Multi-**Agent** Oriented Programming using JaCaMo

Jomi F. Hübner

Federal University of Santa Catarina, Brazil

Summer School on AI for Industry 4.0
Saint-Etienne 2020

Context

From knowledge to **action**

From theoretical to **practical** reasoning

From mind to body & environment & others (inter**action**)

From individuals to **societies**

cc BY

Context

An MAS is a loosely coupled network of problem solvers that interact to solve problems that are beyond the individual capabilities or knowledge of each problem solver
– Durfee and Lesser 1989

Outline

- Agents
  - Practical reasoning
  - *Jason*
- Environment
- Organisation
- MAOP

(slides written together with R. Bordini, O. Boissier, and A. Ricci)

Agent Oriented Programming
— **AOP** —


?
environment

Literature

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

Proceedings: ProMAS, DALT, LADS, EMAS, AGERE, ...

Surveys: [Bordini et al., 2006], [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., 2007],
Jadex [Pokahr et al., 2005], 2APL [Dastani, 2008],
GOAL [Hindriks, 2009], JACK [Winikoff, 2005],
JIAC, ASTRA

But many others languages and platforms...

CC BY

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); ASTRA (Rem Collier); SARL (Stephane Galland); *simpAL*, *ALOO* (Ricci, ...);

•••

CC BY

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

CC BY

- Use of **mentalistic** notions and a **societal** view of computation [Shoham, 1993]

- Heavily influenced by the **BDI** architecture and reactive planning systems [Bratman et al., 1988]

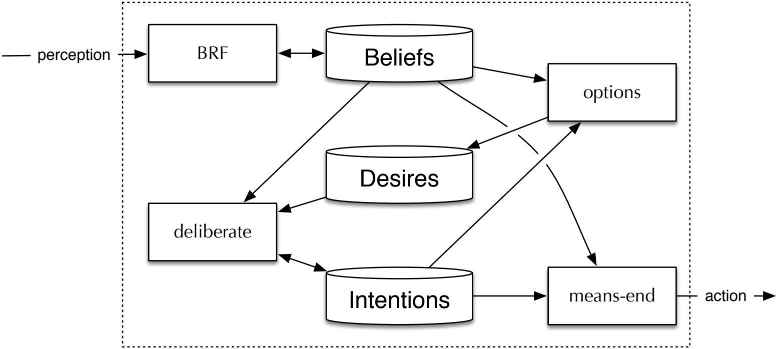
Motivation for BDI — **autonomous** robot

[Cohen and Levesque, 1990]

household robot.[1] You say "Willie, bring me a beer." The robot replies "OK, boss." Twenty minutes later, you screech "Willie, why didn't you bring that beer?" It answers "Well, I intended to get you the beer, but I decided to do something else." Miffed, you send the wise guy back to the manufacturer, complaining about a lack of commitment. After retrofitting, Willie is returned, marked "Model C: The Committed Assistant." Again, you ask Willie to bring a beer. Again, it accedes, replying "Sure thing." Then you ask: "What kind did you buy?" It answers: "Genessee." You say "Never mind." One minute later, Willie trundles over with a Genessee in its gripper. This time, you angrily return Willie for overcommitment. After still more tinkering, the manufacturer sends Willie back, promising no more problems with its commitments. So, being a somewhat trusting consumer, you accept the rascal back into your household, but as a test, you ask it to bring you your last beer. Willie again accedes, saying "Yes, Sir." (Its attitude problem seems to have been fixed.) The robot gets the beer and starts towards you. As it approaches, it lifts its arm, wheels around, deliberately smashes the bottle, and trundles off. Back at the plant, when interrogated by customer service as to why it had abandoned its commitments, the robot replies that according to its specifications, it kept its commitments as long as required—commitments must be dropped when fulfilled or impossible to achieve. By smashing the last bottle, the commitment

BDI architecture

(the mentalistic view)



CC BY

BDI architecture [Wooldridge, 2009]

```
while true do
    B ← brf(B, perception())              // belief revision
    D ← options(B, I)                     // desire revision
    I ← deliberate(B, D, I)               // get intentions
    π ← meansend(B, I, A)                 // gets a plan
    while π ≠ ∅ do
        execute( head(π) )
        π ← tail(π)
```

BDI architecture [Wooldridge, 2009]

```
while true do
    B ← brf(B, perception())                    // belief revision
    D ← options(B, I)                           // desire revision
    I ← deliberate(B, D, I)                     // get intentions
    π ← meansend(B, I, A)                       // gets a plan
    while π ≠ ∅ do
        execute( head(π) )
        π ← tail(π)
```

fine for pro-activity, but not for reactivity (over **commitment**)

BDI architecture [Wooldridge, 2009]

```
while true do
    B ← brf(B, perception())                  // belief revision
    D ← options(B, I)                         // desire revision
    I ← deliberate(B, D, I)                   // get intentions
    π ← meansend(B, I, A)                     // gets a plan
    while π ≠ ∅ do
        execute( head(π) )
        π ← tail(π)
        B ← brf(B, perception())
        if ¬sound(π, I, B) then
            π ← meansend(B, I, A)
```

revise commitment to plan – re-planning for context adaptation

BDI architecture [Wooldridge, 2009]

```
while true do
    B ← brf(B, perception())                    // belief revision
    D ← options(B, I)                            // desire revision
    I ← deliberate(B, D, I)                      // get intentions
    π ← meansend(B, I, A)                        // gets a plan
    while π ≠ ∅ and ¬succeeded(I, B) and ¬impossible(I, B) do
        execute( head(π) )
        π ← tail(π)
        B ← brf(B, perception())
        if ¬sound(π, I, B) then
            π ← meansend(B, I, A)
```

revise commitment to intentions – Single-Minded Commitment

```
while true do
    B ← brf(B, perception())                          // belief revision
    D ← options(B, I)                                 // desire revision
    I ← deliberate(B, D, I)                           // get intentions
    π ← meansend(B, I, A)                             // gets a plan
    while π ≠ ∅ and ¬succeeded(I, B) and ¬impossible(I, B) do
        execute( head(π) )
        π ← tail(π)
        B ← brf(B, perception())
        if reconsider(I, B) then
            D ← options(B, I)
            I ← deliberation(B, D, I)
        if ¬sound(π, I, B) then
            π ← meansend(B, I, A)
```

reconsider the intentions (not always!)

