
- Common ARtifact infrastructure for AGent Open environment (CArtAgO) [Ricci et al., 2009a]
- Computational framework / infrastructure to implement and run artifact-based environment [Ricci et al., 2007]
  - Java-based programming model for defining artifacts
  - set of basic API for agent platforms to work within artifact-based environment
- Distributed and open MAS
  - workspaces distributed on Internet nodes
    - agents can join and work in multiple workspace at a time
  - Role-Based Access Control (RBAC) security model
- Open-source technology
  - available at `http://cartago.sourceforge.net`

Example: A Simple Counter Artifact

```
class Counter extends Artifact {

  void init(){
    defineObsProp("count",0);
  }

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



- Some API spots
  - `Artifact` base class
  - `@OPERATION` annotation to mark artifact?s operations
  - set of primitives to work define/update/.. observable properties
  - `signal` primitive to generate signals

Example: User and Observer Agents

USER(S)

```
!create_and_use.

+!create_and_use : true
  <- !setupTool(Id);
     // use
     inc;
     // second use specifying the Id
     inc [artifact_id(Id)].

// create the tool
+!setupTool(C): true
  <- makeArtifact("c0","Counter",C).
```

OBSERVER(S)

```
!observe.

+!observe : true
  <- ?myTool(C);  // discover the tool
     focus(C).

+count(V)
  <- println("observed new value: ",V).

+tick [artifact_name(Id,"c0")]
  <- println("perceived a tick").

+?myTool(CounterId): true
  <- lookupArtifact("c0",CounterId).

-?myTool(CounterId): true
  <- .wait(10);
     ?myTool(CounterId).
```

- Working with the shared counter

Example: Internal Processes – A Clock

CLOCK

```
public class Clock extends Artifact {

  boolean working;
  final static long TICK_TIME = 100;

  void init(){ working = false; }

  @OPERATION void start(){
    if (!working){
      working = true;
      execInternalOp("work");
    } else {
      failed("already_working");
    }
  }

  @OPERATION void stop(){ working = false; }

  @INTERNAL_OPERATION void work(){
    while (working){
      signal("tick");
      await_time(TICK_TIME);
    }
  }
}
```

CLOCK USER AGENT

```
!test_clock.

+!test_clock
  <- makeArtifac("myClock","Clock",[],Id);
     focus(Id);
     +n_ticks(0);
     start;
     println("clock started.").

@plan1
+tick: n_ticks(10)
  <- stop;
     println("clock stopped.").