Jason

(let's go **programming** those nice concepts)

(BDI & **Jason**) Hello World – agent bob

```
happy(bob).                                  // B
!say(hello).                                 // D

+!say(X) : happy(bob)                        // I
   <- .print(X).
```

CC BY

(BDI & **Jason**) Hello World – agent bob

beliefs

- prolog like (FOL)

```
happy(bob).                                  // B
!say(hello).                                 // D

+!say(X) : happy(bob)                        // I
   <- .print(X).
```

desires

- prolog like
- with **!** prefix

```
happy(bob).
!say(hello).                                      // D

+!say(X) : happy(bob)                             // I
    <- .print(X).
```

CC BY

(BDI & **Jason**) Hello World – agent bob

```
happy(bob).
!say(hello).

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

plans

- define when a desire becomes an intention
  ⇝ **deliberate**
- how it is satisfied
- are used for practical reasoning
  ⇝ **means-end**

CC BY

Hello World

desires from perception — **options**

```
+happy(bob) <- !say(hello).

+!say(X) : not today(monday)
    <- .print(X).
```

CC BY

Hello World

source of beliefs

```
+happy(bob)[source(A)]
    :  someone_who_knows_me_very_well(A)
    <- !say(hello).

+!say(X) : not today(monday) <- .print(X).
```

Hello World

plan selection

```
+happy(H)[source(A)]
    :  sincere(A) & .my_name(H)
    <- !say(hello).

+happy(H)
    :  not .my_name(H)
    <- !say(i_envy(H)).

+!say(X) : not today(monday) <- .print(X).
```

Hello World

intention revision

```
+happy(H)[source(A)]
    :  sincere(A) & .my_name(H)
    <- !say(hello).

+happy(H)
    :  not .my_name(H)
    <- !say(i_envy(H)).

+!say(X) : not today(monday) <- .print(X); !say(X).
```

Hello World

intention revision

```
+happy(H)[source(A)]
    :  sincere(A) & .my_name(H)
    <- !say(hello).

+happy(H)
    :  not .my_name(H)
    <- !say(i_envy(H)).

+!say(X) : not today(monday) <- .print(X); !say(X).

-happy(H)
    :  .my_name(H)
    <- .drop_intention(say(hello)).
```

CC BY

Hello World

intention revision

```
+happy(H)[source(A)]
   :  sincere(A) & .my_na
   <- !say(hello).

+happy(H)
   :  not .my_name(H)
   <- !say(i_envy(H)).

+!say(X) : not today(mond

-happy(H)
   :  .my_name(H)
   <- .drop_intention(say(hello)).
```

features

- we can have several intentions based on the same plans
  ⇝ running concurrently
- long term goals running
  ⇝ reaction meanwhile
  ⇝ not overcommitted
- plan selection based on circumstance
- actions (partially) computed by the interpreter
  ⇝ programmer **declares** plans

CC BY

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

cc BY

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, Rafael H. Bordini, and others

Main Language Constructs

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

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)].
```

CC BY

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)]
```

CC BY

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
```

CC BY

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).
```

CC BY

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);
...
```

CC BY

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
```

CC BY

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 unifies with Answer
```

CC BY

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

CC BY

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)

CC BY

Plans — Operators for 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
      !g3[hard_deadline(3000)].  // goal with deadline
```

CC BY

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**

cc BY

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

CC BY

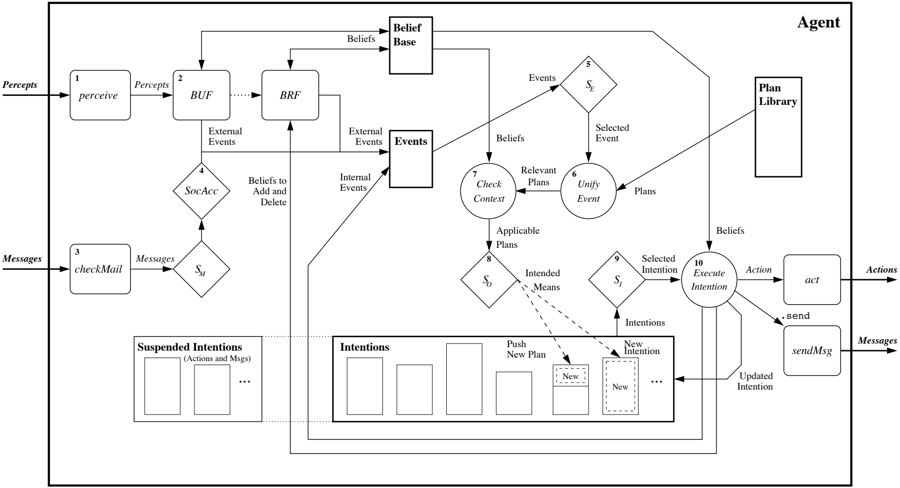
Basic Reasoning cycle

runtime interpreter

- 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
- execute one step of the selected intention

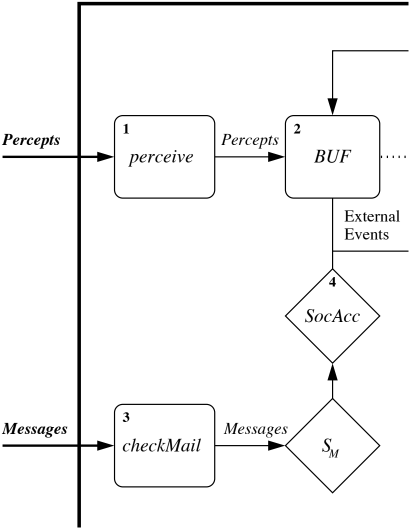
CC BY

Jason Reasoning Cycle



CC BY

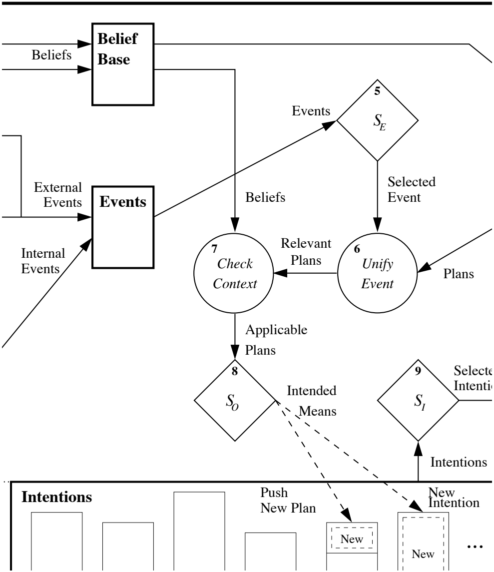
Jason Reasoning Cycle



- machine perception
- belief revison
- knowledge representation
- communication, argumentation
- trust
- social power

cc BY

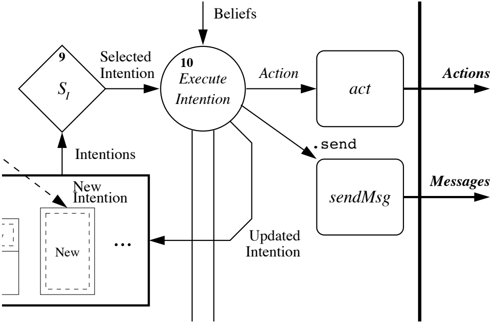
Jason Reasoning Cycle



- planning
- reasoning
- decision theoretic techniques
- learning (reinforcement)

CC BY

Jason Reasoning Cycle



- intention reconsideration
- scheduling
- action theories

CC BY

A note about "Control"

Agents can control (manipulate) their own (and influence the others)

- beliefs
- goals
- plan

By doing so they control their behaviour

The developer provides initial values of these elements and thus also influence the behaviour of the agent

Failure Handling: Contingency Plans

Example (an agent blindly committed to g)