@plan2 [atomic]
+tick: n_ticks(N)
  <- -+n_ticks(N+1);
     println("tick perceived!").
```

- Internal operations
  - execution of operations triggered by other operations
  - implementing controllable **processes**

Other Features

- Other CArtAgO features not discussed in this lecture
  - linkability
    - executing chains of operations across multiple artifacts
  - multiple workspaces
    - agents can join and work in multiple workspaces, concurrently
    - including remote workspaces
  - RBAC security model
    - workspace artifact provides operations to set/change the access control policies of the workspace, depending on the agent role
    - ruling agents' access and use of artifacts of the workspace
  - ...
- See CArtAgO papers and manuals for more information

Applying CArtAgO and JaCa

- Using CArtAgO/JaCa for building real-world applications and infrastructures
- Some examples
  - JaCa-Android
    - implementing mobile computing applications on top of the Android platform using JaCa [Santi et al., 2011]
    - `http://jaca-android.sourceforge.net`
  - JaCa-WS / CArtAgO-WS
    - building SOA/Web Services applications using JaCa [Ricci et al., 2010a]
    - `http://cartagows.sourceforge.net`
  - JaCa-Web
    - implementing Web 2.0 applications using JaCa
    - `http://jaca-web.sourceforge.net`

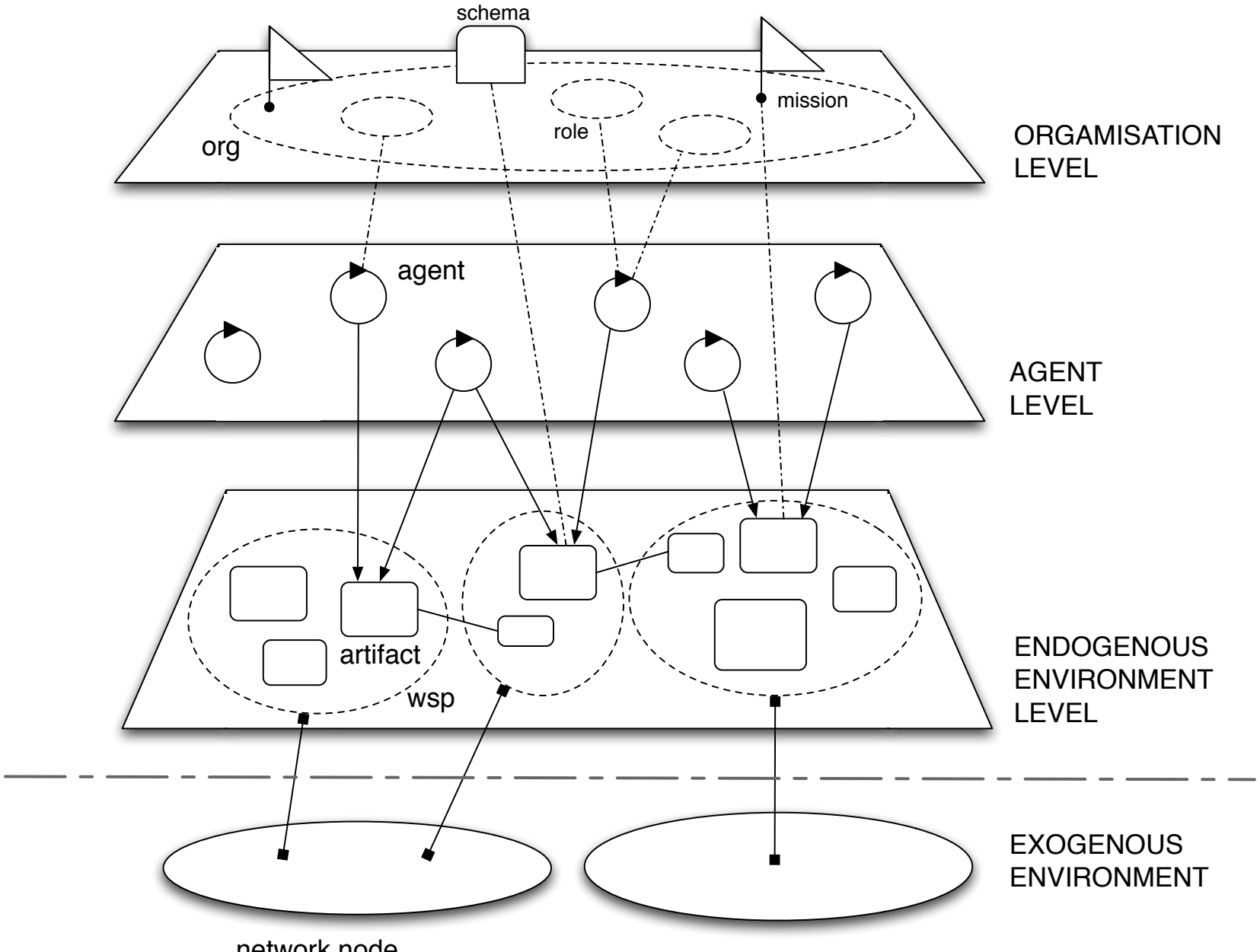
Summary

- Environment programming
  - environment as a programmable part of the MAS
  - encapsulating and modularising functionalities useful for agents' work
- Artifact-based environments
  - artifacts as first-class abstraction to design and program complex software environments
    - usage interface, observable properties / events, linkability
  - artifacts as first-order entities for agents
    - interaction based on use and observation
    - agents dynamically co-constructing, evolving, adapting their world
- CArtAgO computational framework
  - programming and executing artifact-based environments
  - integration with heterogeneous agent platforms
  - JaCa case

Organisation Oriented Programming

— OOP —

Abstractions in Multi-Agent Systems



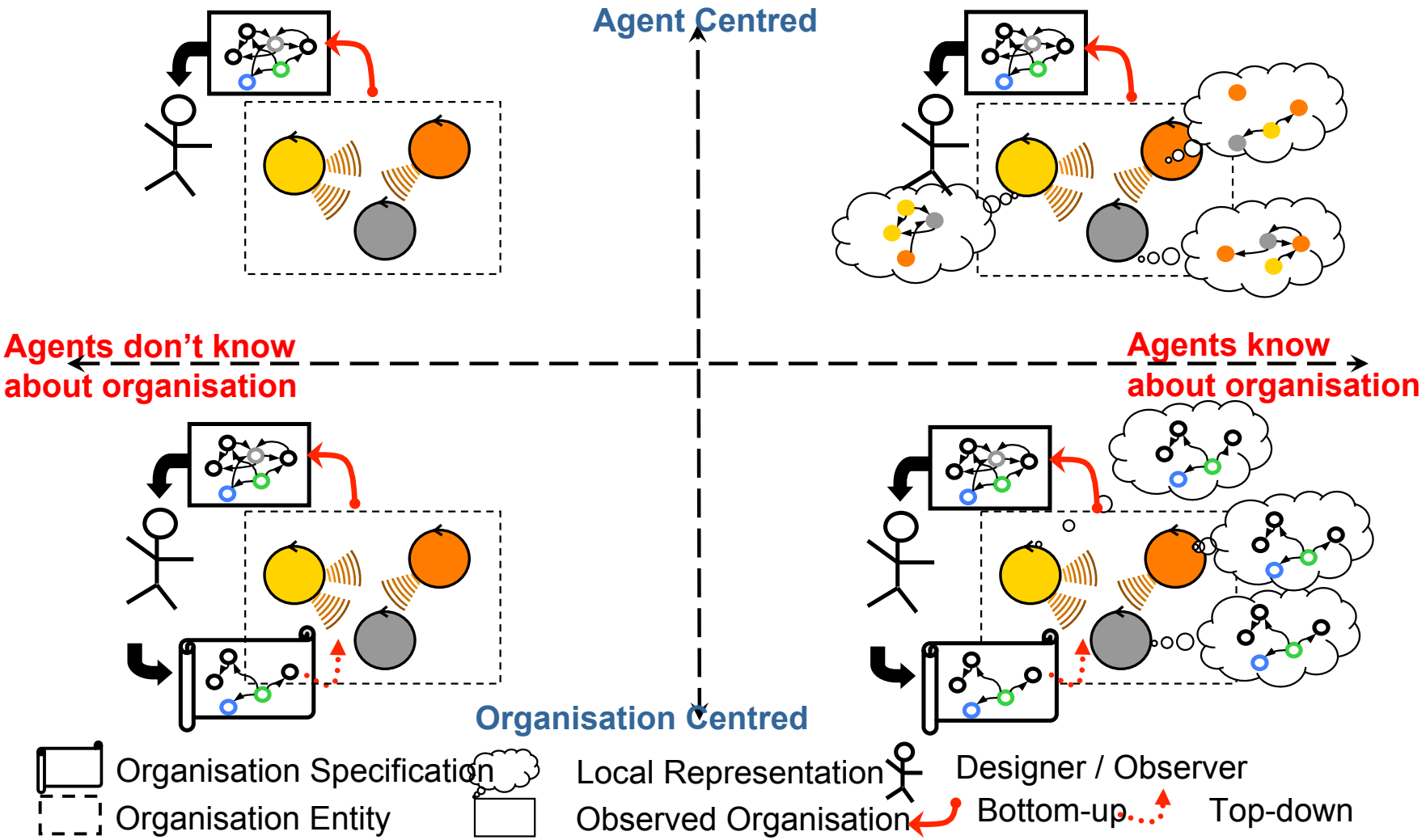
Organisation in MAS – a definition

- Pattern of agent cooperation
  - with a purpose
  - supra-agent
  - emergent or
  - predefined (by designer or agents)

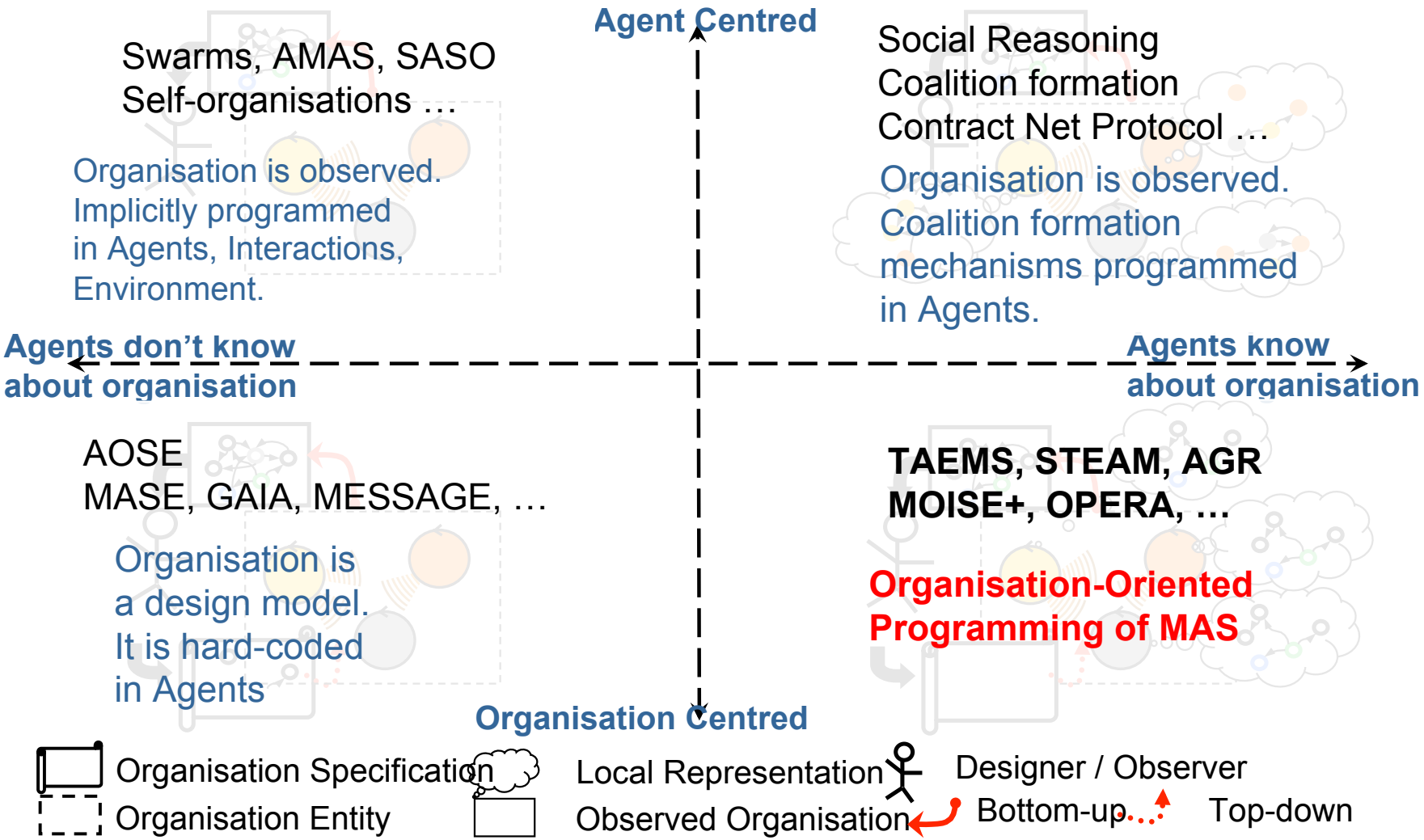
Introduction: Some definitions

- Organisations are structured, patterned systems of activity, knowledge, culture, memory, history, and capabilities that are distinct from any single agent [Gasser, 2001]
  $\rightsquigarrow$ 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]
  $\rightsquigarrow$ definition by the designer, or by actors, to achieve a **purpose**
- An organisation is characterised by: a division of tasks, a distribution of roles, authority systems, communication systems, contribution-retribution systems [Bernoux, 1985]
  $\rightsquigarrow$ **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]
  $\rightsquigarrow$ **pattern of emergent cooperation**

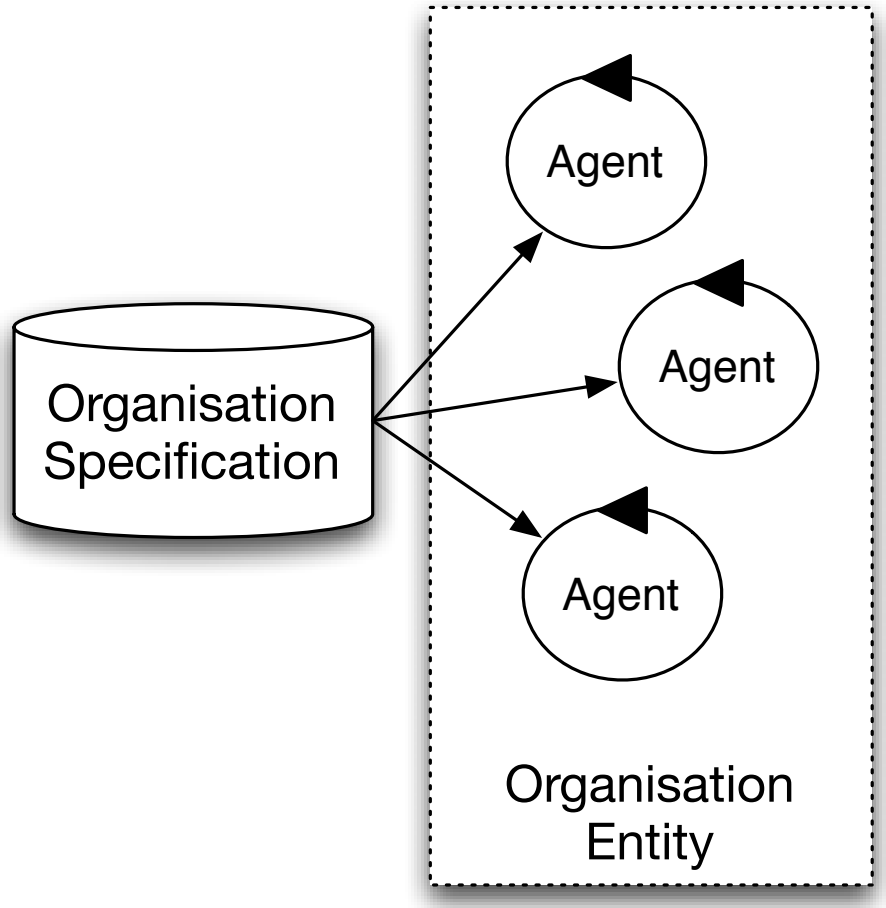
Perspective on organisations from EASSS'05 Tutorial (Sichman, Boissier)



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



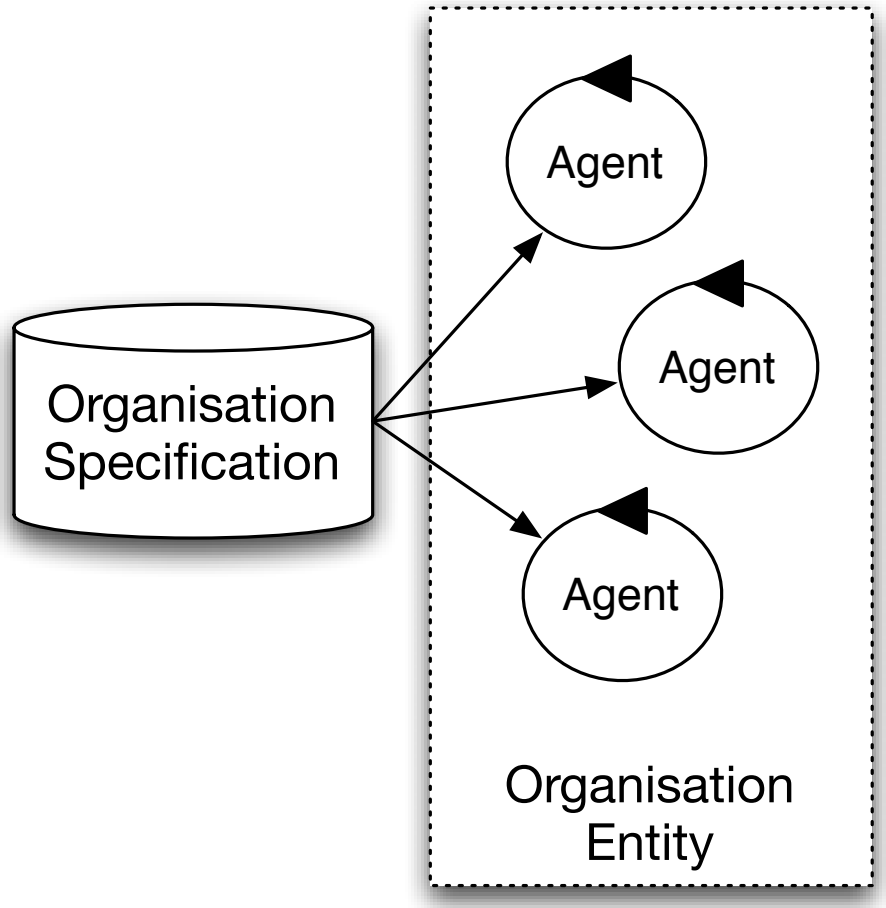
Organisation Oriented Programming (OOP)



- Programming outside the agents
- Using organisational concepts
- To define a cooperative pattern

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

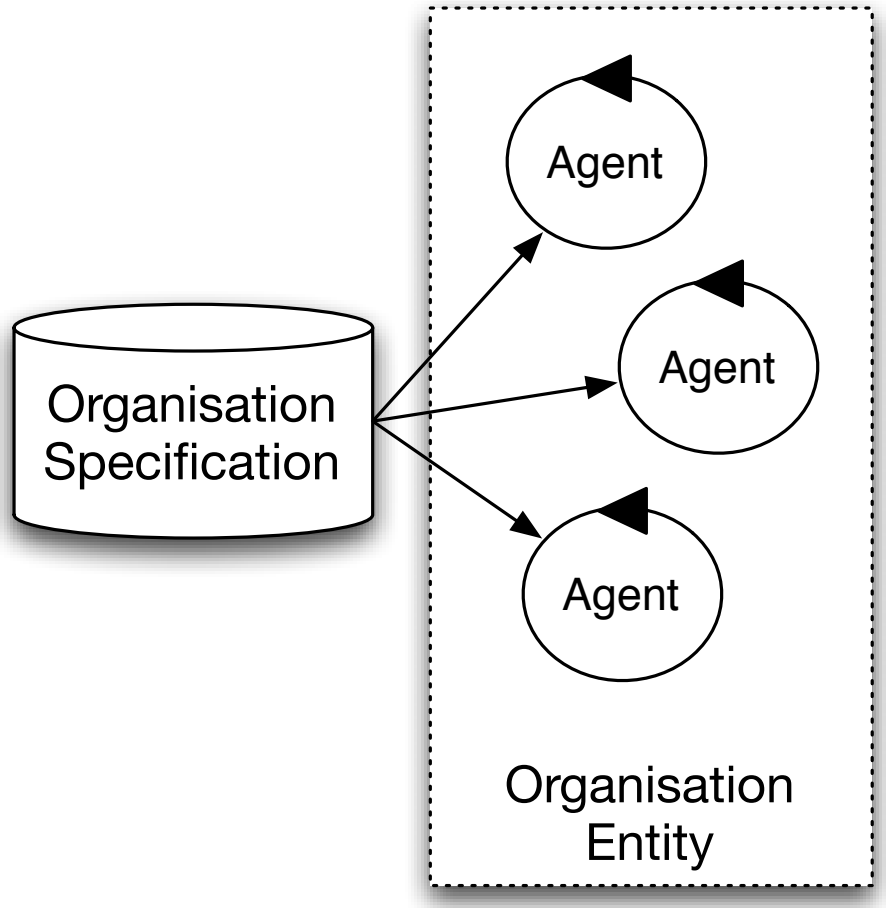
Organisation Oriented Programming (OOP)



First approach

- Agents read the program and follow it

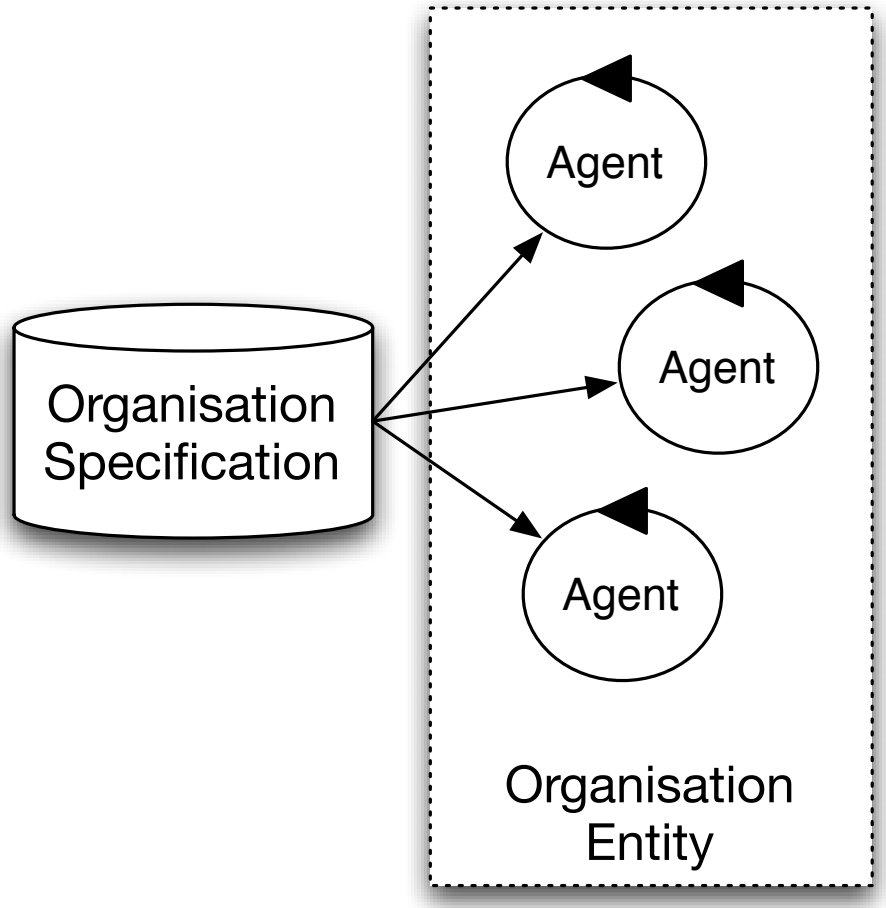
Organisation Oriented Programming (OOP)



Second approach

- Agents **are forced** to follow the program

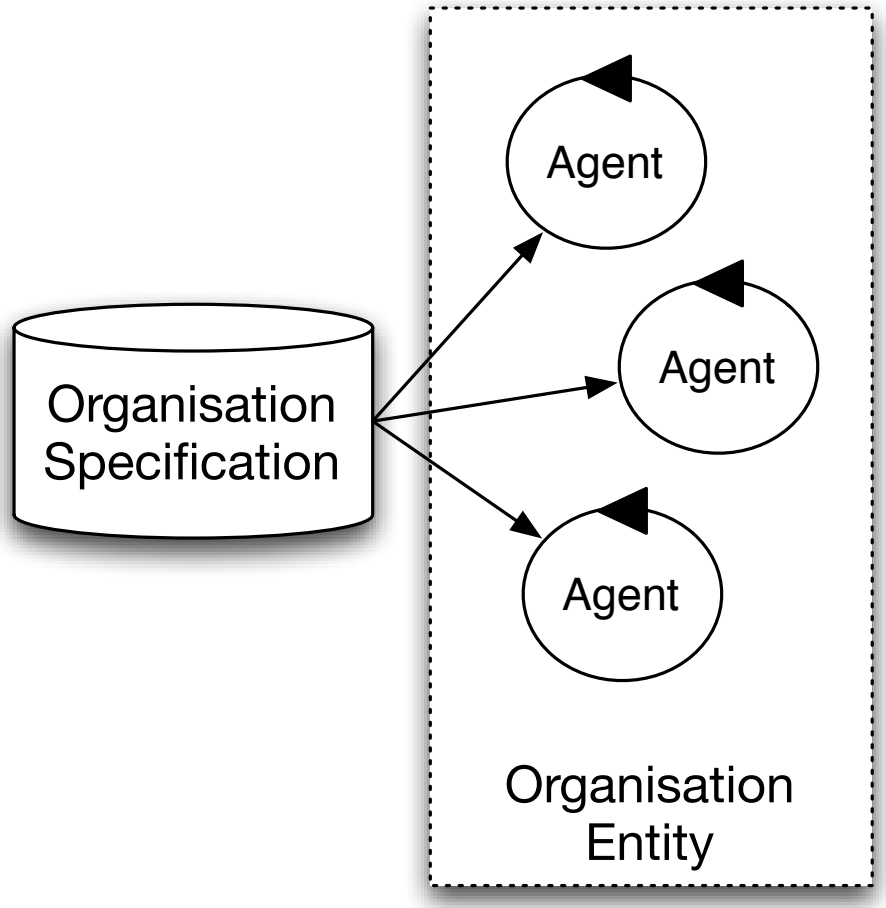
Organisation Oriented Programming (OOP)



Second approach

- Agents **are forced** to follow the program
- Agents **are rewarded** if they follow the program
- ...

Organisation Oriented Programming (OOP)



Components

- Programming language (OML)
- Platform (OMI)
- Integration to agent architectures and environment

Components of OOP: Organisation Modelling Language (OML)

- Declarative specification of the organisation(s)
- Specific constraints, norms and cooperation patterns imposed on the agents
- Based on an organisational **model**

  e.g. AGR [Ferber and Gutknecht, 1998],
  TeamCore [Tambe, 1997],
  Islander [Esteva et al., 2001],
  $\mathcal{M}$oise$^+$ [Hübner et al., 2002],
  Opera [Dignum and Aldewereld, 2010],
  2OPL [Dastani 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 [Piunti et al., 2009b], [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 and decide
  - Openness, scalability, dynamism
- 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 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 rules for that very action