```
+!g : g.     // g is a declarative goal

+!g : ... <- a1; ?g.
+!g : ... <- a2; ?g.
+!g : ... <- a3; ?g.

+!g <- !g. // keep trying
-!g <- !g. // in case of some failure

+g <-.succeed_goal(g).
```

CC BY

Failure Handling: Contingency Plans

Example (single minded commitment)

```
+!g : g.     // g is a declarative goal

+!g : ... <- a1; ?g.
+!g : ... <- a2; ?g.
+!g : ... <- a3; ?g.

+!g <- !g. // keep trying
-!g <- !g. // in case of some failure

+g <-.succeed_goal(g).
+f : .super_goal(g,SG) <-.fail_goal(SG).
```

f is the drop condition for goal *g*

CC BY

Compiler pre-processing – directives

Example (single minded commitment)

```
{ begin smc(g,f) }
     +!g : ... <- a1.
     +!g : ... <- a2.
     +!g : ... <- a3.
{ end }
```

CC BY

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

Other Language Features

Strong Negation

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

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

CC BY

Prolog-like Rules in the Belief Base

```
tall(X) :- woman(X) & height(X, H) & H > 1.70.
tall(X) :- man(X) & height(X, H) & H > 1.80.
```

CC BY

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

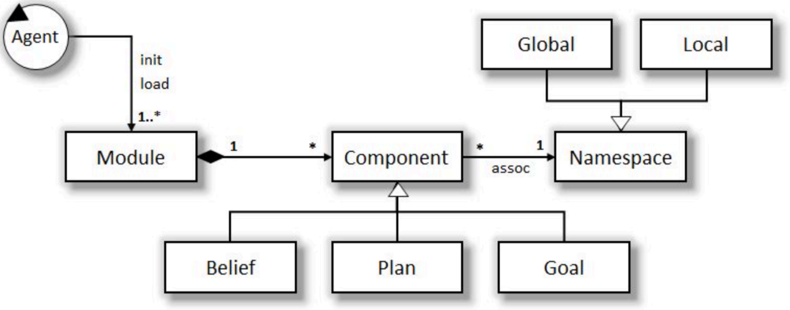
```
libname.action_name(...)
```

Standard Internal Actions

- Standard (pre-defined) internal actions have an empty library name
  - `.print(`$term_1$, $term_2$, ...`)`
  - `.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.

CC BY

Namespaces & Modularity



CC BY

Namespaces & Modularity

```
{include("initiator.asl", pc)}
{include("initiator.asl", tv)}

!pc::startCNP(fix(pc)).
!tv::startCNP(fix(tv)).

+pc::winner(X)
   <- .print(X).
```

Inspection of agent **alice**

- Beliefs

tv::
introduction(participant)[source(compan
propose(*11.075337225252543*)[sourc
propose(*12.043311087442898*)[sourc
propose(*12.81277904935436*)[source
winner(company_A1)[source(self)].

#8priv::
state(finished)[source(self)].

pc::
introduction(participant)[source(compan
propose(*11.389500048463455*)[sourc
propose(*11.392553683771682*)[sourc
propose(*12.348901000262853*)[sourc
winner(company_A2)[source(self)].

Jason Customisations

- **Agent** class customisation:
  selectMessage, selectEvent, selectOption, selectIntention, 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, ...)

CC BY

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

CC BY

Java code – go to gold

```
public class Robot extends Thread {
   boolean seeGold, lowBattery;
   public void run() {
      while (true) {
          while (! seeGold) {
              a = randomDirection();
              doAction(go(a));

          }
          while (seeGold) {
              a = selectDirection();

              doAction(go(a));

} } }   }
```

Java code – charge battery

```
public class Robot extends Thread {
   boolean seeGold, lowBattery;
   public void run() {
      while (true) {
          while (! seeGold) {
              a = randomDirection();
              doAction(go(a));
              if (lowBattery) charge();
          }
          while (seeGold) {
              a = selectDirection();
              if (lowBattery) charge();
              doAction(go(a));
              if (lowBattery) charge();
}  }  }   }
```

CC BY

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

CC BY

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

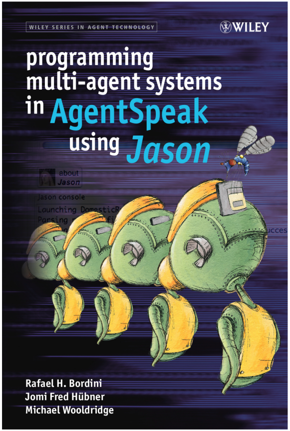
cc BY

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

CC BY

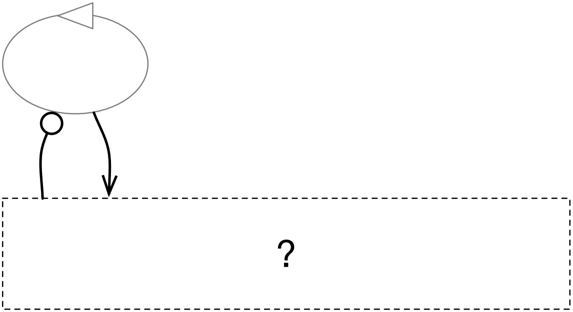
- `http://jason.sourceforge.net`
- R.H. Bordini, J.F. Hübner, and M. Wooldrige
  Programming Multi-Agent Systems in AgentSpeak using *Jason*
  John Wiley & Sons, 2007.



cc BY

Environment Oriented Programming

— **EOP** —

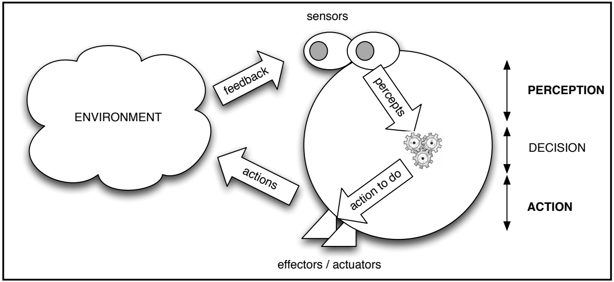


Back to the Notion of Environment in MAS

- 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

CC BY

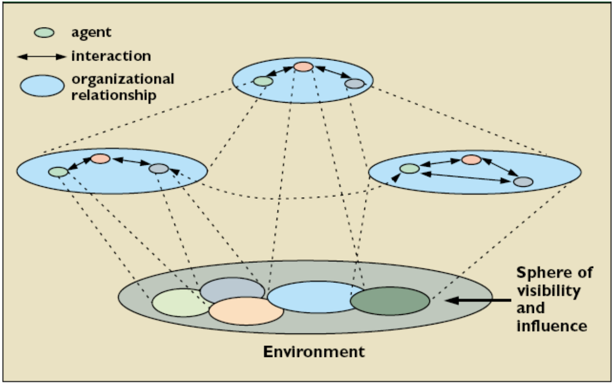
Single Agent Perspective



- **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

CC BY

Multi-Agent Perspective



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

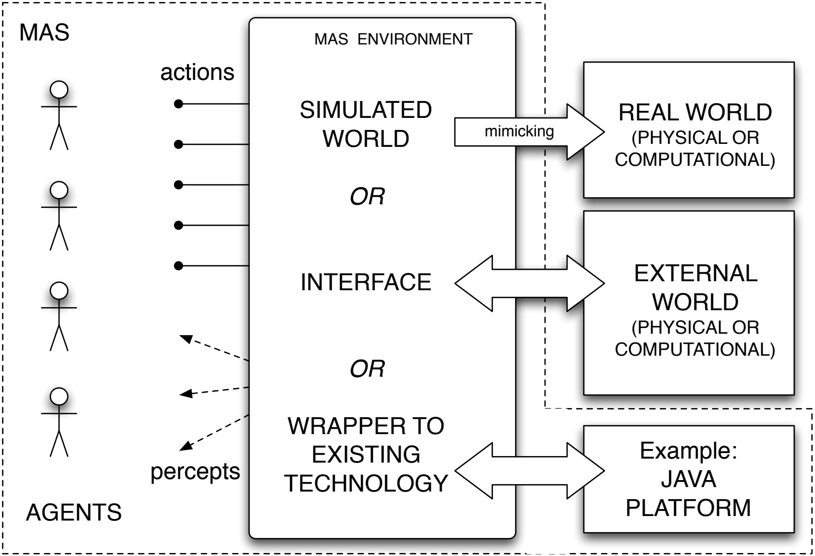
CC BY

Why Environment Programming

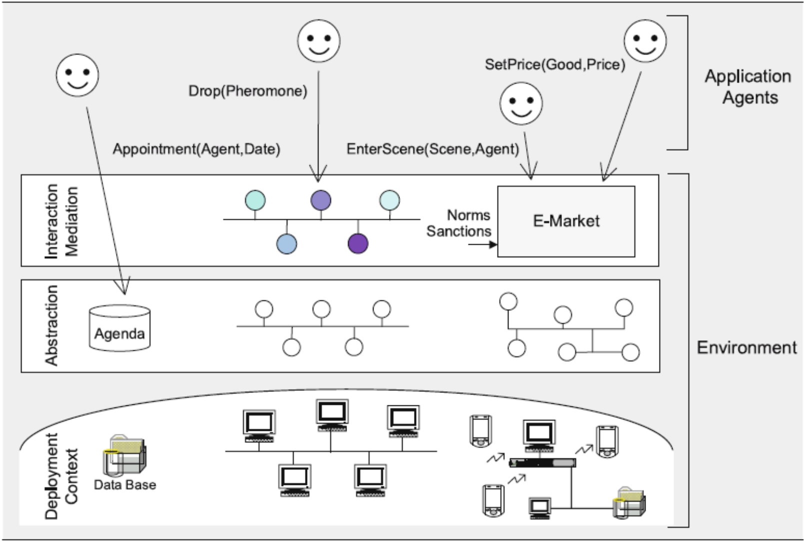
- Basic level
  - to create testbeds for real/external environments
  - to ease the interface/interaction with existing software environments
- Advanced level
  - to uniformly **encapsulate** and **modularise** functionalities of the MAS out of the agents
    - typically related to interaction, coordination, 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

CC BY

Basic Level Overview



Advanced Level Overview [Weyns et al., 2007]



CC BY

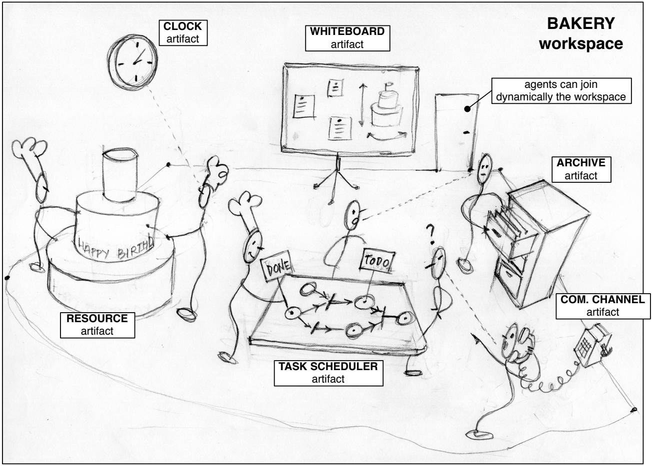
Existing Computational Frameworks

- AGRE / AGREEN / MASQ [Stratulat et al., 2009]
  - AGRE – integrating the AGR (Agent-Group-Role) organisation model with a notion of environment
    - Environment used to represent both the physical and social part of interaction
  - AGREEN / MASQ – extending AGRE towards a unified representation for physical, social and institutional environments