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

e.g. when a soccer team wants to play 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 an 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

Some OOP approaches

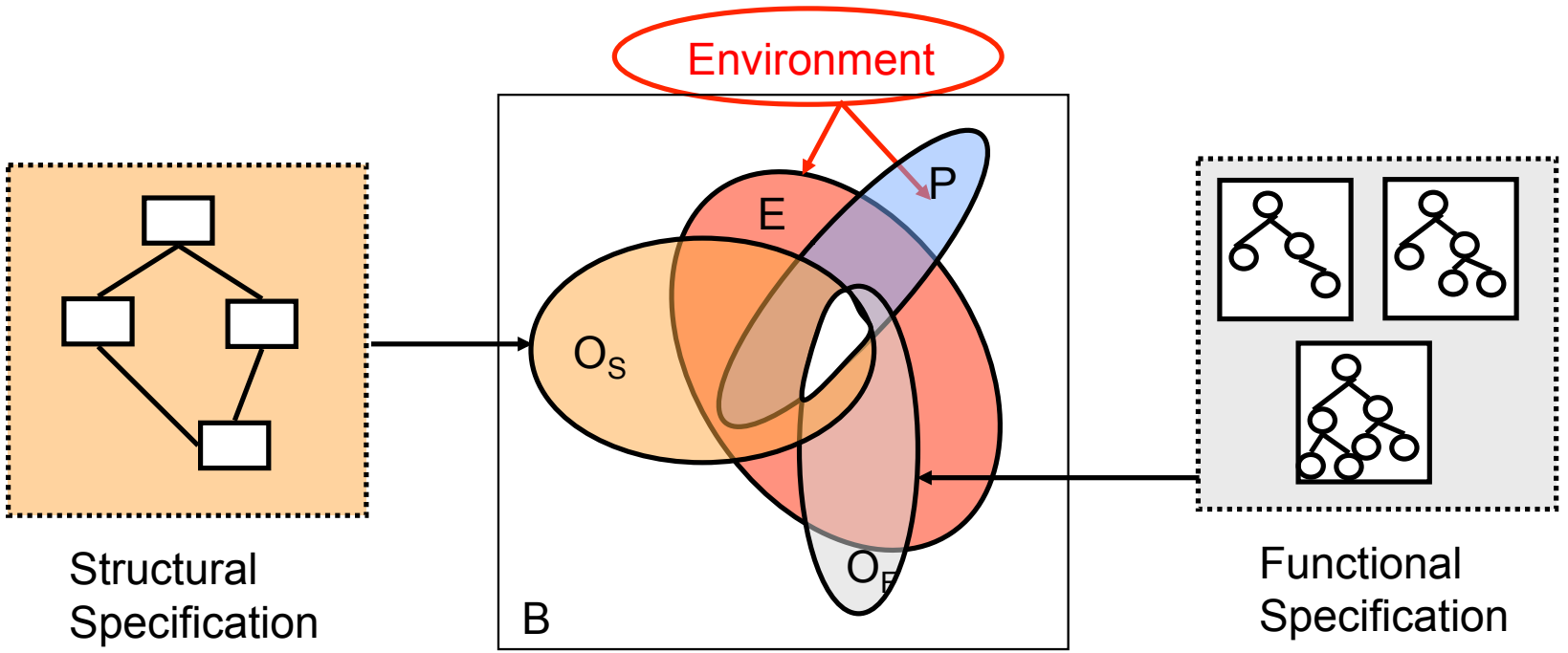
- AGR/Madkit [Ferber and Gutknecht, 1998]
- STEAM/Teamcore [Tambe, 1997]
- ISLANDER/AMELI [Esteva et al., 2004]
- Opera/Operetta [Dignum and Aldewereld, 2010]
- PopOrg [Rocha Costa and Dimuro, 2009]
- 2OPL [Dastani et al., 2009]
- ...

The $\mathcal{M}$oise Framework

$\mathcal{M}$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., 2009b])
  - Environment and Organisation ([Piunti et al., 2009a])
  - Agents and Organisation ($\mathcal{J}$-$\mathcal{M}$oise$^+$ [Hübner et al., 2007])

Moise Modelling Dimensions

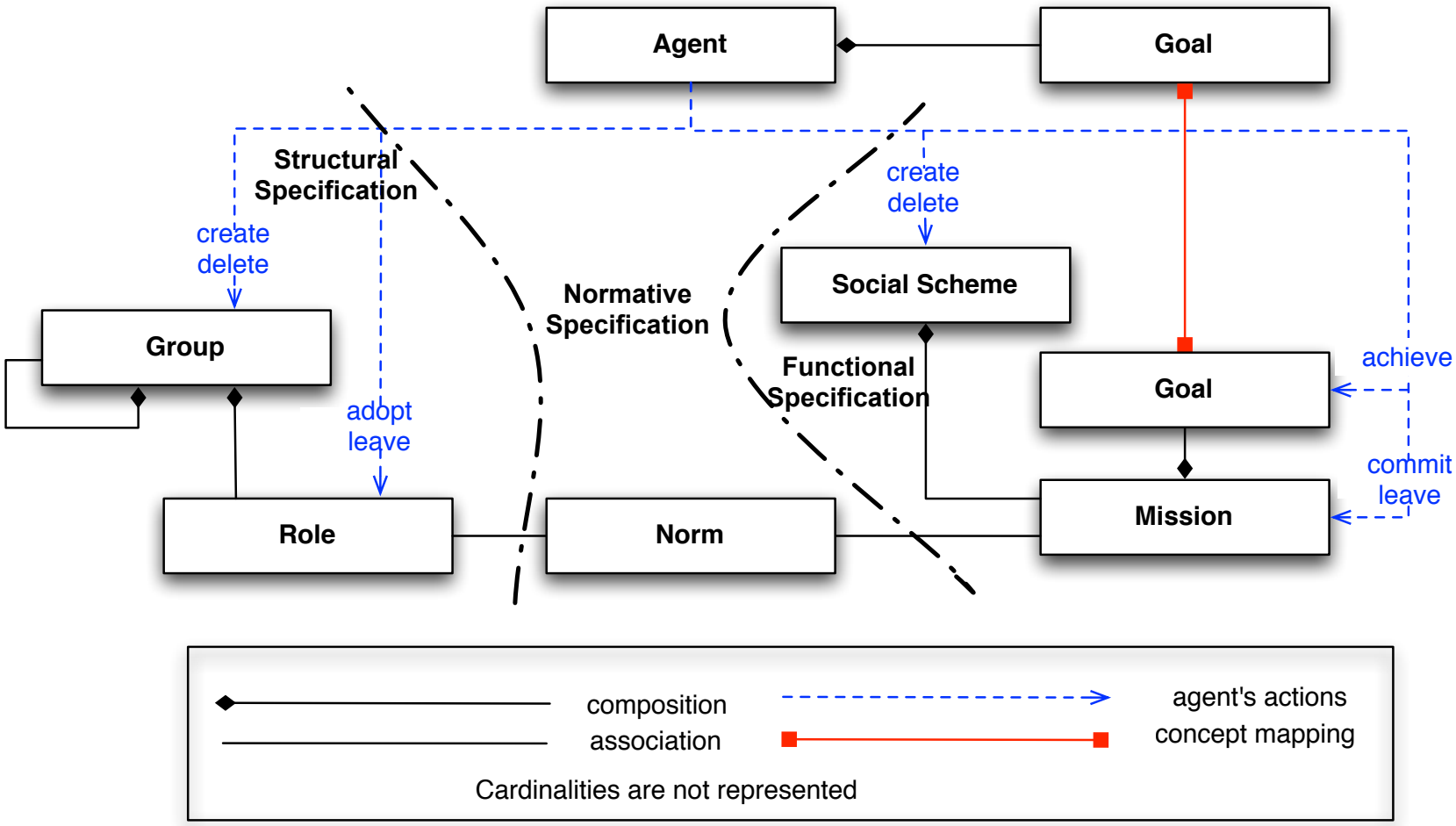


Groups, links, roles
Compatibilities, multiplicities
inheritance

Normative Specification
Permissions, Obligations
Allows agents autonomy!

Global goals, plans,
Missions, schemas,
preferences

Moise OML meta-model (partial view)



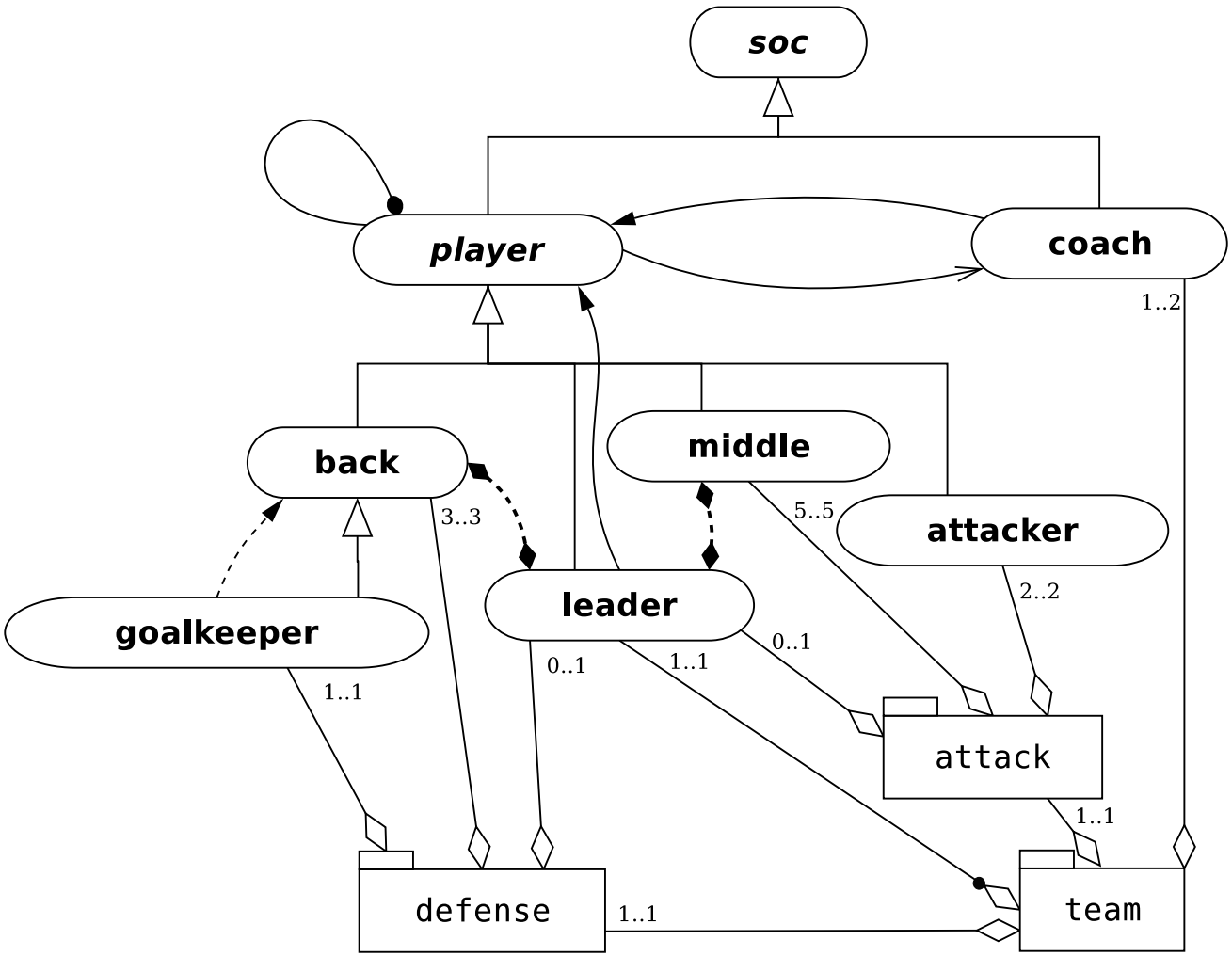
Moise OML

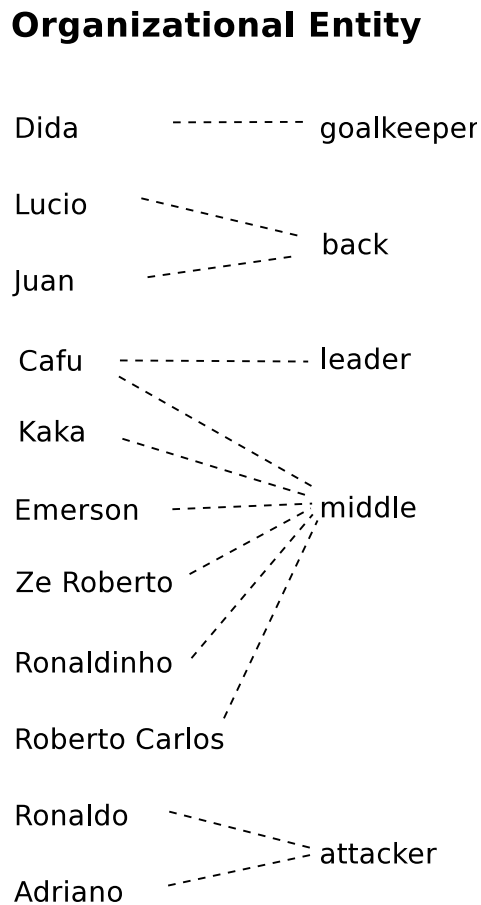
- OML for defining organisation specification **and** organisation entity
- Three independent dimensions [Hübner et al., 2007] ($\rightsquigarrow$ well adapted for the reorganisation concerns):
  - **Structural**: Roles, Groups
  - **Functional**: Goals, Missions, Schemes
  - **Normative**: Norms (obligations, permissions, interdictions)
- Abstract description of the organisation for
  - the designers
  - the agents
    - $\rightsquigarrow$ $\mathcal{J}$-Moise$^+$ [Hübner et al., 2007]
  - the Organisation Management Infrastructure
    - $\rightsquigarrow$ ORA4MAS [Hübner et al., 2009]

Moise OML 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



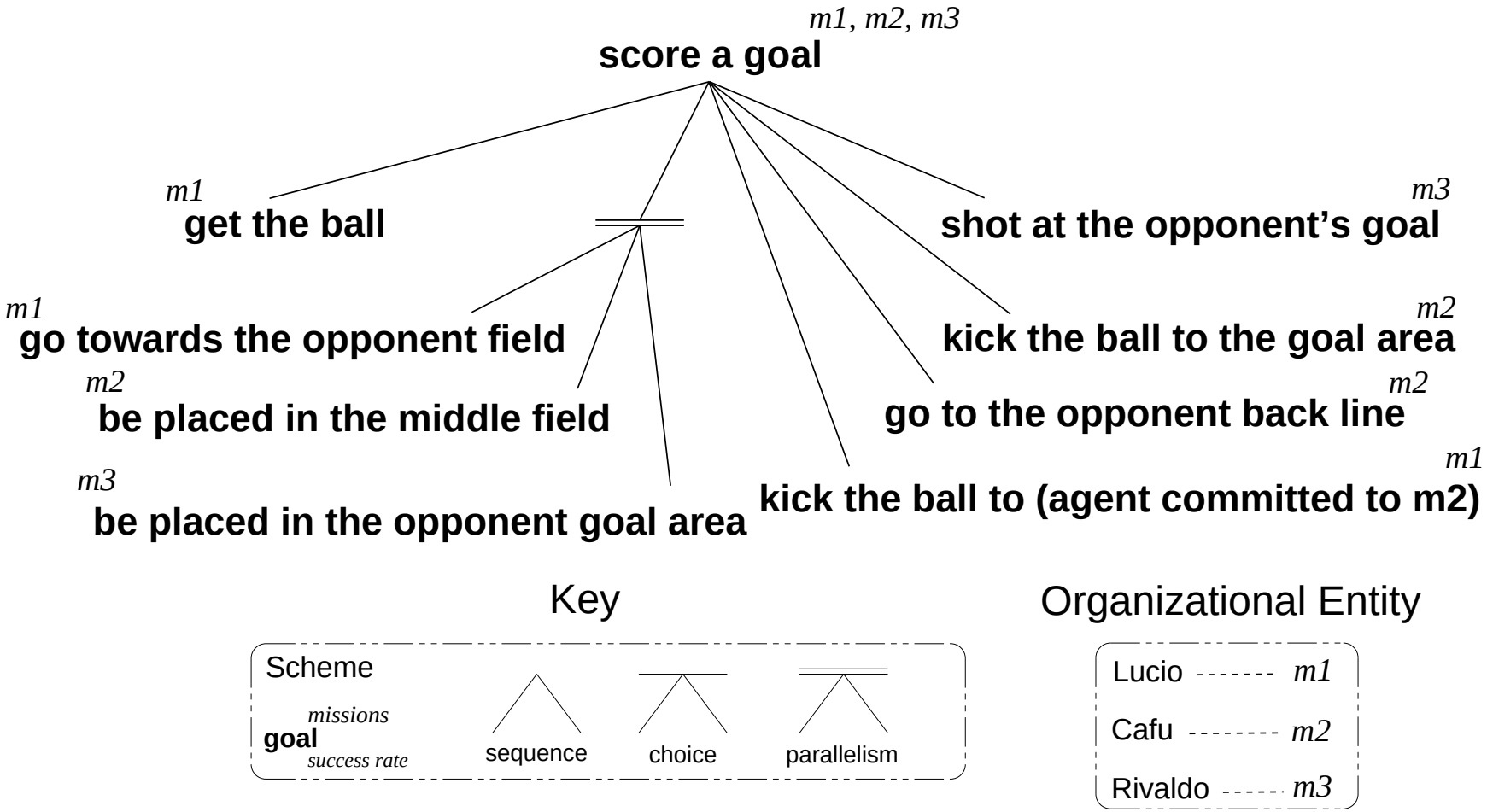


Graphical representation of structural specification of 3-5-2 Joj Team

Moise OML Functional Specification

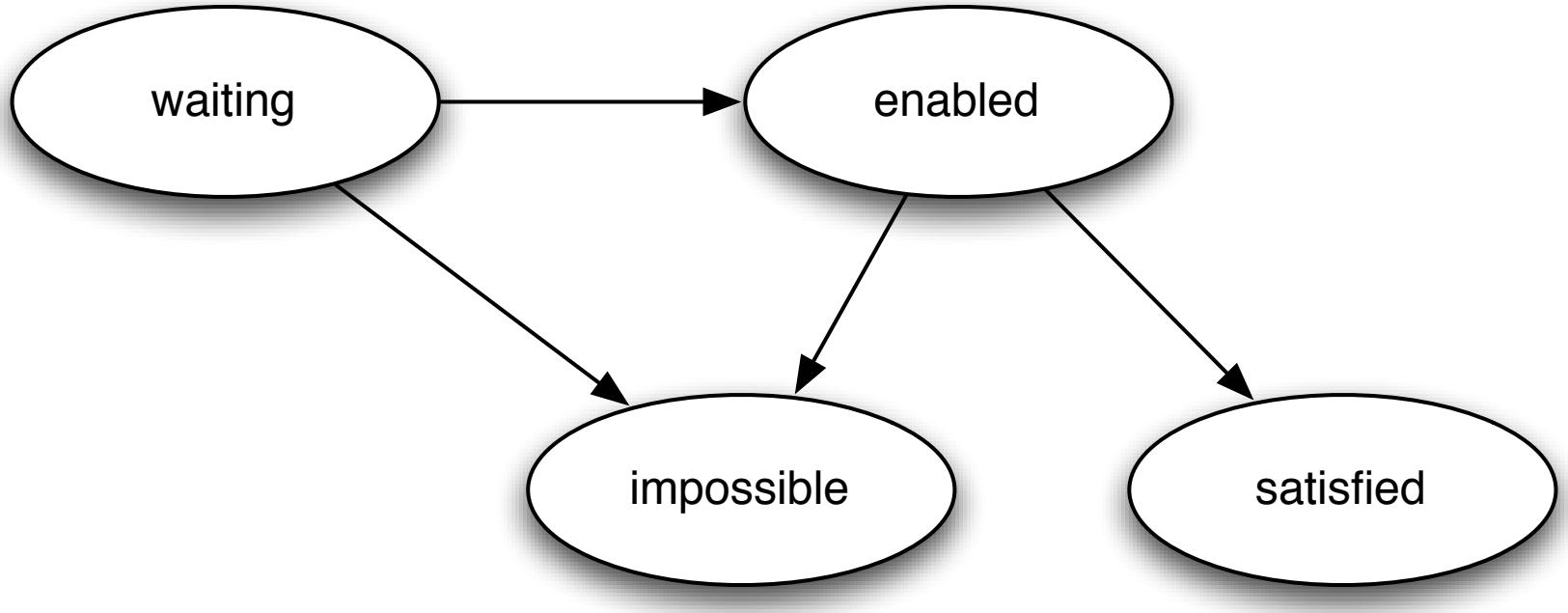
- Specifies the expected behaviour of an MAS in terms of **goals** along two levels:
  - **Collective** with **Scheme**
  - **Individual** with **Mission**
- Components:
  - **Goals**:
    - **Achievement goal** (default type). Goals of this type should be declared as satisfied by the agents committed 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



Graphical representation of social scheme "side_attack" for joj team