  - Based on MadKit platform [Gutknecht and Ferber, 2000]
- GOLEM [Bromuri and Stathis, 2008]
  - Logic-based framework to represent environments for situated cognitive agents
  - composite structure containing the interaction between cognitive agents and objects
- A&A and CArtAgO [Ricci et al., 2010a]
  - introducing a computational notion of artifact to design and implement agent environments

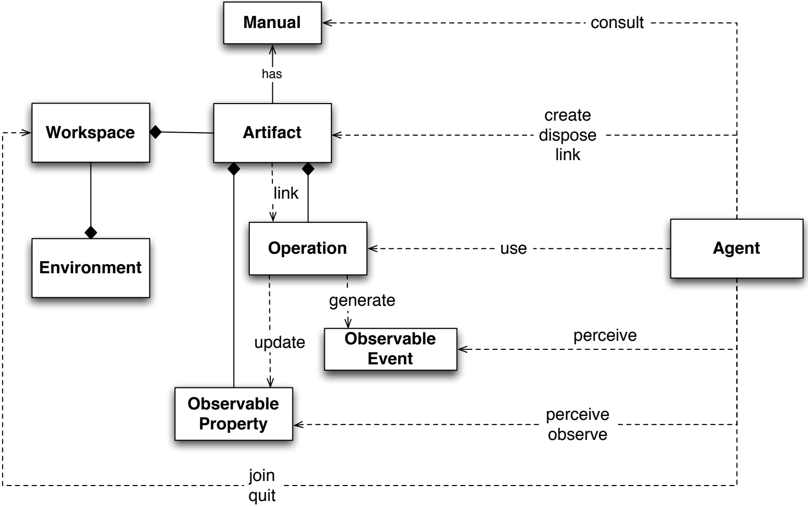
CC BY

A&A and CArtAgO

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

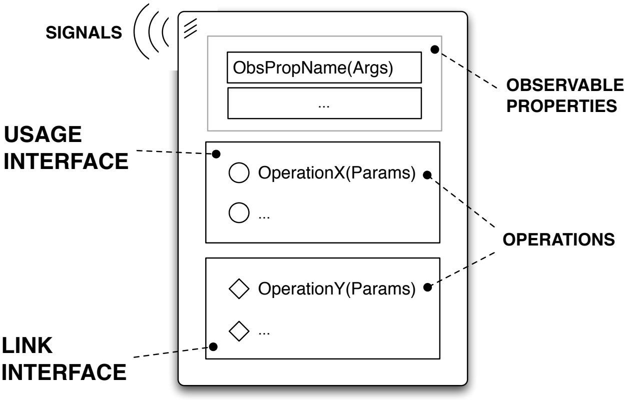


A&A Meta-Model in More Detail [Ricci et al., 2010a]

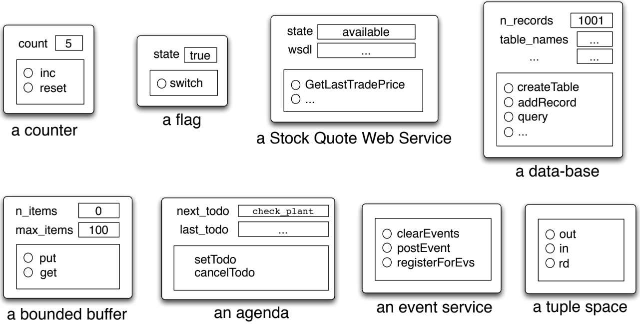


CC BY

Artifact Abstract Representation



A World of Artifacts



cc BY

Actions and Percepts in Artifact-Based Environments [Ricci et al., 2010b]

actions $\longleftrightarrow$ artifacts' operation

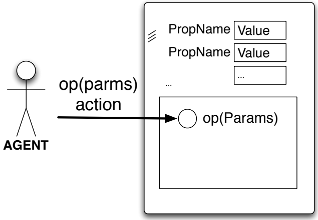
the action repertoire is given by the dynamic set of operations provided by the overall set of artifacts available in the workspace can be changed by creating/disposing artifacts

- 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

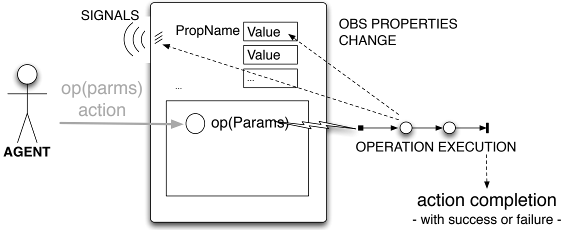
Interaction Model: Use



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

CC BY

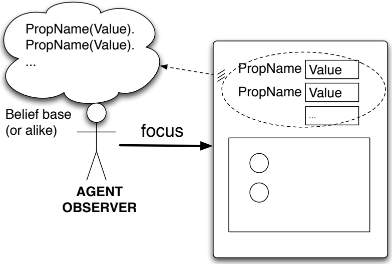
Interaction Model: Operation execution



- a process structured in one or multiple transactional steps
- asynchronous with respect to agent
  - **...which can proceed possibly reacting to percepts and executing actions of other plans/activities**
- operation completion causes action completion
  - action completion events with success or failure, possibly with action feedbacks

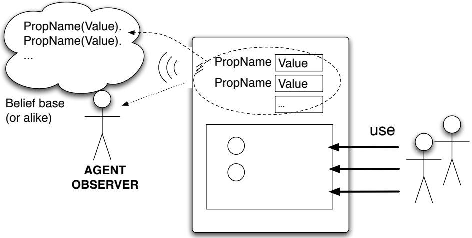
CC BY

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 `https://github.com/CArtAgO-lang/cartago`

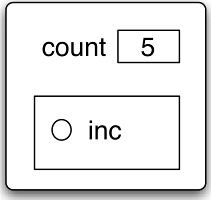
cc BY

Example 1: 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

CC BY

Example 1: 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

CC BY

Action Execution & Blocking Behaviour

- Given the action/operation map, by executing an action the intention/activity is suspended until the corresponding operation has completed or failed
  - action completion events generated by the environment and automatically processed by the agent/environment platform bridge
  - no need of explicit observation and reasoning by agents to know if an action succeeded
- However **the agent execution cycle is not blocked**!
  - the agent can continue to process percepts and possibly execute actions of other intentions

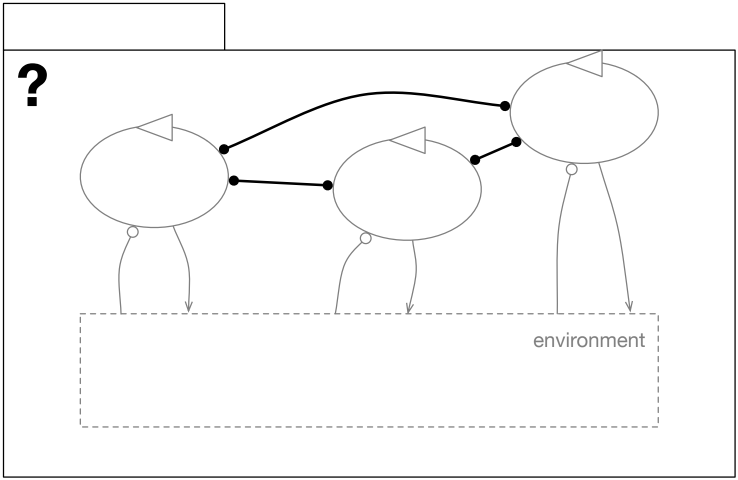
CC BY

Wrap-up

- 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

Organisation Oriented Programming

— **OOP** —


?
environment

Introduction: Some definitions

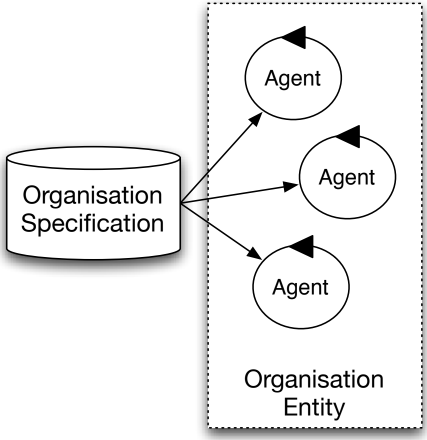
- Organisations are structured, patterned systems of activity, knowledge, culture, memory, history, and capabilities that are distinct from any single agent [Gasser, 2001]
  ⇝ organisations are **supra-individual** phenomena
- A decision and communication schema which is applied to a set of actors that together fulfill a set of tasks in order to satisfy goals while guarantying a global coherent state [Malone, 1999]
  ⇝ 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]
  ⇝ **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]
  ⇝ **pattern of emergent cooperation**

cc BY

Organisation in MAS – a definition

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

Organisation Oriented Programming (OOP)

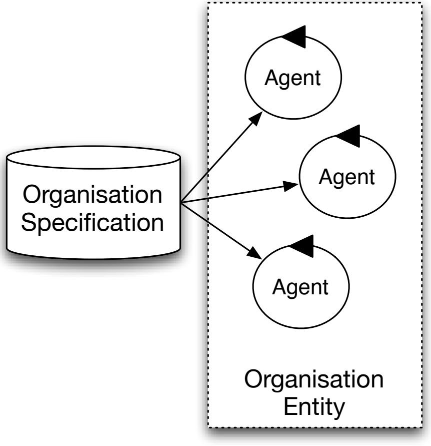


- Programming outside the agents
- Using organisational concepts
- To define a cooperative pattern

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

CC BY

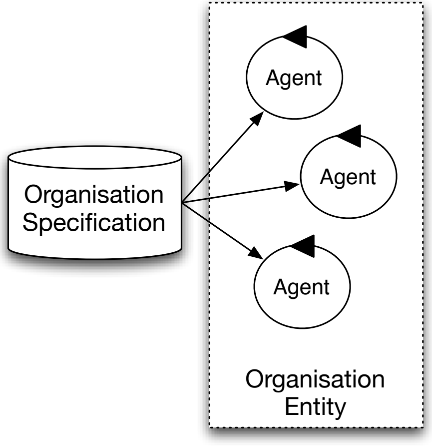
Organisation Oriented Programming (OOP)



First approach

- Agents read the program and follow it

CC BY

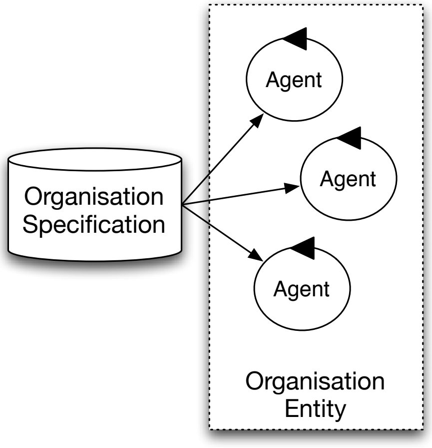


Second approach

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

cc BY

Organisation Oriented Programming (OOP)

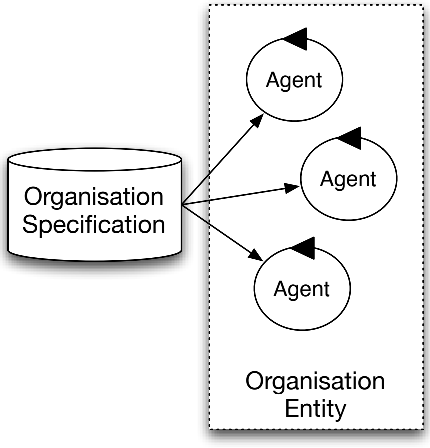


Second approach

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

CC BY

Organisation Oriented Programming (OOP)



Components

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

cc BY

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: **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

CC BY

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]
- THOMAS [Criado et al., 2011],
- ...

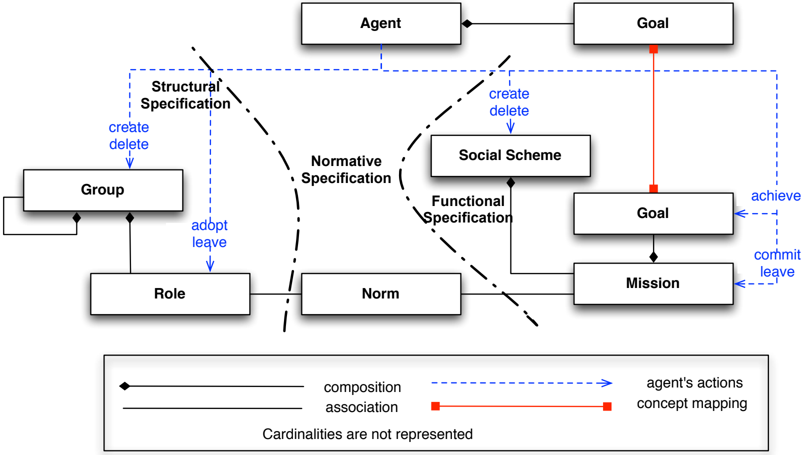