Goal States



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

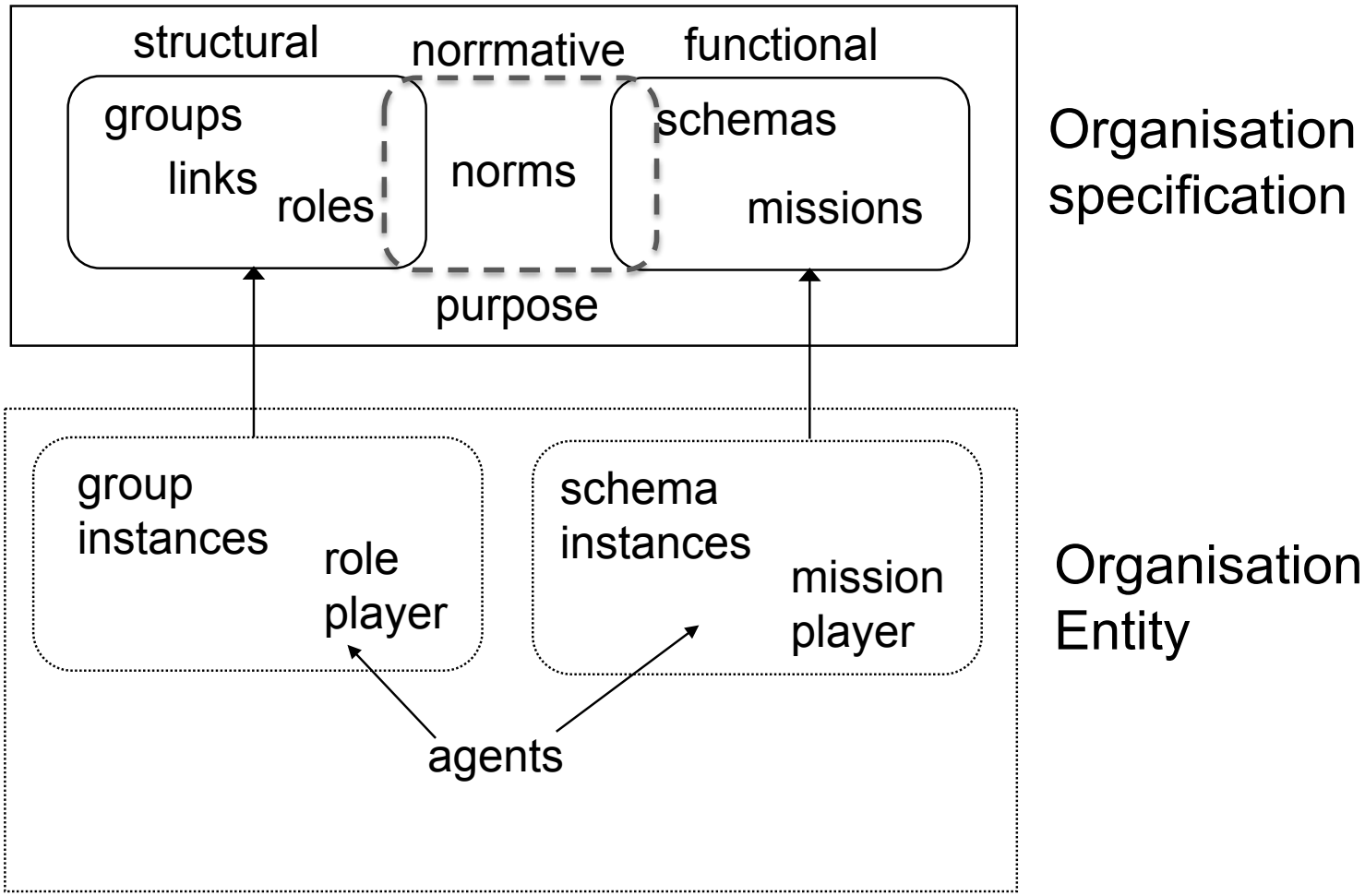
Moise OML Normative Specification

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

Norm Specification – example

role	**deontic**	**mission**		TTF
back	*obliged*	*m*1	get the ball, go ...	1 minute
left	*obliged*	*m*2	be placed at ..., kick ...	3 minute
right	*obliged*	*m*2		1 day
attacker	*obliged*	*m*3	kick to the goal, ...	30 seconds

Organisational Entity



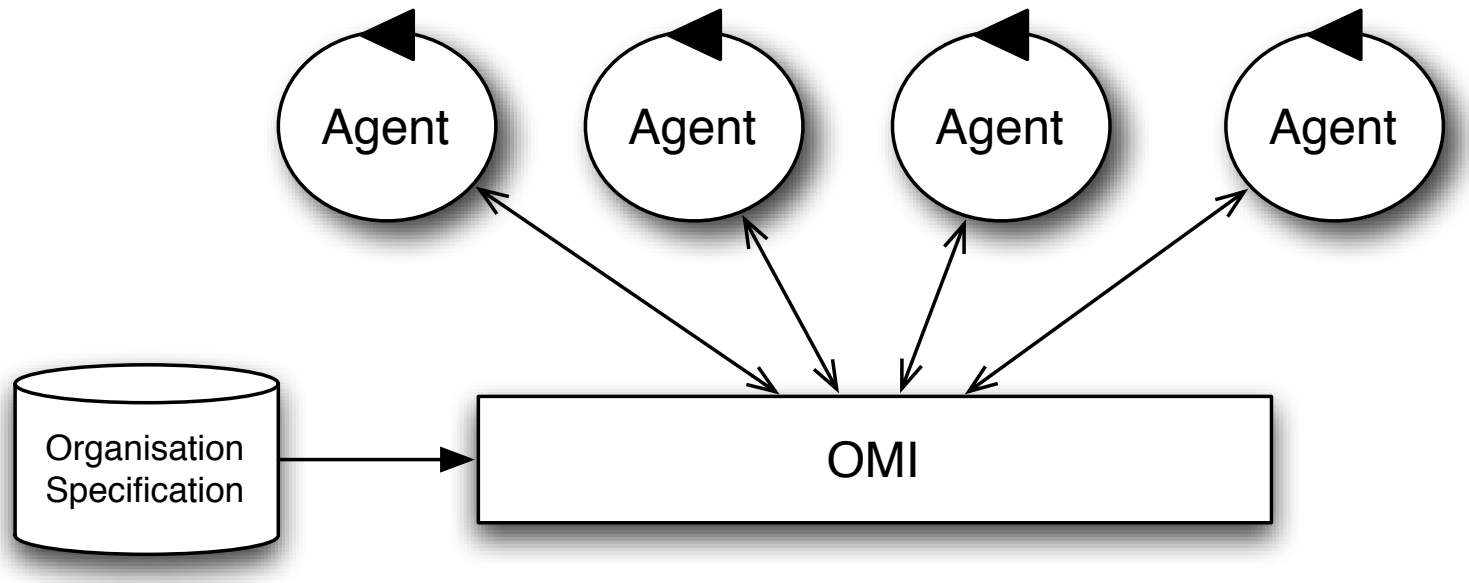
Organisation Entity Dynamics

1. Organisation is created (by the agents)
   - instances of groups
   - instances of schemes
2. Agents enter into groups **adopting** roles
3. Groups 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

Organisation management infrastructure (OMI)

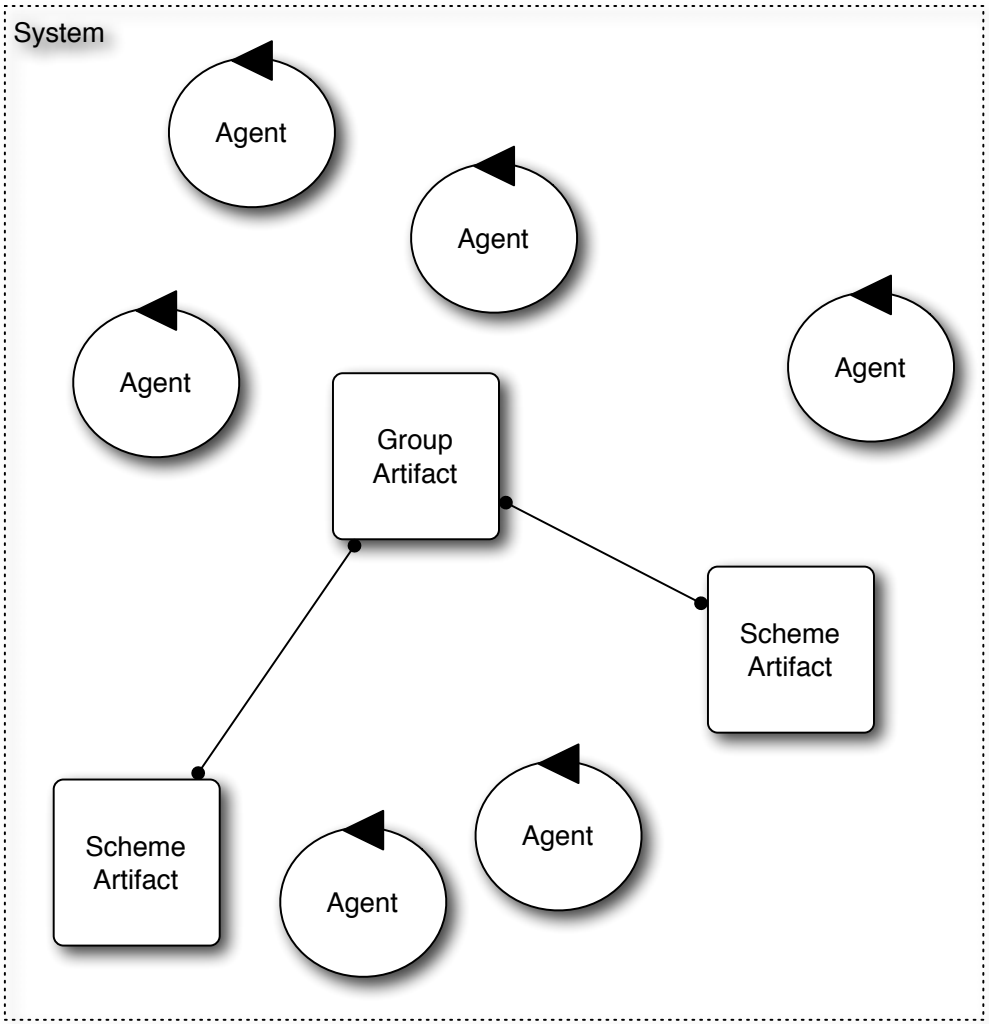
Responsibility

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



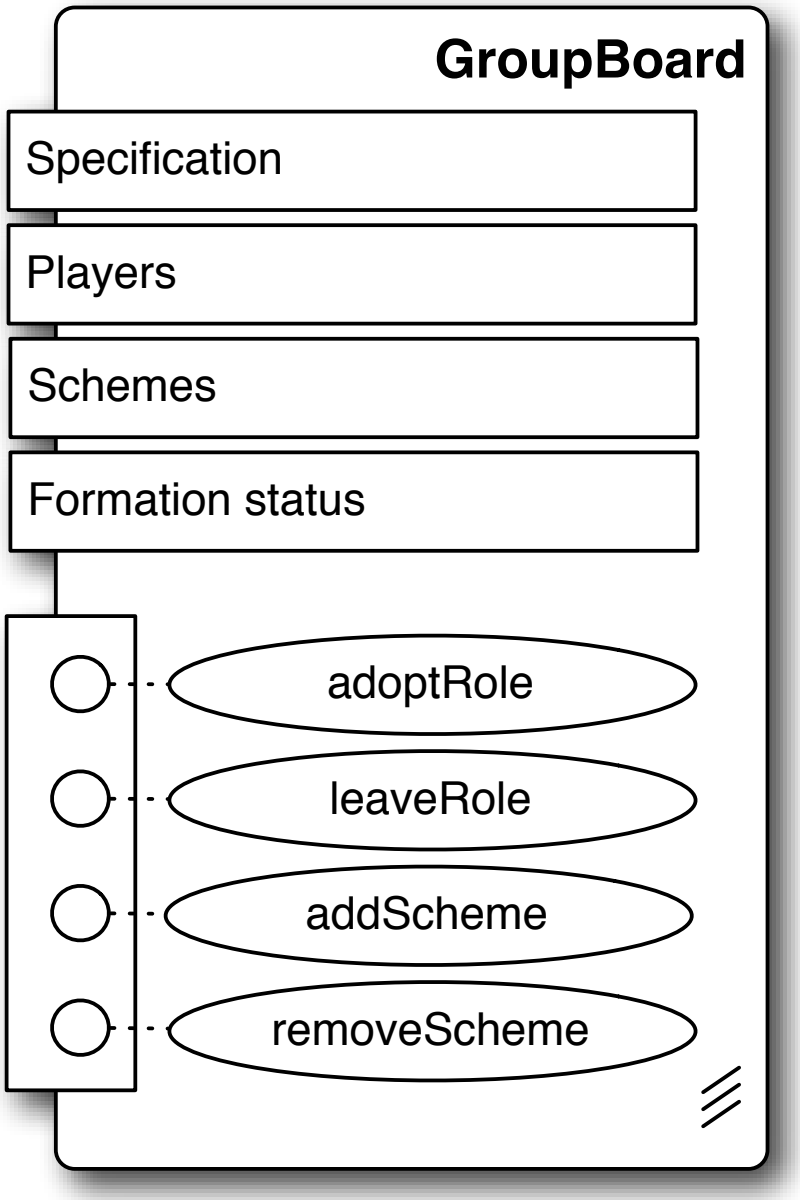
(e.g. MadKit, AMELI, $\mathcal{S}$-$\mathcal{M}$oise$^+$, ...)

Organisational artifacts in ORA4MAS



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

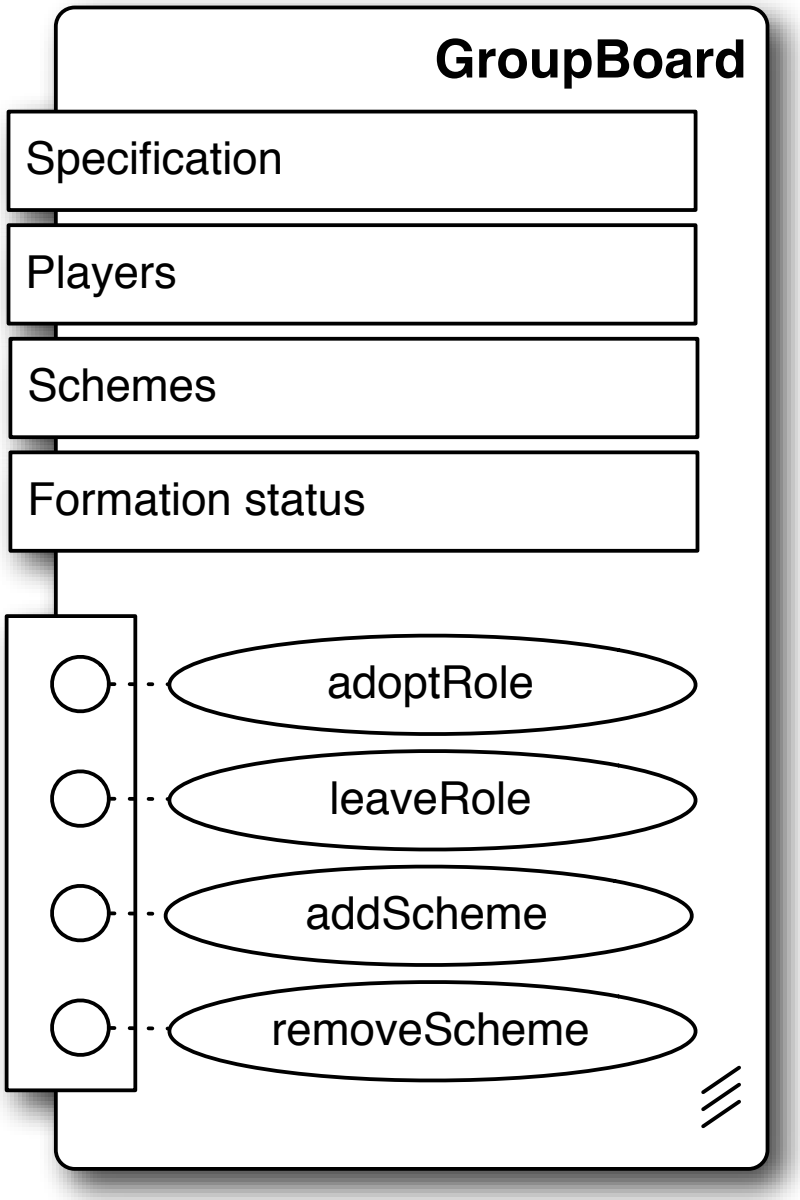
ORA4MAS – GroupBoard artifact



Observable Properties:

- specification: the specification of the group in the OS (an object of class moise.os.ss.Group)
- 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

ORA4MAS – 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)

ORA4MAS – 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

ORA4MAS – 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

Environment integration: 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, ...)

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**

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 perception

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

- specification
- 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)
- ....

Organisational perception – example

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

Beliefs

commitment(bob,mManager,"sch2")[artifact_id(cobj_4),c
cept),artifact_name(cobj_4,"sch2"),artifact_type(cobj_4,"ora4m
commitment(bob,mManager,"sch1")[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)[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)[artifact_id(cot

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)
  <- leaveMission(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!").
```