CC BY

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

CC BY

Moise OML meta-model (partial view)



CC BY

$\mathcal{M}$oise OML

- OML for defining organisation specification **and** organisation entity
- Three independent dimensions [Hübner et al., 2007] (⇝ 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
    - ⇝ $\mathcal{J}$-$\mathcal{M}$oise$^+$ [Hübner et al., 2007]
  - the Organisation Management Infrastructure
    - ⇝ ORA4MAS [Hübner et al., 2009]

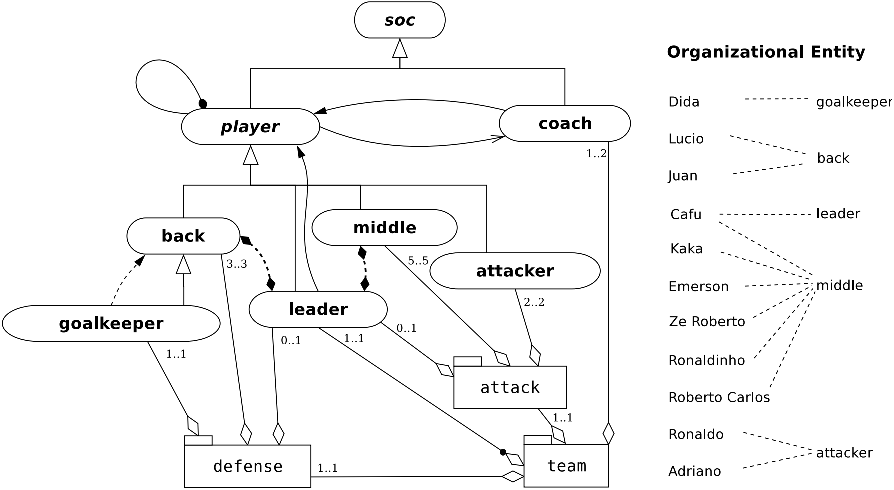
CC BY

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

CC BY

Structural Specification Example



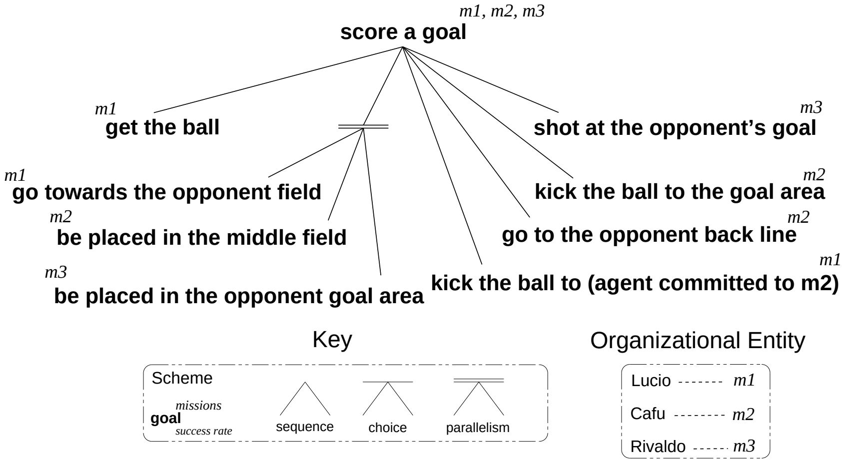
Graphical representation of structural specification of 3-5-2 Joj Team

CC BY

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

CC BY



Graphical representation of social scheme "side_attack" for joj team

CC BY

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

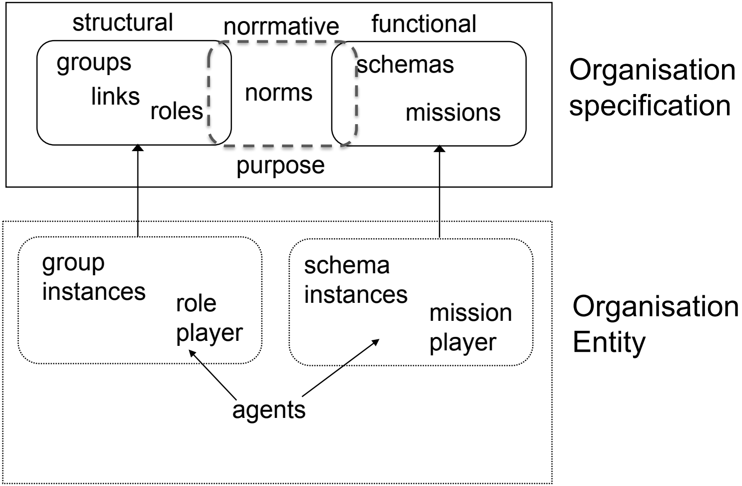
CC BY

Norm Specification – example

role	**deontic**	**mission**		TTF
back	*obliged*	$m1$	get the ball, go ...	1 minute
left	*obliged*	$m2$	be placed at ..., kick ...	3 minute
right	*obliged*	$m2$		1 day
attacker	*obliged*	$m3$	kick to the goal, ...	30 seconds

CC BY

Organisational Entity



CC BY

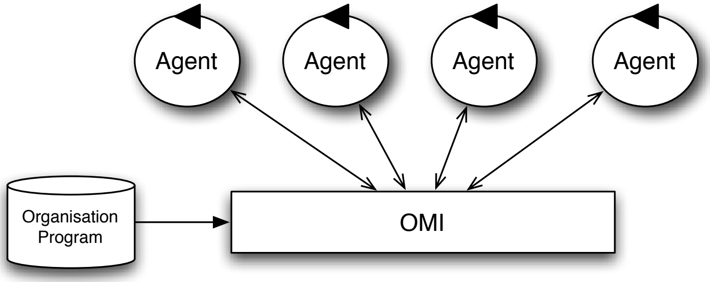
Organisation Entity Dynamics

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

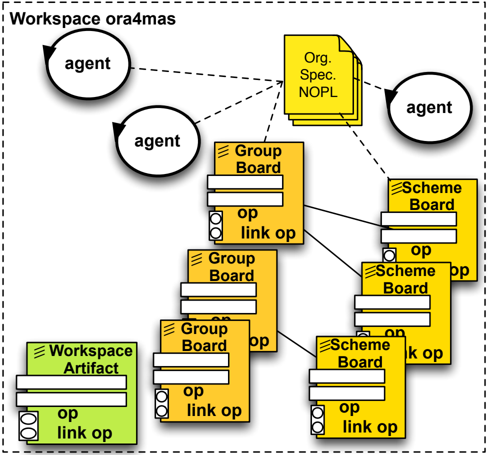
- Managing – coordination, regulation – the agents' execution within organisation defined in an organisational specification



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

cc BY

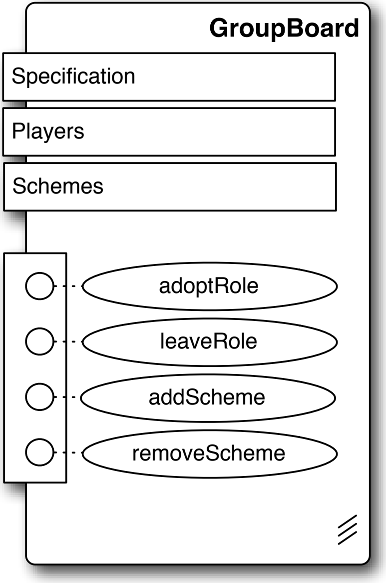
Organisational artifacts in ORA4MAS



- based on A&A and $\mathcal{M}$OISE
- 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

CC BY

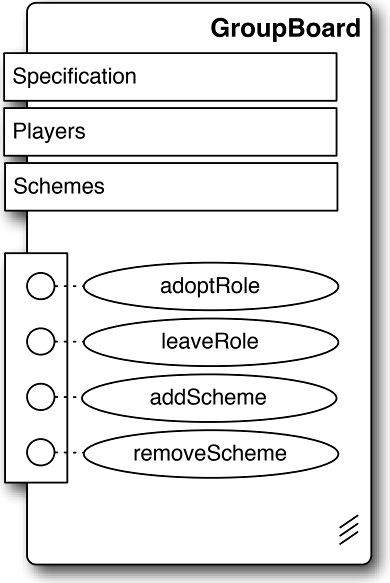
ORA4MAS – GroupBoard artifact



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)

CC BY

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

CC BY

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

CC BY

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**

CC BY

Example (GroupBoard)

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

CC BY

Organisational **actions** in *Jason* II

Example (SchemeBoard)

```
...
makeArtifact(
    "sch1",
    "ora4mas.nopl.SchemeBoard",
    ["auction-os.xml", doAuction],
    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(cob

CC BY

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 and norm violation)

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

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

CC BY

Typical plans for obligations

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

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

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

CC BY

- 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

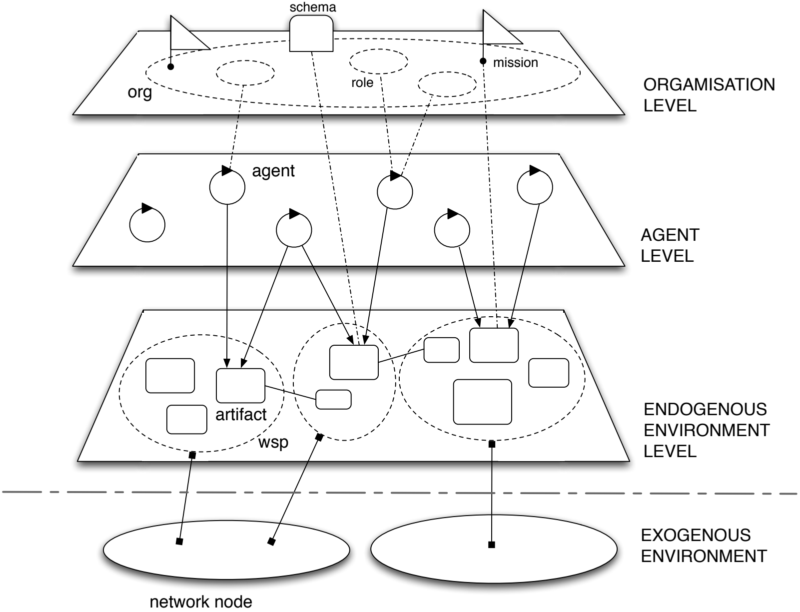
`http://moise.souceforge.net`

CC BY

Conclusions

- MAS is an **organisation** of autonomous **agents** interacting together to achieve their goals within a shared **environment**
- MAOP is a conceptual and practical tool to design and implement distributed, complex, huge, open, ... systems

CC BY


schema
mission
role
org
ORGAMISATION LEVEL
agent
AGENT LEVEL
artifact
wsp
ENDOGENOUS ENVIRONMENT LEVEL
EXOGENOUS ENVIRONMENT
network node

Agents

Programming **actions** with

- high level abstraction
  (beliefs, plans, goals, ...)
- concurrent, distributed, decoupled, open, …

Environment

Programming **tools** for the agents

- high level abstraction
  (workspaces, artifacts, perception, action, ...)
- concurrent, distributed, decoupled, open, …

Organisation

Helping the agents to live **together**

- high level abstraction
  (group, roles, schemes, norms, ...)
- concurrent, distributed, decoupled, open, …

What we have learnt in this project?

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

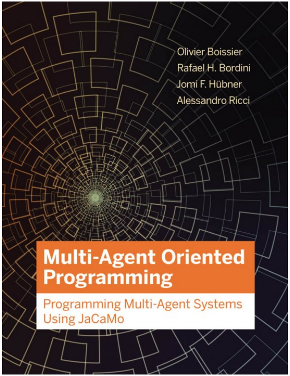
⇝ **separation of concerns**

⇝ the right tool for each problem

CC BY

Further Resources

- `http://jacamo.sourceforge.net`
- Olivier Boissier, Rafael H. Bordini, Jomi Hübner and Alessandro Ricci
  Multi-Agent Oriented Programming: Programming Multi-Agent Systems Using JaCaMo
  MIT Press, 2020.



CC BY

Acknowledgements

- Various colleagues and students
- JaCaMo users for helpful feedback
- CNPq, CAPES, ANP for supporting some of our current researh

Bibliography I

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. (2006).
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 (2005).
Multi-Agent Programming: Languages, Platforms and Applications, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*.
Springer.

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

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

Bratman, M. E., Israel, D. J., and Pollack, M. E. (1988).
Plans and resource-bounded practical reasoning.
Computational Intelligence, 4:349–355.

Bromuri, S. and Stathis, K. (2008).
Situating Cognitive Agents in GOLEM.
In Weyns, D., Brueckner, S., and Demazeau, Y., editors, *Engineering Environment-Mediated Multi-Agent Systems*, volume 5049 of *LNCS*, pages 115–134. Springer Berlin / Heidelberg.

Cohen, P. R. and Levesque, H. J. (1990).
Intention is choice with commitment.
Artificial Intelligence, 42:213–261.

Criado, N., Argente, E., and Botti, V. (2011).
THOMAS: An agent platform for supporting normative multi-agent systems.
Journal of Logic and Computation, 23(2):309–333.

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

Bibliography III

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., 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.

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 *PROMAS*, 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.

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.

CC BY

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.

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.

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 VI

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.

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

Bibliography VII

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

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.

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

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., Piunti, M., and Viroli, M. (2010a).
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. (2010b).
  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)*.
- 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.

Bibliography IX

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.

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

Stratulat, T., Ferber, J., and Tranier, J. (2009).
MASQ: towards an integral approach to interaction.
In *AAMAS (2)*, pages 813–820.

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

Weyns, D., Omicini, A., and Odell, J. J. (2007).
Environment as a first-class abstraction in multi-agent systems.
Autonomous Agents and Multi-Agent Systems, 14(1):5–30.

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

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

Wooldridge, M. (2009).
An Introduction to MultiAgent Systems.
John Wiley and Sons, 2nd edition.

TOC I

TOC II

CC BY