Summary – Moise

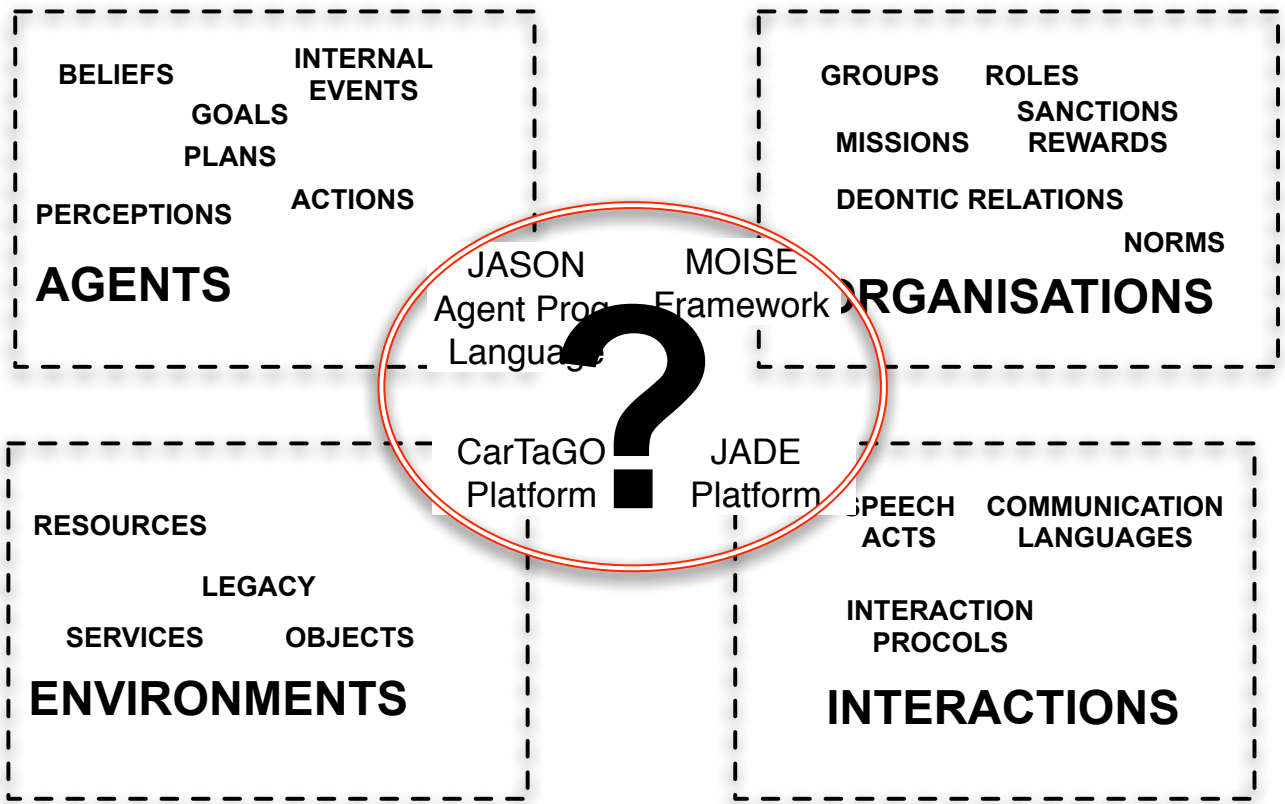
- Ensures that the agents follow some of the constraints specified for the organisation
- Helps the agents to work together
- The organisation is **interpreted at runtime**, it is not hardwired in the agents code
- The agents 'handle' the organisation (i.e. their artifacts)
- It is suitable for open systems as no specific agent architecture is required

- All available as open source at

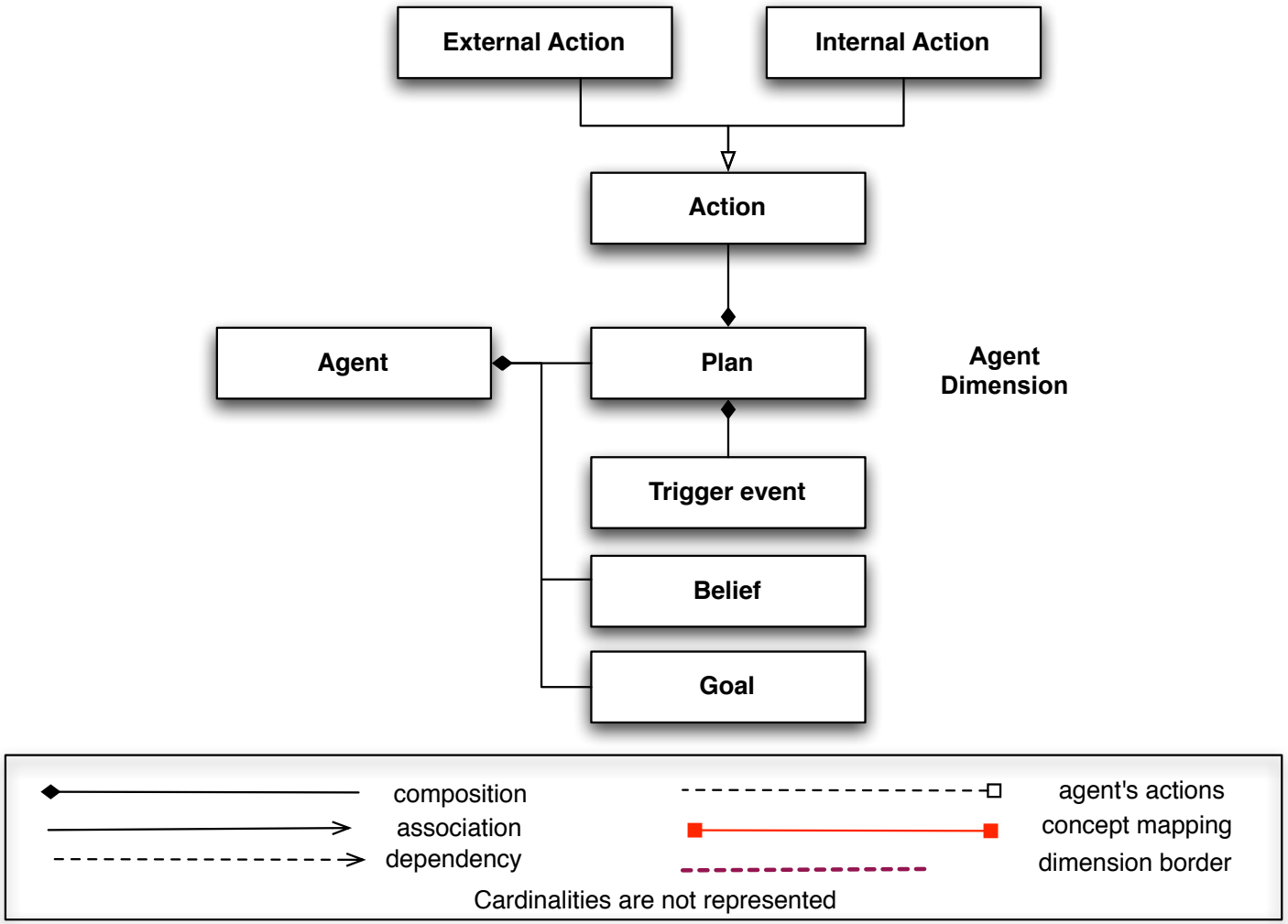
`http://moise.souceforge.net`

Conclusions

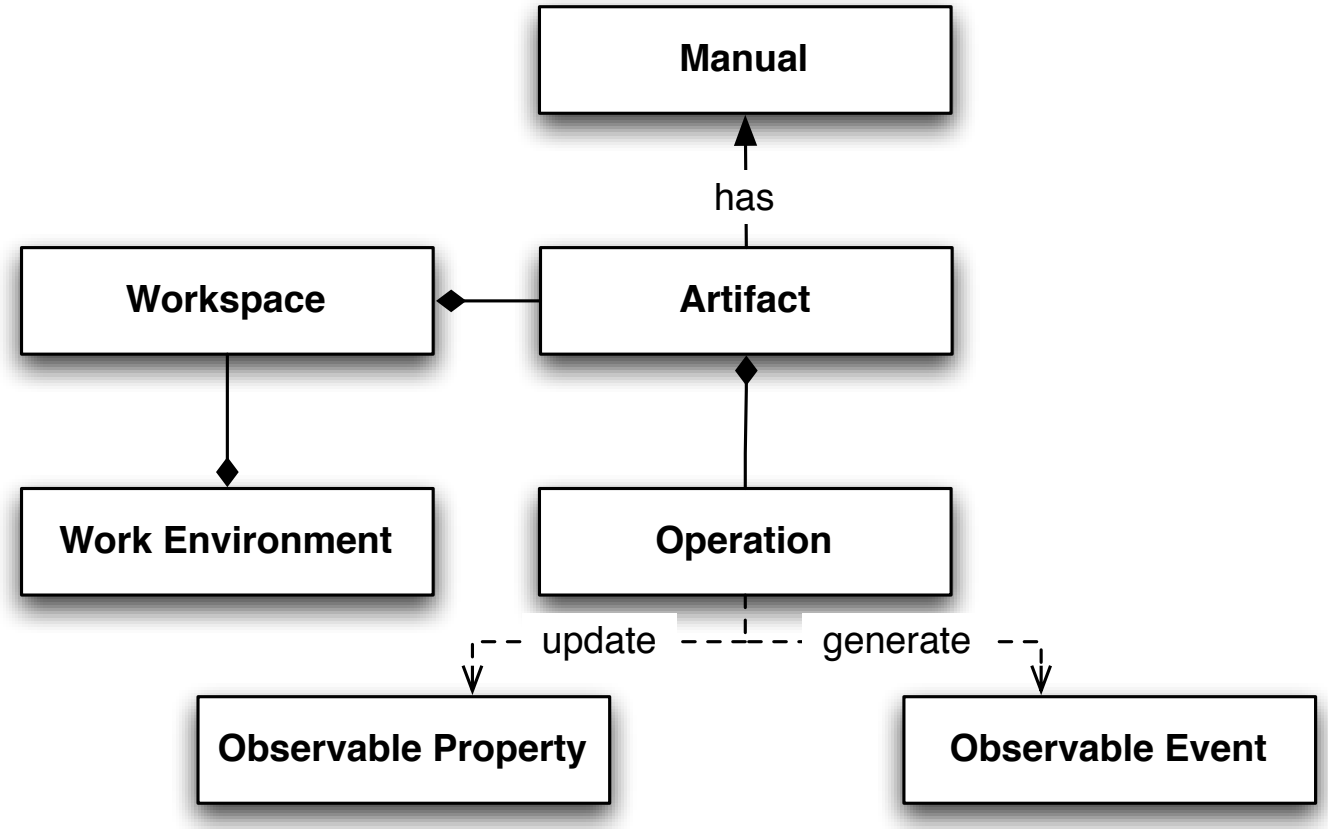
Putting the Pieces Together

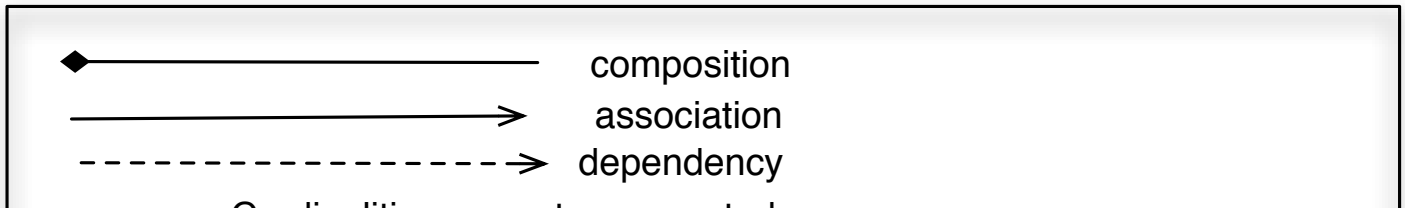


Agent meta-model

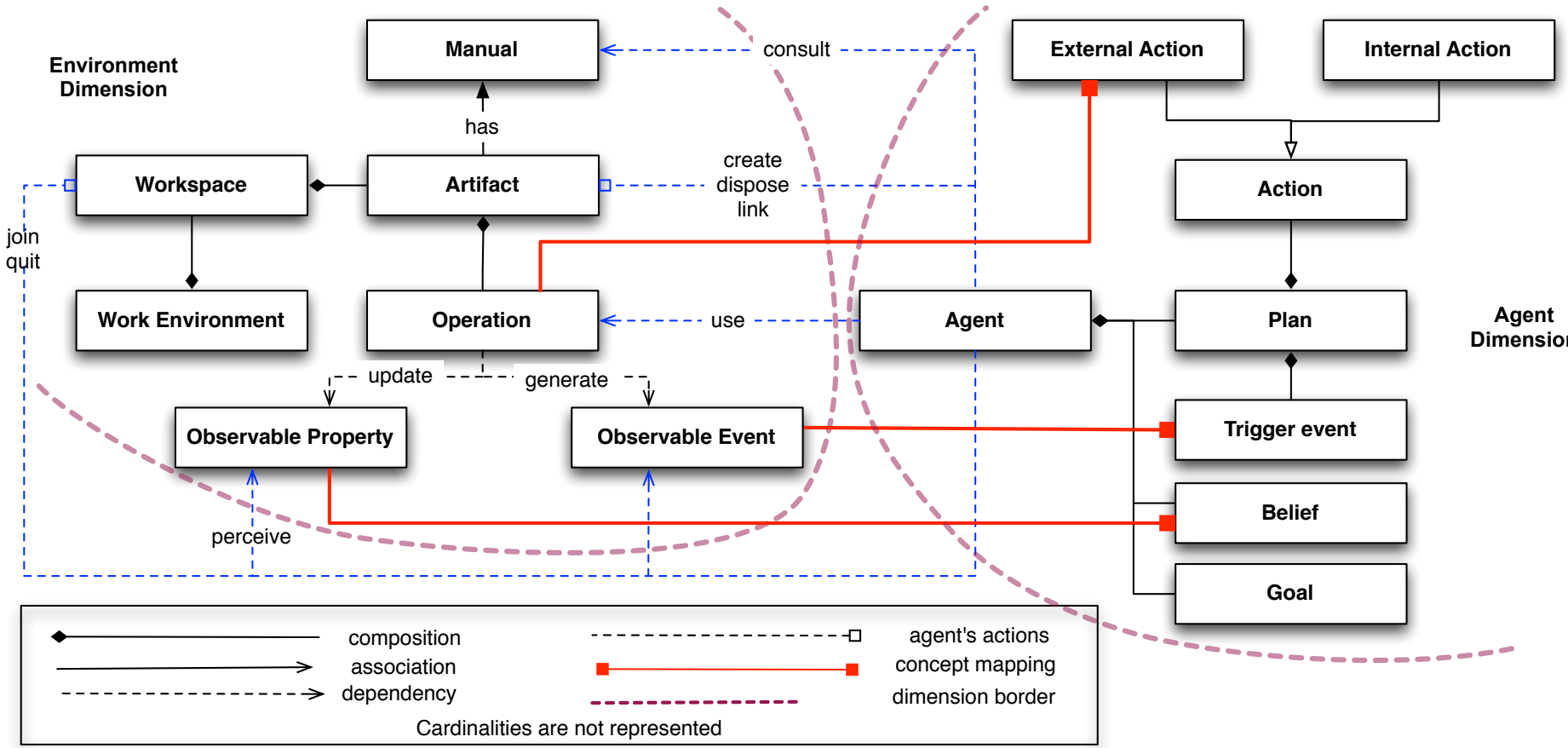


Environment meta-model

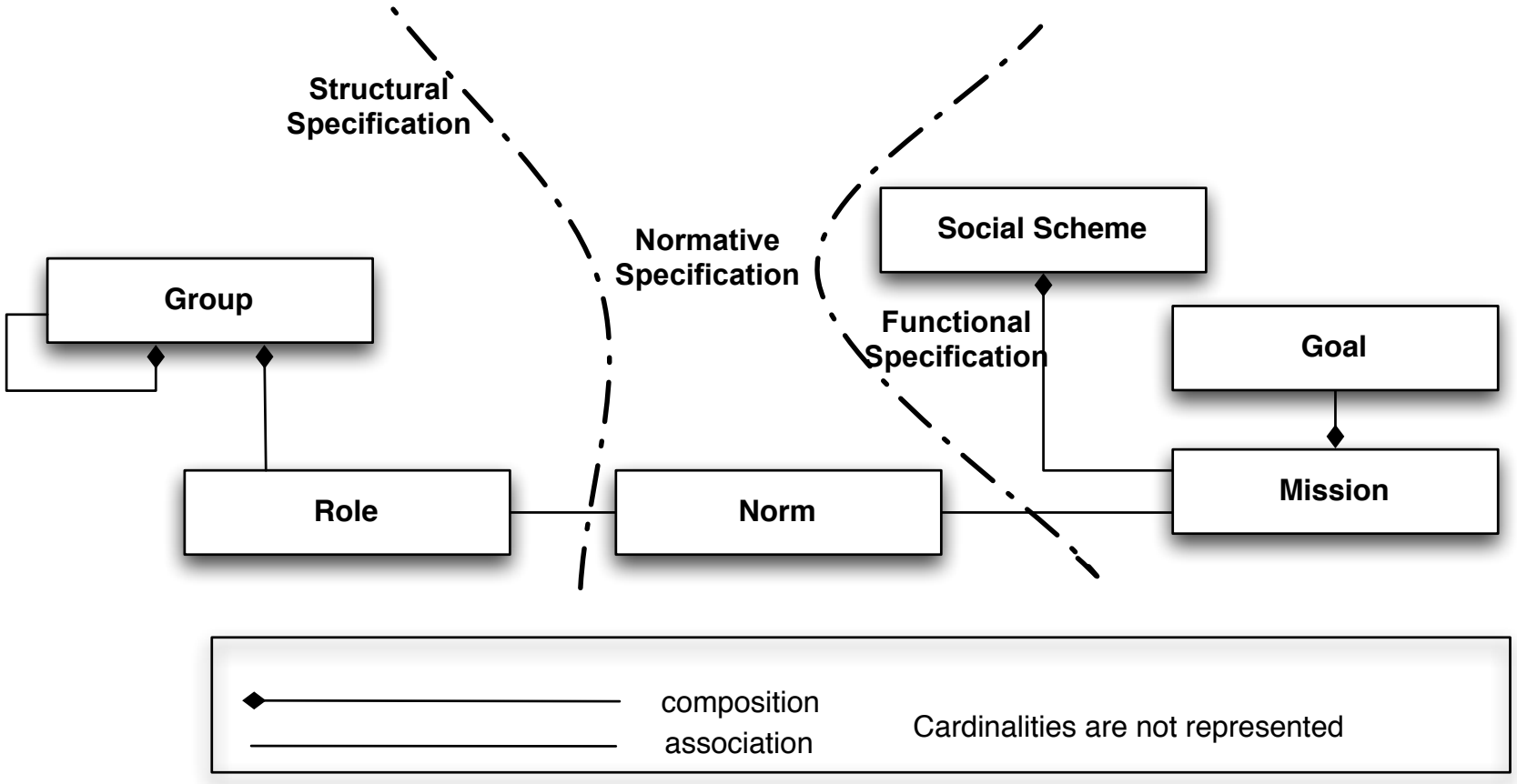




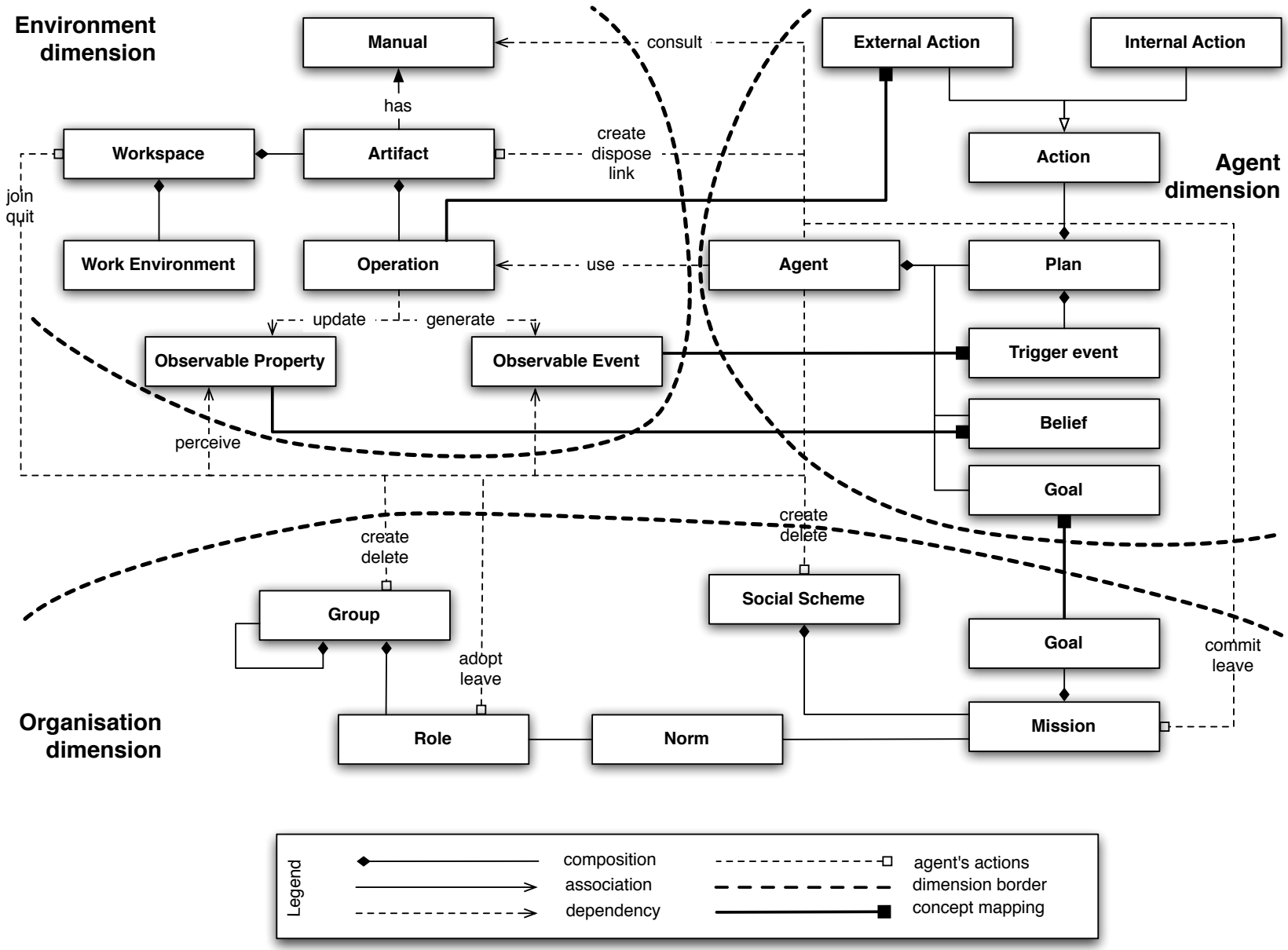
A & E Interaction meta-model



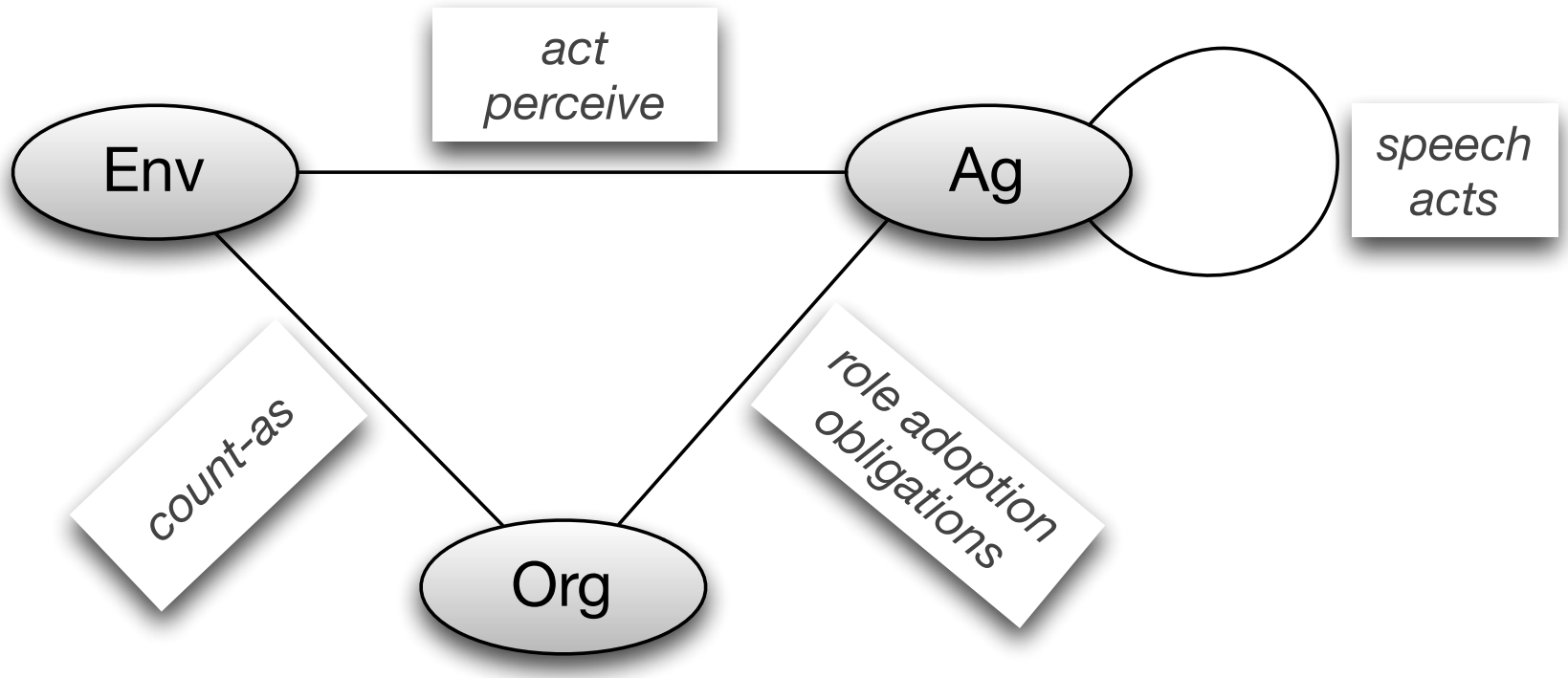
Organisation meta-model



JaCaMo Meta-Model



JaCaMo binding concepts



Research on Multi-Agent Systems...

—

Whatever you do in MAS, make it available in a **programming language/platform** for MAS!!!

—

Bibliography I


Baldoni, M., Bentahar, J., van Riemsdijk, M. B., and Lloyd, J., editors (2010). *Declarative Agent Languages and Technologies VII, 7th International Workshop, DALT 2009, Budapest, Hungary, May 11, 2009. Revised Selected and Invited Papers*, volume 5948 of *Lecture Notes in Computer Science*. Springer.


Baldoni, M. and Endriss, U., editors (2006). *Declarative Agent Languages and Technologies IV, 4th International Workshop, DALT 2006, Hakodate, Japan, May 8, 2006, Selected, Revised and Invited Papers*, volume 4327 of *Lecture Notes in Computer Science*. Springer.


Baldoni, M., Endriss, U., Omicini, A., and Torroni, P., editors (2006). *Declarative Agent Languages and Technologies III, Third International Workshop, DALT 2005, Utrecht, The Netherlands, July 25, 2005, Selected and Revised Papers*, volume 3904 of *Lecture Notes in Computer Science*. Springer.


Baldoni, M., Son, T. C., van Riemsdijk, M. B., and Winikoff, M., editors (2008). *Declarative Agent Languages and Technologies V, 5th International Workshop, DALT 2007, Honolulu, HI, USA, May 14, 2007, Revised Selected and Invited Papers*, volume 4897 of *Lecture Notes in Computer Science*. Springer.

Bibliography II


Baldoni, M., Son, T. C., van Riemsdijk, M. B., and Winikoff, M., editors (2009).
Declarative Agent Languages and Technologies VI, 6th International Workshop, DALT 2008, Estoril, Portugal, May 12, 2008, Revised Selected and Invited Papers, volume 5397 of *Lecture Notes in Computer Science*. Springer.


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


Bordini, R. H., Braubach, L., Dastani, M., Fallah-Seghrouchni, A. E., Gómez-Sanz, J. J., Leite, J., O'Hare, G. M. P., Pokahr, A., and Ricci, A. (2006a).
A survey of programming languages and platforms for multi-agent systems.
Informatica (Slovenia), 30(1):33–44.


Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E., editors (2005a).
Multi-Agent Programming: Languages, Platforms and Applications, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*.
Springer.

Bibliography III


Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E., editors (2005b).
Programming Multi-Agent Systems, Second International Workshop ProMAS 2004, New York, NY, USA, July 20, 2004 Selected Revised and Invited Papers, volume 3346 of *Lecture Notes in Computer Science*. Springer.


Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E., editors (2006b).
Programming Multi-Agent Systems, Third International Workshop, ProMAS 2005, Utrecht, The Netherlands, July 26, 2005, Revised and Invited Papers, volume 3862 of *Lecture Notes in Computer Science*. Springer.


Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E., editors (2007a).
Programming Multi-Agent Systems, 4th International Workshop, ProMAS 2006, Hakodate, Japan, May 9, 2006, Revised and Invited Papers, volume 4411 of *Lecture Notes in Computer Science*. Springer.


Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E., editors (2009).
Multi-Agent Programming: Languages, Tools and Applications.
Springer.

Bibliography IV


Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007b).
Programming Multi-Agent Systems in AgentSpeak Using Jason.
Wiley Series in Agent Technology. John Wiley & Sons.


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.


Dastani, M. (2008).
2apl: a practical agent programming language.
Autonomous Agents and Multi-Agent Systems, 16(3):214–248.


Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E., editors (2004).
Programming Multi-Agent Systems, First International Workshop, PROMAS 2003, Melbourne, Australia, July 15, 2003, Selected Revised and Invited Papers, volume 3067 of *Lecture Notes in Computer Science*. Springer.

Bibliography V


Dastani, M., Fallah-Seghrouchni, A. E., Leite, J., and Torroni, P., editors (2008a).
Languages, Methodologies and Development Tools for Multi-Agent Systems, First International Workshop, LADS 2007, Durham, UK, September 4-6, 2007. Revised Selected Papers, volume 5118 of *Lecture Notes in Computer Science*. Springer.


Dastani, M., Fallah-Seghrouchni, A. E., Leite, J., and Torroni, P., editors (2010).
Languages, Methodologies, and Development Tools for Multi-Agent Systems, Second International Workshop, LADS 2009, Torino, Italy, September 7-9, 2009, Revised Selected Papers, volume 6039 of *Lecture Notes in Computer Science*. Springer.


Dastani, M., Fallah-Seghrouchni, A. E., Ricci, A., and Winikoff, M., editors (2008b).
Programming Multi-Agent Systems, 5th International Workshop, ProMAS 2007, Honolulu, HI, USA, May 15, 2007, Revised and Invited Papers, volume 4908 of *Lecture Notes in Computer Science*. Springer.

Bibliography VI


Dastani, M., Grossi, D., Meyer, J.-J., and Tinnemeier, N. (2009).
Normative multi-agent programs and their logics.
In Meyer, J.-J. and Broersen, J., editors, *Knowledge Representation for Agents and Multi-Agent Systems*, volume 5605 of *Lecture Notes in Computer Science*, pages 16–31. Springer Berlin / Heidelberg.


Dignum, V. and Aldewereld, H. (2010).
Operetta: Organization-oriented development environment.
In *Proceedings of LADS @ MALLOW 2010*, pages 14–20.


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 L., J. (2004).
AMELI: An agent-based middleware for electronic institutions.
In Jennings, N. R., Sierra, C., Sonenberg, L., and Tambe, M., editors, *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'04)*, pages 236–243, New York, USA. ACM.

Bibliography VII


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.


Fisher, M. (2005).
Metatem: The story so far.
In [Bordini et al., 2006b], pages 3–22.


Fisher, M., Bordini, R. H., Hirsch, B., and Torroni, P. (2007).
Computational logics and agents: A road map of current technologies and future trends.
Computational Intelligence, 23(1):61–91.


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.

Bibliography VIII


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.


Giacomo, G. D., Lespérance, Y., and Levesque, H. J. (2000).
Congolog, a concurrent programming language based on the situation calculus.
Artif. Intell., 121(1-2):109–169.


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 IX


Hindriks, K. V. (2009).
Programming rational agents in GOAL.
In [Bordini et al., 2009], pages 119–157.


Hindriks, K. V., de Boer, F. S., van der Hoek, W., and Meyer, J.-J. C. (1997).
Formal semantics for an abstract agent programming language.
In Singh, M. P., Rao, A. S., and Wooldridge, M., editors, *ATAL*, volume 1365 of *Lecture Notes in Computer Science*, pages 215–229. Springer.


Hindriks, K. V., Pokahr, A., and Sardiña, S., editors (2009).
Programming Multi-Agent Systems, 6th International Workshop, ProMAS 2008, Estoril, Portugal, May 13, 2008. Revised Invited and Selected Papers, volume 5442 of *Lecture Notes in Computer Science*. 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.

Bibliography X


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.


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.


Leite, J. A., Omicini, A., Sterling, L., and Torroni, P., editors (2004).
Declarative Agent Languages and Technologies, First International Workshop, DALT 2003, Melbourne, Australia, July 15, 2003, Revised Selected and Invited Papers, volume 2990 of *Lecture Notes in Computer Science*. Springer.

Bibliography XI


Leite, J. A., Omicini, A., Torroni, P., and Yolum, P., editors (2005).
Declarative Agent Languages and Technologies II, Second International Workshop, DALT 2004, New York, NY, USA, July 19, 2004, Revised Selected Papers, volume 3476 of *Lecture Notes in Computer Science*. Springer.


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 et al., 2009], pages 29–46.


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 XII


Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., and Tummolini, L. (2004).
Coordination artifacts: Environment-based coordination for intelligent agents.
In *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'04)*, volume 1, pages 286–293, New York, USA. ACM.


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. (2009a).
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.


Piunti, M., Ricci, A., Boissier, O., and Hübner, J. F. (2009b).
Embodied organisations in mas environments.
In Braubach, L., van der Hoek, W., Petta, P., and Pokahr, A., editors, *Proceedings of 7th German conference on Multi-Agent System Technologies (MATES 09), Hamburg, Germany, September 9-11*, volume 5774 of *LNCS*, pages 115–127. Springer.

Bibliography XIII


Pokahr, A., Braubach, L., and Lamersdorf, W. (2005).
Jadex: A bdi reasoning engine.
In [Bordini et al., 2005a], pages 149–174.


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.


Rao, A. S. (1996).
Agentspeak(l): Bdi agents speak out in a logical computable language.
In de Velde, W. V. and Perram, J. W., editors, *MAAMAW*, volume 1038 of *Lecture Notes in Computer Science*, pages 42–55. Springer.


Ricci, A., Denti, E., and Piunti, M. (2010a).
A platform for developing SOA/WS applications as open and heterogeneous multi-agent systems.
Multiagent and Grid Systems International Journal (MAGS), Special Issue about "Agents, Web Services and Ontologies: Integrated Methodologies".
To Appear.

Bibliography XIV


Ricci, A., Piunti, M., and Viroli, M. (2010b).
Environment programming in multi-agent systems – an artifact-based perspective.
Autonomous Agents and Multi-Agent Systems.
Published Online with ISSN 1573-7454 (will appear with ISSN 1387-2532).


Ricci, A., Piunti, M., Viroli, M., and Omicini, A. (2009a).
Environment programming in CArtAgO.
In Bordini, R. H., Dastani, M., Dix, J., and El Fallah-Seghrouchni, A., editors, *Multi-Agent Programming: Languages, Platforms and Applications, Vol. 2*, pages 259–288. Springer Berlin / Heidelberg.


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


Ricci, A., Santi, A., and Piunti, M. (2010c).
Action and perception in multi-agent programming languages: From exogenous to endogenous environments.
In *In Proceedings of International Workshop on Programming Multi-Agent Systems (ProMAS-8)*.

Bibliography XV


Ricci, A., Viroli, M., and Omicini, A. (2007).
CArtAgO: A framework for prototyping artifact-based environments in MAS.
In Weyns, D., Parunak, H. V. D., and Michel, F., editors, *Environments for MultiAgent Systems III*, volume 4389 of *LNAI*, pages 67–86. Springer.
3rd International Workshop (E4MAS 2006), Hakodate, Japan, 8 May 2006. Selected Revised and Invited Papers.


Rocha Costa, A. C. d. and Dimuro, G. (2009).
A minimal dynamical organization model.
In Dignum, V., editor, *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, chapter XVII, pages 419–445. IGI Global.


Russell, S. and Norvig, P. (2003).
Artificial Intelligence, A Modern Approach (2nd ed.).
Prentice Hall.


Santi, A., Guidi, M., and Ricci, A. (2011).
Jaca-android: An agent-based platform for building smart mobile applications.
In *Languages, Methodologies and Development Tools for Multi-agent systems*, volume 6822 of *LNAI*. Springer Verlag.

Bibliography XVI


Shoham, Y. (1993).
Agent-oriented programming.
Artif. Intell., 60(1):51–92.


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


Winikoff, M. (2005).
Jack intelligent agents: An industrial strength platform.
In [Bordini et al., 2005a], pages 175–193.


Wooldridge, M. (2002).
An Introduction to Multi-Agent Systems.
John Wiley & Sons, Ltd.